BAOU
Education for All

Dr.Babasaheb Ambedkar Open University

(Established by Government of Gujarat)

MSCCS-201
Web Development Tools

Web Development Tools

Master of Science – Cyber Security (MSCCS)

2021

Web Development Tools

Dr. Babasaheb Ambedkar Open University

Web Development Tools

Course Writer

Mr. Dhaval K. Raval — Assistant Professor, Department of Computer Science, Ganpat University

Mr. Narendra L. Patel — Assistant Professor, Department of Computer Science, Ganpat University

Content Editor

Dr. Himanshu Patel — Assistant Professor, Dr. Babasaheb Ambedkar Open University

Content Reviewer

Prof. (Dr.) Nilesh K. Modi — Professor & Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University

ISBN:

Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad

Web Development Tools

Block-1: .NET architecture and Programming

Block-2: C# Control structure, Properties, Delegates & Exception Handling

Block-3: Inheritance, Interface and Generics

Block-4: Threading, File handling, C# controls

Block-1

.NET architecture and Programming

Unit 1: .NET Architecture 1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to understand:

- Basics of .Net Architecture
- How .Net application compile, run and execute.
- To code in C# programming language

1.2 INTRODUCTION

This topic will cover details about Components of .Net architecture. There will be a detail description on various .net framework topics like CLR, CTS, BCL, IL, Managed and Unmanaged Code and Assemblies.

1.3 COMPONENTS OF .NET ARCHITECTURE

.Net Framework is also known as MS .Net Framework as it is designed and Developed by Microsoft. It is the infrastructure for building, running and deploying applications and services. In 2002, the first version of .net framework was released. The .net framework acts like virtual machine which compile and execute programs written in different languages like VB.Net, F#, C# etc.Find the detailed Architecture of .net framework in Figure 1.1

Using .Net framework one can develop Console application, web services, Forms and Web-based application, mobile phone applications and many more.

.Net framework mainly contains two components:

i) CLR - Common Language Runtime.
ii) BCL - Base Class Library

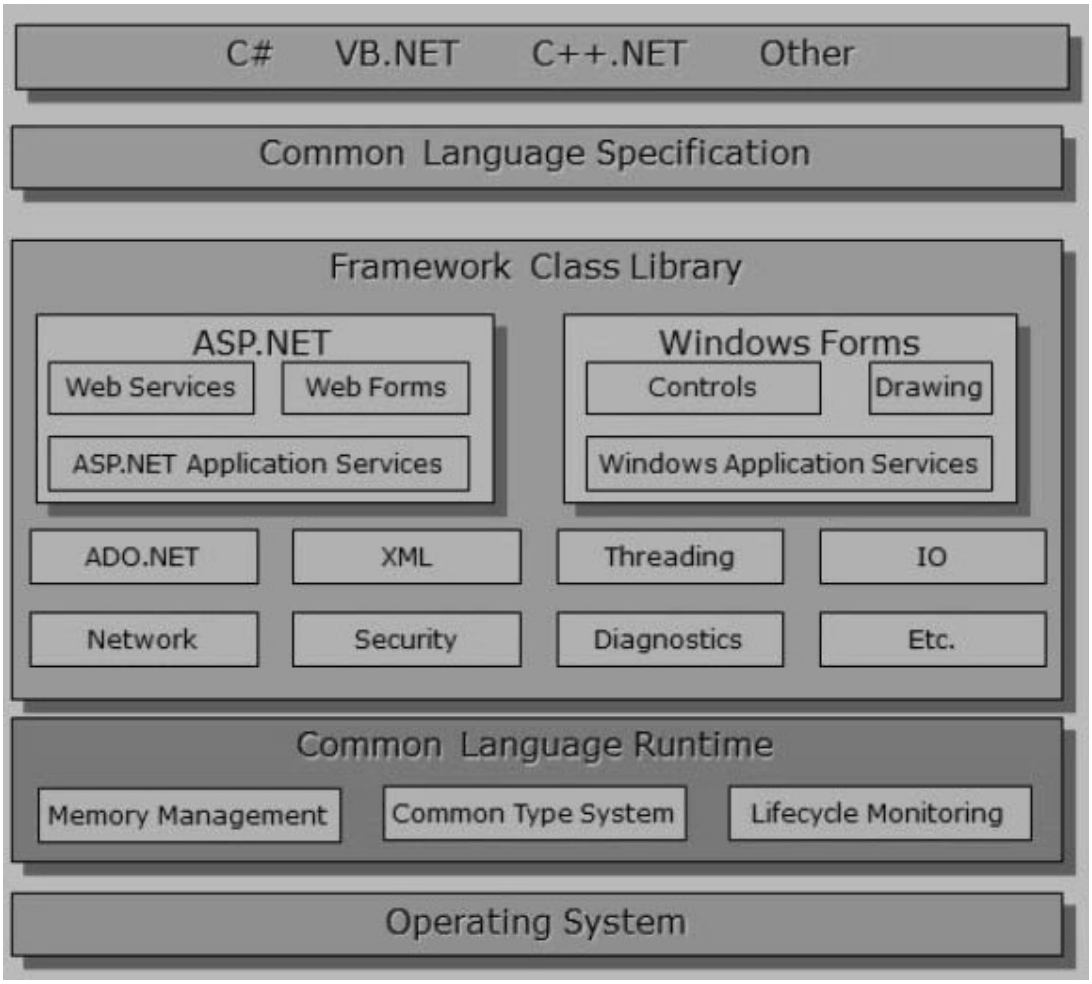


Figure 1.1: .Net Framework Architecture

Check Your Progress 1

1. List out the main components of .Net Framework.
2. List out the types of applications which can be developed using .net framework.

1.4 MS .NET RUNTIME

Common Language Runtime (CLR)

The run time environment provided by .Net Framework is known as Common Language Runtime (CLR). CLR provides an environment for running all types of .Net Programs and applications. The execution of all kinds of .net programs is managed

by CLR irrespective of their underlying .net programming language. So basically CLR provides memory management, thread management and other services needed to execute a .net program. Find the detail architecture of CLR in figure 1.2

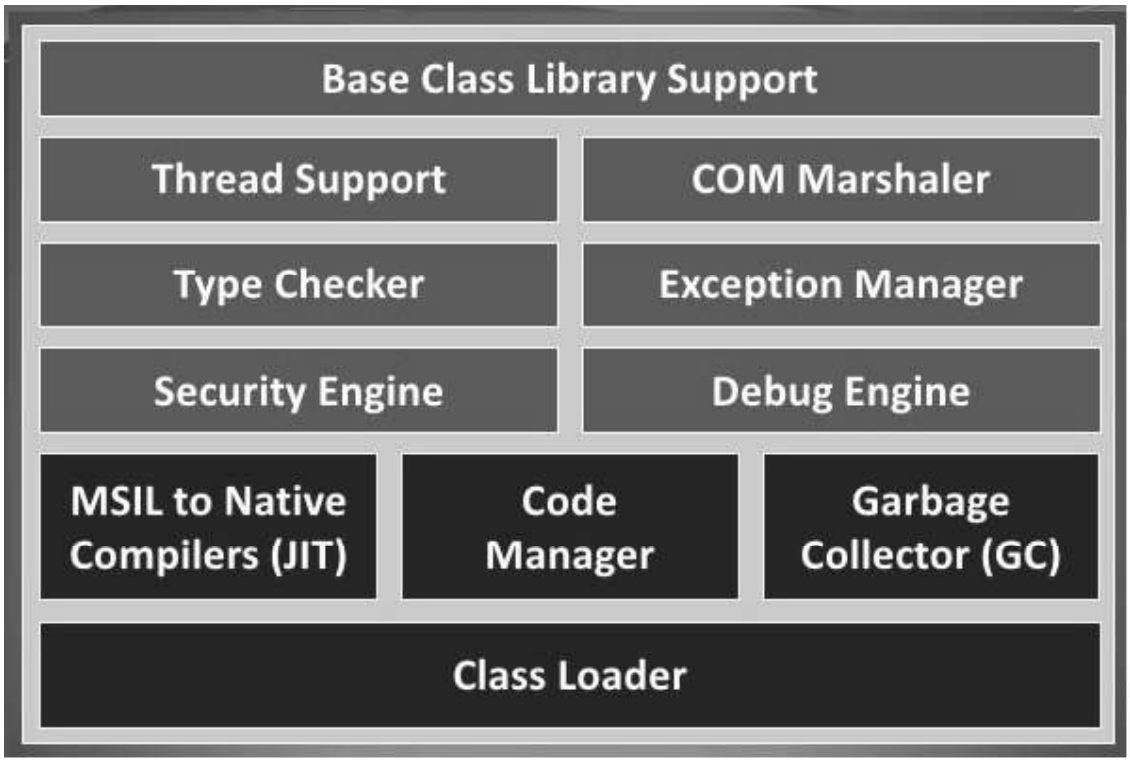


Figure 1.2: Common Language Runtime Architecture.

Suppose, any program written in C#, VB.net or any other .net programming language is compiled to Microsoft Intermediate language(MSIL) along with its metadata using specific compiler. The MSIL code is platform independent code. After successful generation of MSIL code CLR provides runtime environment and needed services to MSIL code. Internally CLR contains JIT (Just In Time) compiler which generates machine /native code from the MSIL code. The machine / native code further executed by CPU. Find the illustration of .net program execution in figure 1.3

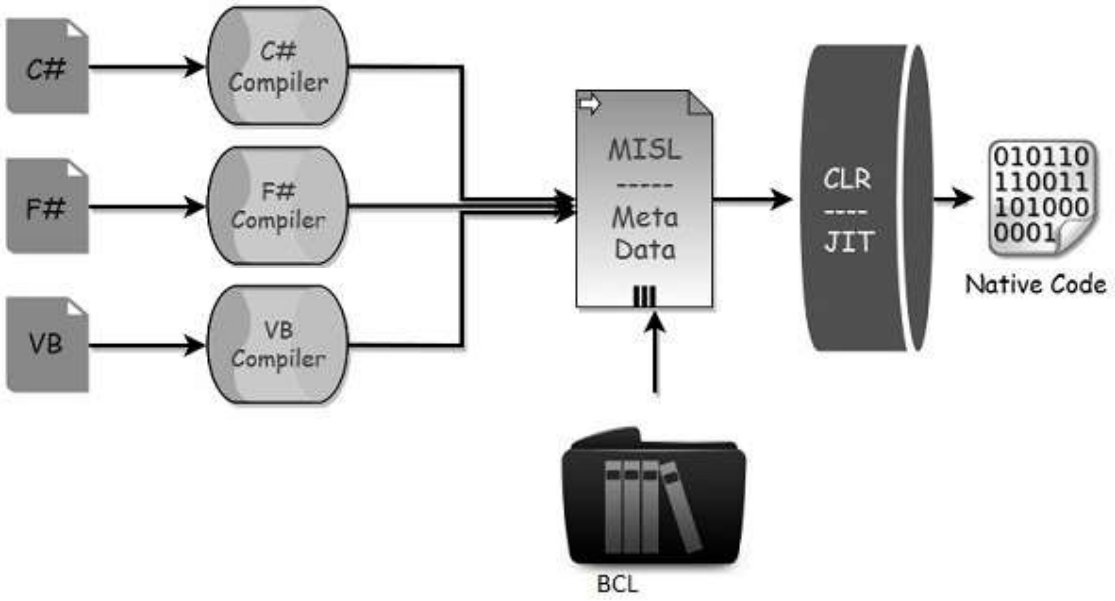


Figure: 1.3 .Net Program Execution.

Check your progress 2

1. Give Full Forms: CLR, MSIL
2. MSIL code is executed by _____ .

1.4 MANAGED / UNMANAGED CODE

Managed Code:

The code which is executed by managed runtime execution environment like CLR is known as Managed code. This code is executed by CLR and cannot be accessed from outside of the environment and also any direct call from outside the run environment is not allowed.

Unmanaged Code:

Code which is not developed in .Net framework and do not run under the control of CLR is known as unmanaged code. This types of code compiles directly to machine code and is executed by Operating System. This code is compiled to target a specific CPU architecture and will only run on the intended platform. So code written for specific architecture, cannot be run on different architecture. If you want to run

the same code on different architecture, then you have to recompile code for the particular architecture.

Code which is compiled by C or C++ compilers are known as Unmanaged code.

Check your progress 3

1. Managed Code can be accessed outside CLR (TRUE/FALSE)
2. Unmanaged Code does not execute by Operating System. (TRUE/FALSE).

1.5 INTERMEDIATE LANGUAGE (IL)

As it is developed by Microsoft it is also known as Microsoft Intermediate Language(MSIL) or Common Intermediate Language (CIL). Code written in different .net programming language is compiled by specific compiler to MSIL code.This MSIL code is a CPU-independent set of instructions whichwill be converted to the native code. At runtime the MSIL code is converted to native code by JIT (Just in Time) compiler of CLR.

