13BECS7E10 C # AND . NET FRAMEWORK 3 0 0 3

INTENDED OUTCOMES:

- To cover the fundamental concepts of the C# language and the .NET framework.
- The student will gain knowledge in the concepts of the .NET framework as a whole and the technologies that constitute the framework.
- The student will gain programming skills in C# both in basic and advanced levels.
- By building sample applications, the student will get experience and be ready for large-scale projects.

UNIT- I INTRODUCTION TO C#

Introducing C#, Understanding .NET, Overview of C#, Literals, Variables, Data Types, Operators, Expressions, Branching, Looping, Methods, Arrays, Strings, Structures, Enumerations.

UNIT –II OBJECT ORIENTED ASPECTS OF C#

Classes, Objects, Inheritance, Polymorphism, Interfaces, Operator Overloading, Delegates, Events, Errors and Exceptions.

UNIT- III APPLICATION DEVELOPMENT ON .NET

Building Windows Applications, Accessing Data with ADO.NET.

UNIT- IV WEB BASED APPLICATION DEVELOPMENT ON .NET

Programming Web Applications with Web Forms, Programming Web Services.

UNIT -V THE CLR AND THE .NET FRAMEWORK

Assemblies, Versioning, Attributes, Reflection, Viewing MetaData, Type Discovery, Reflecting on a Type, Marshaling, Remoting, Understanding Server Object Types, Specifying a Server with an Interface, Building a Server, Building the Client, Using SingleCall, Threads.

TEXT BOOKS:

S.NO	**Author(s) Name**	**Title of the book**	**Publisher**	**Year of publication**
1	E. Balagurusamy	Programming in C#	Tata McGraw-Hill	2010
2	J. Liberty	Programming C# (2nd Edition)	O'Reilly	2007

REFERENCE BOOKS:

S.NO	**Author(s) Name**	**Title of the book**	**Publisher**	**Year of publication**
1	Herbert Schildt	The Complete REFERENCE BOOKS: C#	Tata McGraw-Hill	2004
2	Robinson et al	Professional C# (2nd edition)	Wrox Press	2004
3	Andrew Troelsen	C# and the .NET Platform	A! Press	2003
4	Thamarai Selvi.S, Murugesan.R	A Textbook on C#	Pearson Education	2012

KARPAGAM ACADEMY OF HIGHER EDUCATION

(*Deemed to be University Established Under Section 3 Of UGC Act 1956*)
Pollachi Main Road, Eachanari Post, Coimbatore – 641 021. INDIA
Phone No: 0422-6471113-4, 6453777 Fax No: 0422 – 2980022-23
Email ID: info@karpagam.com Web: www.kahedu.edu.in

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Subject Code: 13BECS7E10

Name of the Subject : C # AND .NET FRAMEWORK

S.NO	TABLE OF CONTENTS	Lecture Hours	Books / Websites referred	PAGE NO	Teaching Aids
UNIT I INTRODUCTION TO C#					
1	Introducing C# and Understanding .NET	1	T2	9 - 13	T2
2	Overview of C# Literals Variables, and Data Types	1	T2	26 - 30	BB
3	Operators, Expression and Branching	1	T2	36 - 44	BB
4	**Tutorial 1 Revision on Literals Variables, Data Types, Operators and Expression with Examples**	1			BB
5	Branching	1	T2	36 - 44	BB
6	Looping and Arrays	1	T2	45- 51 and 156	BB
7	Structures	1	T2	123-129	BB
8	**Tutorial 2 Branching, Looping and Arrays with Examples**	**1**			**BB**
9	Structures and Enumerations	1	T2	123-129 and 33 – 35	BB

UNIT IV WEB BASED APPLICATION DEVELOPMENT ON .NET					
28	Visual Studio ASP.NET Web Applications	1	T1	42	BB
	Programming Web Applications with Web Forms		T1	42	BB
29	Windows Forms	1	T1	43	BB
	Web Forms		T1	43	PPT
30	**Tutorial 9 - Comparing Windows Forms and Web Forms**	1	T1	44	BB
31	Event Handling in Windows Forms Introduction to Events in Windows Forms	1	T1	45 - 46	BB
32	Introduction to Event Handlers in Windows Forms	1	T1	46	BB
	Creating Event Handlers on the Windows Forms Designer		T1	47	PPT
33	**Tutorial 10 - Creating Default Event Handlers on the Windows Forms Designer**	1	T1	48	BB
34	Creating Event Handlers at Run Time for Windows Forms	1	T1	49	BB
	Connecting Multiple Events to a Single Event Handler in Windows Forms		T1	50	BB
35	Creating Event Handlers in the Visual Basic Code Editor	1	T1	51	PPT
36	**Tutorial 11 - Programming Web Services (Tutorial)**	1	T1	57	BB
UNIT V GRAPHS					
37	Assemblies	1	T1	65	BB
	Versioning		T1	67	BB
38	Attributes	1	T1	70	BB
	Reflection		T1	72	BB
40	**Tutorial 12 - Viewing Metadata**	1	T1	72	BB
41	Type Discovery	1	T1	73	BB
	Reflecting on a Type, Marshalling		T1	74	BB
	Remoting		T1	75	BB
42	Understanding Server Object Types	1	T1	76	BB
43	**Tutorial 13 - Specifying a Server with an Interface**	1	T1	77	BB
	Building a Server		T1	77	BB

44	Building the Client	1	T1	80	BB
	Using Single Call		T1	82	BB
45	**Tutorial 14 - Threads**	1	T1	83	BB

Total no. of Lecture Hours : 31
Total no. of Tutorial Hours : 14
Total no. of Hours : 45

TEXT BOOKS :

S.NO	Author(s) Name	Title of the book	Publisher	Year of publication
1	E. Balagurusamy	Programming in C#	Tata McGraw-Hill	2010
2	J. Liberty	Programming C# (2nd Edition)	O'Reilly	2007

REFERENCE BOOKS:

S.NO	Author(s) Name	Title of the book	Publisher	Year of publication
1	Herbert Schildt	The Complete REFERENCE BOOKS: C#	Tata McGraw-Hill	2004
2	Robinson et al	Professional C# (2nd edition)	Wrox Press	2004
3	Andrew Troelsen	C# and the .NET Platform	A! Press	2003
4	Thamarai Selvi.S, Murugesan.R	A Textbook on C#	Pearson Education	2012

Staff – In Charge **HOD/CSE**

UNIT- I

INTRODUCTION TO C#

Introducing C#

C# (pronounced "C sharp") is a simple, modern, object-oriented, and type-safe programming language. It will immediately be familiar to C and C++ programmers. C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++.

Visual C# .NET is Microsoft's C# development tool. It includes an interactive development environment, visual designers for building Windows and Web applications, a compiler, and a debugger. Visual C# .NET is part of a suite of products, called Visual Studio .NET, that also includes Visual Basic .NET, Visual C++ .NET, and the JScript scripting language. All of these languages provide access to the Microsoft .NET Framework, which includes a common execution engine and a rich class library. The .NET Framework defines a "Common Language Specification" (CLS), a sort of lingua franca that ensures seamless interoperability between CLS-compliant languages and class libraries. For C# developers, this means that even though C# is a new language, it has complete access to the same rich class libraries that are used by seasoned tools such as Visual Basic .NET and Visual C++ .NET. C# itself does not include a class library.

Understanding .NET

Visual Studio .NET is a complete set of development tools for building ASP Web applications, XML Web services, desktop applications, and mobile applications. Visual Basic .NET, Visual C++ .NET, Visual C# .NET, and Visual J# .NET all use the same integrated development environment (IDE), which allows them to share tools and facilitates in the creation of mixed-language solutions. In addition, these languages leverage the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP Web applications and XML Web services.

Overview of C#

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by Anders Hejlsberg and his team during the development of .Net Framework.

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 on different computer platforms and architectures.

The following reasons make C# a widely used professional language:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

Strong Programming Features of C#

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 a number of programmers worldwide.

Following is the list of few important features of C#:

- 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

Literals

A literal is a source code representation of a value.

```
literal:
        boolean-literal
        integer-literal
        real-literal
        character-literal
        string-literal
        null-literal
```

Variables

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:

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

C# also allows defining other value types of variable such as **enum** and reference types of variables such as **class**, which we will cover in subsequent chapters.

Defining Variables

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, and variable_list may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

You can initialize a variable at the time of definition as:

```
int i = 100;
```

Initializing Variables

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as:

```
<data_type> <variable_name> = value;
```

Some examples are:

```
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'. */
```

It is a good programming practice to initialize variables properly, otherwise sometimes program may produce unexpected result.

Data Types

The variables in C#, are categorized into the following types:

- Value types
- Reference types
- Pointer types

Value Type

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, and float**, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an **int** type, the system allocates memory to store the value.

Reference Type

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 multiple variables, 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**.

Pointer Type

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

Syntax for declaring a pointer type is:

type* identifier;

For example,

char* cptr;
int* iptr;

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# has rich set of built-in operators and provides the following type of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Expressions

Operator precedence determines the grouping of terms in an expression. This affects evaluation of an expression. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so the first evaluation takes place for 3*2 and then 7 is added into it.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators are evaluated first.

Examples

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right

Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Branching

 Branching Statements

- ***if Statement :***

1. It executes its block only if the condition is true
2. **Syntax:**
```
if ( condition )
{
//statement;
}
```

- ***if else Statement***

1. It executes if block if condition is true; otherwise it will execute the else block
2. **Syntax:**
```
if (condition                                                    )
{
//statement;
}
else
{
//statement;
}
```

- ***else if Statement:***

1. It checks the condition of both if and else if block and executes the respective block; otherwise it will execute the else block.
2. **Syntax:**
```
if (                                                   condition)
{
//statement;
}
else                     if(                           condition)
{
```

```
//statement;
}
else
{
//statement;
}
```

- ***Switch Statement***

1. The switch block consists of several cases which includes a default case too.
2. Each case has break statement to jump out of switch block on its execution.
3. The cases are matched and then executed provided the condition for cases in switch statement..
4. **Syntax:**

```
switch                                                    (variable)
{
case 1:
//statement;
break;
case 2:
//statement;
break;
default:
//statement;
break;
}
```

Looping

- Looping **Statements**

- ***while Statement***
    1. It executes the block until the condition fails.
    2. It will execute its block only if the condition is true and continues to loop
    3. **Syntax:**
    ```
    while ( condition )
    {
    //statement;
    }
    ```
- ***do-while Statement***
    1. It executes its statements and checks the condition.
    2. It continues looping if condition is true; else it aborts.
    3. **Syntax:**
    ```
    do
    {
    //statement;
    }
    while ( condition );
    ```
- ***for Statement***

1. It executes its block until the condition fails.
2. **Syntax:**
   ```
   for( initialization; condition; iteration)
   {
   //statement;
   }
   ```

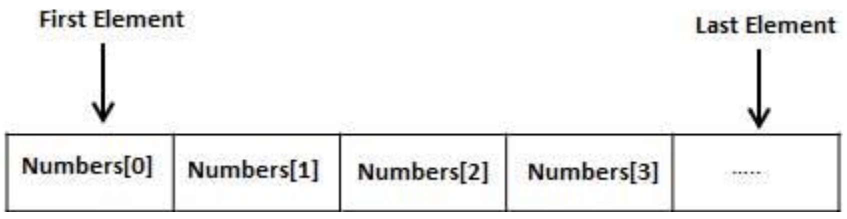
- ***foreach Statement***
  1. It executes the block for each values.
  2. **Syntax:**
     ```
     foreach (datatype values in variable)
     {
     //statement;
     }
     ```

Arrays

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 stored at contiguous memory locations.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C#, you can use the following syntax:

datatype[] arrayName;

where,

- d*atatype* is used to specify the type of elements 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;
```

Strings

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 an object to its string representation

Structures

In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The **struct** keyword is used for creating a structure.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title
- Author
- Subject
- Book ID

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member for your program.

For example, here is the way you can declare the Book structure:

```
struct Books
{
  public string title;
  public string author;
  public string subject;
  public int book_id;
};
```

Enumerations

An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.

C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

Declaring *enum* Variable

The general syntax for declaring an enumeration is:

```
enum <enum_name>
{
   enumeration list
};
```

Where,

- The *enum_name* specifies the enumeration type name.
- The *enumeration list* is a comma-separated list of identifiers.

Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0.

For example:

enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

UNIT –I

TWO MARKS

1. **How is c# better than java?**
   C# is a new language developed exclusively to suit the features of .NET platform. It can be used for a variety of application that is supported by .NET platform:

Console application
Windows application
Developing windows controls
Developing ASP.NET projects
Creating web controls
Providing web services

Developing .NET component library

2.What are the differences between the c# and c++? * C# complies straight from source code to executable code, with no object files. * In c++, class definition does not use a semicolon at the end * C# does not support #include statement * In c#, switch can also be used on string values.

3. What are the differences between the c# and java?
Although c# uses .net run time that is similar to java runtime the c# complier produces an

Executable code.
C# has more primitive data types
Unlike java, all c# data types are objects.
Arrays are declared differently in c++

4. State some of the new features that are unique to c# language?
It is a brand new language derived from the c/c++ family.
It simplifies and modernizes c++
It is the only component oriented language available today
It is a concise, lean and modern language
It has a lean and consistent syntax.

5. What is .net? .net is the software framework that includes everything required for developing software for web services.

6. What is managed code? The code that satisfies the CLR at runtime in order to execute is referred to as managed code. For example c# complier generates II, code(managed code).

7. What is the importance of the main method in a c# program? C# includes a feature that enables us to define more than one class with the main method. Since main is the entry point for program execution there are now than one entry points. In facts there should be only one. This problem can be resolved by specifying which main is to be used to the complier at the time of compilation as shown below;Csc filename.cs/main: classname

8. What is a local variable? Variable declared inside methods are called local variables. Local variable can also be declared inside program blocks that are defined between an opening brace {and the closing brace}. The scope of a local variable starts after immediately after its identifier in the declaration and extends up to the end of the block containing the declaration.

9. What is initialization? Why is it necessary? A variable must be given a value after it has been declared but before it is used in an expression. A simple method of giving value to a variable is through the assignment statement as follow, Variable name=value;

10.Why do we require type conversion operation? There is a need to convert a data of one type to another before it is used in arithmetic operations or to store a value of one type into a variable of another type .for example consider the code below : byte b1 = 50; byte b2 = 60; byte b3 = b1 + b2;

UNIT –II

OBJECT ORIENTED ASPECTS OF C#

Classes

When you define a class, you define a blueprint for a data type. This does not actually define any data, but it does define what the class name means. That is, what an object of the class consists of and what operations can be performed on that object. Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

Defining a Class

A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces. Following is the general form of a class definition:

```
<access specifier> class  class_name
{
  // member variables
  <access specifier> <data type> variable1;
  <access specifier> <data type> variable2;
  ...
  <access specifier> <data type> variableN;
  // member methods
  <access specifier> <return type> method1(parameter_list)
  {
    // method body
  }
  <access specifier> <return type> method2(parameter_list)
  {
    // method body
  }
  ...
  <access specifier> <return type> methodN(parameter_list)
  {
    // method body
  }
}
```

Note:

- Access specifiers specify the access rules for the members as well as the class itself. If not mentioned, then the default access specifier for a class type is **internal**. Default access for the members is **private**.
- Data type specifies the type of variable, and return type specifies the data type of the data the method returns, if any.
- To access the class members, you use the dot (.) operator.
- The dot operator links the name of an object with the name of a member.

Objects

A class or struct definition is like a blueprint that specifies what the type can do. An object is basically a block of memory that has been allocated and configured according to the blueprint. A program may create many objects of the same class. Objects are also called instances, and they can be stored in either a named variable or in an array or collection. Client code is the code that uses these variables to call the methods and access the public properties of the object. In an object-oriented language such as C#, a typical program consists of multiple objects interacting dynamically.

Inheritance

One of the most important concepts in object-oriented programming is 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 speeds up 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.

The idea of inheritance implements the **IS-A** relationship. For example, mammal **IS A** animal, dog **IS-A** mammal hence dog **IS-A** animal as well, and so on.

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.

Interfaces

An interface is defined as a syntactical contract that all the classes inheriting the interface should follow. The interface defines the **'what'** part of the syntactical contract and the deriving classes define the **'how'** part of the syntactical contract.

Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

Abstract classes to some extent serve the same purpose, however, they are mostly used when only few methods are to be declared by the base class and the deriving class implements the functionalities.

Declaring Interfaces

Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default. Following is an example of an interface declaration:

```
public interface ITransactions
{
   // interface members
   void showTransaction();
   double getAmount();
}
```

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. similar to any other function, an overloaded operator has a return type and a parameter list.

For example, go through 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;
}
```

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.

Delegates

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

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

For example, consider a delegate:

```
public delegate int MyDelegate (string s);
```

The preceding delegate can be used to reference any method that has a single *string* parameter and returns an *int* type variable.

Syntax for delegate declaration is:

```
delegate <return type> <delegate-name> <parameter list>
```

Events

Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.

Errors and Exceptions

An exception is a problem that arises during the execution of a program. A C# 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. C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

- **try**: A try block identifies a block of code for which particular exceptions is activated. It is 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.

Syntax

Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. 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
{
  // statements causing exception
}
catch( ExceptionName e1 )
{
  // error handling code
}
catch( ExceptionName e2 )
{
  // error handling code
}
catch( ExceptionName eN )
{
  // error handling code
}
finally
{
  // statements to be executed
}
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

UNIT- II

TWO MARKS

1. What is a constructor?

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type. The general form of constructor if shown here: Access class-name () {//constructor code}

2. What is a static constructor? A static constructor is called before any object of the class is created. This is useful to do any housekeeping work that needs to be done once. it is usually used to assign initial values to static data members.

