APPLICATION OF .NET TECHNOLOGY

E-BOOK

रा.इ.सू.प्रौ.सं
NIELIT

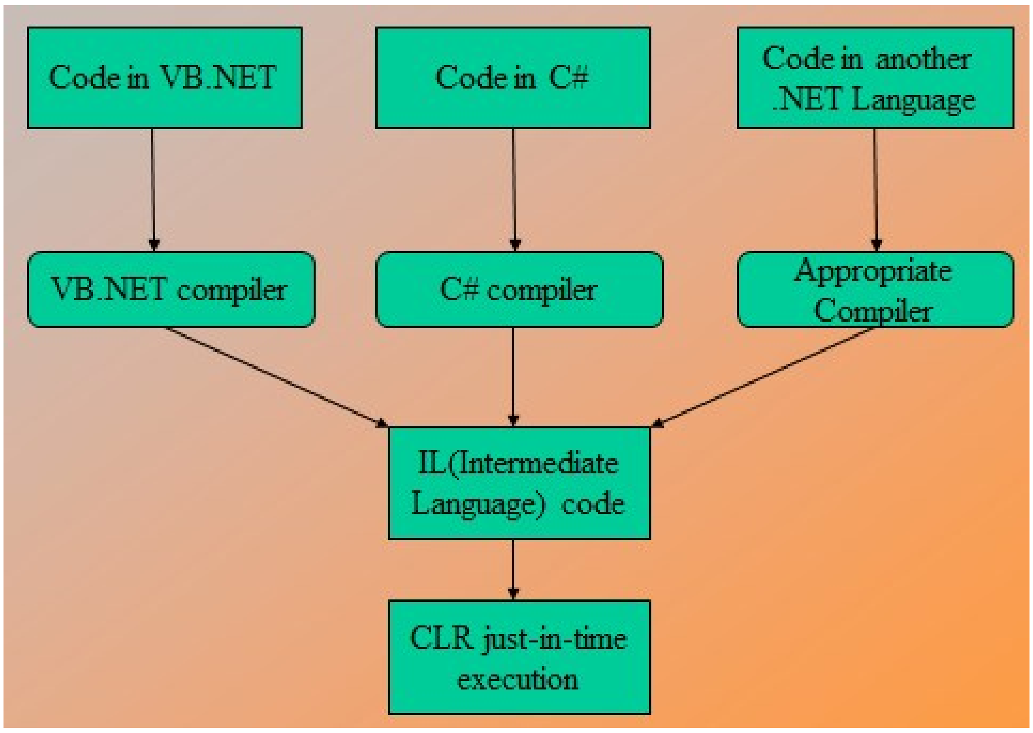
UNIT - 1
The .NET framework

Introduction

- .NET is a wonderful platform for developing modern applications.
- It provides a rich set of functionality out of the box.
- This means that as a developer you need not go into low level details of many operations such as file IO, network communication and so on. ASP.NET provides an event driven programming model with complex user interface.
- .NET provides a fully object oriented environment.
- .NET provides multi language programming for applications.
- .NET provides promising platform for programming such devices.
- Memory leaks were major reason in failure of applications.
- .NET takes this worry away from developer by handling memory on its own.
- .NET is the only platform that has built with XML right into the core framework.
- .NET does not require any registration as such; much of the deployment is simplified.
- .NET platform safe and secure for enterprise applications.

Common language runtime

- CLR works like a virtual machine in executing all languages.
- All .NET languages must obey the rules and standards imposed by CLR. Examples:
  - Object declaration, creation and use,
  - Data types, language libraries,
  - Error and exception handling,
  - Interactive Development Environment (IDE).
- Common language runtime is considered as the heart of the .NET framework.
- .NET applications are compiled to a common language known as Microsoft Intermediate Language or “IL”.
- The CLR, then, handles the compiling the IL to machine language, at which point the program is executed.
- It works like a virtual machine for executing and to execute the code written in different programming languages must obey the rules imposed by CLR as shown.


Code in VB.NET
Code in C#
Code in another .NET Language
VB.NET compiler
C# compiler
Appropriate Compiler
IL(Intermediate Language) code
CLR just-in-time execution

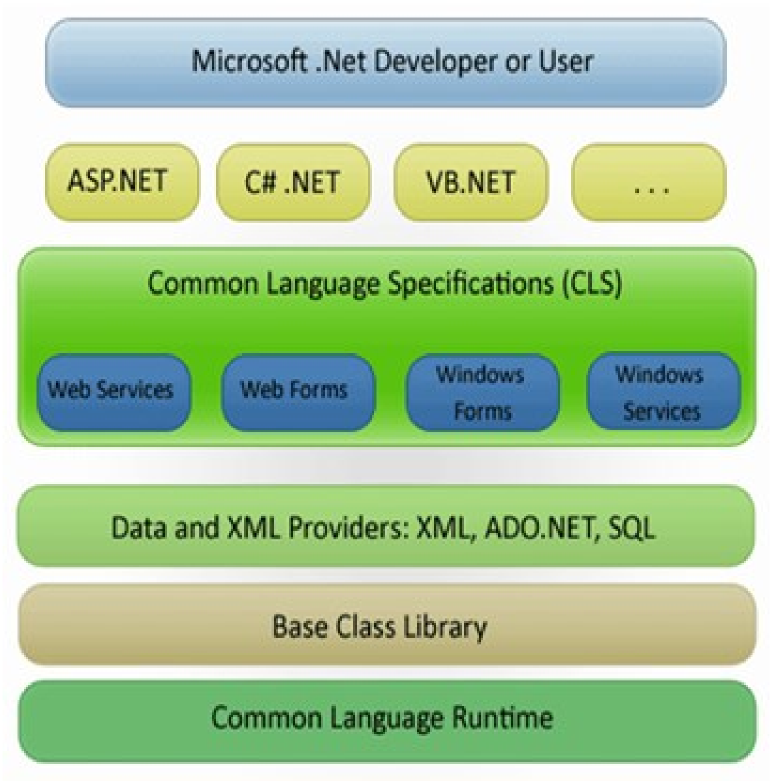
- Common language runtime is a mixture of different programming languages.
- With the help of common language specification and common type system a standard class framework is also developed for automatic memory management.
- Common language runtime handles the errors and execute the code in a safer way.
- It allows for deployment in multi-platform by removal of registration dependencies it increases the safety by versioning the application.

Common Type System

- This binary compatibility between language types is called the Common Type System (CTS).
    - To support multiple programming languages.
    - Ability to reuse the FCL (Framework Class Library).
    - Each programming language must be compatible.
    - The built-in types are represented as types in the FCL.
- It also have the built in types for different data types like a C# “int” data type is the same as a VB.NET “Integer” type and their .NET type is System.
- “Int32”, which is a 32-bit integer named Int32 in the System namespace of the FCL (framework class libraries).

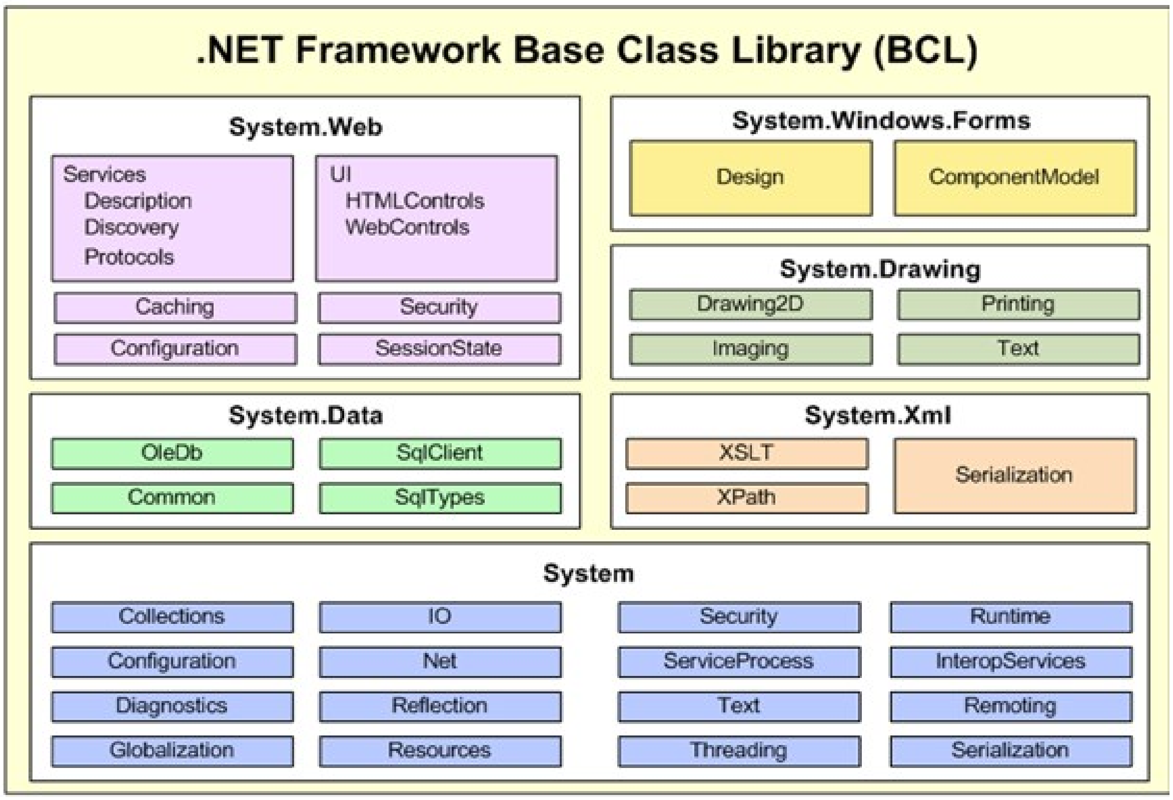
Common Language Specifications

- CLS is a set of rules that specifies features that all languages should support.
  - Goal: have the .NET framework support multiple languages.
  - CLS is an agreement among language designers and class library designers about the features and usage conventions that can be relied upon Ada, BASIC, Fortran, SQL, Pascal.
  - Example: public names should not rely on case for uniqueness since some languages are not case sensitive.
- However, one of the benefits of having a CLR with CTS that understands Intermediate Language, and an Framework Class Library that supports all languages, is the ability to write code in one language that is consumable by other languages.



The Base Class Library

- In .NET framework provides a set of base class libraries which provide functions and features which can be used with any programming language which implements .NET, such as Visual Basic, C#, Visual C++, etc.
  - Collections,
  - XML,
  - Data Type definitions.
- It contains standard programming features such as Collections, XML, Data Type definitions, etc.



- In .NET framework base class libraries contains IO (for reading and writing to files), Reflection and Globalization to name a few.
- All of which are contained in the System namespace.
- As well, it contains some non-standard features such as LINQ, ADO.NET (for database interactions), drawing capabilities, forms and web support.

The Base Class Library Namespace

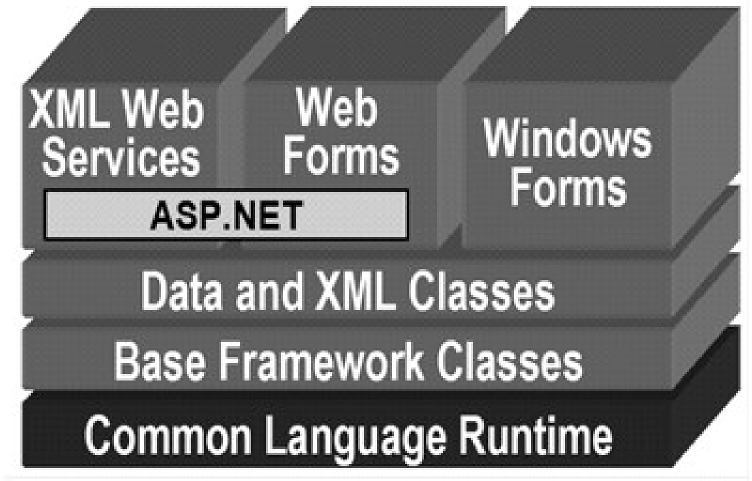
- Here we are seeing few base class libraries like system is a class library contains the fundamentals for programming such as the data types, console, match and arrays, etc.
- In another class library System.CodeDom Supports the creation of code at runtime and the ability to run it and also class library like system.Collections contains lists, stacks, hash tables and dictionaries.

Few Base Class Library Namespace

- System.
- System.CodeDom.
- System.Collections.
- System.Configuration.

.Net Class Library

- .Net class libraries are a standard library for .NET Framework implementation of the Standard Libraries as defined in the Common Language Infrastructure.

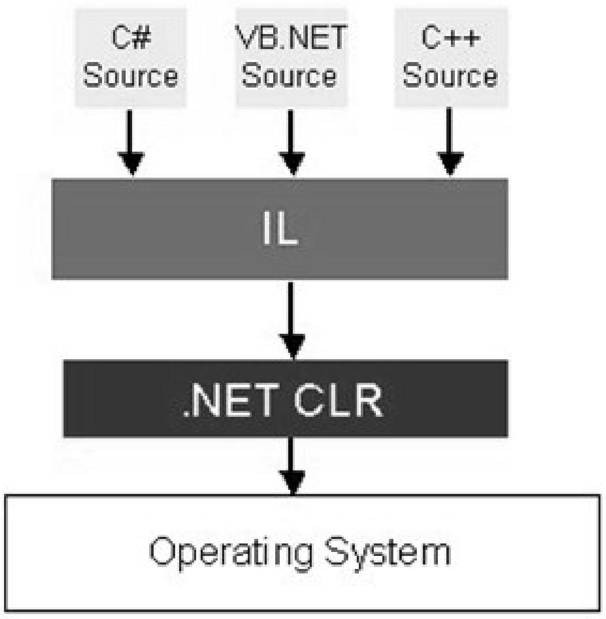


- The FCL is a collection of reusable classes, interfaces and value types.
  - Input/output,
  - String manipulation,
  - Security management,
  - Network communications,
  - Thread management and other functions.
- The BCL is the core of the FCL and provides the most fundamental functionality, which includes classes for
  - Input/output,
  - String manipulation,
  - Security management,
  - Network communications,
  - Thread management and other functions.
- In the class library framework above the base classes data and xml classes are hosted.
- Data classes support persistent data management and include SQL classes for manipulating persistent data stores through a standard SQL interface.
- XML classes enable XML data manipulation and XML searching and translations.
- .Net Class Libraries have another layer of Web Forms include classes that enable you to rapidly develop web GUI applications.

- Windows Forms support a set of classes that allow you to :
  - Develop Windows GUI applications,
  - Facilitating drag-and-drop GUI development and providing a common,
  - Consistent development interface across all languages supported by the .NET Framework.

Intermediate Language

- Intermediate language (IL) is an object-oriented programming language designed to be used by compilers for the .NET Framework before static or dynamic compilation to machine code.



- The IL is used by the .NET Framework to generate machine-independent code as the output of compilation of the source code written in any .NET programming language.
- IL is a stack-based assembly language that gets converted to bytecode during execution of a virtual machine.
- It is defined by the common language infrastructure (CLI) specification.
- As IL is used for automatic generation of compiled code, there is no need to learn its syntax.
- This term is also known as Microsoft intermediate language (MSIL) or common intermediate language (CIL).
- Intermediate Language can be,
  - Executed on any computer architecture,
  - Allowing compiled source code,
  - Supporting the CLI specification.
- Intermediate language provides
  - Better performance,
  - Preserves memory,
  - Saves time during application initialization.
- The platform and CPU independence feature.

- Verification of code safety, for IL code, provides better security and reliability than natively-compiled executable files.
- The metadata, describing the MSIL code in the portable executable, eliminates the need for type libraries and intermediate definition language files as was used in the Component Object Model (COM) technology.
- Combined with metadata and a common type system, IL forms the means to integrate modules written in different languages into one single application, thus enabling language independence.
- Although IL is similar to Java bytecode in its usage by compilers, it differs from the latter in that it is designed for platform independence and language independence.
- It also differs in that it is compiled and not interpreted.

Instruction Sets

- Two types of instruction sets are included with IL; base instructions, similar to native CPU instructions, and Object model instructions used by the high-level language.
- IL includes all instructions necessary for loading, storing, initializing, and calling methods on objects.
- It also includes arithmetic and logical operations, control flow, direct memory access, exception handling and other operations.
- Unlike the common object file format used for executable content in the traditional Microsoft portable executable, the portable executable generated, after the compilation of managed code, contains both IL instructions and metadata.
- Intermediate language is associated with two IL code tools are the MSIL Assembler (ilasm.exe) and the MSIL Disassemble (ildasm.exe).
- The former generates a portable executable file from IL code and permits viewing the IL code in human-readable format, while the latter converts a portable executable file back to a text file, for viewing and modification.
- Both are automatically installed as part of Visual Studio.

Just-in-Time compilation

- The JIT compiler is part of the Common Language Runtime (CLR).
- The CLR manages the execution of all .NET applications.
- In addition to JIT compilation at runtime, the CLR is also responsible for garbage collection, type safety and for exception handling.
- JIT is a Modern software programming languages (like C# and VB.NET) utilize a human-friendly syntax that is not directly understandable by computers.
- Software commands in this human-friendly syntax are referred to as Source Code.
- Before a computer can execute the source code, special programs called compilers must rewrite it into machine instructions, also known as object code.
- This process (commonly referred to simply as “compilation”) can be done explicitly or implicitly.
- Explicit compilation converts the upper level language into object code prior to program execution.
- Ahead of time (AOT) compilers are designed to ensure that, the CPU can understand every line in the code before any interaction takes place.
- Implicit compilation is a two-step process.
  - The first step is converting the source code to intermediate language (IL) by a language-specific compiler.
  - The second step is converting the IL to machine instructions.
- The main difference with the explicit compilers is that only executed fragments of IL code are compiled into machine instructions, at runtime.
- The .NET framework calls this compiler the JIT (Just-In-Time) compiler.

JIT compilers		
Normal-JIT Compiler	Pre-JIT Compiler	Econo-JIT

- There are two types of JIT compilers available in the latest .NET Framework version.
  - Normal-JIT Compiler
    - It compiles methods in runtime as needed.
  - Pre-JIT Compiler
    - It compiles the entire assembly before it has been used.

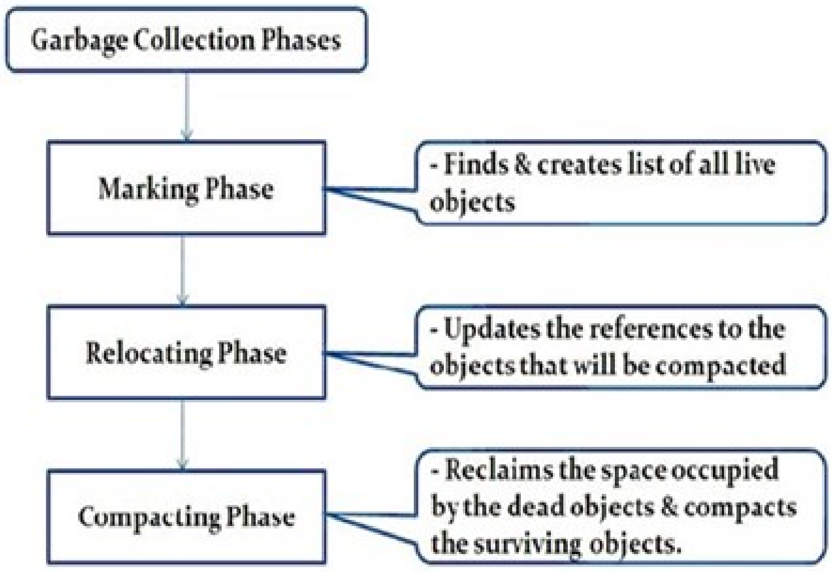
      Usually the compilation happens during application deployment.

  - - You can pre-compile an assembly using Ngen.exe tool (Native Image Generator).

- Econo-JIT
  - It was a runtime compiler which hadn′t done any optimization and removed method′s machine code when it is not required.
  - It was done to improve compilation time.
  - The compiler is obsolete since .NET 2.0 and you cannot enable it anymore.

Garbage Collection

- The .NET Framework manages memory for managed code
  - All allocations of objects and buffers made from a Managed Heap.
  - Unused objects and buffers are cleaned up automatically through Garbage Collection.
- Some of the worst bugs in software development are not possible with managed code
  - Leaked memory or objects,
  - References to freed or non-existent objects,
  - Reading of uninitialized variables.
  - Pointer less environment.
- Garbage Collector Working Phase
  - Marking Phase,
  - Relocating Phase,
  - Compacting Phase.


Garbage Collection Phases
Marking Phase
- Finds & creates list of all live objects
Relocating Phase
- Updates the references to the objects that will be compacted
Compacting Phase
- Reclaims the space occupied by the dead objects & compacts the surviving objects.

- In the marking phase garbage collector finds and creates a list of all live objects.
- In relocation phase garbage collector updates the references to the objects that will be compacted.
- In the compacting phase garbage collector reclaims the memory occupied by the dead objects and compacts the surviving objects.
- The compacting phase moves the surviving objects toward the older end of the memory segment.

Application Installation & Assemblies

- Hardware Requirements.
  - In order to install .NET framework SDK following hardware is required:
    - Computer/Processor : Intel Pentium class, 133 megahertz (MHz) or higher.
    - Minimum RAM Requirements : 128 megabytes (MB) (256 MB or higher recommended).
    - Hard Disk :Hard disk space required to install: 600 MB.
    - Hard disk space required: 370 MB.
    - Display : Video: 800x600, 256 colors.
    - Input Device : Microsoft mouse or compatible pointing device.
- Software Requirements
  - Microsoft Internet Explorer 5.01 or later is required.
  - Microsoft Data Access Components 2.6 is also required (Microsoft Data Access Components 2.7 is recommended).
- Operating System
  - Microsoft Windows® 2000, with the latest Windows service pack and critical updates available from the Microsoft Security Web page.

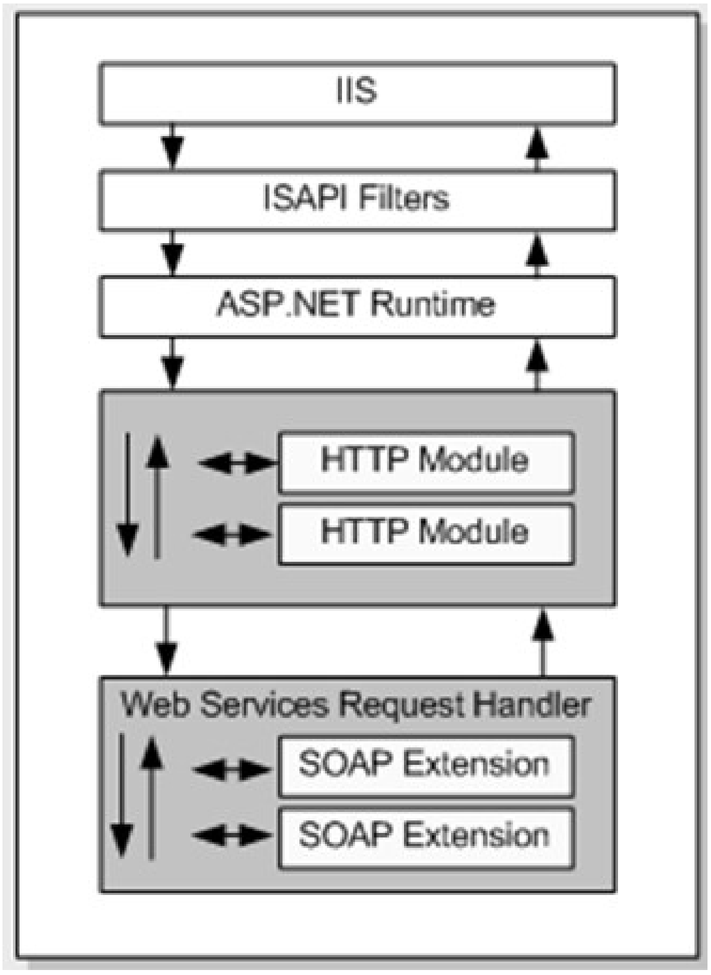
Assemblies

- DLL or EXE file.
- Smallest deployable unit in the CLR.
- Have unique version number.
- No version conflicts (known as DLL hell).
- Contains IL code to be executed.
- An assembly is a set of boundaries like
  - A security boundary - the unit to which permissions are requested and granted.
  - A type boundary - the scope of an assembly uniquely qualifies the types contained within.
  - A reference scope boundary – specifies the types that are exposed outside the assembly.
  - A version boundary - all types in an assembly are versioned together as a unit.
  - Avoid multiple version problem for DLL's.

- An assembly contains a “manifest”, which is a catalog of component metadata containing:
  - Assembly name.
  - Version (major, minor, revision, build).
  - Assembly file list - all files “contained” in the assembly.
  - Type references - mapping the managed types included in the assembly with the files that contain them.
  - Scope - private or shared.
  - Referenced assemblies.

Web Services

- Web services are components on a Web server that a client application can call by making HTTP requests across the Web.
- ASP.NET enables you to create custom Web services or to use built-in application services, and to call these services from any client application.
- Web service uses a standardized XML messaging system.XML is used to encode all communications to a Web service.
- In simple sense, Web Services are means for interacting with objects over the Internet.
- The Web Service consumers are able to invoke method calls on remote objects by using SOAP and HTTP over the Web.
- Web Service is language independent and Web Services communicate by using standard web protocols and data formats, such as: Http, Xml and Soap.



Unified Classes

- The .NET unified classes provide the foundation on which you build your applications, regardless of the language you use.
- Whether you are simply concatenating a string, or building a Windows Service or a multiple-tier Web-based application, you will be using these unified classes.
- The unified classes provide a consistent method of accessing the platform′s functionality.
- You no longer need to learn and master different API architectures to write your applications.
- You won′t need to deploy myriad dependencies, such as a separate data access library, XML parser, and network API, because all of this functionality is part of the .NET Framework.

UNIT - 2
C# Basics

Introduction

- C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages to be used on different computer platforms and architectures.
- Following reasons make C# a widely used professional language.
    - It is a modern, general purpose programming language with object oriented and component oriented.
    - C# is easy to learn and it is a structured language for effective programs.
    - It can be compiled on a variety of computer platforms.
    - Part of .Net Framework.
    - Although C# constructs closely follow traditional high-level languages, C and C++ and being an object-oriented programming language, it has strong resemblance with Java, it has numerous strong programming features that make it endearing to multitude of programmers worldwide.
- Following is the list of few important features:
    - Boolean Conditions,
    - Automatic Garbage Collection,
    - Standard Library,
    - Assembly Versioning,
    - Properties and Events,
    - Delegates and Events Management,
    - Easy-to-use Generics,
    - Indexers,
    - Conditional Compilation,
    - Simple Multithreading,
    - LINQ and Lambda Expressions,
    - Integration with Windows.

C# Program Structure

- Before we study basic building blocks of the C# programming language, let us look at a bare minimum C# program structure consists of namespace declaration, class, class methods, class attributes, main method, statements & expression and comments.

```
using System;
namespace HelloWorldApplication
{
class HelloWorld
{
static void Main(string[] args)
{
/* my first program in C# */
Console.WriteLine("Hello World");
Console.ReadKey();
}
}
}
```

It's worth to note the following points

- C# is a case sensitive with all statements and expression must end with a semicolon (;).
- Every C# program execution starts at the main method.
- It is not like java for file name it could be different from the class name.

Compile & Execute a C# Program

- To compile and execute a C# program following steps have to follow starting from visual studio and from menu bar choose file, new and project.
- A window will appear then choose Visual C# from templates, and then choose Windows.
- Choose Console Application.
- Specify a name for your project, and then choose the OK button.
- The new project appears in Solution Explorer.
- Write code in the Code Editor.
- Click the Run button or the F5 key to run the project.
- A Command Prompt window appears that contains the line Hello World.

C# Basic Syntax

- C# is an object-oriented programming language.
- In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions.
- The actions that an object may take are called methods.
- Objects of the same kind are said to have the same type or, more often, are said to be in the same class.

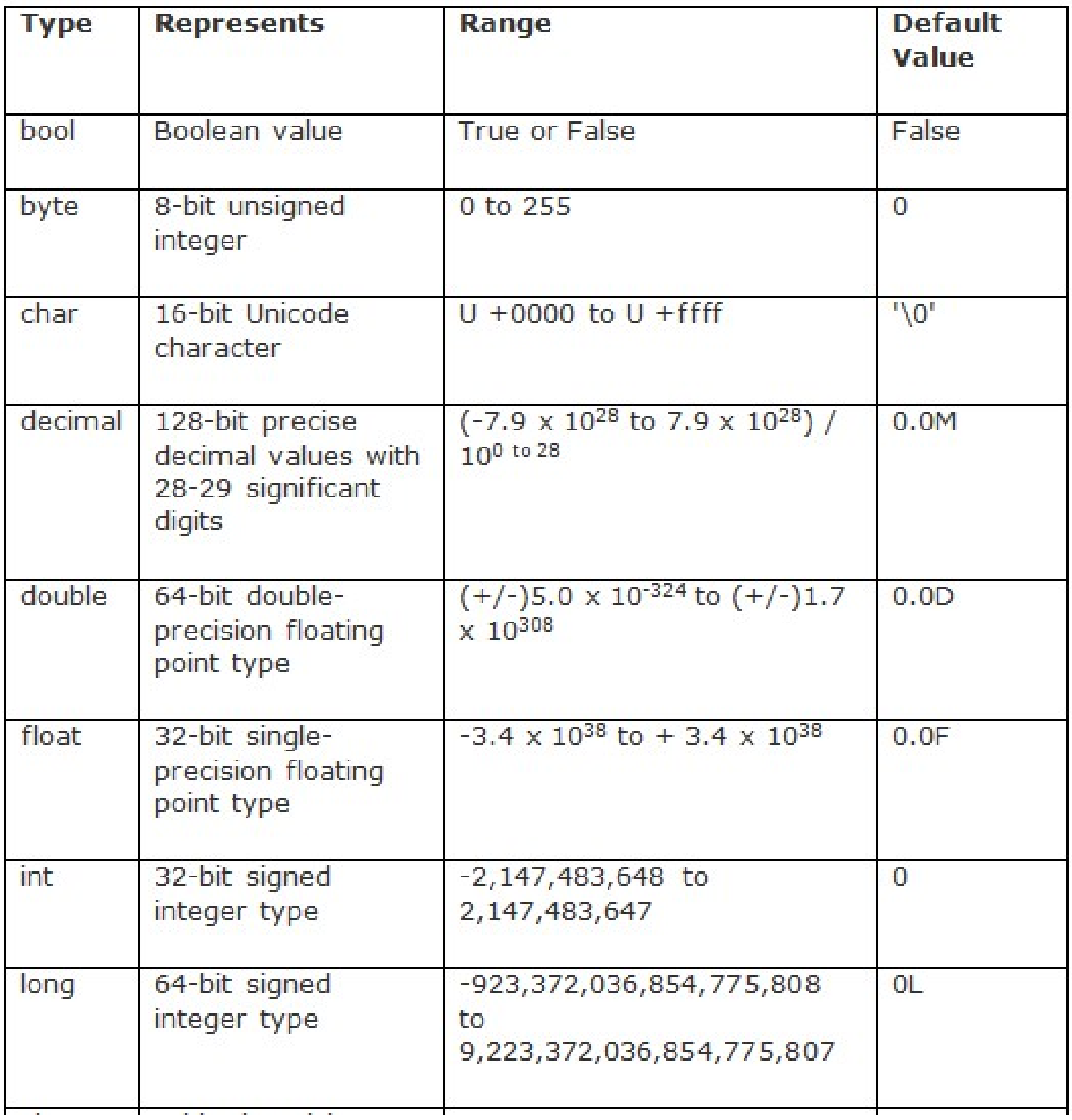
- The first statement in any C# program is “using system;” and the class keyword is used for declaring a class.
- Comments are used for explaining code, Variables and functions are data members and task of a specific code.

Data Types

- In C# programming the variables are categorized into three following types namely
  - Value types,
  - Reference types and
  - Pointer types.

Value Types

- In c# programming language value type variables can be assigned a value directly.
- They are derived from the class System.ValueType.
- The value types directly contain data.
- Some examples are int, char, float, which stores numbers, alphabets, floating point numbers, respectively.
- When you declare an int type, the system allocates memory to store the value.



Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-923,372,036,854,775,808 to 9,223,372,036,854,775,807	0L

sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

Reference Types

- The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.
- In other words, they refer to a memory location.
- Using more than one variable, the reference types can refer to a memory location.
- If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.
- Example of built-in reference types are: object, dynamic and string.
- The Object Type is the ultimate base class for all data types in C# Common Type System (CTS).

```
object obj;

obj = 100;
```

- Dynamic type can store any type of value in the dynamic data type variable.

```
dynamic <variable_name> = value;

dynamic d = 20;
```

- The String Type allows you to assign any string values to a variable.

```
String str = "NIELIT";
```

Pointer Types

- Pointer type variables store the memory address of another type.
- Pointers in C# have the same capabilities as in C or C++.

- Its syntax for declaring a pointer type is

```
type* identifier;
```

- For example

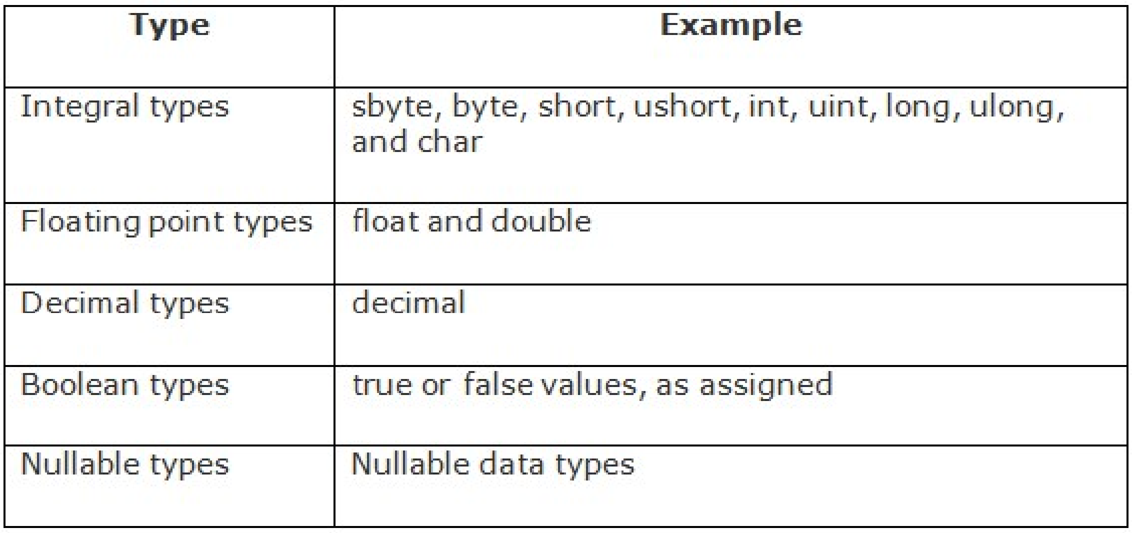
```
char* cptr;

int* iptr;
```

Identifiers

- An identifier is a name used to identify a class, variable, function, or any other user-defined item.
- The basic rules for naming classes in C# are as follows:
  - A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore.
  - The first character in an identifier cannot be a digit.
  - It must not contain any embedded space or symbol like ? - +! @ # % ^ & * ( ) [ ] { } . ; : " ' / and \.
  - However, an underscore (_) can be used.
  - It should not be a C# keyword.

Variables

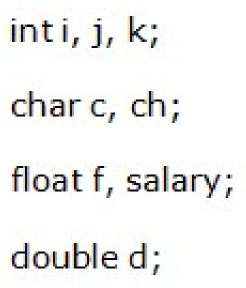
- In C# a variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in C# has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The basic value types provided in C# can be categorized as shown.



Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

Variable Definition in C#

- Syntax for variable definition in C# is:
  - < data_type > < variable_list >;
- Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, etc., and variable_list may consist of one or more identifier names separated by commas.
- Some valid variable definitions are shown in figure.



```
int i, j, k;

char c, ch;

float f, salary;

double d;
```

Variable Initialization in C#

- The general form of initialization is:
  - < data_type > < variable_name > = value;

- Variables can be initialized (assigned an initial value) in their declaration.
- The initializer consists of an equal sign followed by a constant expression.
- It is a good programming practice to initialize variables properly, otherwise sometimes program would produce unexpected result.
- Some examples are shown.

```
int d = 3, f = 5;    /* initializing d and f. */

byte z = 22;         /* initializes z. */

double pi = 3.14159; /* declares an approximation of pi. */

char x = 'x';        /* the variable x has the value 'x'. */
```

Accepting Values from User

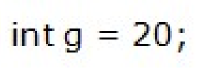
- The Console class in the System namespace provides a function ReadLine() for accepting input from the user and store it into a variable.
- For example,

```
int num;

num = Convert.ToInt32(Console.ReadLine());
```

- The function Convert.ToInt32() converts the data entered by the user to int data type, because Console.ReadLine() accepts the data in string format.

Lvalues and Rvalues in C#

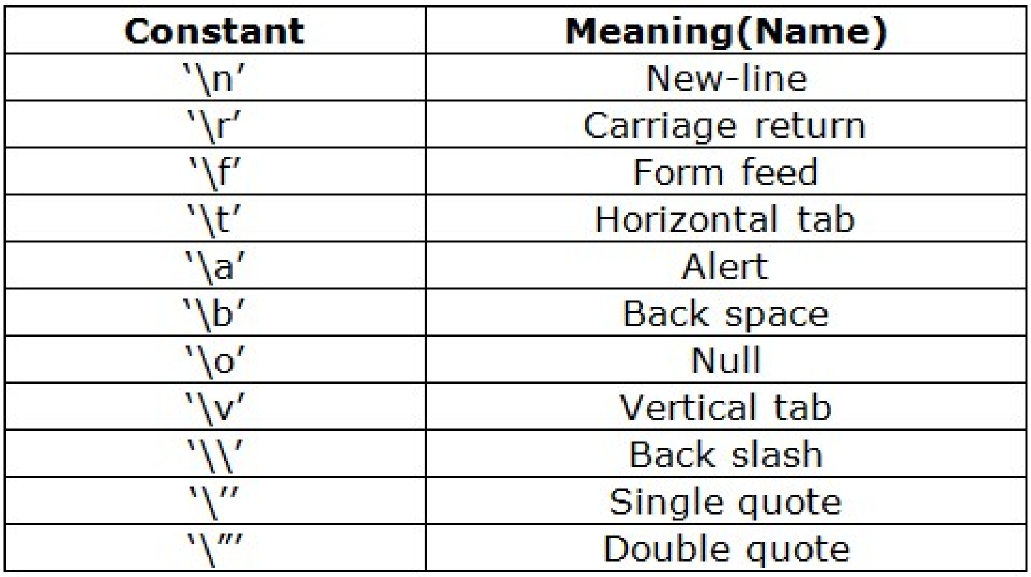
- There are two kinds of expressions in C#:
  - Lvalue:
    - An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
  - Rvalue:
    - An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.
- Variables are lvalues and so may appear on the left-hand side of an assignment.
- Numeric literals are rvalues and so may not be assigned and cannot appear on the left-hand side.
- Following is a valid statement:



```
int g = 20;
```

Constants

- In C# the constants refer to fixed values that the program may not alter during its execution.
- These fixed values are also called literals.
- Constants can be of any of the basic data types like an
  - Integer constant,
  - A floating constant,
  - A character constant, or a string literal.
- There are also enumeration constants as well.
- The constants are treated just like regular variables except that their values cannot be modified after their definition.



Constant	Meaning(Name)
'\n'	New-line
'\r'	Carriage return
'\f'	Form feed
'\t'	Horizontal tab
'\a'	Alert
'\b'	Back space
'\o'	Null
'\v'	Vertical tab
'\\'	Back slash
'\''	Single quote
'\"'	Double quote

Integer Literals

- Here an integer literal can be a
  - Decimal constant,
  - Octal constant,
  - Hexadecimal constant.
- A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and no prefix is required for decimal numbers.
- An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively.
- The suffix can be uppercase or lowercase and can be in any order.

- Here are some examples of integer literals:

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */

85          /* decimal */
0213        /* octal */
0x4b        /* hexadecimal */
30          /* int */
30u         /* unsigned int */
30l         /* long */
30ul        /* unsigned long */
```

Floating-point Literals

- A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.
- You can represent floating point literals either in decimal form or exponential form.
- Here are some examples of floating-point literals:

```
3.14159       /* Legal */
314159E-5L    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```

- Signed exponent is introduced by e or E.

- While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form you must include the integer part, the fractional part, or both.

Character constant

- Character literals are enclosed in single quotes, e.g., ‘x’ and can be stored in a simple variable of char type.
- A character literal can be
  - A plain character (e.g., ‘x’),
  - An escape sequence (e.g., ‘\t’), or
  - A universal character (e.g., ‘\u02C0’).
- List of some escape sequence codes:

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh . . .	Hexadecimal number of one or more digits

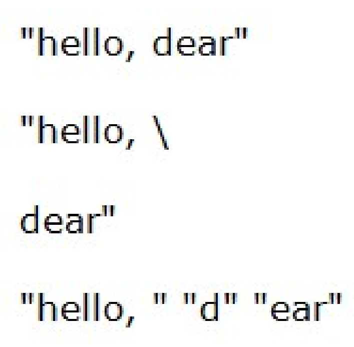
- There are certain characters in C# when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t).

String Literals

- String literals or constants are enclosed in double quotes "" or with @"".
- A string contains characters that are similar to character literals:
  - Plain characters,
  - Escape sequences,
  - Universal characters.

C# program that concats string literals

- You can break a long line into multiple lines using string literals and separating the parts using whitespaces.
- Here are some examples of string literals.



```
"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"
```

- All the three forms are identical strings.

Defining Constants

- Constants are defined using the const keyword.
- Syntax for defining a constant is:
  - const < data_type > < constant_name > = value;
- The following program demonstrates defining and using a constant in your program:

```
#include <stdio.h>

int main()

{

   const int  LENGTH = 2;

   const int  WIDTH  = 5;

   const char NEWLINE = '\n';

   int area;

   area  = LENGTH * WIDTH;

   printf("value of area : %d", area);

   printf("%c", NEWLINE);

   return 0;

}
```

- When the above code is compiled and executed, it produces the following result:

```
value of area : 10
```

Statements

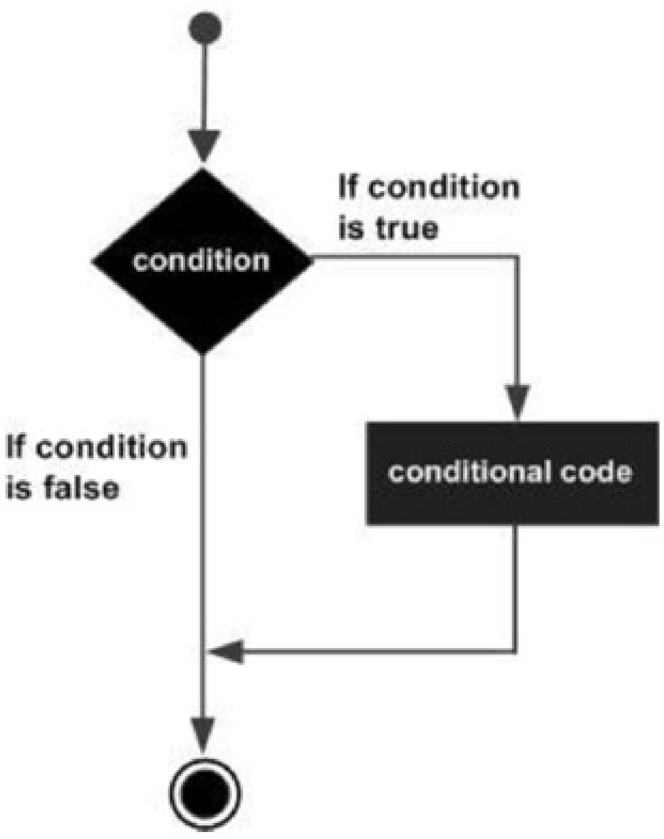
- Statements in C# are single entities that cause a change in the program's current state.
- A statement ends with a semicolon (;), which will generate a compiler error if forgotten.
- Statements may span multiple lines, which could help make your code more readable.
- Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

If Statements

- In C# if statements allow evaluation of an expression and, depending on the truth of the evaluation, the capability to branch to a specified sequence of logic.
- C# provides three forms of if statements:
  - Simple if,
  - If-then-else, and
  - If-else if-else.
- Syntax: The syntax of an if statement in C# is:

```
if(boolean_expression)
{
/* statement(s) will execute if the boolean expression is true */
}
```

- The flow diagram of if statement is shown.

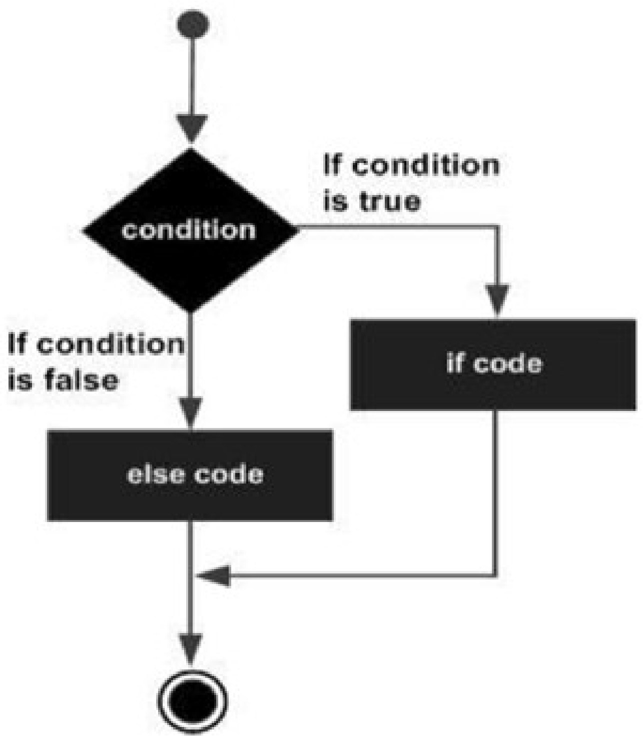


If…else statement

- The simple if statement guarantees you can only perform certain actions on a true condition.
- It's either done or it's not.
- To handle both the true and false conditions, use the syntax of an if...else statement in C# is:

```
if(boolean_expression)
{
/* statement(s) will execute if the boolean expression is true */
}
else
{
/* statement(s) will execute if the boolean expression is false */
}
```

- The flow diagram of if ... else statement is shown.



The if...else if...else Statement

- An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.
- The syntax of an if...else if...else statement in C# is:

```
if(boolean_expression 1)
{
/* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
/* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
/* Executes when the boolean expression 3 is true */
}
else
{
/* executes when the none of the above condition is true */
}
```

- When using if, else if, else statements there are few points to keep in mind.
- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

Nested if Statements

- In C# It is always legal to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).
- You can nest else if...else in the similar way as you have nested if statement.
- The syntax for a nested if statement is as follows:

```
if( boolean_expression 1)
{
/* Executes when the boolean expression 1 is true */
if(boolean_expression 2)
{
/* Executes when the boolean expression 2 is true */
}
}
```

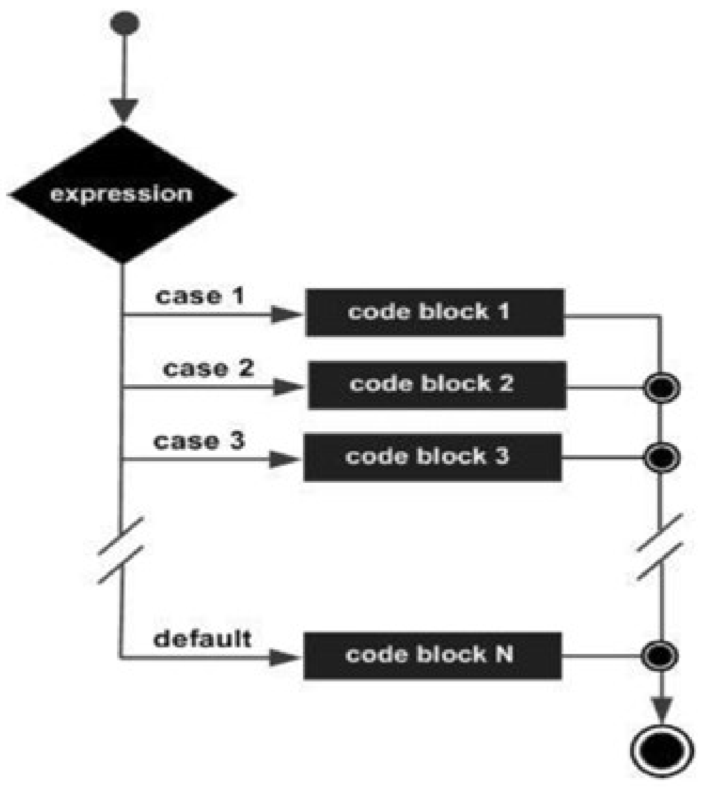
Switch Statement

- When there are many conditions to evaluate, the if-else if-else statement can become complex and verbose.
- Sometimes, a much cleaner solution is the switch statement.
- The Switch statement allows testing any integral value or string against multiple values.
- The syntax for a switch statement in C# is as follows:

```
switch(expression)
{
case constant-expression :
statement(s);
break; /* optional */
case constant-expression :
statement(s);
break; /* optional */
/* you can have any number of case statements */
default : /* Optional */
statement(s);
}
```

- A switch statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each switch case.
- The following rules apply to a switch statement:
  - The expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
  - You can have any number of case statements within a switch.
  - Each case is followed by the value to be compared to and a colon.

- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break.
- If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch.
- The default case can be used for performing a task when none of the cases is true.
- No break is needed in the default case.



Nested Switch Statement

- It is possible to have a switch as part of the statement sequence of an outer switch.
- The syntax for a nested switch statement is as follows:

```
{
case 'A':
printf("This A is part of outer switch" );
switch(ch2)
{
case 'A':
printf("This A is part of inner switch" );
break;
case 'B': /* inner B case code */
}
break;
case 'B': /* outer B case code */
}
```

C# program that uses nested switch statements

- Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

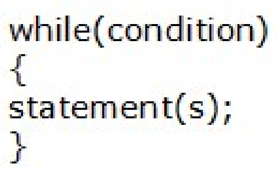
```
using System;

class Program
{
    static void Main()
    {
            int value = 5;
            switch (value)
            {
                case 1:
                        Console.WriteLine(1);
                        break;
                case 5:
                        Console.WriteLine(5);
                        break;
            }
    }
}
```

C# Loops

- In C#, there are four types of loops:
  - ➢ While loop,
  - ➢ Do loop,
  - ➢ For loop,
  - ➢ For each loop.
- Each has its own benefits for certain tasks.

While Loop

- There may be a situation, when you need to execute a block of code several numbers of times.
- The syntax of a while loop in C# is:

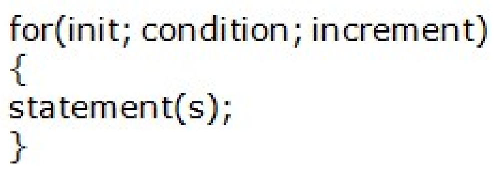


```
while(condition)
{
statement(s);
}
```

- In general, statements are executed sequentially:
  - ➢ Here, statement(s) may be a single statement or a block of statements.
  - ➢ The condition may be any expression, and true is any nonzero value.
  - ➢ The loop iterates while the condition is true.
  - ➢ When the condition becomes false, program control passes to the line immediately following the loop.

For Loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- The syntax of a for loop in C# is:



```
for(init; condition; increment)
{
statement(s);
}
```

- Initialize
  - ➢ This step allows you to declare and initialize any loop control variables.

- Condition
  - The condition is evaluated.
  - If it is true, the body of the loop is executed.
  - If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- Increment (or) Decrement
  - The condition is now evaluated again.
  - If it is true, the loop executes and the process repeats itself (increment (or) Decrement step).
  - After the condition becomes false, the for loop terminates.

Do…while Loop

- A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.
- If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again.
- This process repeats until the given condition becomes false.
- The syntax of a do...while loop in C# is:

```
do
{
statement(s);
}while( condition );
```

- Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

Nested Loops

- C# allows using one loop inside another loop.
- The syntax for a nested for loop statement in C# is as follows:

```
for ( init; condition; increment )
{
for ( init; condition; increment )
{
statement(s);
}
statement(s);
}
```

- The syntax for a nested while loop statement in C# is as follows:

```
while(condition)
{
while(condition)
{
statement(s);
}
statement(s);
}
```

Break Statement

- The example generates the following text to the console window:
- The break statement in C# has following two usages:
  - When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
  - It can be used to terminate a case in the switch statement.
  - If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- The syntax for a break statement in C# is as follows:

```
break;
```

Continue Statement

- The continue statement in C# works somewhat like the break statement.
- The syntax for a continue statement in C# is as follows:

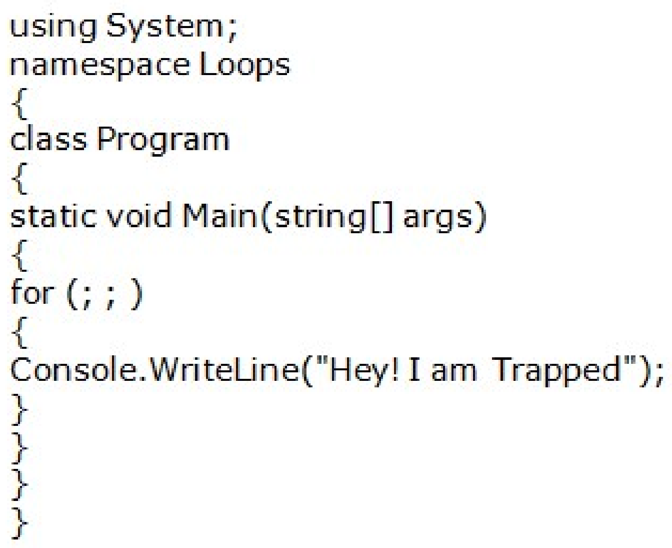
```
continue;
```

- Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.
- For the for loop, continue statement causes the conditional test and increment portions of the loop to execute.
- For the while and do...while loops, continue statement causes the program control passes to the conditional tests.

The Infinite Loop

- A loop becomes infinite loop if a condition never becomes false.

- The for loop is traditionally used for this purpose.
- Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.
- When the conditional expression is absent, it is assumed to be true.
- You may have an initialization and increment expression, but programmers more commonly use the for(;;) construct to signify an infinite loop.



```
using System;
namespace Loops
{
class Program
{
static void Main(string[] args)
{
for (; ; )
{
Console.WriteLine("Hey! I am  Trapped");
}
}
}
}
```

Designing Objects

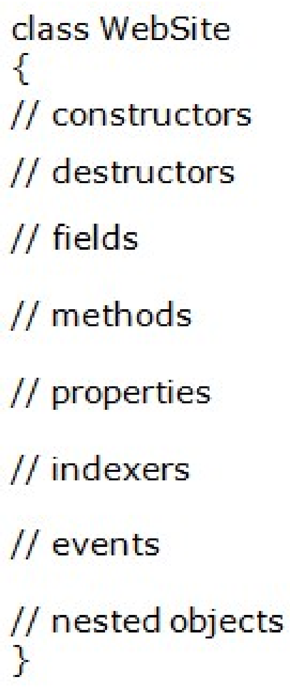
- From a software perspective, those items selected are referred to as objects, such as a custom class or struct.
- You would further refine those objects to hold members, which define attributes and behavior.

C# program that uses object

- This session shows you how to create objects and define their members.
- The attributes and behavior would map to C# language elements.
- An attribute could be a field or property, and a behavior could be a method or event.
- This is the world of objects, and by building objects that represent real-world entities, you can build more meaningful systems.

Object Members

- In .NET Objects are a self-contained with a single purpose in mind.
- All included members should be compatible.
- Here's a simple class skeleton:



```
class WebSite
{
// constructors
// destructors
// fields
// methods
// properties
// indexers
// events
// nested objects
}
```

- In this example, the class keyword tells that this is a class—a custom reference type.
- The name WebSite is the name of the class, and the class members are contained within the braces.
- This could have been a struct instead, but that would have made it a value type.

Instance and Static Members

- Each member of an object can be classified in one of two ways:
  - Instance member,
  - Static member.
- When a copy of an object is created, it is considered instantiated.
- At that point, the object exists as a sole entity, with its own set of attributes and behavior.
- If a second instance of that object were created, it would normally have a separate set of data from that of the first object instance.
- In C#, object members belong to the instance, unless modified as static.
- An example of instance objects is a Customer class, where every customer has different information.

Fields

- Fields comprise the primary “data” portion of a class.
- They are the state of an object.
- They are members of a class as opposed to local variables, which are defined inside of methods and properties.
- Constants are efficient.
- Their values are known at compile time.
- This enables certain optimizations unavailable to other types of fields.
- By definition, constants are also static.
- Readonly fields are similar to constant fields in that they can′t be modified after initialization.
- The biggest difference between the two is when they′re initialized: Constants are initialized during compilation, and readonly fields are initialized during runtime.

Methods

- Methods are some of the most common object members you′ll work with.
- A C# method is similar to functions, procedures, subroutines, and so on in other programming languages.
- The preceding method doesn′t return any value, which is why you see the void return type.
- The name is MyMethod, and you would replace that with a meaningful name of what the method does.
- This method doesn′t accept parameters. “Coding Methods and Custom Operators”.

- For now, here's a simple example:

```
void MyMethod()
{
// statements go here
}
```

Properties

- A C# property enables you to protect access to the state of your object.
- You use them like fields, but they operate much like methods.
- The following sections show you how to declare and use properties.

Declaring Properties.

```
private string m_description;
public string Description
{
get
{
return m_description;
}
set
{
m_description = value;
}
}
```

- The get and set accessors can have any logic you want to define.
- A get accessor returns a value, and the set accessor sets a value.
- Notice the value keyword in the set accessor; it holds whatever value was assigned to the property.

```
static void Main()
{
WebSite site = new WebSite();

site.Description = "cool site";
string desc = site.Description;
}
```

Auto-implemented properties.

- public int Rating { get; set; }
- This Rating property encapsulates some value of type int.

- Because calling code doesn′t have access to the backing store anyway, knowing its name doesn′t matter.
- The C# compiler will create an int field behind the scenes for us.

Indexers

- Indexers behave like arrays in that they use the square-bracket syntax to access their members.
- The .NET collection classes use indexers to accomplish the same goals.
- Their elements are accessed by index.
- Indexers are implemented like properties because they have get and set accessors, following the same syntax.
- Given an index, they obtain and return an appropriate value with a get access or.
- Similarly, they set the value or responding to the index with the value passed into the indexer.

Designing Object - Oriented Programs

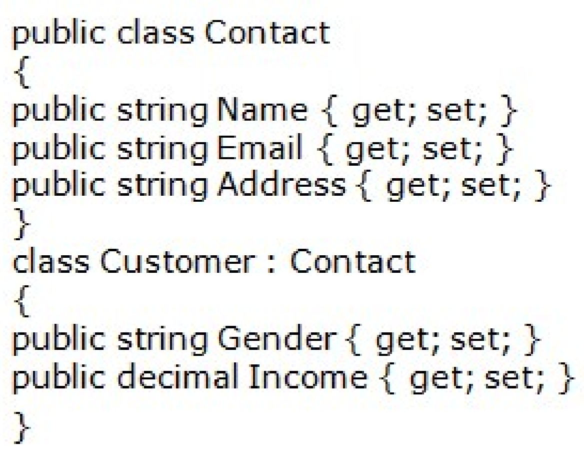
- The following sections drill down on these subjects, which are three of the principles of object-oriented programming:
    - Inheritance,
    - Encapsulation, and
    - Polymorphism
- Abstraction, a fourth pillar of object-oriented programming.
- Building objects from a higher-level perspective so that you don't have to be concerned about the lower-level details.
- In this session, you learn about building objects and managing the public members to create new abstractions.