Metadata is also generated while the MSIL code is generated by compiler.Metadata and MSIL are contained in a portable executable (PE) file. This MSIL code have instructions for storing, initializing, loading, and calling methods on objects, it also have instructions for logicaland arithmetic operations, direct memory access, control flow, exception handling, and other operations

Check your progress 4

1. MSIL code is platform independent (TRUE / FALSE)
2. At run time CLR is responsible for executing MSIL code (TRUE/FALSE)

1.6 COMMON TYPE SYSTEM (CTS)

The CTS - Common Type System is a standard for defining and using data types for any .NET framework program. CTSalso defines a collection of data types, which is used and managed by run-time to facilitate integration between different languages

CTS provides common types so that different .net programs, applications and controls written in different programming languages can share information easily. It also describes different sets of data type which can be used in different .Net languages in common. Because of that CTSconfirms that objects written in different .Net languages can interact with each other.

The common type system supports two categories of types:

Value types:

Value types contains the value or data directly; The instances of value types allocated on the stack or allocated in a structure. Value types can be user-defined, built-in or enumerations.

Reference types:

The reference types store the reference of value's memory address on the heap memory. It can be pointer types, self-describing type or interface type. The type of reference type is obtained from value of self-describing type. It is further split into arrays and class types.

Check your progress 5

1. Give Full Form: CTS
2. Different .Net Programs can share information easily because of CTS (TRUE/FALSE)

1.7 MS .NET BASE CLASS LIBRARY (BCL)

This is also called as Framework Class Library(FCL) and it is common for all types of applications.The way for accessing Library Classes and Methods in C#, VB.NET will be same and common for all other .net programming languages.

Following are different types of applications that can use .net class library.

- Console Application
- Windows Application.
- XML Web Services.

- WCF
- WPF
- Web Application

There are comprehensive set of framework classes,many of them are shown in figure 1.4.

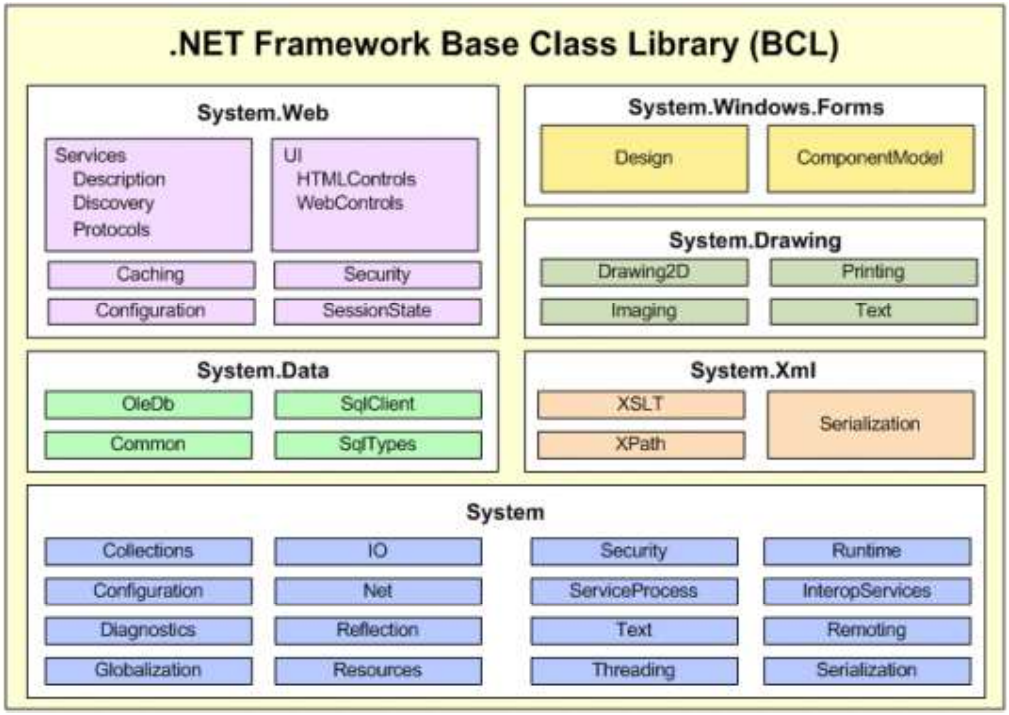


Figure1.4 .net framework Base class library

In short, any developers who want to develop any .net application can just import the BCL in their language code and use its methods and properties to implement common and complex methods like writing and reading file, database interaction, XML document manipulation and graphic rendering.

Check your progress 6

1. Give Full Form: BCL
2. The BCL is a standard library available to all language using the .net framework. (TRUE/FALSE).

1.8 ASSEMBLIES

Assemblies are basic building blocks of .NET Framework applications. Assemblies form the fundamental unit of deployment, reuse, activation scoping, version control, and security permissions. In short it is a compiled output of any program that is used for easy deployment of a program or application. They are executable files in the form of either dll or exe. It contains collection of resources which were used while building the application and it alsoaccountable for all the logical functioning. Refer figure 1.5 for Assembly file contents.

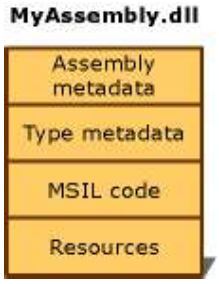


Figure 1.5 Assembly file

An assembly perform following functions:

- Forms security boundary.
- Ensures type safety by forming name scope for types at the runtime.
- It holds IL code that will be executed by common language runtime.
- An assembly is the unit at which permissions are requested and granted.
- It also permits side-by-side execution of multiple versions of same assembly.
- It also contains version information.

There are various types of Assemblies

Static and Dynamic Assemblies:

Static assemblies which include .NET Framework types (classes and interfaces), and resources for the assembly like bitmaps, JPEG files, resource files, and so on. These assemblies are stored on disk in PE (portable executable) files.

The Dynamic Assemblies directly run from memory and not saved to disk before execution. You can save dynamic assemblies to disk after they have executed.

Private and Shared Assemblies:

Private Assemblies are considered to be used by one application and must reside in that application's directory or subdirectory.Shared assemblies shared by multiple application at a time and it also ensure reusability. The shared assemblies stored in GAC (Global Assembly Cache)

Check your progress 7

1. List out the types of Assembly Based on its usages.
2. Private Assembly is used by multiple application at a time (TRUE/FALSE)
3. The Dynamic Assemblies directly run from memory and not saved to disk before execution. (TRUE/FALSE)

1.9 LET US SUM UP

In this Unit we have learnt about basics of .Net Framework. Now we are able to explain the importance of .Net framework and how it will be useful for any developer who are coming from different programming background. Any developer can develop .net application using their own choice of programming language; this application can also integrate with any other application which were not written in same programming language. We also learn that any programme which is written in Unmanaged code can also be integrate with managed code.

1.10 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check Your Progress 1

1. The main components of .net Framework are
    i. Common language runtime
    ii. Framework class library

2. Following are the types of applications which are developed using .net framework
    - Console application
    - Form based application
    - Web based application
    - Phone application
    - Web services and many more….

Check your progress 2

1. Give Full Forms:
    a. CLR: COMMON LANGUAGE RUNTIME
    b. MSIL: MICROSOFT INTERNEDIATE LANGUAGE
2. MSIL code is executed by CLR

Check your progress 3

1. FALSE
2. FALSE

Check your progress 4

1. TRUE
2. TRUE

Check your progress 5

1. Full Form:
   CTS: COMMON TYPE SYSTEM
2. TRUE

Check your progress 6

1. Give Full Form:
   BCL: BASE CLASS LIBRARY
2. TRUE

Check your progress 7

1. Types of Assembly Based on its usages:
   Private and Shared Assemblies are two assemblies categorized based on their usages.
2. FALSE
3. TRUE

1.11 FURTHER READING

- In depth detail on Components of .Net architecture can be refer from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/framework/
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

1.12 ASSIGNMENTS

1. Explain about main component of .Net architecture.
2. Explain with figure: Execution of .net application.
3. Differentiate Managed Code Vs Unmanaged Code.
4. Explain how the MSIL code is platform independent?
5. What is Assembly? Explain different types of assemblies.

1.13 ACTIVITIES

- Study about Assembly files of .net programs.

1.14 CASE STUDIES

- Study the different versions of .net framework.

Unit 2: Metadata and Modules 2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to understand:

- Metadata and Modules
- The importance of Just In Time Compiler and its working mechanism
- How .net framework manages memory.

2.2 INTRODUCTION

This topic will cover details about Components of Metadata and modules. There will be a detail description on working of Just in time compiler. There is also a detail discussion on Garbage Collection for memory management in .net application.

2.3 METADATA AND MODULES

In past the software components (.exe or .dll) written in one language and unable to use other programs which were written in another language. COM component has provided solutions to this problem. .Net framework has made interoperations of components easier by letting compilers to produce extra declarative information into assemblies and modules. This types of information known as metadata which helps different software components to interact seamlessly.

Metadata is like binary information of the program which is stored in a PE (Portable Executable) file in memory. PE file contains Metadata in one portion and in another portion MSIL. Types and members those are defined in a module or assembly are described in metadata. So when the code is being executed by the runtime, it loads metadata in memory and based on metadata it references needed members, classes, inheritance and so on.

Metadata Stores following information:

- Types Description: contains visibility, Name, Base Class, methods, properties, events and interfaces implemented
- Assembly Description: contains information of other assemblies on which this assembly depends, security permissions, identity and so on.

- Attributes: Additional elements which modify types and members.

Advantages of Metadata:

- Self-descriptions of files.One of the great advantage of metadata is that it allows your code to describe itself, thus removing the need for Interface Definition Language (IDL) and type libraries. Metadata contains everything that needed to interact with another module.
- Language Interoperability: Metadata contains all information required to inherit a class from a PE file written in different language. So the program can create an instance of any class irrespective of its based language without worrying about explicit marshaling.
- Attributes: You can declare specific kinds of metadata, known as attributes. Attributes are used to control how the program will behave at run time.

Check your progress 1

1. Metadata stored in _____ file.
2. List Advantages of Metadata.

2.4 JUST IN TIME COMPILATION

As discussed in Common Language Runtime,Just-In Time (JIT) compiler is a part of CLR. It is responsible for the execution of any .Net Program irrespective of underlying programming language. As we have learned in CLR that a language specific compiler compiles the source code into MSIL code. This MSIL code is converted to specific computer's environment native / machine code by JIT compiler.

The machine code from MSIL code is generated by JIT compiler based on the requirement meaning that JIT do not convert entire MSIL code into machine code at a time but it covert to machine code as and when needed.

There are three types of JIT compilers.

i) **Pre-JIT:** The Pre-JIT compiler compiles all the source code to machine code at same time in single compilation process. This compilation done at application deployment time.

ii) **Normal JIT:**The portion of source code or methods which are required at run time will be converted to machine code at the first time by Normal JIT. After that the compiled code stored into cache and used from cache whenever it called again.

iii) **Econo JIT:** The portion of source code or methods which are required at run time will be converted to machine code by Econo JIT. And this compiled code is removed from memory as it will not be required in future.

JIT Compiler Advantages:

- Less memory usages: as only those methods are compiled which are needed at a time.
- Reduced Page faults: Most probably the methods required are stored in same memory page.
- Code is optimized during run-time based on statistical analysis.

JIT compiler Disadvantages:

- JIT compiler takes more startup time during the first execution.
- Heavy usages of cache memory by JIT to store source code methods during execution.

Check your progress 2

1. JIT converts _____ code to _____
2. List the types of JIT.

2.5 GARBAGE COLLECTION

Garbage collector in .Net manages allocation and de-allocation of memory for the .net application. Each time CLR allocates memory to new object from the managed HEAP. The run time will allocate memory from the managed Heap till the address space is available.

As the memory is not infinite, garbage collector has to perform collection to make some memory free. The optimizing engine of the garbage collector fixes best time to perform a collection, based on the allocation made. Garbage Collector checks

objects that are no longer used by application in the managed heap memory and perform necessary actions to make the memory free.

Benefits of Garbage collection

- Optimization of memory usages.
- No need to write memory de-allocation code in application.
- Auto clean-up of memory from the objects that are no longer in use.

Check your progress 3

1. CLR allocates memory to new object from the _____ .
2. List benefits of Garbage collection.

2.6 LET US SUM UP

In this unit we have learned about Metadata and its important in .net program execution. We now able to understand types of JIT compiler and how it helps to convert MSIL code to native code. We also learn about Garbage collections and how it is helpful in .net program execution and memory management.

2.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your progress 1

1. Metadata stored in PE file.
2. Advantages of Metadata:
    - Self-descriptions of files
    - Language interoperability
    - Attributes

Check your progress 2

1. JIT converts MSIL code to native code.
2. Types of JIT:
    - Pre JIT
    - Normal JIT
    - Econo JIT

Check your progress 3

1. CLR allocates memory to new object from the managed Heap.
2. Benefits of Garbage collection:
    - Optimization of memory usages.
    - No need to write memory de-allocation code in application.
    - Auto clean-up of memory from the objects that are no longer in use.