3. What are the restrictions of static methods?
That there is no access modifier on static constructors. It cannot take any. A class can have only one static constructor.

4. What is property?
Another type of class members is the property. A property combines a field with the methods that access it. If you want to create a field that is available to users of an objects, but you want to maintain control over the operations allowed on that field.

5. What is read-only property?
There are situations where we would like to decide the value of a constant member at run-time. We may also like to have different constant values for different objects of the class. To overcome these shortcomings, c# provides another modifier known as read only

6. What is an indexer?
Array indexing is performed using the []operator. An indexer allows an object to be indexed like an array. The main use of indexes is to support the creation of specified arrays that are subjects to one or more constraints.

7. What are the two form of inheritance?
The two form of inheritance classical form, containment form

8. What is hiding a method?
We can use the modifier new to tell the compiler that the derived class method "hides" the base class method.

9. What is polymorphism?

Polymorphism means ’one name, many form’. Polymorphism can be achieved in two ways. c# supports both of them *Operation polymorphism *Inclusion polymorphism

10. What is early binding?

The compiler is able to select and bind the appropriate method to the object for a particular call at complier time itself. This process is called early binding, or static binding, it is also known as complier time polymorphism.

UNIT- III

APPLICATION DEVELOPMENT ON .NET

Building Windows Applications

A windows application is an application that runs on windows desktop. Windows applications will have graphical user interface (GUI). Because of GUI, designing the application will be easy and fast.

System.Windows.Forms namespace used to build Windows-based applications.

To Develop Windows Application. Open Visual Studio

Click on File-> New -> Project -> Visual C#(or lang you want)->Windows->Windows Form Application.

Provide suitable name to the windows application as shown in the screen shot below

Developing Winforms application .net

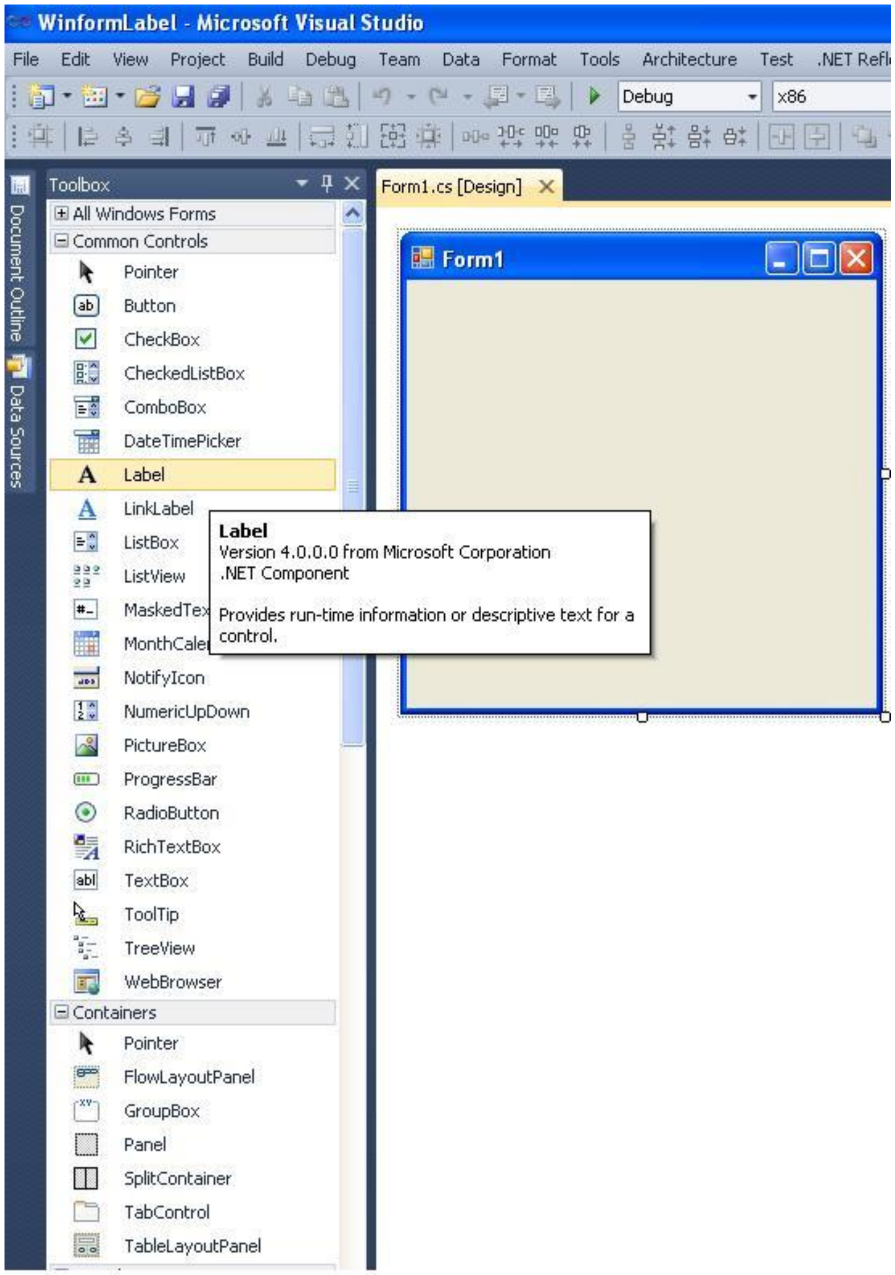
Example for Windows Application

Label in C# Winforms

Label is used to display the static text on windows form.

In the below screenshot, you can see a form.
Now if you want to add some text on the form, double click on the label control in the control’s toolbox. The label control will be added to the form.

Label in C# Winforms .NET



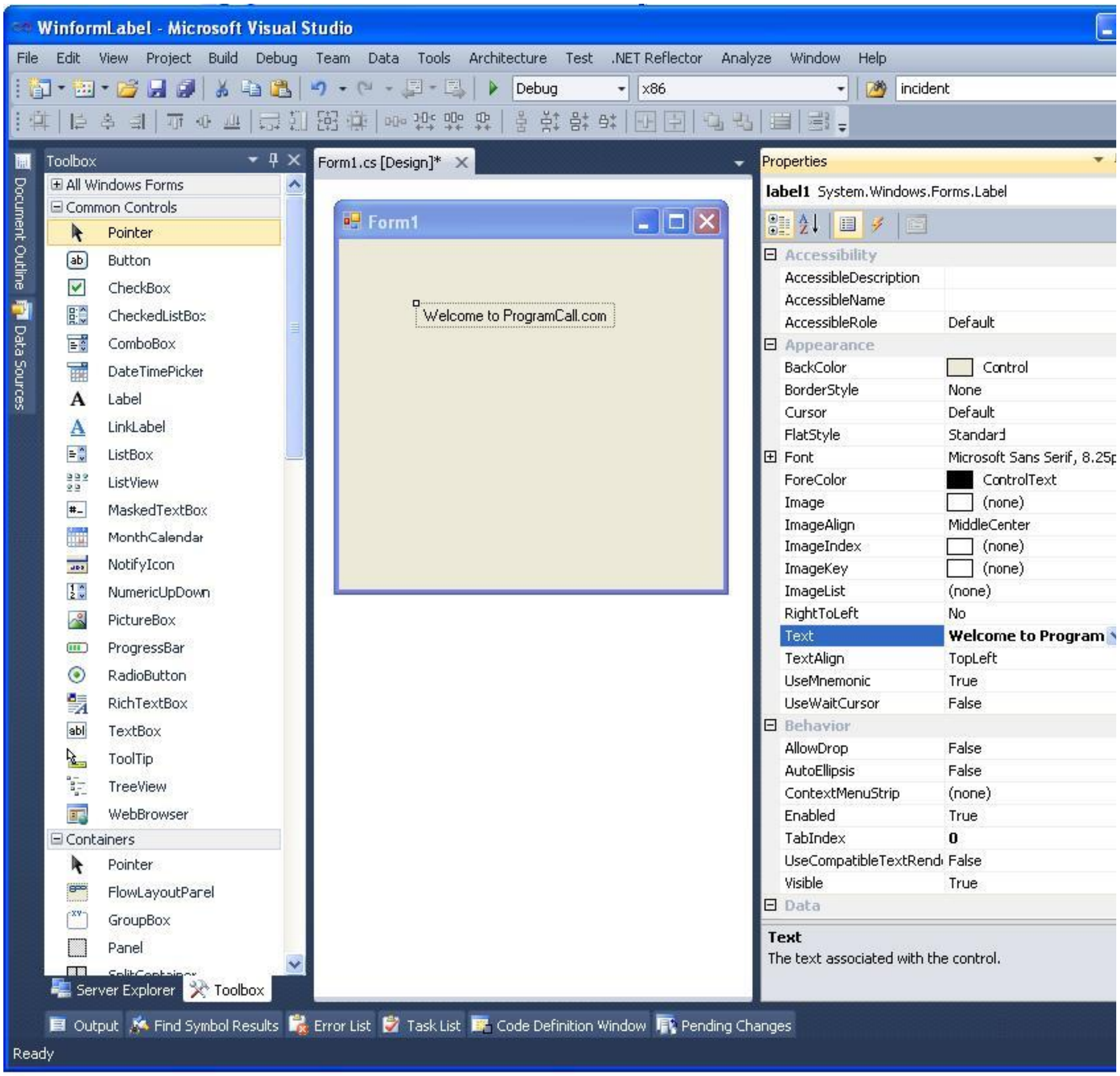
Label
Version 4.0.0.0 from Microsoft Corporation
.NET Component

Provides run-time information or descriptive text for a control.

Now select the Label and press F4. The Label Properties windows will be text. Find the property - Text and add the text you want . The text you added will be displayed on the form.

Thus the label control is used to display the static text on the windows form.

Label Control in C# Winforms .NET



label1 System.Windows.Forms.Label

Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
BorderStyle	None
Cursor	Default
FlatStyle	Standard
Font	Microsoft Sans Serif, 8.25p
ForeColor	ControlText
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	**Welcome to Program**
TextAlign	TopLeft
UseMnemonic	True
UseWaitCursor	False
Behavior	
AllowDrop	False
AutoEllipsis	False
ContextMenuStrip	(none)
Enabled	True
TabIndex	**0**
UseCompatibleTextRend	False
Visible	True
Data	

Text
The text associated with the control.

You can also add the label text in program as below

```
//Adding label text in program
        label1.Text = "Welcome to ProgramCall.com";
```

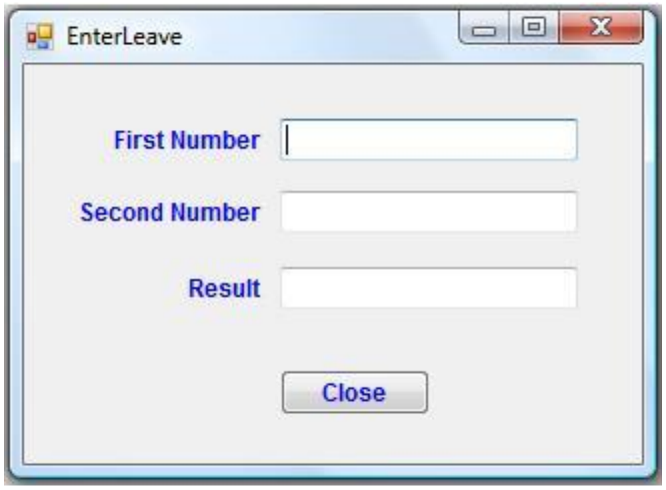
Example for winforms keypress event

Keypress event is used when you want to write the code based on the keys having ASCII value. The second argument e for keypress event is of type **keypresseventargs** and it has a property keychar that contains the character pressed by the user on the keyboard and this property is used to write the codein keypress event.

Within the keypress event it is not possible to access modifier keys. The second argument e in keypress event has a property Handled that can be used to exit from keypress event in middle. For this set **Handled** property to true.

Example : The following example accepts two numbers from user and performs addition or subtraction or multiplication or division based on the keys A, S, M and D respectively. C will clear the text boxes and X will close the form.

1. Add a win form to the project and design it as follows.



2. Set the following properties to the controls.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

3. Write the following code within the keypress event of the form. For creating the keypress event for the form, open properties of the form and within the properties window click on events button and then double click on keypress event.

```
private void KeypressEvent_KeyPress(object sender, KeyPressEventArgs e)
```

```
        {
            switch (e.KeyChar)
            {
                case 'A':
                case 'a':
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
Convert.ToInt32(TxtNum2.Text)).ToString();
                    e.Handled = true;
                    break;
                case 'S':
                case 's':
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) -
Convert.ToInt32(TxtNum2.Text)).ToString();
                    e.Handled = true;
                    break;
                case 'M':
                case 'm':
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) *
Convert.ToInt32(TxtNum2.Text)).ToString();
                    e.Handled = true;
                    break;
                case 'D':
                case 'd':
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) /
Convert.ToInt32(TxtNum2.Text)).ToString();
                    e.Handled = true;
                    break;
                case 'C':
                case 'c':
                    TxtNum1.Clear();
                    TxtNum2.Clear();
                    TxtResult.Clear();
                    TxtNum1.Focus();
                    e.Handled = true;
                    break;
                case 'X':
                case 'x':
                    this.Close();
                    break;
            }
        }
```

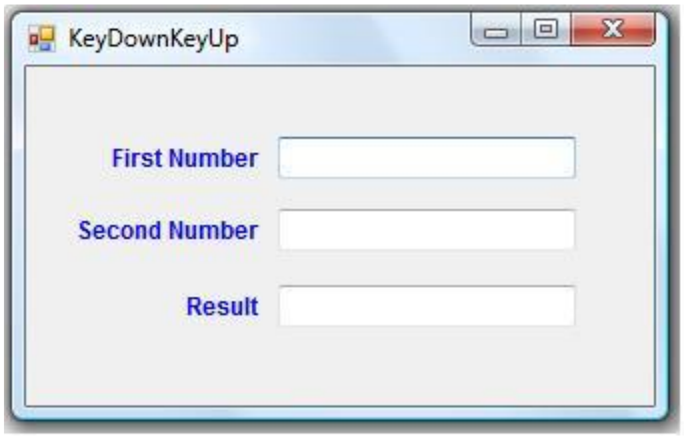
Keypress event of the text box can be used to restrict the text box to accept only alphabets or only digits or only specified characters.

Example for Winforms keydown event with modifier keys

Control, shift and alt keys are called **modifier** keys. If you want to execute the code in keydown or keyup event only when user press a key along with modifier keys then use modifiers property of second argument e that contains the modifier keys pressed along with the key pressed on the keyboard.

Example : The following example accepts two numbers from user and performs addition or subtraction or multiplication or division based on the key combination Ctrl+F1, Ctrl+F2, Ctrl+F3 and Ctrl+F4 respectively. Ctrl+F5 will clear the text boxes and Escape will close the form.

1. Add a win form to the project and design it as follows.



2. Set the following properties to the controls.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

3. write the following code within the keydown event of the form. For creating the keydown event for the form, open properties of the form and within the properties window click on events button and then double click on keydown event.

```
private void KeyDownModifiers_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Modifiers == Keys.Control)
            {
                switch (e.KeyCode)
                {
                    case Keys.F1:
                        TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
Convert.ToInt32(TxtNum2.Text)).ToString();
                        break;
                    case Keys.F2:
                        TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) -
Convert.ToInt32(TxtNum2.Text)).ToString();
                        break;
                    case Keys.F3:
                        TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) *
Convert.ToInt32(TxtNum2.Text)).ToString();
                        break;
                    case Keys.F4:
                        TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) /
Convert.ToInt32(TxtNum2.Text)).ToString();
                        break;
                    case Keys.F5:
                        TxtNum1.Clear();
                        TxtNum2.Clear();
                        TxtResult.Clear();
                        TxtNum1.Focus();
                        break;
                }
            }
            if (e.KeyCode == Keys.Escape)
                this.Close();
        }
```

4. Run the application with shortcut F5.

5. if you want to check for the condition with the combination of more than one modifier keys then write the condition by specifying the modifier keys in parentheses separated by single pipe(|) symbol .

```
if (e.Modifiers == (Keys.Control | Keys.Alt))
{
}
```

Working with forms keyup and keydown events

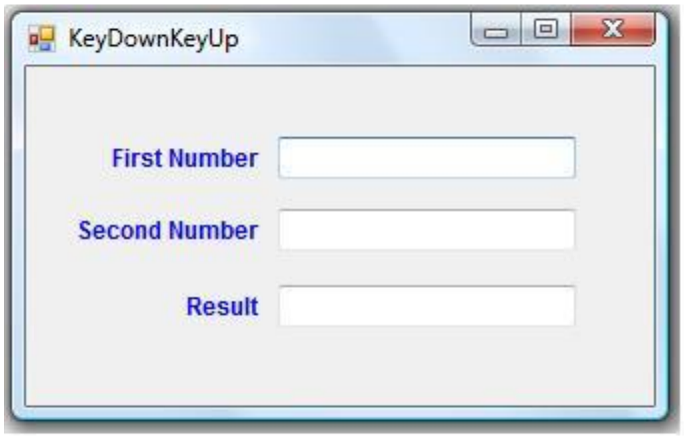
Keydown and keyup events are used when you want to write the code based on the keys having keycode but not ASCII value.

The second argument e for keydown and keyup events is of type **keyeventargs** and it has a property **keycode** that contains the keycode of the key pressed by the user on the keyboard and this property is used to write the code in either keydown or keyup event.

When you are writing the code in keyboard events of the form, then the **keypreview** property of the form must be set to true, because by default only keyboard events of the control that contains the focus will be raised and not the keyboard events of the form. When you set keypreview property true, then the keyboard events of both the control that contains the focus and form will be raised.

Example : The following example accepts two numbers from user and performs addition or subtraction or multiplication or division based on the function keys F1,F2,F3 and F4 respectively. F5 will clear the text boxes and Escape will close the form.