Inheritance

- Inheritance is an object-oriented principle relating to how one class, a derived class, can share the characteristics and behavior from another class, a base class.
- This can be thought of as an “is a” relationship, because the derived class can be identified by both its class type and its base class type.
- The benefits gained by this are the ability to reuse the base class members and to add additional members to the derived class.
- The derived class then becomes a specialization of the base class (parent).
- This specialization can continue for as many levels as necessary, each new level derived from the base class above it.

Base Classes

- Normal base classes may be instantiated themselves, or inherited.
- Derived classes inherit each base class member marked with protected or greater access.
- The derived class is specialized to provide more functionality, in addition to what its base class provides.
- Here's an example for base class.

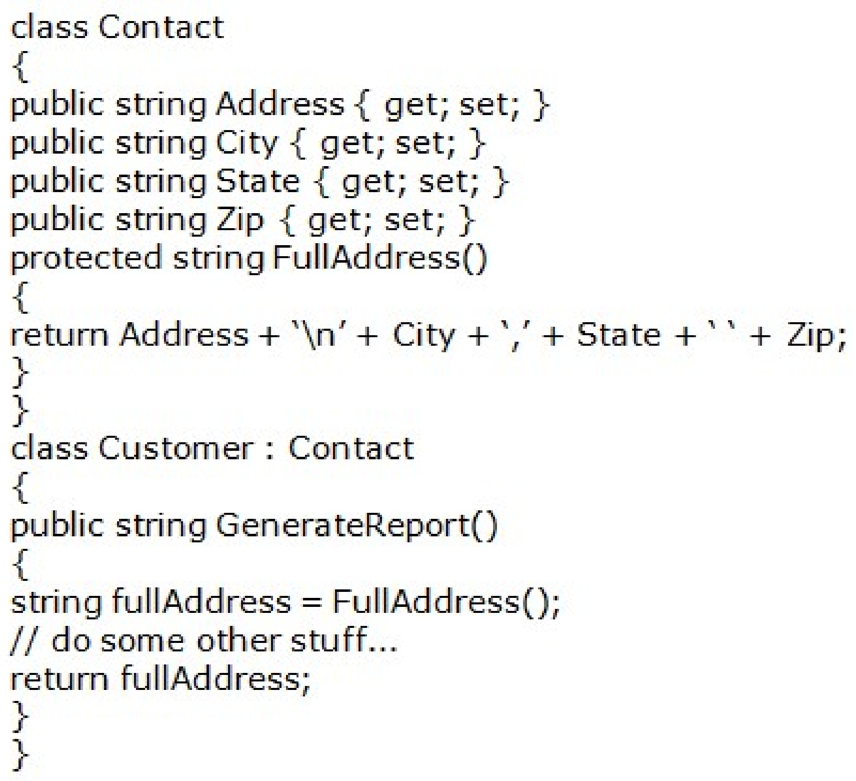


```
public class Contact
{
public string Name { get; set; }
public string Email { get; set; }
public string Address { get; set; }
}
class Customer : Contact
{
public string Gender { get; set; }
public decimal Income { get; set; }
}
```

- A derived class declares that it inherits from a base class by adding a colon, :, and the base class name after the derived class name.

Calling Base Class Members

- Derived classes can access the members of their base class if those members have protected or greater access.
- A later section on the object-oriented principle of encapsulation goes into greater depth on access modifiers.
- That a base class member with a protected or public modifier can be accessed by derived classes.
- Just use the member name as if that member were a part of the derived class itself.
- Here's an example:



```
class Contact
{
public string Address { get; set; }
public string City { get; set; }
public string State { get; set; }
public string Zip { get; set; }
protected string FullAddress()
{
return Address + ‘\n’ + City + ‘,’ + State + ‘ ‘ + Zip;
}
}
class Customer : Contact
{
public string GenerateReport()
{
string fullAddress = FullAddress();
// do some other stuff...
return fullAddress;
}
}
```

- In this example, the GenerateReport method of the Customer class calls the FullAddress method in its base class, Contact.

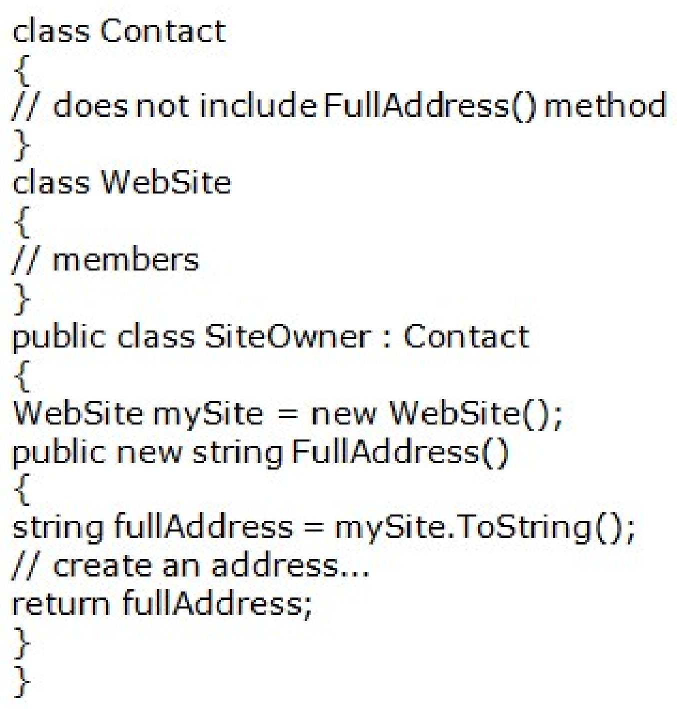
- All classes have full access to their own members without qualification.
- Qualification refers to using a class name with the dot operator to access a class member—MyObject.SomeMethod, for instance.

Hiding Base Class Members

- Derived class members have the same name as a corresponding base class member, meaning that they are redefining the base class method in the derived class.
- When using an instance of the derived class, you invoke the derived classes specialized behavior and not the behavior of that method in the base class.
- In this case, the derived member is said to be “hiding” the base class member.
- When hiding occurs, the derived member is masking the functionality of the base class member.
- Users of the derived class won′t be able to see the hidden member; they′ll see only the derived class member.

Versioning

- Versioning, in the context of inheritance, is a C# mechanism that allows modification of classes (creating new versions) without accidentally changing the meaning of the code.
- Hiding a base class member with the methods previously described generates a warning message from the compiler.
- This is because of the C# versioning policy.
- It′s designed to eliminate a class of problems associated with modifications to base classes.
- Here′s the scenario: A developer creates a class that inherits from a third-party library.



```
class Contact
{
// does not include FullAddress() method
}
class WebSite
{
// members
}
public class SiteOwner : Contact
{
WebSite mySite = new WebSite();
public new string FullAddress()
{
string fullAddress = mySite.ToString();
// create an address...
return fullAddress;
}
}
```

- We assume that the Contact class represents the third-party library.

Encapsulation

- Encapsulation is the object-oriented principle associated with hiding the internals of an object from the outside world.
- C# has several mechanisms for supporting encapsulation, including properties, indexers, methods, and access modifiers.
- In this section, you see how to use the features of C# to achieve encapsulation.
- Several reasons to take advantage of C#'s built-in mechanisms
  - Good encapsulation reduces coupling.
  - By clearly defining a public interface (what other code sees) and hiding everything else users can write code with less dependency on that object.
  - Internal implementation of an object can freely change.
  - This reduces the possibility of breaking someone else's code.
  - An object has a much cleaner interface.
  - Users see only those members that are exposed, which reduces the amount of understanding they need to use the object.
  - It simplifies reuse.

Data Hiding

- One of the most useful forms of encapsulation is data hiding.

- Most of the time, users shouldn't have access to the internal data of an object.
- Data represents the state of an object, and an object normally has full control of its own state to ensure consistency.
- Anytime access to data is open, the potential of someone else wreaking havoc with the operation of that object increases.
- Sometimes it's logical and necessary to expose object data—especially if it's necessary to expose constants, enumerations, and read-only fields.
- Perhaps a design goal is to increase the efficiency of data access for a field that's accessed frequently.

Modifiers Supporting Encapsulation

- C# manage object encapsulation with appropriate use of C# access modifiers, which specify which code can access class members.
- They also control the method of access.
- You apply different sets of access modifiers, depending on whether you are working with objects or object members.
- You see access modifiers applicable to object members first, followed by which modifiers apply to types.

Public Access Specifier

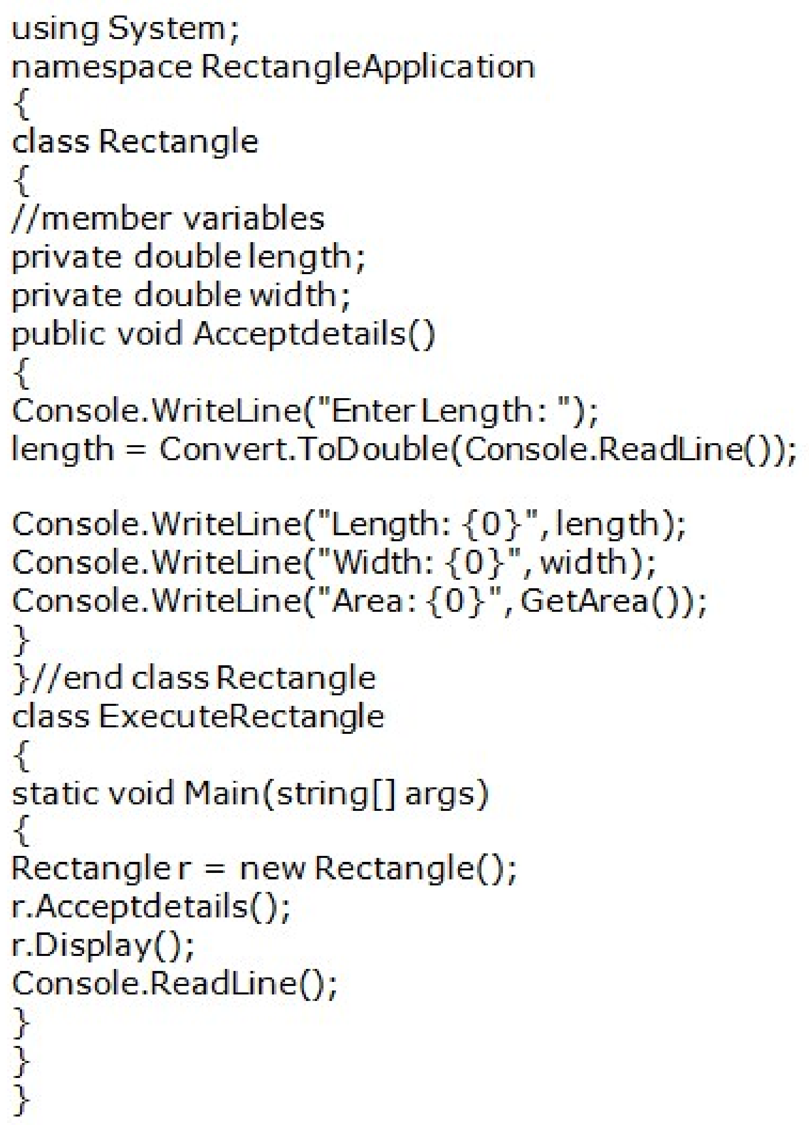
- Public access specifier allows a class to expose its member variables and member functions to other functions and objects.
- Any public member can be accessed from outside the class.
- The following example illustrates this:

```
using System;
namespace RectangleApplication
{
class Rectangle
{
//member variables
public double length;
public double width;
public double GetArea()
{
return length * width;

}
public void Display()
{
Console.WriteLine("Length: {0}", length);
Console.WriteLine("Width: {0}", width);
Console.WriteLine("Area: {0}", GetArea());
}
}//end class Rectangle
class ExecuteRectangle
{
static void Main(string[] args)
{
Rectangle r = new Rectangle();
r.length = 4.5;
r.width = 3.5;
r.Display();
Console.ReadLine();
  }
}
```

- In the preceding example, The member variables length and width are declared public, so they can be accessed from the function Main() using an instance of the Rectangle class, named r.
- The member function Display() and GetArea() can also access these variables directly without using any instance of the class.
- The member functions Display() is also declared public, so it can also be accessed from Main() using an instance of the Rectangle class, named r.

Private Access Specifier

- Private access specifier allows a class to hide its member variables and member functions from other functions and objects.
- Only functions of the same class can access its private members.
- Even an instance of a class cannot access its private members.
- The following example illustrates this:



```
using System;
namespace RectangleApplication
{
class Rectangle
{
//member variables
private double length;
private double width;
public void Acceptdetails()
{
Console.WriteLine("Enter Length: ");
length = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Length: {0}", length);
Console.WriteLine("Width: {0}", width);
Console.WriteLine("Area: {0}", GetArea());
}
}//end class Rectangle
class ExecuteRectangle
{
static void Main(string[] args)
{
Rectangle r = new Rectangle();
r.Acceptdetails();
r.Display();
Console.ReadLine();
}
}
}
```

- In the preceding example, the member variables length and width are declared private, so they cannot be accessed from the function Main().
- The member functions AcceptDetails() and Display() can access these variables.
- Since the member functions AcceptDetails() and Display() are declared public, they can be accessed from Main() using an instance of the Rectangle class, named r.

Protected Access Specifier

- Protected access is a little less restrictive than private access but more restrictive than public.
- The only way to use a protected member is via members of the same class or through inheritance.
- A derived class has full access to protected base class members.
- Another feature of this particular example is that the protected member is a property and not a field.
- This gives Contact the capability to encapsulate its implementation, yet share access at a level necessary for the job.

- Had the Age information been exposed as a protected field, the coupling between derived classes, such as Customer, would constrain Contact and make maintenance more difficult.

```
class Contact
{
protected int Age { get; set; }
}
class WebSite
{
// members
}
class Customer : Contact
{
public bool IsContentAppropriate(WebSite site)
{
return Age > 18;
}
}
```

Internal Access Specifier

- Internal access restricts visibility to only code within the same assembly.
- I briefly described an assembly as a unit of
  - Deployment,
  - Execution,
  - Identity,
  - and Security.
- An assembly is also a unit of containment where you may restrict access via internal modifiers.
- Here's an example of internal members:

```
sealed class CustomerStats
{
internal bool Gender { get; set; }
internal decimal Income { get; set; }
internal int NumberOfVisits { get; set; }
}
```

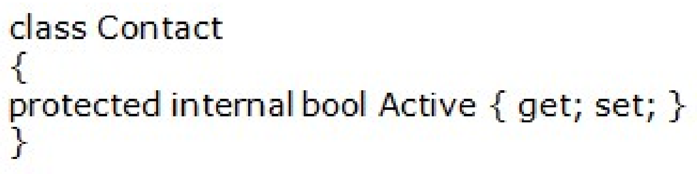
- To use internal access effectively, you create a class library project in VS2008.
- Any code inside of that class library can access your internal members, but other code that references the assembly can't.

Protected Internal Access Specifier

- The protected internal access specifier allows a class

- ➢ To hide its member variables.
- ➢ Member functions from other class objects and functions.

- Here's an example of Protected Internal Access Specifier:



```
class Contact
{
protected internal bool Active { get; set; }
}
```

- Except a child class within the same application.
- This is also used while implementing inheritance.

Polymorphism

- The word polymorphism means having many forms.
- In object-oriented programming paradigm, polymorphism is often expressed as
  - ➢ 'one interface multiple functions'.
- Polymorphism can be static or dynamic.
- In static polymorphism, the response to a function is determined at the compile time.
- In dynamic polymorphism, it is decided at run-time.

Static Polymorphism

- In C# the mechanism of linking a function with an object during compile time is called early binding.
- It is also called static binding.
- C# provides two techniques to implement static polymorphism.
- These are:
  - ➢ Function overloading.
  - ➢ Operator overloading.

Function Overloading

- You can have multiple definitions for the same function name in the same scope.
- The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.
- You cannot overload function declarations that differ only by return type.
- Following is the example where same function print() is being used to print different data types:

```
using System;
namespace PolymorphismApplication
{
class Printdata
{
void print(int i)
{
Console.WriteLine("Printing int: {0}", i );
}
static void Main(string[] args)
{
Printdata p  = new Printdata();
// Call print to print integer
p.print(5);
// Call print to print float
p.print(500.263);
// Call print to print string
p.print("Hello C++");
Console.ReadKey();
}
}
}
```

Operator Overloading

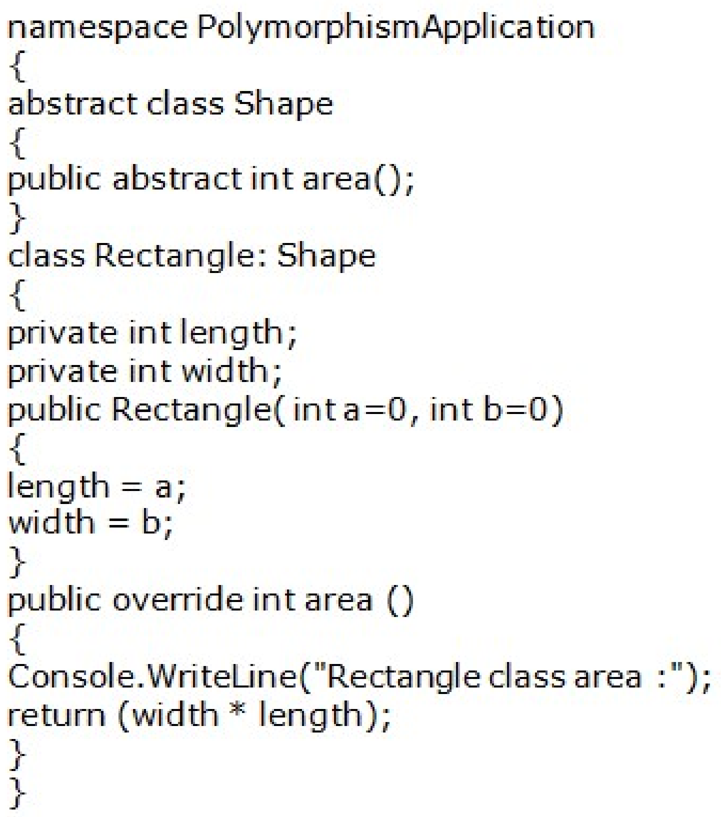
- You can redefine or overload most of the built-in operators available in C#.
- Thus a programmer can use operators with user-defined types as well.
- Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined.
- Like any other function, an overloaded operator has a return type and a parameter list.
- The above function implements the addition operator (+) for a user-defined class Box.
- It adds the attributes of two Box objects and returns the resultant Box object.
- For example, look at the following function:

```
public static Box operator+ (Box b, Box c)
{
Box box = new Box();
box.length = b.length + c.length;
box.breadth = b.breadth + c.breadth;
box.height = b.height + c.height;
return box;
}
```

Dynamic Polymorphism

- C# allows you to create abstract classes that are used to provide partial class implementation of an interface.

- Implementation is completed when a derived class inherits from it.
- Abstract classes contain abstract methods, which are implemented by the derived class.
- The derived classes have more specialized functionality.
- Please note the following rules about abstract classes:
  - You cannot create an instance of an abstract class.
  - You cannot declare an abstract method outside an abstract class.
  - When a class is declared sealed, it cannot be inherited, abstract classes cannot be declared sealed.
- The following program demonstrates an abstract class:
- When the above code is compiled and executed, it produces the following result:



```
namespace PolymorphismApplication
{
abstract class Shape
{
public abstract int area();
}
class Rectangle: Shape
{
private int length;
private int width;
public Rectangle( int a=0, int b=0)
{
length = a;
width = b;
}
public override int area ()
{
Console.WriteLine("Rectangle class area :");
return (width * length);
}
}
```

Object and Classes

Class

- class definition
- A class definition starts with the keyword Class followed by the class name; and the class body, ended by the End Class statement.
- Following is the general form of a class definition:

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ]
[ MustInherit | NotInheritable ] [ Partial ] _
Class name [ ( Of typelist ) ]
[ Inherits classname ]     [ Implements interfacenames ]     [ statements ]
End Class
```

- Where,
    - attribute list is a list of attributes that apply to the class. Optional.
    - access modifier defines the access levels of the class, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
    - Shadows indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
    - Must Inherit specifies that the class can be used only as a base class and that you cannot create an object directly from it, i.e., an abstract class. Optional.

Object

- It is a real time entity.
- An object can be considered a "thing" that can perform a set of related activities.
- The set of activities that the object performs defines the object's behavior.
- In pure OOP terms an object is an instance of a class.

Example for an object

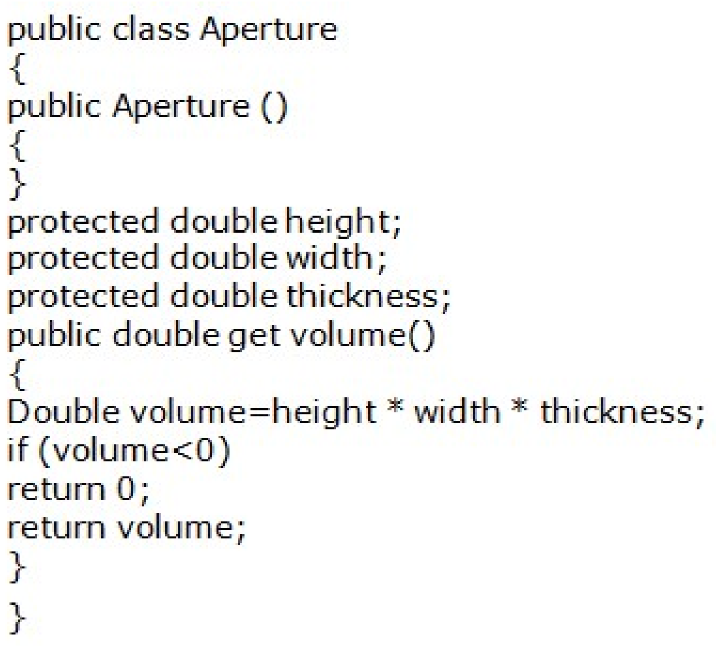
```
public class student
{
}
student objstudent=new student ();
```

Member Functions and Encapsulation

- A member function of a class is a function that has its

    - Definition,
    - Prototype within the class definition like any other variable.

- It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.
- Member variables are attributes of an object (from design perspective) and they are kept private to implement encapsulation.
- These variables can only be accessed using the public member functions.

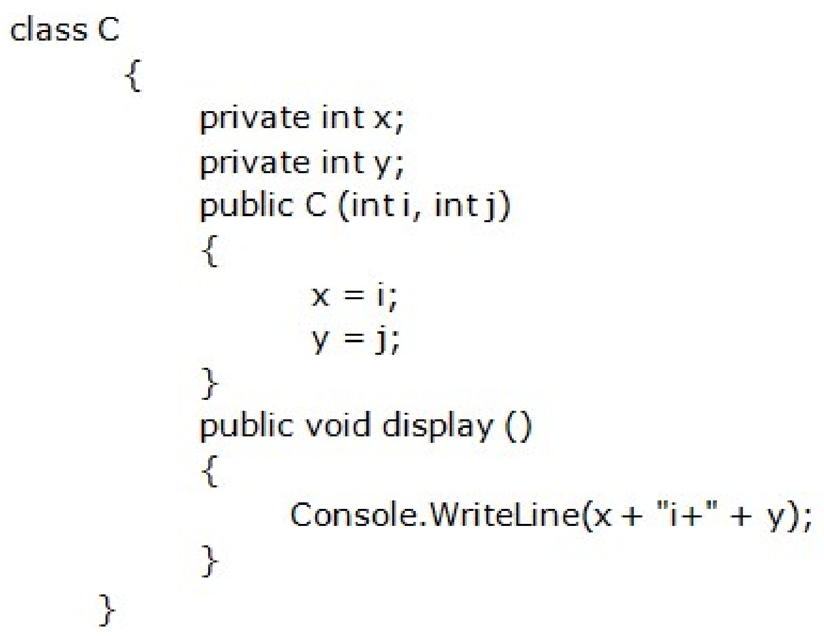


```
public class Aperture
{
public Aperture ()
{
}
protected double height;
protected double width;
protected double thickness;
public double get volume()
{
Double volume=height * width * thickness;
if (volume<0)
return 0;
return volume;
}
}
```

- In this example we encapsulate some data such as height, width, thickness and method Get Volume.
- Other methods or objects can interact with this object through methods that have public access modifier.

Constructors and Destructors

- In C# Classes have complicated internal structures.
- Constructors and destructors are special member functions of classes that are used to construct and destroy class objects.
- Construction may involve memory allocation and initialization for objects.
- Destruction may involve cleanup and deallocation of memory for objects.
- Constructors and destructors do not have return types nor can they return values.
- References and pointers cannot be used on constructors and destructors because their addresses cannot be taken.
- Constructors cannot be declared with the keyword virtual.



```
class C
    {
          private int x;
          private int y;
          public C (int i, int j)
          {
                x = i;
                y = j;
          }
          public void display ()
          {
              Console.WriteLine(x + "i+" + y);
          }
    }
```

Array

- An array stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data.
- Instead of declaring individual variables, such as number0, number1, ..., and number99.
- Declare one array variable such as numbers and use numbers [0], numbers [1], and ..., numbers [99] to represent individual variables.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.

Declaring Array

- To declare an array in C#, you can use the following syntax:
  - Datatype[ ]arrayName;
- Where,
  - Datatype is used to specify the type of elements to be stored in the array.
  - [ ] specifies the rank of the array.
  - The rank specifies the size of the array.
  - arrayName specifies the name of the array.
- For example,
  - double[ ] balance;

Initializing an Array

- Declaring an array does not initialize the array in the memory.
- When the array variable is initialized, you can assign values to the array.
- Array is a reference type, so you need to use the new keyword to create an instance of the array.
- For example,
  - double[ ] balance = new double[10];

Assigning Values to an Array

- You can assign values to individual array elements, by using the index number, like:
  - double[ ] balance = new double[10]; balance[0] = 4500.0;

- You can assign values to the array at the time of declaration, like:
    - double[ ] balance = { 2340.0, 4523.69, 3421.0};
- You can also create and initialize an array, like:
    - int [ ] marks = new int[5] { 99, 98, 92, 97, 95};
- When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type.
- For example for an int array all elements would be initialized to 0.

Accessing Array Elements

- An element is accessed by indexing the array name.
- This is done by placing the index of the element within square brackets after the name of the array.
    - double salary = balance[9];

Multi-dimensional arrays

- C# allows multidimensional arrays.
- Multi-dimensional arrays are also called rectangular array.
- You can declare a 2-dimensional array of strings as: string [,] names; or,
- A 3-dimensional array of int variables: int [, ,] m;
- The simplest form of the multidimensional array is the 2-dimensional array.
- A 2-dimensional array is, in essence, a list of one-dimensional arrays.
- A 2-dimensional array can be thought of as a table, which will have x number of rows and y number of columns.
- Following is a 2-dimentional array, which contains 3 rows and 4 columns:

Jagged Arrays

- A Jagged array is an array of arrays.
- You can declare a jagged array scores of int values as:
    - int [ ][ ] scores;

Passing arrays to functions

- You can pass an array as a function argument in C#.
- The following example demonstrates this:

```
using System;
namespace ArrayApplication
{
class MyArray
{
double getAverage(int[] arr, int size)
{
int i;
double avg;
int sum = 0;
for (i = 0; i < size; ++i)
{
sum += arr[i];
}
avg = (double)sum / size;
return avg;
}
static void Main(string[] args)
{
MyArray app = new MyArray();
int [] balance = new int[]{1000, 2, 3, 17, 50};
double avg;
avg = app.getAverage(balance, 5 ) ;

Console.WriteLine( "Average value is: {0} ", avg );
Console.ReadKey();
}
}
}
```

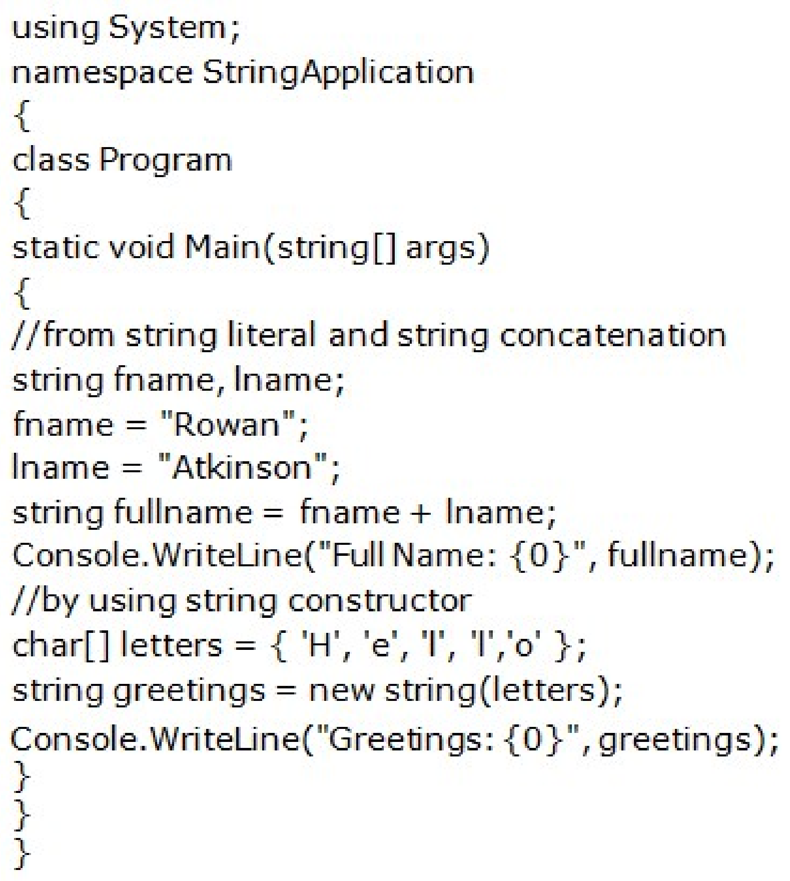
- When the above code is compiled and executed, it produces the following result:
- Average value is: 214.4

Array Class

- The Array class is the base class for all the arrays in C#.
- It is defined in the System namespace.
- The Array class provides various properties and methods to work with arrays.
  - IsFixedSize Gets a value indicating whether the Array has a fixed size.
  - IsReadOnly Gets a value indicating whether the Array is read-only.
  - Length Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
  - LongLength Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
  - Rank Gets the rank (number of dimensions) of the Array.

Strings

- [H][e][l][l][o]
- In C#, you can use strings as array of characters, however, more common practice is to use the string keyword to declare a string variable.
- The string keyword is an alias for the System.String class.
- Creating a String Object You can create string object using one of the following methods:
  - By assigning a string literal to a String variable.
  - By using a String class constructor.
  - By using the string concatenation operator (+).
  - By retrieving a property or calling a method that returns a string.
  - By calling a formatting method to convert a value or object to its string representation.
- Example for Creating a String Object



```
using System;
namespace StringApplication
{
class Program
{
static void Main(string[] args)
{
//from string literal and string concatenation
string fname, lname;
fname = "Rowan";
lname = "Atkinson";
string fullname = fname + lname;
Console.WriteLine("Full Name: {0}", fullname);
//by using string constructor
char[] letters = { 'H', 'e', 'l', 'l','o' };
string greetings = new string(letters);
Console.WriteLine("Greetings: {0}", greetings);
}
}
}
```

- The String class has the following two properties:
  - Chars gets the Char object at a specified position in the current String object.
  - Length gets the number of characters in the current String object.

System Collections

- Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.
- These classes create collections of objects of the Object class, which is the base class for all data types in C#.
- Collection classes are specialized classes for data storage and retrieval.
- These classes provide support for stacks, queues, lists, and hash tables.
- Most collection classes implement the same interfaces.

ArrayList

- It represents an ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array.
- However unlike array you can add and remove items from a list at a specified position using an index and the array resizes itself automatically.
- It also allow dynamic memory allocation, adding, searching and sorting items in the list.
- The following table lists some of the commonly used properties of the ArrayList class:

Property	Description
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

S.N	Method Name & Purpose
1	public virtual int Add(object value); Adds an object to the end of the ArrayList.
2	public virtual void AddRange(ICollection c); Adds the elements of an ICollection to the end of the ArrayList.
3	public virtual void Clear(); Removes all elements from the ArrayList.

4	public virtual bool Contains(object item); Determines whether an element is in the ArrayList.
5	public virtual ArrayList GetRange(int index, int count); Returns an ArrayList which represents a subset of the elements in the source ArrayList.
6	public virtual int IndexOf(object); Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.

Hashtable

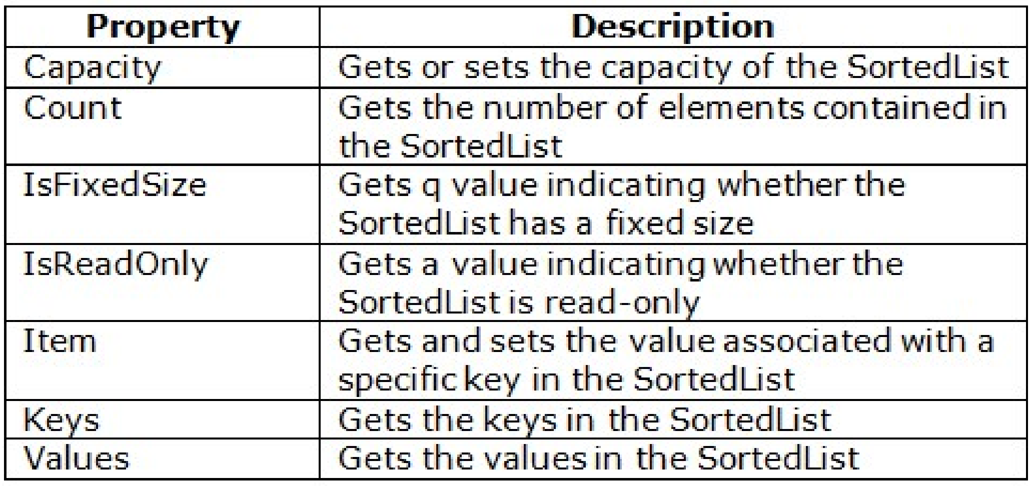
- The Hashtable class represents a collection of key-and-value pairs
- Based on the hash code of the key that are organized.
- It uses the key to access the elements in the collection.
- The following table lists some of the commonly used properties of the Hashtable class:

Property	Description
Count	Gets the number of key-and-value pairs contained in the Hashtable.
IsFixedSize	Gets a value indicating whether the Hashtable has a fixed size.
IsReadOnly	Gets a value indicating whether the Hashtable is read-only.
Item	Gets or sets the value associated with the specified key.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.

- A hash table is used when you need to access elements by using key, and you can identify a useful key value.
- Each item in the hash table has a key/value pair.
- The key is used to access the items in the collection.

SortedList

- The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.
- A sorted list is a combination of an array and a hash table.



Property	Description
Capacity	Gets or sets the capacity of the SortedList
Count	Gets the number of elements contained in the SortedList
IsFixedSize	Gets q value indicating whether the SortedList has a fixed size
IsReadOnly	Gets a value indicating whether the SortedList is read-only
Item	Gets and sets the value associated with a specific key in the SortedList
Keys	Gets the keys in the SortedList
Values	Gets the values in the SortedList

- It contains a list of items that can be accessed using a key or an index.
- If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable.
- The collection of items is always sorted by the key value.

Stack

- It represents a last-in, first out collection of object.
- It is used when you need a last-in, first-out access of items.
- When you add an item in the list, it is called pushing the item.
- When you remove it, it is called popping the item.
- The following table lists some of the commonly used properties of the Stack class:

Property	Description
Count	Gets the number of elements contained in the Stack.

Queue

- It represents a first-in, first out collection of object.
- It is used when you need a first-in, first-out access of items.
- When you add an item in the list, it is called enqueue.
- When you remove an item, it is called dequeue.
- The following table lists some of the commonly used properties of the Queue class:

Property	Description
Count	Gets the number of elements contained in the Queue.

BitArray

- The BitArray class manages a compact array of bit values.
- Where true indicates that the bit is on (1) and false indicates the bit is off (0).
- It is used when you need to store the bits but do not know the number of bits in advance.
- You can access items from the BitArray collection by using an integer index, which starts from zero.

Delegates and Events

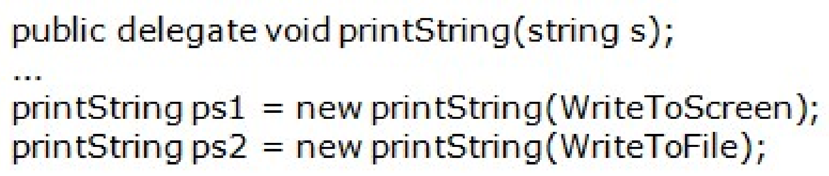
- C# delegates are similar to pointers to functions in C or C++.
- A delegate is a reference type variable that holds the reference to a method.
- The reference can be changed at runtime.
- Delegates are especially used for implementing events and the call-back methods.
- All delegates are implicitly derived from the System.Delegate class.
    - Declaring Delegates.
    - Instantiating Delegates.
    - Multicasting of a Delegate.
    - Use of Delegate.

Declaring Delegates

- Syntax for delegate declaration is:
    - delegate < return type > < delegate-name > < parameter list >
- Example
    - public delegate int MyDelegate (string s);
- Delegate declaration determines the methods that can be referenced by the delegate.
- A delegate can refer to a method, which have the same signature as that of the delegate.

Instantiating Delegates

- Once a delegate type has been declared, a delegate object must be created with the new keyword and be associated with a particular method.
- When creating a delegate, the argument passed to the new expression is written like a method call, but without the arguments to the method.
- For example:



```
public delegate void printString(string s);
...
printString ps1  = new printString(WriteToScreen);
printString ps2  = new printString(WriteToFile);
```

- Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

```
using System;
delegate int NumberChanger(int n);
namespace DelegateAppl
{
  class TestDelegate
  {
    static int num = 10;
    public static int AddNum(int p)
    {
      num += p;
      return num;
    }

    public static int MultNum(int q)
    {
      num *= q;
      return num;
    }
    public static int getNum()
    {
      return num;
    }

    static void Main(string[] args)
    {
      //create delegate instances
      NumberChanger nc1 = new NumberChanger(AddNum);
      NumberChanger nc2 = new NumberChanger(MultNum);

      //calling the methods using the delegate objects
      nc1(25);
      Console.WriteLine("Value of Num: {0}", getNum());
      nc2(5);
      Console.WriteLine("Value of Num: {0}", getNum());
      Console.ReadKey();
    }
  }
}
```

- When the above code is compiled and executed, it produces the following result:

```
Value of Num: 35
Value of Num: 175
```

Multicasting of a Delegate

- The following program demonstrates multicasting of a delegate:

```
using System;
delegate int NumberChanger(int n);
namespace DelegateAppl
{
  class TestDelegate
  {
    static int num = 10;
    public static int AddNum(int p)
    {
      num += p;
      return num;
    }
    public static int MultNum(int q)
    {
      num *= q;
      return num;
    }
        public static int getNum()
    {
      return num;
    }
    static void Main(string[] args)
    {
      //create delegate instances
      NumberChanger nc;
      NumberChanger nc1 = new NumberChanger(AddNum);
      NumberChanger nc2 = new NumberChanger(MultNum);
      nc = nc1;
      nc += nc2;

      //calling multicast
      nc(5);
      Console.WriteLine("Value of Num: {0}", getNum());
      Console.ReadKey();
    }
  }
}
```

- When the above code is compiled and executed, it produces the following result:

```
Value of Num: 75
```

- Delegate objects can be composed using the "+" operator.
- A composed delegate calls the two delegates it was composed from.
- Only delegates of the same type can be composed.
- The "-" operator can be used to remove a component delegate from a composed delegate.
- Using this useful property of delegates you can create an invocation list of methods that will be called when a delegate is invoked.

- This is called multicasting of a delegate.

Use of Delegate

- The following example demonstrates the use of delegate.

```
using System;
using System.IO;
namespace DelegateAppl
{
  class PrintString
  {
    static FileStream fs;
    static StreamWriter sw;
     // delegate declaration
    public delegate void printString(string s);
    // this method prints to the console
    public static void WriteToScreen(string str)
    {         Console.WriteLine("The String is: {0}", str);      }
      //this method prints to a file
    public static void WriteToFile(string s)
    {
      fs = new FileStream("c:\\message.txt",
      FileMode.Append, FileAccess.Write);
      sw = new StreamWriter(fs);
      sw.WriteLine(s);
      sw.Flush();
      sw.Close();
      fs.Close();
    }
     // this method takes the delegate as parameter and uses it to
    // call the methods as required
    public static void sendString(printString ps)
    {
      ps("Hello World");
    }
    static void Main(string[] args)
    {
      printString ps1 = new printString(WriteToScreen);
      printString ps2 = new printString(WriteToFile);
      sendString(ps1);
      sendString(ps2);
      Console.ReadKey();
    }   }  }
```

- The delegate printString can be used to reference methods that take a string as input and return nothing.
- When the above code is compiled and executed, it produces the following result:

```
The String is: Hello World
```

Events

- An event is a delegate with special features in the areas of type membership, limitations on invocation, and assignment.
- By having events as first-class type members, along with fields, methods, and properties, C# becomes a more component-oriented language.
- This means that other objects can listen for changes in your object and receive notifications via event invocations.

Using Delegates with Events

- The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class.
- The class containing the event is used to publish the event.
- This is called the publisher class.
- Some other class that accepts this event is called the subscriber class.
- Events use the publisher-subscriber model.
- A publisher is an object that contains the definition of the event and the delegate.
- The event-delegate association is also defined in this object.
- A publisher class object invokes the event and it is notified to other objects.
- Subscriber: A subscriber is an object that accepts the event and provides an event handler.

```
using System;
namespace SimpleEvent
{
  using System;

  public class EventTest
  {
    private int value;
    public delegate void NumManipulationHandler();
    public event NumManipulationHandler ChangeNum;
    protected virtual void OnNumChanged()
    {
      if (ChangeNum != null)
      {
        ChangeNum();
      }
      else
      {
        Console.WriteLine("Event fired!");
      }
    }

    public EventTest(int n )
    {
      SetValue(n);
    }
     public void SetValue(int n)
    {
      if (value != n)
      {
        value = n;
        OnNumChanged();
      }
    }
  }

    public class MainClass
    {
      public static void Main()
      {
        EventTest e = new EventTest(5);
        e.SetValue(7);
        e.SetValue(11);
        Console.ReadKey();
      }
    }
}
```

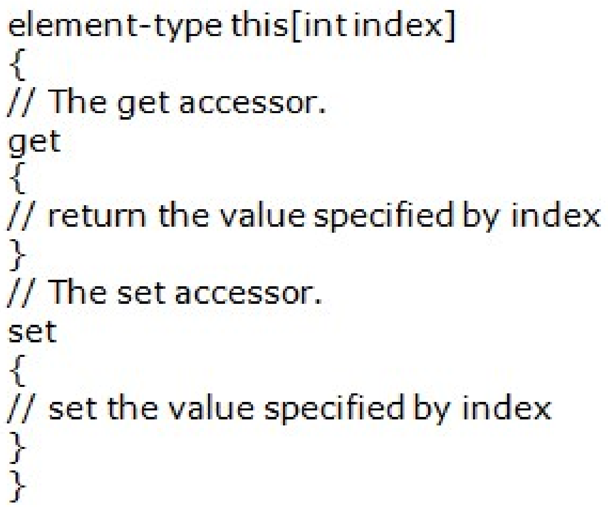
- When the above code is compiled and executed, it produces the following result:

```
Event Fired!
Event Fired!
Event Fired!
```

- The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Indexes

- Declaration of behavior of an indexer is to some extent similar to a property.

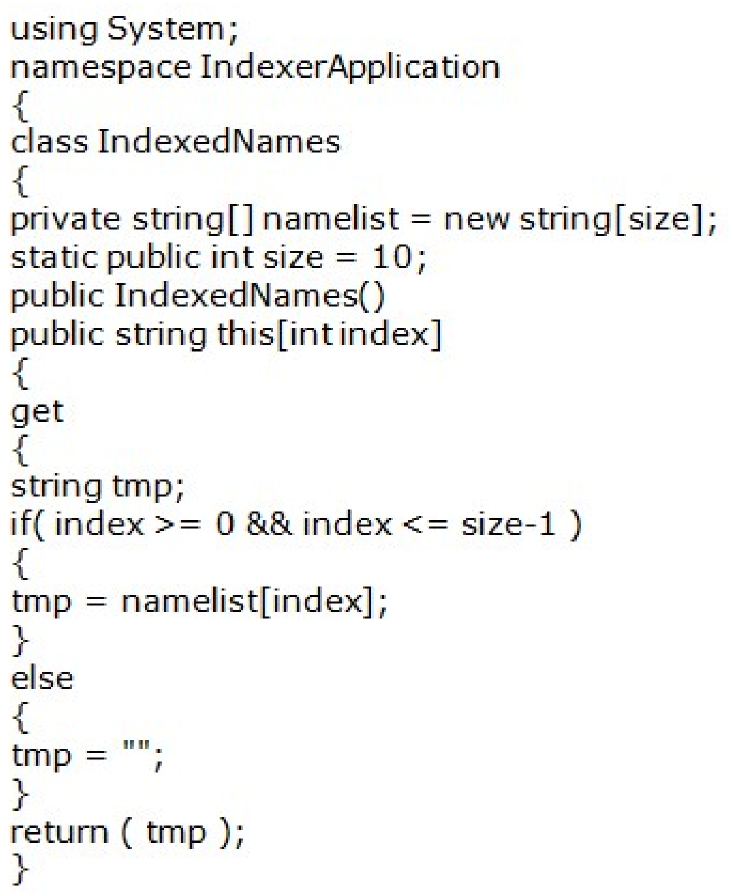


```
element-type this[int index]
{
// The get accessor.
get
{
// return the value specified by index
}
// The set accessor.
set
{
// set the value specified by index
}
}
```

- Like properties, you use get and set accessors for defining an indexer.
- However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instance.
- In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part.
- An indexer allows an object to be indexed like an array.
- Define an indexer for a class, this class behaves like a virtual array.
- You can then access the instance of this class using the array access operator ([ ]).

Use of Indexers

- Defining a property involves providing a property name.
- Indexers are not defined with names, but with the this keyword, which refers to the object instance.
- The following example demonstrates the concept.



```
using System;
namespace IndexerApplication
{
class IndexedNames
{
private string[] namelist = new string[size];
static public int size = 10;
public IndexedNames()
public string this[int index]
{
get
{
string tmp;
if( index >= 0 && index <= size-1 )
{
tmp = namelist[index];
}
else
{
tmp = "";
}
return ( tmp );
}
```

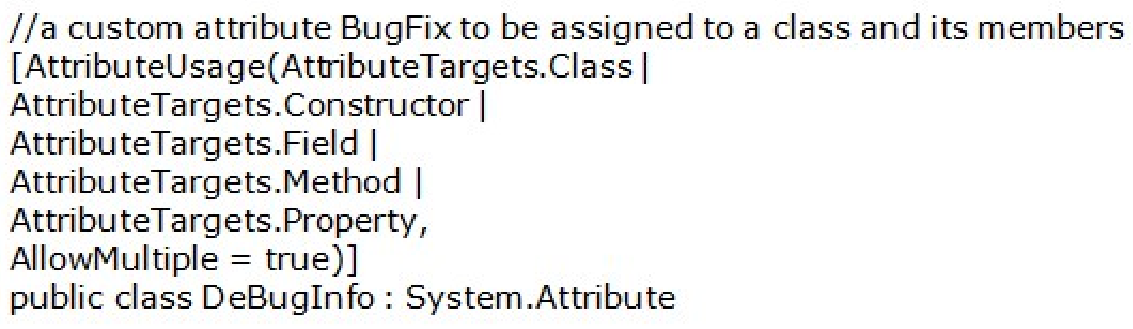
Overloaded Indexers

- Indexers can be overloaded.
- Indexers can also be declared with multiple parameters and each parameter may be a different type.
- It is not necessary that the indexes have to be integers.
- C# allows indexes to be of other types, for example, a string.

Attributes

- An attribute is a declarative tag that is used to convey information to runtime about the behaviors of various elements like classes, methods, structures, enumerators, assemblies etc., in your program.
- You can add declarative information to a program by using an attribute.
- A declarative tag is depicted by square ([ ]) brackets placed above the element it is used for.
- Attributes are used for adding metadata, such as compiler instruction and other information such as comments, description, methods and classes to a program.
- The .Net Framework provides two types of attributes: the pre-defined attributes and custom built attributes.
- In C# under different classifications the attributes are specified and some predefined attributes are also available in C#
  - AttributeUsage,
  - Conditional,
  - Obsolete.
- To create an custom attribute few steps have to follow like
  - Declaring a custom attribute,
  - Constructing the custom attribute,
  - Apply the custom attribute on a target program element,
  - Accessing Attributes Through Reflection.
- Syntax for specifying an attribute is as follows:
  - [attribute(positional_parameters, name_parameter = value, ...)] Element.
- Name of the attribute and its values are specified within the square brackets, before the element to which the attribute is applied.
- Positional parameters specify the essential information and the name parameters specify the optional information.
- Syntax for specifying this attribute is as follows:
  - [Conditional (conditionalSymbol)]
- The pre-defined attribute AttributeUsage describes how a custom attribute class can be used.
- It specifies the types of items to which the attribute can be applied.

- This predefined attribute marks a conditional method whose execution depends on a specified preprocessing identifier.
- It causes conditional compilation of method calls, depending on the specified value such as Debug or Trace.
- For example, it displays the values of the variables while debugging a code.
- Syntax for specifying this attribute is as follows:
  - [Obsolete(message)]
    [Obsolete(message, iserror)]
- This predefined attribute marks a program entity that should not be used.
- It enables you to inform the compiler to discard a particular target element.
- For example, when a new method is being used in a class, but you still want to retain the old method in the class, you may mark it as obsolete by displaying a message the new method should be used, instead of the old method.
- The .Net Framework allows creation of custom attributes that can be used to store declarative information and can be retrieved at run-time.
- This information can be related to any target element depending upon the design criteria and application need.
- Creating and using custom attributes involve four steps:
  - Declaring a custom attribute.
  - Constructing the custom attribute.
  - Apply the custom attribute on a target program element.
  - Accessing Attributes Through Reflection.
- The Last step involves writing a simple program to read through the metadata to find various notations.
- Metadata is data about data or information used for describing other data.
- This program should use reflections for accessing tributes at runtime.



```
//a custom attribute BugFix to be assigned to a class and its members
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]
public class DeBugInfo : System.Attribute
```

UNIT - 3
C# Using Libraries

Namespace-Systems

- Namespaces are C# program elements designed to help you organize your programs.
- They also provide assistance in avoiding name clashes between two sets of code.
- Implementing Namespaces in your own code is a good habit because it is likely to save you from problems later when you want to reuse some of your code.
- For example, if you created a class named Console, you would need to put it in your own namespace to ensure that there wasn't any confusion about when the System.Console class should be used or when your class should be used.

Functionality provided by System namespace

- C# library system provides a list of different functionalities for name spaces and are listed here
  - Commonly-used value,
  - Mathematics,
  - Remote and local program invocation,
  - Application environment management,
  - Reference data types,
  - Events and event handlers,
  - Interfaces Attributes Processing exceptions,
  - Data type conversion,
  - Method parameter manipulation.

Classes provide by System namespace

- Along with different functionalities namespaces in C# provides a list of classes and listed here
  - AccessViolationException,
  - Array,
  - ArgumentNullException,
  - AttributeUsageAttribute,
  - Buffer,
  - Console,
  - Convert,

- Delegate,
- Exception,
- InvalidCastException.

Interfaces provided by System namespace

- Few interface provided by the namespace system in C# are listed here
  - Public interface ICloneable.
  - Public interface IComparable.
  - Public interface IComparable < T >.
  - Public interface IConvertible.
  - Public interface ICustomFormatter.
  - Public interface IDisposable.
  - Public interface IEquatable < T >.
  - Public interface IFormatProvider.

Namespaces

- A namespace is designed for providing a way to keep one set of names separate from another.
- The class names declared in one namespace will not conflict with the same class names declared in another.
  - Defining a namespace,
  - The Using keyword,
  - Nested Namespaces.

Defining a Namespace

- A namespace definition begins with the keyword namespace followed by the namespace name as follows:

```
namespace namespace_name
{
// code declarations
}
```

- To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:

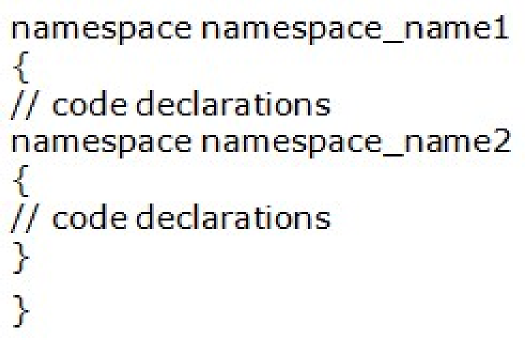
  namespace_name.item_name;

➢ The using Keyword

- using System;
  namespace first_space
- The using keyword states that the program is using the names in the given namespace.
- For example, we are using the System namespace in our programs.
- The class Console is defined there.
- We just write:
  - ➢ Console.WriteLine ("Hello there");
- We could have written the fully qualified name as:
  - ➢ System.Console.WriteLine("Hello there");
- With the using namespace directive.
- This directive tells the compiler that the subsequent code is making use of names in the specified namespace.

Nested Namespaces

- Namespaces can be nested where you can define one namespace inside another namespace as follows:



```
namespace namespace_name1
{
// code declarations
namespace namespace_name2
{
// code declarations
}
}
```

- You can access members of nested namespace by using the dot (.) operator.
- Namespaces can hold other types as follows:
  - ➢ Classes,
  - ➢ Structures,
  - ➢ Interfaces,
  - ➢ Enumerations and
  - ➢ Delegates.

Input Output

- The input & output in C# is based on streams.
  - A stream is an abstraction of a sequence of bytes, such as a file, an input/output device, an inter-process communication pipe, or a TCP/IP socket.
  - Streams transfer data from one point to another point.
  - Streams are also capable of manipulating the data; for example they can compress or encrypt the data.
  - In the .NET Framework, the System.IO namespaces contain types that enable reading and writing on data streams and files.

C# I/O Classes

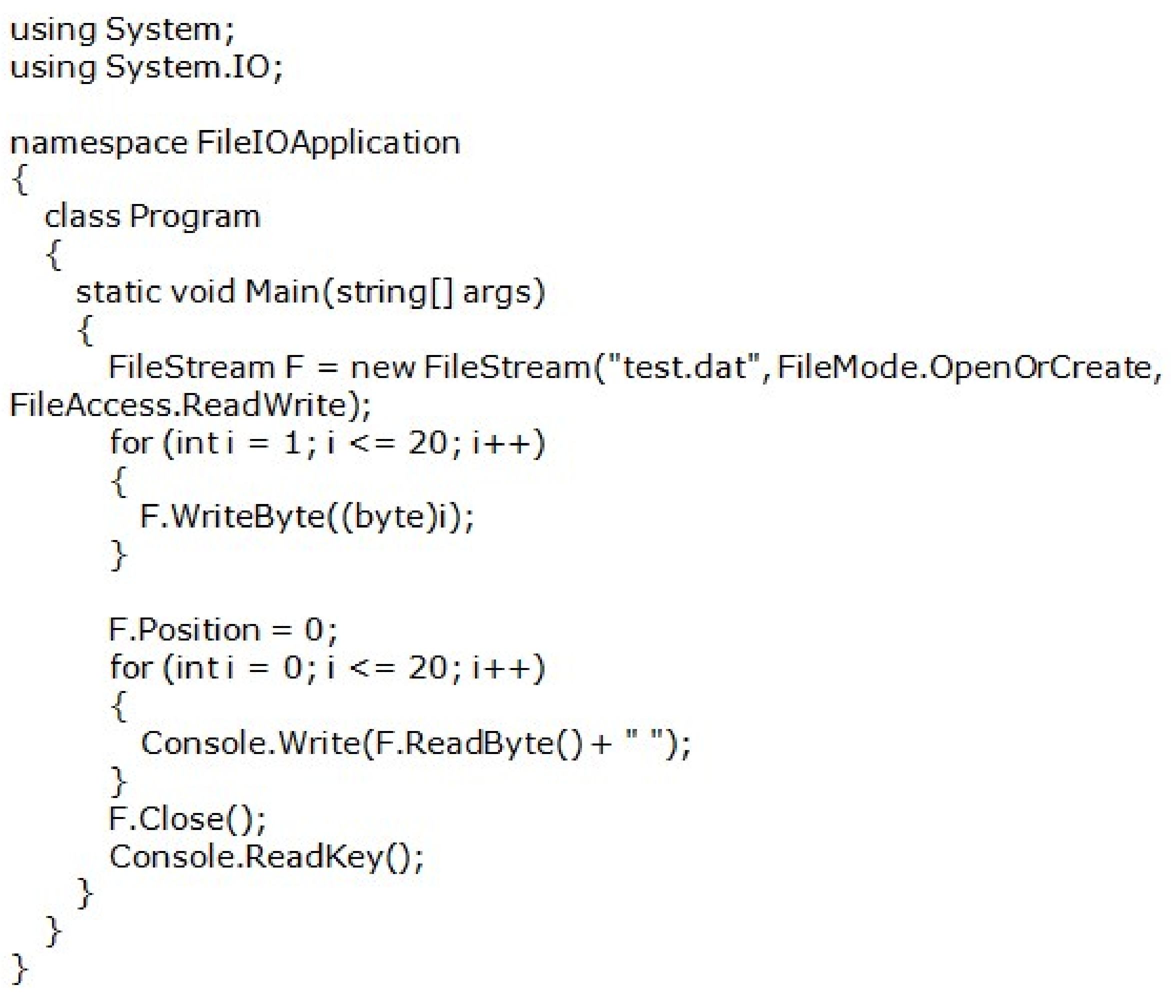
- The System.IO namespace has various class that are used for performing various operation with files, like creating and deleting files, reading from or writing to a file, closing a file etc.
- The following table shows some commonly used non-abstract classes in the System.IO namespace:

I/O Class	Description
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access to streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.

StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

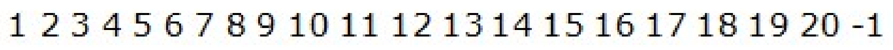
The FileStream Class

- The FileStream class in the System.IO namespace helps in reading from, writing to and closing files.
- This class derives from the abstract class Stream.
- You need to create a FileStream object to create a new file or open an existing file.
- The syntax for creating a FileStream object is as follows:
  - FileStream < object_name > = new FileStream(< file_name >, < FileMode Enumerator >, < FileAccess Enumerator >, < FileShare Enumerator >);
- The following program demonstrates use of the FileStream class:



```
using System;
using System.IO;

namespace FileIOApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         FileStream F = new FileStream("test.dat", FileMode.OpenOrCreate,
FileAccess.ReadWrite);
         for (int i = 1; i <= 20; i++)
         {
            F.WriteByte((byte)i);
         }

         F.Position = 0;
         for (int i = 0; i <= 20; i++)
         {
            Console.Write(F.ReadByte() + " ");
         }
         F.Close();
         Console.ReadKey();
      }
   }
}
```

- When the above code is compiled and executed, it produces the following result:



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

Reading from and Writing into Text files

- The StreamReader and StreamWriter classes are used for reading from and writing data to text files.
- These classes inherit from the abstract base class Stream, which supports reading and writing bytes into a file stream.

The StreamReader Class

- The StreamReader class also inherits from the abstract base class TextReader that represents a reader for reading series of characters.
- The following table describes some of the commonly used methods of the StreamReader class:

public override void Close() It closes the StreamReader object and the underlying stream, and releases any system resources associated with the reader.
public override int Peek() Returns the next available character but does not consume it.
public override int Read() Reads the next character from the input stream and advances the character position by one character.

The StreamWriter Class

- When the above code is compiled and executed, it produces the following result:

```
Zara Ali
Nuha Ali
```

- The following table describes some of the commonly used methods of the Stream Writer class:

1	public override void Close() Closes the current StreamWriter object and the underlying stream.
2	public override void Flush() Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
3	public virtual void Write(bool value) Writes the text representation of a Boolean value to the text string or stream. (Inherited from TextWriter.)
4	public override void Write(char value) Writes a character to the stream.

5	public virtual void Write(decimal value) Writes the text representation of a decimal value to the text string or stream.
6	public virtual void Write(double value) Writes the text representation of an 8-byte floating-point value to the text string or stream.
7	public virtual void Write(int value) Writes the text representation of a 4-byte signed integer to the text string or stream.
8	public override void Write(string value) Writes a string to the stream.
9	public virtual void WriteLine() Writes a line terminator to the text string or stream.

Multithreading

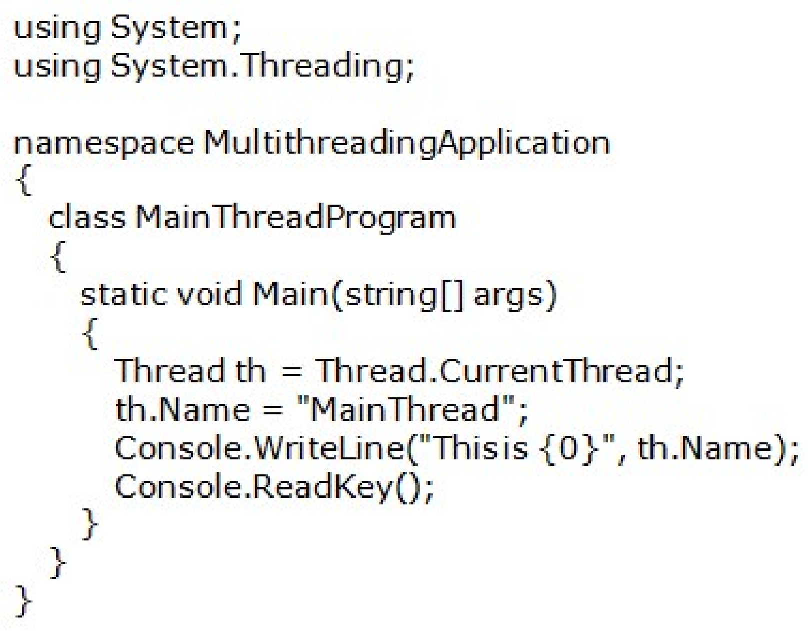
- A thread is defined as the execution path of a program.
- Each thread defines a unique flow of control.
- If your application involves complicated and time consuming operations then it is often helpful to set different execution paths or threads, with each thread performing a particular job.
- Threads are lightweight processes.
- One common example of use of thread is implementation of concurrent programming by modern operating systems.
- Use of threads saves wastage of CPU cycle and increase efficiency of an application.
- However, this way the application can perform one job at a time.
- To make it execute more than one task at a time, it could be divided into smaller threads.

Thread Life Cycle

- The life cycle of a thread starts when an object of the System.Threading.
- Thread class is created and ends when the thread is terminated or completes execution.
- The various states in the life cycle of a thread:
- The Unstated State:
    - It is the situation when the instance of the thread is created but the Start method has not been called.
- The Ready State:
    - It is the situation when the thread is ready to run and waiting CPU cycle.
- The Not Runnable State:
    - A thread is not runnable, when:
        - Sleep method has been called,
        - Wait method has been called,
        - Blocked by I/O operations.
- The Dead State:
    - It is the situation when the thread has completed execution or has been aborted.

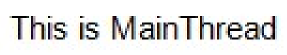
The Main Thread

- When a C# program starts execution, the main thread is automatically created.
- The threads created using the Thread class are called the child threads of the main thread.
- You can access a thread using the CurrentThread property of the Thread class.



