Semester-5

BCA 503

.Net Framework & C#

(According to Purvanchal University Syllabus)

"Full Line By Line Notes"

Created By D.P.Mishra

On August 22/08/19

Unit – 1

The .Net Framework

Introduction–

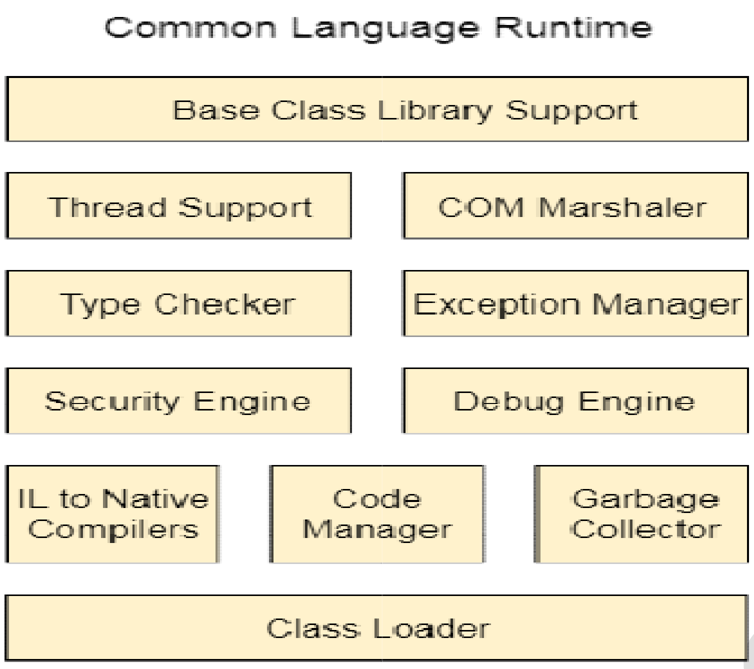
- .NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000.
- It is used to develop applications for web, Windows, phone. Moreover, it provides a broad range of functionalities and support.
- This framework contains a large number of class libraries known as Framework Class Library (FCL). The software programs written in .NET are executed in the execution environment, which is called CLR (Common Language Runtime). These are the core and essential parts of the .NET framework.
- The .Net Framework supports more than 60 programming languages such as C#, F#, VB.NET, J#, VC++, JScript.NET, APL, COBOL, Perl, Oberon, ML, Pascal, Eiffel, Smalltalk, Python, Cobra, ADA, etc.

The .NET Framework is composed of four main components:

- Common Language Runtime (CLR)
- Framework Class Library (FCL),
- Core Languages (WinForms, ASP.NET, and ADO.NET), and
- Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library, etc.)

Common Language Run Time–

It is a program execution engine that loads and executes the program. It converts the program into native code. It acts as an interface between the framework and operating system. It does exception handling, memory management, and garbage collection. Moreover, it provides security, type-safety, interoperability, and portability. A list of CLR components are given below:



Common Type System–

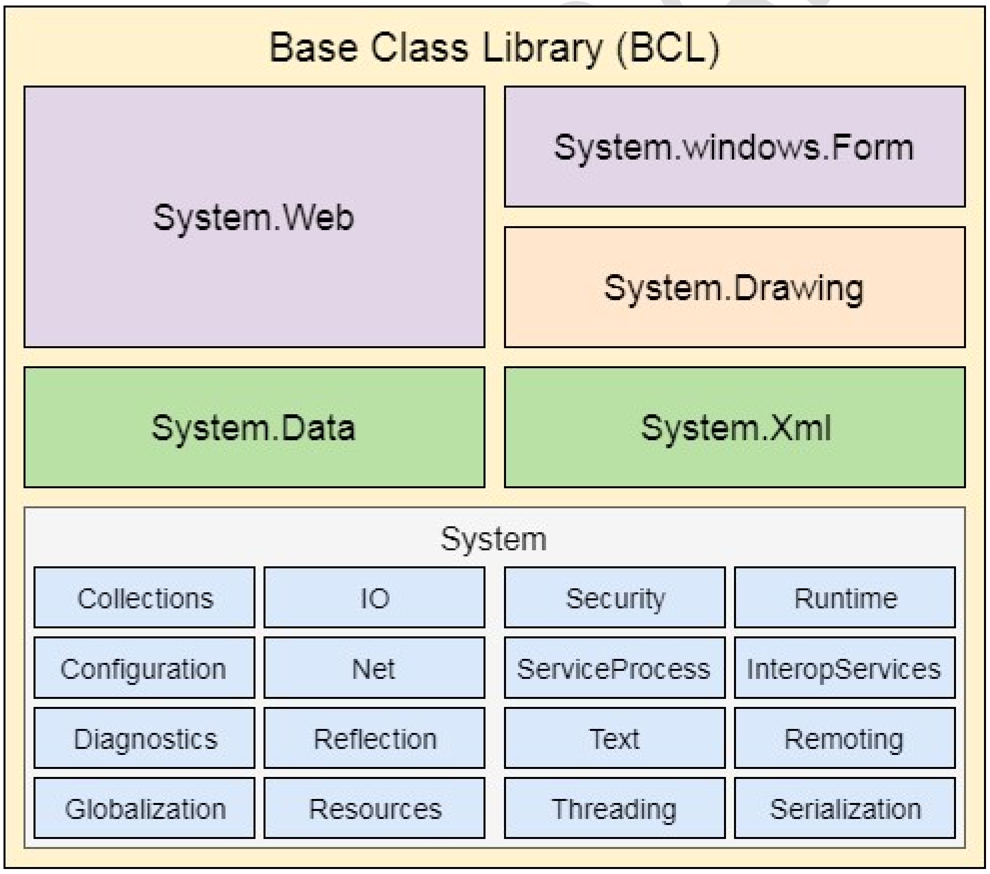
- The Common Type System (CTS) is a standard for defining and using data types in the .NETframework. CTS defines a collection of data types, which are used and managed by the run time to facilitate cross-language integration.
- CTS provide the types in the .NET Framework with which .NET applications, components and controls are built in different programming languages so information is shared easily.

Common Language Specification–

- The Common Language Specification (CLS) is a fundamental set of language features supported by the Common Language Runtime (CLR) of the .NET Framework.
- CLS is a part of the specifications of the .NET Framework. CLS was designed to support language constructs commonly used by developers and to produce verifiable code, which allows all CLS-compliant languages to ensure the type safety of code.

The base class library–

- .NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.
- It contains thousands of classes that support the following functions.
  - Base and user-defined data types
  - Support for exceptions handling
  - input/output and stream operations
  - Communications with the underlying system
  - Access to data
  - Ability to create Windows-based GUI applications
  - Ability to create web-client and server applications
  - Support for creating web services
- .NET Base Class Library is the sub part of the Framework that provides library support to Common Language Runtime to work properly. It includes the System namespace and core types of the .NET framework.



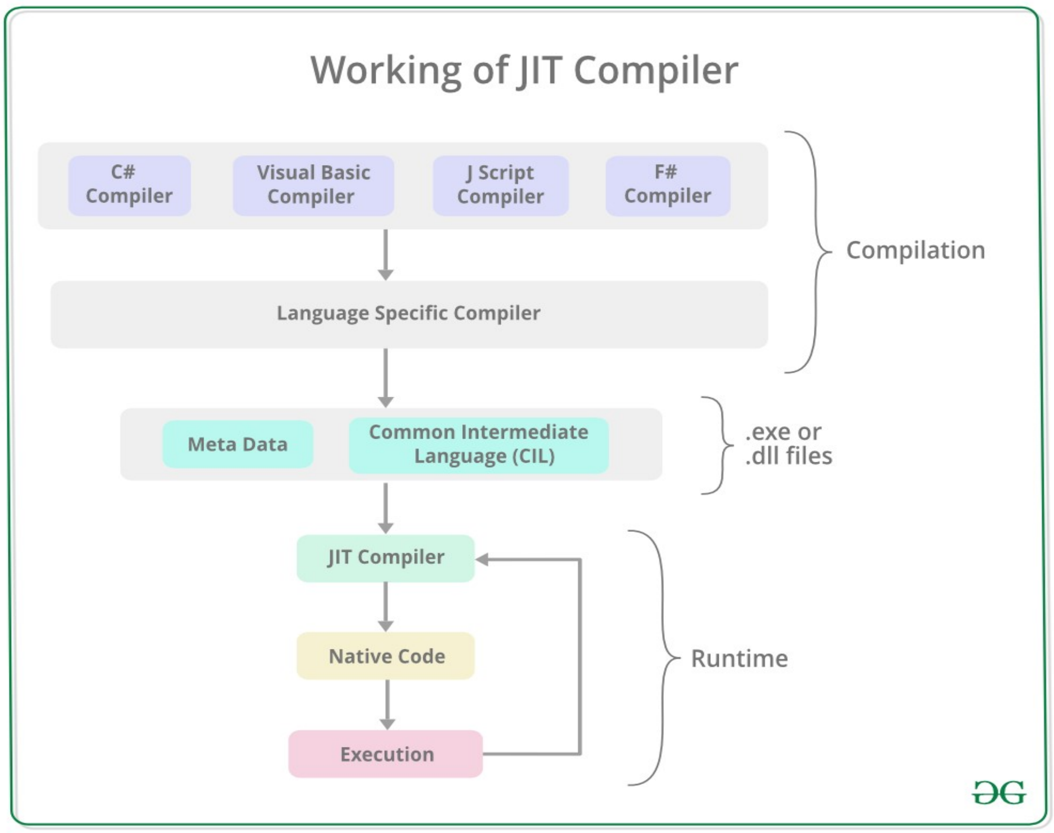
The .Net class Library intermediate language–