1. Add a win form to the project and design it as follows.



2. Set the following properties to the controls.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

3. write the following code within the keydown event of the form. For creating the keydown event for the form, open properties of the form and within the properties window click on events button and then double click on keydown event.

```
private void KeyDownKeyUp_KeyDown(object sender, KeyEventArgs e)
        {
            switch (e.KeyCode)
            {
                case Keys.F1:
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
Convert.ToInt32(TxtNum2.Text)).ToString();
                    break;
                case Keys.F2:
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) -
Convert.ToInt32(TxtNum2.Text)).ToString();
                    break;
                case Keys.F3:
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) *
Convert.ToInt32(TxtNum2.Text)).ToString();
                    break;
                case Keys.F4:
                    TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) /
Convert.ToInt32(TxtNum2.Text)).ToString();
                    break;
                case Keys.F5:
                    TxtNum1.Clear();
                    TxtNum2.Clear();
                    TxtResult.Clear();
                    TxtNum1.Focus();
                    break;
                case Keys.Escape:
                    this.Close();
                    break;
            }
        }
```

4. Run the application with shortcut F5.

Control, **shift** and **alt** keys are called modifier keys. If you want to execute the code in keydown or keyup event only when user press a key along with modifier keys then use modifiers property of second argument e that contains the modifier keys pressed along with the key pressed on the keyboard.

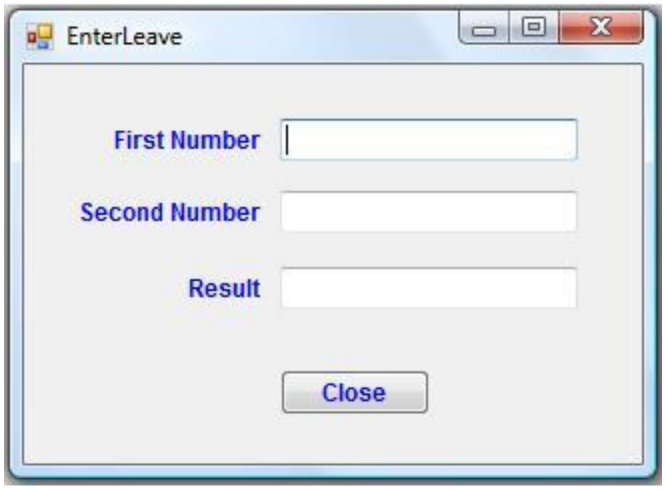
Working With Winform Textbox Enter And Leave Events

When focus enters in to a control then enter event will be raised and when a control lost focus then leave event will be raised. These events can be used to write the code that needs to be executed when a control got and lost the focus.

1. Create a windows application and design the form as follows.

Open Visual Studio ->File -> New Project ->Visual C#-> select Windows Forms Application

Give the name of the application and click on OK



2. Set following properties to the controls on the form. Select the control and press F4. the properties window will be opened and set the properties.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult
Button	Name	: BtnClose
	Text	: Close

3. Choose leave event of second textbox txtnum2 by opening properties window, clicking on events button and then double clicking on leave event and write the following code in it.

```
private void TxtNum2_Leave(object sender, EventArgs e)
        {
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) + 
Convert.ToInt32(TxtNum2.Text)).ToString();
        }
```

4. Double Click on close button and write the following code in its click event to close the form.

```
private void BtnClose_Click(object sender, EventArgs e)
{
    this.Close();
}
```

5. Run the application with shortcut F5.

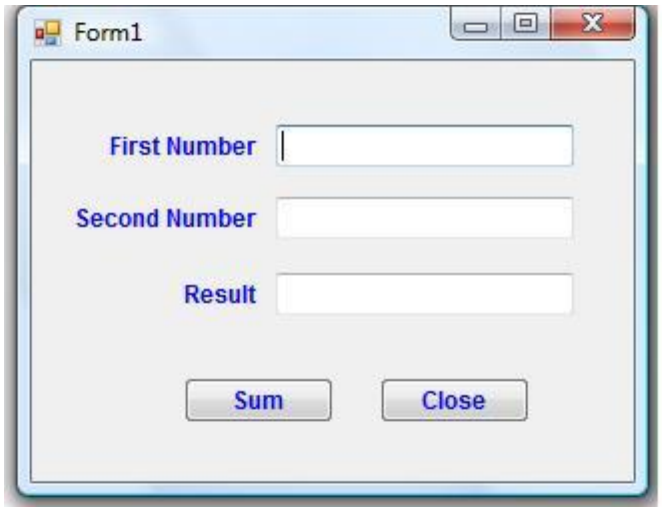
Example for Windows Application

The following example creates a C# windows Form application with three text boxes and two button controls. When user enters two numbers in to first two text boxes and clicks on first button then sum has to be calculated for those two numbers and display the result in third text box.

1. Create a windows application and design the form as follows.

Open Visual Studio ->File -> New Project ->Visual C#-> select Windows Forms Application

Give the name of the application and click on OK



2. Set the following properties for the controls on the form.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

Button1	Name	: BtnSum
	Text	: Sum
Button2	Name	: BtnClose
	Text	: Close

3. Double click on the sum button and write the following code in the click event of that button to calculate sum and display the result in result textbox

```
private void BtnSum_Click(object sender, EventArgs e)
        {
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
Convert.ToInt32(TxtNum2.Text)).ToString();
        }
```

4. Double click on close button and write the following code in the click event of that button to close the form.

```
private void BtnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }
```

5. Run the Application with shortcut F5.

6. in the above example if you want to calculate sum when user press enter key and close the form when user press escape key, set sum button as acceptbutton for the form and close button as cancelbutton for the form.

Windows Forms Events or Winform Events

Every control has its own events and events are used to write the code for windows application.

Every event has two arguments; **sender** and **e**. sender contains the information regarding the control that raises the event and e contains data related to the event.

For all the events type of sender is object and type of e varies depending on the event.

The following set of events is common for almost every windows control.

Sno	EventName	Description
1	Click	Will be raised when user clicks on the control with any button of the mouse. In this event it is not possible to determine which button of the mouse was clicked by the user. Type of second argument e is **EventArgs**.
2	MouseClick	Will be raised when user clicks on the control with any button of the mouse. In this event you can determine which button of the mouse was clicked by the user. Type of second argument e is **MouseEventArgs**.
3	DoubleClick	Will be raised when user double clicks on a control with any button of the mouse. In this event it is not possible to determine which button of the mouse was double clicked by the user. Type of second argument e is **EventArgs**.

4	MouseDoubleClick	Will be raised when user double clicks on a control with any button of the mouse. In this event it is possible to determine which button of the mouse was double clicked by the user. Type of second argument e is **MouseEventArgs**.
5	Enter	Will be raised when the focus enters in to the control and type of second argument e is **EventArgs**.
6	Leave	Will be raised when the focus leaves the control and type of second argument e is **EventArgs**.
7	Validating	Will be raised immediately after the leave event and is used to write validation code. Type of second argument e is **CancelEventArgs**. In this event you can cancel navigation of cursor to next control when validation was failed.
8	Validated	Will be raised immediately after the validating event and is also used to write validation code. Type of second argument e is **EventArgs**. In this event you can't cancel navigation of cursor to next control when validation was failed.
9	KeyDown	Will be raised when any key of the keyboard was hold down. This event is used to handle the keys that doesn't have ASCII value and have keycode. Type of second argument e is **KeyEventArgs**.
10	KeyUp	Will be raised when any key of the keyboard was released. This event is used to handle the keys that doesn't have ASCII value and have keycode. Type of second argument e is **KeyEventArgs**.
11	KeyPress	Will be raised when any key of the keyboard was completely down. This event is used to handle the keys that have an ASCII value. Type of second argument e is **KeyPressEventArgs**.
12	MouseDown	Will be raised when a button of the mouse was down on the control and type of second argument e is **MouseEventArgs**.
13	MouseUp	Will be raised when a button of the mouse was up on the control and type of second argument e is **MouseEventArgs**. Will be raised when a button of the mouse was up on the control and type of second argument e is MouseEventArgs.
		MouseDown and MouseUp events are used to draw graphics and to start and stop drag and drop operations.
14	MouseEnter	Will be raised when mouse pointer enters in to the control and type of second argument e is EventArgs. Will be raised when mouse pointer enters in to the control and type of second argument e is EventArgs.
15	MouseLeave MouseLeave	Will be raised when mouse pointer leaves the control and type of second argument e is EventArgs.
		MouseEnter and MouseLeave events are used to change the icon while performing drag and drop operations based on

		whether drop is possible.
16	MouseMove	Will be raised when mouse pointer moves over the control and type of second argument e is **MouseEventArgs**.
17	MouseHover	Will be raised when mouse pointer was placed over the control constantly for at least one second and type of second argument e is EventArgs.

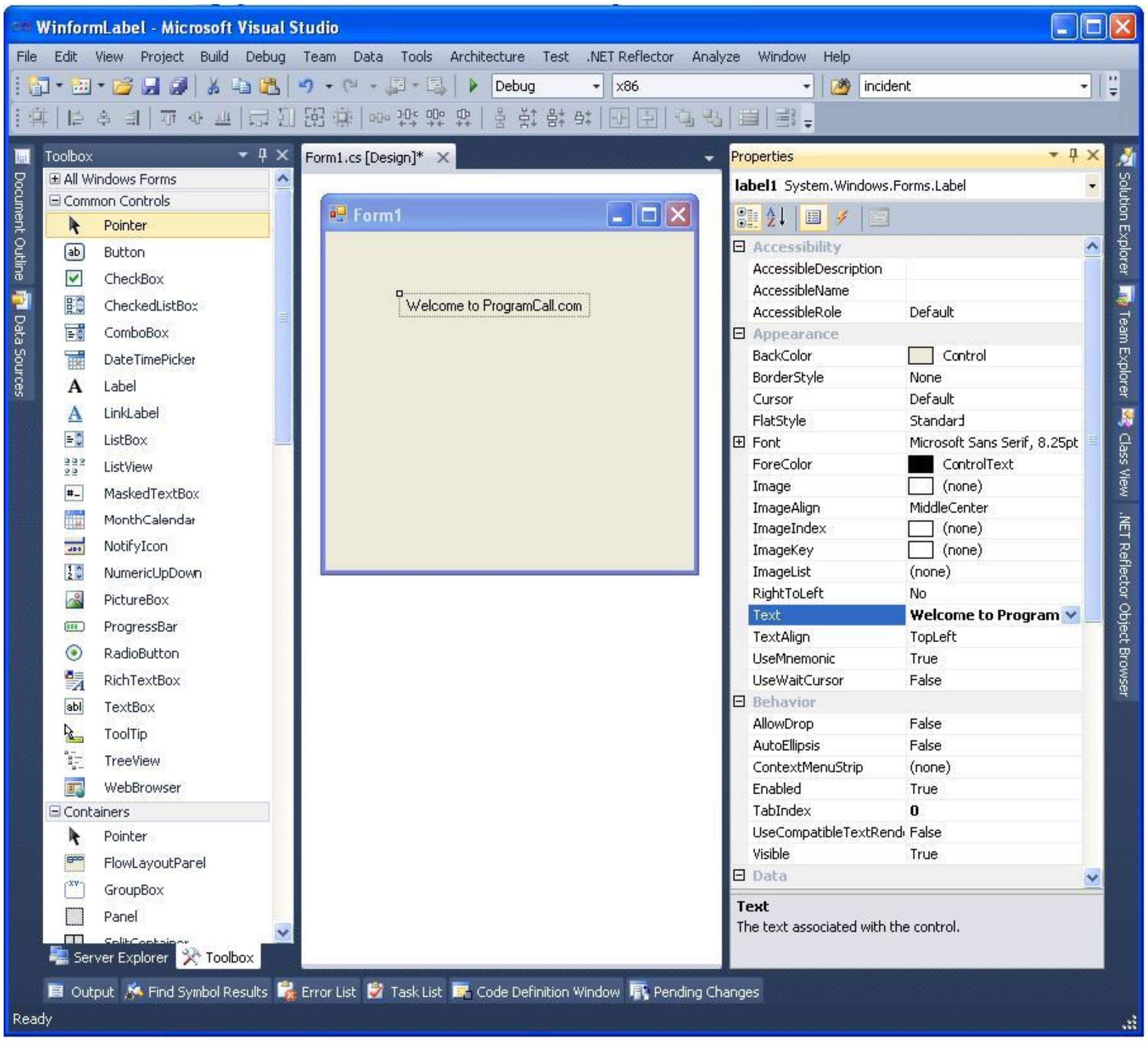
The following set of events is available only for a form.

Sno	**EventName**	**Description**
1	Load	This is the default event for form and will be raised whenever the form is loaded. This event can be used to initialize variables and type of second argument e is EventArgs.
2	FormClosing	Will be raised when the form is about to be closed. This event can be used to write the code that needs to be executed while the form is closed and type of second argument e is **FormClosingEventArgs**.
3	FormClosed	Will be raised immediately after FormClosing event. This event can be used to write the code that needs to be executed while the form is closed and type of second argument e is **FormClosedEventArgs**.
4	Resize	Will be raised when form is resized and type of second argument e is EventArgs.

Windows Application Form

A windows form is used to design and write the code for windows application.

Winform has design and code interfaces. To switch between design and code interfaces, use the shortcut F7. A windows form is also a class and it is created as partial class inherited from the class form available in system.windows.forms namespace. To refer to the current form in code, use the keyword this. A winform has the following important properties.



label1 System.Windows.Forms.Label

Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
BorderStyle	None
Cursor	Default
FlatStyle	Standard
Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	**Welcome to Program**
TextAlign	TopLeft
UseMnemonic	True
UseWaitCursor	False
Behavior	
AllowDrop	False
AutoEllipsis	False
ContextMenuStrip	(none)
Enabled	True
TabIndex	**0**
UseCompatibleTextRend	False
Visible	True
Data	

Text
The text associated with the control.

Windows Form Properties

Sno	Property Name	Decription
1	Name	Used to uniquely identify the control in code.
2	AcceptButton	Used to set a button available on the form as accept button so that the code written for that button is automatically executed when user press enter key wherever the focus is on the form.
3	AutoScroll	Indicates whether scroll bars are displayed when a control on the form is out side the boundary of the form.
4	BackColor	Used to set a background color for the control.

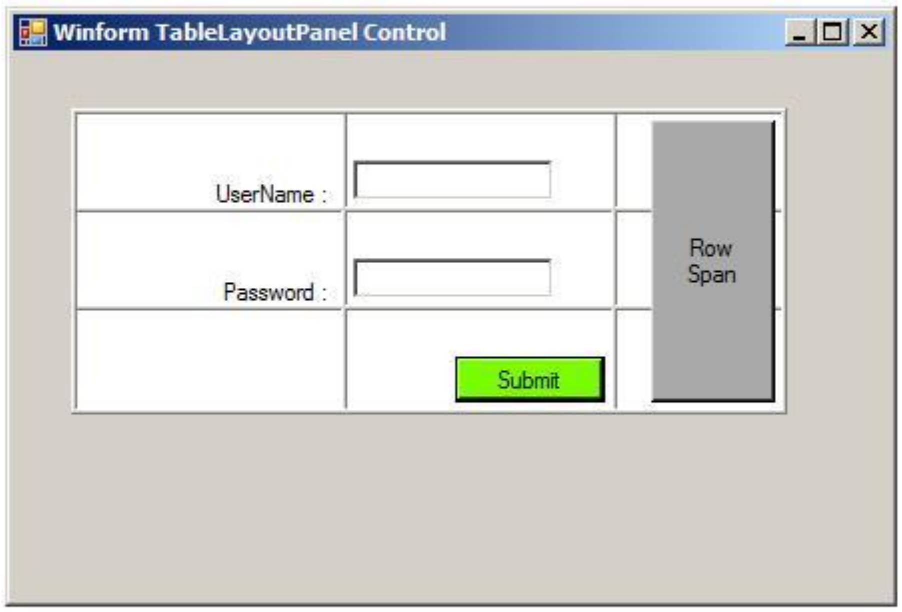
5	BackgroundImage	Used to display an image in the background of the form.
6	BackgroundImageLayout	Used to set layout for the background image on the form.
7	CancelButton	Used to set a button available on the form as cancelbutton so that the code written for that button is automatically executed when user press escape key wherever the focus is on the form.
8	ControlBox	Indicates whether minimize, maximize and close buttons and system icon are displayed in the title bar of the form.
9	Enabled	Indicates whether the events of the control are raised.
10	Font	Used to set font for the text displayed in the control
11	ForeColor	Used to set Font Color for the control.
12	Icon	Used to set the icon to display in the title bar of the form.
13	Location	Used to set position of the control from left and top.
14	Size	Used to set width and height of the control.
15	Text	Used to set a caption for the form.
16	WindowState	Indicates the initial state of the form when it was displayed for the first time. It has three possible values., minimized, maximized and normal.

Windows Form Methods

Sno	Method Name	Description
1	Close()	Used to close the form.
2	Hide()	Used to hide the form.
3	Show()	Used to display the form as non-modal window
4	ShowDialog()	Used to display the form as modal window.

c# tablelayoutpanel

Winforms Tablelayoutpanel is used to align winform controls in rows and columns.



When you add a control in tablelayoutpanel control cell, by default is positions in top left control of the cell, to change it to the required postion use the control's anchor property as below

```
//Use anchor property of control, to set the position of the control in tablelayoutpanel cell
        this.textBox1.Anchor = ((System.Windows.Forms.AnchorStyles)
           ((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)));
        this.label1.Anchor = ((System.Windows.Forms.AnchorStyles)
           ((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Right)));
        this.button2.Anchor = ((System.Windows.Forms.AnchorStyles)
           ((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
```

TableLayoutPanel RowSpan or ColumnSpan

To span control across multiple rows or columns , use contro'ls rowspan or columnspan property. The below lines of code can be used to span control across multiple rows or multiple columns.

```
//To span control across 3 rows
        this.tableLayoutPanel1.SetRowSpan(this.button2, 3);
        //To span control across 2 columns
        this.tableLayoutPanel1.SetColumnSpan(this.label1, 2);
```

Accessing Data with ADO.NET.