2.8 FURTHER READING

- In depth detail refer from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/framework/
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

2.9 ASSIGNMENTS

1. What is Metadata? Explain briefly
2. Briefly explain Just-In-Time compiler.
3. What Garbage collection do? Explain its advantages.

2.10 ACTIVITIES

- Compare code compilation process of Java code with .net code.

2.11 CASE STUDIES

- Study the Memory management of other programming language e.g. JAVA

Unit 3: Introduction to C# .NET language

3

Unit Structure

3.1 LEARNING OBJECTIVES

After studying this unit students should be able to understand:

- Programming using C# language, its compilation and execution.
- Console application development.
- Working with Classes and Methods.
- Working with Libraries and Namespaces.
- Different types of Comments and Keywords.

3.2 INTRODUCTION

This unit covers basics of C# .net programming language, there will be a detail discussion on C# Comments, Classes, Methods, Libraries and Namespaces. This unit also covers steps to compile and execute any C# program.

3.3 INTRODUCTION TO C# .NET LANGUAGE

C# is an object oriented programming language. It is pronounced as *C-Sharp*. It is a type-safe object oriented language which allows developers to develop robust and secure application that runs on .net framework. Using C# one can develop windows client apps, Web services, client server apps, distributed components, database apps and many more.

C# is simple and easy to learn programming language. Any developer who is familiar to C, C++ or Java can easily learn programming in C#. C# has very simple syntax compares to C++ complex syntax and it also provides powerful
features like delegates, enumerations, direct memory access, and lambda expressions.

As C# is an object oriented programming language, it supports inheritance, polymorphism and encapsulation. In addition to this basics principle of OOP it also supports Delegates, Properties, Attributes and LINQ.

Build process of C# program is simple compared to C, C++ and Java as there is no particular order defined for declaration types and methods as well as there is no need to have separate headers files.

Check your Progress 1

1. C# supports oops concepts (TRUE/FALSE).
2. Other than OOP which additional features C# supports?

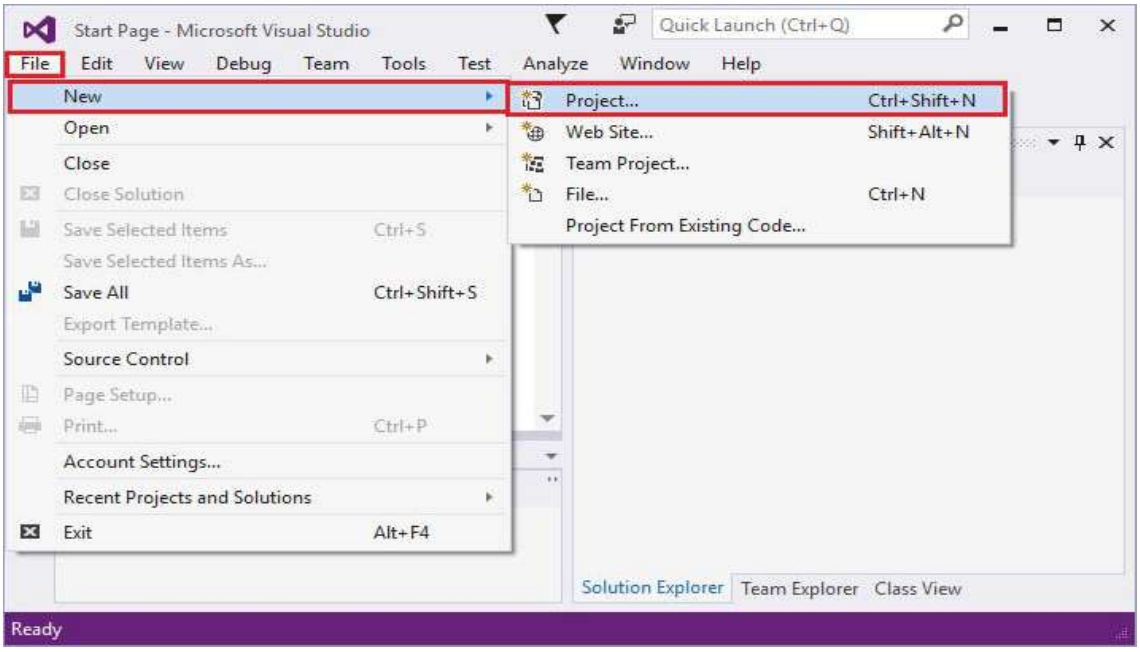
3.4 C# PROGRAM CONSOLE APPLICATION DEVELOPMENT

The Console application is a types of application that will run in windows command prompt. Generally,building first console application is ideal for beginner on .net.

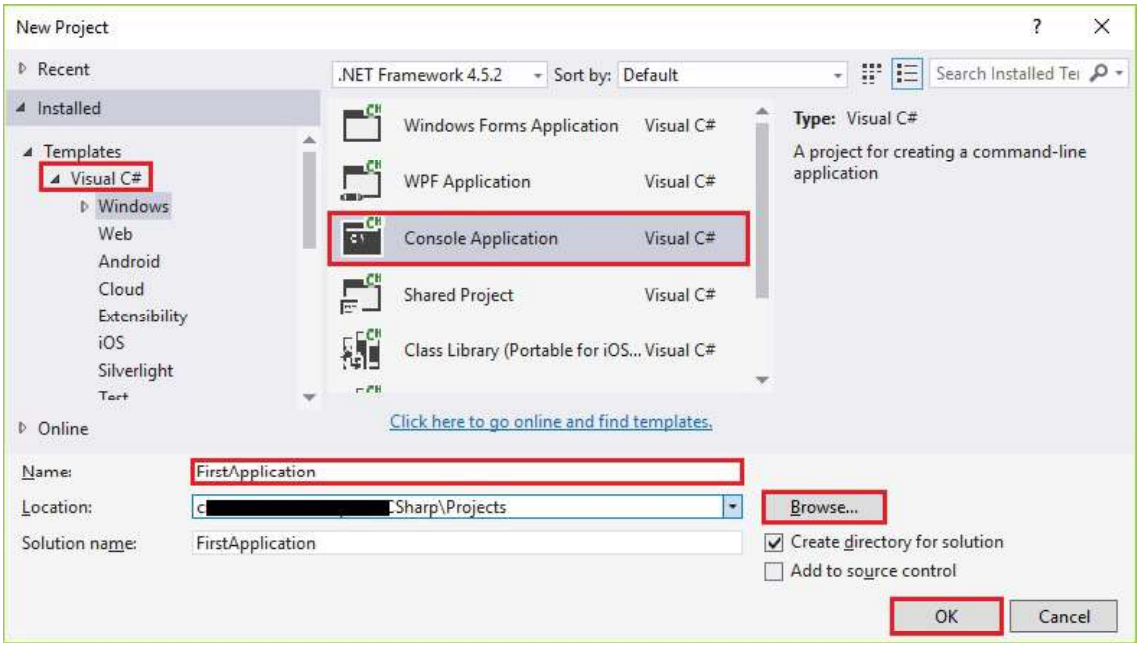
We will use Visual Studio to create console application. Later on we learn to develop the first console application “Hello World…!”

Follow the given steps for the first console application.

1. Open Visual studio(Install the visual studio if you have not installed it)
2. Open File Menu and Choose **New --> Project...** It will open the New Project dialog box as per below figure.

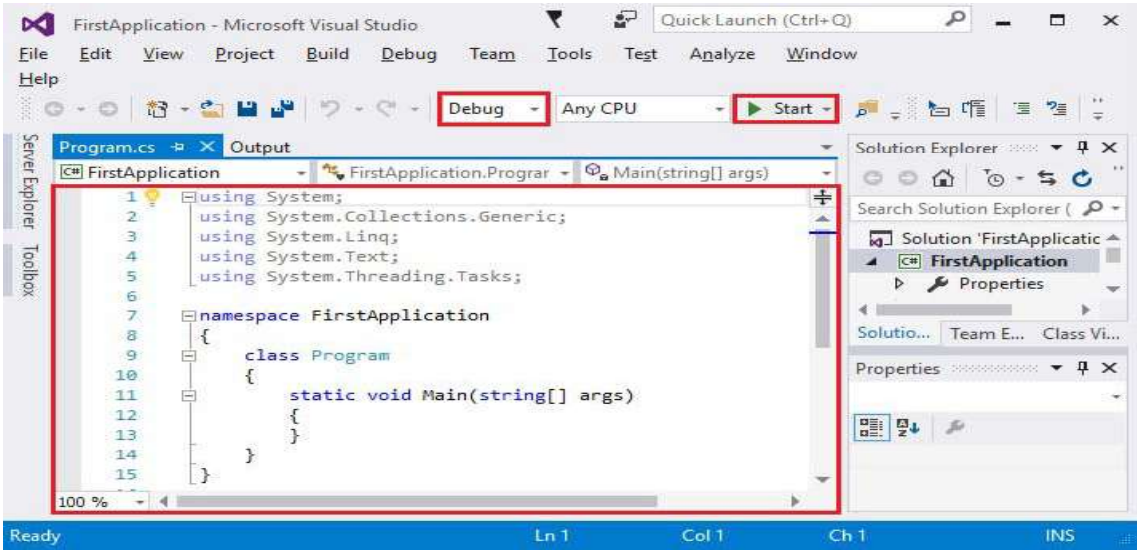


3. Now from left side menu expand **Installed, Templates, Visual C#** and select Windows then Choose Console Application. As per below figure.



4. Type appropriate Name for your project in the Name box then click on OK button it will create a new project in Solution Explorer.

5. If Program.cs is not open in Code Editor window, then open shortcut menu for program.cs fromm solution explorer and choose **view code.**



6. Write the following code in the Code Editor window for your first **"Hello World…!"** program

```
// A Hello World! program in C#.
using System;
namespaceFirstApplication
{
class Program
   {
static void Main()
      {
Console.WriteLine("Hello World…!");

         // Below code will keep the console window open till any key is pressed.
Console.WriteLine("Press any key to exit….");
Console.ReadKey();
      }
   }
}
```

7. Now you can run this project by pressing F5 key or from Debug menu.
8. A command prompt window will appear and it contains the line "**Hello World…!"**as per below figure.

Now let us understand the entire code line by line.

- **// A Hello World! program in C#.**
  The first line of the program starts with a comment with characters**: '//'**

- **using System;**
  to use Classes and methods from System namespace we have to use this namespace.

- **namespace FirstApplication**

  Your class will be placed in a namespace, by default it will take the name given in New Project dialog.

- **{ }**

  Curley brackets used to separate blocks of codes.

- **class Program**

  This line will create a class named Progrma.

- **static void Main()**

  Every console application must have Main method. It is the starting points of a program where the objects are created and other methods executed. The Main method is a static method. Here **void** is the return type, means this Main method returns nothing. There can be any return type like int, string, etc…

- **Console.WriteLine("Hello World…!");**

  Console is one of the class of .net framework run time library. It contains several methods for Input and Output operations. **WriteLine()** is the output methods of the Console class. It displays the string parameters on the standard output stream, Here **Hello World…!** will be displayed on the output.

- **Console.ReadKey();**

  As mentioned earlier that Console class contains methods for input and output; the **Readkey()** is a input method. This method will wait to read a key from user and thus it prevents the program from terminating instantly.

Check Your Progress 2

1. WriteLine() and ReadKey() are methods of _____ class.
2. Main() method can have only void as a return type (TRUE/ FALSE)

3.5 COMPILING AND EXECUTING

We see in previous section how to compile and run the console application using Visual Studio Integrated Development Environment (IDE).

Besides Visual studio IDE you can also compile and Execute the program from the command line. Follow below steps for the same.

1. Copy and Paste the code into any text editor from the previous procedure.
2. Save the text file as Program.cs. Here extension for C# source code is .cs
3. Set the environment variables for command line.
4. Open the command prompt window.
5. Navigate to the folder which contains Program.cs file in the command-prompt window.
6. Enter **csc Program.cs** command in command prompt to compile Program.cs program.
7. If the Program has no compilation error then an executable file named Protram.exe will be created.
8. Now enter the .exe file name **Program** in command prompt window to run the program.

Check you progress 3

1. _____command is used to compile any C# program?
2. _____ file is generated when you compile a C# file.

3.6 DEFINING A CLASS

class keyword is used to declare classes in C#.

The syntax to declare a class in C# is:

[access modifier] [class] [identifier]

{

}

Find Below an example to declare a class in C#.

```
public class MyFirstClass
{
        // properties, methods, fields and events declared here…
}
```

In the above example '*public'* is the access modifier which denotes that anyone can create instance of that class. The second word is *class* key word which used to declare a class in C#. The third is a class name which should be any valid C# identifier name. The body of the class contains between opening ({) and closing (}) curly brackets. Class contains *class members* like properties, methods, events etc…

Creating Objects of class

Object is an instance of a class. It is a concrete entity based on a class.