- Intermediate language (IL) is an object-oriented programming language designed to be used by compilers for the .NET Framework before static or dynamic compilation to machine code. The IL is used by the .NET Framework to generate machine-independent code as the output of compilation of the source code written in any .NET programming language.
- IL is a stack-based assembly language that gets converted to bytecode during execution of a virtual machine. It is defined by the common language infrastructure (CLI) specification. As IL is used for automatic generation of compiled code, there is no need to learn its syntax.

Just in time compilation–

- Just-In-Time compiler (JIT) is a part of **Common Language Runtime (CLR)** in *.NET* which is responsible for managing the execution of *.NET* programs regardless of any *.NET* programming language.
- A language-specific compiler converts the source code to the intermediate language. This intermediate language is then converted into the machine code by the Just-In-Time (JIT) compiler. This machine code is specific to the computer environment that the JIT compiler runs on.

Working of JIT Compiler: The JIT compiler is required to speed up the code execution and provide support for multiple platforms. Its working is given as follows:



Garbage Collection–

- Automatic memory management is made possible by **Garbage Collection in .NET Framework**. When a class object is created at runtime, certain memory space is allocated to it in the heap memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

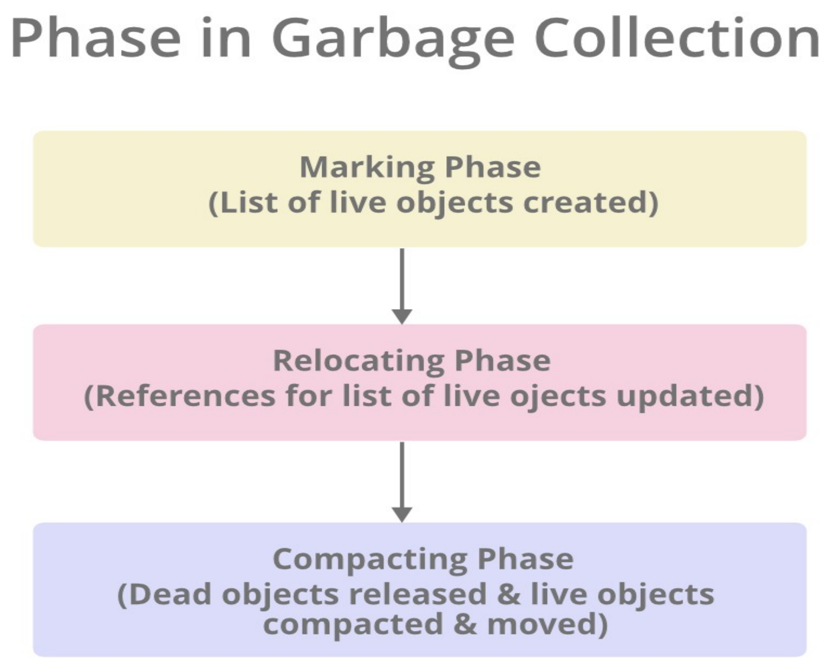
- Garbage collection will always work on **Managed Heap** and internally it has an Engine which is known as the **Optimization Engine**.

Garbage Collection occurs if at least one of multiple conditions is satisfied. These conditions are given as follows:

  - If the system has low physical memory, then garbage collection is necessary.
  - If the memory allocated to various objects in the heap memory exceeds a pre-set threshold, then garbage collection occurs.
  - If the *GC.Collect* method is called, then garbage collection occurs. However, this method is only called under unusual situations as normally garbage collector runs automatically.

Phases in Garbage Collection

There are mainly **3** phases in garbage collection. Details about these are given as follows:


Phase in Garbage Collection
Marking Phase (List of live objects created)
Relocating Phase (References for list of live ojects updated)
Compacting Phase (Dead objects released & live objects compacted & moved)

1. **Marking Phase:** A list of all the live objects is created during the marking phase. This is done by following the references from all the root objects. All of the objects that are not on the list of live objects are potentially deleted from the heap memory.
2. **Relocating Phase:** The references of all the objects that were on the list of all the live objects are updated in the relocating phase so that they point to the new location where the objects will be relocated to in the compacting phase.
3. **Compacting Phase:** The heap gets compacted in the compacting phase as the space occupied by the dead objects is released and the live objects remaining are moved. All the live objects that remain after the garbage collection are moved towards the older end of the heap memory in their original order.

Unit – 2

C# Basics

Introduction–

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.

Creating Hello World Program

A C# program consists of the following parts −

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

Let us look at a simple code that prints the words "Hello World" –

```
using System;

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

When this code is compiled and executed, it produces the following result –

```
Hello World
```

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.

The following table lists the available value types in C# 2010 –

Type	Represents	Range	Default Value
bool	Boolean value	True or False	False

byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	(-7.9 x 10^{28} to 7.9 x 10^{28}) / 10^{0} to 28	0.0M
double	64-bit double-precision floating point type	(+/-)5.0 x 10^{-324} to (+/-)1.7 x 10^{308}	0.0D
float	32-bit single-precision floating point type	-3.4 x 10^{38} to + 3.4 x 10^{38}	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0

Example:

```
using System;

namespace DataTypeApplication {
  class Program {
    static void Main(string[] args) {
      Console.WriteLine("Size of int: {0}", sizeof(int));
      Console.ReadLine();
    }
  }
}
```

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

Size of int: 4

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**.

Object Type

The **Object Type** is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System.Object class. The object types can be assigned values of any other types, value types, reference types, predefined or user-defined types. However, before assigning values, it needs type conversion.

When a value type is converted to object type, it is called **boxing** and on the other hand, when an object type is converted to a value type, it is called **unboxing**.

```
object obj;
obj = 100; // this is boxing
```

Dynamic Type

You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

Syntax for declaring a dynamic type is −

dynamic <variable_name> = value;

For example,

```
dynamic d = 20;
```

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

String Type

The **String Type** allows you to assign any string values to a variable. The string type is an alias for the System.String class. It is derived from object type. The value for a string type can be assigned using string literals in two forms: quoted and @quoted.

For example:

```
String str = "DPMishrakiDiary";
```

A @quoted string literal looks as follows −

```
@"DPMishrakiDiary";
```

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;
```

Identifiers–

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in C# are as follows −

- A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
- It must not contain any embedded space or symbol such as? - + ! @ # % ^ & * ( ) [ ] { } . ; : " ' / and \. However, an underscore (_) can be used.
- It should not be a C# keyword.

The following are some examples of identifiers −

Class Names

```
class Calculation
class Demo
```

Valid Functions Names

```
void display()
void getMarks()
```

Variable Names

```
int a;
int marks;
int rank;
double res_marks;
```

Variables and Constants–

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

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.

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. */
```

Accepting Values from User

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

For **example,**

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

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

Constants:

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

C# Statements–

It contains three types of statements.

1. Simple Statements

2. Compound Statements

3. Control Statements

Simple Statements

A simple statement is any expression that terminates with a semicolon. For example

```
varl = var2 + var3;

var3 = varl++;

var3++;
```

Compound Statements:

Related statements can be grouped together in braces to form a compound statement or block. For example:

```
{
int i = 4;
Console.WriteLine (i);
i++;
}
```

Semantically, a block behave like a statement and can be used anywhere a single statement is allowed. There is no semicolon after the closing braces. Any variable declared in a block remain in scope up to the closing brace. Once the block is exited, the block variables cease to exist. Blocking improves readability of program code and can help make your program easier to control and debug.

Control Statements

Again control statements are divided into three statements

(a) Loop statements

(b) Jump statements

(c) Selection statements

(a) Loop Statements

C# provides a number of the common loop statements:

• while

• do-while

• for

• foreach

while loops

Syntax: while (expression) statement[s]

A 'while' loop executes a statement, or a block of statements wrapped in curly braces, repeatedly until the condition specified by the Boolean expression returns false. For instance, the following code.