```
using System;
using System.Threading;

namespace MultithreadingApplication
{
   class MainThreadProgram
   {
      static void Main(string[] args)
      {
         Thread th = Thread.CurrentThread;
         th.Name = "MainThread";
         Console.WriteLine("This is {0}", th.Name);
         Console.ReadKey();
      }
   }
}
```

- When the above code is compiled and executed, it produces the following result:

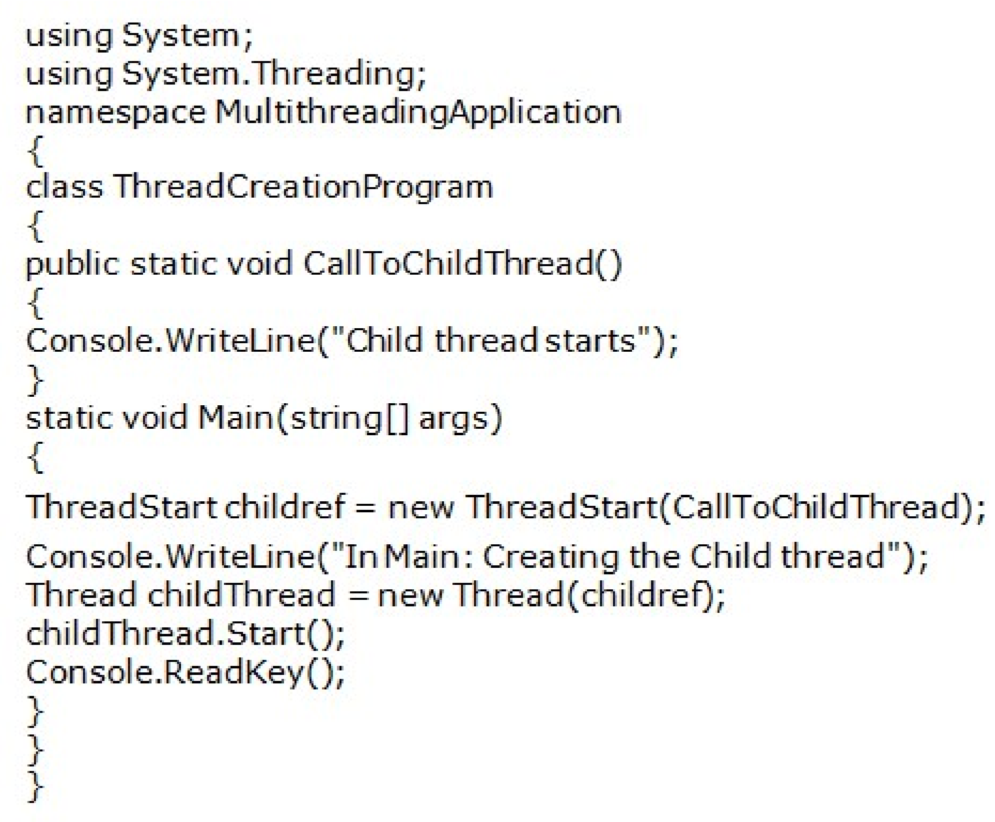


```
This is MainThread
```

- In C#, the System.Threading. Thread class is used for working with threads.
- It allows creating and accessing individual threads in a multithreaded application.
- The first thread to be executed in a process is called the main thread.

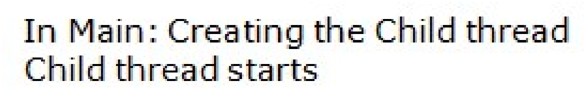
Creating Threads

- The following program demonstrates the concept:



```
using System;
using System.Threading;
namespace MultithreadingApplication
{
class ThreadCreationProgram
{
public static void CallToChildThread()
{
Console.WriteLine("Child thread starts");
}
static void Main(string[] args)
{
ThreadStart childref = new ThreadStart(CallToChildThread);
Console.WriteLine("In Main: Creating the Child thread");
Thread childThread = new Thread(childref);
childThread.Start();
Console.ReadKey();
}
}
}
```

- When the code is compiled and executed, it produces the following result:

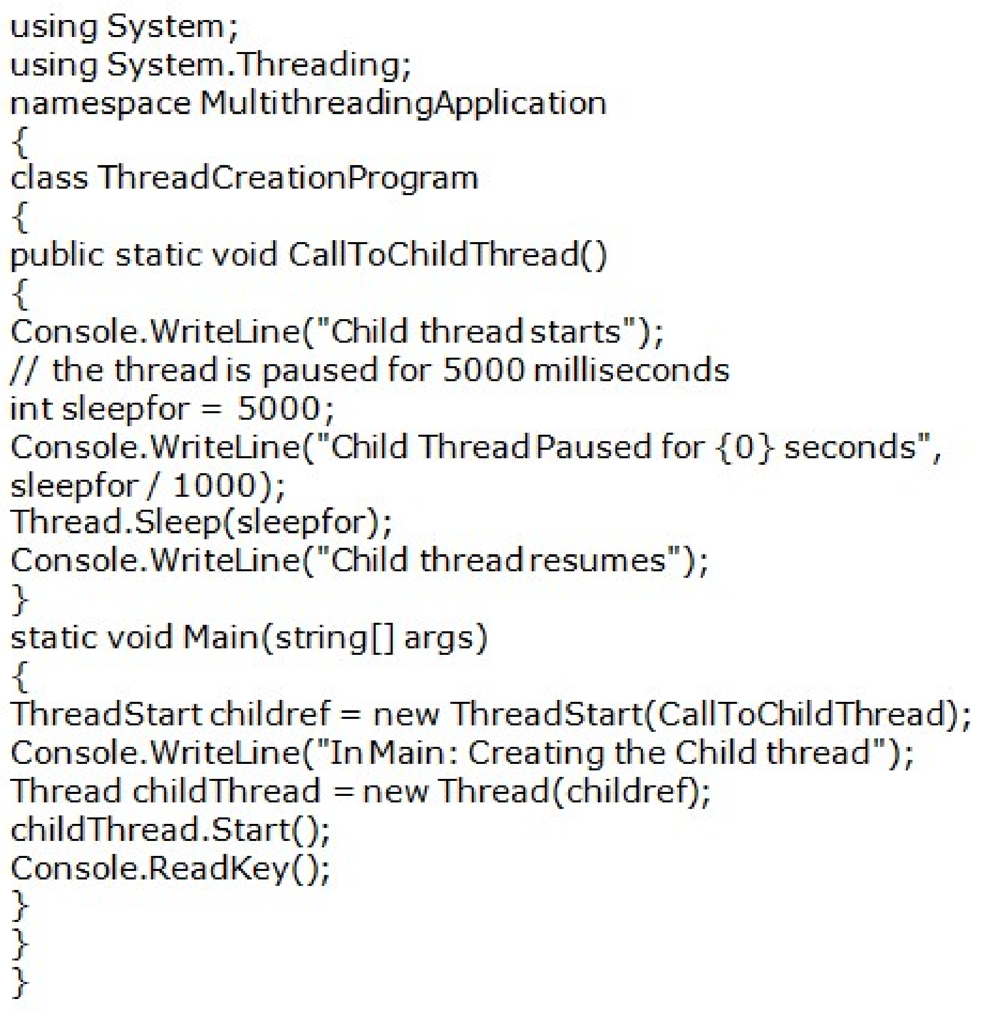


```
In Main: Creating the Child thread
Child thread starts
```

- Threads are created by extending the Thread class.
- The extended Thread class then calls the Start()method to begin the child thread execution.

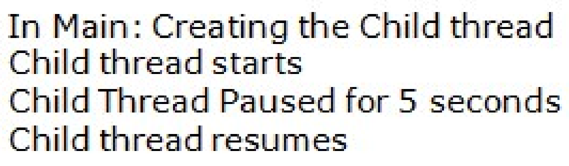
Managing Threads

- The Thread class provides various methods for managing threads.
- The following example demonstrates the use of the sleep() method for making a thread pause for a specific period of time.



```
using System;
using System.Threading;
namespace MultithreadingApplication
{
class ThreadCreationProgram
{
public static void CallToChildThread()
{
Console.WriteLine("Child thread starts");
// the thread is paused for 5000 milliseconds
int sleepfor = 5000;
Console.WriteLine("Child Thread Paused for {0} seconds",
sleepfor / 1000);
Thread.Sleep(sleepfor);
Console.WriteLine("Child thread resumes");
}
static void Main(string[] args)
{
ThreadStart childref = new ThreadStart(CallToChildThread);
Console.WriteLine("In Main: Creating the Child thread");
Thread childThread = new Thread(childref);
childThread.Start();
Console.ReadKey();
}
}
}
```

- When the code is compiled and executed, it produces the following result:



```
In Main: Creating the Child thread
Child thread starts
Child Thread Paused for 5 seconds
Child thread resumes
```

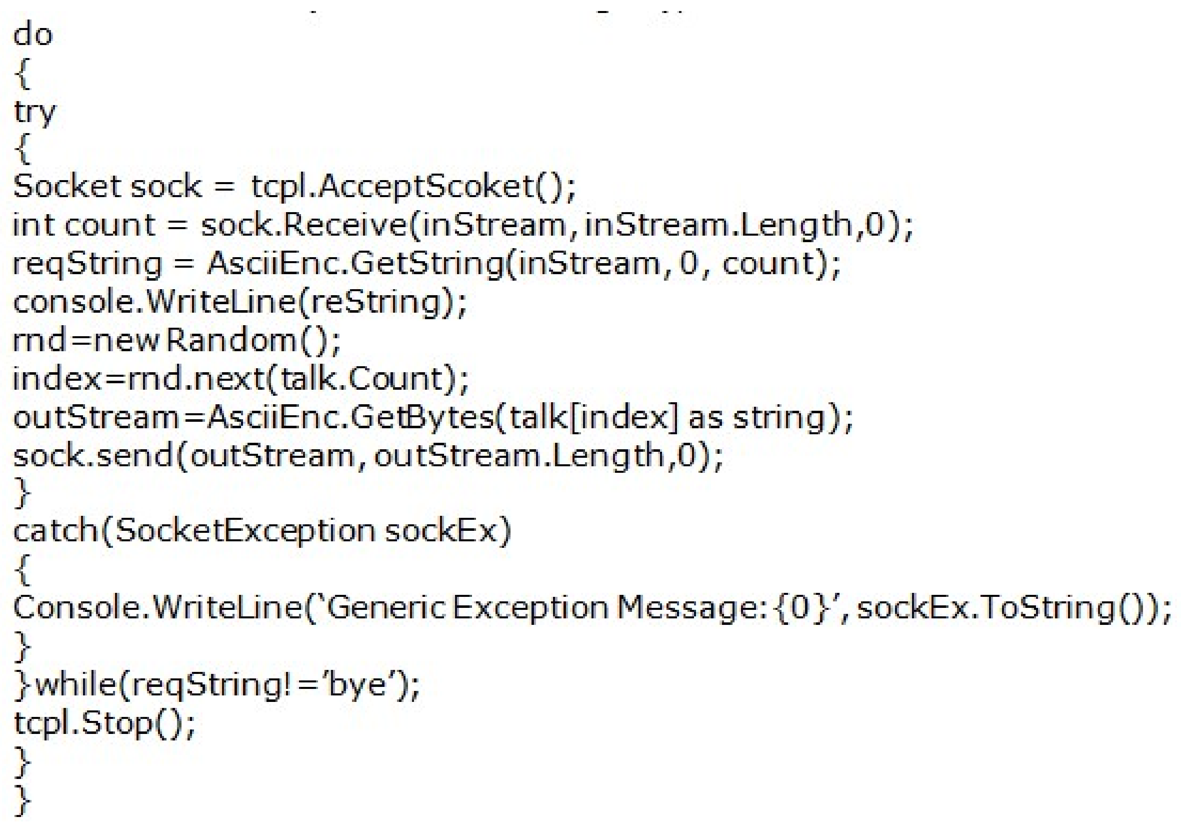
Destroying Threads

- The Abort() method is used for destroying threads.
- The runtime aborts the thread by throwing a ThreadAbortException.
- Exception cannot be caught, the control is sent to the finally block, if any.

Networking & Sockets

- C# language has access to an entire suite of networking libraries.
- Some of the capabilities range from low-level socket connections to wrapped HTTP classes.
  - Implementing Sockets,
  - A Socket Server,
  - A Socket Client,
  - Working with HTTP,
  - Performing FTP File Transfers,
  - Sending SMTP Mail.
- This server program instantiates a List < string >, talk, and initializes it with quote strings during constructor processing.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Text;
class MoneyServer
{
static void Main()
{
const int StreamSize = 256;
const int Port = 2010;
var ipAddr = new IPAddress(
new byte[] { 127,0, 0, 1 });
var talk = new List<string>
{
“A penny saved is too small, make it a buck.”,
“Keep your wooden nickel. It’ll be worth something someday.”,
“It’s your dime, but you’re better off dialing 10-10-XXX.”
};
var mSvr = new MoneyServer();
var AsciiEnc = new ASCIIEncoding();
var inStream = new byte[StreamSize];
var outStream = new byte[StreamSize];
Random rnd = null;
string reqString = string.Empty;
int index = 0;
var tcpl = new TcpListener(ipAddr, Port);
tcpl.Start();
Console.WriteLine(“Server is Running...”);
```



```
do
{
try
{
Socket sock = tcpl.AcceptScoket();
int count = sock.Receive(inStream, inStream.Length,0);
reqString = AsciiEnc.GetString(inStream, 0, count);
console.WriteLine(reString);
rnd=new Random();
index=rnd.next(talk.Count);
outStream=AsciiEnc.GetBytes(talk[index] as string);
sock.send(outStream, outStream.Length,0);
}
catch(SocketException sockEx)
{
Console.WriteLine('Generic Exception Message:{0}', sockEx.ToString());
}
}while(reqString!='bye');
tcpl.Stop();
}
}
```

- The real action for this program starts in the Main method.
- Socket operations are encapsulated in the TCP classes.
- This program uses the TcpListener class to create a socket connection on the local host.
- The example accepts a single parameter, indicating the port number.
- After the TcpListener class is instantiated, it must be started with the Start method.
- The client program uses sockets to request information from a server.
- It makes a socket connection, sends a request, and receives a reply.

Working with HTTP

- HTTP is what enables communication across the Web.

```
using System;
using System.IO;
using System.Net;
using System.Text;
namespace HttpRequestDemo
{
class Program
{
static void Main()
{
const int BufferSize = 2048;
try
{
var AsciiEnc = new ASCIIEncoding();
var buf = new byte[BufferSize];
HttpWebRequest httpReq =
WebRequest.Create(
"http://www.csharp-station.com")
as HttpWebRequest;
HttpWebResponse httpResp =
httpReq.GetResponse() as HttpWebResponse;
Stream httpStream = httpResp.GetResponseStream();
int count = httpStream.Read(buf, 0, buf.Length);
Console.WriteLine(
AsciiEnc.GetString(buf, 0, count));
}
catch (Exception e)
{
Console.WriteLine("Generic Exception: {0}",
e.Message);
}
}
}
}
```

- Knowing how to request a web page can be useful for caching or screen scraping, which is a good application of regular expressions.
- The Create method accepts a string representation of an URL.
- Here's the statement.

```
HttpWebRequest httpReq =
WebRequest.Create(
"http://www.csharp-station.com")
as HttpWebRequest;
```

- After an HttpWebRequest object is created, it can be used to obtain an HttpWebResponse object.
- This happens by invoking its GetResponse method, which returns a WebResponse object.

  The WebResponse object is an abstract base class of the HttpWebResponse class, and a cast

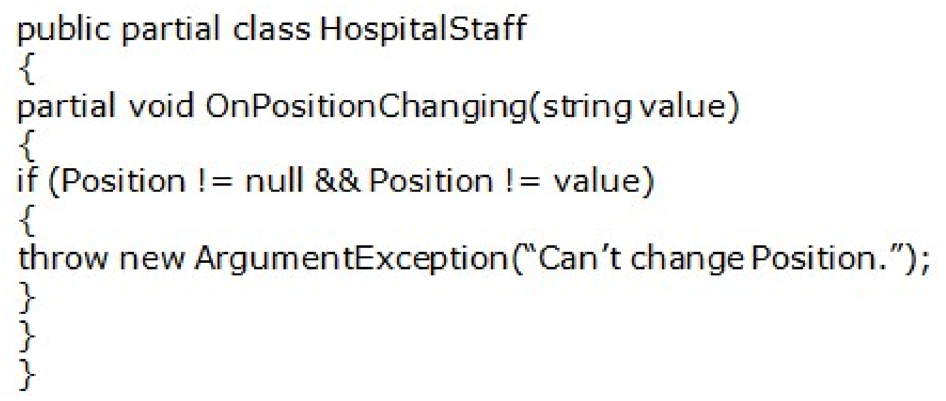
- operation is necessary for conversion.

Performing FTP File Transfers

- Another Internet protocol for moving files around is FTP, which is supported by the .NET FCL.
- The following sections show you how to get and put files with FTP.
- Putting Files on an FTP Server
  - To upload files to an FTP server, you need to create an FtpRequest object, get a stream reference to where you want to put the file, and write the file to the stream.
- Getting Files from an FTP Server
  - To get a file from an FTP server, you need to create the FtpWebRequest, open the stream to the file, read the bytes, open the stream to the file you need to create, and write the bytes to the new file.
- FTP servers require some type of credentials, even if itprs anonymous access with a username of “anonymous” and a password that consists of your email address.
- The rest of the code uses using statements to open the response stream, read bytes from the FTP server, open a file stream to the new file, and then write those bytes to the new file.
- Remember that the using statements are essential because they close the file streams for you.

Data Handling Logic

- Customizing Insert, Update, and Delete with Partial Methods.
- Customizing Property Changes with Partial Methods.
- Extend the logic of your system by providing an implementation for any of these partial methods.
- The following example shows how to detect a property change for the Position property of the HospitalStaff entity.



```
public partial class HospitalStaff
{
partial void OnPositionChanging(string value)
{
if (Position != null && Position != value)
{
throw new ArgumentException("Can't change Position.");
}
}
}
```

Standard Query Operators

- The following sections provide a quick overview of available query operators some with C# aliases and others without.
- For reference and provide a simple example in each category so that you can see how they are used.
  - Sorting Operators
  - Set Operators
  - Filtering Operators
  - Quantifier Operators
  - Projection Operators
  - Partitioning Operators
  - Join Operators
  - Grouping Operators
  - Generation Operators
  - Equality Operators

Sorting Operators

Sorting operations affect the ordering of the results.

- - OrderBy
    - Sets sort order.
  - OrderByDescending
    - Sorts in descending order.
  - ThenBy
    - Subsequent sorts.
  - ThenByDescending
    - Subsequents sets in descending order.
  - Reverse
    - Reverses results.

Set Operators

- Set operators are for set-based operations.
- Distinct returns unique value.
- Except Like a SQL left join returns results from one set that aren′t in another.
- Intersect Returns common elements from each set.
- Union Returns all objects from both sets.

Filtering Operators

- Filtering operators return a subset of a collection
  - ofType Objects of the specified type.
  - where Objects meeting specified predicate.
- The C# where clause aliases the where operator, but there isn′t an alias for the OfType operator.
- You could give where a predicate, such as where staff.GetType == typeof(TempStaffMember). TempStaffMember is a class that derives from StaffMember.

Projection Operators

- Projection operators alter the shape of query results.
- If a projection operator alters the shape, a projection, the results may contain more or fewer fields or perform some manipulation of an existing field for the result set.
  - Select

    - Defines the fields/properties to return.

- SelectMany
    - Flattens a multilevel hierarchy to access results.

Partitioning Operators

- Partitioning operators to specify which group of records you want from a collection.
    - Skip
        - Skips over a specified number of records.
    - SkipWhile
        - Skips over records while the specified condition is true.
    - Take
        - Takes the specified number of records.
    - TakeWhile
        - Takes records while the specified condition is true.
- A common scenario for the partitioning operators is to facilitate paging.
- In many cases, the number of records in a record set is too large to hold in memory, so you want to bring in only the minimum number necessary.

Join Operators

- Join operators enable you to combine sets of data.
    - Join
        - Returns set of records where keys in two sets are equal.
    - GroupJoin
        - Builds a hierarchy of objects based on child record set keys that match a parent key.
- Here's an example of a GroupJoin so that you can see the operator syntax:

Grouping Operators

- A grouping operator allows you to create groups of data.
    - GroupBy
        - Returns set of records where keys in two sets are equal.
    - ToLookUp

- Creates an ILookup dictionary of lists with specified key.

Equality Operators

- There is only one equality operator, which tells you whether two collections are equal.
  - SequenceEqual
    - Returns true if two collections are equal.

Element Operators

- To return just one record from a collection, you can use an element operator.
  - ElementAt
    - Returns the element at a specified position.
  - ElementAtOrDefault
    - Same as ElementAt, but returns the default value of type if not found.
  - First
    - Returns the first element in results.
  - FirstOrDefault
    - Same as First, but returns the default value of type if not found.
  - Last
    - Returns the last element in results.
  - LastOrDefault
    - Same as Last, but returns the default value of type if not found.
  - Single
    - Returns only element in results.
  - SingleOrDefault
    - Same as Single, but returns the default value of type if not found.
- Each element operator has a version that returns a default value.
- The difference is that if you invoke an element operator that isn't a default version and the collection doesn't have that value, you'll get an InvalidOperationException exception.

Conversion Operators

- A conversion operator allows you to transform results from one type or collection to another.

- AsEnumerable
  - Converts to IEnumerable< T >
- AsQueryable
  - Converts to IQueryable< T >
- Cast
  - Converts weakly typed collection (for example, ArrayList) to IEnumerable< T >
- OfType
  - Filters collection based on type.
- ToArray
  - Converts to an array.
- ToDictionary
  - Converts to an Dictionary.
- ToList
  - Converts to a list.
- ToLookup
  - Converts to a lookup.

Concatenation Operator

- There is one concatenation operator, and it allows you to concatenate one collection to another.
  - Concat
    - Concatenates one collection to another to form a single collection.

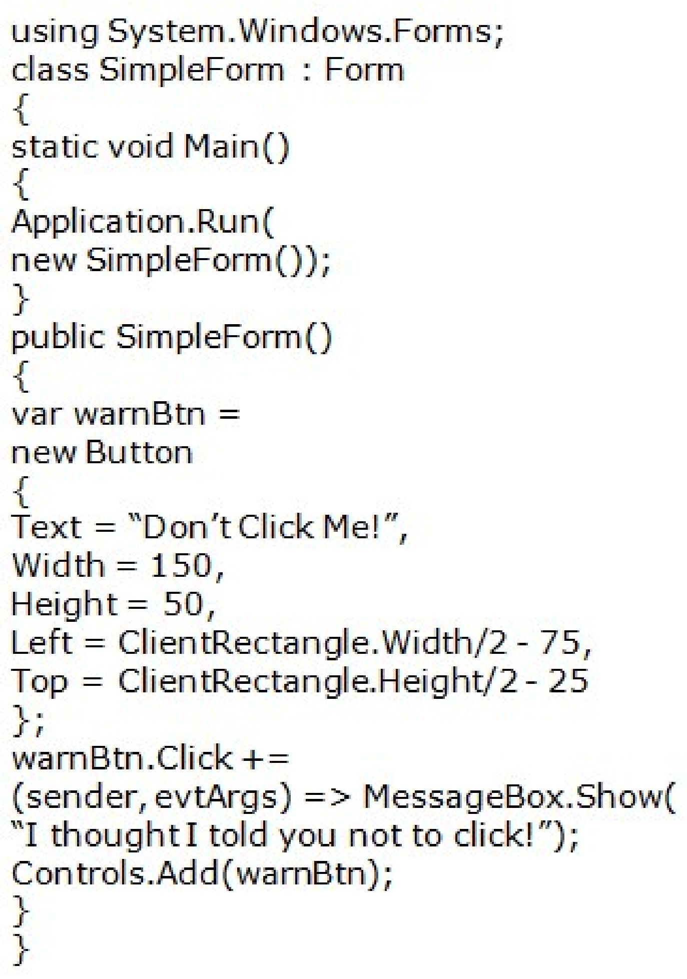
Aggregate Operators

- An aggregate operator will perform a computation on a group of values and provide a single result.
  - Aggregate
    - Creates a custom aggregation.
  - Average
    - Returns an average.
  - Count

  - Returns the number of items in a collection of size int.Max or less.
- LongCount
  - Returns the number of items in a large collection up to size long.MAX.
- Max
  - Returns the max item.
- Min
  - Returns the min item.
- Sum
  - Returns the sum of all item.

Windows Forms Fundamentals

- To get started, create a new Windows Forms application named WinFormFundamentals.
- You can do this in VS2008 by creating a new project and selecting Windows Forms Application, or right-click an existing Solution file and select Windows Forms Application.
- The results are that you have a new project with references to System.Windows.Forms.dll, which the library is holding the Windows Forms API.
- A form is a class, just like any other type you write C# code against.



```
using System.Windows.Forms;
class SimpleForm : Form
{
static void Main()
{
Application.Run(
new SimpleForm());
}
public SimpleForm()
{
var warnBtn =
new Button
{
Text = "Don't Click Me!",
Width = 150,
Height = 50,
Left = ClientRectangle.Width/2 - 75,
Top = ClientRectangle.Height/2 - 25
};
warnBtn.Click +=
(sender, evtArgs) => MessageBox.Show(
"I thought I told you not to click!");
Controls.Add(warnBtn);
}
}
```

- When the above code is compiled and executed, it produces the following result:
- All controls on the form are also classes, and you use them just like any other type in C# Windows Forms is a classic example of the effective implementation and use of delegates and events.

The .NET Framework Class Library - Windows Forms API

- You have the Windows Forms API that belongs to the .NET Framework Class Library that exposes events.
- Then you use the delegates that those events are based on to connect your code to the API to make your program work.

Support for Windows Forms

- The Visual Design Environment.
- Files in a Windows Forms Application.
- The Windows Forms Entry Point.
- User Code.
- Visual Designer–Generated Code.
- Modifying the Visual Designer.
- Using Windows Forms Controls.

The Visual Design Environment

- Visual Studio has a helpful visual design environment that helps you build user interfaces with ease which contains controls that you can drag and drop onto the design surface.
- The design surface is in the middle of the screen.
- You can see that it is a WYSIWYG environment, where you can resize the form and visually see the results of your work.
- Although not shown in this example, nonvisual controls appear below the form in the visual designer in an area called the component tray.
- You've see the Solution Explorer already when building console applications, and it is a necessary part of every application.
- The Properties window is context- sensitive.
- It will change, depending on which window or control that is selected on the design surface.
- In such cases, you must use the drop-down window at the top of the Properties window to select the control.

Files in a Windows Forms Application

- That there are a few different files that were created by the Windows Application Project Wizard.
  - Form1.cs
    - User code for working with the form.
    - You normally work in this file most of the time.
    - All your event handlers and code that call your business logic are added here.
  - Form1.Designer.cs

    Visual Designer–generated code.

- ❖ ❖ You normally don't ever need to open this file.
- ❖ If you need to write your own code to work with the form, it will typically be done through Form1.cs.

➤ Form1.resx

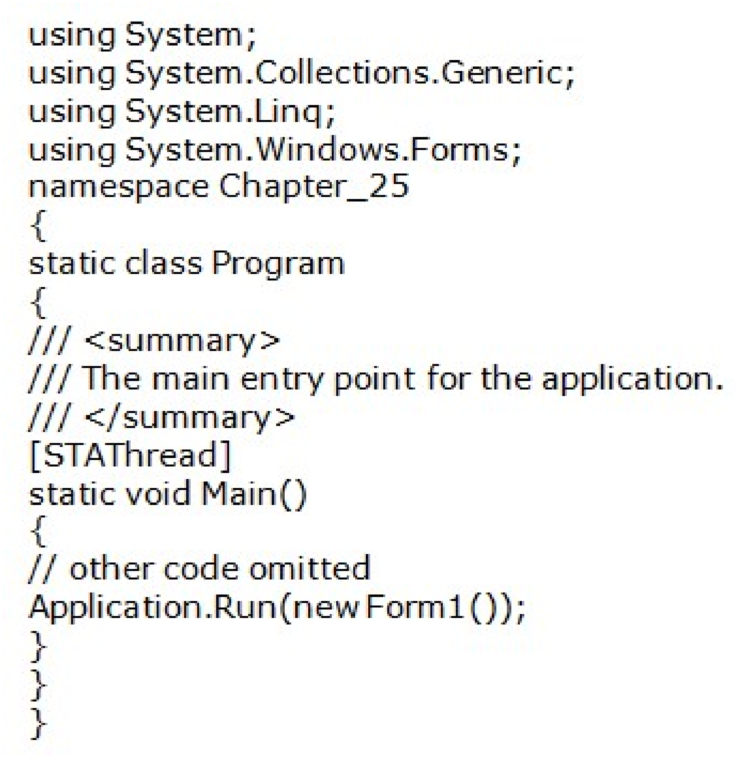
- ❖ Resource file for bitmaps, strings, icons, and so on.
- ❖ “Localizing and Globalization,” discusses resource files in greater depth.

➤ Program.cs

- ❖ Contains the Main method.

The Windows Forms Entry Point

- If you open the Program.cs file, you'll see a Main method.
- This is much like you've seen in console applications, whose entry point is also the Main method.
- The Application.Run method will start the program and not return until the application closes.

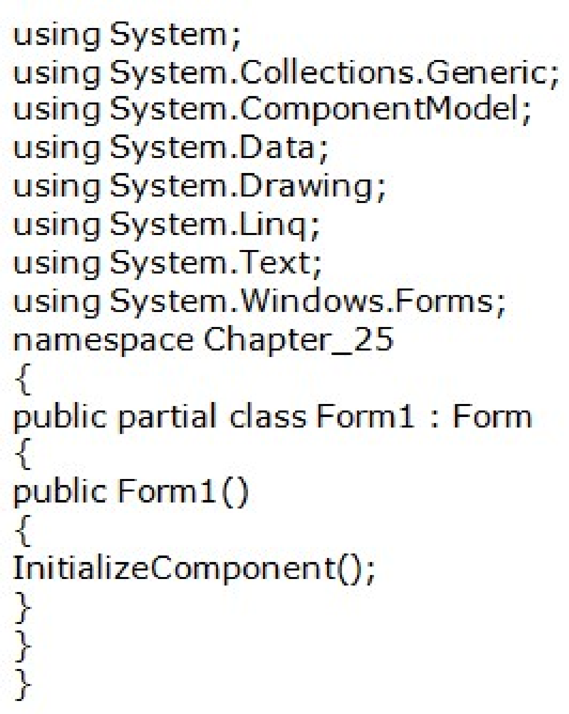


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace Chapter_25
{
static class Program
{
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
// other code omitted
Application.Run(newForm1());
}
}
}
```

User Code

- When adding specific behavior to a Windows Forms application, you'll use the Form1.cs class.
- The Windows application project creates for you.
- You can see this code by right-clicking either the form in the Visual Designer or Form1.cs in Solution Explorer and selecting View Code.
- But there are a couple significant observations to make at this point. First, Form1 is a partial

class.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Chapter_25
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
}
}
```

Visual Designer–Generated Code

- Form1.Designer.cs holds the matching partial to the Form1 partial class defined in Form1.cs.
- When this code is compiled, C# combines these two files into a single class definition.
- Considering that Form1 is a partial class and its constructor calls an InitializeComponent method the other Form1 partial class and the definition of the InitializeComponent method.

When Modifying the Visual Designer

- The files in a Windows Forms application, and the reason for the contents of each file.
- This will demonstrate to you how the Form1.cs and Form1.Designer.cs work together while you ′re building a UI.
- Drag and drop a Button control onto the design surface.
- If you want to delete anything that you accidentally drop on the design surface, select the control and press the Delete key or right-click the control and select Delete.
- Go to the Properties window and set the following properties. (You can find Height and Width under Size.)
- Set Text to Don′t Click Me!.
- Set Height to 50.
- Set Width to 150.
- Set (Name) to btnWarn.

Using Windows Forms Controls

- A control is a specialized window with specific features and a unique purpose.
- These are things like
  - Button,
  - Checkbox,
  - Checklist,
  - Combobox,
  - DataGridView,
  - DateTimeicker,
  - Form,
  - Groupbox,
  - Lable,
  - Linklable,
  - Listbox,
  - ListView etc.,
- Button control that can be clicked to perform some desired action.
- Checkbox Primarily used for displaying a binary state of an object.
- Clicking the check box causes it to toggle between a checked or unchecked state.
- Checklist List box with a column of check boxes.
- Combobox A drop-down list of choices that operates similar to the list box.
- The primary difference is that the combo box is more compact and efficient with screen real estate.

The Web Application Model

- When writing for the web, you need to understand where code resides and executes.
- An ASP.NET application is hosted on a web server, which is where C# executables reside, but it renders HTML to clients via browsers such as Internet Explorer (IE).

C# Web Application

- In all of this processing, your assemblies, which were built with C# stay on the server.
- When ASP.NET receives the request, it sends it to your code for processing.
- The great majority of the time, you don’t ever have to emit HTML manually because the object-oriented ASP.NET APIs take care of it for you.
- The most important point to remember is that your C# code runs on the server, not on the browser.
- Scalability is a property of software that defines how well it can handle an increased workload.
- Applications that reach a threshold and then crash or slow to a crawl aren’t very scalable.
- You want a scalable application that performs well at the projected peak capacity for your requirements.

To add the component

- In Solution Explorer, right-click the project name.
- On the shortcut menu, click Add, and then click Add Component.
- The Add New Item dialog box appears and the Component Class in the right pane will be selected by default.
- Accept the default name (Component1) and click Open.
- Unless you choose another name for the component, this creates a new file in your project named Component1.cs or Component1.vb depending on the application language.
- The Component Designer opens a design view on Component1.cs or Component1.vb.

Creating the Web Application Project Using Visual C#

- To create the Web Form.
- On the File menu, click New, and then click Project.
- The New Project dialog box appears.
- In the Project Type pane, click Visual Basic Projects or Visual C# Projects, and in the Templates pane, select ASP.NET Web Application.

- Name your application MyWebForm by changing the default name in the Location box (such as http://localhost/WebApplication1 to http://localhost/MyWebForm).
- Click OK.
- The application wizard will create the necessary project files, including the following files:
  - WebForm1.aspx
    - Contains the visual representation of the Web Form.
  - WebForm1.aspx.cs or WebForm1.aspx.vb
    - The code-behind file that contains the code for event handling and other programmatic tasks.
    - To see this file in Solution Explorer, click the Show All Files icon, and then expand the WebForm1.aspx node.
  - Web Form Files
  - Note If Solution Explorer is not open, on the View menu, click Solution Explorer.

State Management

- In C# web application development the global state management of the application is taken care of different methods.
- Global State with Application is used to hold information common to all of these application instances.
- This is where Application state can be used.
- Here's an example of how to use Application state.
  - Holding Updatable Information in Cache is to hold information in memory to avoid the overhead of creating or retrieving that information yourself.
  - Holding State for a Single Request features hold on to that state for an extended period of time.
  - Holding too much information in memory affects the scalability of your application.
  - Issuing Cookies is information with a max size of 4K that you can ask users to hold in their browsers.
  - Whenever the browser visits your site, it presents all cookies that your site gave to it.
  - User-Specific Information with Session State which holds information for a specific user.
  - When the user visits the page, ASP.NET issues a cookie with a session ID.
  - It then manages Session state for that user, based on the user's session ID.

- Understanding Page State in ViewState.
- Page Reuse with Master Pages and Custom Controls more work than necessary to duplicate this information on each page, especially with ASP.NET, which instead enables you to implement user controls and master pages.

Navigation

- In C# web application development to use a HyperLink control, but that is a pretty limited version of navigation.
- To add menus and other sophisticated controls, like breadcrumbs, to make a site more usable.
- ASP.NET enables you to do this with Menu,
  - TreeView, and SiteMapPath controls.
  - Defining Site Layout with Web.sitemap.
  - Navigation with the Menu Control.
  - Theming a Site.
  - Securing a Website.
  - Data Binding.

Theming a site

- With themes, you can design a set of skins and CSS styles that apply to the entire site.
- To add a theme,
  - Right-click the web project, select Add ASP.NET Folder, Theme, and name it Custom.
  - You can add multiple themes; they must have different names.
- Creating Skins.
- Add a Label control to a page.
- Right-click the Custom theme folder, select Add New Item, Skin File, name the file Custom.skin, and click the Add button.
  - < asp:Label ID="Label1" runat="server" Text="Label"
    BorderColor="Green"
    BorderWidth="2" Font-Italic="True" > < /asp:Label >
- Open Custom.Skin, copy the Label control from step 1, paste the Label control into the Custom.skin file.

Securing a Website

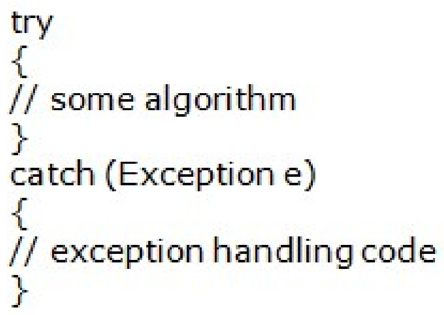
- IN c# web application Select WebSite, ASP.NET Configuration.
    - A website by selecting File, New, Project, ASP.NET Web Application,
    - ASP.NET Configuration in the Projects menu.
    - Click either the Security link or the Security tab.
    - Click Use the Security Setup Wizard. The wizard, which walks you through seven steps, will open.
    - The first screen is a welcome message. Click Next.
    - ASP.NET enables you to use Windows or ASP.NET security.
    - To set up a custom database, run aspnet_regsql.exe.
    - Check the Enable Roles for This Web Site check box.

Data Binding

- Data binding on a number of controls, including ListView, FormView, ListBox, DropDownList, and others.
- These controls has a DataSource property that takes an IEnumerable collection to bind to Drag and drop a ListView control onto the design surface of a web form.
- From the ListView Action list, select New Data Source in the Choose Data Source area.
- After a couple seconds, the Data Source Configuration Wizard will appear.
- You'll see several data sources, most binding directly to the data source.
- Select Object, which is the ObjectDataSource control, and click the Next button.
- Select HospitalManager and click the Next button.
- If you recall, HospitalManager is the business object we created in the preceding section.

Error Handling

- C# exception handling, consider the error-handling methods used in programming languages with no built-in exception-handling mechanism.



```
try
{
// some algorithm
}
catch (Exception e)
{
// exception handling code
}
```

- C# error-handling methods used in programming languages with no built-in exception-handling mechanism.
- Error Handler Syntax:
  - The Basic try/catch Block.
  - Finally Blocks.
  - Handling Exceptions.
  - Checked and unchecked Statements.

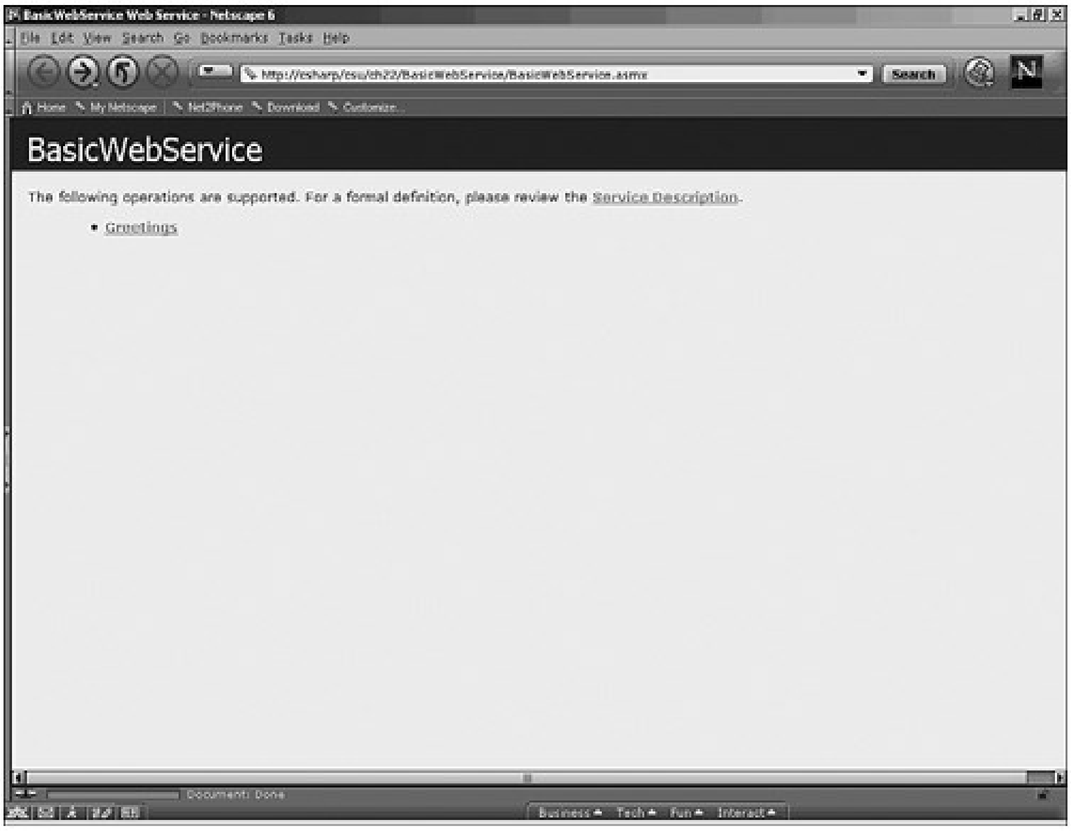
UNIT - 4
Advanced Features Using C#

Web Services

- A web service is a network accessible interface to application functionality, built using standard Internet technologies.
- A web service is a distributed computing technology that enables the exposure and reuse of logical business entities over the Internet.
- Creating ASP.NET web services is incredibly easy.
- ASP.NET web services are web services supported by the ASP.NET infrastructure.
- Several open standards technologies play a significant role in making web services a reality.
- These standards can be categorized by description, discovery, and transmission.
- The Web Services Description Language (WSDL) is an XML-based format for describing a web service.
- It describes what the web service is, its parameters, and how to use it.
- Universal Description Discovery and Integration (UDDI) directories support discovery.
- These directories manage WSDL documents and provide a means for clients to find and use web services.
- The Simple Object Access Protocol (SOAP) is a communications protocol that enables clients to interact with UDDI directories and web services.

Viewing Web Service Info

- The ASP.NET infrastructure provides the means to view information and test the operation of a web service.
- To do so, point your browser to the location of the web service on a web server.
- Figure shows the results of pointing a browser at a web service.


BasicWebService Web Service - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://esharp/csu/ch22/BasicWebService/BasicWebService.asmx
Search
Home My Netscape Net2Phone Download Customize...
Document: Done
Business Tech Fun Interact

BasicWebService

The following operations are supported. For a formal definition, please review the Service Description.

- Greetings

Using Web Services

- If an application wraps a method call into a SOAP envelope and uses HTTP, it could use the method to communicate with the BasicWebService web service.
- Alternatively, the client could use one of the HTTP GET or POST methods
- It's possible, but that would be a lot of work.
- The .NET Framework comes with a utility called wsdl that frees a client from creating all this plumbing.
- This utility takes the URL to a web service and creates a proxy, which is used by the client to call the web service.
- Using a web service requires a client to declare an instance of the proxy class and then call the necessary web service operation, defined in the proxy class.
- Web services provide a platform-independent means of exposing business logic over the Internet.
- They are created using several open standards technologies.
- Creating ASP.NET web services is easy and abstracts much of the complexity associated with Internet communications.
- This allows developers to concentrate on business logic rather than underlying plumbing.
- Use of a web service involves creating a proxy class that communicates with the web service on behalf of a client.

- The client application then communicates directly with the proxy to invoke necessary web service operations.

Windows Services

- Visual Studio has supported Windows service projects since .NET 1.0.
- To create a Windows service project, select File, New, Project, select the Visual C#\Windows branch of the navigation tree, and then select Windows Service.
- The Windows service project contains a couple items that are unique.
- This section explains what they are and how to get to the code are created on a new Windows service project.
- Notice that there aren't any spaces in the name of the project, which names the assembly by default spaces don't work with Windows services.
- A Windows service consists of method overrides that you implement to make your service work.
- The code for a service has several overridden methods, corresponding to the events of a service, such as Start, Pause, Continue, and Stop.
- To implement OnStart and OnStop, which are already provided in the shell code that the Windows Service Project Wizard creates for you.
- To view the code for the Windows service, either select the blank design surface or the Solution Explorer file for Service1.cs, right-click, and select View Code.

Implementing Windows Service Method Overrides

- The method overrides in a Windows service to get your program to work.
- To implement OnStart and OnStop methods as a minimal implementation.
- The OnStart method can be called when the OS boots and starts services marked as Automatic.
- It can also be called manually via the Services console that you can find via your OS Administrative Tools panel.
- The OnStart method of a Windows service has a total of 60 seconds to complete.
- To avoid the timeout, you can launch long-running processes on a thread, which enables the OnStart method to finish on time.

Configuring a Windows Service

- There are different configuration settings for a Windows service.
- Service itself permits specifying logging options, naming, and allowed states (for example, continue and pause).
- The method overrides in a Windows service to get your program to work to implement OnStart and OnStop methods as a minimal implementation.

- The OnStart method can be called when the OS boots and starts services marked as Automatic.
- It can also be called manually via the Services console that you can find via your OS Administrative Tools panel.
- The OnStart method of a Windows service has a total of 60 seconds to complete.
- After that, Windows will fail your service, which you can verify by looking at the Windows Event Log if it happens to you. To avoid the timeout, you can launch long-running processes on a thread, which enables the OnStart method to finish on time.

Installing a Windows Service

- The installer has two parts: a ServiceProcessInstaller and a ServiceInstaller.
- The ServiceProcessInstaller Configuring a ServiceInstaller.
- The ServiceInstaller provides setup for individual services, meaning that you can have multiple services running in the same process.
- Select the ServiceProcessInstaller and open the Properties window to configure it.
- The most important property to set is Account.
- The available settings are one of the ServiceAccount enum's values.
- The most configurable option is User, where you can give the service a machine or domain account.

Controller to Communicate with a Windows Service

- Control a Windows service from a separate application.
- SQL Server taskbar app is a good example.
- The example we use is a WPF application.
- So if you are following along, create a new WPF application and add a couple buttons to the window with Content properties set to Start and Stop and with Name properties set to btnStart and btnStop, respectively.

Messaging

- Provides access to the properties needed to define a Message Queuing message.
- Inheritance Hierarchy
  - System.Object,
  - System.MarshalByRefObject,
  - System.ComponentModel.Component,
  - System.Messaging.Message.
- Use the Message class to peek or receive messages from a queue to have fine control over message properties when sending a message to a queue.
- MessageQueue uses the Message class when it peeks or receives messages from queues, because both the MessageQueue.Peek and MessageQueue.Receive methods create a new instance of the Message class and set the instance′s properties.
- The Message class′s read-only properties apply to retrieving messages from a queue, while the read/write properties apply to sending and retrieving messages.
- The MessageQueue class′s Send method allows you to specify any object type for a message being sent to that queue.
- You can use the MessageQueue instance′s DefaultPropertiesToSend property to specify settings for generic messages sent to the queue.
- The types of settings include
  - Formatter,
  - Label,
  - Encryption, and
  - Authentication.
- You can also specify values for the appropriate DefaultPropertiesToSend members when you coordinate your messaging application to respond to acknowledgment and report messages.
- Using a Message instance to send a message to the queue gives you the flexibility to access and modify many of these properties—either for a single message or on a message-by-message basis.
- Message properties take precedence over DefaultPropertiesToSend.
- Message data is stored in the Body property and to a lesser extent, the AppSpecific and Extension properties.

- Data is encrypted, serialized, or deserialized, only the contents of the Body property are affected.
- The contents of the Body property are serialized.
- Change the Body or Formatter properties at any time before sending the message, and the data will be serialized appropriately when you call Send.
- The properties defined by the MessageQueue.DefaultPropertiesToSend property apply only to messages that are not of type Message.
- If you specify the DefaultPropertiesToSend property for a MessageQueue, the identically named properties in a Message instance sent to that queue cause these default properties to be ignored.
- Syntax
  - public class Message : Component.

Reflection

- Reflection objects are used for obtaining type information at runtime.
- The classes that give access to the metadata of a running program are in the System.Reflection namespace.
- The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values and objects to the application.

Use of Reflection

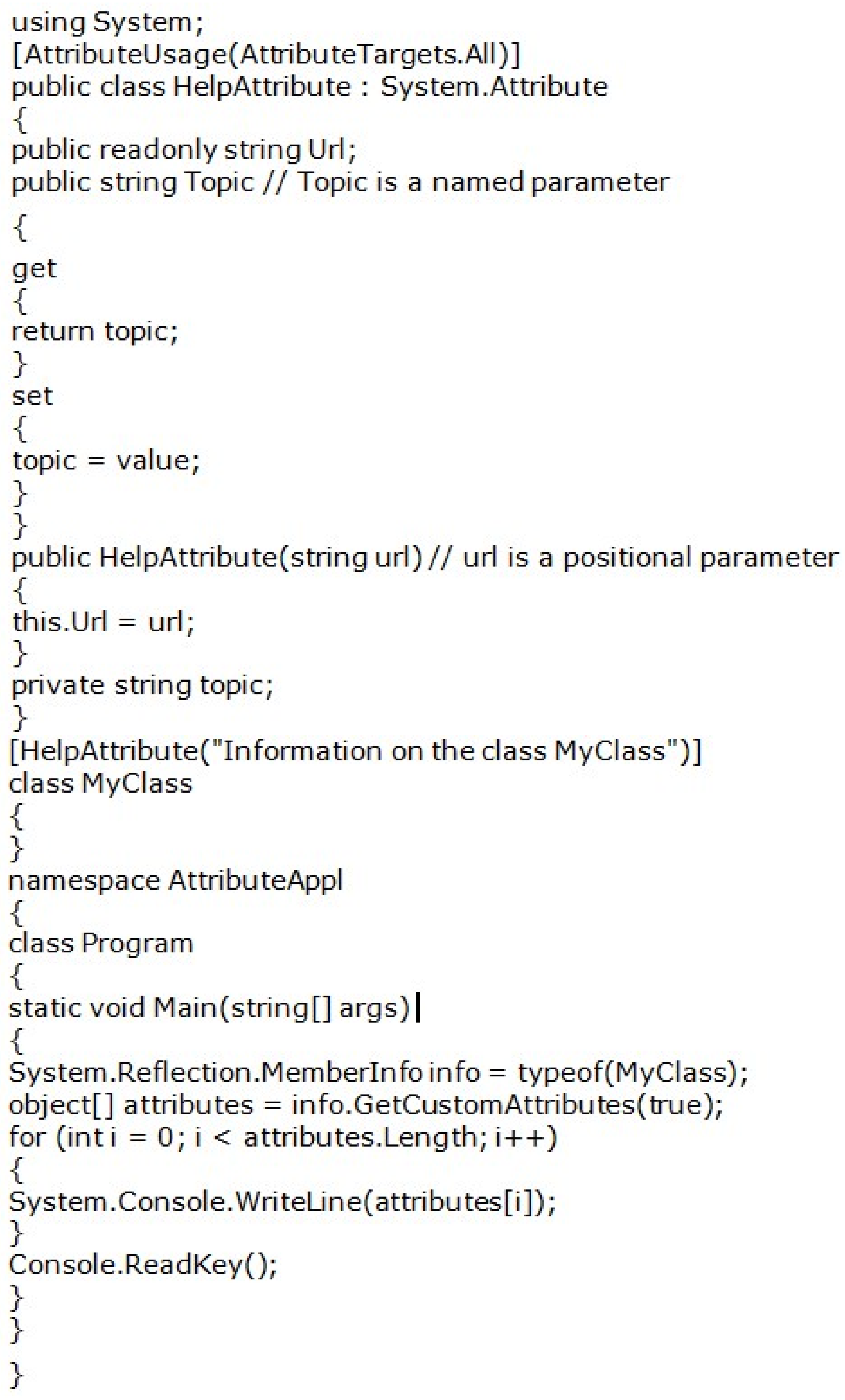
- Reflection has the following uses:
  - It allows view attribute information at runtime.
  - It allows examining various types in an assembly and instantiate these types.
  - It allows late binding to methods and properties.
  - It allows creating new types at runtime and then performs some tasks using those types.

Viewing Metadata

- Using reflection you can view the attribute information.
- The MemberInfo object of the System.Reflection class need to be initialized for discovering the attributes asscociated with a class.
- To do this, you define an object of the target class, as:
  - System.Reflection.MemberInfo info = typeof(MyClass);

C# Reflection

- The following program demonstrates this:



```
using System;
[AttributeUsage(AttributeTargets.All)]
public class HelpAttribute : System.Attribute
{
public readonly string Url;
public string Topic // Topic is a named parameter
{
get
{
return topic;
}
set
{
topic = value;
}
}
public HelpAttribute(string url) // url is a positional parameter
{
this.Url = url;
}
private string topic;
}
[HelpAttribute("Information on the class MyClass")]
class MyClass
{
}
namespace AttributeAppl
{
class Program
{
static void Main(string[] args)
{
System.Reflection.MemberInfo info = typeof(MyClass);
object[] attributes = info.GetCustomAttributes(true);
for (int i = 0; i < attributes.Length; i++)
{
System.Console.WriteLine(attributes[i]);
}
Console.ReadKey();
}
}
}
```

- When it is compiled and run, it displays the name of the custom attributes attached to the classMyClass:

  - “HelpAttribute”.

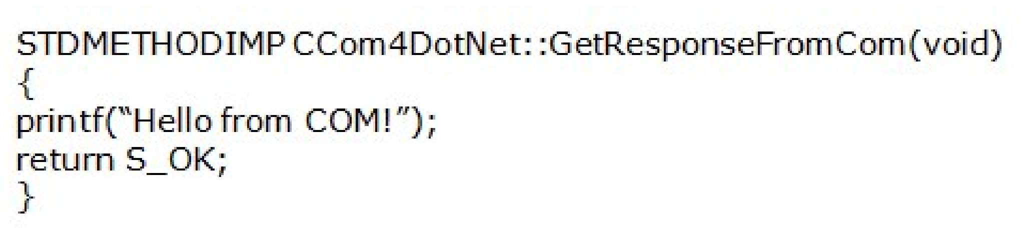
COM and C#

Communicating with COM

- One of the most likely interop scenarios is communicating from .NET to existing COM components.
- This can allow preservation of existing infrastructure and reduction in overall development cost.
- COM components may be called via either early or late binding.
- Via early binding, legacy COM components can be made to appear as managed objects in the .NET environment.
- This is accomplished by a utility that reads an existing type library and creates a proxy for the .NET component to interact with.
- Early bound components are those that are bound at compile time.
- This promotes type safety and improves a program’s overall performance.

Early-Bound COM Component Calls

- A C++ COM Component: ComObj.dll



```
STDMETHODIMP CCom4DotNet::GetResponseFromCom(void)
{
printf("Hello from COM!");
return S_OK;
}
```

- A .NET Framework to create a proxy for the COM component.
- Output:
  - Hello from COM!
- The proxy is then compiled into the C# program the COM component can be instantiated and called just like any other managed component.