'new' keyword is used to create an object of the class. Refer below code to create an object

```
MyFirstClass Obj1 = new MyFirstClass();
MyFirstClass Obj2 = new MyFirstClass();
MyFirstClass Obj3 = new MyFirstClass();
```

When an instance of a class is created, its reference is passed back to the programmer. Here Obj1, Obj2 and Obj3 are reference to an objects that is based on *MyFirstClass.* These objects refer to new objects but they do not contain any data.

Check your progress 4

1. Write syntax to create a class in C#.
2. _____key word is used to create an object of a class.

3.7 DECLARING THE MAIN () METHOD

Every C# application contains at least one Main() method. It is the entry point for any C# application. The Main method is the first method which will be invoked when the application is started.

There is only one entry point for any C# application. If your program contains more than one Main method than you need to specify which Main method will be used as an entry point during compile time.

Find the declaration of Main method in below example.

```
class DemoClass
{
        static void Main (string[] args)
        {
                System.Console.WriteLine("No of Arguments: "+args.Length);
        }
}
```

Overview of Code:

- The Main method is always static, means this method can be called without any object.
- By default, access modifier is private.
- The Main method can have void or int return type.
- The Main method may also be declare with string[] parameter. This parameter contains command-line arguments.
- Above program will print number of arguments passed when it is executed.

Check your progress 5

1. _____ method is called first when C# programme is executed.
2. Main method can be non-static (TRUE/FALSE)

3.8 ORGANIZING LIBRARIES WITH NAMESPACES

Namespaces used to organize too many classes so that it will be easy to handle any C# application. It also helps to keep one set of classes separate from another. A programmer can create same named classes in different namespaces and no conflict arises due to same name of classes.

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

```
using System;
namespace DemoNamespace
{
        class Demo
        {
                static void Main()
                {
Console.WriteLine("DemoClass from DemoNamespace");
                }
        }
}
```

Overview:

- Namespace is used to organize large objects.
- The first line of above program shows the use of **System** namespace with '*using'* keyword
- The second line shows how to declare a namespace, here in above program '*DemoNamespace'* contains class named '*Demo'*.
- We have directly used WriteLine method of Console class without specifying '*System'* namespace (before Console.WriteLine()) as we have already used the System namespace in first line of code.
- Namespaces are delimited by the '**.**' (dot) operator.

Check your progress 6

1. Same named class can be created in different namespace (TRUE/FALSE)
2. To import a namespace in a C# program _____ keyword is used.

3.9 USING THE USING KEYWORD

There are three major usages of 'using' keyword.

- The *using statement* defines a scope: means at the end of the statement an object will be disposed.

  ```
  using (Font f1 = new Font("Times new Roman", 10.0f))
  {
        Byte charset = f1.GdiCharSet;
  }
  ```

  Here File and Font are examples of managed types that access unmanaged resources. All such types must implement IDisposable interface.

  The using statement calls the Dispose method on the object, and it also causes the object itself to go out of scope as soon as Dispose is called. The using statement also ensures that Dispose is called even if an exception raised.

- The *using directive*is used to create an alias for a namespace and also used to imports types defined in other namespaces.

  The using directive has three uses:

  - To allow the use of types in a namespace.

    e.g. using System.Text;

  - To allow you to access static members and nested types of a type without having to qualify the access with the type name.

    e.g. using static System.Math;

  - To create an alias for a namespace or a type.

  e.g. using Project1 = PC.MyOrganization.Project;

- The *using static directive* used to importmembers of a single class.

  The using static directive entitles a type whose static members and nested types can be accessed without specifying a type name.

The syntax is: using static <fully-qualified-type-name>;

e.g. using static System.Math;

Check you progress 7:

1. using statement is also used to dispose an object (TRUE / FALSE)
2. using directive is also used to create an alias of a namespace (TRUE/FALSE)

3.10 ADDING COMMENTS

Comments are used for detail explanation of code in any programming language. Compiler do not execute the commented code and ignore that line(s).

In C# there are three types of Comments.

1. Single line comments (//)
   This comment is used to comment a single line.

   Syntax: // This is single line comment.

2. Multiline Comments (/*…….*/)
   This comment is used to comment multiple lines. It is used by programmers if they want to comments a block of code in a program.

   Syntax:
   /* this is
   Multiline comment.
   And it comments out more than one line */

3. XML Documentation comments (///)
   It is the special types of comments which is used to create a documentation of code by using XML elements in a program.

   Syntax:
   /// <summary>
   /// Summary of the program goes here….
   ///</summary>

Example:

```
/// <summary>This program performs addition operation
/// on two variable values
///</summary>

static void Main()
{
        /* The value for variables
                Are pre-defined */
        int a=50;
        int b=30;
        int sum=0;

        // The following statement will perform Addition operation.
        sum= a+b;
        System.Console.WrieLine("The Answer is: {0}",sum);
}
```

Check you progress 8:

1. _____ characters used for a single line comment.
2. _____ characters are used for XML document comment.
3. The comment starts with characters **/*** and ends with ***/** is known as _____ comment.

3.11 LET US SUM UP

In this unit we learned about C# programming language. You learned to create a simple console application in C# programming with the use of visual studio as an IDE. We also learned the compilation and execution of a C# program without the use of visual studio. We have also learned about use of namespace, how to defining a class, Main method and different types of comments in this unit.

3.12 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

1. TRUE
2. Other than OOP C# supports: delegates, enumerations, direct memory access, lambda expressions and many more.

Check Your Progress 2

1. WriteLine() and ReadKey() are methods of **Console** class.
2. **FALSE**

Check you progress 3

1. **CSC** command is used to compile any C# program?
2. **.EXE** file is generated when you compile a C# file

Check your progress 4

1. Syntax to create a class in C#:

   [Access modifier] class [class name] { block of statements }
2. **new** key word is used to create an object of a class.

Check your progress 5

1. **Main()** method is called first when C# programme is executed.
2. Main method can be non-static: **FALSE**

Check your progress 6

1. **TRUE**
2. To import a namespace in a C# program **using** keyword is used.

Check you progress 7

1. **TRUE**
2. **TRUE**

Check you progress 8

1. __//__ characters used for a single line comment.
2. __///__ characters are used for XML document comment.
3. The comment starts with characters **/*** and ends with ***/** is known as **multiline** comment.

3.13 FURTHER READING

- In depth detail refer from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/csharp/
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

3.14 ASSIGNMENTS

1. What is C#?
2. Explain why there is a namespace in C#.
3. Explain in detail the Main() method of C#.

3.15 ACTIVITIES

1. Create C# console application program which will print your name, Date of Birth and city which are entered by user.
2. Create a C# console application which will print the numbers of arguments passed.

3.16 CASE STUDIES

- Compare Java, C and C++ with C# programming language to find the differences and similarities among them.

Unit 4: C# Data Types

4

Unit Structure

4.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand various Data Types used in C#. Student will also learn about Value types and Reference types data types.

4.2 INTRODUCTION

In this unit there will be a detail discussion on various Data Types used in C#.This unit also covers Value types and Reference types.

4.3 C# DATA TYPES

In any programming language a data type is something that tells its compiler the kind of value a variable hold. There are many in-built data types for different kinds of data in C#, e.g. number, String, float, etc.

Every data types can have a specific range of value. The following table represents different data types of C# with its size

Data Type	**Type**	**Size (bits)**	**Range (values)**
Byte	Unsigned integer	8	0 to 255
Sbyte	Signed integer	8	-128 to 127
Short	Signed integer	16	-32,768 to 32,767
ushort	Unsigned integer	16	0 to 65,535
int	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	Unsigned integer	32	0 to 4294967295
long	Signed integer	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	Unsigned integer	64	0 to 18,446,744,073,709,551,615
float	Single-precision floating point type	32	-3.402823e38 to 3.402823e38

double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
char	A single Unicode character	16	Unicode symbols used in text
bool	Logical Boolean type	8	True or False
object	Base type of all other types		
string	A sequence of characters		
DateTime	Represents date and time		0:00:00am 1/1/01 to 11:59:59pm 12/31/9999

Check your progress 1

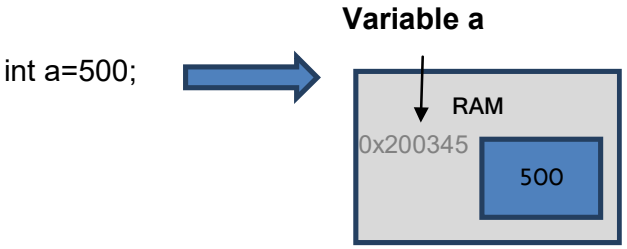
1. Define : Data Type

4.4 VALUE TYPES-PRIMITIVE DATA TYPES

There are two types of data types in C# based on the value store in memory. One is Value Type and another one is Reference Types.

The Value Type data type holds data value within its own memory space.
For Example, int a = 500;

In value type, system will store value 500 in the same memory space which allocated to variable ‘a’. Refer following image for the illustration.



Following are the example of Value types data types

- byte
- bool
- char
- double
- decimal
- enum
- float
- int
- sbute
- long
- short
- struct
- uint
- ulong
- ushort

Pass by Value:

When a value is passed from one method to another method, there will be a separate copy of a variable is created in another method. So change of value in one method does not affect the value stored in another method.
Example: Passing By Value:

```
Static void AlterValue(int i)
{
        j = 10;
        Console.WriteLine(j);
}

static void Main (string[] args)
{
        int k = 20;
```

```
        Console.WriteLine(k);
        AlterValue(k);
        Console.WriteLine(k);
}
```

OUTPUT:

20

10

20

In the above example value of variable k in Main() method will not change even after AlterValue() method call.

Check your progress 2

1. What is Value type data type?
2. Is 'enum' a value type data type? (YES/NO)

4.5 REFERENCE TYPES

Reference types does not store value directly in its own memory. But it stores the address where the value is actually stored. It means reference types variable stores the address of memory location where the actual value is stored. Refer below figure for more detail.

For example,
String s = "Hi, how are you?"

Reference type variable stores the Address where actual value is stored.

Actual Value

As you can see in the above figure that, system has selected a random memory location (0x802034) for variable s. The value of that variable is 0x500800 which is memory location of actual data.

Following are the example of Reference type data types:

- Arrays
- String
- Class
- Delegates

Pass by Reference

When a reference type variable passes from one method to another method, it will not create copy of it; but it passes address of that variable. If you change the value of that variable in a method than it will also be reflected in the calling method.

```
static void ChangeRef_Type(Student s2)
{
   s2.StudentName = "ABC";
}

static void Main(string[] args)
{
   Student s1 = new Student();
   s1.StudentName = "XYZ";

   ChangeRef_Type(s1);

Console.WriteLine(s1.StudentName);
}
```

OUTPUT:

ABC

Here in example, when we send the Student object s1 to ChangeRef_Type() method, we actually send the memory address of s1. So, when the ChangeRef_Type() method changes StudentName, it actually change the StudentName of s1, As s1 and s2 both points to the same address in memory.
Because of that the Output is ABC

Boxing and unboxing

Boxing is the process of converting a value type to a reference type data type. While unboxing is the process to convert reference type to value type data type.

Refer below code for boxing and unboxing in C#

```
int a = 10;

Object obj1 = a; //Boxing

a = (int) obj; //Unboxing
```

Check your progress 3

1. What is Reference type data type?
2. Conversion of a value type to reference type is known as _____ .

4.6 LET US SUM UP

In this unit we learned about Data Types of C#. Now we are in a position to distinguished value type and reference type data type. We also learn difference between Pass by value and Pass by reference. We have also learned about what is Boxing and Unboxing in C#.

4.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your progress 1

1. Data Type: Data type tells its compiler which the type of value stored in the variable.

Check your progress 2

1. Value type data type:

   The Value Type data type is a data type which holds data value within its own memory space. E.g. int a = 200;
2. Is ‘enum’ a value type data type? : YES

Check your progress 3

1. Reference type data type:

   A Reference types data type does not store value directly, but it stores the address where the value is actually stored.
2. Conversion of a value type to reference type is known as **Boxing**.

4.8 FURTHER READING

- In depth detail refer from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/csharp/
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

4.9 ASSIGNMENTS

1. Differentiate Value Type Vs Reference type data types.
2. Explain what is pass by value and pass by reference.

4.10 ACTIVITIES

- Create a C# console application to perform Boxing and Unboxing operations.

4.11 CASE STUDIES

- Learn about Value types and reference types of other programming languages like C, C++ or JAVA.

Block-2

C# Control structure, Properties, Delegates & Exception Handling

Unit 1: C# Control Structures

1

Unit Structure

1.1. LEARNING OBJECTIVES

After studying this unit student should be able to understand various control statements of C# programming language. Students will also be able to control the flow of a C# program using various conditional, looping and jumping statements.