```
int a =0;
While (a < 3) {
```

```
System.Console. WriteLine (a);
a++;
}
```

Produces the following output:

```
0
1
2
```

do-while loops

Syntax: do statement [s] while (expression)

A 'do-while' loop is just like a 'while' loop except that the condition is evaluated after the block of code specified in the 'do' clause has been run. So even where the condition is initially false, the block runs once. For instance, the following code outputs '4':

```
Int a = 4;
do
{
System.Console. WriteLine (a);
a++;
} while (a < 3);
```

for loops

Syntax: for (statement1; expression; statement2) statement[s]3

The 'for' clause contains three part. Statementl is executed before the loop is entered.

The loop which is then executed corresponds to the following 'while' loop:

```
Statementl

while (expression) {statement[s]3; statement2}
```

'for' loops tend to be used when one needs to maintain an iterator value. Usually, as in the following example, the first statement initializes the iterator, the condition evaluates it against an end value, and the second statement changes the iterator value.

```
for (int a =0; a<5; a++)
```

```
{
system.console.writeLine(a);
}
```

foreach loops

syntax:foreach (variablel in variable2) statement[s]

The 'foreach' loop is used to iterate through the values contained by any object which implements the IEnumerable interface. When a 'foreach' loop runs, the given variablel is set in turn to each value exposed by the object named by variable2. As we have seen previously, such loops can be used to access array values. So, we could loop through the values of an array in the following way:

```
int[] a = new int[]{1,2,3};
foreach (int b in a)
system.console.writeLine (b);
```

The main drawback of 'foreach' loops is that each value extracted (held in the given example by the variable 'b') is read-only.

(b) Jump Statements

The jump statements include

- break
- continue
- goto
- return
- throw

break

The following code gives an example - of how it could be used. The output of the loop is the numbers from. 0 to 4.

```
int a = 0;
while (true)
{
system.console.writeLine(a);
a++;
if (a == 5)
break;
}
```

Continue

The 'continue' statement can be placed in any loop structure. When it executes, it moves the program counter immediately to the next iteration of the loop. The following code example uses the 'continue' statement to count the number of values between 1 and 100 inclusive that are not multiples of seven. At the end of the loop the variable y holds the required value.

```
int y = 0;
for (int x=l; x<101; x++)
{
if ((x % 7) == 0)
continue;
y++;
}
```

Goto

The 'goto' statement is used to make a jump to a particular labeled part of the program code. It is also used in the 'switch' statement described below. We can use a'goto' statement to construct a loop, as in the following example (but again, this usage is not recommended):

```
int a = 0;
start:
system.console.writeLine(a);
a++;
if (a < 5)
goto start;
```

(c) *Selection Statements*

C# offers two basic types of selection statement:

• if-else

• switch-default

if-else

'if-else' statements are used to run blocks of code conditionally upon a boolean expression evaluating to true. The 'else' clause, present in the following example, is optional.

```
if (a == 5)
system.console.writeLine("A is 5");
else
system.console.writeLine("A is not 5");
```

switch-default

'switch' statements provide a clean way of writing multiple if - else statements. In the following example, the variable whose value is in question is 'a'. If 'a' equals 1, then the output is 'a>0'; if a equals 2, then the output is 'a> 1 and a>0'. Otherwise, it is reported that the variable is not set.

```
switch(a)
{
case 2:
Console.writeLine("a>l and ");
goto case 1;
case 1:
console.writeLine("a>0");
break;
default:
console.writeLine("a is not set");
break;
}
```

Object Oriented Concept–

Object Oriented Programming (OOP) is a programming model where programs are organized around objects and data rather than action and logic.

OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

1. The software is divided into a number of small units called objects. The data and functions are built around these objects.

2. The data of the objects can be accessed only by the functions associated with that object.
3. The functions of one object can access the functions of another object.

OOP has the following important features.

Class

A class is the core of any modern Object Oriented Programming language such as C#.

In OOP languages it is mandatory to create a class for representing data.

A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.

A class will not occupy any memory space and hence it is only a logical representation of data.

To create a class, you simply use the keyword "class" followed by the class name:

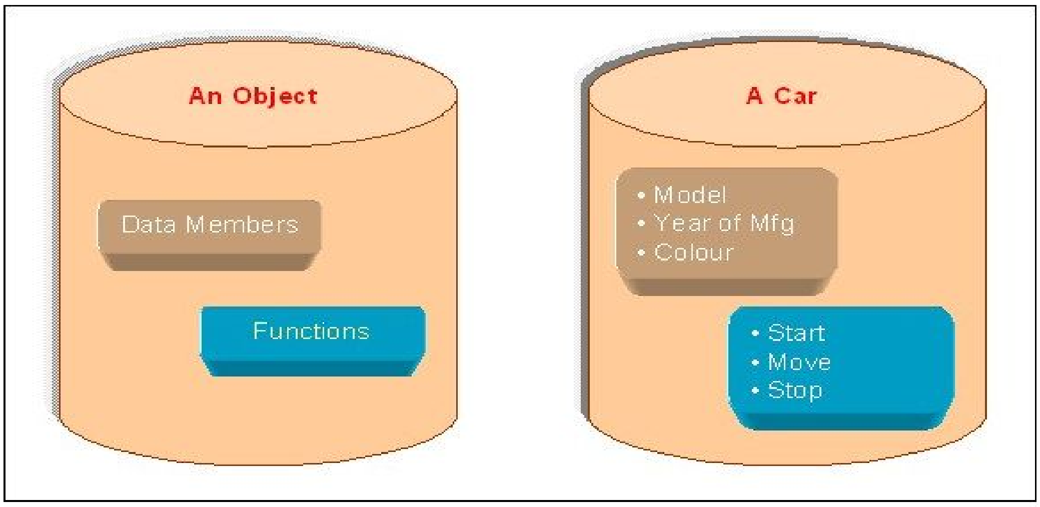
```
class Employee

{

}
```

Object

Objects are the basic run-time entities of an object oriented system. They may represent a person, a place or any item that the program must handle.

"An object is a software bundle of related variable and methods."

"An object is an instance of a class"



A class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, that is called an object.

When an object is created using the new operator, memory is allocated for the class in the heap, the object is called an instance and its starting address will be stored in the object in stack memory.

When an object is created without the new operator, memory will not be allocated in the heap, in other words an instance will not be created and the object in the stack contains the value **null**.

When an object contains null, then it is not possible to access the members of the class using that object.

```
class Employee
{

}
```

Syntax to create an object of class Employee:

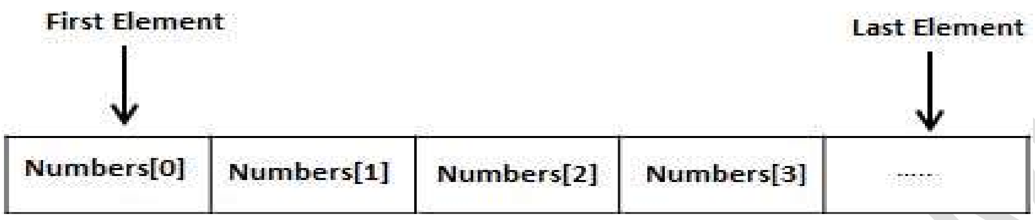
```
Employee objEmp = new Employee();
```

Array and Strings–

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,

- *datatype* 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;

Initializing an Array

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array.

Array is a reference type, so you need to use the **new** keyword to create an instance of the array. For **example**,

double[] balance = new double[10];

Assigning Values to an Array

You can assign values to individual array elements, by using the index number, **like** −

```
double[] balance = new double[10];
balance[0] = 4500.0;
```

You can assign values to the array at the time of declaration, **as shown** −

double[] balance = { 2340.0, 4523.69, 3421.0};

String

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

The following example demonstrates this –

```
using System;

namespace StringApplication {

  class Program {

    static void Main(string[] args) {
      //from string literal and string concatenation
      string fname, lname;
      fname = "DP";
      lname = "Mishra";

      char []letters= { 'H', 'e', 'l', 'l','o' };
      string [] sarray={ "Hello", "From", "DPMishraki", "Diary" };

      string fullname = fname + lname;
      Console.WriteLine("Full Name: {0}", fullname);

      //by using string constructor { 'H', 'e', 'l', 'l','o' };
      string greetings = new string(letters);
      Console.WriteLine("Greetings: {0}", greetings);
```

```
        //methods returning string { "Hello", "From", "DPMishraki", "Diary" };
        string message = String.Join(" ", sarray);
        Console.WriteLine("Message: {0}", message);