ADO.NET Code Snippets

A quick look at Database Technologies provided by Mircosoft

To access data from database like SQL server and oracle, .net framework provides ADO.Net. when we look at database technologies provided by Microsoft, they are as follows.

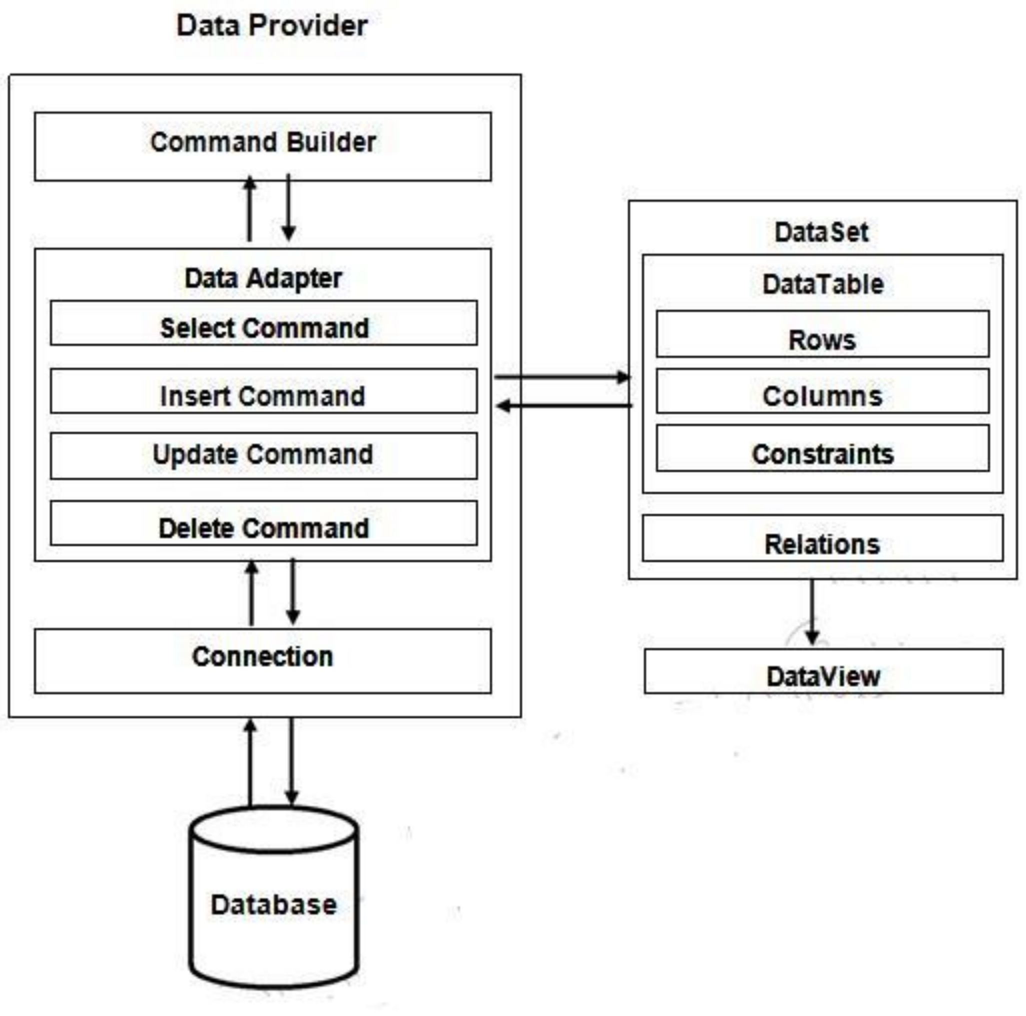
Database Technology	Description
DAO (Data Access Objects)	○ This works based on JetEngine or DBEngine. ○ It is efficient in accessing desktop databases like MS Access, Foxpro and Excel. ○ It is not efficient in accessing remote databases like SQL Server and Oracle.
RDO (Remote Data Objects)	○ This works based on ODBC(Open DatabaseConnectivity). ○ It is efficient in accessing remote databases. ○ It can not access the databases that don't support ODBC.
ADO (Activex Data Objects)	○ This works based on OLEDB (Object Linking and Embedding DataBase). ○ It can access almost any database that may be desktop database or remote database and that may support ODBC or may not support ODBC. ○ It is efficient than DAO and RDO. ○ It doesn't support pure architecture. ○ It doesn't support XML.
ADO.Net	○ To overcome the drawbacks of ADO, in .net, Microsoft provides ADO.Net. ○ Same as ADO it can access almost any database that may be desktop database or remote database and that may support ODBC or may not support ODBC. ○ It supports pure disconnected architecture and XML.

Data providers in .net

Different databases will have different storage formats. Different languages will support different data formats, this language formats will not be understandable to databases; this requires a translator between a language application and database. This translator is called driver or provider. Driver or provider is a software component, this acts like mediator between application and database.

Disconnected Architecture in ADO.NET

The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, data adapter, command builder an dataset and data view.



Connection : Connection object is used to establish a connection to database and connection it self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.

CommandBuilder : by default dataadapter contains only the select command and it doesn’t contain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.
To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

Da.Fill(Ds,”TableName”);

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

As connection to database was closed, any changes to the data in dataset will not be directly sent to the database and will be made only in the dataset. To send changes made to data in dataset to the database, **Update()** method of the dataadapter is used that has the following syntax.

Da.Update(Ds,”Tablename”);

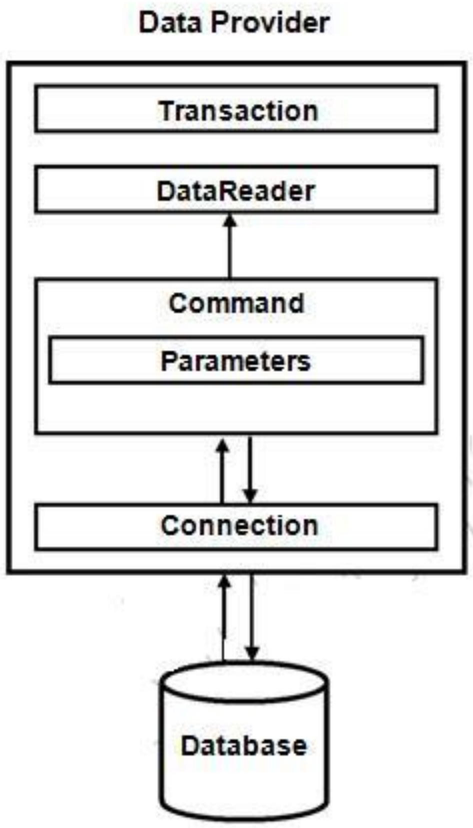
When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was not required, this architecture is called disconnected architecture.

A dataset can contain data in multiple tables.

DataView : DataView is a view of table available in DataSet. It is used to find a record, sort the records and filter the records. By using dataview, you can also perform insert, update and delete as in case of a DataSet.

Connected Architecture of ADO.NET

The architecture of ADO.net, in which connection must be opened to access the data retrieved from database is called as connected architecture. Connected architecture was built on the classes connection, command, data reader and transaction.



Connection : in connected architecture also the purpose of connection is to just establish a connection to database and it self will not transfer any data.

DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

To access one by one record from the DataReader, call **Read()** method of the DataReader whose return type is **bool.** When the next record was successfully read, the Read() method will return true and otherwise returns false.

Command Object in ADO.NET

Command : Command is used to execute almost any SQL command from within the .net application. The SQL command like insert, update, delete, select, create, alter, drop can be executed with command object and you can also call stored procedures with the command object. Command object has the following important properties.

- Connection : used to specify the connection to be used by the command object.
- CommandType : Used to specify the type of SQL command you want to execute. To assign a value to this property, use the enumeration CommandType that has the members Text, StoredProcedure and TableDirect. Text is the default and is set when you want to execute ant SQL command with command object. StoredProcedure is set when you want to call a stored procedure or function and **TableDirect** is set when you want to retrieve data from the table directly by specifying the table name without writing a select statement.
- CommandText : Used to specify the SQL statement you want to execute.
- Transaction : Used to associate a transaction object to the command object so that the changes made to the database with command object can be committed or rollback.

Command object has the following important methods.

- **ExecuteNonQuery() :** Used to execute an SQL statement that doesn't return any value like insert, update and delete. Return type of this method is int and it returns the no. of rows affected by the given statement.

- **ExecuteScalar() :** Used to execute an SQL statement and return a single value. When the select statement executed by executescalar() method returns a row and multiple rows, then the method will return the value of first column of first row returned by the query. Return type of this method is object.

- **ExecuteReader() :** Used to execute a select a statement and return the rows returned by the select statement as a DataReader. Return type of this method is **DataReader.**

ADO.NET Dataset vs DataReader

DataSet and DataReader are called fundamental objects of ADO.net as they are used to store data and make it available for .net application and they have the following differences.

Differences Between DataSet and DataReader

DataSet	DataReader
It is disconnected object and can provide access to data even when connection to database was closed.	It is connected object and can not provide access to data when connection to database was closed.
It can store data from multiple tables	It can store data from only one table.
It allows insert, update and delete on data	It is read only and it doesn't allow insert, update and delete on data.
It allows navigation between record either forward or backward.	It allows only forward navigation that also only to immediate next record.
It can contain multiple records.	It can contain only one record at a time.
All the data of a dataset will be on client system.	All the data of a DataReader will be on server and one record at a time is retrieved and stored in datareader when you call the Read() method of datareader.

UNIT- III

TWO MARKS

1. List out the different types of applications that can be created on .net?

.Net offers closely related but distinguishable suites of tools for building windows or web applications. Both are based on the premises that many applications have user interfaces centered on interacting with the user through form and controls, such as buttons, list boxes, text, and so forth

2. What are the advantages in using a dataset?

Dataset is a subset of the entire database cached on your machines without a continuous connection to the database-disconnected architecture which reduce burden on the database server which may help your application scale well.

3. Define relational database?

A database is a repository of data. A relational database organizes your data into tables. Consider the north wind database provided with Microsoft SQL server and Microsoft access.

4. What are the advantages of ADO.NET?

The significant advantage to disconnecting your data architecture from your database. The biggest advantage is that your application, whether running on the web or on a local machine, will create a reduced burden on the database server which may help your application to scale well. A disconnected architecture is resource frugal.

5. What is declarative reference integrity?

Relational database use declarative reference integrity to establish constraints on the relationship among the various tables. This helps avoid two types of mistakes. First, you can't enter a record with an invalid customer ID Secondly, you can't delete a customer record it that customerID is used in any order. The integrity of your data and its relationship is thus protected.

6. Define Data Adapter?

The dataset is an abstraction of relational data base.ADO.NET uses a data Adapter as a bridge between the dataset and data source, which is the underlying database.dataAdapter provides the Fill () method to retrieve data from the database and populate the dataset.

7. Define Data Reader?

The data reader provides connected, forward-only, read-only access to a Collection of tables. By executing either a SQL statement or stored procedures.datareader is a lightweight object.

8. What are the advantages of web applications?

- They can be accessed from any browser that can connect to the server. - Update can be made at the server - You can achieve better performance by building a desktop application.

9. What is the step to create windows application?

First open visual studio and file ----->new ----> project. In project window, create a new c# windows application and name it sharpwindowsform.

10. Differentiate between data reader and dataset?

The actually uses a data reader to populate itself. A data reader is lean, mean access Methods that return results as soon as they are available, rather than for the whole of the query to be populated into a dataset.

UNIT- IV

WEB BASED APPLICATION DEVELOPMENT ON .NET

Visual Studio ASP.NET Web Applications

A Visual Studio Web application is built around ASP.NET. ASP.NET is a platform — including design-time objects and controls and a run-time execution context — for developing and running applications on a Web server.

ASP.NET in turn is part of the .NET Framework, so that it provides access to all of the features of that framework. For example, you can create ASP.NET Web applications using any .NET programming language (Visual Basic, C#, Managed Extensions for C++, and many others) and .NET debugging facilities. You access data using ADO.NET. Similarly, you can access operating system services using .NET Framework classes, and so on.

ASP.NET Web applications run on a Web server configured with Microsoft Internet Information Services (IIS). However, you do not need to work directly with IIS. You can program IIS facilities using ASP.NET classes, and Visual Studio handles file management tasks such as creating IIS applications when needed and providing ways for you to deploy your Web applications to IIS.

Programming Web Applications with Web Forms

When designing applications that involve a user interface, you have two choices: Windows Forms and Web Forms. Both have full design-time support within the development environment and can provide a rich user interface and advanced application functionality to solve business problems. How, then, to decide which technology is appropriate for a given application?

Certain application considerations might make the choice obvious: For example, if you are creating an e-commerce Web site that will be accessible to the public over the Internet, then of course you would develop the application using Web Forms pages. If you are building a processing-intensive, highly responsive application that needs to take advantage of the full functionality of the client machine — such as an office productivity application — of course you would use Windows Forms. However, in other cases the choice might not be so clear.

In the sections that follow, you can find information about the features and characteristics of each type of technology that will help you determine which is most suited to your application.

Windows Forms

Windows Forms are used to develop applications where the client is expected to shoulder a significant amount of the processing burden in an application. These include classic Win32 desktop applications, the kinds that were traditionally developed in previous versions of Visual Basic and Visual C++. Examples include drawing or graphics applications, data-entry systems, point-of-sale systems, and games.

A common feature these applications share is that they rely on the power of the desktop computer for processing and high-performance content display. Some Windows Forms applications might be entirely self-contained and perform all application processing on the user's computer. Games are often written this way. Others might be part of a larger system and use the desktop computer primarily for processing user input. For example, a point-of-sale system often requires a responsive, sophisticated user interface that is created on the desktop computer, but is linked to other components that perform back-end processing.

Because a Windows application that uses Windows Forms is built around a Windows framework, it has access to system resources on the client computer, including local files, the Windows registry, the printer, and so on. This level of access can be restricted to eliminate any security risks or potential problems that arise from unwanted access. Additionally, Windows Forms can take advantage of the .NET GDI+ graphics classes to create a graphically rich interface, which is often a requirement for data-mining or game applications.

Web Forms

ASP.NET Web Forms are used to create applications in which the primary user interface is a browser. Naturally, this includes applications intended to be available publicly via the World Wide Web, such as e-commerce applications. But Web Forms are useful for more than just creating Web sites — many other applications lend themselves to a "thin front end" as well, such as an intranet-based employee handbook or benefits application. An important feature is that there is no distribution cost, since users already have installed the only piece of the application that they need— the browser.

Web Forms applications are, by definition, platform-independent — that is, they are "reach" applications. Users can interact with your application regardless of what type of browser they have and even what type of computer they are using. At the same time, Web Forms applications can be optimized to take advantage of features built into the most recent browsers such as Microsoft Internet Explorer 5 to enhance performance and responsiveness. (In many cases, this optimization is built into the Web Forms components you are using, which can automatically detect browser levels and render pages accordingly.)

Web Forms offer some features that are useful even in non-Web contexts. Because they rely on HTML, they are suitable for text-intensive applications of any sort, and especially those in which text formatting is important. Because browsers are usually limited in their access to a user's system resources, they are useful precisely in situations where you want to limit users' access to portions of your application.

Comparing Windows Forms and Web Forms

The following table provides a side-by-side comparison of different application criteria and how Windows Forms and Web Forms technologies address these criteria.

Feature/Criterion	Windows Forms	Web Forms
Deployment	Windows Forms allow "no-touch" deployment, where applications can be downloaded, installed, and run directly on the users' machines without any alteration of the registry.	Web Forms have no client deployment; the client requires only a browser. The server must be running the Microsoft .NET Framework. Updates to the application are made by updating code on the server.
Graphics	Windows Forms include GDI+, which allows sophisticated graphics to be used for games and other extremely rich graphical environments.	Interactive or dynamic graphics require round trips to the server for updates when used on Web Forms. GDI+ can be used on the server to create custom graphics.
Responsiveness	Windows Forms can run entirely on the client computer; they can provide the quickest response speed for applications requiring a high degree of interactivity.	If you know that users will have Internet Explorer 5 or later, a Web Forms application can take advantage of the browser's dynamic HTML (DHTML) capabilities to create a rich, responsive user interface (UI). If users have other browsers, most processing (including UI-related tasks such as validation) requires a round trip to the Web server, which can affect responsiveness.
Form and text flow control	Windows Forms grid positioning gives you precise two-dimensional control (x and y coordinates) over the placement of controls. To display text on Windows Forms requires that you insert it into controls (for example, the **Label** control, the **Textbox** control, or the **RichTextBox** control). Formatting is limited.	Web Forms are based around HTML-style flow layout and therefore support all the features of Web page layout. They are particularly rich in text formatting support. Control layout can be managed adequately (with some limitations, such as no overlapping controls). If the users have DHTML-capable browsers, you can specify much more precise layout with two-dimensional (x- and y-coordinate) layout.
Platform	Windows Forms require the .NET Framework running on the client computer.	Web Forms require only a browser. DHTML-capable browsers can take advantage of extra features, but Web Forms can be designed to work with all browsers. The Web server must

		be running the .NET Framework.
Access to local resources (file system, Windows registry, and so forth)	Applications, when permitted, can have complete access to local computer resources. If required, the application can be restricted with precision from using specific resources.	Browser security prevents the application from accessing resources on the local computer.
Programming model	Windows Forms are based on a client-side, Win32 message-pump mode, where instances of components are created, used, and discarded by the developer.	Web Forms rely on a largely asynchronous, disconnected model, where components are loosely coupled to the application front end. Typically, application components are invoked via HTTP. This model may not be suitable for applications requiring extreme throughput from the user end or for those with high-volume transactions. Similarly, Web Forms applications may not be suitable for database applications that require high levels of concurrency control (for example, pessimistic locking).
Security	Windows Forms use granular permissions in its implementation of code access security to protect computer resources and sensitive information. This allows careful exposure of functionality, while retaining security. For instance the Printing Permission, which at one level would allow printing to the default printer only, at another level would allow printing to any printer.	Authorization to gain access to the resources of a web application is typically controlled on a per-URL basis by authenticating the credentials (for example, a name/password pair) of the requestor. Web Forms enable the ability to control the identity under which server application code is executed. Applications can execute code with the identity of the requesting entity, which is known as impersonation. Applications can also dynamically tailor content based on the requestor's identity or role. For example, a manager could receive access to a site, or a higher level of content than someone with lower permissions.