1.2. INTRODUCTION

This unit covers basic control statements of C# programming language. Student will learn various conditional statements like if, if-else and switch cases. They will also learn looping and jumping statements like for, while, do while, break, continue, return and goto.

1.3. USING THE IF STATEMENT

The if statement evaluates Boolean expression. The block of statements inside if block will be executed if the output of the Boolean expression is *true*. If the Boolean expression output is *false*, then the statements inside if block will not be executed.

Syntax:

```
if (Boolean Expression)
{
        //statements
        //these statements will be executed if the condition is true
}
```

Example:

```
using System;
namespace ifDemo
{
        class if_demo
        {
                public static void Main(string[] args)
                {
```

```
            int a = 3;
            if (a< 10)
            {
            Console.WriteLine("The given number {0} is less than
            10", a);
            }
        Console.WriteLine("This statement is outside of if block and
        always executed.");
        }
    }
}
```

OUTPUT:

The given number 3 is less than 10

This statement is outside of if block and always executed.

Code explanation:

- The first line of code inside main method contains variable 'a' and its value is initialized to number 3.
- The second line contains conditional statement *if* to check the value of variable '*a*' is less than 10 or not. Here 3 is less than 10 so it will return *true* value
- As the if statement returns true the code from the next line will be executed and will print the message *"The given number 3 is less than 10".*
- The next line of code will always be executed and will print message "*This statement is outside of if block and always executed"* because this code is outside the if block.

Check your progress 1

1. The if statement evaluates _____ expression.
2. What values the Boolean expression will return?

1.4. USING THE IF-ELSE STATEMENT

Every if statement in C# have an optional *else*statement. The statements from the *else*block will be executed when the Boolean expression from if expression returns *false.*

Syntax:

```
if (Boolean Expression)
{
        //statements
        //these statements will be executed if the condition is true
}
else
{
        //these statements will be executed if the condition is false.
}
```

Example:

```
using System;
namespace if_else_Demo
{
        class if_else_demo
        {
                public static void Main(string[] args)
                {
                        int a = 13;
                        if (a< 10)
                        {
                        Console.WriteLine("The given number {0} is less than
                        10", a);
                        }
                        else
                        {
                        Console.WriteLine("The given number {0} is greater than
                        10", a);
```

```
                    }
                }
            }
        }
```

Code explanation:

- The first line of code inside main method contains variable ‘a’ and its value is initialized to number 13.
- The second line contains conditional statement *if* to check the value of variable ‘*a*’is less than 10 or not. Here variable value 13 which is greater than 10 so the if statement will return *false* value and the if block will be skipped.
- Here, statement from else block will be executed as the value returns by if statement is false, and the statement will print the message *“The given number 13 is greater than 10”.*

Check your progress 2

1. When the code from else block is executed?

1.5. USING THE SWITCH CASE STATEMENT

The Switch statement is one types of decision making statement like as if statement. It is a selection statement which selects a single switch case from a list of switch cases based on a pattern match. The switch statement is also used as an alternative to if…else if statement.

The switch expression can be of different type such as char, int,short,byte, string orenumeration type. The expression will be checked for different cases and the matched case will be executed.

Switch statement haveat least one switch sections with case labels‘*case:’*or ‘*default:’*and it is followed by one or more statements. The execution from one switch section to the next switch section is not allowed. To meet this requirement one has to

use jump statements like break, goto or return to manually exit from the switch section.

Syntax:

```
switch (expression) {

case value1: // sequence of statements
break;
case value2: // sequence of statements
break;
.
.
.
case valueN: // sequence of statements
break;
default:   // sequence of statements for default case
}
```

Consideration for Switch statement:

- Duplicate values for case isnot allowed.
- The data type of the variable in the switch and value of a case must be of the same type.
- The value of a case must be a literal or a constant. Variables are not allowed.
- The break in switch statement is used to terminate the current sequence.
- Only one default statementis allowed and it is optional.

Example:

```
using System;
public enum Shape {Triangle, Square, Circle}
public class Example
{
        public static void Main()
        {
```

```
        Shapes = (Shape) (new Random()).Next(0, 3);
        switch (s)
        {
        caseShape.Triangle:
        Console.WriteLine("The Shape is Trianlge");
        break;
        caseShape.Square:
        Console.WriteLine("The Shape is Square");
        break;
        caseShape.Circle:
        Console.WriteLine("The Shape is Circle");
        break;
        default:
        Console.WriteLine("The Shape is unknown.");
        break;
        }
        }
}
```

OUTPUT:

The Shape is Square

Code Explanation:

- In the above example Shape is the enum which contains three different shapes Triangle, Square and Circle.
- In the first line of main method ‘s’ is a type of Shape enum which will contain any one value of the Shape from the three Shapes value at a time.
- The next line of code is the switch statement it will compare the value of Shape ‘s’ with the given switch cases and execute the statements from the matching case.
- In this example every switch case contains a break statement which will break the execution and pass the control out of the switch block.

- This example also has a default case; which will be executed when no matching value found in the given cases.

Check your progress 3

1. Switch statement contains more than one default statement (TRUE/FALSE).
2. The execution from one switch section to another switch section is not allowed (TRUE/FALSE).

1.6. C# LOOPING CONTROLS AND JUMPING STATEMENT

1.6.1 Using the for statement

The for statement is used to executes block of statements repetitively till the Boolean expression of for loop evaluates to true. The break statement can be used to break out from the loop. You can also use continue statement within for block to step to the next iteration.

SYNTAX:

```
for (initialization variable; condition; steps)
{
    //Block of statements
}
```

The syntax contains three parts:

- Initialization variable: declaration and initialization of a variable which will be used in condition and steps expression.
- Condition: It is a boolean expression which will return true or false.
- Steps: It defines the increment or decrement part of for loop.

Example:

```
for (int i=0; i<10; i++)
{
    if(i==3)
```

```
        Continue;
        if(i==6)
        break;
        Console.WriteLine(Value of i: {0}, i);
    }
```

Output:

Value of i:0
Value of i:1
Value of i:2
Value of i:4
Value of i:5

Code Explanation:

In the given example the first part of for loop contains integer variable i and it is initialized to value 0. The second part contains boolean expression i<10; which means the for loop iterate to 10 times or till it is manually terminated by a break statement. The third part contains steps to increment the value of i by 1.

The block of code of for loop also contains a continue statement which will iterate the next iteration when the value of i reaches to 3. It also contains a break statement which will break the for loop when the value of i reaches to 6. In rest of the cases the last statement of for loop will be executed and will print the message on console as displayed in Output.

1.6.2 Using the while statement

The while statement is also a looping statement like a for loop but the only difference here is that it has only one part that is boolean expression. The while loop can also be executed zero times as the boolean expression is executed before the while loop's block.

You can also use break and continue statements in while loop.

Syntax:

```
while (boolean expression)
{
//blocks of statements
}
```

Example:

```
int j=0;
while (j<4)
{
        Console.WriteLine(j);
        j++;
}
```

OUTPUT:

```
O
1
2
3
```

Code Explanation:

Here the first line of code contains the initialization of integer variable j to value 0. The while loop in the second line contains boolean expression j<4 which will be evaluated for every iteration. The block of while loop contains two line of code; the first line will print the value of j as displayed in output and the second line will be used to increment the value of variable j by 1.

1.6.3 Using the do while statement

Statements of do while loop will be executed when a boolean expression evaluates to true. The do-while loop can be executed one

or more times as the boolean expression is evaluated after execution of the loop.

You can also use break and continue statements in do-while loop.

Syntax:

```
do
{
    //block of statements.
} while(boolean expression)
```

Example:

```
intj=5;
do
{
    Console.WriteLine(j);
    j - -;
}while(j>0);
```

OUTPUT:

5
4
3
2
1

Code Explanation:

In the first line of the above example a integer variable j is initialized to value 5. The next line contains do keyword which is an entry point for the do-while loop. The do-while block will always be executed at the first time as the boolean expression is evaluated after the do-while block. This example will print value of variable j as displayed in output

and the value will be decremented for each iteration by j-- code in each iteration. The do-while loop will be terminated when the value of variable j reaches to 0.

1.6.4 Using the break statement

The break statement used to terminate looping or switch statement.

Example:

```
for (int i=0; i<5; i++)
{
    if(i==3)
    break;
    Console.WriteLine(Value of i: {0}, i);
}
Console.WriteLine("This statement is out of for loop")
```

OUTPUT:

```
Value of i:0
Value of i:1
Value of i:2
This statement is out of for loop
```

In the above example the break statement will break the execution of for loop and executes the next line after the for loop when the value of variable i reaches to 3.

1.6.5 Using the continue statement

The continue statement is used to pass control to the next iteration in looping statement.

Example:

```
for(int i=5; i<=5; i++)
{
    if (i<4)
    {
            continue;
    }
Console.WriteLine(i)
}
```

OUTPUT:

4

5

Code Explanation:

In the above example the iteration of for loop will be skipped through continue statement till the value of variable i remains less than 4. The program will print value of i as displayed in output when its value became greater than or equal to 4.

1.6.6 Using the return statement

The return statement is used to terminates the execution of the method in which it appears and returns control to the calling method. If the type of method is void, then the return statement will be omitted.

Example:

```
class ReturnDemo
{
    static int squareArea(int a)
    {
            int area = a*a;
            return area;
    }
```

```
    static void Main()
    {
        int length_of_side = 5;
        int result = squareArea(length_of_side);
        Console.WriteLine("The Area of Square is {0}", result);
        Consloe.ReadKey();
    }
}
```

Code Explanation:

In the example there is a ReturnDemo class which contains Main() and a squareArea() methods. Notice the *return* keyword in squareArea() method which is used to return integer value from the squareArea() method. The squareArea method will be called from the Main method and the value returned by squareArea method will be stored in result variable.

1.6.7 Using the goto statement

The goto statement is used to transfer the program control directly to a labeled statement. It is also useful to get out from the deeply nested loops.

Example:

```
class GotoDemo
{
static void Main(string[] args)
      {
string yourname;
L1:  // creating label
Console.WriteLine("Enter your name: ");
yourname = Console.ReadLine();
Console.WriteLine("Welcome {0}", yourname);
```

```
Console.WriteLine("Ctrl + C to Exit\n");
gotoL1; //jump to L1 statement
    }
}
```

Code Explanation:

In the above example L1: is the label and the last line of code is a goto statement, the last line contains goto L1, means the program control will be passed to label L1 in the program thus the program will never be terminated as the goto statement transfer the control to lable L1 every time. So to terminate the program one has to press Ctrl+C key to exit.

Check your progress 4

1. List out the looping statements of C# language.
2. The do-while loop can be executed at least one time (TRUE/FALSE).
3. Execution of a method can be terminated using return statement (TRUE/FALSE).
4. _____statementused to terminate looping or switch statement.
5. _____ statement is used to pass control to the next iteration in looping statement.
6. _____statementis used to transfer the program control directly to a labelled statement.

1.7. LET US SUM UP

In this unit we learned C# control, looping and jumping statements

1.8. CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your progress 1

1. The if statement evaluates**Boolean** expression.
2. The Boolean expression will return true or false.

Check your progress 2

1. The statements from the *else* block will be executed when the Boolean expression from if expression returns *false.*

Check your progress 3

1. False
2. True

Check your progress 4

1. List out the looping statements:**While, do-while and for loop.**
2. The do-while loop can be executed at least one time: **TRUE**
3. Execution of a method can be terminated using return statement: **TRUE**
4. **break**statement used to terminate looping or switch statement.
5. **continue** statement is used to pass control to the next iteration in looping statement.
6. **goto** statement is used to transfer the program control directly to a labelled statement.

1.9. FURTHER READING

- In depth detail can be referred from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/csharp/
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

1.10.ASSIGNMENTS

1. Differentiate Switch case Vs if-else if statements
2. Compare various looping statements of C#.
3. State the usages of C# jumping statements.

1.11. ACTIVITIES

1. Create a C# Console application to find the number given by user is ODD or EVEN.
2. Create a C# Console application to Check the number given by user is Palindrome or not.
3. Create a C# Console application to display sum of digits of given number.
4. Create a C# Console application to find the given number is prime number or not prime.
5. Create a C# Console application to print Month according to the number entered by user (e.g. if user enter 3 then the program should display March). The program should also print message like “Entered number is invalid” if user enter any number greater than 12 or less than 1.

1.12. CASE STUDIES

- Compares Conditional, looping and jumping statements of C# programming language with other OOP language like Java, C++, etc.

Unit 2: C# Properties 2

Unit Structure

2.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand concepts of property of C# programming language. They will also learn how to work with value of private variables of any C# program.

2.2 INTRODUCTION

In this unit there will be a detail discussion on one of the C# concept named ‘property’. With the use of special methods of properties we can get or set values in the local private variables.