        //formatting method to convert a value
        DateTime waiting = new DateTime(2012, 10, 10, 17, 58, 1);
        string chat = String.Format("Message sent at {0:t} on {0:D}", waiting);
        Console.WriteLine("Message: {0}", chat);
      }
    }
}
```

When the above code is compiled and executed, it produces the **following result –**

```
Full Name: DPMishra
Greetings: Hello
Message: Hello From DPMishraki Diary
Message: Message sent at 5:58 PM on Wednesday, October 10, 2012
```

Unit – 3

C# Using Libraries

Namespace Systems–

A **namespace** is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows −

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

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

```
namespace_name.item_name;
```

The following program demonstrates use of namespaces −

```
using System;

namespace first_space {
  class namespace_cl {
    public void func() {
      Console.WriteLine("Inside first_space");
    }
  }
}
namespace second_space {
  class namespace_cl {
    public void func() {
      Console.WriteLine("Inside second_space");
    }
  }
}
class TestClass {
  static void Main(string[] args) {
    first_space.namespace_cl fc = new first_space.namespace_cl();
    second_space.namespace_cl sc = new second_space.namespace_cl();
    fc.func();
    sc.func();
    Console.ReadKey();
  }
}
```

When the above code is compiled and executed, it produces the following **result –**

```
Inside first_space
Inside second_space
```

Input-Output–

The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

The following table shows some commonly used non-abstract classes in the System.IO namespace –

Sr.No.	I/O Class & Description
1	**BinaryReader** Reads primitive data from a binary stream.
2	**BinaryWriter** Writes primitive data in binary format.
3	**BufferedStream** A temporary storage for a stream of bytes.
4	**Directory** Helps in manipulating a directory structure.
5	**DirectoryInfo** Used for performing operations on directories.
6	**DriveInfo** Provides information for the drives.

7	**File** Helps in manipulating files.
8	**FileInfo** Used for performing operations on files.
9	**FileStream** Used to read from and write to any location in a file.
10	**MemoryStream** Used for random access to streamed data stored in memory.
11	**Path** Performs operations on path information.
12	**StreamReader** Used for reading characters from a byte stream.
13	**StreamWriter** Is used for writing characters to a stream.
14	**StringReader** Is used for reading from a string buffer.
15	**StringWriter** Is used for writing into a string buffer.

Multi-threading–

- A **thread** is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

- Threads are **lightweight processes**. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.
- So far we wrote the programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads.

Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread –

- **The Unstarted State** – It is the situation when the instance of the thread is created but the Start method is not called.
- **The Ready State** – It is the situation when the thread is ready to run and waiting CPU cycle.
- **The Not Runnable State** – A thread is not executable, when
  - Sleep method has been called
  - Wait method has been called
  - Blocked by I/O operations
- **The Dead State** – It is the situation when the thread completes execution or is aborted.

Networking and Sockets–

The .NET Framework has a layered, extensible, and managed implementation of networking services. You can easily integrate them into your applications. Use the System.Net; namespace.

Let us see how to acess the Uri class:.In C#, it provides object representation of a uniform resource identifier (URI) –

```
Uri uri = new Uri("http://www.example.com/");
WebRequest w = WebRequest.Create(uri);
```

Let us now see the System.Net class. This is used to encorypt connections using using the Secure Socket Layer (SSL). If the URI begins with "https:", SSL is used; if the URI begins with "http:", an unencrypted connection is used.

The following is an example. For SSL with FTP, set the EnableSsl property to true before calling the GetResponse() method.

```
String uri = "https://www.example.com/";
WebRequest w = WebRequest.Create(uri);

String uriServer = "ftp://ftp.example.com/new.txt"
FtpWebRequest r = (FtpWebRequest)WebRequest.Create(uriServer);
r.EnableSsl = true;
r.Method = WebRequestMethods.Ftp.DeleteFile;
```

The following is an example showing the usage of System.Net namespace and using the Dns.GetHostEntry, Dns.GetHostName methods and IPHostEntry property AddressList –

Example

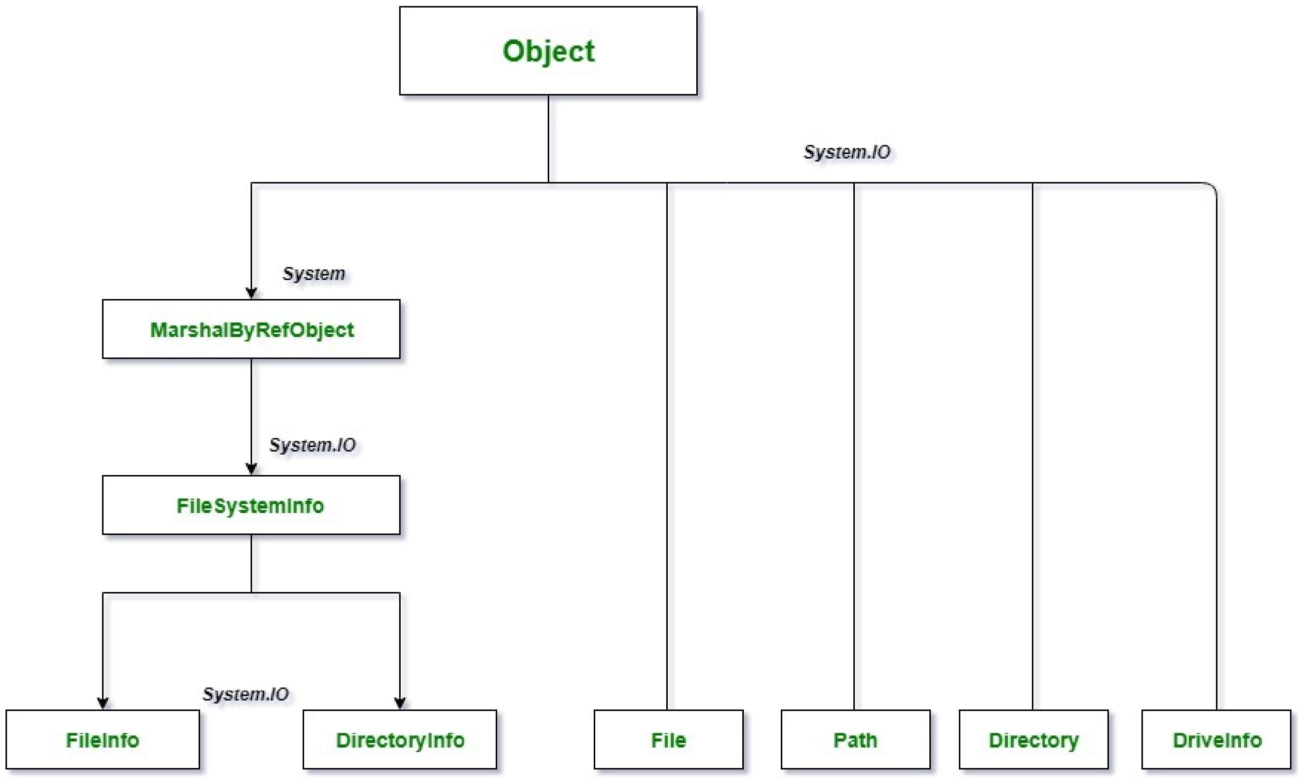
```
using System;

using System.Net;



class Program {

  static void Main() {



    String hostName = string.Empty;

    hostName = Dns.GetHostName();

    Console.WriteLine("Hostname: "+hostName);

    IPHostEntry myIP = Dns.GetHostEntry(hostName);



    IPAddress[] address = myIP.AddressList;



    for (int i = 0; i < address.Length; i++) {

      Console.WriteLine("IP Address {1} : ",address[i].ToString());
    }

    Console.ReadLine();