Event Handling in Windows Forms

Visual Studio .NET 2003

An event handler is a procedure in your code that determines the actions to be performed when an event occurs, such as the user clicking a button or a message queue receiving a message. When an event is raised, the event handler (or handlers) that receives the event is executed.

Events can be assigned to multiple handlers, and the methods that handle particular events can be changed dynamically. You can use the Windows Forms Designer to create event handlers.

Introduction to Events in Windows Forms
Visual Studio .NET 2003

An event is an action which you can respond to, or "handle," in code. Events can be generated by a user action, such as clicking the mouse or pressing a key; by program code; or by the system.

Event-driven applications execute code in response to an event. Each form and control exposes a predefined set of events that you can program against. If one of these events occurs and there is code in the associated event handler, that code is invoked.

The types of events raised by an object vary, but many types are common to most controls. For example, most objects will handle a **Click** event — if a user clicks a form, code in the form's **Click** event handler is executed.

Note Many events occur in conjunction with other events. For example, in the course of the **DoubleClick** event occurring, the **MouseDown**, **MouseUp**, and **Click** events occur.

Introduction to Event Handlers in Windows Forms
Visual Studio .NET 2003

An event handler is a method that is bound to an event. When the event is raised, the code within the event handler is executed. Each event handler provides two parameters that allow you to handle the event properly. The following example shows an event handler for a **Button** control's **Click** event:

```
' Visual Basic
Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles button1.Click

End Sub
// C#
private void button1_Click(object sender, System.EventArgs e)
{

}
// C++
private:
  System::Void button1_Click(System::Object *  sender,
    System::EventArgs *  e)
  {
```

}

The first parameter, sender, provides a reference to the object that raised the event. The second parameter, e in the example above, passes an object specific to the event that is being handled. By referencing the object's properties (and, sometimes, its methods), you can obtain information such as the location of the mouse for mouse events or data being transferred in drag-and-drop events.

Typically each event produces an event handler with a different event-object type for the second parameter. Some event handlers, such as those for the **MouseDown** and **MouseUp** events, have the same object type for their second parameter. For these types of events, you can use the same event handler to handle both events.

You can also use the same event handler to handle the same event for different controls. For example, if you have a group of **RadioButton** controls on a form, you could create a single event handler for the **Click** event and have each control's **Click** event bound to the single event handler.

Creating Event Handlers on the Windows Forms Designer
Visual Studio .NET 2003

In nearly all applications, it is essential to respond to user or system events. In C# and C++, you can create event handlers using the Properties window. In Visual Basic, you can use the Class Name and Method Name drop-down boxes in the Code Editor. For more information on creating default-named event handlers, see Creating Default Event Handlers on the Windows Forms Designer.

To create an event handler in Visual Basic

1. Right-click the form and choose **View Code**.
2. From the **Class Name** drop-down box, select the control that you want to add a specific event handler to.
3. From the **Method Name** drop-down box, select the event for which you want to add a specific handler.
4. The Code Editor inserts the appropriate event handler and positions the insertion point within the method. In the example below, it is the Click event for the Button control.
5. ' Visual Basic
6. Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
7.   ' Add event handler code here.
8. End Sub
9. Add the appropriate code to the event handler.

To create an event handler in C# or C++

1. Click the form or control that you want to create an event handler for.
2. In the Properties window, click the **Events** button (⚡).
3. In the list of available events, click the event that you want to create an event handler for.
4. In the box to the right of the event name, type the name of the handler and press ENTER.

   Tip Name the event handler according to the functionality of the event; for example, for the **Click** event, you can type **StartProcess** as the event handler.

   The Code Editor appears, showing the code for the form, and an event handler method is generated in your code similar to the following:

   ```
   // C#
   private void StartProcess(object sender, System.EventArgs e)
   {
     // Add event handler code here.
   }
   // C++
   private:
     System::Void StartProcess(System::Object *  sender,
       System::EventArgs *  e)
     {
       // Add event handler code here.
     }
   ```

5. Add the appropriate code to the event handler.

Creating Default Event Handlers on the Windows Forms Designer

Visual Studio .NET 2003

Within the Windows Forms Designer, double-clicking the design surface (either the form or a control) creates an event handler for the default action for that item.

To create the event handler for a default action

1. Double-click the item on the Windows Forms Designer for which you wish to create a handler.

   The **Code Editor** opens and the mouse pointer is located inside the newly created default event handler.

2. Add the appropriate code to the event handler.

   When you finish, your code might look like this:

```
' Visual Basic
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
Button1.Click
'Add Event Handlers Code Here
End Sub

// C#
private void button1_Click(object sender, System.EventArgs e)
{

}
// C++
private:
  System::Void button1_Click(System::Object *  sender,
    System::EventArgs *  e)
  {
  }
```

The preceding code is the default event handler for a Button control.

Creating Event Handlers at Run Time for Windows Forms

Visual Studio .NET 2003

In addition to creating events using the Windows Forms Designer, you can also create an event handler at run time. This action allows you to connect event handlers based on conditions in code at run time as opposed to having them connected when the program initially starts.

To create an event handler at run time

1. Open the form in the Code Editor that you want to add an event handler to.
2. Add a method to your form with the method signature for the event that you want to handle.

   For example, if you were handling the **Click** event of a **Button** control, you would create a method such as the following:

```
' Visual Basic
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   ' Add event handler code here.
End Sub

// C#
private void button1_Click(object sender, System.EventArgs e)
{
// Add event handler code here.
}

// C++
private:
  System::Void button1_Click(System::Object *  sender,
    System::EventArgs *  e)
  {
    // Add event handler code here.
  }
```

3. Add code to the event handler as appropriate to your application.
4. Determine which form or control you want to create an event handler for.
5. In a method within your form's class, add code that specifies the event handler to handle the event. For example, the following code specifies the event handler button1_Click handles the **Click** event of a **Button** control:
6. ' Visual Basic
7. AddHandler Button1.Click, AddressOf Button1_Click
8.
9. // C#
10. button1.Click += new EventHandler(button1_Click);
11.
12. // C++
13. button1->add_Click(new System::EventHandler(this, button1_Click));

    The **AddHandler** method demonstrated in the Visual Basic code above establishes a click event handler for the button.

Connecting Multiple Events to a Single Event Handler in Windows Forms

Visual Studio .NET 2003

In your application design, you may find it necessary to have a single event handler used for multiple events or the multiple events fire the same procedure. For example, it is often a powerful time-saver to have a menu command fire the same event as a button on your form does if they expose the same functionality. You can do this by using the Events view of the Properties window in C# or using the Handles keyword and the **Class Name** and **Method Name** drop-down boxes in the Visual Basic Code Editor.

To connect multiple events to a single event handler in Visual Basic

1. Right-click the form and choose **View Code**.
2. From the **Class Name** drop-down box, select one of the controls that you want to have the event handler handle.
3. From the **Method Name** drop-down box, select one of the events that you want the event handler to handle.

4. The Code Editor inserts the appropriate event handler and positions the insertion point within the method. In the example below, it is the Click event for the Button control.
5. ' Visual Basic
6. Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
7. 'Add Event Handler Code Here
8. End Sub
9. Append the other events you would like handled to the Handles clause.
10. ' Visual Basic
11. Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click, Button2.Click
12. 'Add Event Handler Code Here
13. End Sub
14. Add the appropriate code to the event handler.

To connect multiple events to a single event handler in C#

1. Select the control to which you want to connect an event handler.
2. In the **Properties** window, click the **Events** button (⚡).
3. Click the name of the event that you want to handle.
4. In the value section next to the event name, click the drop-down button to display a list of existing event handlers that match the method signature of the event you want to handle.
5. Select the appropriate event handler from the list.

   Code will be added to the form to bind the event to the existing event handler.

Creating Event Handlers in the Visual Basic Code Editor
Visual Studio .NET 2003

The Visual Basic Code Editor provides an easy way to create event handlers for your Windows Forms. Although it does not allow you to connect multiple events to a single event handler (see Connecting Multiple Events to a Single Event Handler in Windows Forms), it does provide a quick and easy way to create event handlers while working on your form's code without having to display the form in the Windows Forms Designer.

To create an event handler using the Visual Basic Code Editor

1. In the **Code Editor**, choose the form or control that you want to create an event handler for from the **Class Name** drop-down list.
2. From the **Method Name** list at the top of the Code Editor, choose the event that you want to create an event handler for.

   Visual Basic creates an event handler and adds it to your form's class.

Mouse and Keyboard Events in Windows Forms

Visual Studio .NET 2003

There are a number of events related to the user's employment of the mouse and keyboard. Each of these events has an event handler for which you can write code in your Windows applications. These events include MouseDown, MouseUp, MouseMove, MouseEnter, MouseLeave, MouseHover, KeyPress, KeyDown, and KeyUp. The **MouseDown**, **MouseUp**, and **MouseMove** event handlers receive an argument of MouseEventArgs containing data related to their events. The **MouseEnter**, **MouseLeave**, and **MouseHover** event handlers receive an argument of type EventArgs containing data related to their events. The key-related event handlers receive an argument of type KeyEventArgs containing data related to their events. These events are handled just like any other events that occur on Windows Forms. For more information about handling events, see Introduction to Event Handlers In Windows Forms.

Additionally, be aware that you can change the mouse cursor while operations are occurring. This capability can be paired with the **MouseEnter** and **MouseLeave** events to both provide feedback to the user that computations are occurring as well as limit user interaction for specified amounts of time. For example, if you show a dialog box to indicate files are being copied, it is customary to change the cursor to the hourglass.

Sometimes, the mouse cursor will change because of system events, such as when your application is involved in a drag-and-drop operation. For more information about drag-and-drop procedures, see Drag-and-Drop Operations and Clipboard Support. You can change the mouse cursor by setting the Cursor property of a control. For more information about working with cursors, see Cursors Class.

Order of Events in Windows Forms

Visual Studio .NET 2003

The order in which events are raised in Windows applications is of particular interest to developers concerned with handling each of these events in turn. When a situation calls for meticulous handling of events, such as when you are redrawing parts of the form, an awareness of the precise order in which events are raised at run time is necessary. For an overview of events in Windows Forms, see Introduction to Events in Windows Forms. For details about the makeup of event handlers, see Introduction to Event Handlers in Windows Forms.

Another aspect of Windows Forms architecture is covered in Anatomy of the Visual Basic Code Behind Windows Forms.

Standard Click Event Firing Behavior

All Windows Forms controls raise events in the same order when a mouse button is pressed and released (regardless of which mouse button), except where noted for individual controls. The following is the order of events raised for a single mouse-button click:

1. MouseDown event.
2. Click event.
3. MouseUp event.

The following is the order of events raised for a double mouse-button click:

1. **MouseDown** event.
2. **Click** event.
3. DoubleClick event. (This can vary, depending whether the control in question has its **StandardDoubleClick ControlStyles** bit set to **true**. For more information on how to set a **ControlStyles** bit, see the Control.SetStyle Method.)
4. **MouseUp** event.

Individual Controls

The following controls **do not** conform to the standard **Click** event behavior:

- **Button** control
  - Left click: **Click**
  - Right click: No **Click** events raised
  - Left double-click: **Click**, **Click**
  - Right double-click: No **Click** events raised
- **CheckBox** control
  - Left click: **Click**
  - Right click: No **Click** events raised
  - Left double-click: **Click**, **Click**
  - Right double-click: No **Click** events raised
- **RadioButton** control
  - Left click: **Click**
  - Right click: No **Click** events raised
  - Left double-click: **Click**, **Click**
  - Right double-click: No **Click** events raised
- **TextBox** control
  - Left click: **Click**
  - Right click: No **Click** events raised
  - Left double-click: **Click**, **DoubleClick**
  - Right double-click: No **Click** events raised
- **ListBox** control

  Note The event behavior detailed below happens whether the user clicks on an item or not (that is, a mouse click or double-click anywhere within the **ListBox** control raises these events).

  - Left click: **Click**
  - Right click: No **Click** events raised
  - Left double-click: **Click**, **DoubleClick**

    - Right double-click: No **Click** events raised
- **CheckedListBox** control

  Note The event behavior detailed below happens whether the user clicks on an item or not (that is, a mouse click or double click anywhere within the **ListBox** control raises these events)

    - Left click: **Click**
    - Right click: No **Click** events raised
    - Left double-click: **Click**, **DoubleClick**
    - Right double-click: No **Click** events raised
- **ComboBox** control

  Note The event behavior detailed below happens whether the user clicks on the edit field, the button, or on an item within the list.

    - Left click: **Click**
    - Right click: No **Click** events raised
    - Left double-click: **Click**, **Click**
    - Right double-click: No **Click** events raised
- **RichTextBox** control
    - Left click: No **Click** events raised
    - Right click: No **Click** events raised
    - Left double-click: No **Click** events raised
    - Right double-click: No **Click** events raised
- **ListView** control

  Note The event behavior detailed below happens only when the user clicks on the items in the **ListView** control. No events are raised for clicks anywhere else on the control. In addition to those described below, there are the BeforeLabelEdit and AfterLabelEdit events, which may be of interest to developers wishing to use validation with the **ListView** control.

    - Left click: **Click**
    - Right click: **Click**
    - Left double-click: **Click**, **DoubleClick**
    - Right double-click: **Click**, **DoubleClick**
- **TreeView** control

  Note The event behavior detailed below happens only when the user clicks on the items themselves or to the right of the items in the **TreeView** control. No events are raised for clicks anywhere else on the control. In addition to those described below, there are the BeforeCheck, BeforeSelect, BeforeLabelEdit, AfterSelect, AfterCheck, and

AfterLabelEdit events, which may be of interest to developers wishing to use validation with the **TreeView** control.

- Left click: **Click**
- Right click: **Click**
- Left double-click: **Click**, **DoubleClick**
- Right double-click: **Click**, **DoubleClick**

For details on the data sent by the system when these events are raised, see EventArgs Class. For details on monitoring user input to the keyboard for modifier keys (such as the SHIFT, ALT, and CTRL keys), see Determining Which Modifier Key Was Pressed.

Control Painting Behavior

Toggle controls, such as the controls deriving from the ButtonBase class, have a distinctive painting behavior. It is as follows:

1. The user presses the mouse button.
2. The control paints in the pressed state.
3. The **MouseDown** event is raised.
4. The user releases the mouse button.
5. The control paints in the raised state.
6. The **Click** event is raised.
7. The **MouseUp** event is raised.

Note If the user moves the pointer out of the toggle control while the mouse button is down (such as moving the mouse off the **Button** control while it is pressed), the toggle control will paint in the raised state.

Determining Which Modifier Key Was Pressed

Visual Studio .NET 2003

When creating an application that accepts the user's keystrokes, you may also want to monitor for modifier keys such as the SHIFT, ALT, and CTRL keys. When a modifier key is pressed in combination with other keys, or with mouse clicks, your application can respond appropriately — the letter S may simply cause an "S" to appear on the screen, but if CTRL+S is pressed, the current document may be saved.

To determine which modifier key was pressed

- Use the bitwise AND operator (**And** in Visual Basic, **&** in C# and C++) with the **ModifierKeys** property and a value of the Keys enumeration to determine which modifier key was pressed. (**ModifierKeys** is a shared member of the **Control** class; for more information on shared members, see Shared Members.)
- ' Visual Basic

```
•  Private Sub Button1_KeyPress(ByVal sender As Object, ByVal e As _
•  System.Windows.Forms.KeyPressEventArgs) Handles Button1.KeyPress
•    If (Control.ModifierKeys And Keys.Shift) = Keys.Shift Then
•      MessageBox.Show("Pressed " & Keys.Shift)
•    End If
•  End Sub
•
•  // C#
•  private void button1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
•  {
•    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
•    {
•      MessageBox.Show("Pressed " + Keys.Shift);
•    }
•  }
•
•  // C++
•  private:
•    System::Void button1_KeyPress(System::Object *  sender,
•      System::Windows::Forms::KeyPressEventArgs *  e)
•    {
•      if ((Control::ModifierKeys & Keys::Shift) == Keys::Shift)
•      {
•        MessageBox::Show(String::Concat(S"Pressed ",
•          __box(Keys::Shift)->ToString()));
•      }
•    }
```

Note For C# or C++, be sure that the necessary code to enable the event handler is present. In this case, it would be similar to the following:

```
// C#
this.button1.Click += new System.EventHandler(this.button1_Click);

// C++
this->button1->add_KeyPress
  (new System::Windows::Forms::KeyPressEventHandler
  (this, button1_KeyPress));
```

Overriding Base Methods Using the Visual Basic Code Editor
Visual Studio .NET 2003

When working with Visual Basic classes in Windows Forms, you may need to override a method defined in a base class. In addition to providing a way to create event handlers while writing code, the Visual Basic Code Editor also allows you to override methods contained in your base class.