2.3 USING PROPERTIES- GET ACCESSOR, SET ACCESSOR

Property is a member of a C# class which provides a mechanism to write, read or compute the value of a private field. Properties can be used just like as a public data member, but actually they are special methods named accessors. It allows data to be accessed easily and helps to support the flexibilityand safetyof methods.

Properties enable a class to expose a public way of getting and setting values, while hiding implementation or verification code.

GET and SET Accessors:

Generally, in a class, we declare a data field as private and provide a set of public SET and GET methods to access the data fields. It is a good programming practice to use properties as we can prevent direct access to data fields from outside the class.

A **get** property accessor is used to return the property value, and a **set** property accessor is used to assign a new value.

Syntax:

```
<acces_modifier><return_type><property_name>
{
get
  {
  }
set
  {
  }
}
```

Example:

```
class PropertyDeclare
{
    private int a;
    public int A
    {
            get
            {
            return a;
            }
            set
            {
            a=value;
            }
    }
}
class UseProperty
{
    public static void Main()
    {
            PropertyDeclare d = new PropertyDeclare();
            d.A = 10; //set accessor will be called to assign value to the property
            int x = d.A; // get accessor will be called to get the value from property
```

```
            Console.WriteLine(x);//it will display: 10
        }
    }
```

Code Explanation:

Here in above example a class named PropertyDeclare holds private integer variable ‘a’.

The next line contains declaration of property named A which have integer return type. The property A contains get and set accessors just to assign and retrieve value from private integer variable ‘a’.

Next we have another class named UseProperty, in which we have created an object ‘d’ of PropertyDeclare class.

The next line (d.A=10) is used to assign value to the private integer variable ‘a’ of PropertyDeclare class using property ‘A’. You can notice here that the value is written at the right side of the ‘=’ sign and on the left side we have used property name with its object name. This line will call the set accessor of the property A.

The next line (int x=d.A) is used to get the value from the private variable ‘a’ of PropertyDeclare class using property ‘A’. You can notice here that the property name is written at right side of the ‘=’ sign and on the left side there is a variable name. This will call the get accessor of the property A.

Finally, the next line of code will print the value of the variable ‘x’ on the console.

So from the above example it is clear that we can avoid direct access to data field of a class by declaring them as private, and we can also get and set the value to that data field by using property whose scope is public.

Auto implemented properties

With the use of auto-implemented properties, you can simplify your code. The C# compiler will obviously provide the backing field for you.

To define an auto-implemented property we have to use only the get and set keywords without providing any implementation in property declaration. Refer below example for auto implemented properties.

Example:

```
public class StoreProduct
{
    public string pro_name
    { get; set;}
    public int pro_price
    { get; set;}
}
class DisplayProduct
{
    static void Main(string[] args)
    {
            var item = new StoreProduct{ pro_name = "T-shirt", pro_price = 499};
            Console.WriteLine(${item.pro_name}: price is {item.pro_price}");
    }
}
```

Output:

T-shirt: price is 499

Here in above example we have not declare any private variable to store product name or product price, we have created only public properties named pro_name and pro_price so a C# compiler will automatically create backing fields for both the properties. Also not that we have not implemented any logic for any get or set accessors for both the properties.

The get and set accessors will automatically store and retrieve value to the back fields whenever they are being called by the program. In the above example we have an *item* variable which will be initialized by object of StoreProduct type and will holds value for both the properties pro_name and pro_price. The last line of code will print the product name and price as displayed in the output.

Check your progress:

1. _____and_____ are the special methods (accessors methods) of property.
2. We can property without declaring any private variable and without implementing get and set accessors (TRUE/FALSE)

2.4 LET US SUM UP

In this unit we have learned about one of the C# programming concept property, we learn how to get and set values to private data fields of C# program. We also learn how to restrict values to certain values by using property concept.

2.5 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. **get** and **set** are the special methods (accessors methods) of property.
2. **TRUE.**

2.6 FURTHER READING

- In depth detail can be referred from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/csharp/
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

2.7 ASSIGNMENTS

- Write Code for C# console application in which we can only store the positive values between 10 and 50 in the private integer variable using property.

2.8 ACTIVITIES

- Create a C# console application to store and retrieve your personal details like Name, Date of Birth and City using properties.

2.9 CASE STUDIES

- Compare the concepts of property with other programming languages.

Unit 3: Delegates in C# 3

Unit Structure

3.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand how to use delegate and where to use delegate in C# programming. They will also learn about single cast and multicast delegates.

3.2 INTRODUCTION

This unit covers detail discussion Delegates and its usages. Delegate is one of the type in C# programming which is used to indicate reference to method of specific return type with specific parameter list. Generally delegates used in event driven programming, just like if you want to call a specific method from list of similar methods based on some action then you need use delegate.

3.3 DELEGATES IN C#: SINGLE CAST DELEGATES, MULTICAST DELEGATES

Delegate in C# is like as pointer to functions in C/C++. It is a reference type variable which holds reference to a method. The reference can also be changed at runtime.

Delegate is a type which denotes references to methods with specific return type and parameter list. The delegate instance will be associated with a method having compatible signature and return type when the delegate is instantiated. The method can be invoked through the delegate instance.

Delegates are used to implement events and call-back methods.

Delegates Declaration:

The Declaration of delegates determines which method will be referenced by it. It can refer to a method which has the same signature as that of delegate. i.e. if we have a method which takes two integers as a parameter and another method that takes a single string parameters, then we need to have two separate delegate type for each method.

For example, following delegate used to reference any method which has a single integer parameter and returns a string type variable.

public delegate string DelegateFirst (int a);

Syntax:

Delegate <return type><name of delegate><List of parameters>

Once you declare a delegate you have to create an object of delegate with new keyword and associate it with a particular method.

There are two types of Delegates:

1. Single Cast Delegate
2. Multi Cast Delegate

Single Cast Delegate:

Single cast delegate refers only to a single method at a time.

Refer the following example for declaration, instantiation and use of a single cast delegate.

Single Cast Example:

```
public class DelegateTest
{
        // Delegates declaration without any parameters and return type.
        public delegate void DemoDelegate();
        public void First_Method ()
        {
Console.WriteLine("First_Method Called…");
        }
        public void Second_Method()
        {
Console.WriteLine("Second_Method Called…");
        }
        public void Third_Method()
        {
```

```
        Console.WriteLine("Third_Method Called…");
        }
    }

class Program
{
        static void Main(string[] args)
        {
         DelegateTest test1 = new DeletateTest();

        // Instantiation
        DeletateTest.DemoDelegate M1 = new DeletateTest.DemoDelegate
        (test1.First_method);
        DeletateTest.DemoDelegate M2 = new DeletateTest.DemoDelegate
        (test1.Second_method);
        DeletateTest.DemoDelegate M3 = new DeletateTest.DemoDelegate
        (test1.Third_method);

        //Invocation
M1();
M2();
M3();
Console.ReadKey();
      }
}
```

Output:

First_Method Called…
Second_Method Called…
Third_Method Called…

Code Explanation:

- The delegate is created with following line of code.
  public delegate void DemoDelegate();

- There are three methods in the example.
  public void First_Method();
  public void Second_Method();
  public void Third_Method();

- *Objects of delegates will be created in the main function.*
  DeletateTest.DemoDelegate M1 = new DeletateTest.DemoDelegate (test1.First_method);
  DeletateTest.DemoDelegate M2 = new DeletateTest.DemoDelegate (test1.Second_method);
  DeletateTest.DemoDelegate M3 = new DeletateTest.DemoDelegate (test1.Third_method);
- *At last delegates will be called to execute the methods.*
  M1();
  M2();
  M3();

Multi cast Delegate:

Multi cast delegate is an extension of single cast delegates and it can refer to multiple methods at a time. In multicast, delegates are combined and a whole list of methods will be called. For adding methods to delegates '+' or '+=' operator is used and for removing methods '-' or '-=' operator is used.

Refers following example for multicast delegates.

Multi cast example:

```
namespace MulticastDele
{
class MultiDele
   {
public delegate void DisplayMessage(string s);
public void FirstMessage(string msg)
      {
Console.WriteLine("The First Message is : {0}", msg);
      }
```

```
public void SecondMessage(string msg)
      {
Console.WriteLine("The Second Message is : {0}", msg);
      }
public void ThirdMessage(string msg)
      {
Console.WriteLine("The Third Message is : {0}", msg);
      }
   }
class Program
   {
static void Main(string[] args)
      {
         MultiDele td = new MultiDele();
         MultiDele.DisplayMessage msg = null;
msg += new MultiDele.DisplayMessage(td.FirstMessage);
msg += new MultiDele.DisplayMessage(td.SecondMessage);
msg += new MultiDele.DisplayMessage(td.ThirdMessage);
msg("This is Multicast Delegates");

Console.ReadKey();
      }
   }
}
```

OUTPUT:

```
The First Message is : This is Multicast Delegates
The Second Message is : This is Multicast Delegates
The Third Message is : This is Multicast Delegates
```

Code Explanation:

- The delegate is created with following line of code.
  `public delegate void DisplayMessage(string s);`

- There are three methods in the example.
  public void FirstMessage(string msg)
  public void SeondMessage(string msg)
  public void ThirdMessage(string msg)

- In the main method object of delegate will be created by following line of code
  MultiDele.DisplayMessage msg = null;
- Now all above three methods are multicast to delegates object by following line of code
  msg += new MultiDele.DisplayMessage(td.FirstMessage);
  msg += new MultiDele.DisplayMessage(td.SecondMessage);
  msg += new MultiDele.DisplayMessage(td.ThirdMessage);
- Now delegate object msg will be called by passing string parameter using following line of code and it will call all the three methods in given sequence.
  msg("This is Multicast Delegates");
- Finally all the three methods will be called and prints the message as displayed in output.

Check your progress:

1. The return type, types of parameter and no. of parameters of a delegate must be identical to the referenced method by the delegate. (TRUE/FALSE)
2. The delegate which refers to a single method at a time is known as_____.
3. The delegate which refers to multiple methods at a time is known as_____.
4. In multicast delegates ____ or _____ operator is used to add methods to delegates.

3.4 LET US SUM UP

In this unit we have learned about one of the C# programming concept delegate, we learn how to use delegate to pass method as a reference. We also learn how to call multiple methods at a time using multicast delegate.

3.5 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. The return type, types of parameter and no. of parameters of a delegate must be identical to the referenced method by the delegate. (TRUE/FALSE): TRUE
2. The delegate which refers to a single method at a time is known as Single Cast delegate.
3. The delegate which refers to multiple methods at a time is known as Multi cast delegate.
4. In multicast delegates __+__ or _+=_ operator is used to add methods to delegates.

3.6 FURTHER READING

- In depth detail can be referred from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/csharp/\
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

3.7 ASSIGNMENTS

- Briefly explain why do we need delegate?

3.8 ACTIVITIES

- Create a C# console application to demonstrate concept of delegate in which a delegate is used to reference a method which will perform Addition/ Subtraction of two integer number and returns an integer value.

3.9 CASE STUDIES

- Find some real life example to compare concept of delegate with it.

Unit 4: Exception Handling in C# 4

Unit Structure

4.1 LEARNING OBJECTIVES

After studying this unit student should be able to understand Concepts of Exception handling in C# programming. With the use of Exception handling students will be able to handle any exceptional situations that can cause run-time error, thus avoids unexpected programme termination.

4.2 INTRODUCTION: EXCEPTION HANDLING IN C#

An exception is a problem that arises during the execution of a program means that exceptions are unforeseen errors occurs at run-time of a program. For example, some run time errors like file I/O error, running out of system memory, a database error, divide by zero etc. Such errors can cause unexpected program termination. The techniques to handle such error when they occur is known as exception handling.

When exception occur, it throws an object derived from the System.Exception class and it will be handled by try/catch block of exception handling. The System.Exception class have many methods and properties to obtain information about what went wrong.
It has a message property which provides information on what error occur. We can also obtain information like where the problem occurs through stacktrace property.

Following are various predefined exception classes derived from the System.SystemException class.

- IOException : To handle I/O Error
- IndexOutOfRangeException: To handle errors generated when a method refers to an array index out of range.
- ArrayTypeMismatchException: To handle when type is mismatched with the array type.
- NullReferenceException: To handle errors generated from referencing a null object.

- DivideByZeroException: To handle errors generated from dividing a dividend with zero.
- OutOfMemoryException: To handle errors generated from insufficient free memory.

4.3 USING THE TRY/CATCH AND FINALLY BLOCK

try/catch Block

We use ***try*** block to partition code which may raise some exception during execution of program. There is an associated ***catch***block which is used to handle any resulting exception. The try blocks without an associated catch or finally block will cause compiler error.

The catch block specifies the types of exception to catch. Sometime a try block is followed by multiple catch block, which may use for different exception filters. The multiple catch blocks are evaluated in top to bottom approach, and only one catch block will be executed for each exception. The first catch block generally specifies exact types of thrown exception. If no catch block has matching exception filter, then a catch block that does not have any filter is selected.