  }

}
```

Socket

- Socket programming is a way of connecting two nodes on a network to communicate with each other. Basically, it is a one-way Client and Server setup where a Client connects, sends messages to the server and the server shows them using socket connection.
- One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection.
- Server forms the listener socket while the client reaches out to the server. Before going deeper into Server and Client code, it is strongly recommended to go through **TCP/IP Model**.

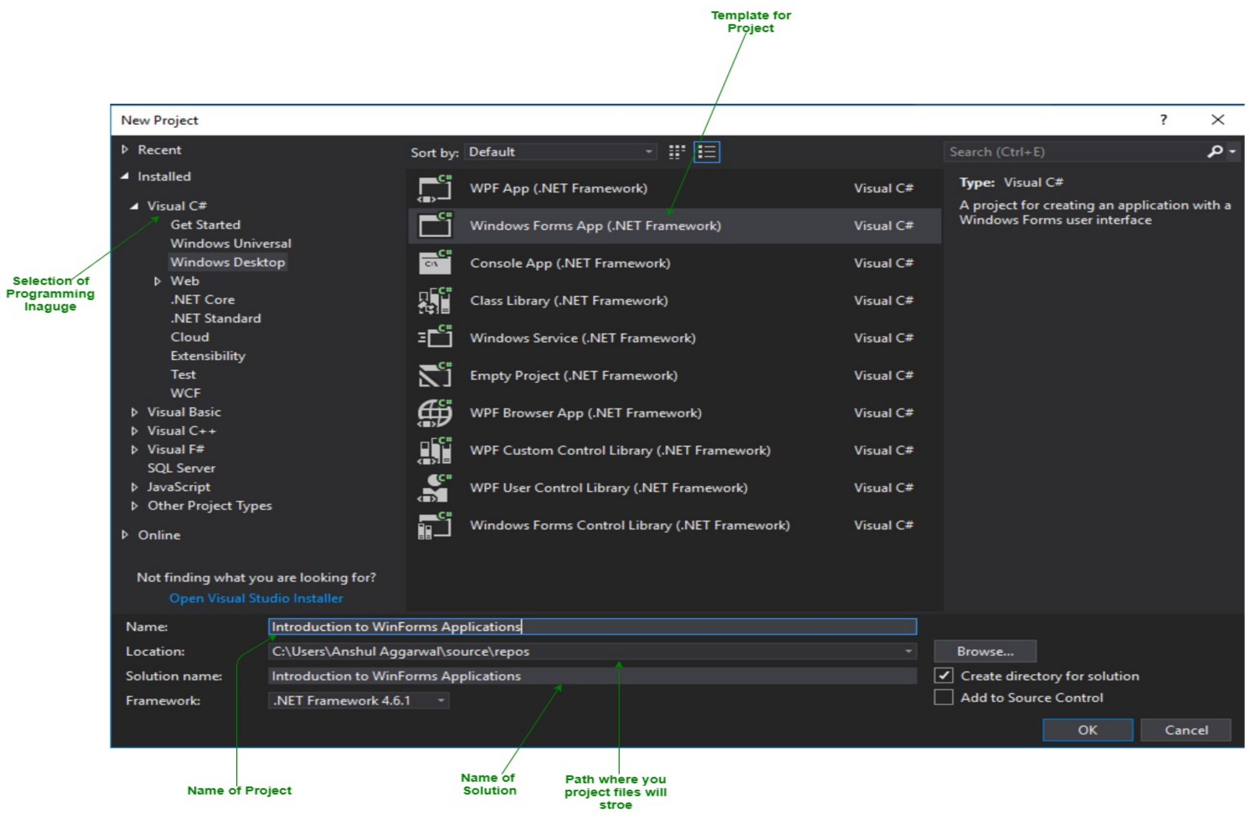
Data handling–

- Generally, the file is used to store the data. The term File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc.
- There are two basic operation which is mostly used in file handling is reading and writing of the file.
- The file becomes stream when we open the file for writing and reading. A stream is a sequence of bytes which is used for communication. Two stream can be formed from file one is input stream which is used to read the file and another is output stream is used to write in the file.
- In C#, *System.IO* namespace contains classes which handle input and output streams and provide information about file and directory structure.



Windows Forms–

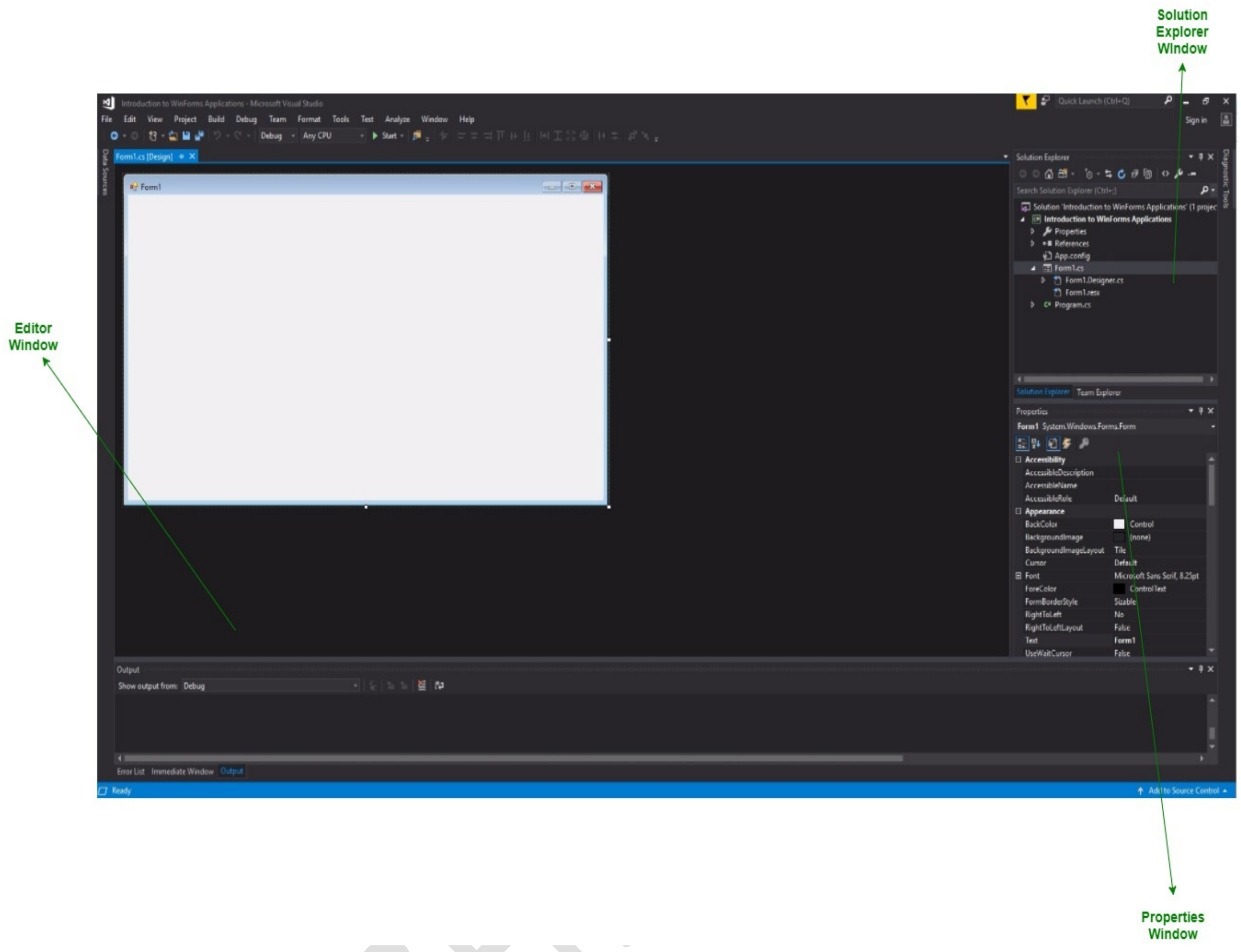
- Windows Forms is a Graphical User Interface(GUI) class library which is bundled in *.Net Framework*.
- Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs.
- It is also termed as the **WinForms**. The applications which are developed by using Windows Forms or WinForms are known as the **Windows Forms Applications** that runs on the desktop computer.
- WinForms can be used only to develop the Windows Forms Applications not web applications. WinForms applications can contain the different type of controls like labels, list boxes, tooltip etc.
- Creating a Windows Forms Application Using Visual Studio 2017
- First, open the Visual Studio then Go to File -> New -> Project to create a new project and then select the language as Visual C# from the left menu. Click on Windows Forms App(.NET Framework) in the middle of current window. After that give the project name and Click OK.


Template for Project
Selection of Programming inaguge
Name of Project
Name of Solution
Path where you project files will stroe
New Project
Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
.NET Core
.NET Standard
Cloud
Extensibility
Test
WCF
Visual Basic
Visual C++
Visual F#
SQL Server
JavaScript
Other Project Types
Online
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
WPF App (.NET Framework) Visual C#
Windows Forms App (.NET Framework) Visual C#
Console App (.NET Framework) Visual C#
Class Library (.NET Framework) Visual C#
Windows Service (.NET Framework) Visual C#
Empty Project (.NET Framework) Visual C#
WPF Browser App (.NET Framework) Visual C#
WPF Custom Control Library (.NET Framework) Visual C#
WPF User Control Library (.NET Framework) Visual C#
Windows Forms Control Library (.NET Framework) Visual C#
Browse...
OK
Cancel

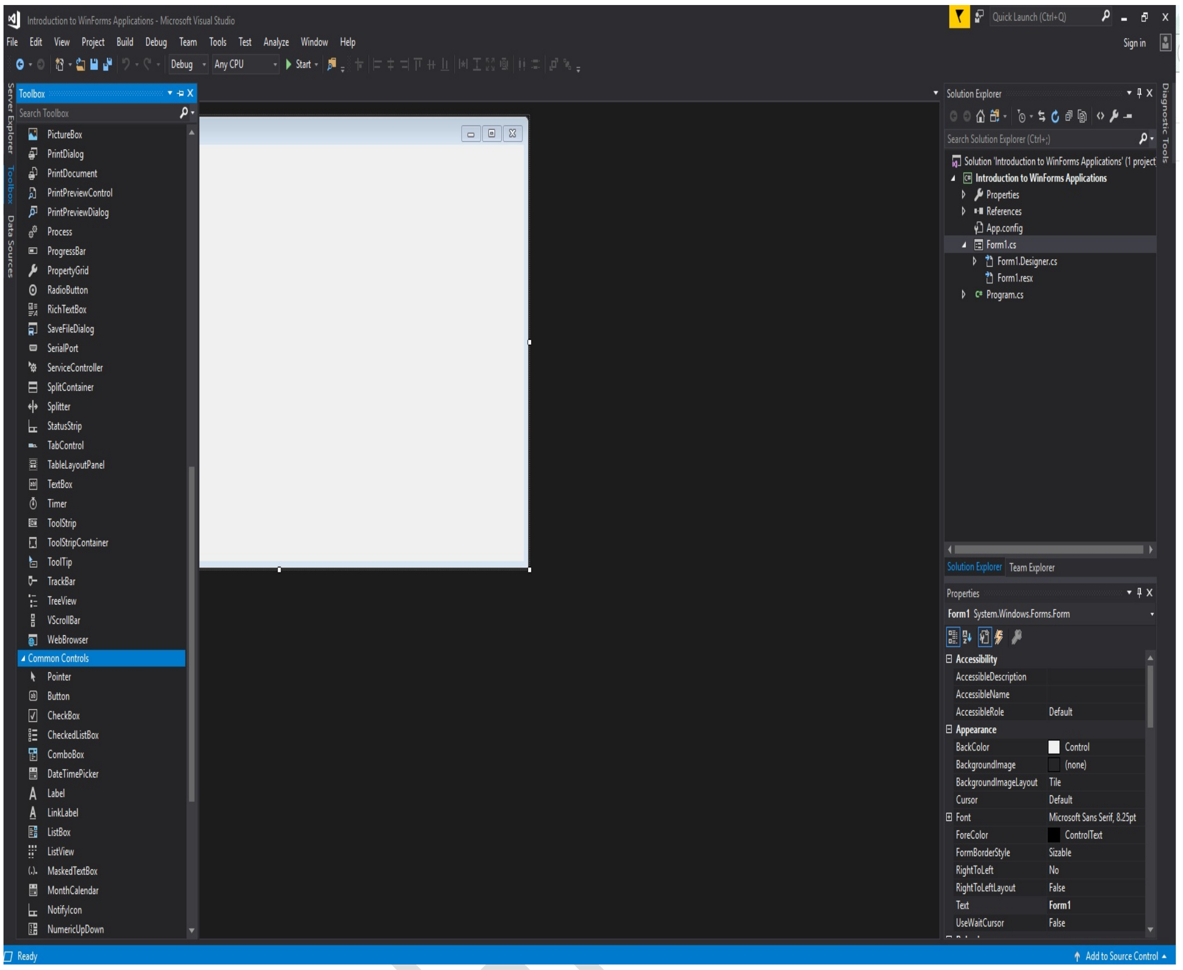
Type: Visual C#