To override a base class method using the Visual Basic Code Editor

1. Open the form that you want to use in the Code Editor.
2. From the **Class Name** drop-down list at the top of the Code Editor, choose **(Overrides)**.

   A list of the methods is added to the **Event** list.

3. From the **Method Name** list at the top of the Code Editor, choose the method for which you want to create an event handler.

   A method is added to your form's class and the **Overrides** keyword is added to the method signature. The following code demonstrates an example of an overridden form method:

   ```
   ' Visual Basic
   Protected Overrides Sub Finalize()
     'Implementation Code Added Here
   End Sub
   ```

Programming Web Services

Introduction to Web Service with Example in ASP.NET

Background

In this article we will learn about web service using the scenario when Our applications often require code to determine the number of days, such as how long the customer is associated with us, also to convert from a date of present days into days or years and so on.

In a normal application I need to write the Business logic repeatedly for the same requirements so due to the requirements you can write a single web service for Multiple applications that allow an access method on any platform used, so let us start with the basics.

Per-requirement to Understand this application

If you are a beginner and you need to understand what a web service is then you can read the article of the author Vidya Vrat Agarwal sir; the article is .NET Web Services.

I hope you read it if you are unfamiliar with web services.

What is web services ?

A "web service is the communication platform between two different or same platform applications that allows to use their web method."

In the preceding definition you observed that I used the two main points in the definition of web service; they are **different or same platform application** and the second is **web method**.
so let us learn some basic about it **.**

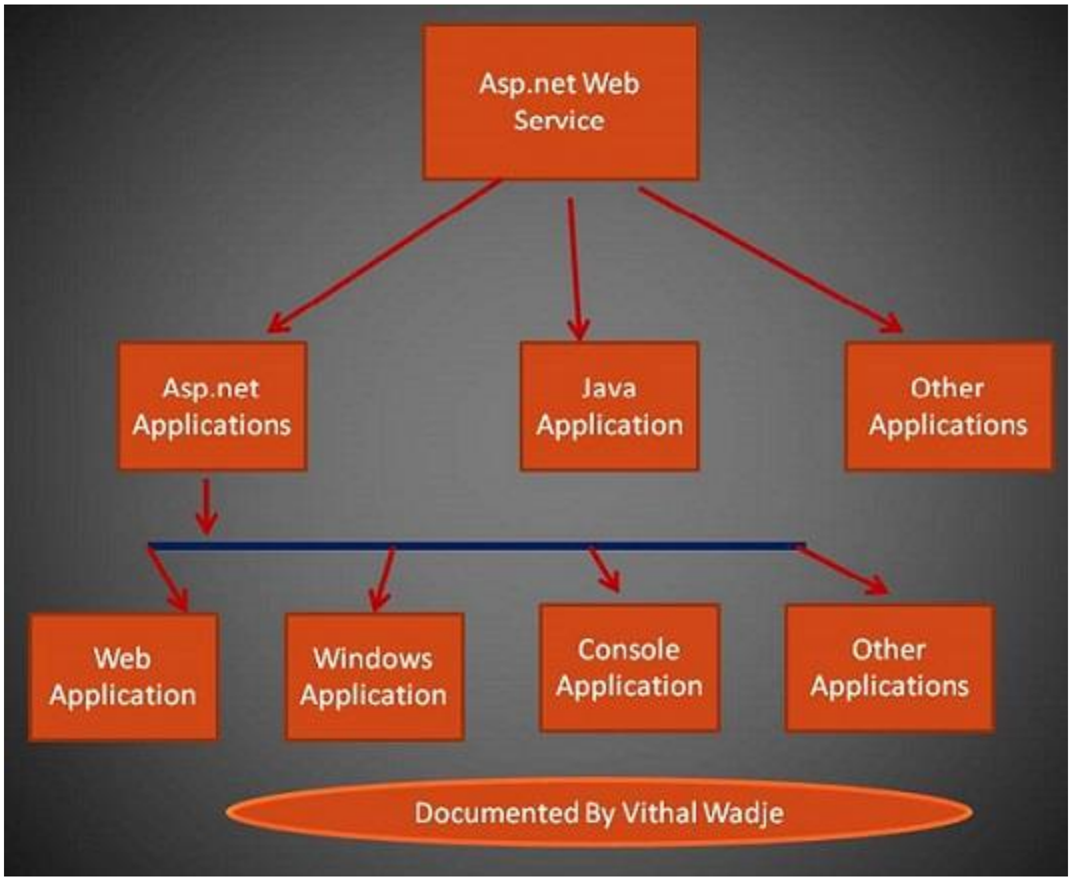
What does different or same application and platform mean

It means that I can create a web service in any language, such as Java or other languages and that the language web service can be used in a .Net based application and also a .Net web service or in another application to exchange the information.

What does *web method* mean

The method in web services always start with [webMethod] attributes, it means that it is a web method that is accessible anywhere, the same as my web application.

to understand more clear let us ,I represent all above given definitions explanation in following diagram



In the above diagram,I have shown,how the Asp.net Web Service is used in different types of applications means i am trying to explain that I can create a web service in any language, such as Java or other languages and that the language web service can be used in a .Net based application as wel as java or other applications and you can also use .Net based web application in Java applications means web Services dont have a any platform restrictions.

I hope you understand the basics of a web service.

So let us start to create the web service.

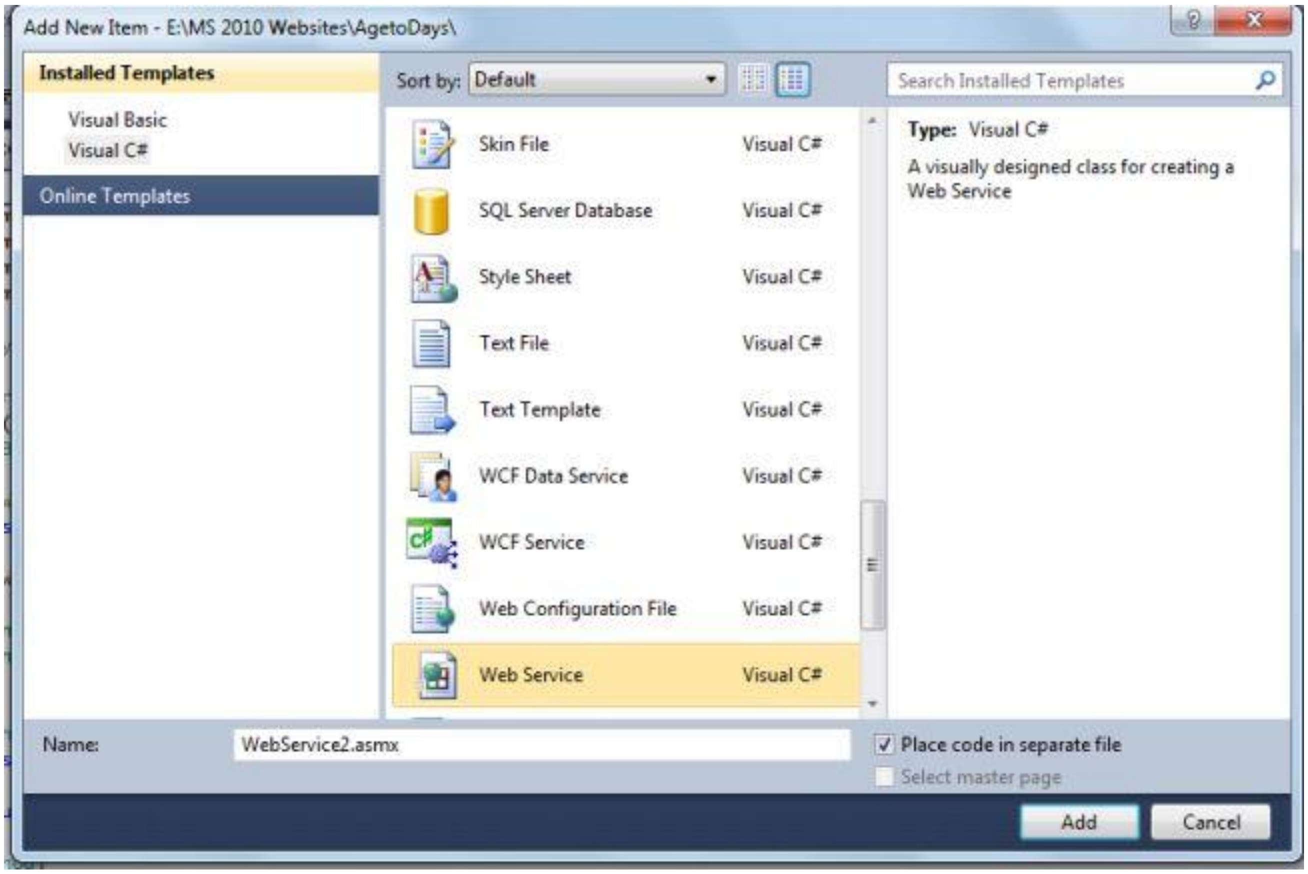
Note

If you closely observe that ,there is no separate web service template in .Framework 2010 as you see in 2008 while adding a project or web site it might be because of WCF.

So let us start using a different way to add a web service using a template

1. "Start" - "All Programs" - "Microsoft Visual Studio 2010"
2. "File" - "New Project" - "C#" - "Empty Web Application" (to avoid adding a master page)
3. Provide the web site a name such as "agetodays" or another as you wish and specify the location

4. Then right-click on Solution Explorer - "Add New Item" - you see the web service templates



Type: Visual C#

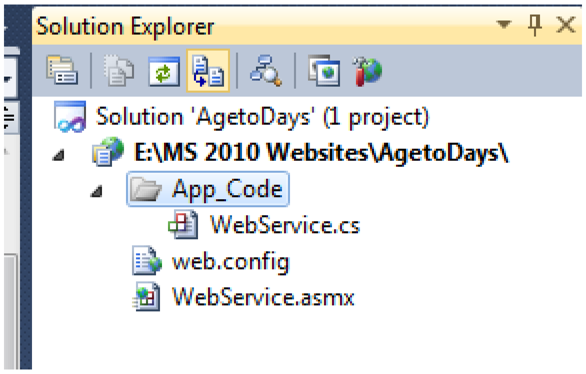
A visually designed class for creating a Web Service

Name: WebService2.asmx

[✓] Place code in separate file

[ ] Select master page

Select Web Service Template and click on add button. then after that the Solution Explorer look like as follows.



```
Solution Explorer
Solution 'AgetoDays' (1 project)
  E:\MS 2010 Websites\AgetoDays\
    App_Code
      WebService.cs
    web.config
    WebService.asmx
```

Then open the Webservice.cs class and write the following method followed by [webMethod] attribute as in.

```
[WebMethod]
public int converttodaysweb(int day, int month, int year)
{
   DateTime dt = new DateTime(year, month, day);
   int datetodays = DateTime.Now.Subtract(dt).Days;
   return datetodays;
}
```

In the code above I have declared a one integer method named converttodaysweb with the three parameters day, month and year for accepting day, month and year from the user.

Then after that I created an object of date time and passed the those variables that I get from the users. I declared another variable in the method that is age today to store the number of days remaining from the user's input date to the current date and finally I return that variable..

The webservice.cs file will then look as in the following

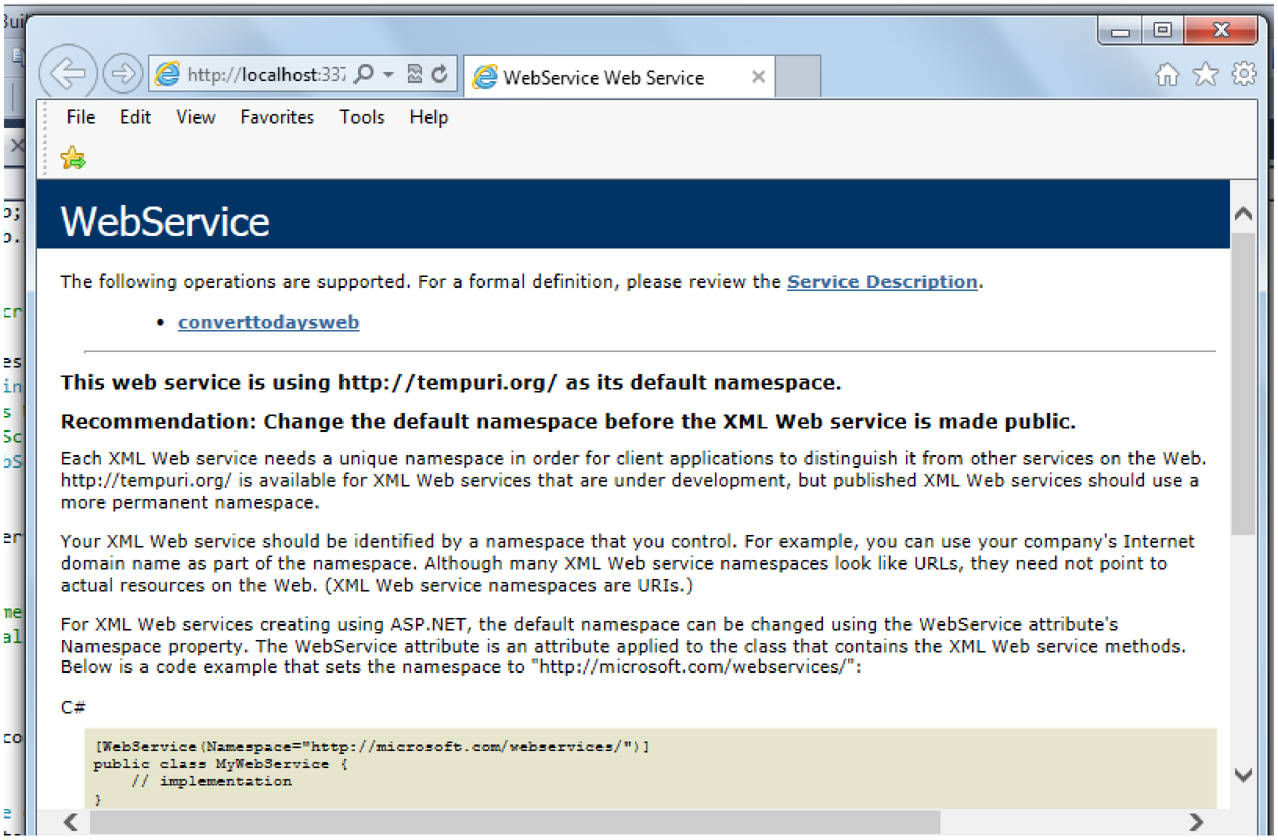
```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.Services;
///<summary>
/// Summary description for UtilityWebService
///</summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class WebService: System.Web.Services.WebService
{
   public WebService()
   {
      //Uncomment the following line if using designed components
      //InitializeComponent();
   }
   [WebMethod]
   public int converttodaysweb(int day, int month, int year)
   {
      DateTime dt = new DateTime(year, month, day);
      int datetodays = DateTime.Now.Subtract(dt).Days;
      return datetodays;

   }

```

28. }

Now run the application that look like as follows.



WebService

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

- converttodaysweb

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

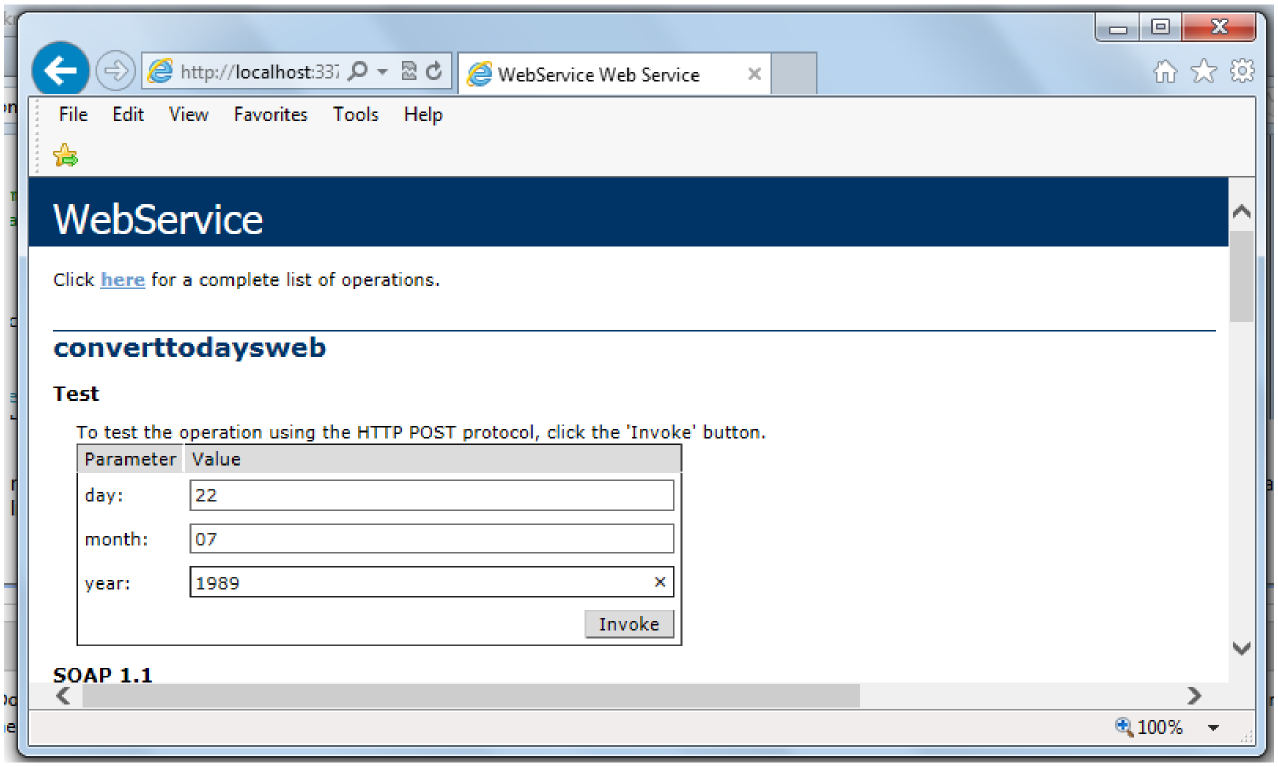
Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Now in the above we see our method that we are created in the webservice.cs file, so click on that method and provide input values and click on the "invoke" link as in.