Late-Bound COM Component Calls

- When a type library isn't available, there's a dynamic invocation requirement, C# program can perform a late-bound call to a COM component.
- Late -bound COM component invocations are performed using C# reflection.
- It first obtains a ProgID from the COM object, as listed in the Windows regis try.
- When a type object is obtained, an object is created using the static CreateInstance method of the Activator class.

- The GetResponseFromCom method of the COM component is then invoked with the InvokeMember method of the Type object, lateBoundType.

Exposing a .NET Component as a COM Component

- A C# Component Exposed as a COM Component: CallFromCom.dll
- C# components are accessible as COM components.
- .NET Framework tools to create an unmanaged proxy and enter the proper settings in the registry.
- This enables unmanaged code to use .NET components as if they were COM components.
- This command line registers the C# library as a COM component.
- In addition, the /tlb option creates a type library to facilitate early binding.
- To generate a type library without registering the library, use the TlbExp program as shown in the following command line:
  - TlbExp CallFromCom.dll
- Calling a C# Component Exposed as a COM Component
  - AVB program calling a C# COM component.
  - A message box showing the response from a C# COM component.

Introduction to .NET Support for COM+ Services

- The .NET Framework provides extensive support for COM+ services such as
  - Transactions,
  - JIT activation,
  - Object pooling, and others.
- COM+ services are activated through the use of attributes, which decorate a specific C# element as appropriate.
- These attributes are analogous to the COM+ concepts.

Installed Assembly

- The other step necessary to make this C# program work as a COM+ service is to register it.
- The ApplicationName attribute identifies the COM+ name, and the AssemblyKeyFile attribute specifies the strong name key to register the assembly with.
- Following command line shows how to create a strong name key:

  n -k CPSkel.snk.

➢ Transactions

- A transaction is a way to combine multiple actions into a single body of work to guarantee that all actions either succeed or fail together.
- A COM+ Transactional Component in C#: CPTrans.cs
- C# programs can participate in COM+ services transactions by inheriting from the ComPlusServices class and marking their classes with a Transaction attribute.
- Another transaction-related attribute is AutoComplete, which enables a transaction to commit automatically if all items succeed.
- However, if an exception is raised, AutoComplete causes the transaction to abort.

JIT Activation

- Just-In-Time (JIT) activation is the capability to instantiate a new component.
- Component go away automatically when it's no longer needed.
- By using COM+ services and associated attributes, a C# component can participate in JIT activation.
- Implementing JIT activation requires specifying the Just-In-Time Activation attribute.
- JIT activation is true by default, but if you want to turn it off, specify false as the first attribute parameter.

Object Pooling

- Another COM+ service enabling efficient use of resources is object pooling.
- An object pool is a group of components that stay activated and ready for connections at all times.
- This reduces the overhead associated with activation and deactivation of components.
- The ObjectPooling attribute has three parameters:
  - ➢ Enabled,
  - ➢ MinPoolSize, and
  - ➢ MaxPoolSize.
- The Enabled parameter turns object pooling on.
- The MinPoolSize parameter specifies the minimum number of objects held in the pool, and the MaxPoolSize parameter specifies the maximum number of objects to be held in the pool.

Other Services

- COM+ services include several other technologies such as
  - Roles,
  - Security, and message queuing.
- Using the techniques described in other sections and examining the applicable attributes in the System.EnterpriseServices namespace.
- C# and COM+ services security are mutually exclusive.

Localization - Resource Files

- Setting up and using resource files is the primary means of localizing programs.
- Resource files are specialized binary files that can be bound to a standalone DLL or added into a program assembly.
- They contain strings, graphics, and other binary resources that assist in localizing a program.

Creating a Resource File

- The resource generator utility ResGen is a string resource creation utility that comes with the Microsoft .NET Framework SDK.
- Given a properly formatted .txt (text) file, ResGen converts it into a .resources (binary resources) file that can subsequently be added to an assembly.
- Without a special resource creation tool, string resources begin life as a specially formatted .txt file.
- They have headers, comments, and name/value pairs.
- Using Resources: StringRes.cs
- output :
  - Greeting: Hello
- Header elements must match the filename without the extension.
- For example, if the resource file's name is myResources.txt, the header contents must be [myResources].
- Comments are useful for delimiting groups of resource strings or adding more information to the use of a string.
- All comments are removed from compiled resources.

Writing a Resource File

- Resource files may be created programmatically.
- This is useful for automated .resources file generation utilities or resource tools in IDEs.
- The steps involved in creating a .resources file are to open a ResourceWriter stream, add whatever resources are needed, and then close the stream.
- Writing a Resource File: ResWrite.cs
- The default ResourceWriter class in the System.Resources namespace implements IResourceWriter.

- This is why it's possible to create an IResourceWriter object within the Main method of the ResourceWriter constructor accepts a string parameter specifying the resource file to create.
- If a file by that name exists, it will be overwritten.

Reading a Resource File

- Reading a Resource File: ResRead.cs
- The ResourceReader class is also useful in creating resource manipulation utilities and IDE tools to manage resources.
- A utility uses the ResourceReader functionality to read in an existing resource file, the program performs any necessary manipulations, and then the ResourceWriter helps write the new resources back to the persistent .resources file.

Converting a Resource File

- Generated .resx File: strings.resx
- Another use of ResGen is to convert between .txt, .resources, and .resx files.
- Resx files are XML format files used for binary resources such as graphics, fonts, icons, and cursors.
- For example, the following command line converts the .resources file to a .resx file:
  - ➢ resgen strings.resources strings.resx
- This produces an XML format fileThe same exact file would have been generated if you had performed the following command line:
  - ➢ resgen strings.txt strings.resx

Creating Graphical Resources

- Path
  - ➢ C:\Program Files\Microsoft.Net\FrameworkSDK\Samples\tutorials\resources and localization\graphics\cs\images directory
- Run Command
  - ➢ ResXGen /i:un.jpg /o:graphics.resx /n:flag
- The .NET Framework SDK includes a couple of sample programs that help manage graphical resources.
- The ResXGen Utility
  - ➢ One is the ResXGen program, which adds a graphic to a .resx file.
- The ResEditor Utility

    - The other is the ResEditor program, which manages all types of resources for .resources files.

- Using Graphical Resources
    - An added bonus is that these two utilities come with source code, enabling you to examine graphical resource manipulation code in detail.

Multiple Locales

- The purpose of resource files is to support multiple locales.
- The official way of specifying locales is via cultures, as specified in RFC 1766, ISO 639, and ISO 6133.
- Cultures are denoted with four-character designations.
- The first two characters specify the language in lowercase, and the second two specify the country or region in uppercase.
- A separate resource file must be created for each locale.
- There are multiple ways to deploy these resources: compiled into a program, via satellite assembly, or via a global assembly.
- Through a combination of resource files and a directory structure geared toward targeted cultures, any program can be localized.
- The directory structure corresponds to each culture implemented in a program.
- The following directory structure supports localization for a program named MultiCulture:
    - MultiCulture,
    - en,
    - en-US,
    - en-GB.
- A Localized Program: MultiCulture.cs
- If the appropriate culture subdirectory is present, a localized program can automatically pick up the resources corresponding to its default locale.

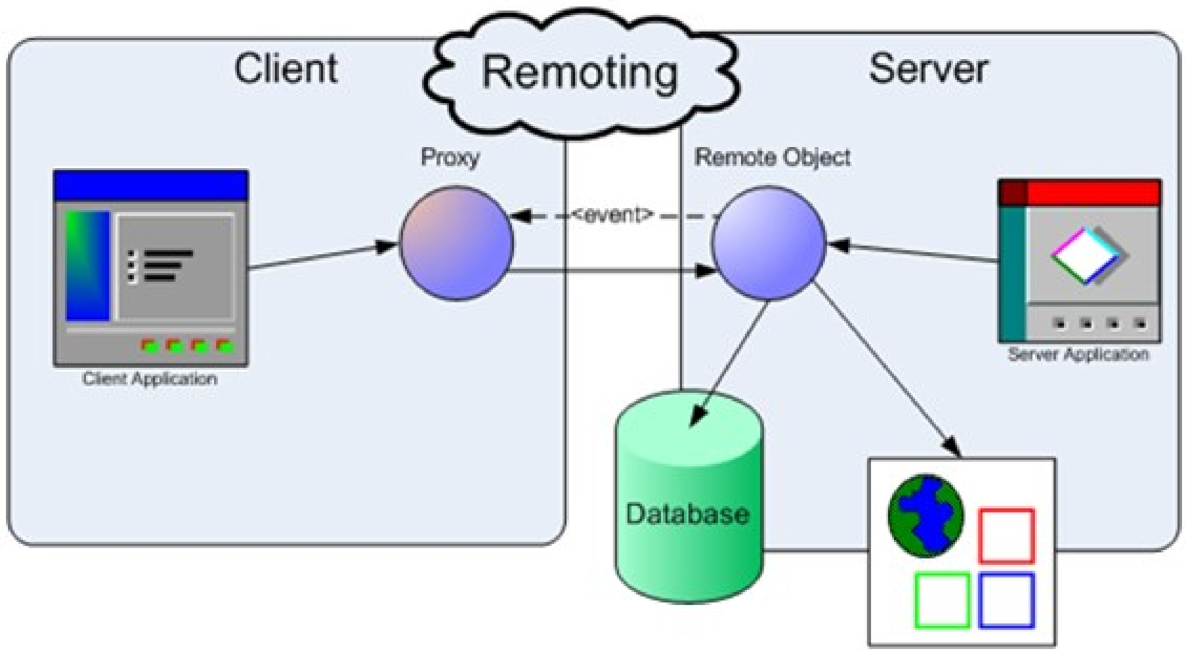
Finding Resources

- Search the global assembly cache for the specific resource.
- The global assembly cache as a central repository for all programs on a machine to access.
- Search culture subdirectories of the localized program.

- Search the global assembly cache for parent resources.
- For instance, if the original resource selected, but not found, was en-US, search the global assembly cache for en only.
- Search culture subdirectories of the localized program for the parent resources.
- Search culture subdirectories of the localized program for parent resources of the last parent resource searched.
- A resource has only a single parent, but the chain of parents can extend multiple levels.

Distributed Application

- Remoting is an infrastructure that allows the developer to use remote objects.
- Remote objects are objects that are based (or instantiated) outside of the caller's Application Domain.



- This example shows you how to use both remote object access mechanisms (pass by value and pass by reference).
- It also shows you the power of remoting for distributed computing with a simple yet powerful implementation of a Task Server.

Basic Remoting

- Remoting is the capability to communicate with components in separate AppDomains.
- Figure shows a simplified view of two objects communicating via remoting: a client component in AppDomain A communicates with a server component in AppDomain B.
- An AppDomain is an execution environment within a process.
- It separates managed applications during execution.
- This provides several benefits including reliability and security.
- Remoting supports multitiered as well as distributed architectures, the server component in AppDomain B could easily be extended to a client role, communicating with a server component in another AppDomain.
- Furthermore, a remote component could be located in another process on either the same or a different machine.
- When a component is set up, the underlying plumbing required to maintain remote

communications is hidden by the remoting architecture.

- There is a server, which is a DLL that must be hosted by another application, either IIS or a custom host that you build.
- There is also a client, which is an executable that makes an out-of-process call to communicate with the remoting server.
- I'll show you how to build the remoting server first, and then the client, and finally, I'll show you a couple ways to host the server.
- After the server is hosted and the host is running.

Remoting Server

- A remoting server object is just a class that derives from MarshalByRefObject.

BasicRemotingServer.cs

- The BasicServer class inherits MarshalByRefObject, which supports the basic functionality for a callable component over the remoting rchitecture.
- This class is implemented as a DLL, negating the need for a Main method.
- The only method in this class is the GetServerResponse method, keeping things simple.

web.config

- Each remoted component requires a configuration file, named web.config, to specify necessary operating parameters.

Remoting Client

BasicRemotingClient.cs

- Writing a remoting client is just a little more involved than writing a remoting server.
- Writing a client requires finding out the type and location of the remote server object, initializing configuration from a file, and creating an instance of the remote object with the type and location information obtained earlier.
- The System.Runtime.Remoting namespace contains all the basic remoting classes.
- Our remote server component that will be instantiated and called is in the BasicServer namespace.

BasicRemotingClient.exe.config

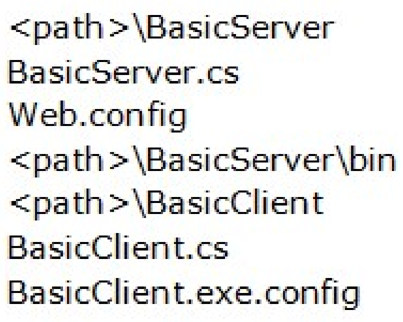
- Within the Main method, the type of the BasicRemotingServer class is obtained with the typeof operator and stored in a Type object.
- Next, the URL of the BasicRemotingServer.

Remoting Setup

- There are two ways to get a remoting application up and running: via a web server or a host utility.
    - Web Server Setup.
    - Host Utility Setup.
- Use Microsoft Internet Information Services (IIS) as the web server to host the remote server component.

Web Server Setup

- Demonstrating the host utility requires code for a new client and server program and the utility.
- The following steps show how to set up a remote server component via the IIS web server.
    - Before actually starting this procedure, work out a directory structure so that it will be easy to follow along.
    - Here's what your directory structure should look like:



```
<path>\BasicServer
BasicServer.cs
Web.config
<path>\BasicServer\bin
<path>\BasicClient
BasicClient.cs
BasicClient.exe.config
```

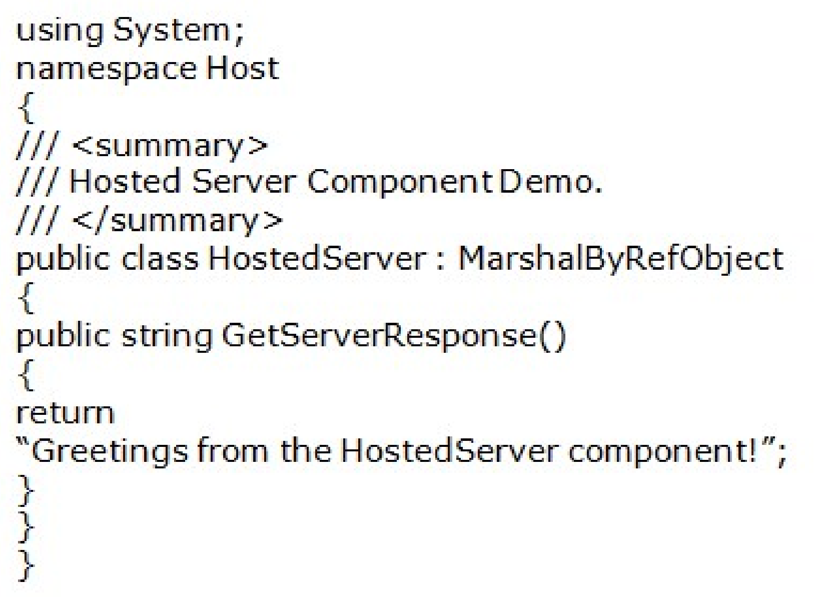
    - Compile the server component from the BasicServer directory with the following command line:
        - csc /t:library BasicServer.cs
    - Open Internet Information Services (IIS).
    - The following steps work for Windows XP but are different for other operating systems.
    - For example, WinNT, Win2K, Win2003, Win2008, and Vista are all different.
    - If the actions don't match your OS exactly, consult your documentation to accomplish the same tasks.
    - Expand the server node under which you want to create a virtual directory and right-click Default Web Site.
    - From the menu, select New, Virtual Directory.
    - This opens the Virtual Directory Creation Wizard. Click Next.

- In the text box for an Alias, type in any meaningful name for the virtual directory, such as BasicServerDemo.
- Click Next.
- In the text box for the physical path that the virtual directory will refer to, enter the < path >\BasicServer directory, where < path > is the actual directory you specified in step 1. Click Next.
- There are several access permissions from which to choose.
- The Read and Run Scripts (such as ASP) options are already checked, and that's fine.
- Accept the defaults, click Next, and then click Finish on the last screen.
- IIS will use the web.config file in the BasicServer directory when it loads the server component.
- The remote server component is now set up.
    - /r:..\BasicServer\BasicServer.dll BasicClient.cs
- Use the following command line to compile the remoting client program from csc
    - /r:..\BasicServer\BasicServer.dll BasicClient.cs Optionally
- Remember to add a reference to the BasicServer project; right-click the References folder in BasicClient, select Add Reference, click the Projects tab, select the BasicServer project, and click the OK button.
- You'll also need to add a using declaration for the BasicServer namespace.
- Finally, run the BasicClient.exe program in the BasicClient folder to test the system out.
- If all goes well, the following output will be printed to the console:
    - Greetings from the BasicRemotingServer component!

Host Utility Setup

- The host utility setup method uses a program that configures the remote server component.
- The process isn't necessarily easier or harder than the web server setup method.
- Replace the BasicServer with HostedServer and the BasicClient with the HostedClient directory names.

Hosted Server Demo: HostedServer.cs



```
using System;
namespace Host
{
/// <summary>
/// Hosted Server Component Demo.
/// </summary>
public class HostedServer : MarshalByRefObject
{
public string GetServerResponse()
{
return
"Greetings from the HostedServer component!";
}
}
}
```

Remoting Client Demo: HostedClient.cs

- Replace the BasicServer withHostedServer and the BasicClient with the HostedClientdirectory names.

Remoting Client Configuration File: HostedClient.exe.config

- The host utility setup method uses a program that configures the remote server component.
- The process isn't necessarily easier or harder than the web server setup method.
- Channels marshal, format, and transmit messages across AppDomains.
- Each of a channel's tasks opens new opportunities for extensibility.

Host Utility Demo: RemotingHost.cs

- For example, message contents can be marshaled to conform to the proper data representation using custom sinks.

Host Utility Configuration File: RemotingHost.exe.config

- The message itself can be formatted via the built-in Simple Object Access Protocol (SOAP) or binary formatters.
- In addition, transport protocols, such as HTTP or TCP, which are built in, can be configured with ease.
- The architecture also supports customizable marshalling, formatting, and transmission components that can be plugged in as needed.

Programmatic Remoting Channel Registration: RemotingProxyClient.cs

- The HTTPChannel, linking the proxy in AppDomain A to the stub in AppDomain B, is a built-in channel component supporting default marshalling, XML/SOAP formatting, and HTTP protocol transmission.

Lifetime Management

- A remote server component exists for a default amount of time and then makes itself available for garbage collection.
- It's often more desirable to explicitly manage the lifetime of remote components.
- This is why the remoting framework provides a leasing mechanism for finer granularity of control in remote component lifetime management.
- Remote leasing operates via a collaborative protocol between one or more client components, a server component, and a lease manager.
- Remote server components begin life with a designated amount of time before garbage collection.
- Client components register with the server component's lease manager for notification of when the server's lifetime is expiring.

Remote Leasing Demo: LeasingDemo.cs

- The lease manager keeps track of server components and notifies clients of when the server will expire.
- When a server has reached its expiration time, the lease manager notifies the client and waits for a designated amount of time for a reply from the client.
- If the client wants the server to remain alive, it returns the amount of time the server can live to the lease manager.
- If the designated reply time from the client to the lease renewal query expires, the lease manager marks the server object for garbage collection.

Extensible Markup Language (XML)

- XML is an international standard, computer systems can communicate effectively, applications can read and parse information with a common API.
- XML is an open standard that can work everywhere, regardless of your hardware platform, operating system, programming language.
- XML permeates nearly every part of the .NET.
- XML class libraries to interact with XML data.
- It is the essential data transport format for web services, the format for configuration files, and the source of data from external systems.
- The .NET APIs make it easy to use XML as a standardized method of passing information between programs, file saving and reading, data validation, and many other useful tasks.
- This chapter explains how to use C# and the XML class libraries to interact with XML data.

Writing XML

- Writing XML documentation is greatly simplified with the System.XML class library.
- The particular class used in this section is the XMLTextWriter class.
- It has numerous convenience methods that make producing XML documents a snap.
- The .NET Framework documentation lists all the available methods for the XMLTextWriter class.

Writing an XML Document with XmlTextWriter

- The XML streaming APIs encapsulate the code necessary to parse XML files and obtain data related to specific tags.
- Sometimes you need the flexibility of working with XML data in memory, via the Document Object Model (DOM).

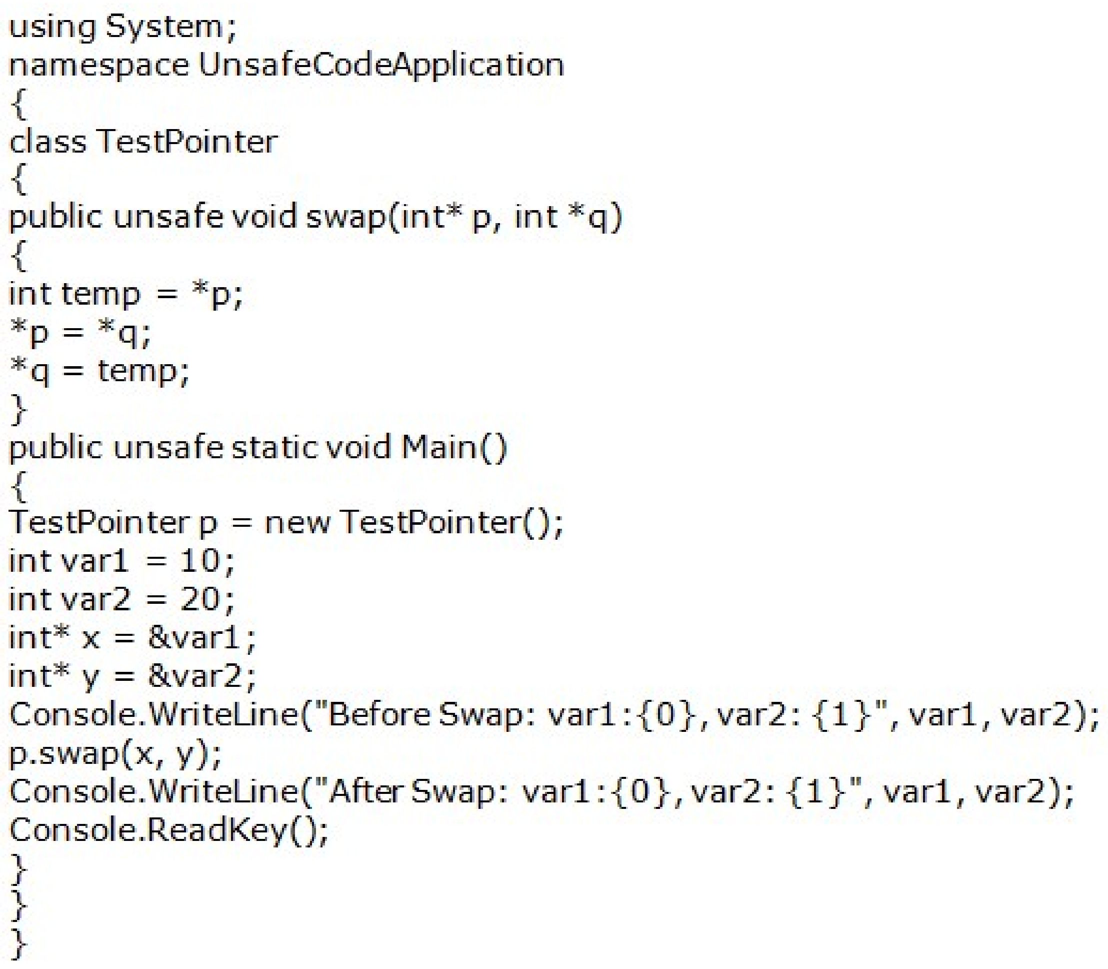
Reading an XML Document with XmlTextReader

- To work with XML in memory, you can use an XPathDocument and then use an XPathNavigator to read each node efficiently.
- The difference between this technique and the XmlTextReader, in addition to what has been previously stated, is that an XPathNavigator can move anywhere in the document, whereas an XmlTextReader is forward only.
- The task of modifying an XML document in memory requires using the XmlDocument and XPathNavigator.
- This is a common technique for making the code more maintainable, because if the node name changes, you need only change the TalkNode constant.

- The second string parameter is the element value.
- There′s no need for formatting, because the WriteElementString method calls do it automatically.

Unsafe Mode

- C# allows using pointer variables in a function of code block when it is marked by the unsafe modifier.
    - Safe,
    - Unsafe,
    - Managed,
    - Unmanaged.
- The unsafe code or the unmanaged code is a code block that uses a pointer variable.
- Unsafe code permits the use of pointers, which supports certain performance optimizations and interface to legacy code and operating systems.
- Unsafe code is identified with a special keyword, unsafe, which marks either a block of code or a field.
- This establishes an unsafe context where pointer operations can be implemented.
- The normal mode of operation in a C# program is safe.
- When code is safe, its type is safe and secure.
- Unsafe code is identified by the unsafe keyword.
- This is code that is allowed to use pointers.
- All C# code is managed. Managed code is under control of the CLR, which has full control of all memory and security operations.
- A pointer is a variable whose value is the address of another variable i.e., the direct address of the memory location.
- Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
- You can retrieve the data stored at the located referenced by the pointer variable, using the ToString()method.
- You can pass a pointer variable to a method as parameter.
- The following example illustrates this:



```
using System;
namespace UnsafeCodeApplication
{
class TestPointer
{
public unsafe void swap(int* p, int *q)
{
int temp = *p;
*p = *q;
*q = temp;
}
public unsafe static void Main()
{
TestPointer p = new TestPointer();
int var1 = 10;
int var2 = 20;
int* x = &var1;
int* y = &var2;
Console.WriteLine("Before Swap: var1:{0}, var2: {1}", var1, var2);
p.swap(x, y);
Console.WriteLine("After Swap: var1:{0}, var2: {1}", var1, var2);
Console.ReadKey();
}
}
}
```

Accessing Array Elements Using a Pointer,

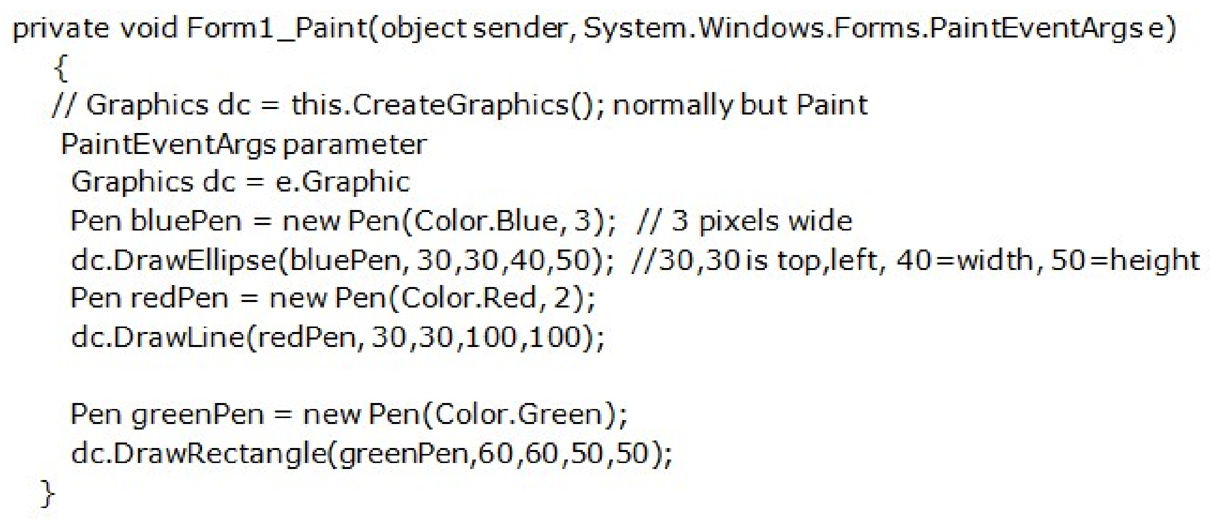
- In C#, an array name and a pointer to a data type same as the array data, are not the same variable type.
- For example,
  - int *p and int[ ] p, are not same type.
- You can increment the pointer variable p because it is not fixed in memory but an array address is fixed in memory, and you can't increment that.
- To access an array data using a pointer variable, as we traditionally do in C, or C++ (please check: C Pointers), you need to fix the pointer using the fixed keyword.
- For compiling unsafe code, you have to specify the /unsafe command-line switch with command-line compiler.

Graphical Device Interface

- The general framework that deals with the DC(Device Context) is referred to as the GDI (Graphical Device Interface).
- In .Net, the GDI interface has been greatly refined and is referred to as GDI+.
- The .Net GDI+ classes encapsulate some of the important Windows API to create graphical outputs. popular GDI+ classes needed in creating windows applications.
- The System.Darwing contains other namespaces for specialized applications such as System.Drawing.Imaging for image processing, System.Drawing.Design for design time controls such as dialog boxes, property sheets etc.., System.Design.Text for controlling fonts, System.Drawing.Printing for controlling printing and print preview.

Example

- Create a windows application. Name the project “gditest”.
- Right click on Form1.cs in the project explorer and choose “view code”.
- Type the following code in the constructor for form1 class.
- Add a Paint event handler by clicking on the lightning symbol above the properties window and double clicking on the Paint event.
- Type the following code in it.
- Note that the Paint event handler gets a DC through the PaintEventArgs parameter, so you can simply use, Graphics dc = e.Graphics; to obtain the DC.



```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
    {
    // Graphics dc = this.CreateGraphics(); normally but Paint
     PaintEventArgs parameter
      Graphics dc = e.Graphic
      Pen bluePen = new Pen(Color.Blue, 3);  // 3 pixels wide
      dc.DrawEllipse(bluePen, 30,30,40,50);  //30,30 is top,left, 40=width, 50=height
      Pen redPen = new Pen(Color.Red, 2);
      dc.DrawLine(redPen, 30,30,100,100);

      Pen greenPen = new Pen(Color.Green);
      dc.DrawRectangle(greenPen,60,60,50,50);
  }
```

- The PaintEventArgs parameter in the Paint handler also contains information about the clipping rectangle.
- The clipping rectangle is the rectangular area that needs to be repainted (this is similar to the invalidated region in MFC).
- The clipping rectangle becomes important when there are a large number of objects to be

redrawn in the Paint handler.

- Using the clipping region information, we can choose to redraw only a few of these objects which are being affected.
- Brushes are used to fill a particular region with a color or a pattern.
- For example, the DC has a FillRectangle method and a FillEllipse method which you can pass the brush and the bounding rectangle to fill the shape with the brush.

Case Study (Chat Program)

- The application is a simple chat tool.
- Anyone who connects to the chat server receives all chat communication between the connected users.
- The application is deliberately kept simple to clarify .NET remoting.
- This application also demonstrates event and delegate usage in an application that uses .NET remoting.
- Most of you will have noticed that there are wizards to develop ASP.NET web services in Visual Studio .NET.
- However, if you want to develop a Client/Server application that runs over a local area network, there are no wizards available.
- This application shows how to do .NET remoting in a Local Area Network.
- This application acts much like a DCOM server and client application.
- This Visual Studio .NET solution contains three modules.
    - ChatCoordinator: The server implementation.
    - ChatClient: The client implementation.
    - ChatServer: The runtime host for Chat coordinator.

UNIT - 5
ASP.NET 2.0

What’s New in ASP.NET

- The Microsoft .NET Framework version 2.0 includes significant enhancements to ASP.NET in virtually all areas.
- ASP.NET has been improved to provide out-of-the-box support for the most common Web application situations.
- You will find that you can get Web sites and pages up and running more easily and with less code than ever before.
- At the same time, you can add custom features to ASP.NET to accommodate your own requirements.
- Specific areas in which ASP.NET has been improved are:
  - Productivity
    - You can easily and quickly create ASP.NET Web pages and applications using new ASP.NET server controls and existing controls with new features.
    - New areas such as membership, personalization, and themes provide system-level functionality that would normally require extensive developer coding.
    - Core development scenarios, particularly data, have been addressed by new data controls, no-code binding, and smart data-display controls.
  - Flexibility and extensibility
    - Many ASP.NET features are extensible so that you can easily incorporate custom features into applications.
    - The ASP.NET provider model, for example, provides pluggable support for different data sources.
  - Performance
    - Features such as pre compilation, configurable caching, and SQL cache invalidation allow you to optimize the performance of your Web applications.
  - Security
    - It is now easier than ever to add authentication and authorization to your Web applications.
  - Hosting
    - ASP.NET includes new features that make it easier to manage a hosting environment and create more opportunities for hosters to add value.
- Computer languages and frameworks evolve as the needs of the development community evolve.

- ASP.NET is certainly no exception.
- ASP.NET 2.0 is the first major update to the ASP.NET framework and includes solutions to common problems encountered with ASP.NET 1.x as well as new features that greatly increase the flexibility and functionality of Web development on the .NET platform.
- If you are still working with ASP applications, then ASP.NET 2.0 offers even more incentive to upgrade.

Goals of ASP.NET 2.0

- Improve the reliability and usability of Web applications:
  - Currently, many ASP.NET 1.x applications run on Microsoft IIS 5.0. ASP.NET 2.0 leverages new IIS 6.0 features for improved performance and scalability.
  - Specifically, IIS 6.0 provides a new process model which greatly enhances the ability of a server to host multiple applications in a truly independent fashion.
  - Each ASP.NET application resides in its own isolated process and cannot accidentally interact with other applications.
  - Quite simply put, applications can no longer break each other.
  - Each application runs completely separated from every other application. If one application crashes, it doesn't affect other applications.
  - In terms of usability, new features such as master pages and themes allow you to develop large Web applications using a consistent structure that is manageable and configurable.
- Reduce the number of lines of code you have to write in common scenarios:
  - ASP.NET 2.0 includes wizards and controls that allow you to perform frequent tasks (e.g. data access) without having to write a single line of code.
  - Visual Studio 2005 includes designers to layout and configure complex pages with data bound tables.
  - As a developer, you can use the wizards to work faster and smarter.
  - ASP.NET 2.0 also leverages changes in the .NET Framework.
- Offer user features to personalize Web applications:
  - ASP.NET 2.0 includes built-in controls to help you manage user accounts and personalize the content and layout of pages in your application.
  - First, the membership service lets you track users.
  - The new login controls integrate with the membership service to allow you to automate account creation and user login without writing any code.

  The new Web Parts feature allows you to create Web applications that contain controls that

   - can be edited and personalized by users.
   - Users can select and customize the parts that are displayed on a Web page as they see fit.
   - Finally, the Profile service provides long-term persistence of user preferences and data through declarative XML configuration.

- Provide enhanced design features to generate consistent layouts and design:
   - ASP.NET 2.0 supports master pages, themes and skins to build applications with a consistent page layout and design.
   - These new features are easy to implement and modify, and greatly enhance the manageability and maintainability of large applications.
   - Many of the new features in ASP.NET 2.0 are aimed specifically at addressing these goals.
   - ASP.NET 2.0 builds on ASP.NET 1.x to provide a powerful Web application development platform.

What Does ASP.NET 2.0 Mean To ASP Developers?

- For ASP developers, the impact of the many new features in ASP.NET 2.0 is largely dependent on your development background.
- You no longer have to worry about HTML tags, POST, GET, and query strings.
- You can focus on developing code that implements business logic.
- This spaghetti code made readability very difficult.
- Scripting code could be inserted anywhere, which also made it hard to build well-designed applications that shared code properly and in an extensible fashion.
- You can also leverage master pages to quickly create a coherent design for your application, or apply skins and themes to programmatically set the look and feel of your Web pages.

Changes in Architecture

- The fundamental architecture of ASP.NET has always been designed for flexibility and extensibility.
- ASP.NET 2.0 continues this tradition by incorporating a new provider model to support many of the new features.
- New utilities and API's have been added to improve site maintenance and improve configuration.
- All of these changes are designed to make developing ASP.NET 2.0 applications a faster and more streamlined process while still providing the flexibility and extensibility that developers were used to with ASP.NET 1.x.

ASP.NET 2.0 Providers

- The provider model defines a set of interfaces and hooks into the data persistence layer that provides storage and retrieval for specified requests.
- In this way the provider model acts as a programming specification that allows ASP.NET 2.0 to service unique client concerns.
- ASP.NET 2.0 uses a wide variety of providers, including:
  - Membership
    - The membership provider manages supports user authentication and user management.
  - Profile
    - The profile provider supports storage and retrieval of user specific data linked to a profile.
  - Personalization
    - The personalization provider supports persistence of Web Part configurations and layouts for each user.
  - Site Navigation
    - The site navigation provider maps the physical storage locations of ASP.NET pages with a logical model that can be used for in-site navigation and linked to the various new navigation controls.
  - Data providers
    - ADO.NET has always used a provider model to facilitate the connection between a database and the ADO.NET API.
    - ASP.NET 2.0 builds upon the data provided by encapsulating many of the ADO.NET data calls in a new object called a data source.
    - Each type of provider acts independently of the other providers.
    - You can therefore replace the profile provider without causing problems with the membership provider.

The ASP.NET 2.0 Coding Model

- In ASP.NET 1.x, you could develop an ASP.NET page in one of two ways.
- First, you could put your code directly inline with your ASP.NET tags.
- The code inline model is very similar to the coding model that was prevalent with classical ASP and other scripting languages.
- However, the code inline model has several problems such as the intermixing of code and HTML.
- ASP.NET 1.0 introduced the code behind model as a replacement.

- The code behind model used an external class to house the code, while the ASPX page contained the HTML and ASP.NET tags.
- The code behind model thus successfully separated code from content; however it created some interesting inheritance issues and forced the developer to keep track of two files for each Web page.
- The primary difference between a code behind file in ASP.NET 1.x and ASP.NET 2.0 is that a code behind file is now a partial class rather than a full class that inherits from the ASPX page.
- A partial class is a new .NET construct that allows you to define a single class in multiple source files.
- In ASP.NET 2.0, a partial class is particularly useful for code behind files as it removes the inheritance relationship that is present with the older code behind model.
- Two partial class are merged into a single class during compilation.
- The code behind file is therefore free of all of the control declarations and inheritance issues associated with the old code behind model.
- The most striking difference can be seen in the actual code behind file, which no longer contains all of the auto-generated code that used to be necessary to maintain the inheritance relationship.
- A new code behind file:
  - When you use the code-behind model in Visual Studio 2005, any code you add to the page will automatically be added to a < script > block within the ASPX file instead of to a code behind class.
  - However, Visual Studio 2005 still displays the code in the code view.
  - In other words, you can keep using Visual Studio like you always have, except that code will be placed directly in the ASPX page instead of a separate class.

Configuration and Site Maintenance

- One of the more difficult challenges as an ASP.NET developer was properly configuring the Web.config file.
- In ASP.NET 2.0 and Visual Studio 2005, you now have several new features to help you with this task.
- IntelliSense in Web.config:
  - First, Visual Studio's IntelliSense feature has now been extended to any XML file that has a valid schema.
  - In the case of the Web.config file, this means that you get full IntelliSense support whenever you edit the Web.config file from within Visual Studio.

- IntelliSense helps reduce the chance of misconfigured files.
- However, ASP.NET 2.0 also includes a new administrative Website and a Microsoft Management Console to make things even easier.

Administrative Website

- To simplify the process of managing users, it provides a built in Website configuration tool.
- The Web Site Administration Tool is a simple Website that can only be accessed on the localhost through Visual Studio 2005.
- Through this tool, an administrator can manage the application by configuring services such as user management, the personalization providers, security, and profiles.
- The tool also allows you to easily configure debugging and tracing information for your application.
- Although the tool is limited to local applications, you can easily access all of the same configuration features through the improved configuration and administration API.

Microsoft Management Console Snap-In

- ASP.NET 2.0 deploys a special Microsoft Management Console (MMC) snap in for IIS that lets you decide which applications should use which versions of the .NET Framework.
- The MMC IIS tab lets you to choose which version of ASP.NET your application uses and displays the Web.config location.
- In addition to managing the framework version, the console has an "Edit configuration" button which lets you visually edit most of the Web.config settings without having to directly manipulate the Web.config XML file.
- As an administrator, you will find that this MMC snap-in provides an incredibly useful tool for configuring and managing multiple ASP.NET applications on a single server.

Enhanced Configuration APIs

System.Configuration.Configuration class

- We can also retrieve and edit configuration information using the System.Configuration.Configuration class.
- API's let you programmatically access XML configuration files.
- The following code displays the type of authentication enabled for an application on your local machine.

```
Configuration config =
WebConfigurationManager.OpenWebConfiguration("~");
AuthenticationSection authSection =
config.GetSection("system.web/authentication") as
AuthenticationSection;
Response.Write("Authentication mode is: " + authSection.Mode);
```

Compilation Options

- ASP.NET 2.0 offers four different compilation models for a Web application:
  - Normal (ASP.NET 1.x)
    - In a normal ASP.NET Web application, the code behind files were compiled into an assembly and stored in the /bin directory.
    - The Web pages (ASPX) were compiled on demand.
    - This model worked well for most Websites.
    - However, the compilation process made the first request of any ASP.NET page slower than subsequent requests.
    - ASP.NET 2.0 continues to support this model of compilation.
  - Batch compilation
    - In ASP.NET 2.0, you can batch compile any application with a single URL request.
    - As with ASP.NET 1.x, batch compiling removes the delay on the first page request, but creates a longer cycle time on startup.
    - In addition, batch compilation still requires that the code behind files are compiled pre-deployment.
  - Deployment pre-compilation
    - A new feature of ASP.NET 2.0 allows for full compilation of your project prior to deployment.
    - In the full compilation, all of the code behind files, ASPX pages, HTML, graphics resources, and other back end code are compiled into one or more executable assemblies, depending on the size of the application and the compilation settings.
    - The assemblies contain all of the compiled code for the Website and the resource files and configuration files are copied without modification.
    - This compilation method provides for the greatest performance and enhanced security.
    - If you are working with highly visible or highly secure Websites, this option is the best choice for final deployment.

    - However, if you are building a small site running on your local intranet, and the site changes frequently, full pre-compilation may be over kill.
  - Full runtime compilation
    - At the other extreme of deployment pre-compilation, ASP.NET 2.0 provides a new mechanism to compile the entire application at runtime.
    - That is, you can put your un-compiled code behind files and any other associated code in the new \code directory and let ASP.NET 2.0 create and maintain references to the assembly that will be generated from these files at runtime.
    - This option provides the greatest flexibility in terms of changing Website content at the cost of storing un-compiled code on the server.
- ASP.NET 2.0 and Visual Studio 2005 also change many of the day to day aspects of Web application development.
- In this section, we will look at several of the areas where features available in ASP.NET 1.x have been heavily modified.

Development Tools

- IntelliSense
  - IntelliSense helps you develop code faster by issuing "popup" programming hints.
  - Every time you reference an object in your code, IntelliSense offers you a list of available methods, properties and events.
  - You no longer have to type out complete methods, or search documentation for parameter specifications.
  - Previous releases of Visual Studio.NET had excellent IntelliSense support, but the Visual Studio 2005 allows you to take advantage of IntelliSense in script blocks, inline CSS style attributes, configuration files and any XML file that contains a DTD or a XML schema references.
- Wizards to auto-generate code for many tasks
  - One of the major goals of ASP.NET 2.0 is to reduce development time and the number of lines of code you have to write.
  - Visual Studio 2005 includes wizards you can use to create data source connections, and many other common tasks.
- Designer support for master pages, themes and skins
  - Classic ASP pages required you to start each page from scratch.
  - Replicating page layout in an application was a time-consuming process.

  - Visual Studio 2005 provides full WYSIWYG support for ASP.NET master pages.
  - Master pages are page templates that can be used to apply a consistent style and layout to each page in your application.
  - After you create a master page, you can easily create new pages that leverage your master page, and let ASP.NET automatically apply the style and layout of the master page.
  - Visual Studio 2005 lets you see how any content you add to a page will look when combined with the master page.

- Improved code editing features
  - Editing code is not an easy task in traditional ASP applications.
  - Visual Studio 2005 makes code editing much easier by preserving all HTML code (including white space, casing, indention, carriage returns, and word wrapping).
  - Visual Studio 2005 allows you to set style formats for the HTML code for your application that can be imported or exported to standardize development for your whole team.
  - Style sheets allow you to format the way your content is displayed to a user.
  - Visual Studio.NET 2005 allows you to customize the format for how your HTML code is written.
  - For example, you can enforce style guidelines to ensure all HTML tags are capitalized, or that all nested HTML elements are indented two spaces.
  - These guidelines allow your entire development team to develop pages with the same formatting rules.
  - Visual Studio 2005 also supports tag-outline and tag-navigation modes to help you edit large documents.
  - These modes allow you to move through, expand, and collapse HTML and ASP.NET tags, and to see open and closed tag relationships clearly.
  - You can easily add and edit tables into HTML pages using the "Insert Table" dialog box.
  - The wizard lets you specify the style, size (table width and the number of rows and columns), and format of a table without writing any code.

- Visual Studio 2005 includes a light weight Web server
  - Visual Studio 2005 includes a light-weight Web server for developing ASP.NET applications.
  - The Web server services local requests only and is perfect for development.
  - By adding this Web server, you no longer need to use IIS on a development machine.
  - You can use the built-in server, and get full debugging support.

- Another important advantage is any user can build and test an application.
- You no longer need to be an administrator on the computer.
- It should be noted that you must still deploy your final applications on IIS.

- Improved debugging environment
  - Debugging applications in ASP is an arduous process, as code is interpreted on the fly.
  - Syntax errors aren't exposed until you run applications.
  - The debugging support is improved in Visual Studio 2005.
  - Visual Studio .NET 2003 provided limited support for edit-and-continue debugging.
  - Visual Studio 2005 improves edit-and-continue debugging and provides more information during the debugging process.
  - For example, an improved implementation of DataTips provides information about variables and expressions as mouse popup windows during the debugging process.
  - Visual Studio 2005 also provides additional features, like advanced debugging with breakpoints on disassembly, registers, addresses (when applicable), and "Just My Code Stepping".

Stages in Web forms Processing

Page Framework Initialiation

- The page's Page_Init event is raised, and the page and control view state are restored.

USES

- During this event, the ASP.NET page framework restores the control properties and postback data.
- This is the stage in which ASP.NET first creates the page.
- It generates all the controls you have defined with tags in the .aspx web page.
- In addition, if the page is not being requested for the first time (in other words, if it's a postback), ASP.NET deserializes the view state information and applies it to all the controls.
- At this stage, the Page.Init event fires.
- However, this event is rarely handled by the web page, because it's still too early to perform page initialization.

User Code Initialiation

- The page's Page_Load event is raised.
- At this stage of the processing, the Page.Load event is fired.
- Most web pages handle this event to perform any required initialization (such as filling in dynamic text or configuring controls).
- The Page.Load event always fires, regardless of whether the page is being requested for the first time or whether it is being requested as part of a postback.

USES

- Read and restore values stored previously:
    - Using the Page.IsPostBack property, check whether this is the first time the page is being processed.
    - If this is the first time the page is being processed, perform initial data binding.
    - Otherwise, restore control values.
    - Read and update control properties.

Validation

- The syntax for creating a Validation server control is:
    - < asp:control_name id="some_id" runat="server" / >

- The Validate method of any validator Web server controls is invoked to perform the control's specified validation.

Validation Server Control	Description
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value.
CustomValidator	Allows you to write a method to handle the validation of the value entered.
RangeValidator	Checks that the user enters a value that falls between two values.
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern.
RequiredFieldValidator	Makes an input control a required field.
ValidationSummary	Displays a report of all validation errors occurred in a Web page.

- ASP.NET includes validation controls that can automatically validate other user input controls and display error messages.
- These controls fire after the page is loaded but before any other events take place.

USES

- There is no user hook at this stage.
- You can test the outcome of validation in an event handler.

Event Handling

- The general syntax of an event is:
  - private void EventName (object sender, EventArgs e);
- If the page was called in response to a form event, the corresponding event handler in the page is called during this stage.

markup file.aspx

- When you use the data source controls, ASP.NET automatically performs updates and queries against your data source as part of the page life cycle.

USES

- Handle the specific event raised.
- If the page contains Types of Validation for ASP.NET Server Controls.
- Manually save the state of page variables that you are maintaining yourself.

- Check the IsValid property of the page or of individual validation controls.
- Manually save the state of controls dynamically added to the page.

Cleanup

- The Page_Unload event is called because the page has finished rendering and is ready to be discarded.
- At the end of its life cycle, the page is rendered to HTML.
- After the page has been rendered, the real cleanup begins, and the Page.Unload event is fired.
- At this point, the page objects are still available, but the final HTML is already rendered and can't be changed.
- Remember, the .NET Framework has a garbage collection service that runs periodically to release memory tied to objects that are no longer referenced.
- If you have any unmanaged resources to release, you should make sure you do this explicitly in the cleanup stage or, even better, before.
- When the garbage collector collects the page, the Page.Disposed event fires.
- This is the end of the road for the web page.

USES

- Perform final cleanup work:
  - Closing files.
  - Closing database connections.
  - Discarding objects.

Introduction to server controls

- It's important to understand how server controls operate and how they are completely different from the way you define controls in other languages like classic ASP or PHP — another popular programming language creating dynamic web sites.
- The text in a text box in these languages, you would use plain HTML and mix it with server-side code.
  - < input type="text" value="Hello World, the time is < %=Time()% >" / >
- As you can see, this code contains plain HTML, mixed with a server-side block, delimited by < % and % > that outputs the current time using the equals (=) symbol.
- This type of coding has a major disadvantage:
  - The HTML and server-side code is mixed, making it difficult to write and manage your pages.
- Although this is a trivial example in which it's still easy to understand the code, this type of programming can quickly result in very messy and complex pages.
- Server controls work differently.
- In ASP.NET, the controls "live" on the server inside an ASPX page.
- When the page is requested in the browser, the server-side controls are processed by the ASP.NET runtime — the engine that is responsible for receiving and processing requests for ASPX pages.
- The controls then emit client-side HTML code that is appended to the final page output.
- It's this HTML code that eventually ends up in the browser, where it's used to build up the page.
- Instead of defining HTML controls in your pages directly, you define an ASP.NET Server Control with the following syntax, where the italicized parts differ for each control.

Button1_Click

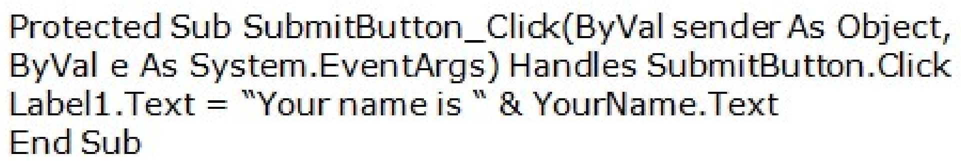
- < asp:TypeOfControl ID="ControlName" Runat="Server" / >
- Note that the control has two attributes: ID and Runat.
- The ID attribute is used to uniquely identify a control on the page, so you can program against it.
- It's important that each control on the page has a unique ID; otherwise the ASP.NET runtime won't understand what control you're referring to.
- If you accidentally type a duplicate control ID, VWD will signal the problem in the error list.
- The mandatory Runat attribute is used to indicate that this is a control that lives on the server.
- Without this attribute, the controls won't be processed and will end up directly in the HTML

source.

- If you ever feel you′re missing a control in the final output in the HTML of the browser, ensure that the control has this required attribute.
- Note that for non-server elements, like plain HTML elements, the Runat attribute is optional.
- With this attribute on non-server controls, they can be reached by your programming code.
- You can easily add the Runat attribute to an existing element by typing runat and pressing the Tab key.
- For the controls that ship with ASP.NET 4 you always use the asp: prefix followed by the name of the control.
  - < asp:TextBox ID="Message" Runat="Server" / >
- For example, to create a TextBox that can hold the same welcome message and current time,
  - < asp:TextBox ID="Message" Runat="Server" > < /asp:TextBox >
- The preceding example of the TextBox is using a self-closing tag where the closing slash (/) is embedded in the opening tag.
- VB.NET
  - Message.Text = "Hello World, the time is " & DateTime.Now.TimeOfDay.ToString()
- C#
  - Message.Text = "Hello World, the time is " + DateTime.Now.TimeOfDay.ToString();
- This is quite common for controls that don′t need to contain child content such as text or other controls.
- However, the long version, using a separate closing tag is acceptable as well:
- You can control the default behavior of closing tags per tag using Tools | Options | Text Editor | HTML | Formatting | Tag Specific Options.
- You can program against this text box from code that is either placed inline with the page or in a separate Code Behind file.
- The definition of the control in the markup section of the page is now separated from the actual code that defines the text displayed in the text box, making it easier to define and program the text box (or any other control) because it enables you to focus on one task at a time: either declaring the control and its visual appearance in the markup section of the page, or programming its behavior from a code block.
- You add a TextBox, a Label, and a Button control to a page.
- When you request the page in the browser, these server controls are transformed into HTML, which is then sent to the client.

- By looking at the final HTML for the page in the browser, you'll see how the HTML is completely different from the initial ASP.NET markup.
- In the Demos folder in the Solution Explorer, create a new Web Form called ControlsDemo.aspx.
- Choose your programming language and make sure the Web Form uses Code Behind.
- Switch to Design View.
- From theToolbox, drag a TextBox, a Button, and a Label control onto the design surface within the dashed lines of the < div > tag that was added for you when you created the page.
- Type the text Your name in front of the TextBox and add a line break between the Button and the Label by positioning your cursor between the two controls in Design View and then pressing Enter.
- If you're having trouble positioning the cursor between the controls, place it after the Label control and then press the left arrow key twice.
- The first time you press it, the Label will be selected; the second time, the cursor is placed between the two controls, enabling you to press Enter.
- Right-click the Button and choose Properties to open up the Properties Grid for the control.
- Pressing F4 after selecting the Button does the same thing.
- The window that appears, enables you to change the properties for the control, which in turn influence the way the control behaves at runtime.
- Set the control's Text property to Submit Information and set its ID to SubmitButton.
- Change the ID of the TextBox to YourName using the Properties Grid.
- Clear the Text property of the Label using the Properties Grid.
- You can right-click the property's label in the grid and choose Reset, or you can manually remove the text.
- You can leave its ID set to Label1.
- Still in Design View, double-click the button to have VWD add some code to the Code Behind of the page that will be fired when the button is clicked in the browser.
- Add the bolded line of code to the code block that VWD inserted for you:

  - VB .NET



```
Protected Sub SubmitButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles SubmitButton.Click
Label1.Text = "Your name is " & YourName.Text
End Sub
```

  - C#

```
protected void SubmitButton_Click(object sender, EventArgs e)
{
  Label1.Text = "Your name is " + YourName.Text;
}
```

- Save the changes to the page and then open it in the browser by pressing Ctrl+F5.
- When it appears in the browser, don't click the button yet, but open up the source of the page by right-clicking the page in the browser and choosing View Source or View Page Source.
- You should see the following HTML code.

```
<div>
 Your name <input name="YourName" type="text" id="YourName" />
 <input type="submit" name="SubmitButton" value="Submit Information"
id="SubmitButton" />
 <br />
 <span id="Label1"></span>
</div>
<div>
 Your name <input name="YourName" type="text" value="Imar" id="YourName" />
 <input type="submit" name="SubmitButton" value="Submit Information"
                  id="SubmitButton" />
 <br />
 <span id="Label1">Your name is Imar</span>
</div>
```

Switch back to your browser, fill in your name in the text box, and click the button.

- When the page is done reloading, open up the source for the page in the browser again using the browser's right-click menu.
- Label1.Text = "Your name is " + YourName.Text;
- When you click the button, the control causes a postback, which sends the information for the controls in the page to the server, where the page is loaded again.
- Additionally, the code that you wrote to handle the button's Click event is executed.
- This code takes the name you entered in the text box and then assigns it to the Labelcontrol:
  - < span id="Label1" > Your name is Imar< /span >
- The Label control contains the text you entered in the text box, so when it is asked for its HTML, it now returns this:
- Although you defined the Label control with the < asp:Label > syntax, it ends up as a simple < span > element in the browser.
- Because the Text property of the Label control is empty, you don't see any text between the two < span > tags.
- The same applies to other controls; an < asp:TextBox > ends up as < input type="text" >,

whereas the< asp:Button > ends up as < input type="submit" >.

- Simply drag controls from the Toolbox onto the design surface of the page.
- This makes it very easy to add a bunch of controls to a page to get you started.
- However, because of the way the design surface works, it's sometimes difficult to add them exactly where you want them.
- If you look at the Properties Grid for some of the controls in a page, you'll notice that many of them have similar properties.

Properties for all controls

- AccessKey
  - ➢ Enables you to set a key with which a control can be accessed at the client by pressing the associated letter.
- BackColorForeColor
  - ➢ Enables you to change the color of the background (BackColor) and text (ForeColor) of the control.
- BorderColorBorderStyleBorderWidth
  - ➢ Changes the border of the control in the browser.
  - ➢ The similarities with the CSS border properties you saw in the previous chapter are no coincidence.
  - ➢ Each of these three ASP.NET properties maps directly to its CSS counterpart.
- CssClass
  - ➢ Lets you define the HTML class attribute for the control in the browser.
  - ➢ This class name then points to a CSS class you defined in the page or an external CSS file.
- Enabled
  - ➢ Determines whether the user can interact with the control in the browser.
  - ➢ For example, with a disabled text box (Enabled="False") you cannot change its text.
- Font
  - ➢ Enables you to define different font-related settings, such as Font-Size, Font-Names, and Font-Bold.
- HeightWidth
  - ➢ Determines the height and width of the control in the browser.

- TabIndex
  - Sets the client-side HTML tabindex attribute that determines the order in which users can move through the controls in the page by pressing the Tab key.
- ToolTip
  - Enables you to set a tooltip for the control in the browser.
  - This tooltip, rendered as a title attribute in the HTML, is shown when the user hovers the mouse over the relevant HTML element.
- Visible
  - Determines whether or not the control is sent to the browser.
  - You should really see this as a server-side visibility setting because an invisible control is never sent to the browser at all.
  - This means it's quite different from the CSS display and visibility properties you saw in the previous chapter that hide the element at the client.

Types of controls

- ASP.NET comes with a large number of server controls, supporting most of your web development needs.
- To make it easy for you to find the right controls, they have been placed in separate control categories in the Visual Web Developer Toolbox.
- For example, a TextBox has a Text property (among many others), and aListBox has a SelectedItem property.
- Some properties can only be set programmatically and not with the Properties Grid.

Standard Controls

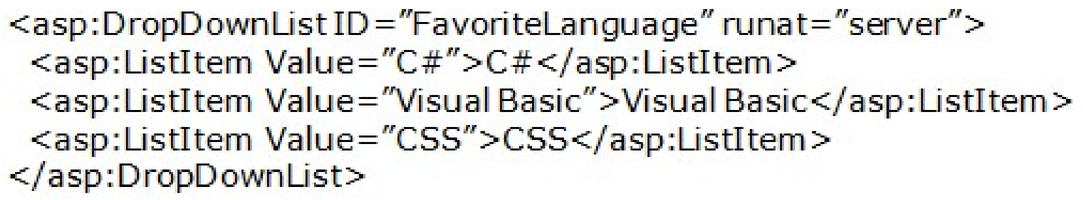
- The Standard category contains many of the basic controls that almost any web page needs.
- You've already seen some of them, like the TextBox, Button, and Labelcontrols earlier in this chapter.
- Many of the controls probably speak for themselves, so instead of giving you a detailed description of them all, the following sections briefly highlight a few important ones.

Simple Controls

- The Toolbox contains a number of simple and straightforward controls, including TextBox, Button, Label, HyperLink, RadioButton, and CheckBox.
- Their icons in the Toolbox give you a good clue as to how they end up in the browser.