A project for creating an application with a Windows Forms user interface

Name: Introduction to WinForms Applications
Location: C:\Users\Anshul Aggarwal\source\repos
Solution name: Introduction to WinForms Applications
Framework: .NET Framework 4.6.1
✓ Create directory for solution
☐ Add to Source Control

- Here the solution is like a container which contains the projects and files that may be required by the program.
- After that following window will display which will be divided into three parts as follows:
- Editor Window or Main Window: Here, you will work with forms and code editing. You can notice the layout of form which is now blank. You will double click the form then it will open the code for that.
- Solution Explorer Window: It is used to navigate between all items in solution. For example, if you will select a file form this window then particular information will be display in the property window.
- Properties Window: This window is used to change the different properties of the selected item in the Solution Explorer. Also, you can change the properties of components or controls that you will add to the forms.



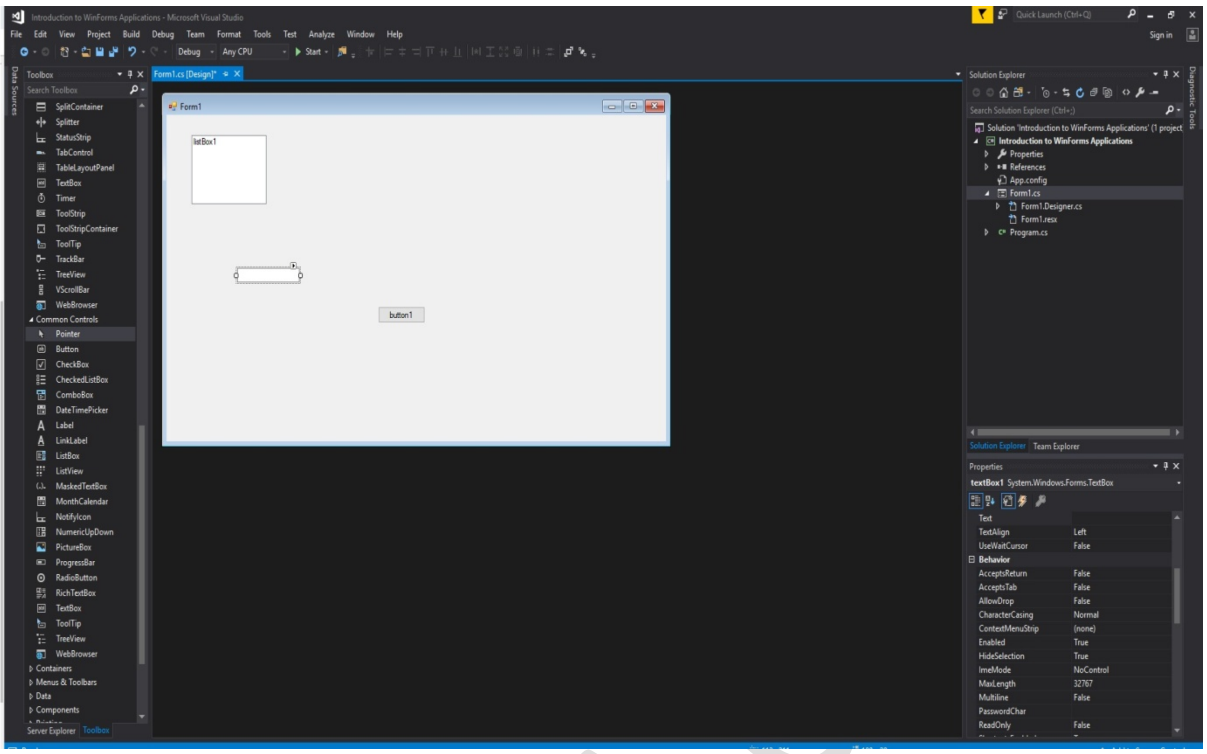
- You can also reset the window layout by setting it to default. To set the default layout, go to Window -> Reset Window Layout in Visual Studio Menu.
- Now to add the controls to your WinForms application go to Toolbox tab present in the extreme left side of Visual Studio. Here, you can see a list of controls. To access the most commonly used controls go to Common Controls present in Toolbox tab.



Form1 System.Windows.Forms.Form

Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Cursor	Default
Font	Microsoft Sans Serif, 8.25pt
ForeColor	ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1
UseWaitCursor	False

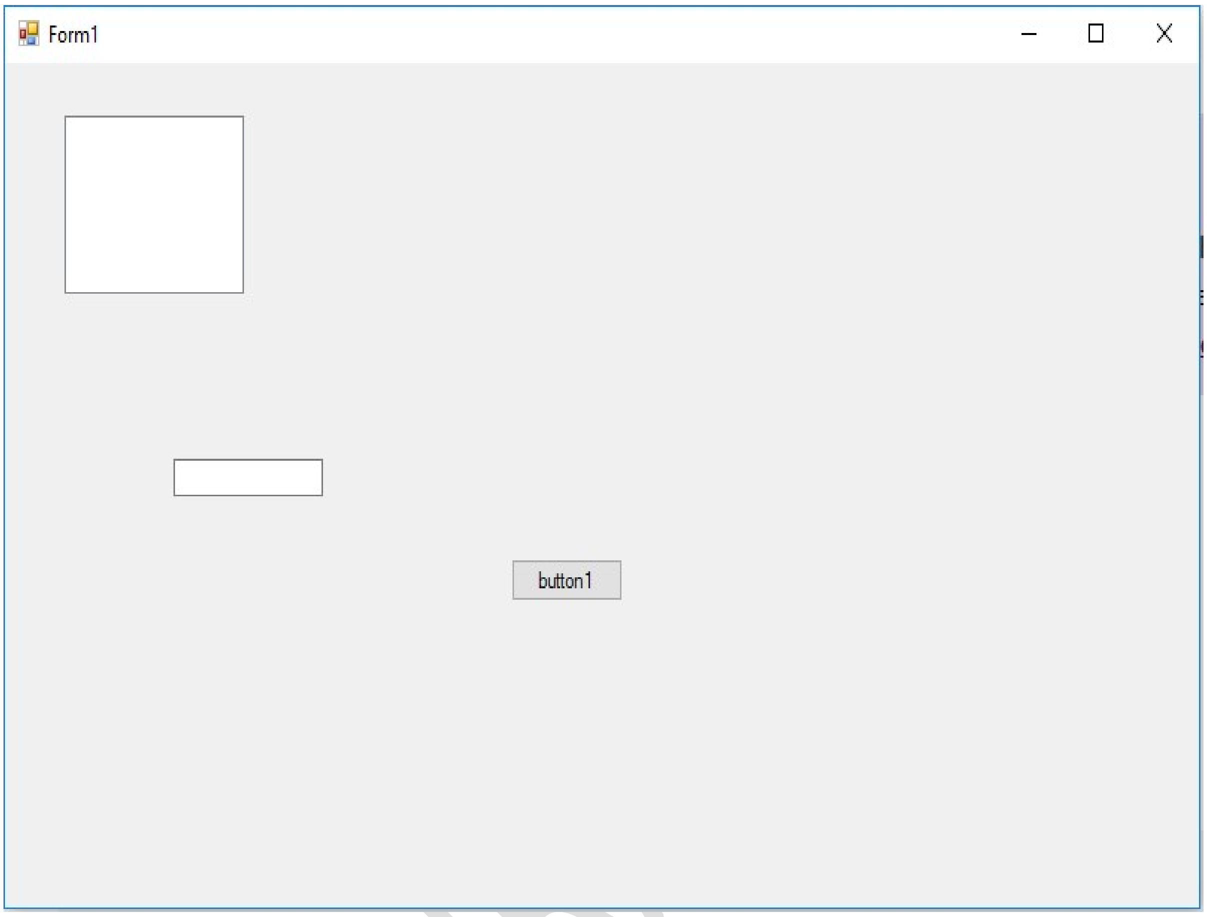
- Now drag and drop the controls that you needed on created Form. For example, if you can add TextBox, ListBox, Button etc. as shown below. By clicking on the particular dropped control you can see and change its properties present in the right most corner of Visual Studio.



textBox1 System.Windows.Forms.TextBox

Text	
TextAlign	Left
UseWaitCursor	False
Behavior	
AcceptsReturn	False
AcceptsTab	False
AllowDrop	False
CharacterCasing	Normal
ContextMenuStrip	(none)
Enabled	True
HideSelection	True
ImeMode	NoControl
MaxLength	32767
Multiline	False
PasswordChar	
ReadOnly	False

- In the above image, you can see the TextBox is selected and its properties like TextAlign, MaxLength etc. are opened in right most corner. You can change its properties' values as per the application need. The code of controls will be automatically added in the background. You can check the Form1.Designer.cs file present in the Solution Explorer Window.
- To run the program you can use an F5 key or Play button present in the toolbar of Visual Studio. To stop the program you can use pause button present in the ToolBar. You can also run the program by going to Debug->Start Debugging menu in the menubar.



Error Handling–

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
}
```