Finally Block

The finally block placed after try or catch block. It will always be executed irrespective of the exception is thrown or not. This block generally used for cleaning-up of code. e.g. for disposing an unmanaged object, closing database connections, etc…

SYNTAX:

```
try
{
   // Code which can cause run time exception.
}
catch (SomeSpecificException ex)
{
```

```
   // Code to handle the exception.
}
finally
{
//Finallly block code goes here.
}
```

Example:

a) C# code without try/catch Block

```
using system;
public class DemoException
{
public static void Main( string[] args)
{
    int x=5;
    int y=0;
    int z=x/y;
    Console.WriteLine("Code after arithmetic operations…");
}
}
```

OUTPUT:

Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.

b) C# Code with try/catch and finally block

```
using system;
public class DemoException
{
    public static void Main( string[] args)
    {
    try
    {
            int x=5;
```

```
            int y=0;
            int z=x/y;
        }
        Catch (Exception e)
        {
            Console.WriteLine("Inside Catch block: {0} Exception thrown."
        e.Message);
        }
        finally
        {
            Console.WriteLine("This code is from Finally block");
        }
        }
    }
```

OUTPUT:

Inside catch block: Attempt to divide by zero Exception thrown.

This code is from Finally block

Code Explanation:

- In example (a) there are two integer variables 'x' and 'y' initialized to 5 and 0 respectively, In next line of code there is a integer variable 'z' which will stores the result of division operation performed on variable x and y.
- The program will terminate unexpectedly with error "*Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.*" as in the division operation the denominator (variable 'y') contains value zero. And also this type of run-time error cannot be determined at compile time.
- The code given in example (b) provides solution for the error occurred in example(a) using exception handling with try/catch block.
- Here in example (b) the try block contains code which is prone to generate any run time error (or exception). The try block is followed by catch block which is used to catch any exception thrown by the try block.

- The next block is catch block and the code inside catch block will only be executed when try block throws an exception. Here in example, *DivideByZeroException*will be thrown by the try block and the catch block will catch it and will print the message *"Inside catch block: Attempt to divide by zero Exception thrown."*
- There is a finally block in the example, which will always be executed irrespective of any exception thrown. So for every execution it will print the message *"This code is from Finally block".*

Check your progress 1

1. The catch block will only be executed when code from try block throws an exception. (TRUE/FALSE).
2. Finally block will not be executed if try block does not throw any exception. (TRUE/FALSE)

4.4 USING THE THROW STATEMENT

In the previous section we seen that how to handle exceptions which is raised by CLR. In this section we will learn how to raise user defined exception manually using throw statement. Any exception derived from Exception class can be raised using the throw keyword. This kind of exception handling is generally known as custom exception handling.

EXAMPLE:

```
class Program
{
        static void Main(string[] args)
        {
                int OrderQty;
                Console.WriteLine("Enter number of Notebook you want to buy (Total
                20 in Stock):");
                OrderQty = Convert.ToInt32(Console.ReadLine());
                try
                {
                        if (OrderQty == 20 || OrderQty < 20)
                        {
                        Console.WriteLine("Congratulations! You have bought {0}
                        Notebooks..!!!", OrderQty);
```

```
                    Console.ReadLine();
                    }
                    else
                    {
                    throw(new OutofStockException("OutofStockException Raised:
                    The number of item you want to buy is out of stock. Please enter
                    total item number within stock"));
                    }
                }
                catch (OutofStockException oex)
                {
                    Console.WriteLine(oex.Message.ToString());
                    Console.ReadLine();
                }

        }
    }

//Custome Exception - OutofStockException
public class OutofStockException : Exception
{
        public OutofStockException(string message) : base(message)
        {
        }
}
```

OUTPUT:1 (It will not railse any exception and shows the Output)

Enter number of Notebook you want to buy (Total 20 in Stock): 12

Congratulations! You have bought 12 Notebooks..!!!

OUTPUT:2 (It will raise OutofStockException)

Enter number of Notebook you want to buy (Total 20 in Stock): 21

OutofStockException Raised: The number of item you want to buy is out of stock. Please enter total item number within stock"

Code Explanation

- In the above example there is a custom exception class named "**OutofStockException**". This class used to catch exception raised by catch block when user enter larger number than the stock available. All custom exception class must be derived from Exception base class and must have a

constructor. Here in example it contains a constructor to throw string message.

- In the main method, we ask user to enter quantity of notebook to buy. If the entered quantity is less than the stock than "*Congratulations*…" message will be displayed but if the buying quantity is greater than stock available than the OutofStckException will be railsed using throw statement and print error message *The number of item you want to buy is out of stock. Please enter total item number within stock*

Check your progress 2

1. In exception handling _____ statement is used to manually raise an exception.
2. All user defined exception must have to inherit Exception class (True/False).

4.5 LET US SUM UP

In this unit we have learned how to handle run time errors using Exception handling. We can now be able to use understand various blocks of exception handling e.g. try, catch and finally. We have also learned use of throw statement in user defined exception handling.

4.6 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your progress 1

1. **TRUE**
2. **FALSE**

Check your progress 2

1. In exception handling **throw** statement is used to manually raise an exception.
2. **TRUE**

4.7 FURTHER READING

- In depth detail can be referred from Microsoft documentation web site: https://docs.microsoft.com/en-us/dotnet/csharp/\
- Reference Book: Beginning C# Programming by Benjamin Perkins, Jacob Vibe Hammer and Jon D. Reid, wrox publication.

4.8 ASSIGNMENTS

- Briefly explain the importance of exception handling in C# programming.

4.9 ACTIVITIES

- Create a C# console application to demonstrate the use of Exception handling with multiple catch blocks.

4.10 CASE STUDIES

- Compare Exception Handling mechanism of C# with other OOP languages.

Block-3

Inheritance, Interface and Generics

Unit 1: Inheritance In C# 1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Inherit class.
- Reuse and enhance functionality of base class.
- Use different types of inheritance with C#.
- Understand and learn concept of OOPs.

1.2 INTRODUCTION

Object oriented programming provides code reusability and very compact to manage code. Object can hold data and code (procedure) to perform specific kind of operation on data.

OOPs provide code reusability functionality by using inheritance. All object oriented programming languages support inheritance for extending existing facility of one class into another class.

In this unit you are going to learn inheritance by using C#.

1.3 INHERITANCE

Inheritance is a process of acquiring properties and assets of the person by legal recipient in general terms. Same way in OOPs one class can inherits properties and attributes of one class into another class.

A class which is inherited by other class is recognising as parent class or base class. In .net framework **System.Object** is ultimate base class of all other classes.

A class which is inherited from base class is known as child class or derived class and it is inherits members of base class. Derived class can extend functionality of base class as per requirements. Derived class can override members of base class to provide different implementation according to requirement.

Derived class can inherits public, protected members of base class and internal members only if base class and derived class available in same assembly.

To perform inheritance in C# special operator **:** is used.

Syntax:

<access modifier> Class <derived class name> : <base class name>

{

}

Check your Progress1

1. How you can categorise System.Object class in .Net?
   A. Derived Class
   B. Base Class.
   C. Ultimate base class
2. In OOPs code reusability achieve by _____.
   A. Polymorphism
   B. Inheritance
   C. Encapsulation
   D. Overriding

1.4 TYPES OF INHERITANCE

There are several forms of inheritance

- Single inheritance
- Multiple inheritance
- Multilevel inheritance
- Hierarchical inheritance
- Hybrid inheritance

C# does not support multiple class inheritance but support multiple interface inheritance.

1.4.1 Single Inheritance

In single inheritance one class derived from other class. Look in figure 1Class2 derived from Class1.

1.4.2 Multiple Inheritance

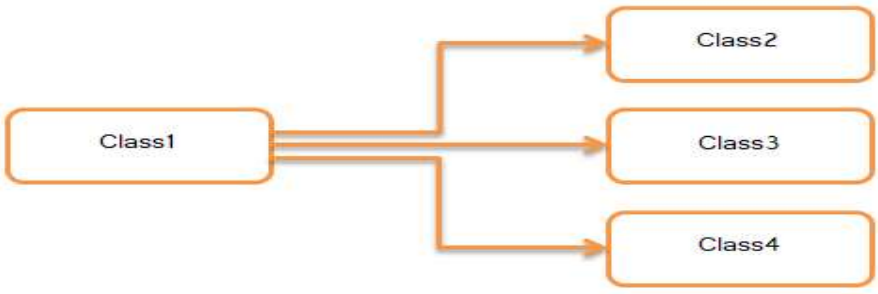
In multiple inheritances one class derived from more than one class. C# only supports multiple interface inheritance.

1.4.3 Multilevel Inheritance

In multilevel inheritances one class derived from other derived class. Like grandson inherits properties of grandfather same way in figure 3 Class3 acquire properties and attributes of Class1 by inherits from Class2

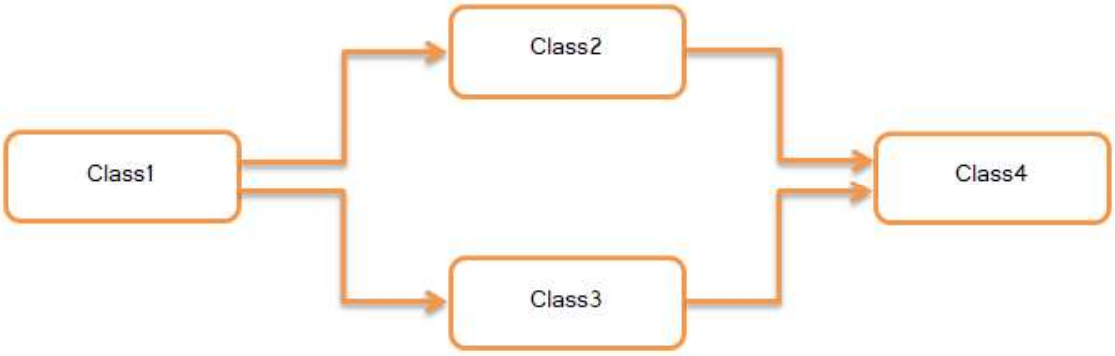
1.4.4 Hierarchical inheritance

In hierarchical inheritance one class can be derived by more than one class. Like children of same parents share common properties of their parent. Look in figure 4 where Class1 is inherited by Class2, Class3 and Class4.



1.4.5 Hybrid inheritance

Hybrid inheritance is mixture of multiple and multilevel inheritance. Like child acquire properties of father's father and mother's father-in-law from father and mother. C# not supports hybrid inheritance for class.



Check your Progress 2

1. Can we do Multiple class inheritance in C#.NET?

   A. Yes

   B. No

2. Tick mark [√] on types of inheritance which are supported by C#.NET for class inheritance.

A.	Single inheritance	[ ]
B.	Multiple inheritance	[ ]
C.	Multilevel inheritance	[ ]
D.	Hierarchical inheritance	[ ]
E.	Hybrid inheritance	[ ]

1.5 IMPLEMENTATION OF INHERITANCE IN C#

Before we start to implement inheritance we need to define visibility of members of base class by using access modifier. Following is list of access modifier with visibility of each access modifier.

Access Modifier	Visibility
private	Members only accessed inside of class
public	Members accessible by any code anywhere.
protected	Members accessed inside of class and by derived class
internal	Members accessible in same assembly
Protected internal	It is combination of protected and internal modifiers

Let's define class for Person which contains attribute of person.

```
public class Person
{
        public string Name {get;set;}
        public string Address {get;set;}
        public DateTime DateOfBirth {get;set;}
}
```

Now derive Student class from Person class by using inheritance.

```
public class Student : Person
{
        public int RollNo {get;set;}
        public string ProgramName {get;set;}
        public string Semester {get;set;}
}
```

In above example Student become derived class and Person become base class. Student class can be access all properties of base class as in Person class all properties mark with "public" access modifier.

Let's make instance of Student class and try to access member of base class.

```
public static class Program
{
        public static void Main()
        {
        //Make instance of Student class
        Student obj = new Student();
        obj.RollNo = 19;
        obj.Name = “Vidit”;
        obj.Address = “Ahmedabad”;
        obj.ProgramName = “M.Sc.(IT)”;
        obj.Semester = “II”;
        obj.DateOfBirth = new DateTime(1999,31,12);
        //put the code to print students details on console
        ……….
        }
}
```

Have you seen that all the members of base class are accessed by using instance of derived class? Try yourself and implement above functionality.

C# provides virtual and override keywords to mark property or method of base class can be override by derived class. If we not mark member of base class as virtual then derived class can define member with same name and same signature it hides base class version. Following example explains you how you can mark base class method as virtual.

```
namespace MethodOverriding
{
class BaseClass
   {
```

```
public virtual void DemoMethod(string msg)
      {
Console.WriteLine("This is base class method " + msg);
      }
   }
class DerivedClass : BaseClass
   {
public override void DemoMethod(string msg)
      {
string s = "This is Derived Class";
          s = s + "   " + msg;
          Console.WriteLine(s);
      }
public void NewMethod()
      {
base.DemoMethod("Testing");
      }
   }
}
```

In above example BaseClass's DemoMethod marked with *virtual* keyword so you can say DemoMethod is virtual method and it can be override by derive class by using *override* keyword.