WebService

Click here for a complete list of operations.

converttodaysweb

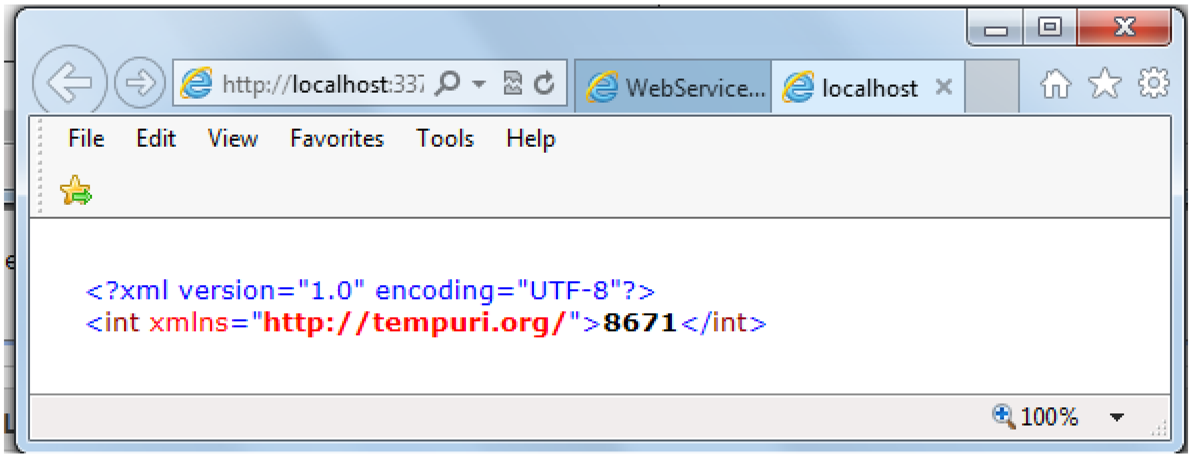
Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
day:	22
month:	07
year:	1989

SOAP 1.1

The output will be as follows



```
<?xml version="1.0" encoding="UTF-8"?>
<int xmlns="http://tempuri.org/">8671</int>
```

In the screen above you see that the output is 8671, that is the days from the input date.

UNIT –IV

TWO MARKS

1. What are the uses of server side controls? .NET wants you to use server side controls that can often generate unnecessary trips back to the server by default. Plus, .NET also want to make the determination as to how will acts with various browsers. While not impossible, it is make it much more cumbersome to create custom workaround for special situations. Not to mention the facts that new browsers are released in between .NET services pack updates.

2. List out the server side state management options supported by ASP.NET.
Application states
Session state
Profile properties
Database support

3. Differentiate the postback events with nonpostback events?

Post back events are those that cause the form to be posted back to the server immediately. In contrast, many event are considered nonpostback in that the form isn't posted back it the server immediately. Instead these events are cached by the control until the next time a post back event occurs.

4. What is WSDL?

A lot of work is being done for you automatically. HTML pages describing your web services and its methods are generated and this page includes links to pages in which the methods can be tested.

5. List down the events in life cycle of a web page?

Initialize, load view state, process postback data, load, handle postback events, prerender, save state, render, dispose

6. Define the prerender event in life cycle of a web page?

This is your last chance to modify the output the output prior to rendering using the OnPreRender () method.

7. Define Client-side support?

You make use of a web service by writing client code that acts as though it were communicating directly with a local object, but in reality communicates with host server through a proxy.

8. Define server-side support?

While creating a web service requires no special programming on your part, you need only write the implementing code, add the [web method] attribute and let the server do rest.

9. What is SOAP?

SOAP is a lightweight, message-based protocol built on XML, HTTP and SMTP. Two other protocols are desirable, but not required. For a client to a use a SOAP-enabled web service.

10. Define web form life cycle?

The life cycle begins with request for the page, which cause the server to load it. When the request is complete, the page is unloading.

UNIT -V

THE CLR AND THE .NET FRAMEWORK

Assemblies

The programs presented so far have stood on their own except for dependence on a few system-provided classes such as System. Console. It is far more common, however, for real-world applications to consist of several different pieces, each compiled separately. For example, a corporate application might depend on several different components, including some developed internally and some purchased from independent software vendors.

Namespaces and assemblies enable this component-based system. Namespaces provide a logical organizational system. Namespaces are used both as an "internal" organization system for a program, and as an "external" organization system — a way of presenting program elements that are exposed to other programs.

Assemblies are used for physical packaging and deployment. An assembly may contain types, the executable code used to implement these types, and references to other assemblies.

There are two main kinds of assemblies: applications and libraries. Applications have a main entry point and usually have a file extension of .exe; libraries do not have a main entry point, and usually have a file extension of .dll.

To demonstrate the use of namespaces and assemblies, this section revisits the "hello, world" program presented earlier, and splits it into two pieces: a class library that provides messages and a console application that displays them.

The class library will contain a single class named HelloMessage. The example

```
// HelloLibrary.cs
namespace Microsoft.CSharp.Introduction
{
  public class HelloMessage
  {
    public string Message {
      get {
        return "hello, world";
      }
    }
  }
}
```

shows the HelloMessage class in a namespace named Microsoft.CSharp.Introduction. The HelloMessage class provides a read-only property named Message. Namespaces can nest, and the declaration

```
namespace Microsoft.CSharp.Introduction
{...}
```

is shorthand for several levels of namespace nesting:

```
namespace Microsoft
{
  namespace CSharp
  {
    namespace Introduction
    {...}
  }
}
```

The next step in the componentization of "hello, world" is to write a console application that uses the HelloMessage class. The fully qualified name for the class — Microsoft.CSharp.Introduction.HelloMessage — could be used, but this name is quite long and unwieldy. An easier way is to use a "using namespace directive," which makes it possible to use all of the types in a namespace without qualification. The example

```
// HelloApp.cs
using Microsoft.CSharp.Introduction;
class HelloApp
{
  static void Main() {
    HelloMessage m = new HelloMessage ();
    System.Console.WriteLine(m.Message);
  }
}
```

shows a using namespace directive that refers to the Microsoft.CSharp.Introduction namespace. The occurrences of HelloMessage are shorthand for Microsoft.CSharp.Introduction.HelloMessage.

C# also enables the definition and use of aliases. A using alias directive defines an alias for a type. Such aliases can be useful in situation in which name collisions occur between two class libraries, or when a small number of types from a much larger namespace are being used. The example

```
using MessageSource = Microsoft.CSharp.Introduction.HelloMessage;
```

shows a using alias directive that defines MessageSource as an alias for the HelloMessage class.

The code we have written can be compiled into a class library containing the class HelloMessage and an application containing the class HelloApp. The details of this compilation step might differ based on the compiler or tool being used. Using the command-line compiler provided in Visual Studio .NET, the command-line invocations are

```
csc /target:library HelloLibrary.cs
csc /reference:HelloLibrary.dll HelloApp.cs
```

which produce a class library named HelloLibrary.dll and an application named HelloApp.exe.

Versioning

Versioning is the process of evolving a component over time in a compatible manner. A new version of a component is source compatible with a previous version if code that depends on the previous version can, when recompiled, work with the new version. In contrast, a new version of a component is binary compatible if an application that depended on the old version can, without recompilation, work with the new version.

Most languages do not support binary compatibility at all, and many do little to facilitate source compatibility. In fact, some languages contain flaws that make it impossible, in general, to evolve a class over time without breaking at least some client code.

As an example, consider the situation of a base class author who ships a class named Base. In the first version, Base contains no F method. A component named Derived derives from Base, and introduces an F. This Derived class, along with the class Base on which it depends, is released to customers, who deploy to numerous clients and servers.

```
// Author A
namespace A
{
  public class Base      // version 1
  {
  }
}
// Author B
namespace B
{
  class Derived: A.Base
  {
    public virtual void F() {
      System.Console.WriteLine("Derived.F");
    }
  }
}
```

From this point, the versioning trouble begins. The author of Base produces a new version, giving it its own F method.

```
// Author A
namespace A
{
  public class Base      // version 2
  {
    public virtual void F() {      // added in version 2
      System.Console.WriteLine("Base.F");
    }
  }
}
```

This new version of Base should be both source and binary compatible with the initial version. (If it weren't possible to simply add a method then a base class could never evolve.) Unfortunately, the new F in Base makes the meaning of Derived's F unclear. Did Derived mean to override Base's F? This seems unlikely, since when Derived was compiled, Base did not even have an F! Further, if Derived's F does override Base's F, then it must adhere to the contract specified by Base — a contract that was unspecified when Derived was written! In some cases, this is impossible. For example, Base's F might require that overrides of it always call the base. Derived's F could not possibly adhere to such a contract.

C# addresses this versioning problem by requiring developers to state their intent clearly . In the original code example, the code was clear, since Base did not even have an F. Clearly, Derived's F is intended as a new method rather than an override of a base method, since no base method named F exists.

If Base adds an F and ships a new version, then the intent of a binary version of Derived is still clear — Derived's F is semantically unrelated, and should not be treated as an override.

However, when Derived is recompiled, the meaning is unclear — the author of Derived may intend its F to override Base's F, or to hide it. Since the intent is unclear, the compiler produces a warning, and by default makes Derived's F hide Base's F. This course of action duplicates the semantics for the case in which Derived is not recompiled. The warning that is generated alerts Derived's author to the presence of the F method in Base.

If Derived's F is semantically unrelated to Base's F, then Derived's author can express this intent — and, in effect, turn off the warning–by using the new keyword in the declaration of F.

```
// Author A
namespace A
{
  public class Base        // version 2
  {
    public virtual void F() { // added in version 2
      System.Console.WriteLine("Base.F");
    }
  }
}
// Author B
namespace B
{
  class Derived: A.Base   // version 2a: new
  {
    new public virtual void F() {
      System.Console.WriteLine("Derived.F");
    }
  }
}
```

On the other hand, Derived's author might investigate further, and decide that Derived's F should override Base's F. This intent can be specified by using the override keyword, as shown below.

```
// Author A
namespace A
{
  public class Base             // version 2
  {
        public virtual void F() { // added in version 2
            System.Console.WriteLine("Base.F");
        }
    }
}
// Author B
namespace B
{
    class Derived: A.Base   // version 2b: override
    {
        public override void F() {
            base.F();
            System.Console.WriteLine("Derived.F");
        }
    }
}
```

The author of Derived has one other option, and that is to change the name of F, thus completely avoiding the name collision. Although this change would break source and binary compatibility for Derived, the importance of this compatibility varies depending on the scenario. If Derived is not exposed to other programs, then changing the name of F is likely a good idea, as it would improve the readability of the program — there would no longer be any confusion about the meaning of F.

Attributes

C# is an imperative language, but like all imperative languages it does have some declarative elements. For example, the accessibility of a method in a class is specified by declaring it public, protected, internal, protected internal, or private. C# generalizes this capability, so that programmers can invent new kinds of declarative information, attach this declarative information to various program entities, and retrieve this declarative information at run-time. Programs specify this additional declarative information by defining and using attributes.

For instance, a framework might define a HelpAttribute attribute that can be placed on program elements such as classes and methods, enabling developers to provide a mapping from program elements to documentation for them. The example

```
using System;
[AttributeUsage(AttributeTargets.All)]
public class HelpAttribute: Attribute
{
    public HelpAttribute(string url) {
        this.url = url;
    }
```

```
  public string Topic = null;
  private string url;
  public string Url {
    get { return url; }
  }
}
```

defines an attribute class named HelpAttribute that has one positional parameter (string url) and one named parameter (string Topic). As explained this attribute may be referenced by its full name, HelpAttribute, or by its implicit short name, Help. Positional parameters are defined by the formal parameters for public instance constructors of the attribute class, and named parameters are defined by public non-static read-write fields and properties of the attribute class.

The example

```
[Help("http://www.microsoft.com/.../Class1.htm")]
public class Class1
{
  [Help("http://www.microsoft.com/.../Class1.htm", Topic = "F")]
  public void F() {}
}
```

shows several uses of the Help attribute.

Attribute information for a given program element can be retrieved at run time by using reflection support. The example

```
using System;
class Test
{
  static void Main() {
    Type type = typeof(Class1);
    object[] arr = type.GetCustomAttributes(typeof(HelpAttribute), true);
    if (arr.Length == 0)
      Console.WriteLine("Class1 has no Help attribute.");
    else {
      HelpAttribute ha = (HelpAttribute) arr[0];
      Console.WriteLine("Url = {0}, Topic = {1}", ha.Url, ha.Topic);
    }
  }
}
```

checks to see if Class1 has a Help attribute, and writes out the associated Topic and Url values if the attribute is present.

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.

Applications of Reflection

Reflection has the following applications:

- 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

The **MemberInfo** object of the **System.Reflection** class needs to be initialized for discovering the attributes associated with a class. To do this, you define an object of the target class, as:

```
System.Reflection.MemberInfo info = typeof(MyClass);
```

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 class *MyClass*:

```
HelpAttribute
```

Type Discovery

- **This allows you to examine the types in an assembly.**

Assemblies are the core unit of deployment. At design time, we can examine the CIL code in a set of reference assemblies with a couple of external tools such as Reflector and ildasm to peek into underlying metadata, MSIL code and the manifest.

But many times it is necessary to discover types at runtime, and this is the situation where .NET Reflection can be used.

Reflecting on a Type

Reflection typically is the process of runtime type discovery to inspect metadata, CIL code, late binding and self-generating code. At run time by using reflection, we can access the same "type" information as displayed by the ildasm utility at design time. The reflection is analogous to reverse engineering in which we can break an existing *.exe or *.dll assembly to explore defined significant contents information, including methods, fields, events and properties.

You can dynamically discover the set of interfaces supported by a given type using the System.Reflection namespace. This namespace contains numerous related types as follows:

Types	Description
Assembly	This static class allows you to load, investigate and manipulate an assembly.
AssemblyName	Allows to exploration of abundant details behind an assembly.
EventInfo	Information about a given event.
PropertyInfo	Holds information of a specified property.
MethodInfo	Contains information about a specified method.

Reflection typically is used to dump out the loaded assemblies list, their reference to inspect methods, properties etcetera. Reflection is also used in the external disassembling tools such Reflector, Fxcop and NUnit because .NET tools don't need to parse the source code similar to C++.

Marshalling

Marshaling is the process of creating a bridge between managed code and unmanaged code; it is the homer that carries messages from the managed to the unmanaged environment and reverse. It is one of the core services offered by the CLR (Common Language Runtime.)

Because much of the types in unmanaged environment do not have counterparts in managed environment, you need to create conversion routines that convert the managed types into unmanaged and vice versa; and that is the marshaling process.

As a refresher, we call .NET code “managed” because it is controlled (managed) by the CLR. Other code that is not controlled by the CLR is called unmanaged.

Remoting

C# Remoting Architecture

The **.NET Remoting** provides an inter-process communication between Application Domains by using Remoting Framework. The applications can be located on the same computer , different computers on the same network, or on computers across separate networks. The .NET Remoting supports distributed object communications over the **TCP** and **HTTP** channels by using Binary or **SOAP** formatters of the data stream.

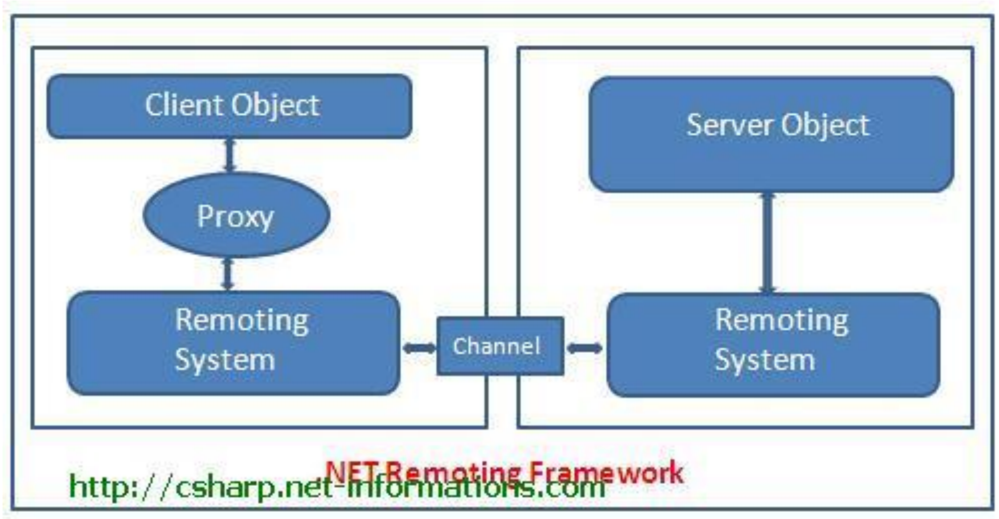
The main three components of a Remoting Framework are :

1. C# Remotable Object

2. C# Remote Listener Application - (listening requests for Remote Object)

3. C# Remote Client Application - (makes requests for Remote Object)

The Remote Object is implemented in a class that derives from ***System.MarshalByRefObject*** .