Unit – 4

Advanced features using C#

Web Services–

A web service is a web-based functionality accessed using the protocols of the web to be used by the web applications. There are three aspects of web service development:

- Creating the web service
- Creating a proxy
- Consuming the web service

Creating a Web Service

A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

To understand the concept let us create a web service to provide stock price information. The clients can query about the name and price of a stock based on the stock symbol. To keep this example simple, the values are hardcoded in a two-dimensional array. This web service has three methods:

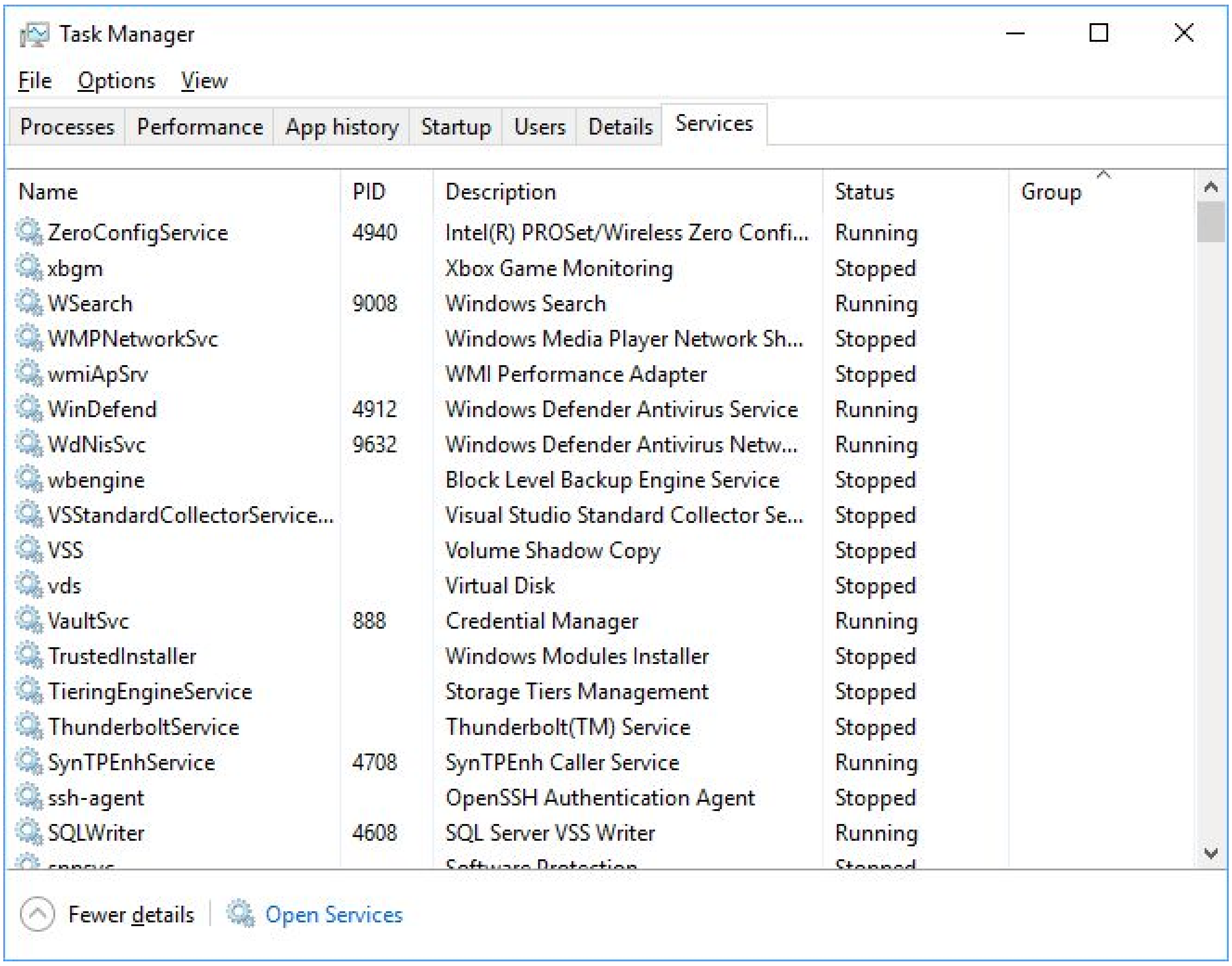
- A default HelloWorld method
- A GetName Method
- A GetPrice Method

Windows Services–

- Windows Services are non UI software applications that run in the background. Windows services are usually starts when an operating system boots and scheduled to run in the background to execute some tasks. Windows services can also be started automatically or manually. You can also manually pause, stop and restart Windows services.
- Windows service is a computer program that runs in the background to execute some tasks. Some examples of Windows services are auto

update of Windows, check emails, print documents, SQL Server agent, file and folder scanning and indexing and so on.