List Controls

- The standard category also contains a number of controls that present themselves as lists in the browser.
- These controls include ListBox, DropDownList, CheckBoxList,RadioButtonList, and BulletedList.
- To add items to the list, you define < asp:ListItem > elements between the opening and closing tags of the control.



```
<asp:DropDownList ID="FavoriteLanguage" runat="server">
  <asp:ListItem Value="C#">C#</asp:ListItem>
  <asp:ListItem Value="Visual Basic">Visual Basic</asp:ListItem>
  <asp:ListItem Value="CSS">CSS</asp:ListItem>
</asp:DropDownList>
```

- The DropDownList enables a user to select only one item at a time.
- To see the currently active and selected item of a list control programmatically, you can look at its SelectedValue, SelectedItem, or SelectedIndex properties.
- SelectedValue returns a string that contains the value for the selected item, like C# or Visual Basic in the preceding example.
- SelectedIndex returns the zero-based index of the item in the list.
- With the preceding example, if the user had chosen C#, SelectedIndex would be 0.
- Similarly, when the user has chosen CSS, the index would be 2 (the third item in the list).
- The BulletedList control doesn't allow a user to make selections, and as such doesn't support these properties.
- For controls that allow multiple selections (like CheckBoxList and ListBox), you can loop through the Items collection and see what items are selected.
- In this case,SelectedItem returns only the first selected item in the list; not all of them.

Container Controls

- This grouping can be done by putting the controls in one of the container controls, like the Panel, the PlaceHolder, the MultiView, or the Wizard.
- For example, you can use the PlaceHolder or the Panel control to hide or show a number of controls at once.
- Instead of hiding each control separately, you simply hide the entire container that contains all the individual controls and markup.

LinkButton and ImageButton

The LinkButton and the ImageButton controls operate similarly to an ordinary Button control.

- - Both of them cause a postback to the server when they are clicked.
- TheLinkButton presents itself as a simple < a > element but posts back (using JavaScript) instead of requesting a new page.
- The ImageButton does the same, but displays an image that the user can click to trigger the postback.

Image and ImageMap

- These controls are pretty similar in that they display an image in the browser.
- The ImageMap enables you to define hotspots on the image that when clicked either cause a postback to the server or navigate to a different page.

Calendar

- The Calendar control presents a rich interface that enables a user to select a date.

FileUpload

- The FileUpload control enables a user to upload files that can be stored on the server.

Literal, Localize, and Substitute

- The Literal is that it renders no additional tag itself; it displays only what you assign to its Text property and is thus very useful to display HTML or JavaScript that you build up in the Code Behind or that you retrieve from a database.
- The Localize control is used in multilingual web sites and is able to retrieve its contents from translated resource files.
- The Substitute control is used in advanced caching scenarios and enables you to update only parts of a page that is otherwise cached completely.

AdRotator

- The AdRotator control enables you to display random advertisements on your web site.
- The ads come from an XML file that you create on your server.
- Because it lacks advanced features like click tracking and logging that are required in most but the simplest scenarios, this control isn't used much in today's web sites.

HiddenField

- The HiddenField control enables you to store data in the page that is submitted with each request.
- This is useful if you want the page to remember specific data without the user seeing it on the page.

XML

- The XML control enables you to transform data from an XML format to another format (like XHTML) for display on a page.

Table

- The < asp:Table > control is in many respects identical to its HTML < table > counterpart.
- However, because the control lives at the server, you can program against it, dynamically creating new columns and rows and adding dynamic data to it.

HTML Control

- The HTML category of the Toolbox contains a number of HTML controls that look similar to the ones found in the Standard category.
- For example, you find the Input (Button) that looks like the < asp:Button >.
- Similarly, there is a Select control that has the < asp:DropDownList > and < asp:ListBox > as its counterparts.
- The HTML server controls are basically the original HTML controls but enhanced to enable server side processing.
- The HTML controls like the header tags, anchor tags and input elements are not processed by the server but sent to the browser for display.
- They are specifically converted to a server control by adding the attribute runat="server" and adding an id attribute to make them available for server-side processing.
- For example, consider the HTML input control:
  - < input type="text" size="40" >
- It could be converted to a server control, by adding the runat and id attribute:
  - < input type="text" id="testtext" size="40" runat="server" >

Advantages of using HTML Server Controls

Control Name	HTML tag
HtmlHead	< head >element
HtmlInputButton	< input type=button|submit|reset >
HtmlInputCheckbox	< input type=checkbox >
HtmlInputFile	< input type = file >
HtmlInputHidden	< input type = hidden >
HtmlInputImage	< input type = image >
HtmlInputPassword	< input type = password >
HtmlInputRadioButton	< input type = radio >
HtmlInputReset	< input type = reset >
HtmlText	< input type = text|password >

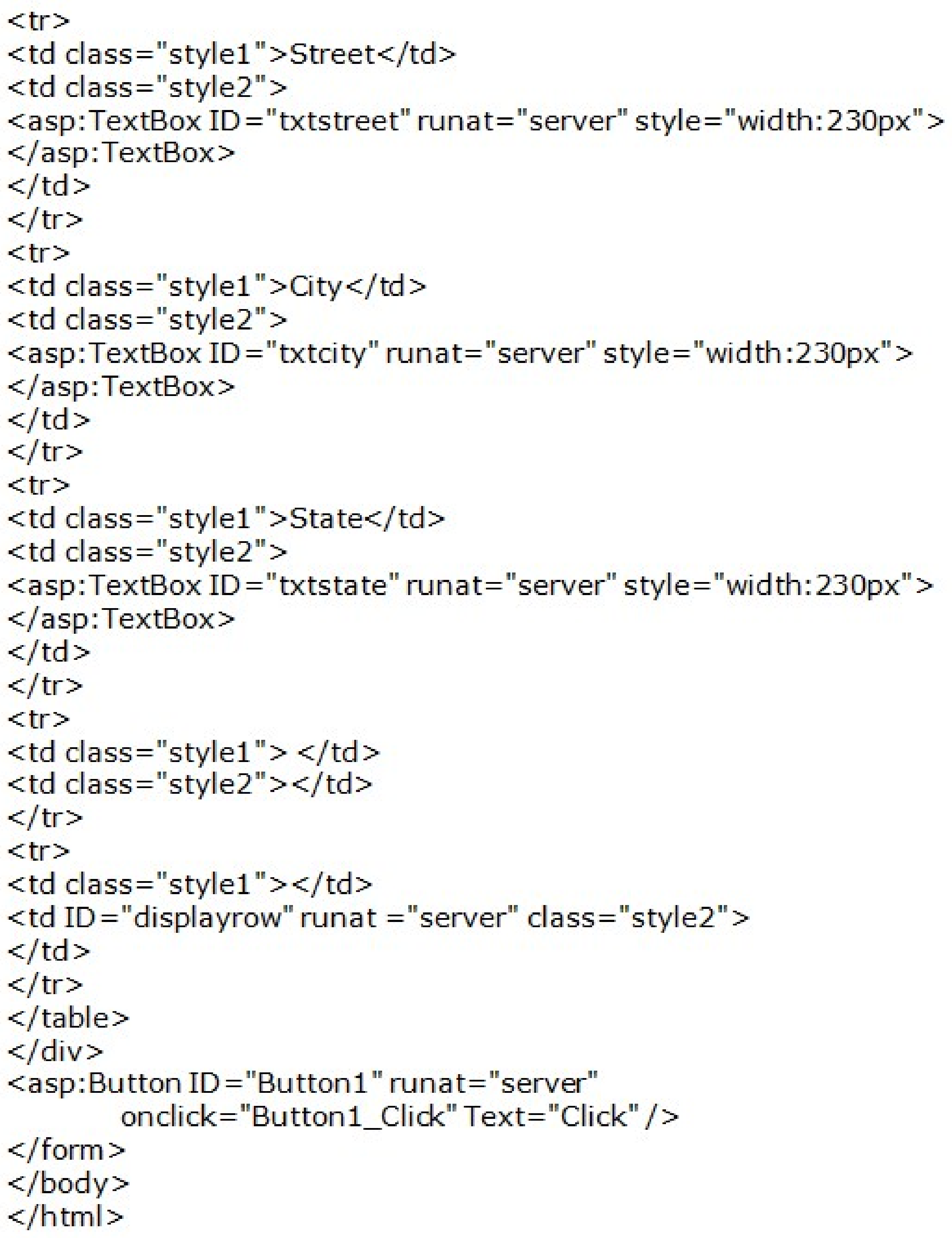
HtmlImage	< img > element
HtmlLink	< link > element
HtmlAnchor	< a > element
HtmlButton	< button > element
HtmlForm	< form > element
HtmlTable	< table > element
HtmlTableCell	< td > and
HtmlTableRow	< tr > element
HtmlTitle	< title > element
HtmlSelect	< select > element
HtmlGenericControl	All HTML controls not listed

- Although ASP.Net server controls can perform every job accomplished by the HTML server controls, the later controls are useful in the following cases:
    - Using static tables for layout purposes.
    - Converting a HTML page to run under ASP.Net.

Example for HTML control

- The following example uses a basic HTML table for layout.

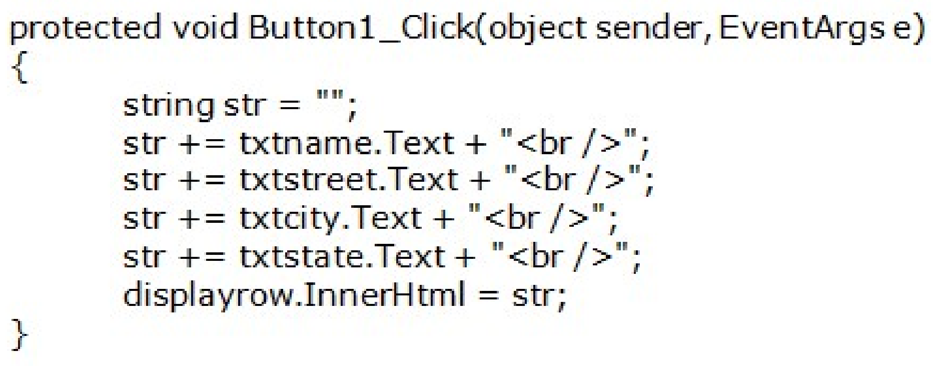
```
<%@ Page Language="C#" AutoEventWireup="true"
                      CodeBehind="Default.aspx.cs"
                      Inherits="htmlserver._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
<style type="text/css">
  .style1
  {
     width: 156px;
  }
  .style2
  {
    width: 332px;
  }
</style>
</head>
<body>
<form id="form1" runat="server">
<div>
<table style="width: 54%;">
<tr>
<td class="style1">Name:</td>
<td class="style2">
<asp:TextBox ID="txtname" runat="server" style="width:230px">
</asp:TextBox>
</td>
</tr>
```



```
<tr>
<td class="style1">Street</td>
<td class="style2">
<asp:TextBox ID="txtstreet" runat="server" style="width:230px">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="style1">City</td>
<td class="style2">
<asp:TextBox ID="txtcity" runat="server" style="width:230px">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="style1">State</td>
<td class="style2">
<asp:TextBox ID="txtstate" runat="server" style="width:230px">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="style1"> </td>
<td class="style2"></td>
</tr>
<tr>
<td class="style1"></td>
<td ID="displayrow" runat ="server" class="style2">
</td>
</tr>
</table>
</div>
<asp:Button ID="Button1" runat="server"
         onclick="Button1_Click" Text="Click" />
</form>
</body>
</html>
```

- It uses some text boxes for getting input from the users like, name, address, city, state etc.
- It also has a button control, which is clicked to get the user data displayed on the last row of the table.

The code behind the button control



```
protected void Button1_Click(object sender, EventArgs e)
{
        string str = "";
        str += txtname.Text + "<br />";
        str += txtstreet.Text + "<br />";
        str += txtcity.Text + "<br />";
        str += txtstate.Text + "<br />";
        displayrow.InnerHtml = str;
}
```

- Observe the followings
  - The normal HTML tags has been used for the page layout.
  - The last row of the HTML table is used for data display.
  - It needed server side processing, so an ID attribute and the runat attribute has been added to it.

Validation Control

- ASP.Net validation controls validate the user input data to ensure that useless, unauthenticated or contradictory data don.t get stored.
- ASP.Net provides the following validation controls:
  - BaseValidator,
  - RequiredFieldValidator,
  - RangeValidator,
  - CompareValidator,
  - RegularExpressionValidator,
  - CustomValidator,
  - ValidationSummary.

The BaseValidator Class

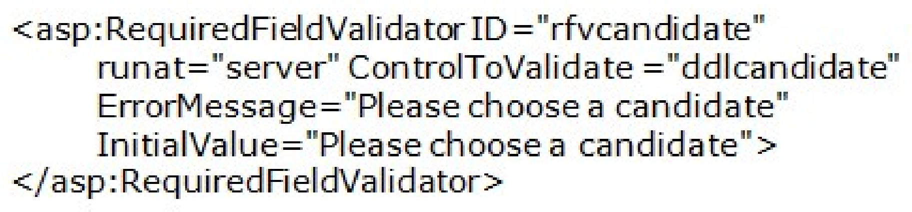
- The validation control classes inherit from the BaseValidator class and inherit its properties and methods.
- Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

- The table show the common validation controls.

The RequiredFieldValidator

- The RequiredFieldValidator control ensures that the required field is not empty.
- It is generally tied to a text box to force input into the text box.
- It is useful to validate whether the user filled the required text box or not, if not the error message would be display.
- The syntax for the control:



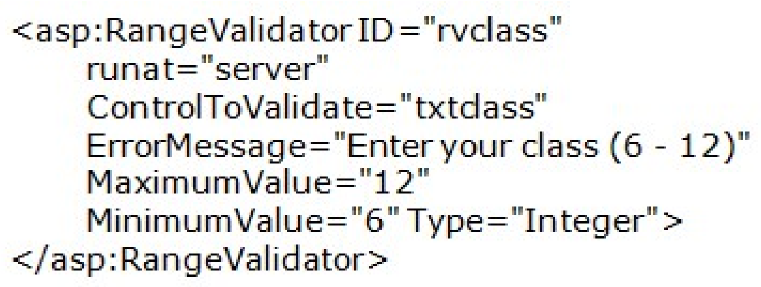
```
<asp:RequiredFieldValidator ID="rfvcandidate"
      runat="server" ControlToValidate ="ddlcandidate"
      ErrorMessage="Please choose a candidate"
      InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

The RangeValidator

- The RangeValidator control verifies that the input value falls within a predetermined range.
- It has three specific properties:

Properties	Description
Types	it defines the type of the data; the available values are: Currency, Date, Double, Integer and String
MinimumValue	it specifies the minimum value of the range
MaximumValue	it specifies the maximum value of the range

- The basic example for the control:



```
<asp:RangeValidator ID="rvclass"
     runat="server"
     ControlToValidate="txtclass"
     ErrorMessage="Enter your class (6 - 12)"
     MaximumValue="12"
     MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

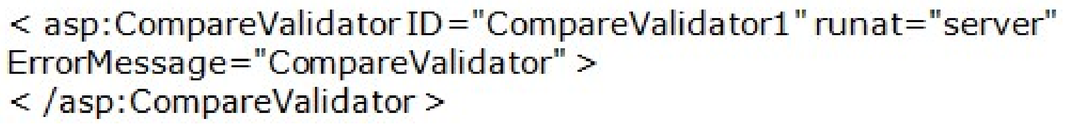
The CompareValidator

- The CompareValidator control compares a value in one control with a fixed value, or, a value in another control.

- It has the following specific properties:

Properties	Description
Type	it specifies the data type
ControlToCompare	it specifies the value of the input control to compare with
ValueToCompare	it specifies the constant value to compare with
Operator	it specifies the comparison operator, the available values are:Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan,LessThanEqual and DataTypeCheck

- The basic syntax for the control:



```
< asp:CompareValidator ID="CompareValidator1" runat="server"
ErrorMessage="CompareValidator" >
< /asp:CompareValidator >
```

The RegularExpressionValidator

- The RegularExpressionValidator allows validating the input text by matching against a pattern against a regular expression.
- The regular expression is set in the ValidationExpression property.

Metacharacters	Description
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

- The regular validation used for input text matching.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

- It matches the input text field with onther text input filed.
- The syntax for the control:

```
< asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
ValidationExpression="string" ValidationGroup="string" >
< /asp:RegularExpressionValidator >
```

The CustomValidator

- The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.
- The client side validation is accomplished through the ClientValidationFunction property.
- The client side validation routine should be written in a scripting language, like JavaScript or VBScript, which the browser can understand.
- The basic syntax for the control:

```
< asp:CustomValidator ID="CustomValidator1" runat="server"
ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator" >
< /asp:CustomValidator >
```

User Controls

- In simple modification we can use the server, the reusable model is called as user control.

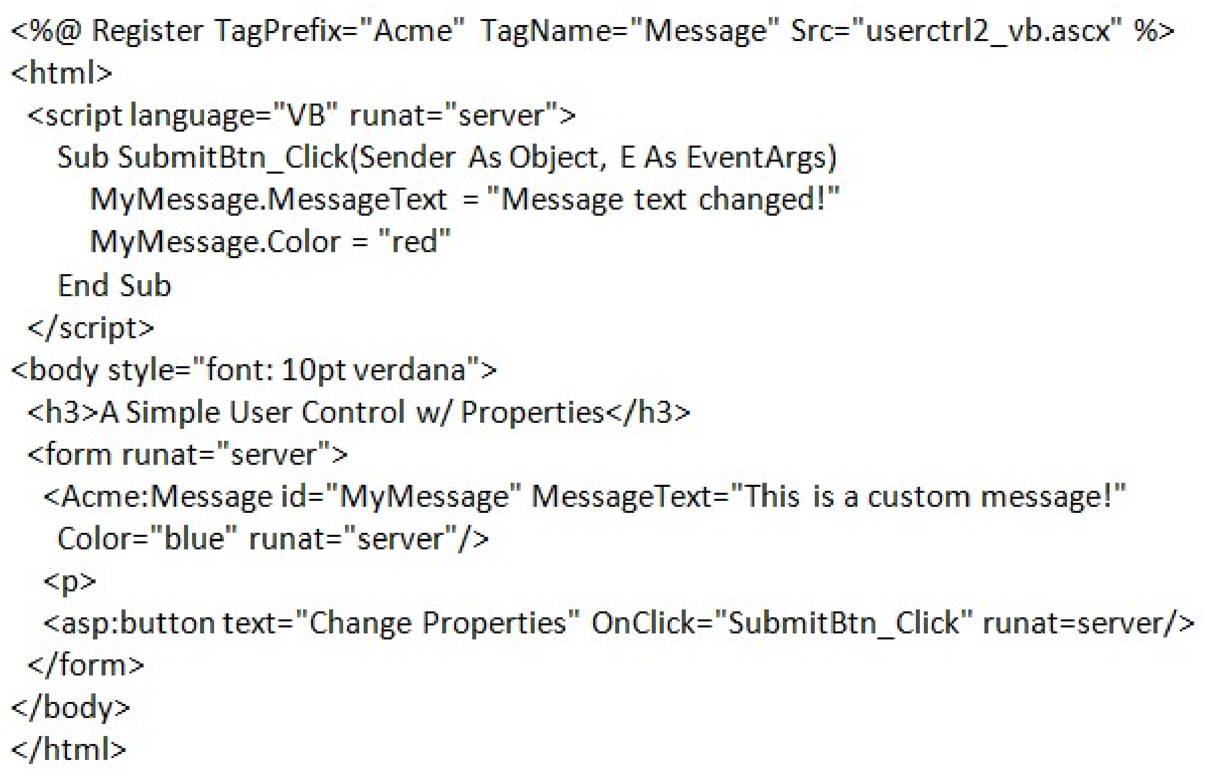
```
< asp:ValidationSummary ID="ValidationSummary1" runat="server"
DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" / >
```

- User customize the program using small changes.

Exposing User Control Properties

When a Web Forms page is treated as a control, the public fields and methods of that Web

- Form are promoted to public properties and methods of the control as well.
- The following example shows an extension of the previous user control example that adds two public String fields.



```
<%@ Register TagPrefix="Acme"  TagName="Message" Src="userctrl2_vb.ascx" %>
<html>
 <script language="VB" runat="server">
    Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
       MyMessage.MessageText = "Message text changed!"
       MyMessage.Color = "red"
    End Sub
 </script>
<body style="font: 10pt verdana">
 <h3>A Simple User Control w/ Properties</h3>
 <form runat="server">
  <Acme:Message id="MyMessage" MessageText="This is a custom message!"
    Color="blue" runat="server"/>
  <p>
  <asp:button text="Change Properties" OnClick="SubmitBtn_Click" runat=server/>
 </form>
</body>
</html>
```

  - The public properties and methods are treated as public fields when it promoted to public control.
  - Notice that these fields can be set either declaratively or programmatically in the containing page.

User Controls

- User controls behaves like miniature ASP.Net pages, or web forms, which could be used by many other pages.
- These are derived from the System.Web.UI.UserControl class. These controls have the following characteristics.
  - They have an .ascx extension.
  - They may not contain any < html >, < body > or < form > tags.
  - They have a Control directive instead of a Page directive.
  - To understand the concept let us create a simple user control, which will work as footer for the web pages.
  - To create and use the user control, take the following steps:
    - Create a new web application.

- Right click on the project folder on the Solution Explorer and choose Add New Item.

Data binding Control

- Every ASP.Net web form control inherits the DataBind method from its parent Control class, which gives it an inherent capability to bind data to at least one of its properties.
- This is known as simple data binding or inline data binding.
- Simple data binding involves attaching any collection (item collection) which implements the IEnumerable interface, or the DataSet and DataTable classes to the DataSource property of the control.
- On the other hand, some controls can bind records, lists, or columns of data into their structure through a DataSource control.
- These controls derive from the BaseDataBoundControl class.
- This is called declarative data binding.
- The data source controls help the data-bound controls implement functionalities like, sorting, paging and editing data collections.
- The BaseDataBoundControl is an abstract class, which is inherited by two more abstract classes:
  - DataBoundControl.
  - HierarchicalDataBoundControl.
- The abstract class DataBoundControl is again inherited by two more abstract classes:
  - ListControl.
  - CompositeDataBoundControl.
- The controls capable of simple data binding are derived from the ListControl abstract class and these controls are:
  - BulletedList.
  - CheckBoxList.
  - DropDownList.
  - ListBox.
  - RadioButtonList.
- The controls capable of declarative data binding are derived from the abstract class CompositeDataBoundControl.
- These controls are:
  - DetailsView.

- FormView.
- GridView.
- RecordList.

Simple Data Binding

- Simple data binding involves the read-only selection lists.
- These controls can bind to an array list or fields from a database.
- Selection lists takes two values from the database or the data source; one value is displayed by the list and the other is considered as the value corresponding to the display.
- Choosing a data source for the bulleted list control involves:
    - Selecting the data source control.
    - Selecting a field to display, which is called the data field.
    - Selecting a field for the value.

Declarative Data Binding

- We have already used declarative data binding in the previous tutorial using GridView control.
- The other composite data bound controls capable of displaying and manipulating data in a tabular manner are the DetailsView, FormView and RecordList control.
- However, the data binding involves the following objects:
    - A dataset that stores the data retrieved from the database.
    - The data provider, which retrieves data from the database, using a command over a connection.
    - The data adapter that issues the select statement stored in the command object; it is also capable of update the data in a database by issuing Insert, Delete, and Update statements.

Configuration

- The behavior of an ASP.NET application is affected by different settings in the configuration files:
  - machine.config,
  - web.config.
- The machine.config file contains default and the machine-specific value for all supported settings.
- The machine settings are controlled by the system administrator and applications are generally not given access to this file.
- An application however, can override the default values by creating web.config files in its roots folder.
- The web.config file is a subset of the machine.config file.
- If the application contains child directories, it can define a web.config file for each folder.
- Scope of each configuration file is determined in a hierarchical top-down manner.
- Any web.config file can locally extend, restrict, or override any settings defined on the upper level.
- Visual Studio generates a default web.config file for each project.
- An application can execute without a web.config file, however, you cannot debug an application without a web.config file.
- In this application, there are two web.config files for two projects i.e., the web service and the web site calling the web service.
- The following code snippet shows the basic syntax of a configuration file:
- The web.config file has the configuration element as the root node.
- Information inside this element is grouped into two main areas: the configuration section-handler declaration area, and the configuration section settings area.

Configuration Section Handler

- The configuration section handlers are contained within the < configSections > tags.
- Each configuration handler specifies name of a configuration section, contained within the file, which provides some configuration data.
- Clear
  - It removes all references to inherited sections and section groups.

  Remove

- ➢ It removes a reference to an inherited section and section group.
- Section
  - ➢ It defines an association between a configuration section handler and a configuration element.
- Section group
  - ➢ It defines an association between a configuration section handler and a configuration section.
- The basic syntax for the configuration:
- Application settings
  - ➢ The application settings allow storing application-wide name-value pairs for read-only access.
- The connection strings
  - ➢ The connection strings shows which database connection strings are available to the website.
- System.web Element
  - ➢ The system.web element specifies the root element for the ASP.NET configuration section and contains configuration elements that configure ASP.NET Web applications and control how the applications behave.
  - ➢ It holds most of the configuration elements needed to be adjusted in common applications.

Anonymous Identification

- Authentication
  - ➢ This is required to identify users who are not authenticated when authorization is required.
- Authorization
  - ➢ It configures the authentication support.
- UrlMappings
  - ➢ Defines the mappings for hiding the actual URL and providing a more user friendly URL.

Personalization

- ASP.Net personalization service is based on user profile.
- User profile defines the kind of information about the user that the site will need, for example, name, age, address, date of birth, phone number etc.
- Web sites are designed for repeated visits from the users.
- Personalization allows a site to remember its user and his/her information details and presents an individualistic environment to each user.
- ASP.Net provides services for personalizing a web site to suit a particular client's taste and preference.
- This information is defined in the web.config file of the application and ASP.Net runtime reads and uses it.
- This job is done by the personalization providers.
- The user profiles obtained from user data is stored in a default database created by ASP.Net.
- You can create your own database for storing profiles.
- The profile data definition is stored in the configuration file web.config.

Example

- Let us create a sample site, where we want our application to remember user details like name, address, date of birth etc.
- Add the profile details in the web.config file within the < system.web > element.
- When the profile is defined in the web.config file, the profile could be used through the Profile property found in the current HttpContext and also available via page.
- Add the text boxes to take the user input as defined in the profile and add a button for submitting the data:

Attributes for the < add > Element

- Apart from the name and type attributes that we have used, there are other attributes to the < add > element.
- Following table illustrates some of these attributes:

Attributes	Description
name	The name of the property.
type	By default the type is string but it allows any fully qualified class name as data type.

serializeAs	The format to use when serializing this value.
readOnly	A read only profile value cannot be changed, by default this property is false.
defaultValue	A default value that is used if the profile does not exist or does not have information.
allowAnonymous	A Boolean value indicating whether this property can be used with the anonymous profiles.
Provider	The profiles provider that should be used to manage just this property.

Session State

- Session and application variables are an important part of ASP programming.
- Whether it's developing simple hit counters or developing advanced chat servers, session and application variables are a must have feature of many applications.
    - A web server (IIS or PWS) capable of running ASP scripts,
    - Basic knowledge of ASP.

Session Object

- The session object is used to store variables for each specific visitor to your web site.
- These variables could represent anything from how many pages the user has viewed to their login details.
- A session is defined as the time from when the user visits your site to the time when he leaves – session variables die after a visitor closes their web browser (i.e. they are not persistent).
- Variables stored in the session object are available to all ASP pages on that particular web site.
- Common information stored in session variables includes name, id, and personalization preferences.
- The server creates a new session object for each new user and destroys the session object when the session expires.
- A session object is specific for every user and varies from user to user.
- IIS will maintain these variables when the client moves across pages within the site.

Properties:

- SessionID:
    - A long number that returns the session identifier for this client.
- Timeout:
    - An integer that specifies a timeout period in minutes.
    - If the client doesn't refresh or request any page from your site within this timeout period then the server ends the current session.
    - If you do not specify any timeout period then the default timeout period of 20 minutes is used.

Methods:

- Abandon:

    Destroys the current session object and releases its resources, meaning that if the client

  - requests any page from your site after the Session.Abandon method has been called then a new session will be started.

- Session_OnEnd:
  - This event occurs when the session is abandoned or times out for a specific user.
- Session_OnStart:
  - Occurs when a new session is started.
  - All ASP objects are available for you to use.
  - You can define your session wide variables here.

Example:

- You can store any value you like in the session object.
- Information stored in the session object is available for the entire session and has "session scope".
- The following script demonstrates how two types of variables are stored:
  - Session("username") = "Janine"
  - Session("age") = 42.

Application Object

- The Application object is used to store variables and to access variables from any page.
- All users share one Application object.

Syntax:

- Application.method

Methods:

- Lock.
  - Prevents other users from changing the application objects properties.
- Unlock
  - Allows other users to change the application objects properties.

Events:

- Application_OnEnd
  - This routine is called when all user sessions expire and the application quits.

- This event lives in the global.asa file, which we will look at shortly.

- Application_Onstart
  - This routine is called when the application object is first references.
  - This event lives in the global.asa file as well.

The Global.asa File

- In the global.asa file you can tell the application and session objects what to do when the application/session starts and what to do when the application/session ends.
- The code for this is placed into event handlers.
- Types of Events:
  - Application_OnStart.
  - Session_OnStart.
  - Session_OnEnd.
  - Application_OnEnd.

Example:

- This event occurs when the FIRST user calls the first page from an ASP application.
- This event also occurs after the web server is restarted or after the global.asa file is edited.
- When this procedure is complete, the "Session_OnStart" procedure runs.
- This event occurs EVERY time a user visits your web site and requests the first page.
- This event occurs EVERY time a user ends a session.
- A user ends a session after a page has not been requested by the user for a specific amount of time.
- This event occurs after the LAST user has ended the session.
- Typically, this event occurs when a web server is stopped/restarted.
- This procedure is used to clean up settings after the application stops, such as deleting records or writing log information to text files.

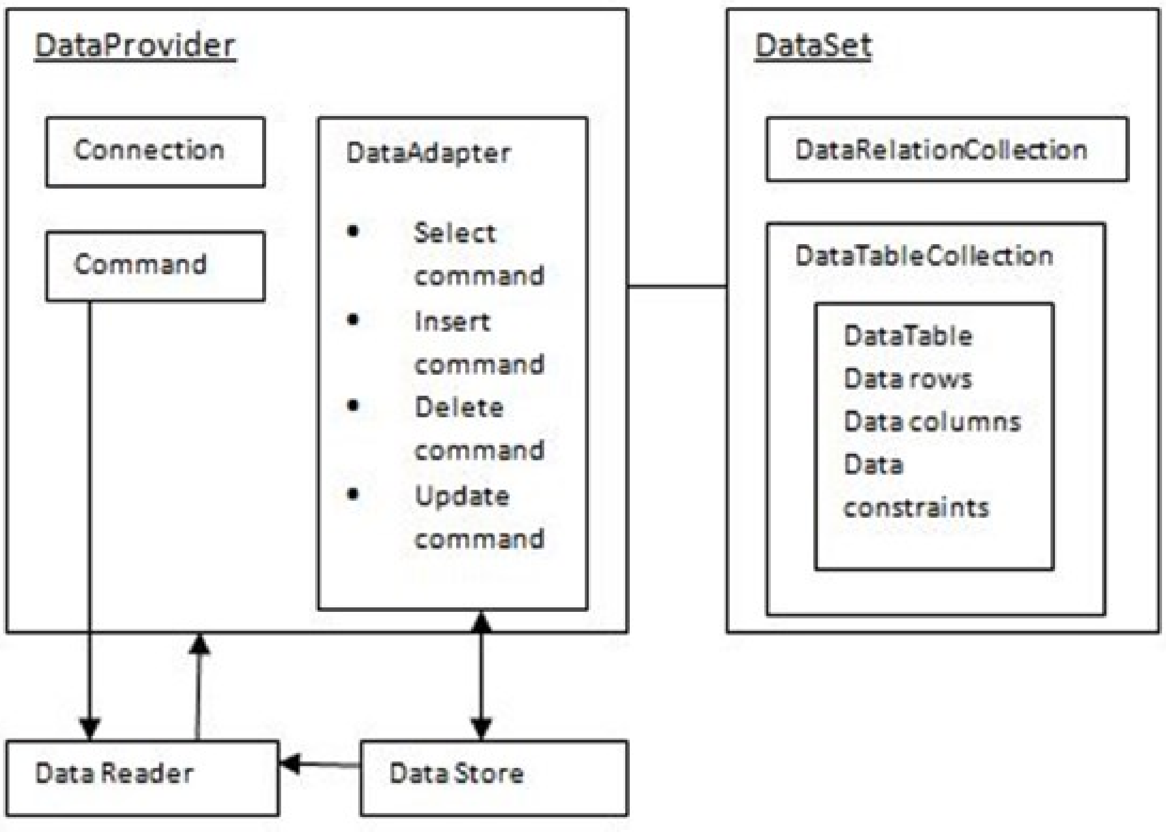
Maintaining Client State with Cookies

- If a visitor comes to your site and types his name into a form then you might want to remember that information.
- You may only want to remember it for his current visit (session), or for all subsequent visits for personalization (i.e. "Welcome back Mike").

- The term state describes all client browser data for the session.
- ASP uses client browser side cookies to remember (or persist) this data.
- Cookies are small text files stored on the visitors PC, usually in %root% \ Windows \ Temporary Internet Files \ Cookie:user_ name@Host_Name file.
- These text files are editable with notepad or Microsoft Word.
- Two types of information are stored in the cookie files:
  - Client Data.
  - Unique Cookie ID.
- Client Data:
  - The variables that make up the cookie for the visitor.
  - These are stored as name/value pairs, such as name=john.
- Unique Cookie ID:
  - This allows the ASP Session to identify the client browser on a page to page and visit to visit basis.

ADO .Net

- ADO.NET provides a bridge between the front end controls and the back end database.
- The ADO.NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.
- The following figure shows the ADO.NET objects at a glance:



The DataSet Class

- Following table shows some important properties of the DataSet class:

Properties	Description
CaseSensitive	Indicates whether string comparisons within the data tables are case-sensitive.
Container	Gets the container for the component.
DataSetName	Gets or sets the name of the current data set.
DefaultViewManager	Returns a view of data in the data set.
DesignMode	Indicates whether the component is currently in design mode.
EnforceConstraints	Indicates whether constraint rules are followed when attempting any update operation.

Events	Gets the list of event handlers that are attached to this component.
ExtendedProperties	Gets the collection of customized user information associated with the DataSet.
HasErrors	Indicates if there are any errors.
IsInitialized	Indicates whether the DataSet is initialized.
Locale	Gets or sets the locale information used to compare strings within the table.
Namespace	Gets or sets the namespace of the DataSet.
Prefix	Gets or sets an XML prefix that aliases the namespace of the DataSet.
Relations	Returns the collection of DataRelation objects.
Tables	Returns the collection of DataTable objects.

The DataTable Class

- The following table shows some important methods of the DataTable class:

Properties	Description
ChildRelations	Returns the collection of child relationship.
Columns	Returns the Columns collection.
Constraints	Returns the Constraints collection.
DataSet	Returns the parent DataSet.
DefaultView	Returns a view of the table.
ParentRelations	Returns the ParentRelations collection.
PrimaryKey	Gets or sets an array of columns as the primary key for the table.
Rows	Returns the Rows collection.

The DataRow Class

- The DataRow object represents a row in a table.
- It has the following important properties:

Properties	Description
HasErrors	Indicates if there are any errors.

Items	Gets or sets the data stored in a specific column.
ItemArrays	Gets or sets all the values for the row.
Table	Returns the parent table.

The DataAdapter Object

- The DataAdapter object acts as a mediator between the DataSet object and the database.
- This helps the data set to contain data from more than one database or other data source.

The DataReader Object

- The DataReader object is an alternative to the DataSet and DataAdapter combination.
- This object provides a connection oriented access to the data records in the database.

DbCommand and DbConnection Objects

- The DbConnection object represents a connection to the data source.
- The connection could be shared among different command objects.
- The DbCommand object represents the command or a stored procedure sent to the database from retrieving or manipulating data.

UNIT - 6
Introduction to Programming with Visual Basic.NET

Overview

- Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework.
- Although it is an evolution of classic Visual Basic language, it is not backwards-compatible with VB6, and any code written in the old version does not compile under VB.NET.
- Like all other .NET languages, VB.NET has complete support for object-oriented concepts.
- Everything in VB.NET is an object, including all of the primitive types (Short, Integer, Long, String, Boolean, etc.) and user-defined types, events, and even assemblies.
- All objects inherits from the base class Object.
- These are the following reasons make VB.Net a widely used professional language
  - Modern, general purpose.
  - Object oriented.
  - Component oriented.
  - Easy to learn.
  - Structured language.
  - It produces efficient programs.
  - It can be compiled on a variety of computer platforms.
  - Part of .Net Framework.
- VB.Net has numerous strong programming features that make it endearing to multitude of programmers worldwide.
- It has Strong Programming Features in VB.Net.
- Some of the features are
  - Boolean conditions,
  - Automatic Garbage Collection,
  - Standard Library,
  - Assembly Versioning,
  - Properties and Events,
  - Delegates and Events Management,
  - Easy-to-use Generics,

- Indexers,
- Conditional Compilation,
- Simple Multithreading.

Integrated Development Environment (IDE)

- The Visual Studio product family shares a single Integrated development environment (IDE) that is composed of several elements: the Menu bar, Standard toolbar, various tool windows docked or auto-hidden on the left, bottom, and right sides, as well as the editor space.
- The tool windows, menus, and toolbars available depend on the type of project or file you are working in.
  - Integrated Development Environment – allows the automation of many of the common programming tasks in one environment.
  - Writing the code.
  - Checking for Syntax (Language) errors.
  - Compiling and Interpreting(Transferring to computer language).
  - Debugging (Fixing Run-time or Logic Errors).
  - Running the Application.
- Microsoft provides the following development tools for VB.Net programming:
  - Visual Studio 2010 (VS).
  - Visual Basic 2010 Express (VBE).
  - Visual Web Developer.

Program Structure

- A VB.Net program basically consists of the following parts:
  - Namespace declaration.
  - A class or module.
  - One or more procedures.
  - Variables.
  - The Main procedure.
  - Statements & Expressions.
  - Comments.

- Let us look at a simple code that would print the words "Hello World":

```
Imports System

Module Module1

  'This program will display Hello World

  Sub Main()

    Console.WriteLine("Hello  World")

    Console.ReadKey()

  End Sub

End Module
```

- When the code is compiled and executed, it produces the following result:
  - Hello world
- VB .NET code can be structured as a module definition, form definition, or class definition.
  - Module definition begins with “Module” and ends with “End Module”.
  - Form definition is used to create a GUI.
  - Class definition is written to represent an object.
- Classes or Modules generally would contain more than one procedure.
- Procedures contain the executable code, or in other words, they define the behaviour of the class.
- A procedure could be any of the following:
  - Function.
  - Sub.
  - Operator.
  - Get.
  - Set.
  - AddHandler.
  - RemoveHandler.
  - RaiseEvent.

Writing a VB .NET Module Definition

- VB .NET code is not case sensitive.
- VB .NET compiler does not require indentation of code, but good programming practice encourages indentation.

AddHandler	AddressOf	Alias	And
AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar
CDate	CDbl	CDec	Char
CInt	Class Constraint	Class Statement	CLng
CObj	Const	Continue	CSByte
CShort	CSng	CStr	CType
CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate
Dim	DirectCast	Do	Double
Each	Else	ElseIf	End Statement
End < keyword >	EndIf	Enum	Erase
Error	Event	Exit	False
Finally	For (in For…Next)	For Each…Next	Friend
Function	Get	GetType	GetXMLNamespace
Global	GoSub	GoTo	Handles
If	If()	Implements	Implements Statement
Imports (.NET Namespace and Type)	Imports (XML Namespace)	In	In (Generic Modifier)
Inherits	Integer	Interface	Is

IsNot	Let	Lib	Like
Long	Loop	Me	Mod
Module	Module Statement	MustInherit	MustOverride
MyBase	MyClass	Namespace	Narrowing
New Constraint	New Operator	Next	Next (in Resume)
Not	Nothing	NotInheritable	NotOverridable
Object	Of	On	Operator
Option	Optional	Or	OrElse
Out (Generic Modifier)	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property
Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume
Return	SByte	Select	Set
Shadows	Shared	Short	Single
Static	Step	Stop	String
Structure Constraint	Structure Statement	Sub	SyncLock
Then	Throw	To	True
Try	TryCast	TypeOf…Is	UInteger
ULong	UShort	Using	Variant
Wend	When	While	Widening
With	WithEvents	WriteOnly	Xor
#Const	#Else	#ElseIf	#End
#If	=	&	&=
*	*=	/	/=
\	\=	^	^=

+	+=	-	-=
>> Operator (Visual Basic)	>>= Operator (Visual Basic)	<<	<<=

- The VB .NET statements consist of keywords and identifiers.
  - Keywords have special meanings to VB .NET.
  - VB .NET identifiers are the names assigned by the programmer to modules, procedures, and variables, etc.
- VB .NET identifiers:
  - Can be of any the alphabet length.
  - Can include any letter or number, but no spaces.
  - Must begin with a letter.
- Comment lines:
  - Add explanations to code.
  - Are ignored by compiler.
  - Begin with a single quote (′).
  - Many statements invoke a method to do some work.
  - Information sent to a method is called an argument.
  - A literal is a value defined within a statement.

Compile & Execute VB .net program

- This is the steps to compile and execute simple VB .Net program.
  - Start Visual Studio.
  - On the menu bar, choose File, New, Project.
  - Choose Visual Basic from templates.
  - Choose Console Application.
  - Specify a name and location for your project using the Browse button, and then choose the OK button.
  - The new project appears in Solution Explorer.
  - Write code in the Code Editor.

- Click the Run button or the F5 key to run the project.
- A Command Prompt window appears that contains the line Hello World.

VB .Net Basic Syntax

- VB.Net is an object oriented programming language.
- In Object Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions.
- The actions that an object may take are called methods.
- Objects of the same kind are said to have the same type or, more often, are said to be in the same class.
    - Object
        - Objects have states and behaviors.
        - Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating, etc.
        - An object is an instance of a class.
    - Class
        - A class can be defined as a template/blueprint that describe the behaviors/states that object of its type support.
    - Methods
        - A method is basically a behavior.
        - A class can contain many methods.
        - It is in methods where the logics are written, data is manipulated and all the actions are executed.
    - Instant Variables
        - Each object has its unique set of instant variables.
        - An object's state is created by the values assigned to these instant variables.
- Member Variables
    - Variables are attributes or data members of a class, used for storing data.
    - In the preceding program, the Rectangle class has two member variables named length and width.
- Member Functions
    - Functions are set of statements that perform a specific task.
    - The member functions of a class are declared within the class.

  - Our sample class Rectangle contains three member functions: AcceptDetails, GetArea and Display.

- Instantiating a class
  - In the preceding program, the class ExecuteRectangle is used as a class, which contains the Main() method and instantiates the Rectangle class.

Identifiers

- An identifier is a name used to identify a class, variable, function, or any other user-defined item.
- The basic rules for naming classes in VB.Net are as follows:
  - A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore.
  - The first character in an identifier cannot be a digit.
  - It must not contain any embedded space or symbol like ? - +! @ # % ^ & * ( ) [ ] { } . ; : " ' / and \.
  - However an underscore (_) can be used.
  - It should not be a reserved keyword.

Data Types

- Data types refer to an extensive system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.
- VB.Net provides a wide range of data types
  - Boolean.
  - Byte.
  - Char.
  - Date.
  - Decimal.
  - Double.
  - Integer.
  - Long.
  - Object.

- Sbyte.
- Short.
- Single.
- String.
- Uinteger.
- Ulong.
- User-defined.
- Ushort.

Variables

- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in VB.Net has a specific type, which determines the size and layout of the variable 's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.
- The basic value types provided in VB.Net can be categorized as:

Type	Example
Integral types	SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char
Floating point types	Single and Double
Decimal types	Decimal
Boolean types	True or False values, as assigned
Date types	Date

- VB.Net also allows defining other value types of variable like Enum and reference types of variables like Class.

Variable Declaration in VB .Net - Dim statement

- The Dim statement is used for variable declaration and storage allocation for one or more variables.
- The Dim statement is used at module, class, structure, procedure or block level.
- Syntax for variable declaration in VB.Net is:
  - [< attributelist >] [accessmodifier] [[Shared] [Shadows] | [Static]]
    [ReadOnly] Dim [WithEvents] variablelist
    - Attributelist is a list of attributes that apply to the variable. Optional.
    - Accessmodifier defines the access levels of the variables, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
    - Shared declares a shared variable, which is not associated with any specific instance of a class or structure, rather available to all the instances of the class or structure. Optional.
    - Shadows indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
    - Static indicates that the variable will retain its value, even when the after termination of the procedure in which it is declared. Optional.

- ReadOnly means the variable can be read, but not written. Optional.
- WithEvents specifies that the variable is used to respond to events raised by the instance assigned to the variable. Optional.
- Variablelist provides the list of variables declared.

Variable Declaration in VB .Net

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

Variablename: is the name of the variable.

- Boundslist: optional. It provides list of bounds of each dimension of an array variable.
- New: optional. It creates a new instance of the class when the Dim statement runs.
- Datatype: Required if Option Strict is On. It specifies the data type of the variable.
- Initializer: Optional if New is not specified. Expression that is evaluated and assigned to the variable when it is created.
- Some valid variable declarations along with their definition are shown here:

```
Dim StudentID As Integer

Dim StudentName As String

Dim Salary As Double

Dim count1, count2 As Integer

Dim status As Boolean

Dim exitButton As New System.Windows.Forms.Button

Dim lastTime, nextTime As Date
```

Variable Initialization in VB .Net

- Variables are initialized (assigned a value) with an equal sign followed by a constant expression.
- The general form of initialization is:
  - variable_name = value;
- For example
  - Dim pi As Double,
    pi = 3.14159.

- You can initialize a variable at the time of declaration as follows:

```
Dim StudentID As Integer = 100

Dim StudentName As String = "Bill Smith"

Example

Module variablesNdataypes

   Sub Main()

      Dim a As Short

      Dim b As Integer

      Dim c As Double

      a = 10

      b = 20

      c = a + b

      Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c)

      Console.ReadLine()

   End Sub

End Module
```

Accepting Values from User

- The Console class in the System namespace provides a function ReadLine for accepting input from the user and store it into a variable.
- For example,
  - Dim message As String
    message = Console.ReadLine
- The following example demonstrates it:

```
Module variablesNdataypes

  Sub Main()

    Dim message As String

    Console.Write("Enter message: ")

    message = Console.ReadLine

    Console.WriteLine()

    Console.WriteLine("Your Message: {0}", message)

    Console.ReadLine()

  End Sub

End Module
```

- When the above code is compiled and executed, it produces the following result (assume the user inputs Hello World):

```
Enter message: Hello World

Your Message: Hello World
```

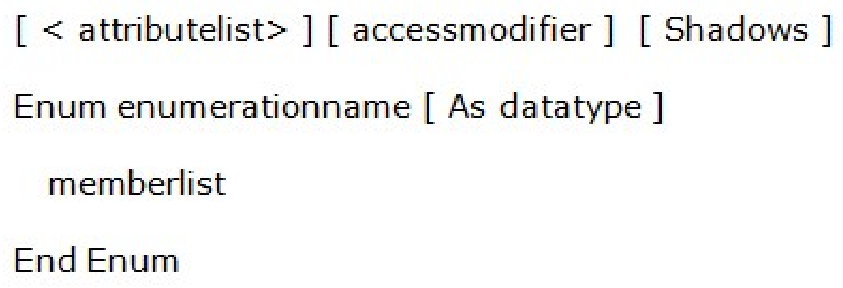
Constants

- The constants refer to fixed values that the program may not alter during its execution.
- These fixed values are also called literals.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.
- There are also enumeration constants as well.
- The constants are treated just like regular variables except that their values cannot be modified after their definition.
- In VB.Net, constants are declared using the Const statement.
- The Const statement is used at module, class, structure, procedure, or block level for use in place of literal values.
- The syntax for the Const statement is:
  - [< attributelist >] [accessmodifier] [Shadows]
    Const constantlist
  - attributelist: specifies the list of attributes applied to the constants; you can provide multiple attributes, separated by commas. Optional.
  - accessmodifier: specifies which code can access these constants. Optional.
  - Values can be either of the: Public, Protected, Friend, Protected Friend, or Private.
  - Shadows: this makes the constant hide a programming element of identical name, in a base class. Optional.
  - Constantlist: gives the list of names of constants declared. Required.
  - Where, each constant name has the following syntax and parts:
  - constantname [As datatype] = initializer.
    - constantname: specifies the name of the constant.
    - datatype: specifies the data type of the constant.
    - initializer: specifies the value assigned to the constant.

Enumerations

- An enumeration is a set of named integer constants.
- An enumerated type is declared using the Enum statement.
- The Enum statement declares an enumeration and defines the values of its members.

- The Enum statement can be used at the module, class, structure, procedure, or block level.
- The syntax for the Enum statement is as follows:



```
[ < attributelist> ] [ accessmodifier ]  [ Shadows ]
Enum enumerationname [ As datatype ]
  memberlist
End Enum
```

  - Attributelist: refers to the list of attributes applied to the variable. Optional.
  - Asscessmodifier: specifies which code can access these enumerations. Optional.
  - Values can be either of the: Public, Protected, Friend or Private.
  - Shadows: this makes the enumeration hide a programming element of identical name, in a base class. Optional.
  - Enumerationname: name of the enumeration. Required.
  - Datatype: specifies the data type of the enumeration and all its members.
  - Memberlist: specifies the list of member constants being declared in this statement. Required.
  - [< attribute list >] member name [= initializer]
  - Where,
    - name: specifies the name of the member. Required.
    - initializer: value assigned to the enumeration member. Optional.

Functions

- The Function statement is used to declare the name, parameter and the body of a function.
- The syntax for the Function statement is:

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
    [Statements]
End Function
```

- Modifiers:
  - Specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- FunctionName:
  - Indicates the name of the function.
- ParameterList:
  - Specifies the list of the parameters.
- ReturnType:
  - Specifies the data type of the variable the function returns.

Function Returning a value

- In VB.Net, a function can return a value to the calling code in two ways:
  - By using the return statement,
  - By assigning the value to the function name.
- The following example demonstrates using the FindMax function:

```
Module myfunctions
   Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
      ' local variable declaration */
      Dim result As Integer
      If (num1 > num2) Then
          result = num1
      Else
          result = num2
      End If
      FindMax = result
   End Function
   Sub Main()
      Dim a As Integer = 100
      Dim b As Integer = 200
      Dim res As Integer
        res = FindMax(a, b)
        Console.WriteLine("Max value is : {0}", res)
        Console.ReadLine()
     End Sub
  End Module
```

- When the above code is compiled and executed, it produces the following result:

OUTPUT:

```
Max value is : 200
```

Recursive Function

- A function can call itself.
- This is known as recursion.
- Following is an example that calculates factorial for a given number using a recursive function:

```
Module myfunctions
  Function factorial(ByVal num As Integer) As Integer
    ' local variable declaration */
    Dim result As Integer
    If (num = 1) Then
      Return 1
    Else
      result = factorial(num - 1) * num
      Return result
    End If
  End Function
  Sub Main()
    'calling the factorial method
    Console.WriteLine("Factorial of 6 is : {0}", factorial(6))
    Console.WriteLine("Factorial of 7 is : {0}", factorial(7))
    Console.WriteLine("Factorial of 8 is : {0}", factorial(8))
    Console.ReadLine()
  End Sub
End Module
```

Param Arrays

- At times, while declaring a function or sub procedure, you are not sure of the number of arguments passed as a parameter.
- VB.Net param arrays (or parameter arrays) come into help at these times.
- The following example demonstrates using the Param Arrays

```
Module myparamfunc
   Function AddElements(ParamArray arr As Integer()) As Integer
      Dim sum As Integer = 0
      Dim i As Integer = 0
      For Each i In arr
          sum += i
      Next i
      Return sum
   End Function
   Sub Main()
      Dim sum As Integer
      sum = AddElements(512, 720, 250, 567, 889)
      Console.WriteLine("The sum is: {0}", sum)
      Console.ReadLine()
   End Sub
End Module
```

- When the above code is compiled and executed, it produces the following result:

```
The sum is: 2938
```

Passing Arrays as Function Arguments

- You can pass an array as a function argument in VB.Net.
- The following example demonstrates Passing Arrays as Function Arguments.

```
Module arrayParameter
   Function getAverage(ByVal arr As Integer(), ByVal size As Integer) As
Double
       Dim i As Integer
       Dim avg As Double
       Dim sum As Integer = 0
       For i = 0 To size - 1
           sum += arr(i)
       Next i
       avg = sum / size
       Return avg
    End Function
    Sub Main()
        Dim balance As Integer() = {1000, 2, 3, 17, 50}
        Dim avg As Double
        avg = getAverage(balance, 5)
               Console.WriteLine("Average value is: {0} ", avg)
        Console.ReadLine()
    End Sub
End Module
```

Statements

- A statement is a complete instruction in Visual Basic programs.
- It may contain keywords, operators, variables, literal values, constants and expressions.
- Statements could be categorized as:
  - Declaration statements
  - Executable statements

Declaration statements

- These are the statements where you name a variable, constant, or procedure, and can also specify a data type.
- The declaration statements are used to name and define procedures, variables, properties, arrays, and constants.
- When you declare a programming element, you can also define its data type, access level, and scope.
- The programming elements you may declare include variables, constants, enumerations, classes, structures, modules, interfaces, procedures, procedure parameters, function returns, external procedure references, operators, properties, events, and delegates.

Executable statements

- These are the statements, which initiate actions.
- These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant.
- In the last case, it is called an Assignment statement.

Statements and description

Dim Statement

- Declares and allocates storage space for one or more variables.
- Example

```
Dim number As Integer

Dim quantity As Integer = 100

Dim message As String = "Hello!"
```

Const Statement

- Declares and defines one or more constants.
- Example

```
Const maximum  As Long = 1000

Const naturalLogBase As Object

= CDec(2.7182818284)
```

Enum statement

- Declares an enumeration and defines the values of its members.
- Example

```
Enum CoffeeMugSize

    Jumbo

    ExtraLarge

    Large

    Medium

    Small

End Enum
```

Class Statement

- Declares the name of a class and introduces the definition of the variables, properties, events, and procedures that the class comprises.
- Example

```
Class Box

Public length As Double

Public breadth As Double

Public height As Double

End Class
```

Structure Statement

- Declares the name of a structure and introduces the definition of the variables, properties, events, and procedures that the structure comprises.

- Example

```
Structure Box
Public length As Double
Public breadth As Double
Public height As Double
End Structure
```

Module statement

- Declares the name of a module and introduces the definition of the variables, properties, events, and procedures that the module comprises.
- Example

```
Public Module myModule
Sub Main()
Dim user As String =
InputBox("What is your name?")
MsgBox("User name is" & user)
End Sub
End Module
```

Interface Statement

- Declares the name of an interface and introduces the definitions of the members that the interface comprises.
- Example

```
Public Interface MyInterface
    Sub doSomething()
End Interface
```

Function Statement

- Declares the name, parameters, and code that define a Function procedure.

- Example

```
Function myFunction
(ByVal n As Integer) As Double
    Return 5.87 * n
End Function
```

Sub statement

- Declares the name, parameters, and code that define a Sub procedure
- Example

```
Sub mySub(ByVal s As String)
    Return
End Sub
```

Declare Statement

- Declares a reference to a procedure implemented in an external file.
- Example

```
Declare Function getUserName
Lib "advapi32.dll"
Alias "GetUserNameA"
(
  ByVal lpBuffer As String,
  ByRef nSize As Integer) As Integer
```

Operator Statement

- Declares the operator symbol, operands, and code that define an operator procedure on a class or structure.
- Example

```
Public Shared Operator +
(ByVal x As obj, ByVal y As obj) As obj
        Dim r As New obj
' implemention code for r = x + y
        Return r
    End Operator
```

Property Statement

- Declares the name of a property, and the property procedures used to store and retrieve the value of the property.
- Example

```
ReadOnly Property quote() As String
    Get
        Return quoteString
    End Get
End Property
```

Event Statement

- Declares a user-defined event.
- Example

```
Public Event Finished()
```

Delegate Statement

- Used to declare a delegate.
- Example

```
Delegate Function MathOperator(
    ByVal x As Double,
    ByVal y As Double
) As Double
```

Executable Statements

- An executable statement performs an action.
- Statements calling a procedure, branching to another place in the code, looping through several statements, or evaluating an expression are executable statements.
- An assignment statement is a special case of an executable statement.
- The following example demonstrates executable statement

```
Module decisions
  Sub Main()
    'local variable definition '
    Dim a As Integer = 10
    ' check the boolean condition using if statement '
    If (a < 20) Then
      ' if condition is true then print the following '
      Console.WriteLine("a is less than 20")
    End If
    Console.WriteLine("value of a is : {0}", a)
    Console.ReadLine()
  End Sub
End Module
```

- When the above code is compiled and executed, it produces the following result:

```
a is less than 20;
value of a is : 10
```

Directives

- The VB.Net compiler directives give instructions to the compiler to preprocess the information before actual compilation starts.
  - #Const constname = expression
- All these directives begin with #, and only white-space characters may appear before a directive on a line.
- These directives are not statements.
- VB.Net compiler does not have a separate preprocessor; however, the directives are processed as if there was one.
- In VB.Net, the compiler directives are used to help in conditional compilation.
- Unlike C and C++ directives, they are not used to create macros.

Compiler Directives in VB.Net.

- VB.Net provides the following set of compiler directives:
  - The #Const Directive,
  - The #ExternalSource Directive,
  - The #If...Then...#Else Directives,
  - The #Region Directive.

#Const Directives

- This directive defines conditional compiler constants.
- Syntax for this directive is:
  - #Const constname = expression
- Where,
  - constname: specifies the name of the constant. Required.
  - expression: it is either a literal, or other conditional compiler constant, or a combination including any or all arithmetic or logical operators except Is.
- For example,
  - #Const state = "WEST BENGAL"
- Example

```
Module mydirectives
#Const age = True
Sub Main()
   #If age Then
      Console.WriteLine("You are welcome to the Robotics Club")
   #End If
   Console.ReadKey()
End Sub
End Module
```

- When the above code is compiled and executed, it produces the following result:

```
You are welcome to the Robotics Club
```

#ExternalSource Directives

- This directive is used for indicating a mapping between specific lines of source code and text external to the source.
- It is used only by the compiler and the debugger has no effect on code compilation.
- This directive allows including external code from an external code file into a source code file.
- Syntax for this directive is:

```
#ExternalSource( StringLiteral , IntLiteral )
   [ LogicalLine ]
#End ExternalSource
```

- The parameters of #ExternalSource directive are the path of external file, line number of the first line, and the line where the error occurred.
- The following code demonstrates a hypothetical use of the directive:

```
Module mydirectives

    Public Class ExternalSourceTester

        Sub TestExternalSource()

        #ExternalSource("c:\vbprogs\directives.vb", 5)

            Console.WriteLine("This is External Code. ")

        #End ExternalSource

        End Sub

    End Class

    Sub Main()

        Dim  t As New ExternalSourceTester()

        t.TestExternalSource()

        Console.WriteLine("In Main.")