You can see the basic workflow of **.Net Remoting** from the above figure. When a client calls the Remote method, actually the client does not call the methods directly . It receives a proxy to the remote object and is used to invoke the method on the **Remote Object** . Once the proxy receives the method call from the Client , it encodes the message using appropriate formatter (**Binary Formatter** or **SOAP Formatter**) according to the Configuration file. After that it sends the call to the Server by using selected Channel (**TcpChannel** or **HttpChannel**). The Server side channel receives the request from the proxy and forwards it to the Server on Remoting system, which locates and invokes the methods on the Remote Object. When the execution of remote method is complete, any results from the call are returned back to the client in the same way.

Before an object instance of a Remotable type can be accessed, it must be created and initialized by a process known as Activation. **Activation** is categorized in two models , they are Client-activated Objects and Server-activated Objects.

C# Remote Activation

The real difference between client-activated and server-activated objects is that a server-activated object is not really created when a client instantiates it. Instead, it is created as needed.

By default the .NET Framework ships with two formatters(Binary Formatter or SOAP Formatter) and two channels(TcpChannel ,HttpChannel).

C# Remote Channels

C# Remote Formatters

Formatters and Channel are configured by using Configuration files. It can be easily Configured by using XML-based files.

C# Remote Configuration

The .NET Remoting is easy to use and powerful, largely because it is based on the Common Type System (CTS) and the Common Language Runtime (CLR).

Understanding Server Object Types

There are two types of server objects supported for remoting in .NET: well-known and client-activated. The communication with well-known objects is established each time a message is sent by the client. There is no permanent connection with a well-known object, as there is with client-activated objects.

Well-known objects come in two varieties: singleton and single-call. With a well-known singleton object, all messages for the object, from all clients, are dispatched to a single object running on the server. The object is created when the server is started and is there to provide service to any client that can reach it. Well-known objects must have a parameterless constructor.

With a well-known single-call object, each new message from a client is handled by a new object. This is highly advantageous on server farms, where a series of messages from a given client might be handled in turn by different machines depending on load balancing.

Client-activated objects are typically used by programmers who are creating dedicated servers, which provide services to a client they are also writing. In this scenario, the client and the server create a connection, and they maintain that connection until the needs of the client are fulfilled.

Specifying a Server with an Interface

The best way to understand remoting is to walk through an example. Here, build a simple four-function calculator class, like the one used in an earlier discussion on web services

Example 19-2. The Calculator interface

```
namespace Programming_CSharp
{
    using System;

    public interface ICalc
    {
        double Add(double x, double y);
        double Sub(double x, double y);
        double Mult(double x, double y);
        double Div(double x, double y);
    }
}
```

Save this in a file named ICalc.cs and compile it into a file named ICalc.dll. To create and compile the source file in Visual Studio, create a new project of type C# Class Library, enter the interface definition in the Edit window, and then select Build Build on the Visual Studio menu bar. Alternatively, if you have entered the source code using Notepad, you can compile the file at the command line by entering:

```
csc /t:library ICalc.cs
```

There are tremendous advantages to implementing a server through an interface. If you implement the calculator as a class, the client must link to that class in order to declare instances on the client. This greatly diminishes the advantages of remoting, because changes to the server require the class definition to be updated on the client. In other words, the client and server would be tightly coupled. Interfaces help decouple the two objects; in fact, you can later update that implementation on the server, and as long as the server still fulfills the contract implied by the interface, the client need not change at all.

Building a Server

To build the server used in this example, create CalcServer.cs in a new project of type C# Console Application (be sure to include a reference to ICalc.dll) and then compile it by selecting Build Build on the Visual Studio menu bar. Or, you can enter the code in Notepad, save it to a file named CalcServer.cs, and enter the following at the command-line prompt:

```
csc /t:exe /r:ICalc.dll CalcServer.cs
```

The `Calculator` class implements `ICalc`. It derives from `MarshalByRefObject` so that it will deliver a proxy of the calculator to the client application:

```
public class Calculator : MarshalByRefObject, ICalc
```

The implementation consists of little more than a constructor and simple methods to implement the four functions.

In this example, you'll put the logic for the server into the `Main( )` method of CalcServer.cs.

Your first task is to create a channel. Use HTTP as the transport because it is simple and you don't need a sustained TCP/IP connection. You can use the `HTTPChannel` type provided by .NET:

```
HTTPChannel chan = new HTTPChannel(65100);
```

Notice that you register the channel on TCP/IP port 65100

Next, register the channel with the CLR ChannelServices using the static method `RegisterChannel`:

```
ChannelServices.RegisterChannel(chan);
```

This step informs .NET that you will be providing HTTP services on port 65100, much as IIS does on port 80. Because you've registered an HTTP channel and not provided your own formatter, your method calls will use the SOAP formatter by default.

Now you are ready to ask the `RemotingConfiguration` class to register your well-known object. You must pass in the type of the object you want to register, along with an endpoint. An endpoint is a name that `RemotingConfiguration` will associate with your type. It completes the address. If the IP address identifies the machine and the port identifies the channel, the endpoint identifies the actual application that will be providing the service. To get the type of the object, you can call the static method `GetType( )` of the `Type` class, which returns a `Type` object. Pass in the full name of the object whose type you want:

```
Type calcType =
  Type.GetType("Programming_CSharp.Calculator");
```

Also pass in the enumerated type that indicates whether you are registering a `SingleCall` or `Singleton`:

```
RemotingConfiguration.RegisterWellKnownServiceType
   ( calcType, "theEndPoint",WellKnownObjectMode.Singleton );
```

The call to `RegisterWellKnownServiceType` does not put one byte on the wire. It simply uses reflection to build a proxy for your object.

Now you're ready to rock and roll. Example 19-3 provides the entire source code for the server.

Example 19-3. The Calculator server

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;

namespace Programming_CSharp
{
   // implement the calculator class
   public class Calculator : MarshalByRefObject, ICalc
   {
      public Calculator( )
      {
         Console.WriteLine("Calculator constructor");
      }

      // implement the four functions
      public double Add(double x, double y)
      {
         Console.WriteLine("Add {0} + {1}", x, y);
         return x+y;
      }
      public double Sub(double x, double y)
      {
         Console.WriteLine("Sub {0} - {1}", x, y);
         return x-y;
      }
      public double Mult(double x, double y)
      {
         Console.WriteLine("Mult {0} * {1}", x, y);
         return x*y;
      }
      public double Div(double x, double y)
      {
         Console.WriteLine("Div {0} / {1}", x, y);
         return x/y;
      }
   }

   public class ServerTest
   {
      public static void Main( )
      {
         // create a channel and register it
         HttpChannel chan = new HttpChannel(65100);
         ChannelServices.RegisterChannel(chan);

         Type calcType =
            Type.GetType("Programming_CSharp.Calculator");

         // register our well-known type and tell the server
         // to connect the type to the endpoint "theEndPoint"
         RemotingConfiguration.RegisterWellKnownServiceType
            ( calcType,
                "theEndPoint",
                 WellKnownObjectMode.Singleton );

          //  "They also serve who only stand and wait."); (Milton)
          Console.WriteLine("Press [enter] to exit...");
          Console.ReadLine( );
       }
   }
}
```

When you run this program, it prints its self-deprecating message:

```
Press [enter] to exit...
```

and then waits for a client to ask for service.

Building the Client

The client must also register a channel, but because you are not listening on that channel, you can use channel 0:

```
HTTPChannel chan = new HTTPChannel(0);
ChannelServices.RegisterChannel(chan);
```

The client now need only connect through the remoting services, passing a `Type` object representing the type of the object it needs (in our case, the `ICalc` interface) and the URI (Uniform Resource Identifier) of the implementing class:

```
MarshalByRefObject obj =
    RemotingServices.Connect
        (typeof(Programming_CSharp.ICalc),
        "http://localhost:65100/theEndPoint");
```

In this case, the server is assumed to be running on your local machine, so the URI is http://localhost, followed by the port for the server (`65100`), followed in turn by the endpoint you declared in the server (`theEndPoint`).

The remoting service should return an object representing the interface you've requested. You can then cast that object to the interface and begin using it. Because remoting cannot be guaranteed (the network might be down, the host machine may not be available, and so forth), you should wrap the usage in a `try` block:

```
try
{
    Programming_CSharp.ICalc calc =
        obj as Programming_CSharp.ICalc;

    double sum = calc.Add(3,4);
```

You now have a proxy of the calculator operating on the server, but usable on the client, across the process boundary and, if you like, across the machine boundary. Example 19-4 shows the entire client (to compile it, you must include a reference to ICalc.dll as you did with CalcServer.cs).

Example 19-4. The remoting Calculator client

```
namespace Programming_CSharp
{
   using System;
   using System.Runtime.Remoting;
   using System.Runtime.Remoting.Channels;
   using System.Runtime.Remoting.Channels.Http;

   public class CalcClient
   {
      public static void Main( )
      {

         int[] myIntArray = new int[3];

         Console.WriteLine("Watson, come here I need you...");

         // create an Http channel and register it
         // uses port 0 to indicate won't be listening
         HttpChannel chan = new HttpChannel(0);
         ChannelServices.RegisterChannel(chan);

         // get my object from across the http channel
         MarshalByRefObject obj =
          (MarshalByRefObject)  RemotingServices.Connect
            (typeof(Programming_CSharp.ICalc),
            "http://localhost:65100/theEndPoint");

         try
         {
            // cast the object to our interface
            Programming_CSharp.ICalc calc =
               obj as Programming_CSharp.ICalc;

            // use the interface to call methods
            double sum = calc.Add(3.0,4.0);
            double difference = calc.Sub(3,4);
            double product = calc.Mult(3,4);
            double quotient = calc.Div(3,4);

            // print the results
            Console.WriteLine("3+4 = {0}", sum);
            Console.WriteLine("3-4 = {0}", difference);
            Console.WriteLine("3*4 = {0}", product);
            Console.WriteLine("3/4 = {0}", quotient);
         }
         catch( System.Exception ex )
         {
            Console.WriteLine("Exception caught: ");
            Console.WriteLine(ex.Message);
            }
        }
    }
}

Output on client:
Watson, come here I need you...
3+4 = 7
3-4 = -1
3*4 = 12
3/4 = 0.75

Output on server:
Calculator constructor
Press [enter] to exit...
Add 3 + 4
Sub 3 - 4
Mult 3 * 4
Div 3 / 4
```

The server starts up and waits for the user to press Enter to signal that it can shut down. The client starts and displays a message to the console. The client then calls each of the four operations. You see the server printing its message as each method is called, and then the results are printed on the client.

It is as simple as that; you now have code running on the server and providing services to your client.

Using SingleCall

To see the difference that `SingleCall` makes versus `Singleton`, change one line in the server's `Main( )` method. Here's the existing code:

```
RemotingConviguration.RegisterWellKnownServiceType
    ( "CalcServerApp","Programming_CSharp.Calculator",
        "theEndPoint",WellKnownObjectMode.Singleton );
```

Change the object to `SingleCall`:

```
RemotingConviguration.RegisterWellKnownServiceType
    ( "CalcServerApp","Programming_CSharp.Calculator",
        "theEndPoint",WellKnownObjectMode.SingleCall );
```

The output reflects that a new object is created to handle each request:

```
Calculator constructor
Press [enter] to exit...
Calculator constructor
Add 3 + 4
Calculator constructor
Sub 3 - 4
Calculator constructor
```

```
Mult 3 * 4
Calculator constructor
Div 3 / 4
```

Threads

Operating systems use processes to separate the different applications that they are executing. Threads are the basic unit to which an operating system allocates processor time, and more than one thread can be executing code inside that process. Each thread maintains exception handlers, a scheduling priority, and a set of structures the system uses to save the thread context until it is scheduled. The thread context includes all the information the thread needs to seamlessly resume execution, including the thread's set of CPU registers and stack, in the address space of the thread's host process.

The .NET Framework further subdivides an operating system process into lightweight managed subprocesses, called application domains, represented by System.AppDomain. One or more managed threads (represented by System.Threading.Thread) can run in one or any number of application domains within the same managed process. Although each application domain is started with a single thread, code in that application domain can create additional application domains and additional threads. The result is that a managed thread can move freely between application domains inside the same managed process; you might have only one thread moving among several application domains.

An operating system that supports preemptive multitasking creates the effect of simultaneous execution of multiple threads from multiple processes. It does this by dividing the available processor time among the threads that need it, allocating a processor time slice to each thread one after another. The currently executing thread is suspended when its time slice elapses, and another thread resumes running. When the system switches from one thread to another, it saves the thread context of the preempted thread and reloads the saved thread context of the next thread in the thread queue.

The length of the time slice depends on the operating system and the processor. Because each time slice is small, multiple threads appear to be executing at the same time, even if there is only one processor. This is actually the case on multiprocessor systems, where the executable threads are distributed among the available processors.

When to Use Multiple Threads

Software that requires user interaction must react to the user's activities as rapidly as possible to provide a rich user experience. At the same time, however, it must do the calculations necessary to present data to the user as fast as possible. If your application uses only one thread of execution, you can combine asynchronous programming with .NET remoting or XML Web services created using ASP.NET to use the processing time of other computers in addition to that of your own to increase responsiveness to the user and decrease the data processing time of your application. If you are doing intensive input/output work, you can also use I/O completion ports to increase your application's responsiveness.

Advantages of Multiple Threads

Using more than one thread, however, is the most powerful technique available to increase responsiveness to the user and process the data necessary to get the job done at almost the same time. On a computer with one processor, multiple threads can create this effect, taking advantage of the small periods of time in between user events to process the data in the background. For example, a user can edit a spreadsheet while another thread is recalculating other parts of the spreadsheet within the same application.

Without modification, the same application would dramatically increase user satisfaction when run on a computer with more than one processor. Your single application domain could use multiple threads to accomplish the following tasks:

- Communicate over a network, to a Web server, and to a database.
- Perform operations that take a large amount of time.
- Distinguish tasks of varying priority. For example, a high-priority thread manages time-critical tasks, and a low-priority thread performs other tasks.
- Allow the user interface to remain responsive, while allocating time to background tasks.

Disadvantages of Multiple Threads

It is recommended that you use as few threads as possible, thereby minimizing the use of operating-system resources and improving performance. Threading also has resource requirements and potential conflicts to be considered when designing your application. The resource requirements are as follows:

- The system consumes memory for the context information required by processes, **AppDomain** objects, and threads. Therefore, the number of processes, **AppDomain** objects, and threads that can be created is limited by available memory.
- Keeping track of a large number of threads consumes significant processor time. If there are too many threads, most of them will not make significant progress. If most of the current threads are in one process, threads in other processes are scheduled less frequently.
- Controlling code execution with many threads is complex, and can be a source of many bugs.
- Destroying threads requires knowing what could happen and handling those issues.

Providing shared access to resources can create conflicts. To avoid conflicts, you must synchronize, or control the access to, shared resources. Failure to synchronize access properly (in the same or different application domains) can lead to problems such as deadlocks (in which two threads stop responding while each waits for the other to complete) and race conditions (when an anomalous result occurs due to an unexpected critical dependence on the timing of two events). The system provides synchronization objects that can be used to coordinate resource sharing among multiple threads. Reducing the number of threads makes it easier to synchronize resources.

Resources that require synchronization include:

- System resources (such as communications ports).
- Resources shared by multiple processes (such as file handles).
- The resources of a single application domain (such as global, static, and instance fields) accessed by multiple threads.

Threading and Application Design

In general, using the ThreadPool class is the easiest way to handle multiple threads for relatively short tasks that will not block other threads and when you do not expect any particular scheduling of the tasks. However, there are a number of reasons to create your own threads:

- If you need a task to have a particular priority.
- If you have a task that might run a long time (and therefore block other tasks).
- If you need to place threads into a single-threaded apartment (all **ThreadPool** threads are in the multithreaded apartment).
- If you need a stable identity associated with the thread. For example, you should use a dedicated thread to abort that thread, suspend it, or discover it by name.

UNIT-V

TWO MARKS

1. What are assemblies?

An assembly is a collection of file that appear to be a single dill or executable (exe) Assemblies are .NET unit of reuse, versioning, security and deployment.

2. What is the difference between single call and singleton?

With a well-known singleton object, all messages for the object, from all clients, are dispatched to a single objet running on the server. With a well-known single-call object, each new message from a client is handling by a new object.

3. Define metadata?

Assemblies are the .NET unit of reuse, versioning, security, and deployment. In addition to the objects code for the application, assemblies contain resources.

4. What is a PF file?

On disk, assemblies are portable executable files, PF files are not new. The format of a.NET PE file is exactly the same as a normal windows PE files. PE files are implemented as DLLS or EXEs.it consists of one or more modules.

5. Define mulimodule assemblies

A multimodule assembly consists of multiple files. The assembly manifest in this case can reside in a standalone file, or it can be embedded in one of the modules. when the assembly is referred the runtime loads the file containing the manifest and then loads the required modules as needed.

6. Define shared assemblies

If you want to share your assembly, it must meet certain stringent requirement Your assembly must have a strong name. strong name are globally unique. To share your assembly place it in the global assembly cache .this is an area of the file system set aside by the CLR to hold shared assemblies.

7. Define private assemblies

Assemblies come in two flavors: Private and shared. Private assemblies are intended to be used by only one application. Shared assemblies are intended to be shared among many applications.

8. Define attributes

Attributes are a mechanism for adding metadata, such as compiler instructions and other data about your data, method, and classes to the program itself. Attributes are inserted into the metadata and are visible through ILDasam and other metadata-reading tools.

9. Define reflection

Reflection is the process by which a program can read its own metadata or metadata from another program. A program is said to reflect on itself or on another program, extracting metadata from the reflected assembly and using that metadata either to inform the user or to modify the program's behavior.

10. Define: Marshaling

The process of moving an object to be remoted is called marshaling.