- If you open your Task Manager and click on Services tab, you will see hundreds of services running on your machine. You can also see the statuses of these services. Some services are running, some have paused, and some have stopped. You can start, stop, and pause a service from here by right click on the service.



Name	PID	Description	Status	Group
ZeroConfigService	4940	Intel(R) PROSet/Wireless Zero Confi...	Running	
xbgm		Xbox Game Monitoring	Stopped	
WSearch	9008	Windows Search	Running	
WMPNetworkSvc		Windows Media Player Network Sh...	Stopped	
wmiApSrv		WMI Performance Adapter	Stopped	
WinDefend	4912	Windows Defender Antivirus Service	Running	
WdNisSvc	9632	Windows Defender Antivirus Netw...	Running	
wbengine		Block Level Backup Engine Service	Stopped	
VSStandardCollectorService...		Visual Studio Standard Collector Se...	Stopped	
VSS		Volume Shadow Copy	Stopped	
vds		Virtual Disk	Stopped	
VaultSvc	888	Credential Manager	Running	
TrustedInstaller		Windows Modules Installer	Stopped	
TieringEngineService		Storage Tiers Management	Stopped	
ThunderboltService		Thunderbolt(TM) Service	Stopped	
SynTPEnhService	4708	SynTPEnh Caller Service	Running	
ssh-agent		OpenSSH Authentication Agent	Stopped	
SQLWriter	4608	SQL Server VSS Writer	Running	
sppsvc		Software Protection	Stopped	

You may also find all services running on your machine in the following ways:

- Go to Control Panel select “Services” inside “Administrative Tools”.
- Open Run window (Window + R) and type services.msc and press ENTER.

Messaging 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.

COM and C#–

COM stands for "Common object modal".

COM IS A KIND OF SPECIFICATION TO MAKE THE COMPONENT REUSABLE. ALL THE ACTIVE COMPONENTS ARE BY DEFAULT COM COMPONENTS. COM IS A CONCEPT-ITS PRACTICAL IMPLEMENTATION IS ACTIVEX.

In .NET for this purpose we use assembly
Assembly is the component modal for .NET.

Assembly

1. It contains the meta data (data about data).
2. Project details: Name/version/author/creation date/creation time/ and IL (Intermediate language).

Assembly have two parts:

1. Manifest : Name, Author, Creation date, Time.
2. Body : IL

To view assembly: C:\>ILDASM A.EXE

Types of assembly

1. Private assembly:for same project
2. Public or shared assembly:for any project

Assembly comes in the form of

.DLL (Dynamic link library)
.EXE (Executable file)
Difference between .dll & .exe

DLL	**EXE**
1. Not self executed	1. Self executable
2. No Main()	2. Main()
3. In Process (Execute in client memory)	3. Out Process (Hold separate memory)
4. light weight	4. heavy weight

Localization–

Localization is the process of customizing your application for a given culture and locale.

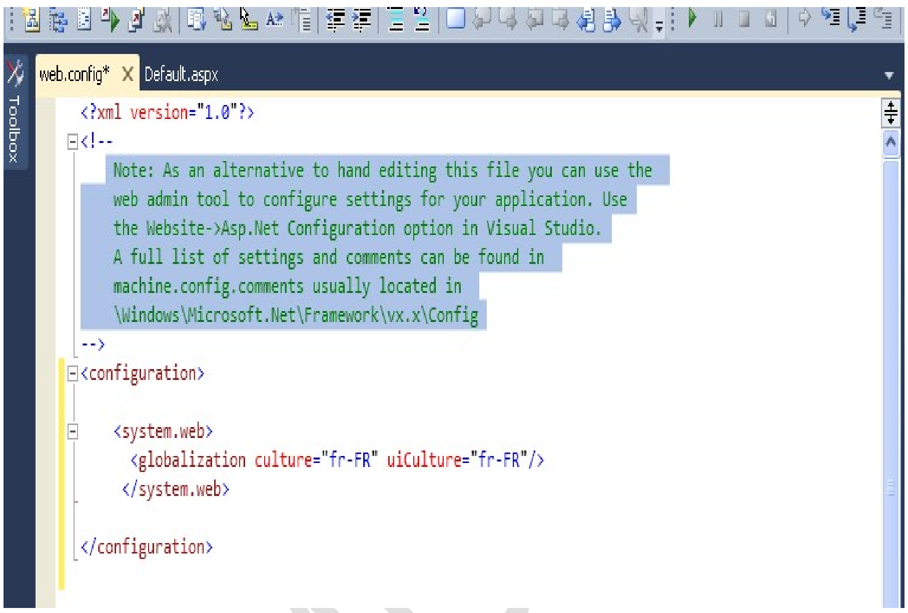
Cultures and Locales

The language needs to be associated with the particular region where it is spoken, and this is done by using locale (language + location). For example: fr is the code for French language. fr-FR means French language in France. So, fr specifies only the language whereas fr-FR is the locale. Similarly, fr-CA defines another locale implying French language and culture in Canada. If we use only fr, it implies a neutral culture (i.e., location neutral).

Set culture information

Application level -In web.config file

```
<configuration>
  <system.web>
    <globalization culture="fr-FR" uiCulture="fr-FR"/>
  </system.web>
</configuration>
```



```
<?xml version="1.0"?>
<!--
    Note: As an alternative to hand editing this file you can use the
    web admin tool to configure settings for your application. Use
    the Website->Asp.Net Configuration option in Visual Studio.
    A full list of settings and comments can be found in
    machine.config.comments usually located in
    \Windows\Microsoft.Net\Framework\vx.x\Config
-->
<configuration>

    <system.web>
      <globalization culture="fr-FR" uiCulture="fr-FR"/>
     </system.web>

</configuration>
```

Unit – 5

Advanced features using C#

Distributed Application in C#–

- Distributed applications are those which don't just run on a single computer but are divided up amongst several networked computers working together.
- To do this the application needs to be split into components which communicate with each other to achieve the overall purpose of the application. Each computer in the network runs one (or more) of these components which perform different tasks.
- **Examples** of distributed applications include industrial control systems, telecommunication systems and scientific applications which require significant computing power.
- However, WCF is concerned with a form of distributed computing known as 'service oriented' applications (SOA's) which provide services to other applications.

XML and C#–

- XML is short for eXtensible Markup Language. It is a very widely used format for exchanging data, mainly because it's easy readable for both humans and machines. If you have ever written a website in HTML, XML will look very familiar to you, as it's basically a stricter version of HTML. XML is made up of tags, attributes and values and looks something like this:

```
<users>
<user name="John Doe" age="42" />
<user name="Jane Doe" age="39" />
</users>
```

- As you can see, for a data format, this is actually pretty easy to read, and because it's such a widespread standard, almost every programming language has built-in functions or classes to deal with it. C# is definitely one of them, with an entire namespace, the System.Xml namespace, to deal with pretty much any aspect of XML.

Unsafe mode–

- Unsafe is a C# programming language keyword to denote a section of code that is not managed by the Common Language Runtime (CLR) of the .NET Framework, or unmanaged code. Unsafe is used in the declaration of a type or member or to specify a block code. When used to specify a method, the context of the entire method is unsafe.
- To maintain type safety and security, C# does not support pointer arithmetic, by default. However, using the unsafe keyword, you can define an unsafe context in which pointers can be used. For more information about pointers, see the topic Pointer types.
- Unsafe code can create issues with stability and security, due to its inherent complex syntax and potential for memory related errors, such as stack overflow, accessing and overwriting system memory. Extra developer care is paramount for averting potential errors or security risks.

Example

The following is an example of Unsafe Mode:

```
unsafe static void Main()
{
  fixed (char* value = "safe")
  {
    char* ptr = value;
    while (*ptr != '\0')
    {
      Console.WriteLine(*ptr);
      ++ptr;
    }
  }
}
```

Output

```
s
a
f
e
```

Graphical device interface with C#–

Graphics Device Interface + (GDI+) is a graphical subsystem of Windows that consists of an application programming interface (API) to display graphics and formatted text on both video display and printer.

GDI+ acts as an intermediate layer between applications and device drivers for rendering two-dimensional graphics, images and text.

The features included in GDI+ are:

- Gradient brushes used for filling shapes, paths and regions using linear and path gradient pushes
- Cardinal splines for creating larger curves formed out of individual curves
- Independent path objects for drawing a path multiple times
- A matrix object tool for transforming (rotating, translating, etc.) graphics
- Regions stored in world coordinates format, which allows them to undergo any transformation stored in a transformation matrix
- Alpha blending to specify the transparency of the fill color
- Multiple image formats (BMP, IMG, TIFF, etc.) supported by providing classes to load, save and manipulate them
- Sub-pixel anti-aliasing to render text with a smoother appearance on a liquid crystal display (LCD) screen.