        Console.ReadKey()

    End Sub
```

The #If...Then...#Else Directives

- This directive conditionally compiles selected blocks of Visual Basic code.
- Syntax for this directive is:

```
#If expression Then
   statements
[ #ElseIf expression Then
   [ statements ]
...
#ElseIf expression Then
   [ statements ] ]
[ #Else
   [ statements ] ]
#End If
```

- The following code demonstrates a hypothetical use of the directive:

```
#Const TargetOS = "Linux"
#If TargetOS = "Windows 7" Then
   ' Windows 7 specific code
#ElseIf TargetOS = "WinXP" Then
   ' Windows XP specific code
#Else
   ' Code for other OS
#End if
```

The #Region Directives

- This directive helps in collapsing and hiding sections of code in Visual Basic files.
- Syntax for this directive is:

```
#Region "identifier_string"
#End Region
```

- Example

```
#Region "StatsFunctions"

    ' Insert code for the Statistical functions here.

#End Region
```

Decision Making

- Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- VB.Net provides the following types of decision making statements.
  - If ... Then statement.
  - If...Then...Else statement.
  - Nested If statements.
  - Select Case statement.
  - Nested Select Case statements.

If ... Then statement

- An If...Then statement consists of a Boolean expression followed by one or more statements.
- It is the simplest form of control statement, frequently used in decision making and changing the control flow of the program execution.

```
If condition Then

[Statement(s)]

End If
```

- Where, condition is a Boolean or relational condition and Statement(s) is a simple or compound statement.

```
If (a <= 20) Then

   c= c+1

End If
```

- If the condition evaluates to true, then the block of code inside the If statement will be executed.
- If condition evaluates to false, then the first set of code after the end of the If statement (after the closing End If) will be executed.

If...Then...Else statement

- An If...Then statement can be followed by an optional Else statement, which executes when the boolean expression is false.

- The syntax of an If...Then... Else statement in VB.Net is as follows:

```
If(boolean_expression)Then
   'statement(s) will execute if the Boolean expression is true
Else
  'statement(s) will execute if the Boolean expression is false
End If
```

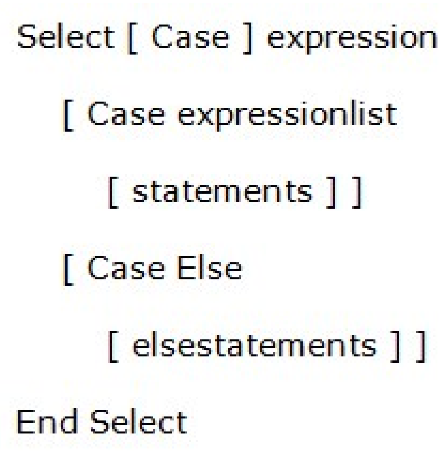
Nested If statements

- You can use one If or Else if statement inside another If or Else if statement(s).
- It is always legal in VB.Net to nest If-Then-Else statements, which means you can use one If or ElseIf statement inside another If ElseIf statement(s).
- The syntax for a nested If statement is as follows:

```
If( boolean_expression 1)Then
   'Executes when the boolean expression 1 is true
   If(boolean_expression 2)Then
         'Executes when the boolean expression 2 is true
   End If
End If
```

Select Case statement

- A Select Case statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each select case.
- The syntax for a Select Case statement in VB.Net is as follows:



```
Select [ Case ] expression
    [ Case expressionlist
        [ statements ] ]
    [ Case Else
        [ elsestatements ] ]
End Select
```

Nested Select Case statements

- You can use one select case statement inside another select case statement(s).
- It is possible to have a select statement as part of the statement sequence of an outer select statement.
- Even if the case constants of the inner and outer select contain common values, no conflicts will arise.

Loops

- There may be a situation when you need to execute a block of code several number of times.
- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages.
- VB.Net provides following types of loops to handle looping requirements.
    - Do Loop,
    - For...Next,
    - For Each...Next,
    - While... End While,
    - With... End With,
    - Nested loops.

Do Loop

- It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True.
- It could be terminated at any time with the Exit Do statement.
- The syntax for this loop construct is:

```
Do { While | Until } condition
    [ statements ]
    [ Continue Do ]
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop
-or-
Do
    [ statements ]
    [ Continue Do ]
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop { While | Until } condition
```

For...Next

- It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.
- The syntax for this loop construct is:

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ counter ]
```

For Each...Next

- It repeats a group of statements for each element in a collection.
- This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.
- The syntax for this loop construct is:

```
For Each element [ As datatype ] In group
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ element ]
```

While... End While

- It executes a series of statements as long as a given condition is True.
- The syntax for this loop construct is:

```
While condition
    [ statements ]
    [ Continue While ]
    [ statements ]
    [ Exit While ]
    [ statements ]
End While
```

- Here, statement(s) may be a single statement or a block of statements.
- The condition may be any expression, and true is logical true.
- The loop iterates while the condition is true.

With... End With

- The syntax for this loop construct is:

```
With object
    [ statements ]
End With
```

- It is not exactly a looping construct.
- It executes a series of statements that repeatedly refers to a single object or structure.

Nested loops

- You can use one or more loops inside any another While, For or Do loop.
  - Nested For loop,
  - Nested While loop,
  - Nested Do...While loop.

Loop Control Statements

- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- VB.Net provides the following control statements.
  - Exit statement,
  - Continue statement,
  - GoTo statement.

Exit statement

- The Exit statement transfers the control from a procedure or block immediately to the statement following the procedure call or the block definition.
- It terminates the loop, procedure, try block or the select block from where it is called.
- If you are using nested loops (i.e., one loop inside another loop), the Exit statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- The syntax for the Exit statement is:

```
Exit { Do | For | Function | Property | Select | Sub | Try | While }
```

- Terminates the loop or select case statement and transfers execution to the statement immediately following the loop or select case.

Continue statement

- The Continue statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- It works somewhat like the Exit statement.
- Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
- For the For...Next loop, Continue statement causes the conditional test and increment portions of the loop to execute.
- For the While and Do...While loops, continue statement causes the program control to pass to the conditional tests.
- The syntax for a Continue statement is as follows:

```
Continue { Do | For | While }
```

- Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

GoTo statement

- The GoTo statement transfers control unconditionally to a specified line in a procedure.
- The syntax for the GoTo statement is:

```
GoTo label
```

- Transfers control to the labeled statement.
- Though it is not advised to use GoTo statement in your program.

Strings - Creating a String Object

- In VB.Net, you can use strings as array of characters, however, more common practice is to use the String keyword to declare a string variable.
- The string keyword is an alias for the System.String class.
- When the above code is compiled and executed, it produces the following result:
- You can create string object using one of the following methods:
  - By assigning a string literal to a String variable.
  - By using a String class constructor.
  - By using the string concatenation operator (+).
  - By retrieving a property or calling a method that returns a string.
  - By calling a formatting method to convert a value or object to its string representation.

Properties of the String Class

- The String class has the following two properties:
  - Chars
    - Gets the Char object at a specified position in the current String object.
  - Length
    - Gets the number of characters in the current String object.

Methods of the String Class

- The String class has numerous methods that help you in working with the string objects
  - Public Shared Function Compare (strA As String, strB As String) As Integer
    - Compares two specified string objects and returns an integer that indicates their relative position in the sort order.
  - Public Shared Function Compare (strA As String, strB As String, ignoreCase As Boolean) As Integer
    - Compares two specified string objects and returns an integer that indicates their relative position in the sort order.
    - However, it ignores case if the Boolean parameter is true.
  - Public Shared Function Concat (str0 As String, str1 As String) As String
    - Concatenates two string objects.

- Public Shared Function Concat (str0 As String, str1 As String, str2 As String) As String
  - Concatenates three string objects.
- Public Shared Function Concat (str0 As String, str1 As String, str2 As String, str3 As String) As String
  - Concatenates four string objects.
- Public Function Contains (value As String) As Boolean
  - Returns a value indicating whether the specified string object occurs within this string.
- Public Shared Function Copy (str As String) As String
  - Creates a new String object with the same value as the specified string.
- Public Sub CopyTo (sourceIndex As Integer, destination As Char(), destinationIndex As Integer, count As Integer)
  - Copies a specified number of characters from a specified position of the string object to a specified position in an array of Unicode characters.
- Public Function EndsWith (value As String) As Boolean
  - Determines whether the end of the string object matches the specified string.
- Public Function Equals (value As String) As Boolean
  - Determines whether the current string object and the specified string object have the same value.
- Public Shared Function Equals (a As String, b As String) As Boolean
  - Determines whether two specified string objects have the same value.
- Public Shared Function Format (format As String, arg0 As Object) As String
  - Replaces one or more format items in a specified string with the string representation of a specified object.
- Public Function IndexOf (value As Char) As Integer
  - Returns the zero-based index of the first occurrence of the specified Unicode character in the current string.
- Public Function IndexOf (value As String) As Integer
  - Returns the zero-based index of the first occurrence of the specified string in this instance.
- Public Function IndexOf (value As Char, startIndex As Integer) As Integer
  - Returns the zero-based index of the first occurrence of the specified Unicode character in this string, starting search at the specified character position.

- Public Function IndexOf (value As String, startIndex As Integer) As Integer
  - Returns the zero-based index of the first occurrence of the specified string in this instance, starting search at the specified character position.
- Public Function IndexOfAny (anyOf As Char()) As Integer
  - Returns the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters.
- Public Function IndexOfAny (anyOf As Char(), startIndex As Integer) As Integer
  - Returns the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters, starting search at the specified character position.
- Public Function Insert (startIndex As Integer, value As String) As String
  - Returns a new string in which a specified string is inserted at a specified index position in the current string object.
- Public Shared Function IsNullOrEmpty (value As String) As Boolean
  - Indicates whether the specified string is null or an Empty string.
- Public Shared Function Join (separator As String, ParamArray value As String()) As String
  - Concatenates all the elements of a string array, using the specified separator between each element.
- Public Shared Function Join (separator As String, value As String(), startIndex As Integer, count As Integer) As String
  - Concatenates the specified elements of a string array, using the specified separator between each element.
- Public Function LastIndexOf (value As Char) As Integer
  - Returns the zero-based index position of the last occurrence of the specified Unicode character within the current string object.
- Public Function LastIndexOf (value As String) As Integer
  - Returns the zero-based index position of the last occurrence of a specified string within the current string object.
- Public Function Remove (startIndex As Integer) As String
  - Removes all the characters in the current instance, beginning at a specified position and continuing through the last position, and returns the string.
- Public Function Remove (startIndex As Integer, count As Integer) As String
  - Removes the specified number of characters in the current string beginning at a specified

position and returns the string.

- Public Function Replace (oldChar As Char, newChar As Char) As String
    - Replaces all occurrences of a specified Unicode character in the current string object with the specified Unicode character and returns the new string.
- Public Function Replace (oldValue As String, newValue As String) As String
    - Replaces all occurrences of a specified string in the current string object with the specified string and returns the new string.
- Public Function Split (ParamArray separator As Char()) As String()
    - Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array.
- Public Function Split (separator As Char(), count As Integer) As String()
    - Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array.
    - The int parameter specifies the maximum number of substrings to return.
- Public Function StartsWith (value As String) As Boolean
    - Determines whether the beginning of this string instance matches the specified string.
- Public Function ToCharArray (startIndex As Integer, length As Integer) As Char()
    - Returns a Unicode character array with all the characters in the current string object, starting from the specified index and up to the specified length.
- Public Function ToCharArray As Char()
    - Returns a Unicode character array with all the characters in the current string object.
- Public Function ToLower As String
    - Returns a copy of this string converted to lowercase.
- Public Function ToUpper As String
    - Returns a copy of this string converted to uppercase.
- Public Function Trim As String
    - Removes all leading and trailing white-space characters from the current String object.

Some of Simple Applications

Simple Application 1

- VB.Net program to accept any character from keyboard and display whether it is vowel or not.
  - Click Start→Programs→MicrosoftVisualStudio 2008 → MicrosoftVisualStudio 2008.
  - Select File →New → Project → Windows Application.
  - Place the label, textbox and button in the window.
  - Write the code in click event of btnresult.
  - Run the application by pressing F5 key or by clicking debug button.

Simple Application 2

- VB .NET program to accept a string and convert the case of the characters.
  - Click Start→Programs→MicrosoftVisualStudio 2008 → MicrosoftVisualStudio 2008.
  - Select File → New→ Project → Windows application.
  - Open the Windows application.
  - Place a label and text as enter the string.
  - Place two textboxes one is used to enter the string, other used to display the result.
  - Then place two buttons, one is used to display result and other used to clear the textboxes.
  - Then write the code in click event of button1 and button2.
  - Run the application by F5 key or pressing debug button.

Simple Application 3

- Develop a menu based VB .NET application to implement a text editor with cut, copy, paste, save and close operations.
  - Click Start→Programs→MicrosoftVisualStudio 2008 → MicrosoftVisualStudio 2008.
  - Select File → New→ Project → Windows application.
  - Open the Windows application.
  - Place the MenuStrip in the form from tool box.
  - Main menu is constructed with number of sub menus named as open, close, save, cut, copy, paste, undo, redo, color, font etc.,
  - Drag and place a RichTextBox in the form.

- Insert the source code for appropriate menu items and finally run the application.

Simple Application 4

- program to implement the calculator with memory and recall operations.
  - Click Start→Programs→MicrosoftVisualStudio 2008 → MicrosoftVisualStudio 2008.
  - Select File → New→ Project → Windows application.
  - Open the windows application form.
  - Place fifteen buttons and named as 0 to 9 and operators +, -, *,⁄, =.
  - Add another three buttons named as memory, recall, cancel.
  - Type the code in appropriate events.
  - Run the application by F5 key or pressing debug button.

Simple Application 5

- Develop a form in VB .NET to pick a date from calendar control and display the day, month, year in separate textboxes.
  - Click Start→Programs→MicrosoftVisualStudio 2008 → MicrosoftVisualStudio 2008.
  - Select File → New → Project → Windows application.
  - Open the windows application form.
  - Place a DateTimePicker to select the date.
  - Place four label and textboxes named as day, date, month, and year.
  - Edit the following code in form load event and value changed event of Datetimepicker.
  - Run the application by pressing F5 key.

Simple Application 6

- Develop a VB .NET application to perform timer based quiz of 10 questions.
  - Click Start→Programs→Microsoft Visual Studio 2008 → Microsoft Visual Studio 2008.
  - Select File → New→Project → Windows application.
  - Open the windows application form.
  - Place a TextBox and named as txtquestion.
  - Place a GroupBox and enter text as Answer.
  - Within the GroupBox place three RadioButton.

- Place a Button named as Result.
- Then place a Timer and set interval property of timer as 3000.
- Insert the code in Form_Load, btnresult,Timer1_Tick and RadioButton_Checked Chenged event.
- Run the application by pressing F5 key.

Simple Application 7

- Develop a VB .NET application using the File and Directory controls to implement a common dialog box.
  - Click Start→Programs→Microsoft Visual Studio 2008 →Microsoft Visual Studio 2008.
  - Select File → New→ Project → Windows application.
  - Open the Windows Application form.
  - Place FolderBrowserDialog and OpenFileDialog from ToolBox.
  - Place a GroupBox and enter Text as Select the Option.
  - Within the GroupBox place two RadioButton and enter Text as Folder Open, File Open.
  - Place a Button named as OK.
  - Place two LabelBox and TextBox from the ToolBox.
  - Insert the code in click event of btn_ok.
  - Run the application by pressing F5 key.

UNIT - 7
File and Database Applications

File Access

- A file is a collection of data stored in a disk with a specific name and a directory path.
- When a file is opened for reading or writing, it becomes a stream.
- File is used to store a data, using that we can operate the data.
- The operations are,
  - Reading,
  - Writing,
  - Close...etc.

Stream

- The stream is basically the sequence of bytes passing through the communication path.
- There are two main streams.
- Input stream:
  - The input stream is used for reading data from file (read operation).
- Output stream:
  - The output stream is used for writing into the file (write operation).

VB.Net I/O Classes

- The System.IO namespace has various classes that are used for performing various operations with files, like creating and deleting files, reading from or writing to a file, closing a file, etc.
- Some commonly used non-abstract classes in the System.IO namespace are:

BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directory.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.

FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access of streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.
StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

The FileStream Class

- The FileStream class in the System.IO namespace helps in reading from, writing to and closing files.
- This class derives from the abstract class Stream.
- You need to create a FileStream object to create a new file or open an existing file.
- The syntax for creating a FileStream object is as follows:

```
Dim <object_name> As FileStream = New FileStream
(<file_name>, <FileMode Enumerator>, <FileAccess Enumerator>, <FileShare
Enumerator>)
```

  - For example, for creating a FileStream object F for reading a file named sample.txt:

```
Dim f1 As FileStream = New FileStream("test.dat", FileMode.OpenOrCreate,
FileAccess.ReadWrite)
```

- The following program demonstrates use of the FileStreamclass:

```
Imports System.IO
Module fileProg
  Sub Main()
    Dim f1 As FileStream = New FileStream("test.dat", _
         FileMode.OpenOrCreate, FileAccess.ReadWrite)
    Dim i As Integer
    For i = 0 To 20
       f1.WriteByte(CByte(i))
    Next i
    f1.Position = 0
    For i = 0 To 20
       Console.Write("{0} ", f1.ReadByte())
    Next i
    f1.Close()
    Console.ReadKey()
  End Sub
End Module
```

- When the code is compiled and executed, it produces the following result:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

- The FileMode enumerator defines various methods for opening files.
- The members of the FileMode enumerator are:
  - Append:
    - It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist.
  - Create:
    - It creates a new file.
  - CreateNew:
    - It specifies to the operating system that it should create a new file.
  - Open:
    - It opens an existing file.
  - OpenOrCreate:
    - It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file.
  - Truncate:
    - It opens an existing file and truncates its size to zero bytes.

FileShare

- FileShare enumerators have the following members:
  - Inheritable:
    - It allows a file handle to pass inheritance to the child processes.
  - None:
    - It declines sharing of the current file.
  - Read:
    - It allows opening the file for reading.
  - ReadWrite:
    - It allows opening the file for reading and writing.
  - Write:
    - It allows opening the file for writing.

Advanced File Operations in VB.Net

- StreamReader and StreamWriter classes.
- BinaryReader and BinaryWriter classes.
- The DirectoryInfo Class.
- The FileInfo Class.
- However, to utilize the immense powers of System.IO classes, you need to know the commonly used properties and methods of these classes.

The StreamReader Class

- The StreamReader class also inherits from the abstract base class TextReader that represents a reader for reading series of characters.
- In follow describes some of the commonly used methods of the StreamReader class:
  - Public Overrides Sub Close
    - It closes the StreamReader object and the underlying stream and releases any system resources associated with the reader.
  - Public Overrides Function Peek As Integer
    - Returns the next available character but does not consume it.
  - Public Overrides Function Read As Integer

- Reads the next character from the input stream and advances the character position by one character.

- The following program demonstrates use of the StreamReader:

```
Imports System.IO
Module fileProg
  Sub Main()
    Try
      ' Create an instance of StreamReader to read from a file.
      ' The using statement also closes the StreamReader.
      Using sr As StreamReader = New StreamReader("e:/jamaica.txt")
         Dim line As String
         ' Read and display lines from the file until the end of
         ' the file is reached.
         line = sr.ReadLine()
         While (line <> Nothing)
            Console.WriteLine(line)
            line = sr.ReadLine()
         End While
      End Using
    Catch e As Exception
      ' Let the user know what went wrong.
      Console.WriteLine("The file could not be read:")
      Console.WriteLine(e.Message)
    End Try
    Console.ReadKey()
  End Sub
End Module
```

The StreamWriter Class

- The StreamWriter class inherits from the abstract class TextWriter that represents a writer, which can write a series of character.

- Some of the most commonly used methods of StreamWriter class:

  - Public Overrides Sub Close
    - Closes the current StreamWriter object and the underlying stream.
  - Public Overrides Sub Flush
    - Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
  - Public Overridable Sub Write (value As Boolean)
    - Writes the text representation of a Boolean value to the text string or stream. (Inherited from TextWriter.)
  - Public Overrides Sub Write (value As Char)

- Writes a character to the stream.

➢ Public Overridable Sub Write (value As Decimal)

- Writes the text representation of a decimal value to the text string or stream.

- The following program demonstrates use of the StreamWriter Class:

```
Imports System.IO
Module fileProg
   Sub Main()
      Dim names As String() = New String() {"Zara Ali", _
         "Nuha Ali", "Amir Sohel", "M Amlan"}
      Dim s As String
      Using sw As StreamWriter = New StreamWriter("names.txt")
         For Each s In names
            sw.WriteLine(s)
         Next s
      End Using
      ' Read and show each line from the file.
      Dim line As String
      Using sr As StreamReader = New StreamReader("names.txt")
         line = sr.ReadLine()
         While (line <> Nothing)
            Console.WriteLine(line)
            line = sr.ReadLine()
         End While
      End Using
      Console.ReadKey()
   End Sub
End Module
```

The BinaryReader Class

- The BinaryReader class is used to read binary data from a file.
- A BinaryReader object is created by passing a FileStream object to its constructor.
- Some of the commonly used methods of the BinaryReader class

➢ Public Overridable Sub Close

- It closes the BinaryReader object and the underlying stream.

➢ Public Overridable Function Read As Integer

- Reads the characters from the underlying stream and advances the current position of the stream.

➢ Public Overridable Function ReadBoolean As Boolean

- Reads a Boolean value from the current stream and advances the current position of the stream by one byte.

The BinaryWriter Class

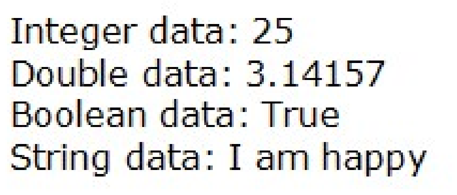
- The BinaryWriter class is used to write binary data to a stream.
- A BinaryWriter object is created by passing a FileStream object to its constructor.
- Some of the commonly used methods of the BinaryWriter class
    - Public Overridable Sub Close
        - It closes the BinaryWriter object and the underlying stream.
    - Public Overridable Sub Flush
        - Clears all buffers for the current writer and causes any buffered data to be written to the underlying device.
    - Public Overridable Function Seek (offset As Integer, origin As SeekOrigin) As Long
        - Sets the position within the current stream.
    - Public Overridable Sub Write (value As Boolean)
        - Writes a one-byte Boolean value to the current stream, with 0 representing false and 1 representing true.

Example

- Example for BinaryReader and BinaryWriter classes:

```
Imports System.IO
Module fileProg
  Sub Main()
    Dim bw As BinaryWriter
    Dim br As BinaryReader
    Dim i As Integer = 25
    Dim d As Double = 3.14157
    Dim b As Boolean = True
    Dim s As String = "I am happy"
    'create the file
    Try
       bw = New BinaryWriter(New FileStream("mydata", FileMode.Create))
    Catch e As IOException
       Console.WriteLine(e.Message + "\n Cannot create file.")
       Return
    End Try
    'writing into the file
    Try
       bw.Write(i)
       bw.Write(d)
       bw.Write(b)
       bw.Write(s)
    Catch e As IOException
       Console.WriteLine(e.Message + "\n Cannot write to file.")
       Return
    End Try
    bw.Close()
    'reading from the file
    Try
       br = New BinaryReader(New FileStream("mydata", FileMode.Open))
    Catch e As IOException
       Console.WriteLine(e.Message + "\n Cannot open file.")
       Return
    End Try
     Try
        i = br.ReadInt32()
        Console.WriteLine("Integer data: {0}", i)
        d = br.ReadDouble()
        Console.WriteLine("Double data: {0}", d)
        b = br.ReadBoolean()
        Console.WriteLine("Boolean data: {0}", b)
        s = br.ReadString()
        Console.WriteLine("String data: {0}", s)
     Catch e As IOException
        Console.WriteLine(e.Message + "\n Cannot read from file.")
        Return
     End Try
     br.Close()
     Console.ReadKey()
   End Sub
 End Module
```

- When the above code is compiled and executed, it produces the following result:



```
Integer data: 25
Double data: 3.14157
Boolean data: True
String data: I am happy
```

The DirectoryInfo Class

- The DirectoryInfo class is derived from the FileSystemInfo class.
- It has various methods for creating, moving, and browsing through directories and subdirectories.
- This class cannot be inherited.
- Following are some commonly used properties of the DirectoryInfo class:
  - Attributes
    - Gets the attributes for the current file or directory.
  - CreationTime
    - Gets the creation time of the current file or directory.
  - Exists
    - Gets a Boolean value indicating whether the directory exists.
  - Extension
    - Gets the string representing the file extension.
  - FullName
    - Gets the full path of the directory or file.
  - LastAccessTime
    - Gets the time the current file or directory was last accessed.
  - Name
    - Gets the name of this DirectoryInfo instance.

The FileInfo Class

- The FileInfo class is derived from the FileSystemInfo class.
- It has properties and instance methods for creating, copying, deleting, moving, and opening of files, and helps in the creation of FileStream objects.
- This class cannot be inherited.

- Following are some commonly used properties of the FileInfo class:

1	Attributes Gets the attributes for the current file.
2	CreationTime Gets the creation time of the current file.
3	Directory Gets an instance of the directory which the file belongs to.
4	Exists Gets a Boolean value indicating whether the file exists.
5	Extension Gets the string representing the file extension.
6	FullName Gets the full path of the file.
7	LastAccessTime Gets the time the current file was last accessed.
8	LastWriteTime Gets the time of the last written activity of the file.
9	Length Gets the size, in bytes, of the current file.
10	Name Gets the name of the file.

Example

- Example for DirectoryInfo class and the FileInfo class:

```
Imports System.IO
Module fileProg
   Sub Main()
      'creating a DirectoryInfo object
      Dim mydir As DirectoryInfo = New DirectoryInfo("c:\Windows")
      ' getting the files in the directory, their names and size
      Dim f As FileInfo() = mydir.GetFiles()
      Dim file As FileInfo
      For Each file In f
         Console.WriteLine("File Name: {0} Size: {1}  ", file.Name, file.Length)
      Next file
      Console.ReadKey()
   End Sub
End Module
```

- When you compile and run the program, it displays the names of files and their size in the Windows directory.

Dialog Box

- There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc., to the user of an application.
- These built-in dialog boxes reduce the developer's time and workload.
- All of these dialog box control classes inherit from the CommonDialog class and override the RunDialog() function of the base class to create the specific dialog box.
- It all works like a normal windows applications.

Common Dialog Boxes

- There are two common dialog boxes.
- They are,
  - Predefined standard dialog boxes for:
    - File Opening and Saving,
    - Font and Color selection,
    - Printing and Previewing.
  - Predefined standard dialog boxes for:
    - Appears in the Component Tray.
- The RunDialog() function is automatically invoked when a user of a dialog box calls its ShowDialog() function.
- The ShowDialog method is used to display all the dialog box controls at run-time.
- It returns a value of the type of DialogResult enumeration.
- The values of DialogResult enumeration are:
  - Abort - returns DialogResult.Abort value, when user clicks an Abort button.
  - Cancel- returns DialogResult.Cancel, when user clicks a Cancel button.
  - Ignore - returns DialogResult.Ignore, when user clicks an Ignore button.
  - No - returns DialogResult.No, when user clicks a No button.
  - None - returns nothing and the dialog box continues running.
  - OK - returns DialogResult.OK, when user clicks an OK button.
  - Retry - returns DialogResult.Retry , when user clicks an Retry button.

- ➤ Yes - returns DialogResult.Yes, when user clicks an Yes button.

Commonly Used Dialog Box Controls

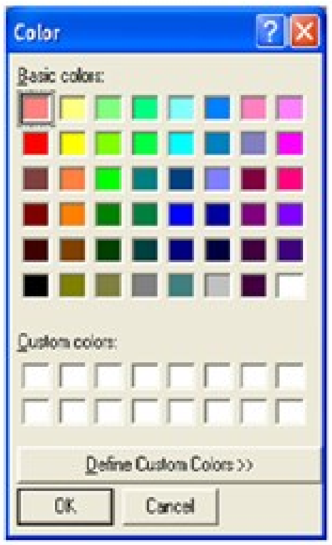
- OpenFileDialog.
- SaveFileDialog.
- FontDialog.
- ColorDialog.
- PrintDialog.
- PrintPreviewDialog.

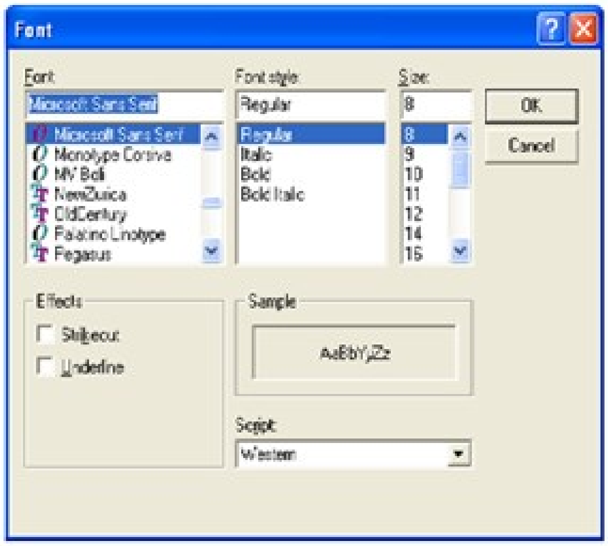
Displaying a Windows Common Dialog Box

- Use ShowDialog method to display the common dialog box at run time.
- ShowDialog only displays the dialog, it does not do anything else.

Using the Information from the Dialog Box

- Code must be written to retrieve and use the choice made by the user in the common dialog box.
- Example
  - ➤ Color Dialog displayed.
  - ➤ User selects color and clicks OK.
- The example is show in the figure.





Font: Microsoft Sans Serif
Microsoft Sans Serif
Monotype Corsiva
MV Boli
NewZuica
OldCentury
Palatino Linotype
Pegasus

Font style: Regular
Regular
Italic
Bold
Bold Italic

Size: 8
8
9
10
11
12
14
16

Script: Western

- Code must be written to apply the selected color to the object(s).
- To write a information in the dialog box we must written a code.

- The table explains the controls and the descriptions.

S.N	Control & Description
1	ColorDialog It represent a common dialog box that display available colors along with controls that enable the user to define custom colors.
2	FontDialog It prompts the user to choose a front from among those installed on the local computer and lets the user selects the font, font size, and color.
3	OpenFileDialog It prompts the user to open a file and allow the user to select a file to open.
4	SaveFileDialog It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
5	PrintDialog It lets the user to print documents by selecting a printer and choosing which section of the document to print from a windows Forms application.

Setting Initial Values

- Setting the initial value is done before executing the dialog box.
  - FontDialog1.Font = .subTotalLabel.Font or ColorDialog1.Color = .BackColor
- It should the user defined value.
- Before executing the Show Dialog method, assign the existing values of the object's properties that will be altered.
- When the dialog box appears, the current values will be selected.
- If the user presses Cancel, property settings for the objects will remain unchanged.

File and Database application

- An exception is a problem that arises during the execution of a program.
- An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- Exceptions provide a way to transfer control from one part of a program to another.
- VB.Net exception handling is built upon four keywords: Try, Catch, Finally and Throw.
  - Object instance of Framework Class Library (FCL)-supplied Exception class Or one of its

subclasses.

- Used by VB .NET to notify you of:
    - Errors
    - Problems
    - Other unusual conditions that may occur while your system is running.

Causing an Exception

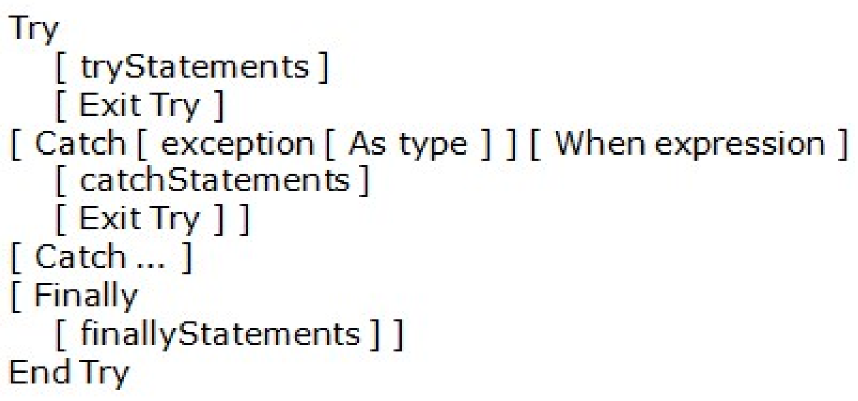
- Create deliberate error.
- Exception is thrown.
    - Message dialog indicates that code did not deal with exception.
    - Execution is interrupted.

Error Handling

- Try:
  - A Try block identifies a block of code for which particular exceptions will be activated.
  - It's followed by one or more Catch blocks.
- Catch:
  - A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
  - The Catch keyword indicates the catching of an exception.
- Finally:
  - The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
  - For example, if you open a file, it must be closed whether an exception is raised or not.
- Throw:
  - A program throws an exception when a problem shows up.
  - This is done using a Throw keyword.
- Every exception will execute at least three of the statements once the error would be accour.

Syntax

- A Try/Catch block is placed around the code that might generate an exception.
- Code within a Try/Catch block is referred to as protected code, and the syntax for using Try/Catch looks like the following:



```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

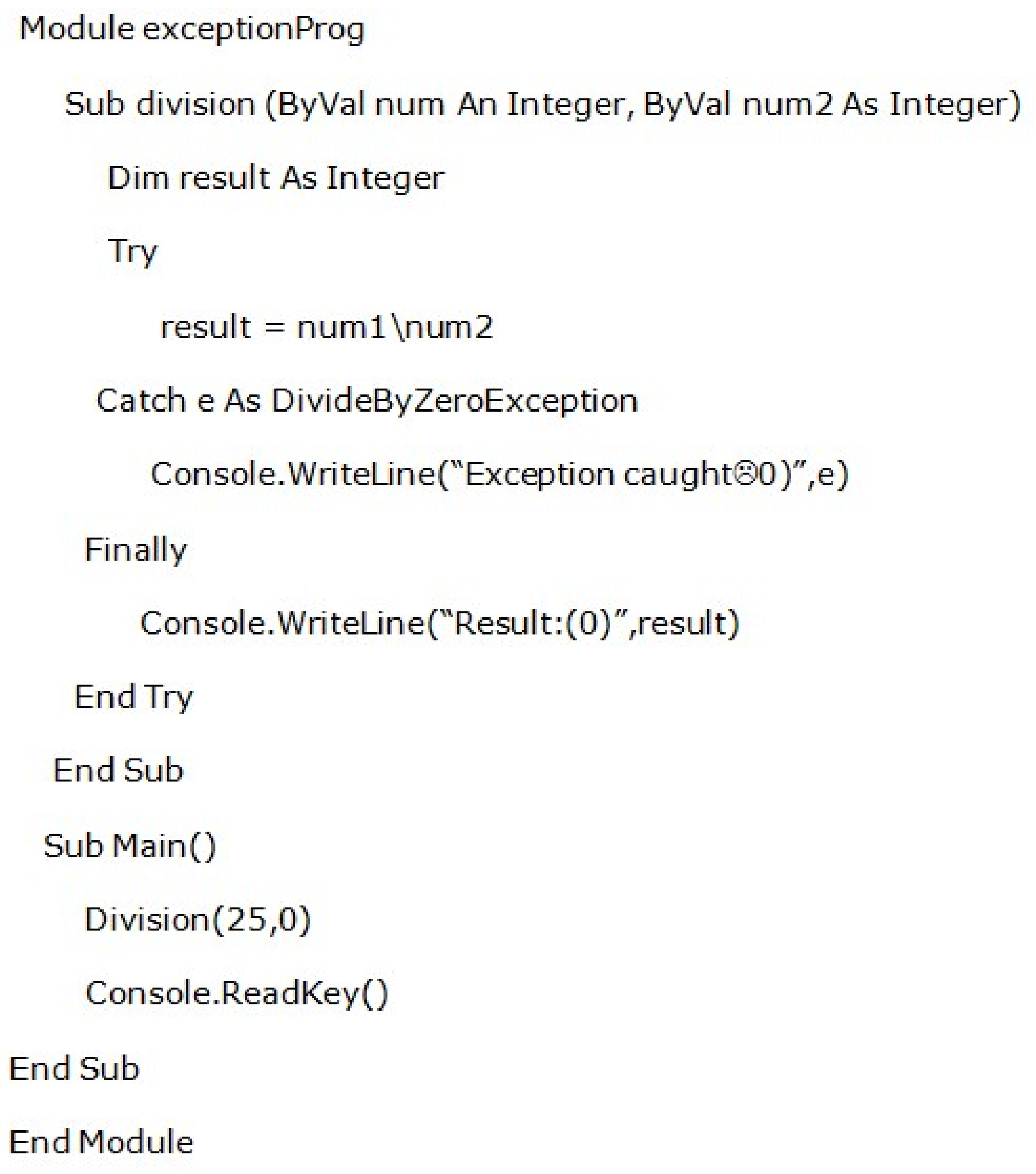
- Raise an exception, a method catches an exception using a combination of the Try and Catch keywords.

Exception Classes in .Net Framework

- In the .Net Framework, exceptions are represented by classes.
- The exception classes in .Net Framework are mainly directly or indirectly derived from the System.Exception class.
- Some of the exception classes derived from the System.Exception class are the System.ApplicationException and System.SystemException classes.
- The System.ApplicationException class supports exceptions generated by application programs.
- So the exceptions defined by the programmers should derive from this class.
- The System.SystemException class is the base class for all predefined system exception.

Handling Exceptions

- VB.Net provides a structured solution to the exception handling problems in the form of try and catch blocks.
- Using these blocks the core program statements are separated from the error-handling statements.
- These error handling blocks are implemented using the Try, Catch and Finally keywords.



```
Module exceptionProg
    Sub division (ByVal num An Integer, ByVal num2 As Integer)
        Dim result As Integer
        Try
            result = num1\num2
        Catch e As DivideByZeroException
            Console.WriteLine("Exception caught☹0)",e)
        Finally
            Console.WriteLine("Result:(0)",result)
        End Try
    End Sub
    Sub Main()
        Division(25,0)
        Console.ReadKey()
End Sub
End Module
```

Creating User-Defined Exceptions

- The following example demonstrates this:

```
Module exceptionProg
   Public Class TempIsZeroException : Inherits ApplicationException
      Public Sub New(ByVal message As String)
         MyBase.New(message)
      End Sub
   End Class
   Public Class Temperature
      Dim temperature As Integer = 0
      Sub showTemp()
         If (temperature = 0) Then
            Throw (New TempIsZeroException("Zero Temperature found"))
         Else
            Console.WriteLine("Temperature: {0}", temperature)
         End If
      End Sub
   End Class
   Sub Main()
      Dim temp As Temperature = New Temperature()
      Try
         temp.showTemp()
      Catch e As TempIsZeroException
         Console.WriteLine("TempIsZeroException: {0}", e.Message)
      End Try
      Console.ReadKey()
   End Sub
End Module
```

- You can also define your own exception.
- User-defined exception classes are derived from the ApplicationException class.

Throwing Objects

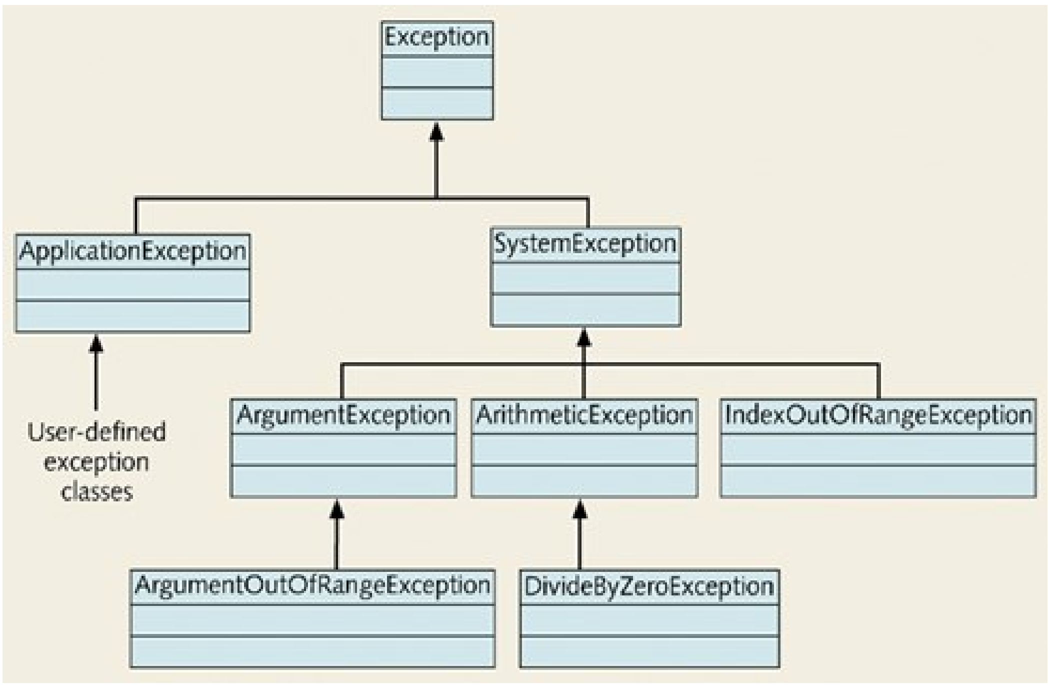
- You can use a throw statement in the catch block to throw the present object as:
   - Throw[expression]

```
Module exceptionProg
   Sub Main()
      Try
         Throw New ApplicationException("A custom exception _
                 is being thrown here...")
      Catch e As Exception
         Console.WriteLine(e.Message)
      Finally
         Console.WriteLine("Now inside the Finally Block")
      End Try
      Console.ReadKey()
   End Sub
End Module
```

- You can throw an object if it is either directly or indirectly derived from the System.Exception class.

Exception class hierarchy

- The diagram shows the class hierarchy of exception class.



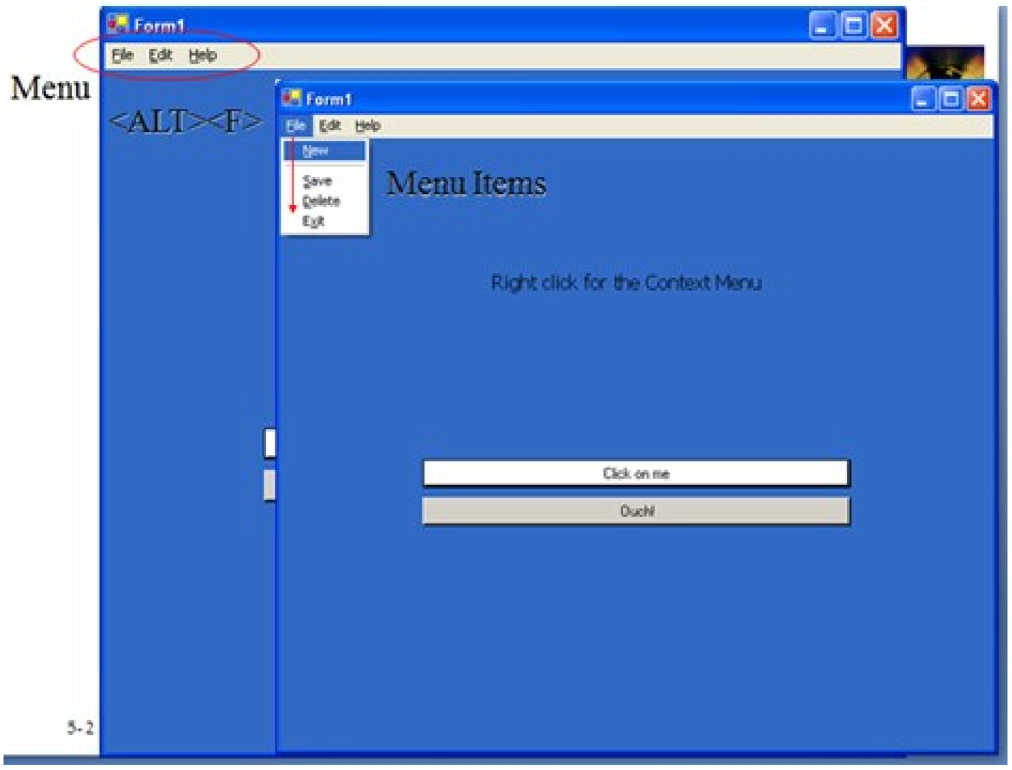
- It is commonly divided into two types of exceptions.
  - Application exception.
  - System exception.

Creating Context Menus

- Shortcut menus that pop up when you right-click.
- Are specific to the component to which user is pointing, reflecting options available for that component or situation.
- A context menu does not have a top level menu.
- Application can have multiple context menus.
- Add ContextMenu component to the form.
- Appears in the Component Tray.
- Click on the words "Context Menu", at the top of the form, the words "Type Here" appear.
- Click on the words "Type Here" and proceed as you did for the main menu.

Menu

- The picture will show you what is menu.



S.N	Property	Description
1	CanOverflow	Gets or sets a value indicating whether the MenuStrip supports overflow functionality.
2	GripStyle	Gets or sets the visibility of the grip used to reposition the control.
		Gets or sets the ToolStripMenuItem that is used to display a list of Multiple-

3	MdiWindowListItem	document interface (MDI) child forms.
4	ShowItemToolTips	Gets or sets a value indicating whether ToolTips are shown for the MenuStrip.
5	Stretch	Gets or sets a value indicating whether the MenuStrip stretches from end to end in its container.

- Menu is used to do multiple operations at output stage.

Menu Bar

- Contains menus which drop down to display list of menu items.
    - Each item has a name and text property.
    - Each item has a click event.
- Add MainMenu control to form.
    - Appears in the Component Tray, below form, where nondisplay controls are shown.
    - Words "Type Here" appear at the top of the form.

Defining Menus

- To create the menus simply type where the words "Type Here" appear at the top of the form.
- Include & symbol as you type to indicate keyboard access keys.
- You are actually entering the Text property for a MenuItem object.

```
Private Sub ExitToolStripMenuItem_Click(sender As Object, e As EventArgs) _
Handles ExitToolStripMenuItem.Click
End
End Sub
```

- Change MenuItem object names in the Properties window and follow consistent naming conventions.

Submenus

- Pop up to the right of a menu item.
- Filled triangle to the right of the menu item indicates to the user the existence of a submenu.
- Create submenus by moving to the right of a menu item and typing the next item's text.

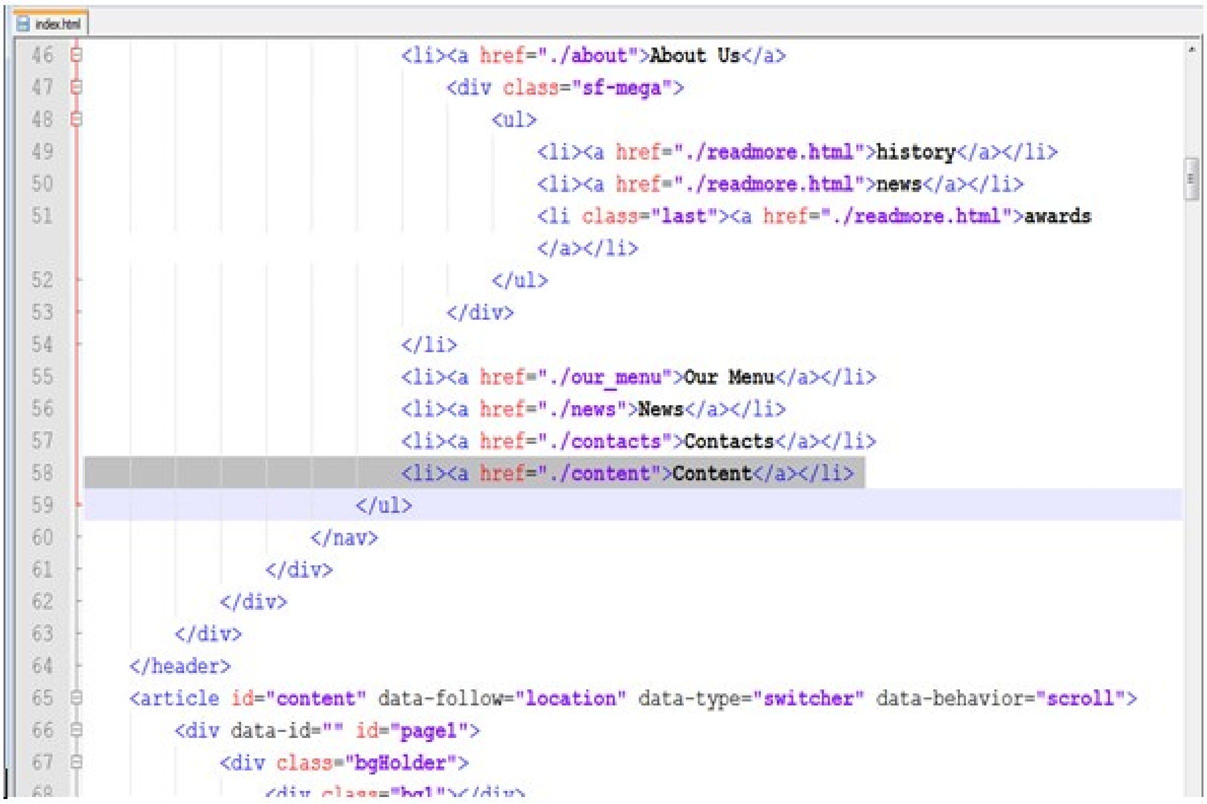
Separator Bars

- Used for grouping menu items according to their purpose.

- Visually represented as a bar across the menu.
- Create using one of two methods.
  - Typing a single hyphen for the text.
  - Right-click on Menu Designer where you want a separator bar and choose Insert Separator.

Coding for Menu Items

- Double-click any menu item to open the item’s click event procedure.
- Write code for the click event procedure.



```
                        <li><a href="./about">About Us</a>
                            <div class="sf-mega">
                                <ul>
                                    <li><a href="./readmore.html">history</a></li>
                                    <li><a href="./readmore.html">news</a></li>
                                    <li class="last"><a href="./readmore.html">awards
                                    </a></li>
                                </ul>
                            </div>
                        </li>
                        <li><a href="./our_menu">Our Menu</a></li>
                        <li><a href="./news">News</a></li>
                        <li><a href="./contacts">Contacts</a></li>
                        <li><a href="./content">Content</a></li>
                    </ul>
                </nav>
            </div>
        </div>
    </div>
</header>
<article id="content" data-follow="location" data-type="switcher" data-behavior="scroll">
    <div data-id="" id="page1">
        <div class="bgHolder">
            <div class="bg1"></div>
```

Modifying Menu Items

- Right-click the Menu Bar to:
  - Delete.
  - Insert New.
  - Insert Separator.
  - Edit Names.
- Displays menu item Name property rather than Text property on form.
- Drag and Drop menu items to new locations.

Menu Properties

- Enabled property, True/False.
- Checked property, True/False.
  - Used to indicate current state of menu item that can be turned on and off.
- Create keyboard shortcuts.
  - In Properties window for menu item, select the Shortcut property.
  - Make choice from drop-down list.

Standards for Windows Menus

- Follow Windows standards for applications.
- Include keyboard access keys.
- Use standards for shortcut keys, if used.
- Place File menu at left end of menu bar and end File menu with Exit.
- Help, if included, is placed at right end of menu bar.

Database connection

- Applications communicate with a database, firstly, to retrieve the data stored there and present it in a user-friendly way, and secondly, to update the database by inserting, modifying and deleting data.
- Microsoft ActiveX Data Objects.Net (ADO.Net) is a model, a part of the .Net framework that is used by the .Net applications for retrieving, accessing and updating data.

ADO.Net Object Model

- ADO.Net object model is nothing but the structured process flow through various components.
- The data residing in a data store or database is retrieved through the data provider.
- Various components of the data provider retrieve data for the application and update data.
- An application accesses data either through a dataset or a data reader.
  - Datasets store data in a disconnected cache and the application retrieves data from it.
  - Data readers provide data to the application in a read-only and forward-only mode.

Data Provider

- A data provider is used for connecting to a database, executing commands and retrieving data, storing it in a dataset, reading the retrieved data and updating the database.
- The data provider in ADO.Net consists of the following four objects:

S.N	Objects & Description
1	Connection This component is used to set up a connection with a data source.
2	Command A command is a SQL statement or a stored procedure used to retrieve, insert, delete or modify data in a data source.
3	DataReader Data reader is used to retrieve data from a dta source in a resd-only mode
4	DataAdapter This is integral to the working of ADO.Net since data is transferred to and from a database through a data adapter. It retrieves data from a database into a dataset and updates the database. When changes are made to the dataset, the changes in the database are actually done by the data adapter.

There are following different types of data providers included in

ADO.Net

- The .Net Framework data provider for SQL Server
  - Provides access to Microsoft SQL Server.
- The .Net Framework data provider for OLE DB
  - Provides access to data sources exposed by using OLE DB.
- The .Net Framework data provider for ODBC
  - Provides access to data sources exposed by ODBC
- The .Net Framework data provider for Oracle
  - Provides access to Oracle data source.
- The EntityClient provider
  - Enables accessing data through Entity Data Model (EDM) applications

DataSet

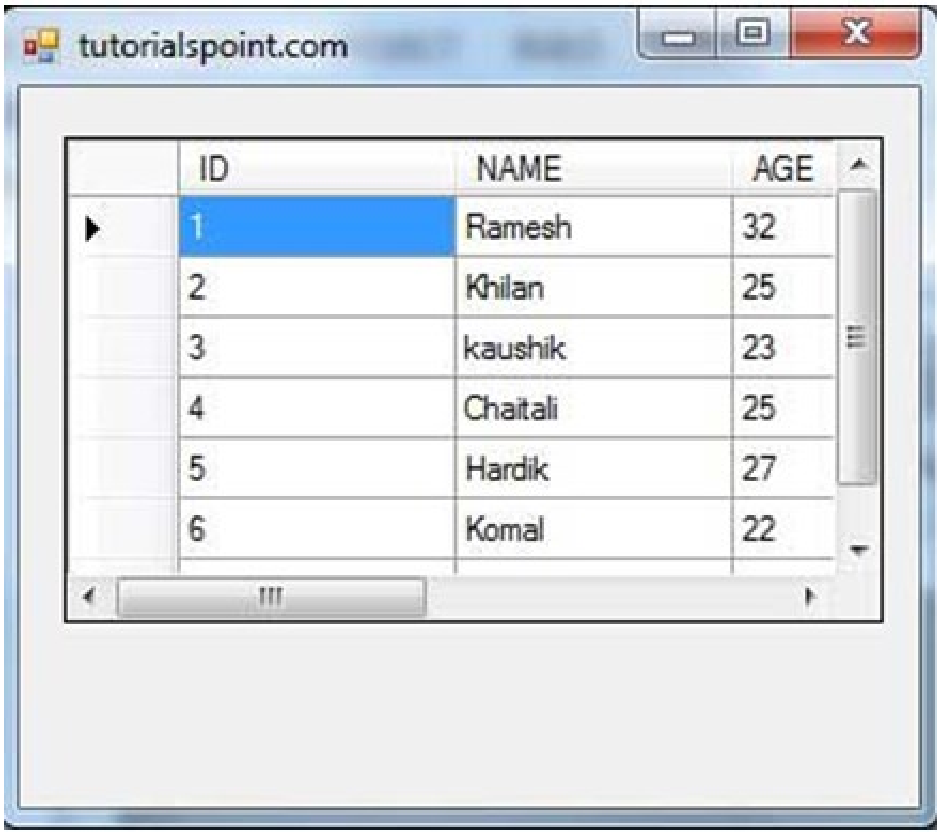
- DataSet is an in-memory representation of data.
- It is a disconnected, cached set of records that are retrieved from a database.
- When a connection is established with the database, the data adapter creates a dataset and stores data in it.
- After the data is retrieved and stored in a dataset, the connection with the database is closed.
- This is called the 'disconnected architecture'.
- The dataset works as a virtual database containing tables, rows, and columns.
- The DataSet class is present in the System.Data namespace.
  - DataTableCollection
    - It contains all the tables retrieved from the data source.
  - DataRelationCollection
    - It contains relationships and the links between tables in a data set.
  - ExtendedProperties
    - It contains additional information, like the SQL statement for retrieving data, time of retrieval, etc.
  - DataTable

        - It represents a table in the DataTableCollection of a dataset.
        - It consists of the DataRow and DataColumn objects.
        - The DataTable objects are case-sensitive.
    - DataRelation
        - It represents a relationship in the DataRelationshipCollection of the dataset.
        - It is used to relate two DataTable objects to each other through the DataColumn objects.
- The DataSet class is present in the System.Data namespace.
    - DataRowCollection
        - It contains all the rows in a DataTable.
    - DataView
        - It represents a fixed customized view of a DataTable for sorting, filtering, searching, editing and navigation.
    - PrimaryKey
        - It represents the column that uniquely identifies a row in a DataTable.
    - DataRow
        - It represents a row in the DataTable.
        - The DataRow object and its properties and methods are used to retrieve, evaluate, insert, delete, and update values in the DataTable.
        - The NewRow method is used to create a new row and the Add method adds a row to the table.
    - DataColumnCollection
        - It represents all the columns in a DataTable.
    - DataColumn
        - It consists of the number of columns that comprise a DataTable.

Connecting to a Database

- The .Net Framework provides two types of Connection classes:
    - SqlConnection - designed for connecting to Microsoft SQL Server.
    - OleDbConnection - designed for connecting to a wide range of databases, like Microsoft Access and Oracle.

- We have a table stored in Microsoft SQL Server, named Customers, in a database named testDB.
- Please consult 'SQL Server' tutorial for creating databases and database tables in SQL Server.
- Let us connect to this database.
- Take the following steps:
  - Select TOOLS -> Connect to Database.
  - Select a server name and the database name in the Add Connection dialog box.
  - Click on the Test Connection button to check if the connection succeeded.
  - Add a DataGridView on the form.
  - Click on the Choose Data Source combo box.
  - Click on the Add Project Data Source link.
  - This opens the Data Source Configuration Wizard.
  - Select Database as the data source type.
  - Choose DataSet as the database model.
  - Choose the connection already set up.
  - Save the connection string.
  - Choose the database object, Customers table in our example, and click the Finish button.
  - Select the Preview Data link to see the data in the Results grid:
  - When the application is run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window:



ID	NAME	AGE
1	Ramesh	32
2	Khilan	25
3	kaushik	23
4	Chaitali	25
5	Hardik	27
6	Komal	22

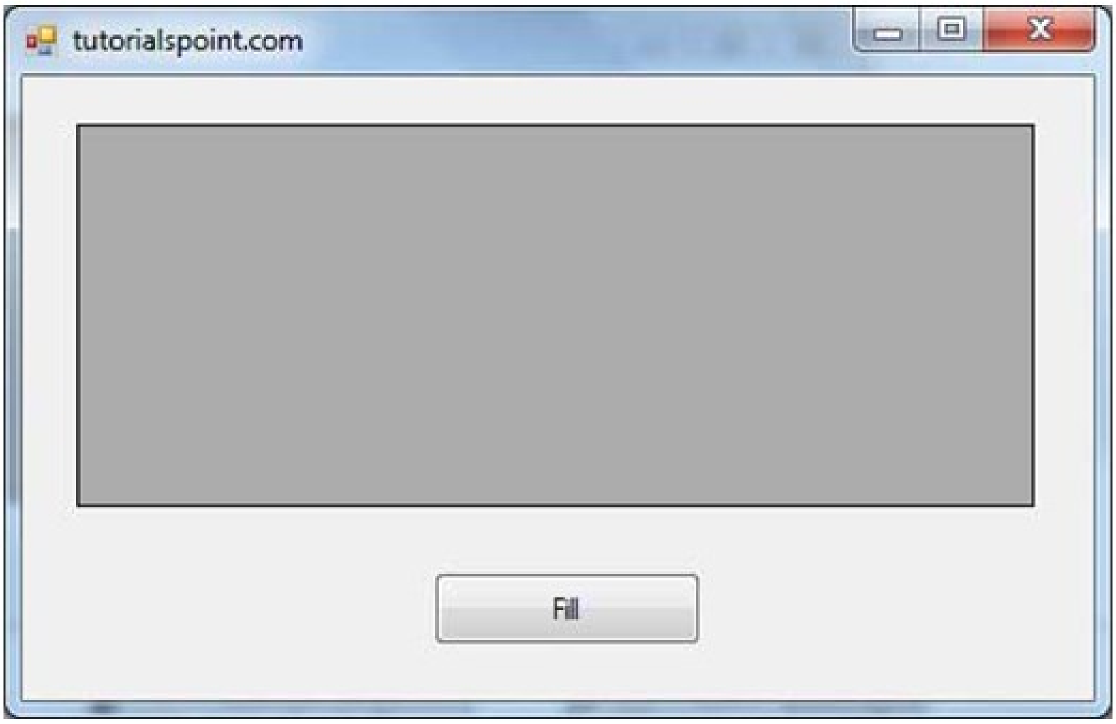
- In this example, let us access data in a DataGridView control using code.

```
Imports System.Data.SqlClient
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs) _
   Handles MyBase.Load
'TODO: This line of code loads data into the 'TestDBDataSet.CUSTOMERS' table.
You can move, or remove it, as needed.
      Me.CUSTOMERSTableAdapter.Fill(Me.TestDBDataSet.CUSTOMERS)
      ' Set the caption bar text of the form.
      Me.Text = "tutorialspoint.com"
   End Sub
Private Sub Button1_Click(sender As Object, e As EventArgs)
         Handles Button1.Click
      Dim connection As SqlConnection = New sqlconnection()
      connection.ConnectionString = "Data Source=KABIR-DESKTOP; _
         Initial Catalog=testDB;Integrated Security=True"
      connection.Open()
      Dim adp As SqlDataAdapter = New SqlDataAdapter _
      ("select * from Customers", connection)
      Dim ds As DataSet = New DataSet()
      adp.Fill(ds)
      DataGridView1.DataSource = ds.Tables(0)
   End Sub
End Class
```

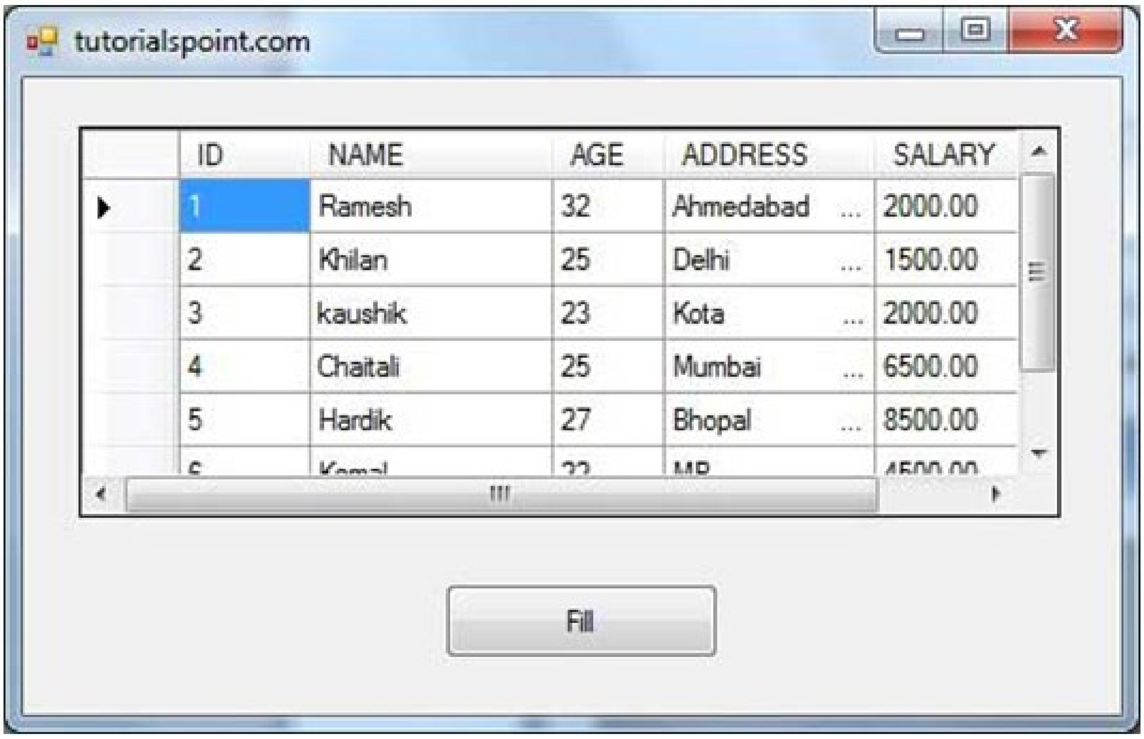
- Take the following steps:
   - Add a DataGridView control and a button in the form.
   - Change the text of the button control to ‘Fill’.
   - Double click the button control to add the required code for the Click event of the button, as

shown :

- When the above code is executed and run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window:



- Clicking the Fill button displays the table on the data grid view control:



	ID	NAME	AGE	ADDRESS	SALARY
▶	1	Ramesh	32	Ahmedabad ...	2000.00
	2	Khilan	25	Delhi ...	1500.00
	3	kaushik	23	Kota ...	2000.00
	4	Chaitali	25	Mumbai ...	6500.00
	5	Hardik	27	Bhopal ...	8500.00
	6	Komal	22	MP	4500.00

- So far, we have used tables and databases already existing in our computer.

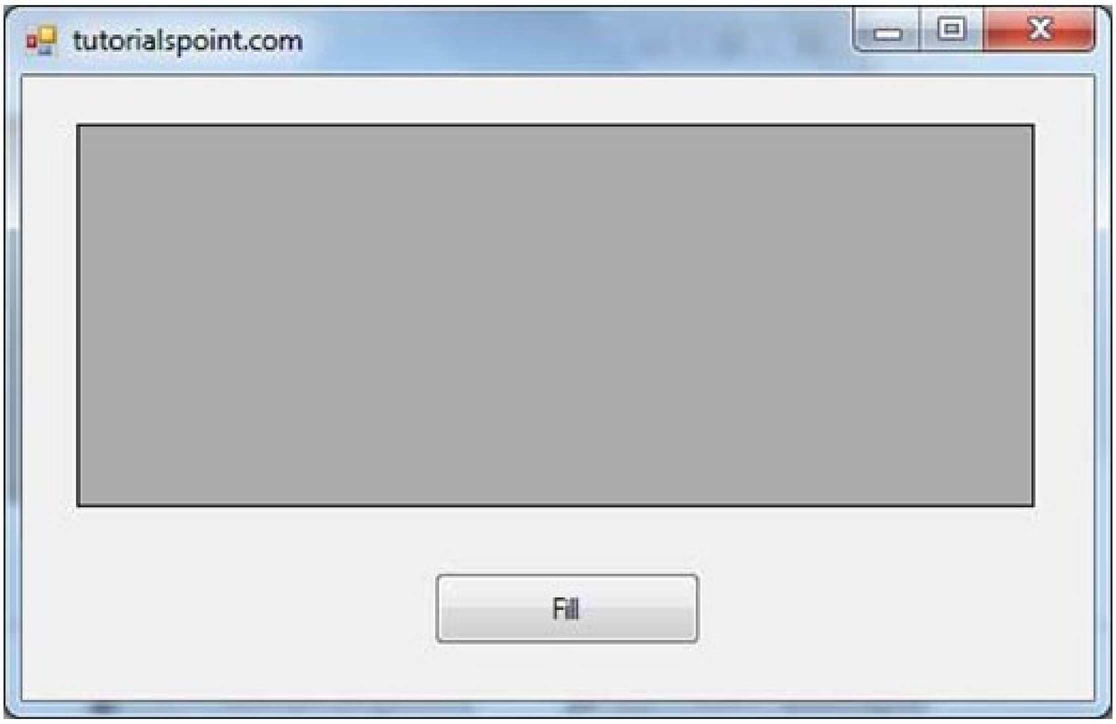
- In this example, we will create a table, add columns, rows and data into it and display the table using a DataGridView object.

```
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs)
          Handles MyBase.Load
      ' Set the caption bar text of the form.
      Me.Text = "tutorialspont.com"
   End Sub
   Private Function CreateDataSet() As DataSet
      'creating a DataSet object for tables
      Dim dataset As DataSet = New DataSet()
      ' creating the student table
      Dim Students As DataTable = CreateStudentTable()
      dataset.Tables.Add(Students)
      Return dataset
   End Function
   Private Function CreateStudentTable() As DataTable
      Dim Students As DataTable
      Students = New DataTable("Student")
      ' adding columns
      AddNewColumn(Students, "System.Int32", "StudentID")
      AddNewColumn(Students, "System.String", "StudentName")
      AddNewColumn(Students, "System.String", "StudentCity")
      ' adding rows
      AddNewRow(Students, 1, "Zara Ali", "Kolkata")
      AddNewRow(Students, 2, "Shreya Sharma", "Delhi")
      AddNewRow(Students, 3, "Rini Mukherjee", "Hyderabad")
      AddNewRow(Students, 4, "Sunil Dubey", "Bikaner")
      AddNewRow(Students, 5, "Rajat Mishra", "Patna")
      Return Students
   End Function
   Private Sub AddNewColumn(ByRef table As DataTable, _
   ByVal columnType As String, ByVal columnName As String)
      Dim column As DataColumn = _
       table.Columns.Add(columnName, Type.GetType(columnType))
   End Sub

    'adding data into the table
    Private Sub AddNewRow(ByRef table As DataTable,
 ByRef id As Integer,_
    ByRef name As String, ByRef city As String)
       Dim newrow As DataRow = table.NewRow()
       newrow("StudentID") = id
       newrow("StudentName") = name
       newrow("StudentCity") = city
       table.Rows.Add(newrow)
    End Sub
    Private Sub Button1_Click(sender As Object, e As EventArgs)
           Handles Button1.Click
       Dim ds As New DataSet
       ds = CreateDataSet()
       DataGridView1.DataSource = ds.Tables("Student")
    End Sub
 End Class
```

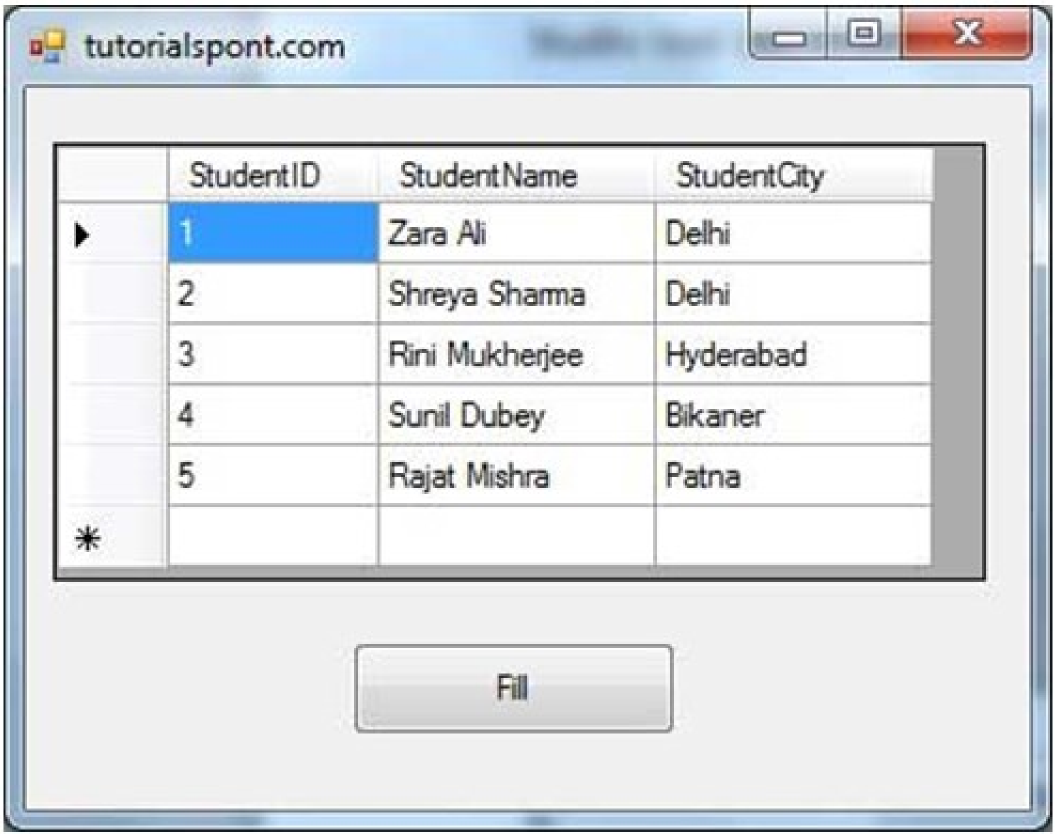
- Take the following steps:

- Add a DataGridView control and a button in the form.
- Change the text of the button control to 'Fill'.
- Add the following code in the code editor.

- When the above code is executed and run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window:



- Clicking the Fill button displays the table on the data grid view control:



	StudentID	StudentName	StudentCity
▶	1	Zara Ali	Delhi
	2	Shreya Sharma	Delhi
	3	Rini Mukherjee	Hyderabad
	4	Sunil Dubey	Bikaner
	5	Rajat Mishra	Patna
*			

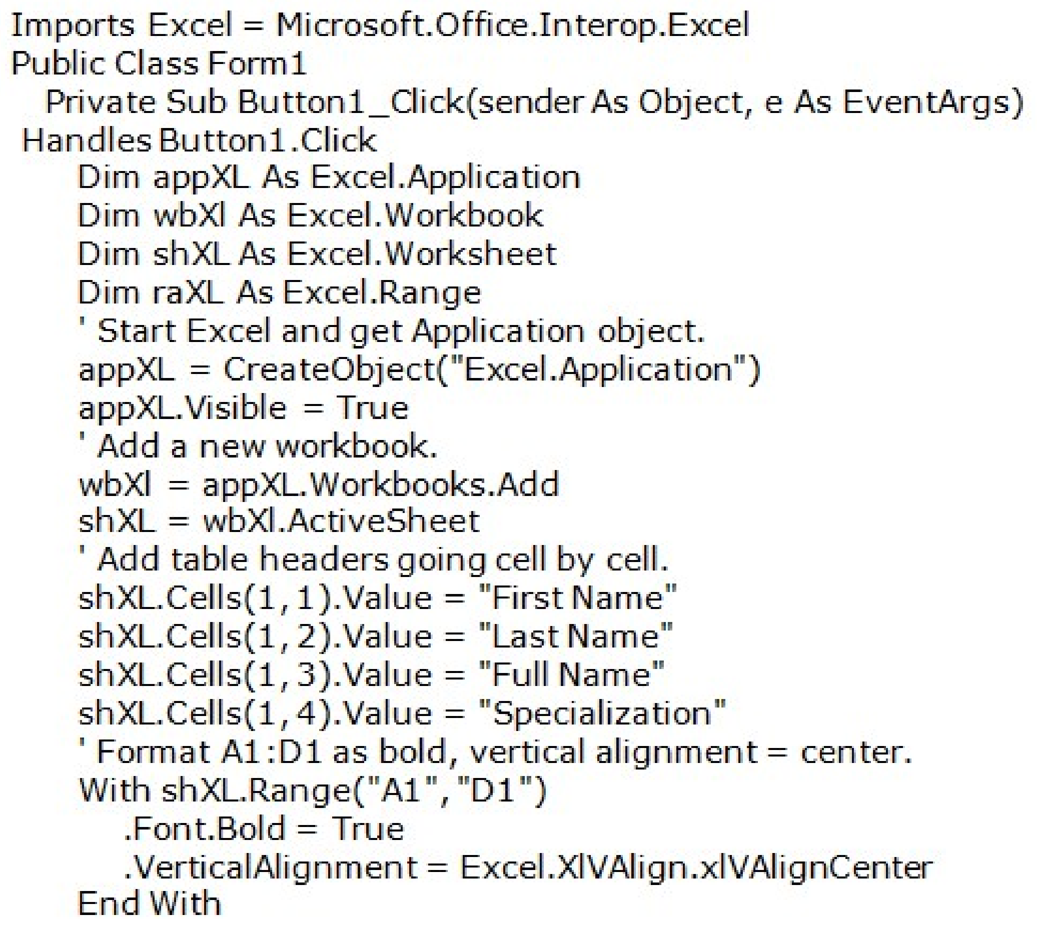
- VB.Net provides support for interoperability between the COM object model of Microsoft Excel

2010 and your application.

- To avail this interoperability in your application, you need to import the namespace Microsoft.Office.Interop.Excel in your Windows Form Application.

Creating an Excel Application from VB.Net.

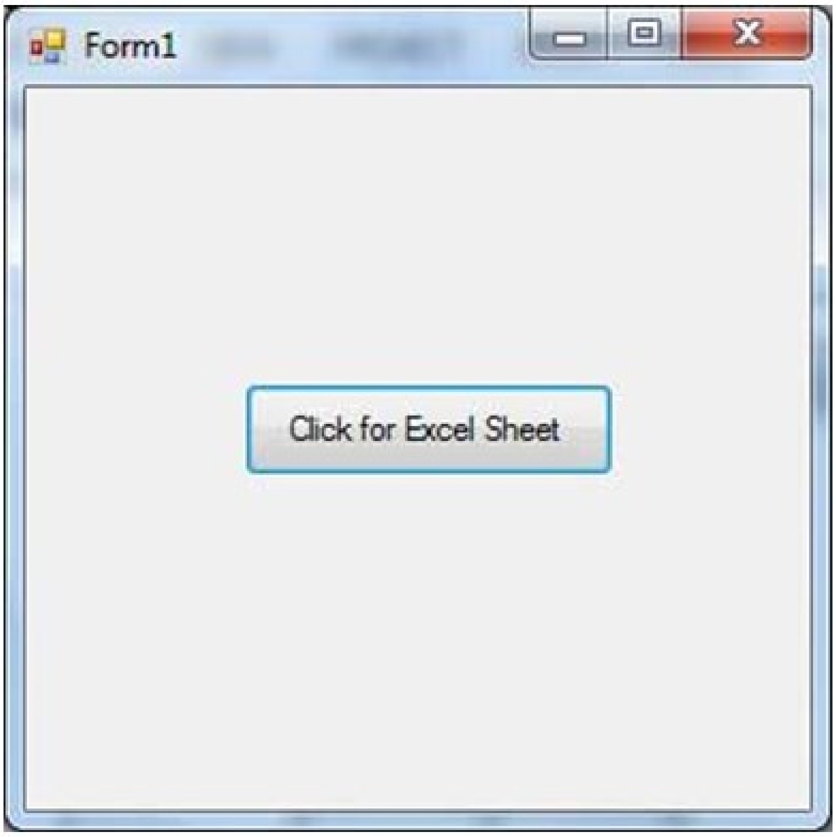
- Let's start with creating a Window Forms Application by following the following steps in Microsoft Visual Studio: File -> New Project -> Windows Forms Applications.
- Finally, select OK, Microsoft Visual Studio creates your project and displays Form1.
- Insert a Button control Button1 in the form.
- Add a reference to Microsoft Excel Object Library to your project.
  - Select Add Reference from the Project Menu.
  - On the COM tab, locate Microsoft Excel Object Library and then click Select.
  - Click OK.
- Double click the code window and populate the Click event of Button1, as shown .



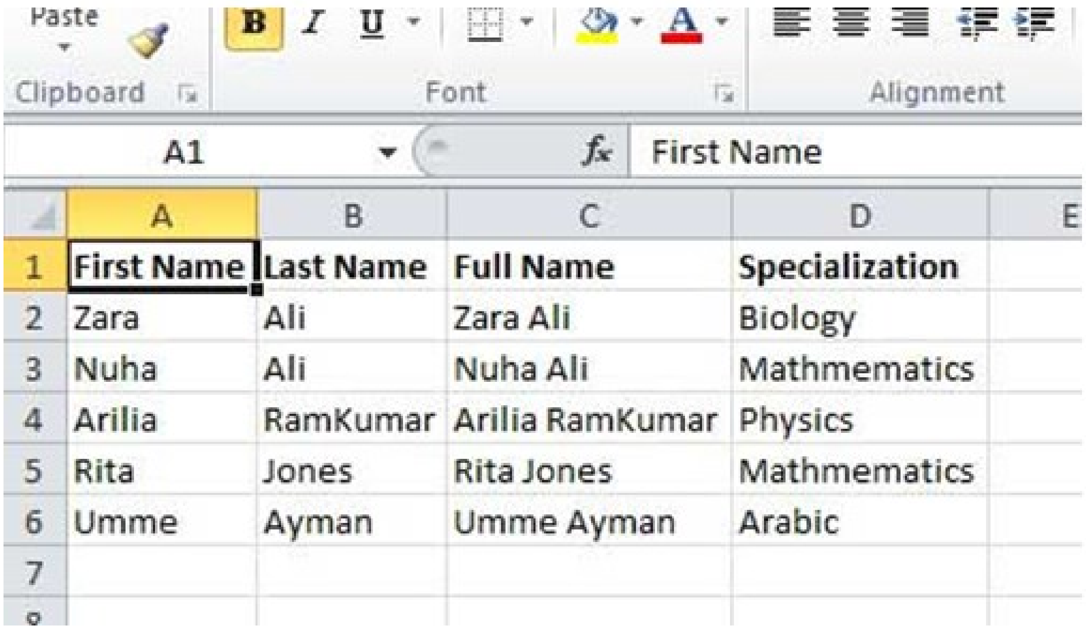
```
Imports Excel = Microsoft.Office.Interop.Excel
Public Class Form1
   Private Sub Button1_Click(sender As Object, e As EventArgs)
 Handles Button1.Click
      Dim appXL As Excel.Application
      Dim wbXl As Excel.Workbook
      Dim shXL As Excel.Worksheet
      Dim raXL As Excel.Range
      ' Start Excel and get Application object.
      appXL = CreateObject("Excel.Application")
      appXL.Visible = True
      ' Add a new workbook.
      wbXl = appXL.Workbooks.Add
      shXL = wbXl.ActiveSheet
      ' Add table headers going cell by cell.
      shXL.Cells(1, 1).Value = "First Name"
      shXL.Cells(1, 2).Value = "Last Name"
      shXL.Cells(1, 3).Value = "Full Name"
      shXL.Cells(1, 4).Value = "Specialization"
      ' Format A1:D1 as bold, vertical alignment = center.
      With shXL.Range("A1", "D1")
          .Font.Bold = True
          .VerticalAlignment = Excel.XlVAlign.xlVAlignCenter
      End With
```

```
' Create an array to set multiple values at once.
Dim students(5, 2) As String
students(0, 0) = "Zara"
students(0, 1) = "Ali"
students(1, 0) = "Nuha"
students(1, 1) = "Ali"
students(2, 0) = "Arilia"
students(2, 1) = "RamKumar"
students(3, 0) = "Rita"
students(3, 1) = "Jones"
students(4, 0) = "Umme"
students(4, 1) = "Ayman"
' Fill A2:B6 with an array of values (First and Last Names).
shXL.Range("A2", "B6").Value = students
 ' Fill C2:C6 with a relative formula (=A2 & " " & B2).
raXL = shXL.Range("C2", "C6")
raXL.Formula = "=A2 & "" "" & B2"
 ' Fill D2:D6 values.
With shXL
    .Cells(2, 4).Value = "Biology"
    .Cells(3, 4).Value = "Mathmematics"
    .Cells(4, 4).Value = "Physics"
    .Cells(5, 4).Value = "Mathmematics"
    .Cells(6, 4).Value = "Arabic"
End With
' AutoFit columns A:D.
raXL = shXL.Range("A1", "D1")
raXL.EntireColumn.AutoFit()

      ' Make sure Excel is visible and give the user control
     ' of Excel's lifetime.
     appXL.Visible = True
     appXL.UserControl = True
      ' Release object references.
     raXL = Nothing
     shXL = Nothing
     wbXl = Nothing
     appXL.Quit()
     appXL = Nothing
     Exit Sub
Err_Handler:
     MsgBox(Err.Description, vbCritical, "Error: " & Err.Number)
  End Sub
End Class
```

- When the code is executed and run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window:



- Clicking on the Button would display the following excel sheet.



	A	B	C	D	E
1	**First Name**	**Last Name**	**Full Name**	**Specialization**	
2	Zara	Ali	Zara Ali	Biology	
3	Nuha	Ali	Nuha Ali	Mathmematics	
4	Arilia	RamKumar	Arilia RamKumar	Physics	
5	Rita	Jones	Rita Jones	Mathmematics	
6	Umme	Ayman	Umme Ayman	Arabic	
7					

- You will be asked to save the workbook:

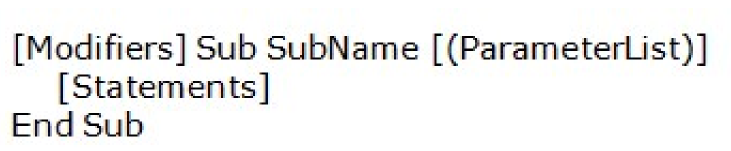
UNIT - 8
Advanced Programming Constructs

Procedures

- A procedure is a group of statements that together perform a task when called.
- After the procedure is executed, the control returns to the statement calling the procedure.
- VB.Net has two types of procedures:
  - Functions,
  - Sub procedures or Subs.
- Functions return a value, whereas Subs do not return a value.

Sub-procedures

- The Sub statement is used to declare the name, parameter and the body of a sub procedure.
- The syntax for the Sub statement is:



```
[Modifiers] Sub SubName [(ParameterList)]
    [Statements]
End Sub
```

- Where,
  - Modifiers: specify the access level of the procedure; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
  - SubName: indicates the name of the Sub.
  - ParameterList: specifies the list of the parameters.

Example for sub- procedure

- The following example demonstrates a Sub procedure Calculate Pay that takes two parameters hours and wages and displays the total pay of an employee:

```
Module mysub
   Sub CalculatePay(ByVal hours As Double, ByVal wage As Decimal)
      'local variable declaration
      Dim pay As Double
      pay = hours * wage
      Console.WriteLine("Total Pay: {0:C}", pay)
   End Sub
   Sub Main()
      'calling the CalculatePay Sub Procedure
      CalculatePay(25, 10)
      CalculatePay(40, 20)
      CalculatePay(30, 27.5)
      Console.ReadLine()
   End Sub
End Module
```

- When the above code is compiled and executed, it produces the following result:

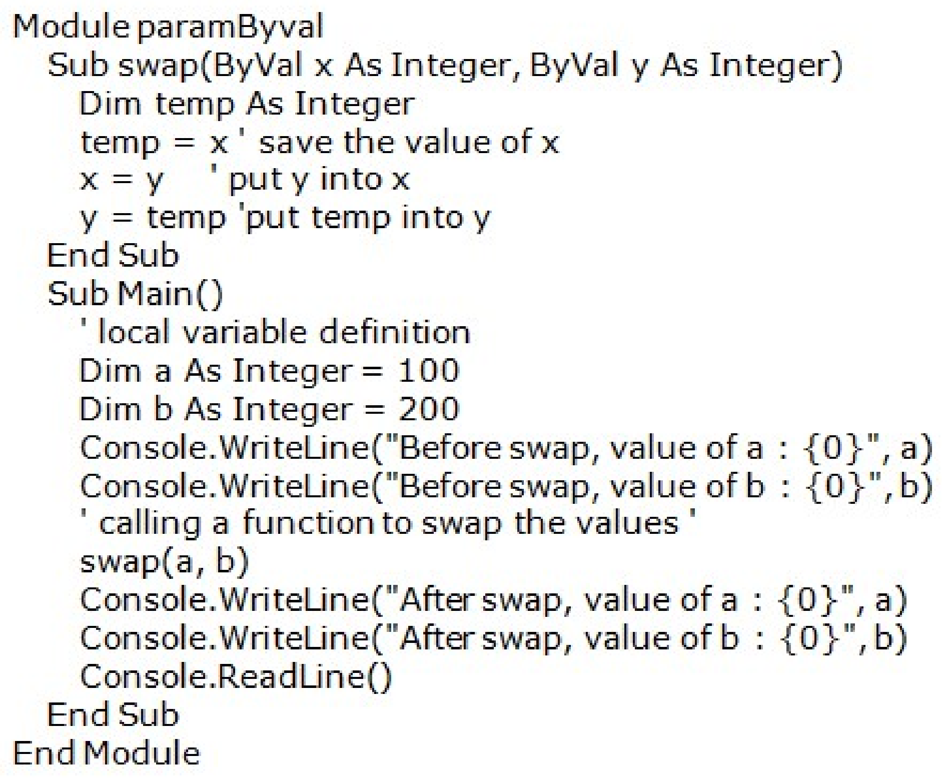
```
Total Pay: $250.00
Total Pay: $800.00
Total Pay: $825.00
```

- The example program shows how to calculate pay by taking the parameters such as hours and wages using sub procedure.
  - Here the procedure Calculate pay is the sub procedure, where local variables are declared.
  - At the end of the sub procedure the main function is stated, is used to call the procedure by different parameters.

Passing Parameters by Value

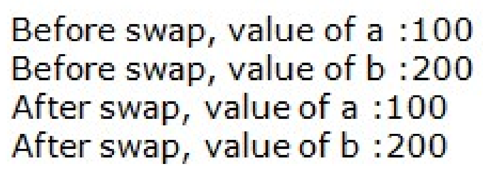
- This is the default mechanism for passing parameters to a method.
- In this mechanism, when a method is called, a new storage location is created for each value parameter.
- The values of the actual parameters are copied into them.
- So, the changes made to the parameter inside the method have no effect on the argument.
- In VB.Net, you declare the reference parameters using the ByVal keyword.
- The following example demonstrates the concept:

Program



```
Module paramByval
   Sub swap(ByVal x As Integer, ByVal y As Integer)
      Dim temp As Integer
      temp = x ' save the value of x
      x = y    ' put y into x
      y = temp 'put temp into y
   End Sub
   Sub Main()
      ' local variable definition
      Dim a As Integer = 100
      Dim b As Integer = 200
      Console.WriteLine("Before swap, value of a : {0}", a)
      Console.WriteLine("Before swap, value of b : {0}", b)
      ' calling a function to swap the values '
      swap(a, b)
      Console.WriteLine("After swap, value of a : {0}", a)
      Console.WriteLine("After swap, value of b : {0}", b)
      Console.ReadLine()
   End Sub
End Module
```

Output



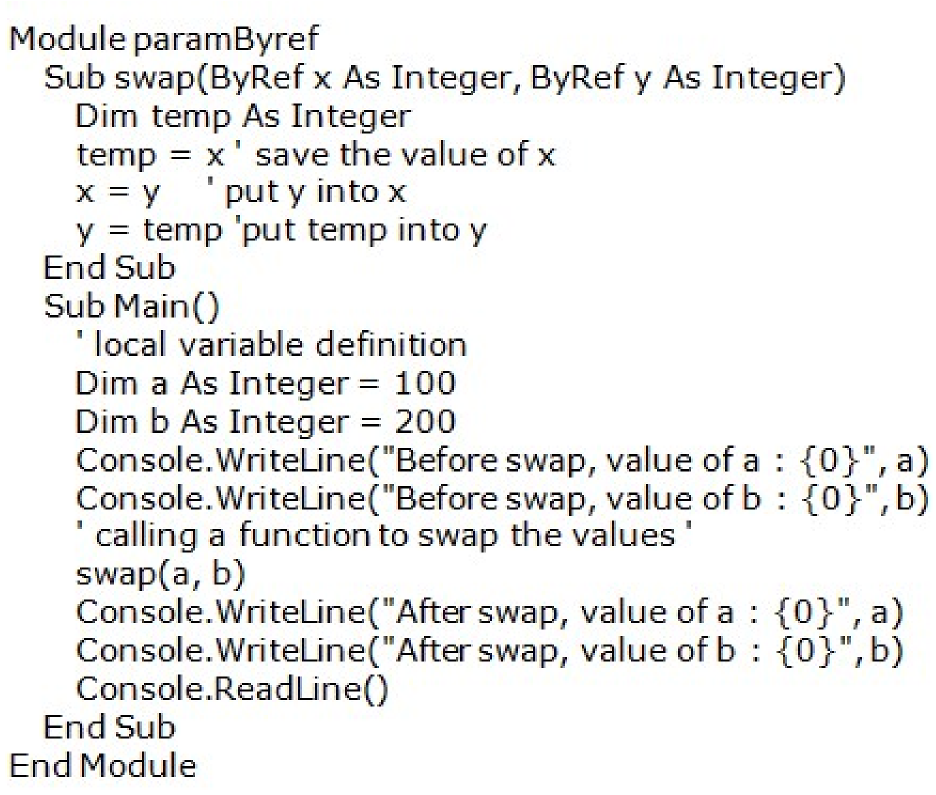
```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

- Passing parameters by values is the default method in any language.
- Here the parameter is passed directly to the function.
- It shows that there is no change in the values though they had been changed inside the function.

Passing Parameters by Reference

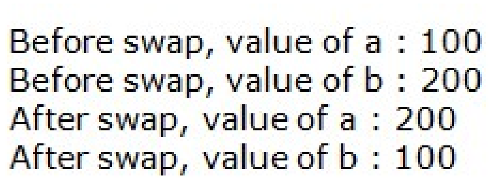
- A reference parameter is a reference to a memory location of a variable.
- When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- In VB.Net, you declare the reference parameters using the ByRef keyword.
- The following example demonstrates this:

Program



```
Module paramByref
   Sub swap(ByRef x As Integer, ByRef y As Integer)
      Dim temp As Integer
      temp = x ' save the value of x
      x = y    ' put y into x
      y = temp 'put temp into y
   End Sub
   Sub Main()
      ' local variable definition
      Dim a As Integer = 100
      Dim b As Integer = 200
      Console.WriteLine("Before swap, value of a : {0}", a)
      Console.WriteLine("Before swap, value of b : {0}", b)
      ' calling a function to swap the values '
      swap(a, b)
      Console.WriteLine("After swap, value of a : {0}", a)
      Console.WriteLine("After swap, value of b : {0}", b)
      Console.ReadLine()
   End Sub
End Module
```

Output



```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100
```

- The example program shows how to the procedure is called by passing parameters by reference.
- In VB.Net, you declare the reference parameters using the ByRef keyword.
- The parameter passing is located by same memory location.
- The pointer concept is used in this method.

Function

- Function is a group of statements.
- It should be executed once it called.
- The function call is referred by a value or by a reference.
- The Function statement is used to declare the name, parameter and the body of a function.
- The syntax for the Function statement is:

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
    [Statements]
End Function
```

- Where,
  - ➤ Modifiers: specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
  - ➤ FunctionName: indicates the name of the function.
  - ➤ ParameterList: specifies the list of the parameters.
  - ➤ ReturnType: specifies the data type of the variable the function returns.

Example for function

- Here a example for function.
- Following code snippet shows a function FindMax that takes two integer values and returns the larger of the two.
- The function name is FINDMAX, is called by the value.
- The program is the example of finding biggest number among the given number.

```
Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
   Dim result As Integer
   If (num1 > num2) Then
      result = num1
   Else
      result = num2
   End If
   FindMax = result
End Function
```

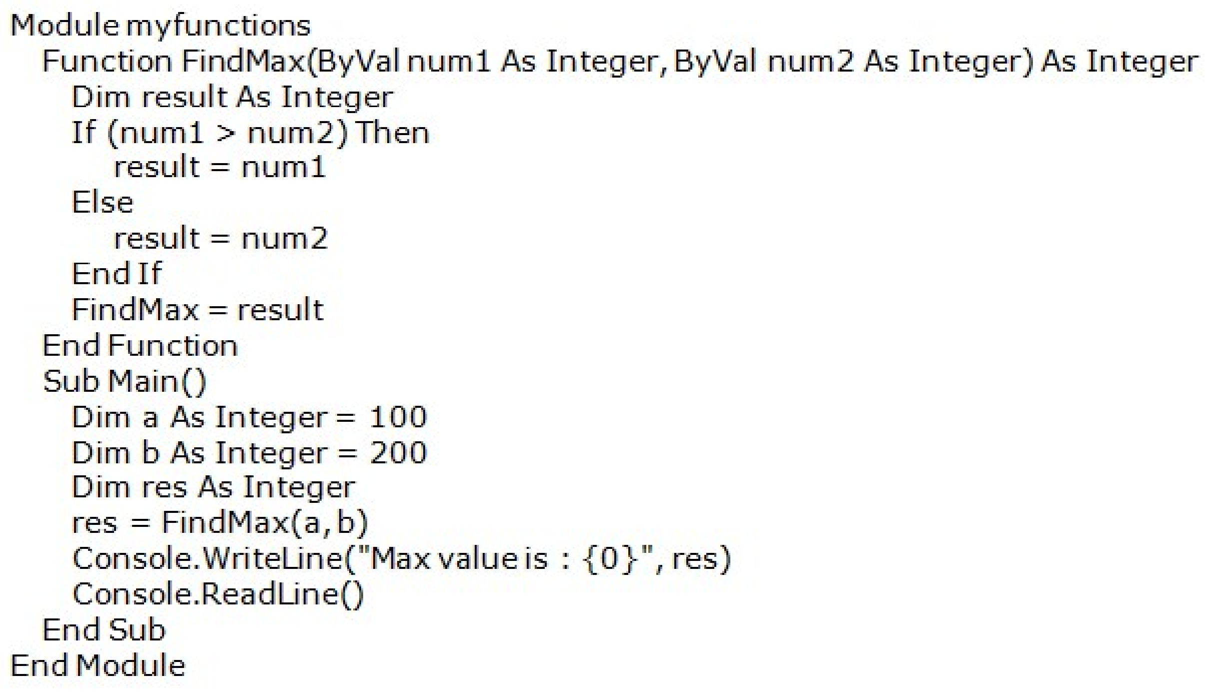
Function Returning a Value

In VB.Net, a function can return a value to the calling code in two ways:

- ➢ By using the return statement.

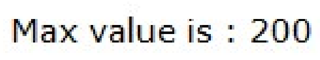
  ➢ By assigning the value to the function name.

- The following example demonstrates using the FindMax function:



```
Module myfunctions
   Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
      Dim result As Integer
      If (num1 > num2) Then
          result = num1
      Else
          result = num2
      End If
      FindMax = result
   End Function
   Sub Main()
      Dim a As Integer = 100
      Dim b As Integer = 200
      Dim res As Integer
      res = FindMax(a, b)
      Console.WriteLine("Max value is : {0}", res)
      Console.ReadLine()
   End Sub
End Module
```

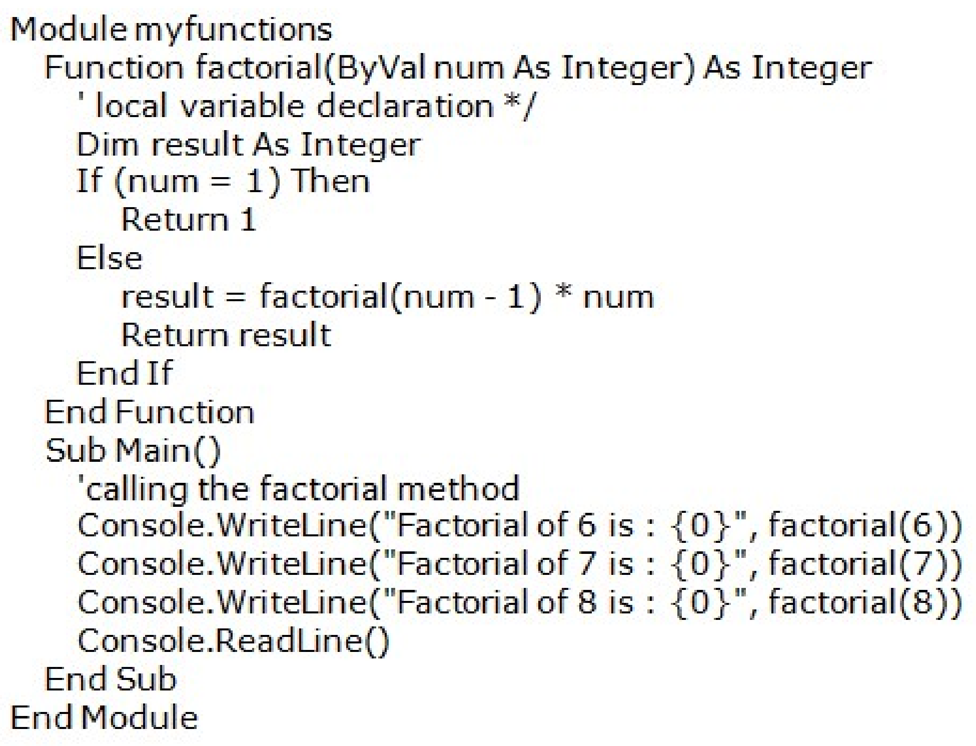
- The example show how to return a value from a function.
- For returning the sub function is executed.
- When the above code is compiled and executed, it produces the following result:



```
Max value is : 200
```

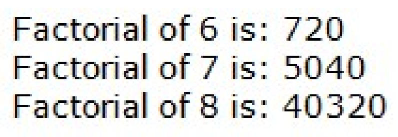
Recursive Function

- A function can call itself.
- This is known as recursion.
- Following is an example that calculates factorial for a given number using a recursive function:



```
Module myfunctions
   Function factorial(ByVal num As Integer) As Integer
      ' local variable declaration */
      Dim result As Integer
      If (num = 1) Then
          Return 1
      Else
          result = factorial(num - 1) * num
          Return result
      End If
   End Function
   Sub Main()
      'calling the factorial method
      Console.WriteLine("Factorial of 6 is : {0}", factorial(6))
      Console.WriteLine("Factorial of 7 is : {0}", factorial(7))
      Console.WriteLine("Factorial of 8 is : {0}", factorial(8))
      Console.ReadLine()
   End Sub
End Module
```

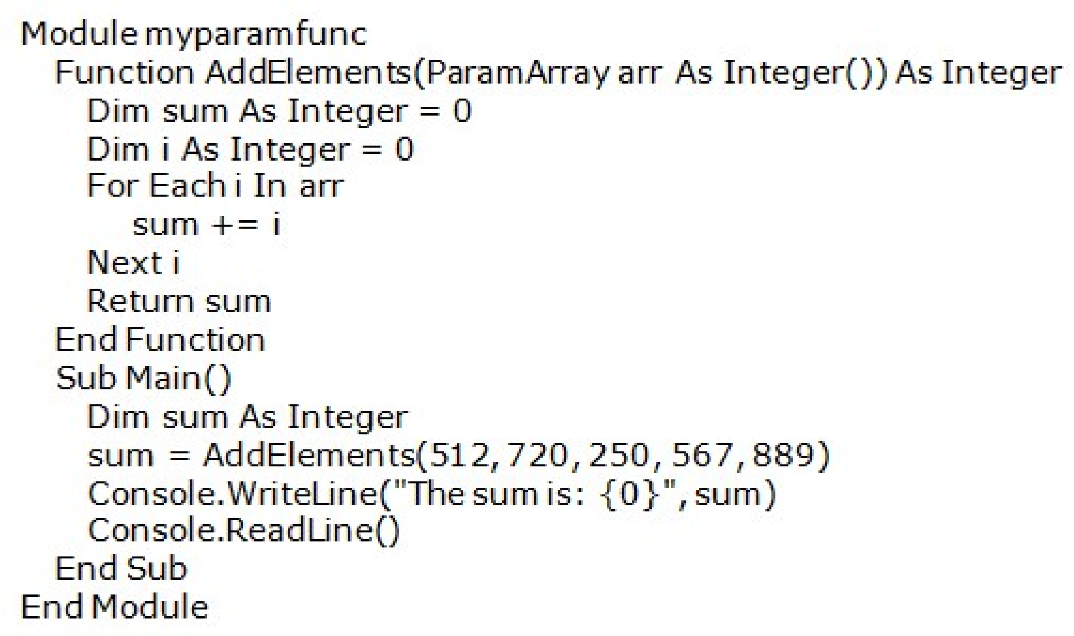
- If a function call itself means that function is called as recursive function.
- Here the example shows the recursive function by calculating the factoid for the given number.
- The result for the program is:



```
Factorial of 6 is: 720
Factorial of 7 is: 5040
Factorial of 8 is: 40320
```

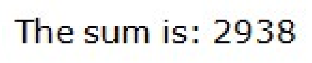
Param Arrays

- At times, while declaring a function or sub procedure, you are not sure of the number of arguments passed as a parameter.
- VB.Net param arrays come into help at these times.
- The following example demonstrates this:



```
Module myparamfunc
   Function AddElements(ParamArray arr As Integer()) As Integer
      Dim sum As Integer = 0
      Dim i As Integer = 0
      For Each i In arr
          sum += i
      Next i
      Return sum
   End Function
   Sub Main()
      Dim sum As Integer
      sum = AddElements(512, 720, 250, 567, 889)
      Console.WriteLine("The sum is: {0}", sum)
      Console.ReadLine()
   End Sub
End Module
```

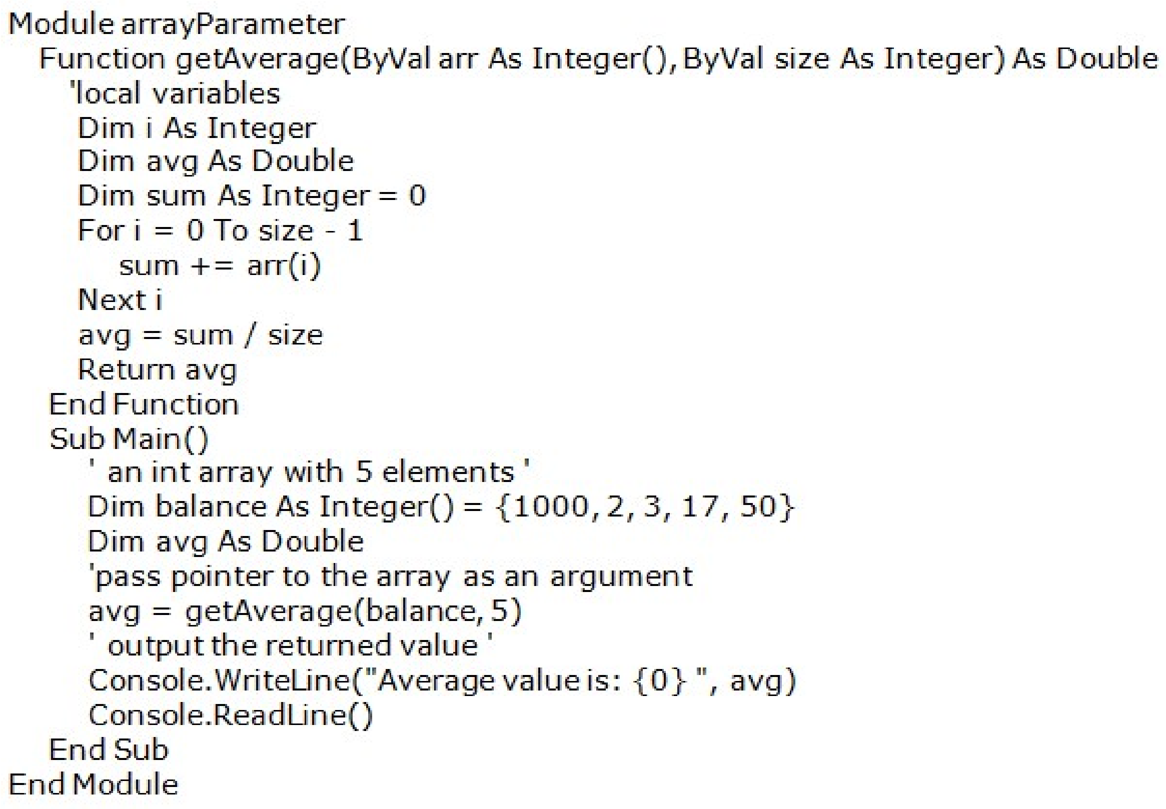
- Param Array is also be called as dynamic array.
- When we dont know the number of arguments passed as a parameter when declaring an array we can use param array.
- The example program shows how to use param array.
- The result of this program is:



```
The sum is: 2938
```

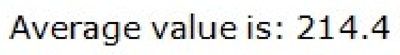
Passing Arrays as Function Arguments

- You can pass an array as a function argument in VB.Net.
- The following example demonstrates this:



```
Module arrayParameter
   Function getAverage(ByVal arr As Integer(), ByVal size As Integer) As Double
      'local variables
       Dim i As Integer
       Dim avg As Double
       Dim sum As Integer = 0
       For i = 0 To size - 1
           sum += arr(i)
       Next i
       avg = sum / size
       Return avg
    End Function
    Sub Main()
        ' an int array with 5 elements '
        Dim balance As Integer() = {1000, 2, 3, 17, 50}
        Dim avg As Double
        'pass pointer to the array as an argument
        avg = getAverage(balance, 5)
        ' output the returned value '
        Console.WriteLine("Average value is: {0} ", avg)
        Console.ReadLine()
    End Sub
End Module
```

- The special in VB is passing an array as an function argument for a function or procedure.
- The example shows an syntax for passing an array as an argument.
- The result for this program is:



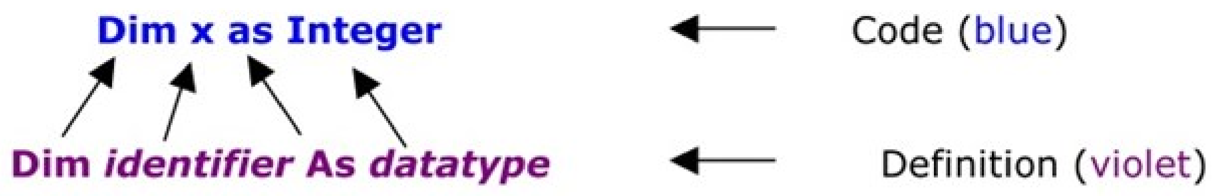
```
Average value is: 214.4
```

Modules

- A VB .Net program is made up of variables and programming statements enclosed in a module.
- This is a big step for someone to understand.
- The simplest way to understand this is just to realize thatvariables and programming statements need to be grouped together by something.
- This something is called a module.
- VB program is set of two modules.
- They are,
  - Variables
  - Programming statement.
- A variable is nothing but a name given to a storage area that our programs can manipulate.
- Each variable in VB.Net has a specific type, which determines the size and layout of the variable 's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

Declaring Variables

- Before you can use a variable in a program it has to be declared.
- To declare a variable you use the Dim keyword and specify a data type list in the preceding data type table.



- In example x is declared as a variable of Integer data type.
- Dim means declare variable, as means specify a data type.
  - Dim number As Integer
    Dim quantity As Integer = 100
    Dim message As String = "Hello!"
- The Dim statement is used for variable declaration and storage allocation for one or more variables.
- The Dim statement is used at module, class, structure, procedure or block level.

Syntax for variable declaration in VB.Net is

```
[ < attributelist> ] [ accessmodifier ] [[ Shared ] [ Shadows ] | [ Static ]]
[ ReadOnly ] Dim [ WithEvents ] variablelist
```

- Explanation:
  - ➢ attributelist is a list of attributes that apply to the variable. Optional.
  - ➢ accessmodifier defines the access levels of the variables, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
  - ➢ Shared declares a shared variable, which is not associated with any specific instance of a class or structure, rather available to all the instances of the class or structure. Optional.
  - ➢ Shadows indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
  - ➢ Static indicates that the variable will retain its value, even when the after termination of the procedure in which it is declared. Optional.
  - ➢ ReadOnly means the variable can be read, but not written. Optional.
  - ➢ WithEvents specifies that the variable is used to respond to events raised by the instance assigned to the variable.
  - ➢ Variablelist provides the list of variables declared.

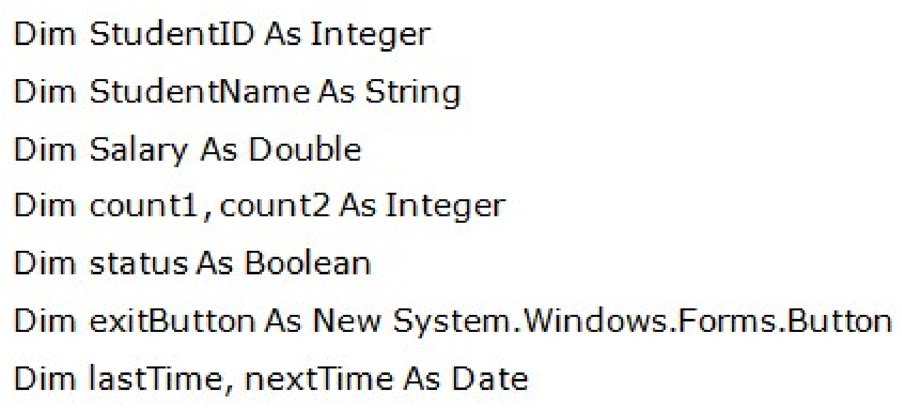
Each variable in the variable list has the following syntax and parts

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

- ➢ variablename: is the name of the variable
- ➢ boundslist: optional. It provides list of bounds of each dimension of an array variable.
- ➢ New: optional. It creates a new instance of the class when the Dim statement runs.
- ➢ datatype: Required if Option Strict is On.
- ➢ It specifies the data type of the variable.
- ➢ initializer: Optional if New is not specified.
- ➢ Expression that is evaluated and assigned to the variable when it is created.

Some valid variable declarations along with their definition are shown here

- The examples shows the valid declarations , VB .net.



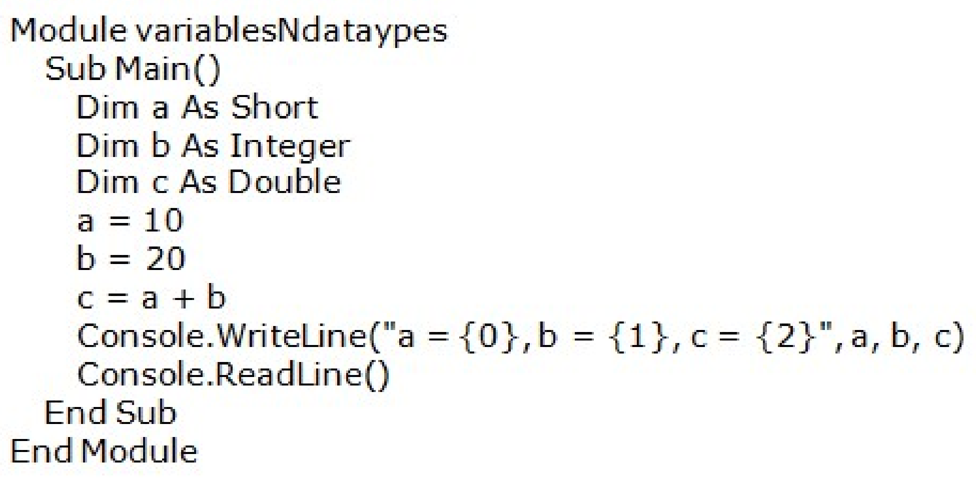
```
Dim StudentID As Integer
Dim StudentName As String
Dim Salary As Double
Dim count1, count2 As Integer
Dim status As Boolean
Dim exitButton As New System.Windows.Forms.Button
Dim lastTime, nextTime As Date
```

Variable Initialization in VB.Net

- Variables are initialized (assigned a value) with an equal sign followed by a constant expression.
- The general form of initialization is:
  - variable_name= value;
- For example,
  - Dim pi AsDouble.
  - pi =3.14159.
    - To initialize a variable we first found the sign, and then the constant.
    - The syntax shows how to initialization the variables in VB.net.
    - This is the common method for all high level language.
- You can initialize a variable at the time of declaration as follows:
  - DimStudentIDAsInteger=100.
  - DimStudentNameAsString="Bill Smith".

Example for Variable declaration and Variable Initialization

- The program shows how to declare the variables and how to initial the variables.



```
Module variablesNdataypes
   Sub Main()
      Dim a As Short
      Dim b As Integer
      Dim c As Double
      a = 10
      b = 20
      c = a + b
      Console.WriteLine("a = {0},b = {1}, c = {2}", a, b, c)
      Console.ReadLine()
   End Sub
End Module
```

- When the above code is compiled and executed, it produces the following result:

```
a = 10, b = 20, c = 30
```

Accepting Values from User

- Dim message As String
  message = Console.ReadLine
- The Console class in the System namespace provides a function ReadLine for accepting input from the user and store it into a variable.
- For example,
  - It is hard to give the input by the program, Like those programs are static programs.
  - No different output would be produced.
  - For the reason the syntax as been created to give the input on running time.
  - The example shows how to declare a variable for run time initialization.

```
Module variablesNdataypes
   Sub Main()
      Dim message As String
      Console.Write("Enter message: ")
      message = Console.ReadLine
      Console.WriteLine()
      Console.WriteLine("Your Message: {0}", message)
      Console.ReadLine()
   End Sub
End Module
```

- When the above code is compiled and executed, it produces the following result.

```
Enter message: Hello World
Your Message: Hello World
```

Lvalues and Rvalues

- There are two kinds of expressions:
  - Lvalue
    - An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
  - Rvalue
    - An expression that is an rvalue may appear on the right- but not left-hand side of an

assignment.

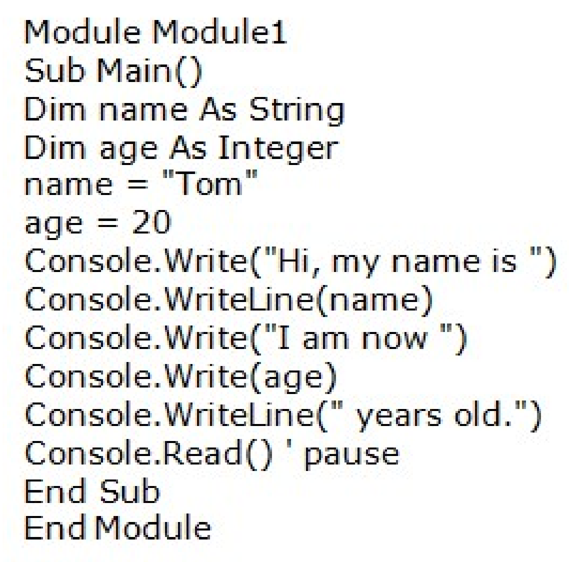
- Example :
  - Dim g As Integer = 20

VB .Net Sample Program

- We are now ready to write our first VB .Net program.
- In this sample program we will initialize some variables and print their values to the screen.
- Programming languages use programming statement to do things.
- A statement includes command words called keywords and data values.
- Our statement will be a statement that allows you to print out messages on the computer screen.
- Messages are String constants enclosed by " double quotes " like “hello”.
- This is a Console program, meaning the output is sent to the screen, and the user types in the input responses.
- A Console program is a little different than a GUI program.
- In our program we first declare a name and age variable.
  - Dim name As String.
  - Dim age As Integer.
- We then initialize name to “Tom” and age to 20.
  - name = "Tom".
  - age = 20.
- Now we want to print the name and age to the screen.
- We use we use the Console.Write and Console.writeLine statements to do this.
- Here we print "Hi, my name is Tom" message on the computer screen.

Example program

- You will notice it is enclosed in a Module and a Sub called Main.
- The Sub encloses the programming statements. The Module encloses the Sub Main.
- A module can have many Subs.
- The Main Sub is the first sub to start executing in a module contained in a program.



```
Module Module1
Sub Main()
Dim name As String
Dim age As Integer
name = "Tom"
age = 20
Console.Write("Hi, my name is ")
Console.WriteLine(name)
Console.Write("I am now ")
Console.Write(age)
Console.WriteLine(" years old.")
Console.Read() ' pause
End Sub
End Module
```

Step-by-step instructions to run the program

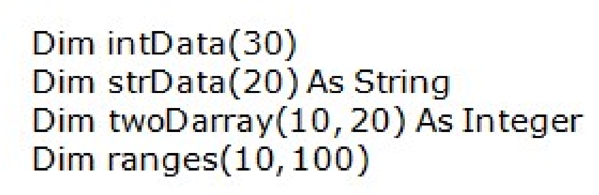
- Before you can write and run this program you first need to create a new Project.
- Make sure you have Visual Basic .Net installed on your computer.
- We are using Visual Basic .Net version 2005.
  - Step1: Open up Visual Basic .Net, from the File menu select New then Project.
  - Step2: Right-click on Form1 in the Project Explorer window and select Remove Form1 from the pop up window. (this is an optional step)
  - Step3: You should make a directory for your programs using Windows Explorer and the use the browse button to select that directory. Select Console Application and then type in the name of the program.
  - Step4: Now type in the program.
  - Step5: To run the program select start from the Debug menu.

Array

- Array is collection of data stored in the continuous order.
- It is a linear Data structure to access the data.
- An array stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.

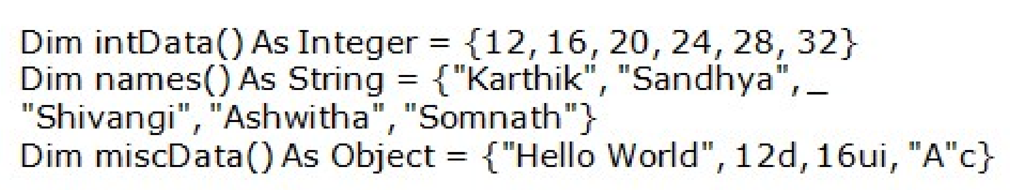
Creating Arrays in VB.Net

- To declare an array in VB.Net, you use the Dim statement.
- For example,



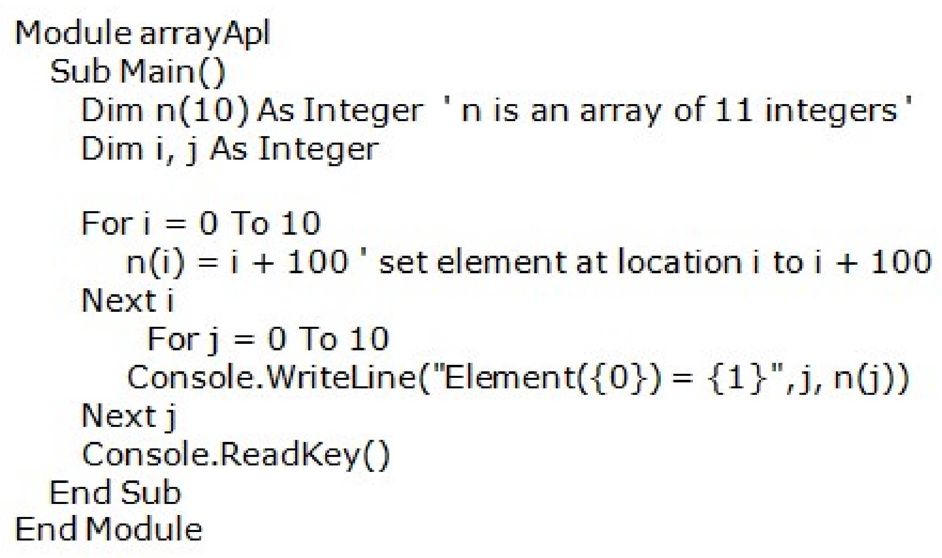
```
Dim intData(30)
Dim strData(20) As String
Dim twoDarray(10, 20) As Integer
Dim ranges(10, 100)
```

- You can also initialize the array elements while declaring the array. For example,



```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim names() As String = {"Karthik", "Sandhya", _
"Shivangi", "Ashwitha", "Somnath"}
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

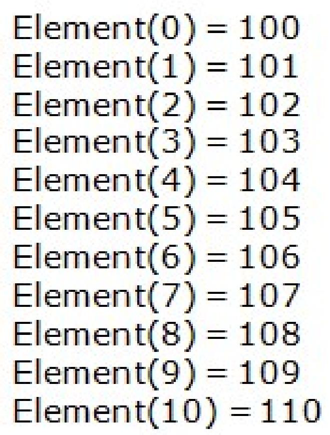
- The example shows how to create, access and operate the array.
- Dim is used to declare the array.
- The value may be declared at the access time or at the run time.
- The elements in an array can be stored and accessed by using the index of the array.
- The following program demonstrates this:



```
Module arrayApl
   Sub Main()
     Dim n(10) As Integer  ' n is an array of 11 integers'
     Dim i, j As Integer

     For i = 0 To 10
         n(i) = i + 100 ' set element at location i to i + 100
     Next i
           For j = 0 To 10
         Console.WriteLine("Element({0}) = {1}",j, n(j))
     Next j
     Console.ReadKey()
   End Sub
End Module
```

- ➢ This is the sample code for declaring and accessing the array.
- ➢ The console is to write the program in the output box.
- ➢ It is an program to print from 100 to 110 using for loop.

- When the above code is compiled and executed, it produces the following result:



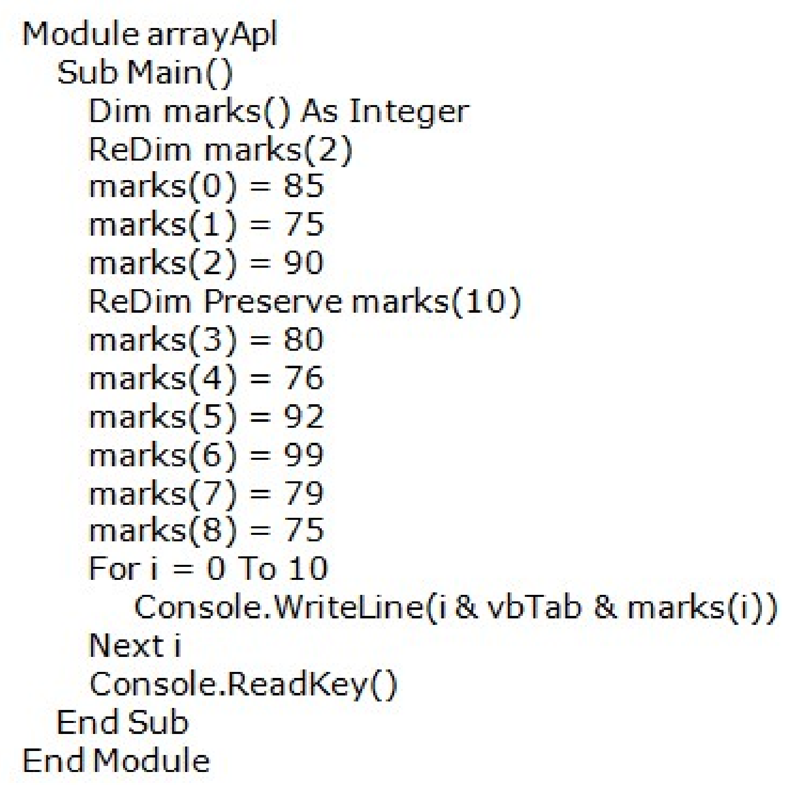
```
Element(0) = 100
Element(1) = 101
Element(2) = 102
Element(3) = 103
Element(4) = 104
Element(5) = 105
Element(6) = 106
Element(7) = 107
Element(8) = 108
Element(9) = 109
Element(10) = 110
```

Dynamic Arrays

- Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program.
- You can declare a dynamic array using the ReDim statement.
- Syntax for ReDim statement:
  - ➢ ReDim [Preserve] arrayname(subscripts)
- Where,
  - ➢ The Preserve keyword helps to preserve the data in an existing array, when you resize it.
  - ➢ array name is the name of the array to re-dimension.

- subscripts specifies the new dimension.

- Sample program:

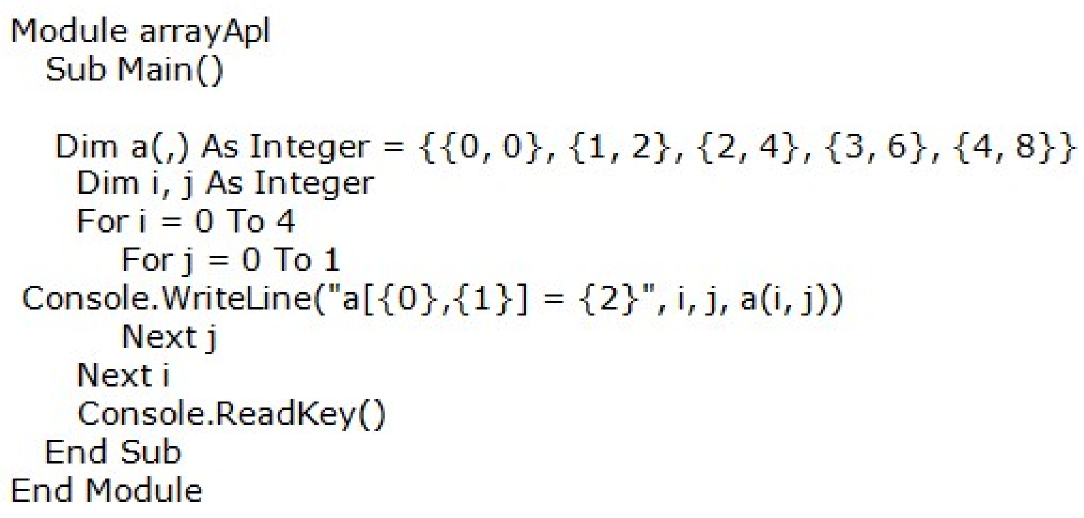


```
Module arrayApl
   Sub Main()
      Dim marks() As Integer
      ReDim marks(2)
      marks(0) = 85
      marks(1) = 75
      marks(2) = 90
      ReDim Preserve marks(10)
      marks(3) = 80
      marks(4) = 76
      marks(5) = 92
      marks(6) = 99
      marks(7) = 79
      marks(8) = 75
      For i = 0 To 10
          Console.WriteLine(i & vbTab & marks(i))
      Next i
      Console.ReadKey()
   End Sub
End Module
```

- In dynamic array the length should be declared dynamically.
- It is declared by using ReDim statement.
- No special reason for ReDim, It's a syntax used for dynamic array declaration.
- It produces the following result:

Multi-Dimensional Arrays

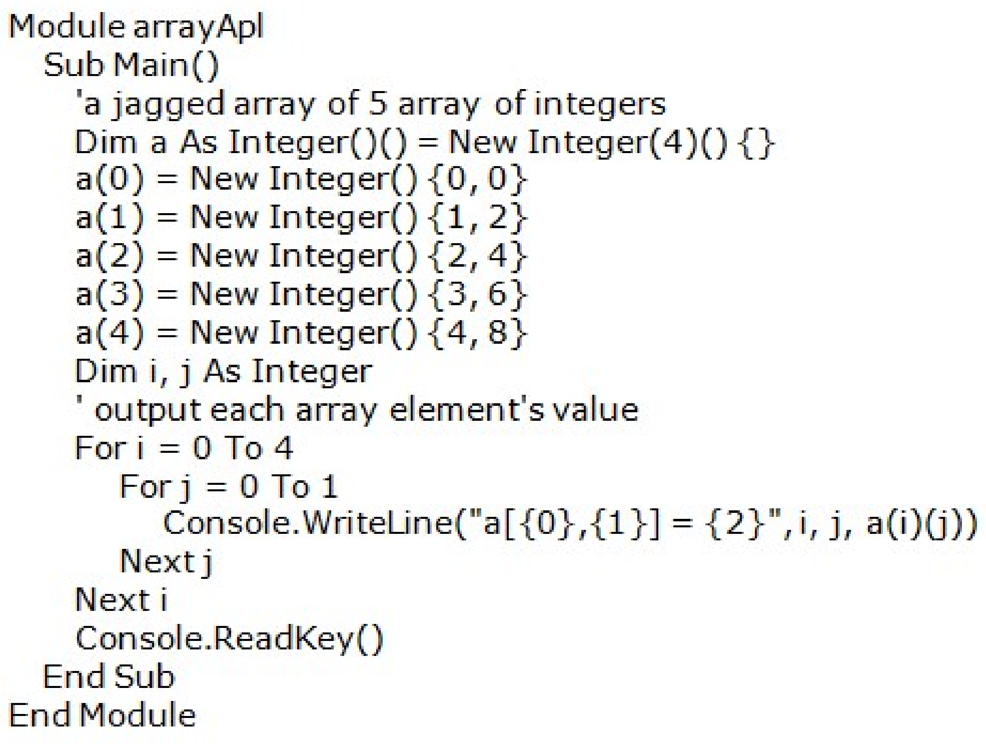
- VB.Net allows multidimensional arrays.
- Multidimensional arrays are also called rectangular arrays.
- You can declare a 2-dimensional array of strings as:
  - DimtwoDStringArray(10,20)AsString
- A 3-dimensional array of Integer variables:
  - DimthreeDIntArray(10,10,10)AsInteger
- The following program demonstrates creating and using a 2-dimensional array:



```
Module arrayApl
   Sub Main()

    Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
      Dim i, j As Integer
      For i = 0 To 4
          For j = 0 To 1
 Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
          Next j
      Next i
      Console.ReadKey()
   End Sub
End Module
```

Jagged Array

- Declaring an array using a defined array is called as Jagged array.
- A Jagged array is an array of arrays.
- The following code shows declaring a jagged array named scores of Integers.
  - Dim scores AsInteger()()=NewInteger(5)(){}
- The following example illustrates using a jagged array:



```
Module arrayApl
   Sub Main()
      'a jagged array of 5 array of integers
      Dim a As Integer()() = New Integer(4)() {}
      a(0) = New Integer() {0, 0}
      a(1) = New Integer() {1, 2}
      a(2) = New Integer() {2, 4}
      a(3) = New Integer() {3, 6}
      a(4) = New Integer() {4, 8}
      Dim i, j As Integer
      ' output each array element's value
      For i = 0 To 4
          For j = 0 To 1
              Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i)(j))
          Next j
      Next i
      Console.ReadKey()
   End Sub
End Module
```

The Array Class

- The Array class is the base class for all the arrays in VB.Net.
- It is defined in the System namespace.

  The Array class provides various properties and methods to work with arrays.

- ## Properties of the Array Class

- IsFixedSize
  - Gets a value indicating whether the Array has a fixed size.
- IsReadOnly
  - Gets a value indicating whether the Array is read-only.
- Length
  - Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
- LongLength
  - Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
- Rank
  - Gets the rank (number of dimensions) of the Array.

Methods of the Array Class

- Public Shared Sub Clear (array As Array, index As Integer, length As Integer)
  - Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
- Public Shared Sub Copy (sourceArray As Array, destinationArray As Array, length As Integer)
  - Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element.
  - The length is specified as a 32-bit integer.
- Public Sub CopyTo(array As Array, index As Integer)
  - Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index.
  - The index is specified as a 32-bit integer.
- Public Function GetLength(dimension As Integer) As Integer
  - Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
- Public Function GetLongLength(dimension As Integer) As Long
  - Gets a 64-bit integer that represents the number of elements in the specified dimension of

the Array.

- Public Function GetLowerBound(dimension As Integer) As Integer
  - Gets the lower bound of the specified dimension in the Array.
- Public Function GetTypeAs Type
  - Gets the Type of the current instance (Inherited from Object).
- Public Function GetUpperBound (dimension As Integer) As Integer
  - Gets the upper bound of the specified dimension in the Array.
- Public Function GetValue (index As Integer) As Object
  - Gets the value at the specified position in the one-dimensional Array.
  - The index is specified as a 32-bit integer.
- Public Shared Function IndexOf (array As Array, value As Object) As Integer
  - Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
- Public Shared Sub Reverse (array As Array)
  - Reverses the sequence of the elements in the entire one-dimensional Array.
- Public Sub SetValue (value As Object, index As Integer)
  - Sets a value to the element at the specified position in the one-dimensional Array.
  - The index is specified as a 32-bit integer.
- Public Shared Sub Sort (array As Array)
  - Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.
- Public Overridable Function ToString As String
  - Returns a string that represents the current object (Inherited from Object).

Example

- The following program demonstrates use of some of the methods of the Array class:

```
Module arrayApl
   Sub Main()
      Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
      Dim temp As Integer() = list
      Dim i As Integer
      Console.Write("Original Array: ")
      For Each i In list
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      ' reverse the array
      Array.Reverse(temp)
      Console.Write("Reversed Array: ")
      For Each i In temp
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      'sort the array
      Array.Sort(list)
      Console.Write("Sorted Array: ")
      For Each i In list
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      Console.ReadKey()
   End Sub
End Module
```

- This is the sample program using the array class methods.
- All the methods are not implemented, the most important methods are clear outer shortly.
- Result of the program:

```
Original Array: 34 72 13 44 25 30 10

Reversed Array: 10 30 25 44 13 72 34

Sorted Array: 10 13 25 30 34 44 72
```

Control Flow

- In a program, statements may be executed sequentially, selectively or iteratively.
- Every programming language provides constructs to support sequence, selection or iteration.
- So there are three types of programming constructs :
    - Sequential Constructs,
    - Selection Constructs,
    - Iterative Constructs.

Sequential Construct

- The sequential construct means the statements are being executed sequentially.
- This represents the default flow of statements.
- Program starts at a point and goes sequence means is called as sequential flow.
- It does not contain any jump statements.
- It is the default control in high level language.

Selection Construct

- The selection construct means the execution of statement(s) depending upon the condition-test.
- If a condition evaluates to true, a course-of-action (a set of statements) is followed otherwise another course-of-action is followed.
- This construct is also called decision construct as it helps in decision making.
- The conditions determines the controlling execution.
- For example jump A, If this statement executes the next step will movies to the statement A.

Iterative Constructs

- The iterative or repetitive constructs means repetition of a set-of-statements depending upon a condition-test.
- A set-of-statements are repeated again and again till the condition or Boolean Expression evaluates to true.
- The iteration constructs are also called as looping constructs.
- The looping statement determines the controlling execution.
- There are many iterative statements, for example for, While, incremental, decremented statements.

- The Boolean expression are used for evaluates to true.

Selection Constructs

- There are two types of selection construct in VB as like as in higher level languages.
- They are,
  - If statement.
  - Select Case statement.

If Statement

- The If Statement : If statement of VB comes in various forms & are given below:
  - If..Then Statement.
  - If..Then..Else Statement.
  - If..Then..ElseIf Statement.
  - Nested If.
- An If..Then statement tests a particular condition; if the condition evaluates to true, a course-of-action is followed otherwise it is ignored.
- If..Then..Else statement provides an alternate choice to the user i.e. if the condition is true then a set of statements are executed otherwise another set of statements are executed.
- If..Then..ElseIf statement is used to test a number of mutually exclusive cases and only executes one set of statements for the case that is true first.
- A nested If is an if that has another If in its if's body or in its else's body.
- The nested if can have one of the following three forms :

Select-Case Statement

- Select-Case is a multiple branching statement and is used to executed a set of statements depending upon the value of the expression.
- It is better to use Select-Case statement in comparison to If.
- Then..ElseIf Statement when the number of checks are more.
- The conditions determines the controlling execution.
- For example jump A, If this statement executes the next step will movies to the statement A.

Different forms of Select-Case

- Select Case : Simplest Form [Exact match]

```
Select Case Expression
Case Value
   'one or more visual basic statements
Case Value
   'one or more visual basic statements
Case Else :
 'one or more visual basic statements
End Select
```

EXAMPLE

```
Select Case byMonth
   Case 1,3,5,7,8,10,12
         number_of_days=31
   Case 2
         number_of_days=28
   Case 4,6,9,11
         number_of_days=30
End Select
```

- Select Case : Second Form [Relational Test]

```
Select Case Expression
Case is relation :
   'one or more visual basic statements
Case is relation :
   'one or more visual basic statements
Case Else :
 'one or more visual basic statements
End Select
```

EXAMPLE

```
Select Case marks
  Case Is < 50
     Result = "Fail"
  Case Is < 60
     Result = "Grade B"
  Case Is < 75
     Result = "Grade A"
  Case Else
     Result = "Grade A+"
End Select
```

- Select Case : Third Format [Range Check]

```
 Select Case Expression
 Case exp1 To exp2:
    'one or more visual basic statements
 Case exp1 To exp2:
    'one or more visual basic statements
 [Case Else:
    'one or more visual basic statements
End Select
```

EXAMPLE

```
   Select Case Age
     Case 2 to 4 : Print "PreNursery"
     Case 4 to 6 : Print "Kindergarden"
     Case 6 to 10 : Print "Primary"
     Case Else : Print "Others"
 End Select
```

Iterative Constructs (Looping Structures)

- Loop : A loop is said to be the set of instructions which are repeated again and again in a program.
- Types of Loops in VB :

  - Sentinel-controlled Loop Structures : repeat statements until a special value called sentinel value (or the terminating value) is reached.
  - Counter-controlled Loop Structures : repeat the set of statements until the value specified by the counter variable is reached.
- VB offers broadly following three types of looping structures :
  - For..Next
  - Do Loop
    - Do While..Loop,
    - Do..Loop While,
    - Do Until..Loop,
    - Do..Loop Until.
  - While..Wend

For..Next Statement

- Syntax :

```
For < counter Variable >= < start_val > To < end_val >
Step < increment/Decrement Value >
` One or more VB Statements
Next < counter Variable >
```

- This type of statement is used when the user knows in advance how many times the loop is going to be executed.

Do While..Loop

- Syntax :

```
Do While < condition or boolean expression >
` One or more VB Statements
Loop
```

- Do While loop is an entry controlled loop in which the condition is placed at the entry point.
- This statement executes the statements specified in the body of the loop till the condition evaluates to true.
- The loop may not be executed at all the if the condition is initially false.

Do Loop While

Syntax :

-

```
Do
One or more VB Statements
Loop While < condition or Boolean Expression >
```

- Do Loop While is an exit controlled loop as the condition is placed at exit point.
- The body of the loop is going to be executed at least once whether the condition evaluates to true or false.
- Loop is executed as long as the result of the condition remains true.

Do Until loop

- Syntax :

```
Do Until < condition or boolean expression >
` One or more VB Statements
Loop
```

- Do Until loop is an entry controlled loop in which the condition is placed at the entry point.
- This statement executes the statements specified in the body of the loop till the condition evaluates to false.
- The loop may not be executed at all the if the condition is initially true.

Do Loop Until

- Syntax :

```
Do
One or more VB Statements
Loop Until < condition or Boolean Expression >
```

- Do Loop Until is an exit controlled loop as the condition is placed at exit point.
- The body of the loop is going to be executed at least once whether the condition evaluates to true or false.
- Loop is executed as long as the result of the condition remains false.

While..Wend loop

- Syntax :

```
While < Condition >
one or more vb statements
Wend
```

- While..Wend loop is functionally equivalent to the Do While..Loop. It executes a set of VB statements till the condition evaluates to true.

Collections

- Collection classes are specialized classes for data storage and retrieval.
- These classes provide support for stacks, queues, lists, and hash tables.
- Most collection classes implement the same interfaces.
- Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index, etc.
- These classes create collections of objects of the Object class, which is the base class for all data types in VB.Net.

Various Collection Classes and Their Usage

- ArrayList.
- Hashtable.
- SortedList.
- Stack.
- Queue.
- BitArray.

ArrayList

- It represents ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array.
- However, unlike array, you can add and remove items from a list at a specified position using an index and the array resizes itself automatically.
- It also allows dynamic memory allocation, add, search and sort items in the list.

Hashtable

- It uses a key to access the elements in the collection.
- A hash table is used when you need to access elements by using key, and you can identify a useful key value.
- Each item in the hash table has a key/value pair.
- The key is used to access the items in the collection.

SortedList

It uses a key as well as an index to access the items in a list.

- - A sorted list is a combination of an array and a hash table.
- It contains a list of items that can be accessed using a key or an index.
- If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable.
- The collection of items is always sorted by the key value.

Stack

- It represents a last-in, first out collection of object.
- It is used when you need a last-in, first-out access of items.
- When you add an item in the list, it is called pushing the item, and when you remove it, it is calledpopping the item.

Queue

- It represents a first-in, first out collection of object.
- It is used when you need a first-in, first-out access of items.
- When you add an item in the list, it is called enqueue, and when you remove an item, it is calleddeque

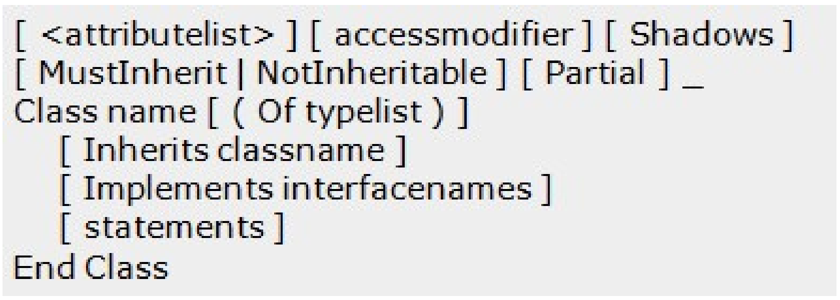
BitArray

- It represents an array of the binary representation using the values 1 and 0.
- It is used when you need to store the bits but do not know the number of bits in advance.
- You can access items from the BitArray collection by using an integer index, which starts from zero.

UNIT - 9
.NET Architecture and Advanced Tools

Object-oriented Programming - Classes & Objects

- When you define a class, you define a blueprint for a data type.
- This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.
- Objects are instances of a class.
- The methods and variables that constitute a class are called members of the class.
- A class definition starts with the keyword Class followed by the class name; and the class body, ended by the End Class statement.
- Following is the general form of a class definition:



```
[ <attributelist> ] [ accessmodifier ] [ Shadows ]
[ MustInherit | NotInheritable ] [ Partial ] _
Class name [ ( Of typelist ) ]
    [ Inherits classname ]
    [ Implements interfacenames ]
    [ statements ]
End Class
```

- Attributelist is a list of attributes that apply to the class. Optional.
- Accessmodifier defines the access levels of the class, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- Shadows indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
- MustInherit specifies that the class can be used only as a base class and that you cannot create an object directly from it, i.e., an abstract class. Optional.
- NotInheritable specifies that the class cannot be used as a base class.
- Partial indicates a partial definition of the class.
- Inherits specifies the base class it is inheriting from.
- Implements specifies the interfaces the class is inheriting from.

Member Functions and Variables

- A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
- It operates on any object of the class of which it is a member and has access to all the members of a class for that object.

- Member variables are attributes of an object (from design perspective) and they are kept private to implement encapsulation.
- These variables can only be accessed using the public member functions.

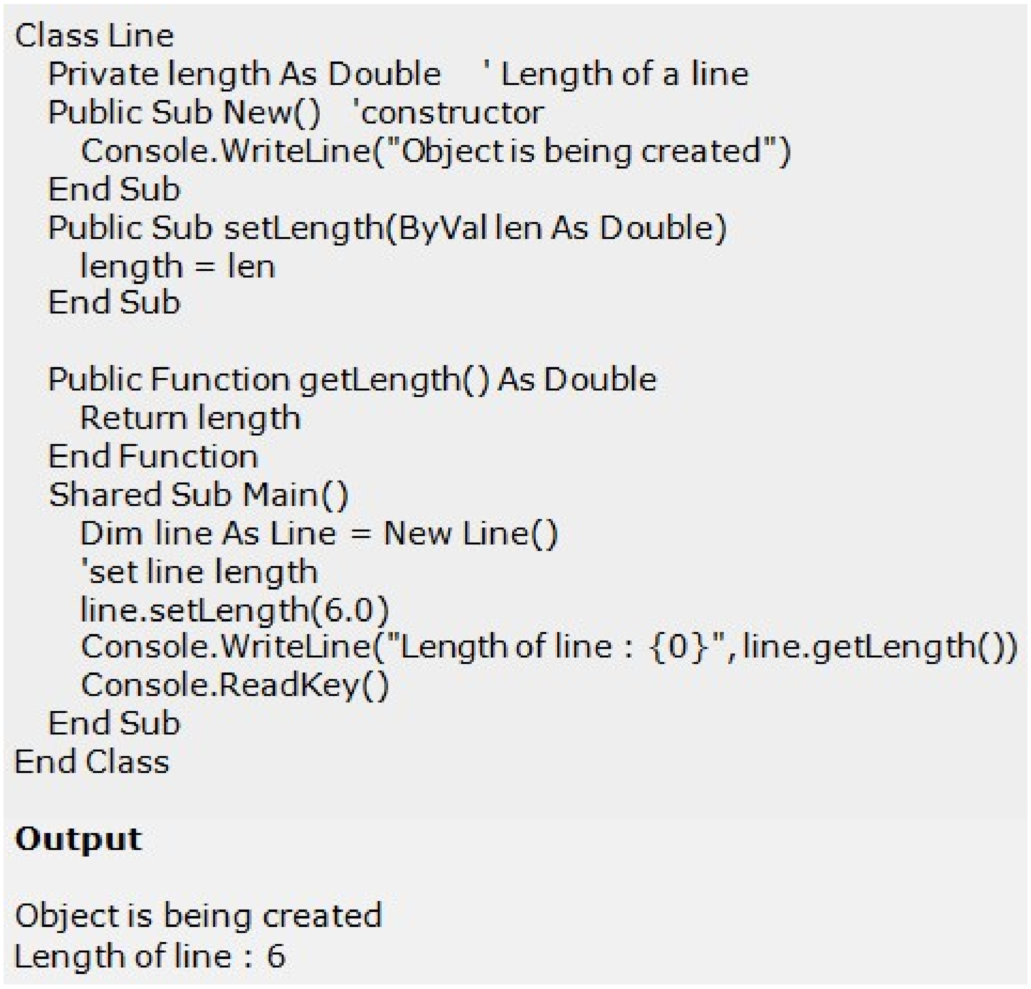
Constructors

- A class constructor is a special member Sub of a class that is executed whenever we create new objects of that class.
- A constructor has the name New and it does not have any return type.

Types of constructor

- Default Constructor
- Parameterized constructors
- A default constructor does not have any parameter, but if you need, a constructor can have parameters.
- Such constructors are called parameterized constructors.
- This technique helps you to assign initial value to an object at the time of its creation.

Example For Constructor



```
Class Line
   Private length As Double    ' Length of a line
   Public Sub New()   'constructor
      Console.WriteLine("Object is being created")
   End Sub
   Public Sub setLength(ByVal len As Double)
      length = len
   End Sub

   Public Function getLength() As Double
      Return length
   End Function
   Shared Sub Main()
      Dim line As Line = New Line()
      'set line length
      line.setLength(6.0)
      Console.WriteLine("Length of line : {0}", line.getLength())
      Console.ReadKey()
   End Sub
End Class
```

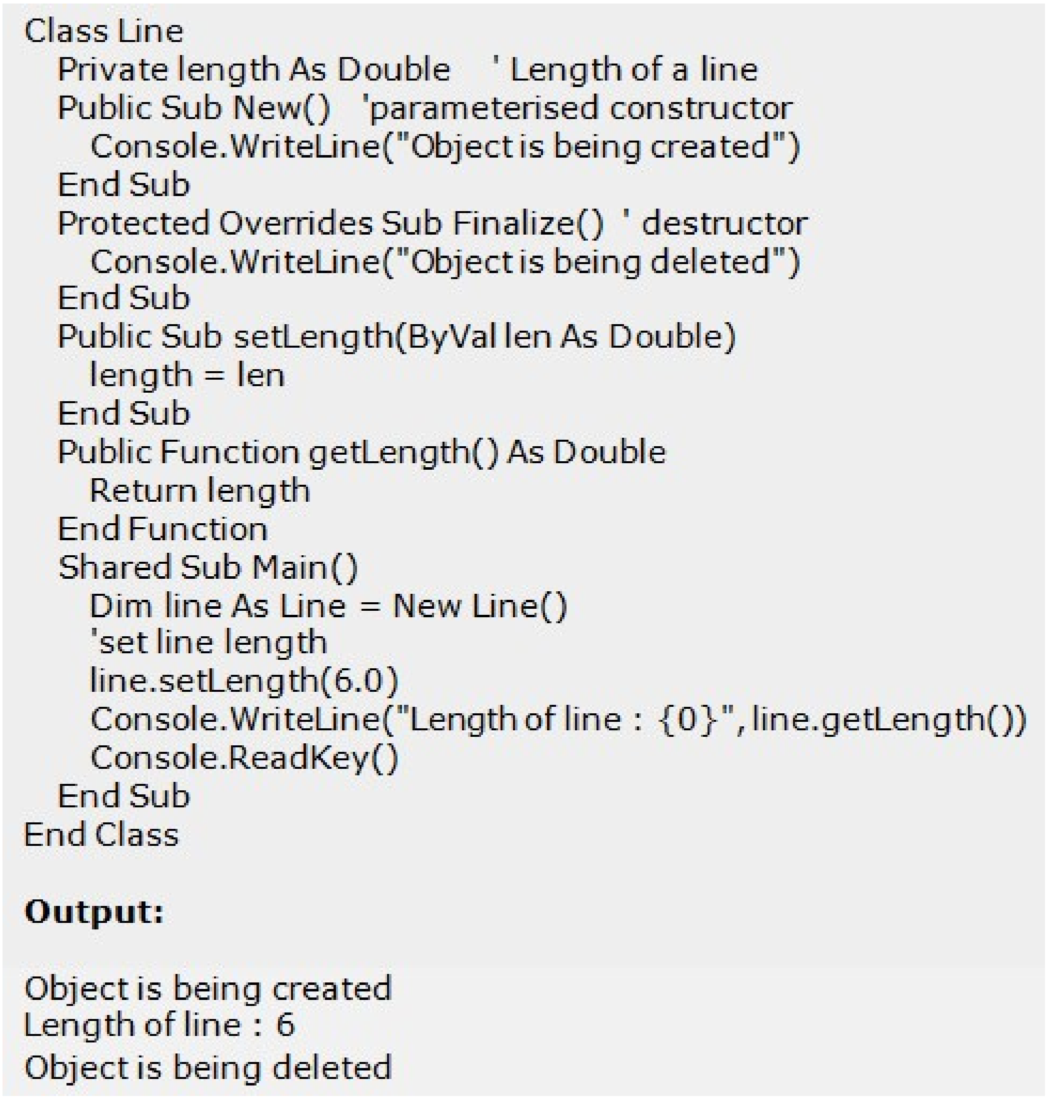
Output

```
Object is being created
Length of line : 6
```

Destructor

- A destructor is a special member Sub of a class that is executed whenever an object of its class goes out of scope.
    - Protected Overrides Sub Finalize() ' destructor
      Console.WriteLine("Object is being deleted")
      End Sub
- A destructor has the name Finalize and it can neither return a value nor can it take any parameters.
- Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories, etc.
- Destructors cannot be inherited or overloaded.

Example For Destructor



```
Class Line
   Private length As Double    ' Length of a line
   Public Sub New()   'parameterised constructor
      Console.WriteLine("Object is being created")
   End Sub
   Protected Overrides Sub Finalize()  ' destructor
      Console.WriteLine("Object is being deleted")
   End Sub
   Public Sub setLength(ByVal len As Double)
      length = len
   End Sub
   Public Function getLength() As Double
      Return length
   End Function
   Shared Sub Main()
      Dim line As Line = New Line()
      'set line length
      line.setLength(6.0)
      Console.WriteLine("Length of line : {0}", line.getLength())
      Console.ReadKey()
   End Sub
End Class
```

Output:

```
Object is being created
Length of line : 6
Object is being deleted
```

Shared Members of a VB .Net Class

- The keyword Shared implies that only one instance of the member exists for a class.

  Shared variables are used for defining constants because their values can be retrieved by

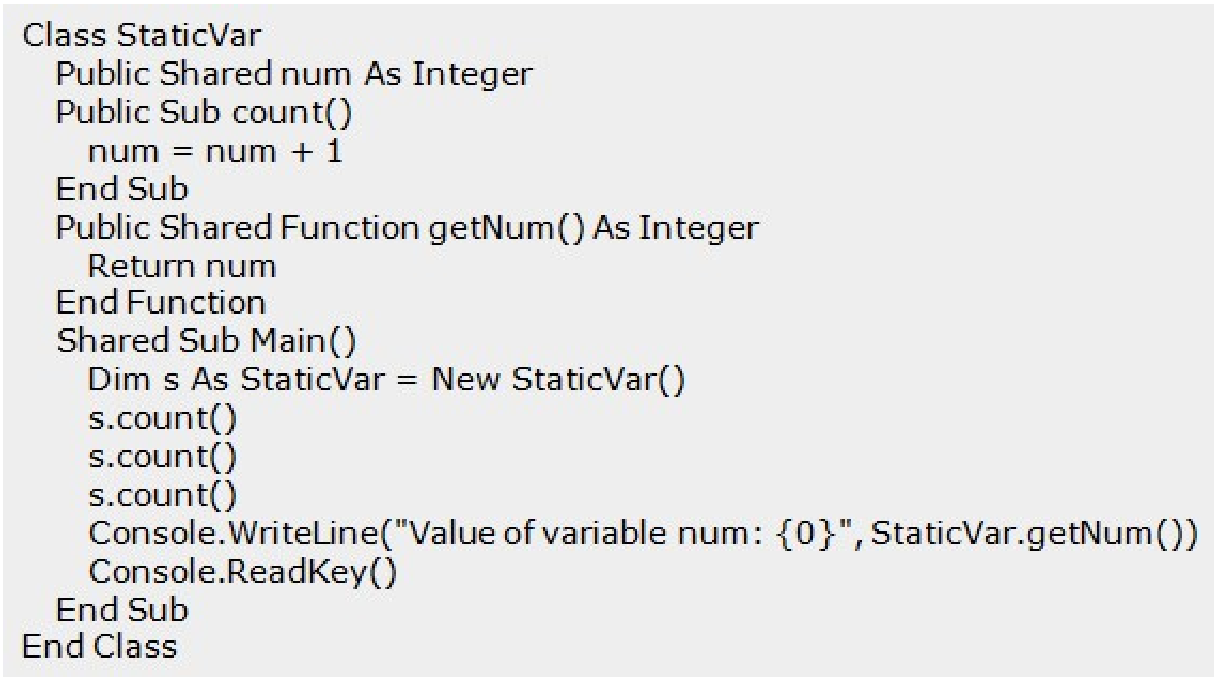
- invoking the class without creating an instance of it.

  ➢ Public Shared Function getNum() As Integer
  Return num
  End Function

- We can define class members as static using the Shared keyword.
- When we declare a member of a class as Shared, it means no matter how many objects of the class are created, there is only one copy of the member.

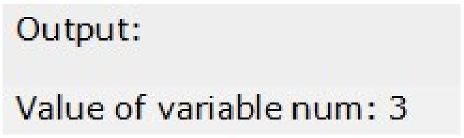
  ➢ Public Shared num As Integer
  Public Sub count()
  num = num + 1
  End Sub

- Shared variables can be initialized outside the member function or class definition.
- You can also initialize Shared variables inside the class definition.
- Example For Shared Members of a VB .Net Class.



```
Class StaticVar
   Public Shared num As Integer
   Public Sub count()
      num = num + 1
   End Sub
   Public Shared Function getNum() As Integer
      Return num
   End Function
   Shared Sub Main()
      Dim s As StaticVar = New StaticVar()
      s.count()
      s.count()
      s.count()
      Console.WriteLine("Value of variable num: {0}", StaticVar.getNum())
      Console.ReadKey()
   End Sub
End Class
```

- When the code is compiled and executed, it produces the following result:



```
Output:

Value of variable num: 3
```

- You can also declare a member function as Shared.
- Such functions can access only Shared variables.
- The Shared functions exist even before the object is created.

Inheritance

- One of the most important concepts in object-oriented programming is that of inheritance.
- Inheritance allows us to define a class in terms of another class which makes it easier to create and maintain an application.
- This also provides an opportunity to reuse the code functionality and fast implementation time.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.
- This existing class is called the base class, and the new class is referred to as the derived class.

Base & Derived Classes

- A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.
- The derived class inherits the base class member variables and member methods.
- Therefore, the super class object should be created before the subclass is created.
- The super class or the base class is implicitly known as MyBase in VB.Net.
- The syntax used in VB.Net for creating derived classes is as follows:

```
<access-specifier> Class <base_class>
...
End Class
Class <derived_class>: Inherits <base_class>
...
End Class
```

Exception Handling

- An exception is a problem that arises during the execution of a program.
- An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- In General it handles the run time error.
- Exceptions provide a way to transfer control from one part of a program to another.
- VB.Net exception handling is built upon four keywords: Try, Catch, Finally and Throw.
- Syntax

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

- Try:
  - A Try block identifies a block of code for which particular exceptions will be activated.
  - It's followed by one or more Catch blocks.
- Catch:
  - A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
  - The Catch keyword indicates the catching of an exception.
- Finally:
  - The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
  - For example, if you open a file, it must be closed whether an exception is raised or not.
- Throw:
  - A program throws an exception when a problem shows up.
  - This is done using a Throw keyword.

Exception Classes in .Net Framework

- In the .Net Framework, exceptions are represented by classes.
- The exception classes in .Net Framework are mainly directly or indirectly derived from the System.Exception class.
- Some of the exception classes derived from the System.Exception class are the System.ApplicationException andSystem.SystemException classes.
- The System.ApplicationException class supports exceptions generated by application programs.
- So the exceptions defined by the programmers should derive from this class.
- The System.SystemException class is the base class for all predefined system exception.

Some of the Predefined Exception Classes

- System.IO.IOException
  - Handles I/O errors.
- System.IndexOutOfRangeException
  - Handles errors generated when a method refers to an array index out of range.
- System.ArrayTypeMismatchException
  - Handles errors generated when type is mismatched with the array type.
- System.NullReferenceException
  - Handles errors generated from dereferencing a null object.
- System.DivideByZeroException
  - Handles errors generated from dividing a dividend with zero.
- System.InvalidCastException
  - Handles errors generated during typecasting.
- System.OutOfMemoryException
  - Handles errors generated from insufficient free memory.
- System.StackOverflowException
  - Handles errors generated from stack overflow.

Handling Exceptions

- VB.Net provides a structured solution to the exception handling problems in the form of try and catch blocks.
- Using these blocks the core program statements are separated from the error-handling statements.
- These error handling blocks are implemented using the Try, Catch and Finally keywords.
- Following is an example of throwing an exception when dividing by zero condition occurs:

```
Module exceptionProg
   Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
      Dim result As Integer
      Try
         result = num1 \ num2
      Catch e As DivideByZeroException
         Console.WriteLine("Exception caught: {0}", e)
      Finally
         Console.WriteLine("Result: {0}", result)
      End Try
   End Sub
   Sub Main()
      division(25, 0)
      Console.ReadKey()
  End Sub
End Module
```

- You can use a throw statement in the catch block to throw the present object as:
  - Throw [expression]
- Example :

```
Module exceptionProg
   Sub Main()
      Try
         Throw New ApplicationException("A custom exception _
                  is being thrown here...")
      Catch e As Exception
         Console.WriteLine(e.Message)
      Finally
         Console.WriteLine("Now inside the Finally Block")
      End Try
      Console.ReadKey()
   End Sub
End Module
```

- You can throw an object if it is either directly or indirectly derived from the System.Exception class.
- Output:

```
A custom exception is being thrown here...
Now inside the Finally Block
```

Creating Distributed Web Applications

- Visual Studio .NET provides the tools you need to design, develop, debug, and deploy Web applications, XML Web services, and traditional client applications.
- Information on application design decisions, such as system architecture, database design, and international considerations, as well as Enterprise Templates.
- Application architects can use the .NET Platform to develop, deploy, and support distributed applications.
- Highly integrated but flexible, this platform enables developers to build end-to-end business solutions that can leverage existing architectures and applications.
- Windows DNA was an architecture for building tightly-coupled, distributed Web applications.
- As distributed applications began to require more loosely-coupled principles, the Microsoft architecture moved to the .NET Platform.
- Developers can build highly scalable and flexible applications by partitioning applications along these lines, by using component-based programming techniques, and by fully using the features of the .NET Platform and the Microsoft Windows operating system.
- A simple distributed application model consists of a client that communicates with the middle layer, which itself consists of the application server and an application containing the business logic.
- The application, in turn, communicates with a database that supplies and stores data.
- The key tenet of distributed applications is the logical partitioning of an application into three fundamental layers:
  - Presentation.
  - Business Logic.
  - Data Access and Storage.

Presentation services

- The presentation layer includes either a rich- or thin-client interface to an application.
- The rich client, either directly by using the Microsoft Win32 API or indirectly through Windows Forms, provides a full programming interface to the operating system's capabilities or uses components extensively.
- The thin client (Web browser) is rapidly becoming the interface of choice for many developers.
- A developer is able to build business logic that can be executed on any of the three application tiers.

With ASP.NET Web applications and XML Web services, the thin client is able to provide a

- visually rich, flexible, and interactive user interface to applications.
- Thin clients also have the advantage of providing a greater degree of portability across platforms.

Business Logic - Application Services

- This layer is divided into application servers and services, which are available to support clients.
- Web applications can be written to take advantage of COM+ services, Message Queuing (MSMQ), directory services, and security services using the .NET Framework.
- Application services, in turn, can interact with several data services on the data access layer.

Data Access and Storage Services

- The data services that support data access and storage consist of:
  - ADO.NET, which provides simplified programmatic access to data by using either scripting or programming languages.
  - OLE DB, which is an established universal data provider developed by Microsoft.
  - XML, which is a mark-up standard for specifying data structures.

Availability

- All applications are available at least some of the time, but Web-based applications and mission-critical enterprise applications must typically provide round-the-clock services.
- If your enterprise application needs to work 24 hours a day, 7 days a week, you probably need to design for high availability.
- Advances in hardware and software have dramatically increased the quality of high-availability applications.
- However, availability is not easy to implement and requires a considerably more complex architectural infrastructure than the previous generation of client/server applications.
- If your application requires high availability, you will want to understand how design choices help maximize application availability and how testing can validate planned service levels.
- Reduce unplanned downtime:
  - Cluster service with a shared disk avoids most downtime and provides automatic recovery from hardware or software failures.
- Reduce planned downtime:
  - Planned downtime is reduced because you can deploy operating system and application upgrades without interrupting normal service by using a rolling upgrade.

Continuous health monitoring:

- ➢ The operational status of your application and server is automatically checked.
    - ➢ When a problem is identified the failed service is transferred to another server.

Manageability

- Managing a modern .NET application requires an efficient way to handle typical local and remote application support processes, including:
    - ➢ Initial deployment.
    - ➢ Configuration tuning.
    - ➢ Scheduled and unscheduled maintenance.
    - ➢ Occasional troubleshooting.
- Management agents:
    - ➢ Each hardware device, operating system service, and application service requires a management agent.
    - ➢ Management agents monitor the local resource and publish data about the resource's current state and performance.
- Collection process:
    - ➢ The information collection process collects, filters, correlates, and stores information from all of the management agents.
- Management console:
    - ➢ The management console workstation aggregates and reports on application management information.
    - ➢ From this central console an administrator can monitor all devices, analyze operational profiles, automate certain recurring activities, receive notifications from managed elements, and initiate remote configuration changes.

Performance

- Identifying Constraints.
- Determining Features.
- Specifying the Load.
- Key application metrics, such as transaction throughput and resource utilization, define application performance.
- Metrics related to hardware, such as network throughput and disk access, are common application performance bottlenecks.

- From a users perspective, application response time defines performance.
- Of course, performance does not come without a price.
- While it is possible to build a high performance application for any given problem space, a key price point is the cost per transaction.
- It is sometimes necessary to sacrifice performance to control cost.

Reliability

- As distributed applications grow both in size and complexity, there is an increasing need to improve the reliability and operating quality of software.
- First, the cost of application failure is often too high.
- Users bypass unreliable Web sites, resulting in lost revenue and reduced future sales, and the expense of repairing corrupted data can further increase the cost of application failure.
- Second, unreliable systems are difficult to maintain or improve because the failure points are typically hidden throughout the system.
- Finally, modern software technology makes it easy to create reliable applications.
- At its best, good reliability design engineering would:
    - Follow the Windows 2000 application design guidelines.
    - Put reliability requirements in the specification.
    - Use good architectural infrastructure.
    - Build management information into the application.
    - Use redundancy for reliability.
    - Use quality development tools.
    - Use built-in application health checks.
    - Use consistent error handling.

Scalability

- Scalability is the capability to increase resources to yield a linear (ideally) increase in service capacity.
- The key characteristic of a scalable application is that additional load only requires additional resources rather than extensive modification of the application itself.
- Although raw performance makes a difference in determining the number of users that an application can support, scalability and performance are two separate entities.

- In fact, performance efforts can sometimes be opposed to scalability efforts.
- The Five Commandments of Designing for Scalability
  - Do Not Wait.
  - Do Not Fight for Resources.
  - Design for Commutability.
  - Design for Interchangeability.
  - Partition Resources and Activities.

Securability

- The ability to provide security to an application and its data is referred to here as securability.
- The securability of an application is impacted by numerous design choices, such as the selection of communication protocols and the method of user authentication.
- Most security vulnerabilities are not in security-related code.
- Instead, they are found in code that is unrelated to security and that was written without attention to security.
- This is why developers must be keenly aware of application securability.
- At Microsoft, we use the acronym STRIDE, which describes the following taxonomy of security threats:
  - Spoofing Identity.
  - Tampering with Data.
  - Repudiability.
  - Information Disclosure.
  - Denial of Service.
  - Elevation of Privilege.

XML

- Extensible Markup Language (XML) is the universal format for data on the Web.
- XML allows developers to easily describe and deliver rich, structured data from any application in a standard, consistent way.
- XML does not replace HTML; rather, it is a complementary format.
- The Extensible Markup Language (XML) is a markup language much like HTML or SGML.
- This is recommended by the World Wide Web Consortium and available as an open standard.
- The System.Xml namespace in the .Net Framework contains classes for processing XML documents.

XML Class

- XmlAttribute
  - Represents an attribute.
  - Valid and default values for the attribute are defined in a document type definition (DTD) or schema.
- XmlCDataSection
  - Represents a CDATA section.
- XmlCharacterData
  - Provides text manipulation methods that are used by several classes.
- XmlComment
  - Represents the content of an XML comment.
- XmlConvert
  - Encodes and decodes XML names and provides methods for converting between common language runtime types and XML Schema definition language (XSD) types.
  - When converting data types, the values returned are locale independent.
- XmlDeclaration
  - Represents the XML declaration node < ?xml version=‘ 1.0 ’...? >.
- XmlDictionary
  - Implements a dictionary used to optimize Windows Communication Foundation (WCF)'s XML reader/writer implementations.

- XmlDictionaryReader
  - An abstract class that the Windows Communication Foundation (WCF) derives from XmlReader to do serialization and deserialization.
- XmlDictionaryWriter
  - Represents an abstract class that Windows Communication Foundation (WCF) derives from XmlWriter to do serialization and deserialization.
- XmlDocument
  - Represents an XML document.
- XmlDocumentFragment
  - Represents a lightweight object that is useful for tree insert operations.
- XmlDocumentType
  - Represents the document type declaration.
- XmlElement
  - Represents an element.
- XmlEntity
  - Represents an entity declaration, such as < !ENTITY... >.
- XmlEntityReference
  - Represents an entity reference node.
- XmlException
  - Returns detailed information about the last exception.
- XmlImplementation
  - Defines the context for a set of XmlDocument objects.
- XmlLinkedNode
  - Gets the node immediately preceding or following this node.
- XmlNode
  - Represents a single node in the XML document.
- XmlNodeList
  - Represents an ordered collection of nodes.
- XmlNodeReader

  - Represents a reader that provides fast, non-cached forward only access to XML data in an XmlNode.

- XmlNotation
  - Represents a notation declaration, such as < !NOTATION... >.
- XmlParserContext
  - Provides all the context information required by the XmlReader to parse an XML fragment.
- XmlProcessingInstruction
  - Represents a processing instruction, which XML defines to keep processor-specific information in the text of the document.
- XmlQualifiedName
  - Represents an XML qualified name.
- XmlReader
  - Represents a reader that provides fast, noncached, forward-only access to XML data.
- XmlReaderSettings
  - Specifies a set of features to support on the XmlReader object created by the Create method.
- XmlResolver
  - Resolves external XML resources named by a Uniform Resource Identifier (URI).
- XmlSecureResolver
  - Helps to secure another implementation of XmlResolver by wrapping the XmlResolver object and restricting the resources that the underlying XmlResolver has access to.
- XmlSignificantWhitespace
  - Represents white space between markup in a mixed content node or white space within an xml:space= ‘preserve’ scope.
  - This is also referred to as significant white space.
- XmlText
  - Represents the text content of an element or attribute.
- XmlTextReader
  - Represents a reader that provides fast, non-cached, forward-only access to XML data.
- XmlTextWriter

  Represents a writer that provides a fast, non-cached, forward-only way of generating streams

  - or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

- XmlWhitespace
  - Represents white space in element content.
- XmlUrlResolver
  - Resolves external XML resources named by a Uniform Resource Identifier (URI).
- XmlWriter
  - Represents a writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data.
- XmlWriterSettings
  - Specifies a set of features to support on the XmlWriter object created by the XmlWriter.Create method.

ADO .NET

- ADO.Net provides a bridge between the front end controls and the back end database.
- The ADO.Net objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.
- ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC.
- Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.
- The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll.
- For sample code that connects to a database, retrieves data from it, and then displays that data in a console window.

The DataSet Class

- The data set represents a subset of the database.
- It does not have a continuous connection to the database.
- To update the database a reconnection is required.
- The DataSet contains DataTable objects and DataRelation objects.

Properties of the DataSet class

- CaseSensitive
  - Indicates whether string comparisons within the data tables are case-sensitive.
- Container
  - Gets the container for the component.
- DataSetName
  - Gets or sets the name of the current data set.
- DefaultViewManager
  - Returns a view of data in the data set.
- DesignMode
  - Indicates whether the component is currently in design mode.
- EnforceConstraints

- Indicates whether constraint rules are followed when attempting any update operation.

Methods of the DataSet class

- Events
    - Gets the list of event handlers that are attached to this component.
- ExtendedProperties
    - Gets the collection of customized user information associated with the DataSet.
- HasErrors
    - Indicates if there are any errors.
- IsInitialized
    - Indicates whether the DataSet is initialized.
- Locale
    - Gets or sets the locale information used to compare strings within the table.
- Namespace
    - Gets or sets the namespace of the DataSet.

The DataTable Class

- The DataTable class represents the tables in the database. It has the following important properties;
- ChildRelations
    - Returns the collection of child relationship.
- Columns
    - Returns the Columns collection.
- Constraints
    - Returns the Constraints collection.
- DataSet
    - Returns the parent DataSet.
- DefaultView
    - Returns a view of the table.
- ParentRelations

   - Returns the ParentRelations collection.
- PrimaryKey
   - Gets or sets an array of columns as the primary key for the table.
- Rows
   - Returns the Rows collection.

The DataRow Class

- The DataRow object represents a row in a table. It has the following important properties:
- HasErrors
   - Indicates if there are any errors.
- Items
   - Gets or sets the data stored in a specific column .
- ItemArrays
   - Gets or sets all the values for the row.
- Table
   - Returns the parent table.
- The DataAdapter Object
   - The DataAdapter object acts as a mediator between the DataSet object and the database.
   - This helps the data set to contain data from more than one database or other data source.
- The DataReader Object
   - The DataReader object is an alternative to the DataSet and DataAdapter combination.
   - This object provides a connection oriented access to the data records in the database.
- DbConnection Objects
   - The DbConnection object represents a connection to the data source.
   - The connection could be shared among different command objects.
- DbCommand Objects
   - The DbCommand object represents the command or a stored procedure sent to the database from retrieving or manipulating data.

Graphics Control

- Graphics
  - ➤ It is the top level abstract class for all graphics contexts.
- Graphics2D
  - ➤ It is a subclass of Graphics class and provides more sophisticated control over geometry, coordinate transformations, color management, and text layout.
- Arc2D
  - ➤ Arc2D is the abstract superclass for all objects that store a 2D arc defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE).
- CubicCurve2D
  - ➤ The CubicCurve2D class is the abstract superclass fpr all objects which store a 2D cubic curve segment and it defines a cubic parametric curve segment in (x, y) coordinate space.
- Ellipse2D
  - ➤ The Ellipse2D is the abstract superclass for all objects which store a 2D ellipse and it describes an ellipse that is defined by a framing rectangle.
- Rectangle2D
  - ➤ The Rectangle2D class is an abstract superclass for all objects that store a 2D rectangle and it describes a rectangle defined by a location (x, y) and dimension (w x h).
- QuadCurve2D
  - ➤ The QuadCurve2D class is an abstract superclass for all objects that store a 2D quadratic curve segment and it describes a quadratic parametric curve segment in (x, y) coordinate space.
- Line2D
  - ➤ This Line2D represents a line segment in (x,y) coordinate space.
- Font
  - ➤ The Font class represents fonts, which are used to render text in a visible way.
- Color
  - ➤ The Color class is used encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.
- Basicstroke

- The BasicStroke class defines a basic set of rendering attributes for the outlines of graphics primitives, which are rendered with a Graphics2D object that has its Stroke attribute set to this BasicStroke.

Printing

- The PrintDialog control lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.
- There are various other controls related to printing of documents.
- The PrintDocument control
  - It provides support for actual events and operations of printing in Visual Basic and sets the properties for printing.
- The PrinterSettings control
  - It is used to configure how a document is printed by specifying the printer.
- The PageSetUpDialog control
  - It allows the user to specify page-related print settings including page orientation, paper size and margin size.
- The PrintPreviewControl control
  - It represents the raw preview part of print previewing from a Windows Forms application, without any dialog boxes or buttons.
- The PrintPreviewDialog control
  - It represents a dialog box form that contains a PrintPreviewControl for printing from a Windows Forms application.
- The following are some of the commonly used properties of the PrintDialog control:
  - AllowCurrentPage
    - Gets or sets a value indicating whether the Current Page option button is displayed.
  - AllowPrintToFile
    - Gets or sets a value indicating whether the Print to file check box is enabled.
  - AllowSelection
    - Gets or sets a value indicating whether the Selection option button is enabled.
  - AllowSomePages
    - Gets or sets a value indicating whether the Pages option button is enabled.
  - Document
    - Gets or sets a value indicating the PrintDocument used to obtain PrinterSettings.

- PrinterSettings
  - Gets or sets the printer settings the dialog box modifies.
- PrintToFile
  - Gets or sets a value indicating whether the Print to file check box is selected.
- ShowHelp
  - Gets or sets a value indicating whether the Help button is displayed.
- ShowNetwork
  - Gets or sets a value indicating whether the Network button is displayed.

Printing Methods

- Reset
  - Resets all options to their default values.
- RunDialog
  - When overridden in a derived class, specifies a common dialog box.
- ShowDialog
  - Runs a common dialog box with a default owner.

Reporting

Designing the report

- In this step we create the JRXML file, which is an XML document that contains the definition of the report layout.
- We can use any text editor or iReportDesigner to manually create it.
- If iReportDesigner is used the layout is designed in a visual way, hence real structure of the JRXML can be ignored.

Compiling the report

- In this step JRXML is compiled in a binary object called a Jasper file(*.jasper).
- This compilation is done for performance reasons.
- Jasper files are what you need to ship with your application in order to run the reports.

Executing the report(Filling data into the report)

- In this step data from the application is filled in the compiled report.
- The class net.sf.jasperreports.engine.JasperFillManager provides necessary functions to fill the data in the reports.
- A Jasper print file (*.jrprint) is created, which can be used to either print or export the report.

Exporting the report to desired format

- In this step we can export the Jasper print file created in the previous step to any format using JasperExportManager.
- As Jasper provides various forms of exports, hence with the same input we can create multiple representations of the data.