To access base class method from derived class C# provides *base* keyword. Look code of NewMethod( ) where DemoMethod of base class called by using *base* keyword.

```
base.DemoMethod("Testing");
```

If virtual and derived keywords not used for method with same name and same signature in base class and derived class respectively Microsoft Visual Studio shows you warning. To avoid the warning mark derived class method with *new* keyword.

Check your Progress 3

1. The member marked with *protected* keyword would be _____.
   - A. accessible anywhere
   - B. accessible only inside of class
   - C. accessible inside of class and derived class
   - D. not accessible
2. *override* keyword used by member of derived class.
   - A. True
   - B. False

1.6LET US SUM UP

In this unit you learn about code reusability and functionality extension by using inheritance. Inheritance is the process of acquiring functionality of one class into other class. C# supports single, multilevel and hierarchical inheritance.

C# provides private, public, protected, internal and protected internal access specifiers. C# compiler by default apply private access modifier to members of the class.

To perform inheritance C# use ":" operator.

In .Net every class is by default inherited by System.Object class if not inherited by other class.

To extend the functionality of base class override members by using *override* keyword.

1.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

Answer – 1: C

In .Net every class is by default inherited by System.Object class if not inherited by other class.

Answer – 2: B

You can reuse and extend functionality by using inheritance.

Check your Progress 2

Answer – 1: B

Only multiple interface inheritance is supported by C#.

Answer – 2: A, C and D

Check your Progress 3

Answer – 1: C

Protected members are accessed by derived class and same class where they are defined.

Answer – 2: A

override keyword used to override virtual member of base class.

1.8 FURTHER READING

- Chapter-4 Inheritance
  Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner, Professional C# 2012 And .Net 4.5, Wrox Publication
- Inheritance (C# Programming Guide)
  https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/inheritance

1.9ASSIGNMENTS

- Implement multilevel inheritance for Person, Student and Exam class. Identify members of Exam class yourself and write C# code.

1.10ACTIVITIES

- **Activity-1**

  Search Object class from object browser in visual studio and list all the methods implemented in System.Object class with parameters list and return types.

- **Activity-2**

  Try to perform multiple class inheritance and note what type of error you got and why.

Unit 2: Interfaces In C# 2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Create and use interface.
- Implement members of interface by using interface inheritance.
- Perform multiple interface inheritance.
- Understand and learn concept of OOPs.

2.2 INTRODUCTION

An interface provides structure for functionality that other classes implement differently and according to requirement but use common signature. Interface is basically used by developers who want to provide common structure for other developers who are going to implement functionality of interface. In general terms interface can be anything that provides functionality but it hide how the functionality works, For example breaking system of car or bike.

In this unit you are going to learn how to create interface and implement interface by using interface inheritance.

2.3 INTERFACE

Interface declares members like properties, methods, events, indexers without implementation. Interface does not contain data members like fields or variables and constructor. Interface is used to define functionality which is implemented by others as per requirements but by using common interface. You cannot make instance of interface. If we want to use functionality of interface first we need to implement it in other class by interface inheritance. Interface is more like abstract class but some time abstract class have implemented members while in interface only declaration of members.

In .Net interface declaration is more similar like class. To declare interface "*interface*" keyword is used by c#. Interface is not permit use of access modifier for declaration of members. In .Net class library all interface name start with capital – I alphabet. For example IEnumerable, IEnumerator, ICollection, IDisposable etc.

Syntax:

interface IInterfaceName

{

}

Check your Progress1

1. Which keyword used to define interface in C#?
   A. abstract
   B. class
   C. interface
   D. override
2. _____ access modifier used by interface to define its members.
   A. public
   B. private
   C. protected
   D. None of these

2.4 DEFINE INTERFACE IN C#

This section describes how to define interface by using example of mobile phone functionality. Let's make list of functionality provided by basic mobile phone.

- Wireless communication
- Make phone call
- Receive call from other
- Get SMS
- Send SMS etc.

Now make interface for mobile phone that compulsory bind other peoples those want to make mobile phone compulsory provides functionality mentioned above.

Let's create interface with name IMobilePhone and define functionality as per list.

```
publicinterface IMobilePhone
    {
void MakeCall(long PhoneNo);
long ReceiveCall();
void SendSMS(long PhoneNo, string Message);
string ReceiveSMS();
    }
```

In above example the IMobilePhone interface created with declaration of four methods with signature and without access modifier. MakeCall method return nothing and take phone number as parameter. ReceiveCall method returns phone number and not take any arguments. SendSMS method returns nothing and takes phone number and message as parameters. ReceiveSMS return message as string and not take any argument. People who use the interface must implement all methods which declared with same signature.

Check your Progress 2

1. Can we use access modifier to declare interface in C#?

A. Yes

B. No

2. We can make instance of interface in C#.

A. True

B. False

2.5 INTERFACE INHERITANCE

Interface is just a guideline for functionality so responsibility of implementing the functionality is on the class who inherit the interface. Interface inheritance can be performed same as class inheritance. Multiple interface inheritance is possible in C#. If class inherit interface and not implement methods define in interface than visual

studio shows compile time error like class does not implement member of the interface for each member. Look in figure-1 for error message.

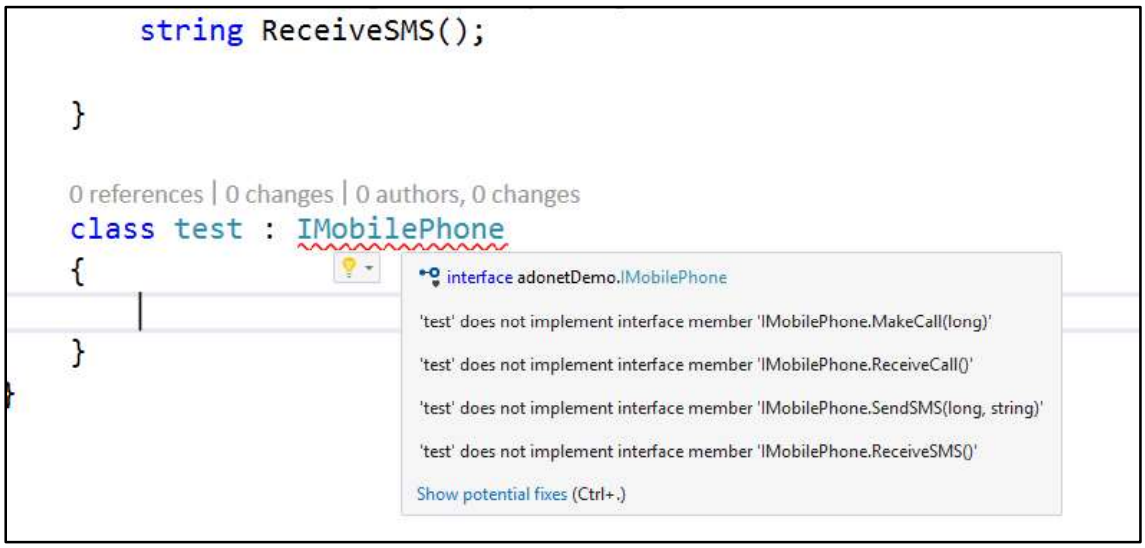


Figure-2.1 Show error

To view potential fixes from visual studio press shortcut key Ctrl+. or right click on interface name and select “Quick Actions and refactorings..” menu item from context menu. Look in figure-2 to view context menu items.

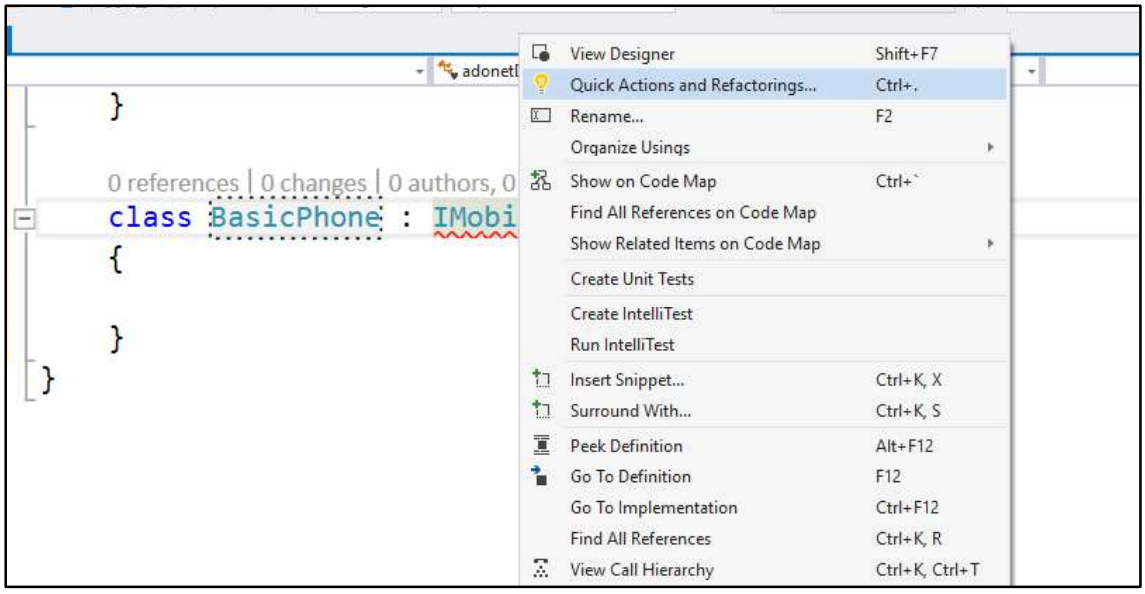


Figure-2.2 Context Menu

Click on Quick Actions and refactorings..which shows you details of members of interface which you need to implement and options to implement interface. Look in figure-3 top left corner for “Implement interface” options.

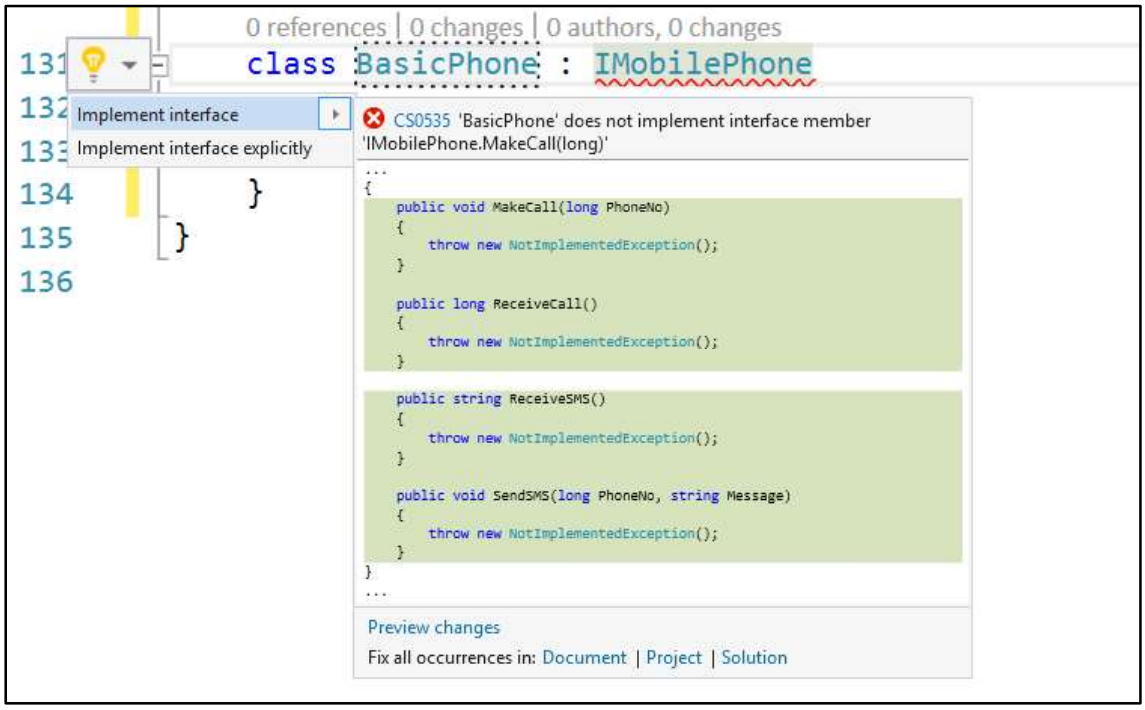


Figure-2.3 Potential Fixes

Now click on implement interface option which create skeleton for each members of interface with one line code.

throw new NotImplementedException();

Which raise runtime exception “NotImplementedException” if you are not write your own code for members. Look in Figure-4 for action taken by visual studio.

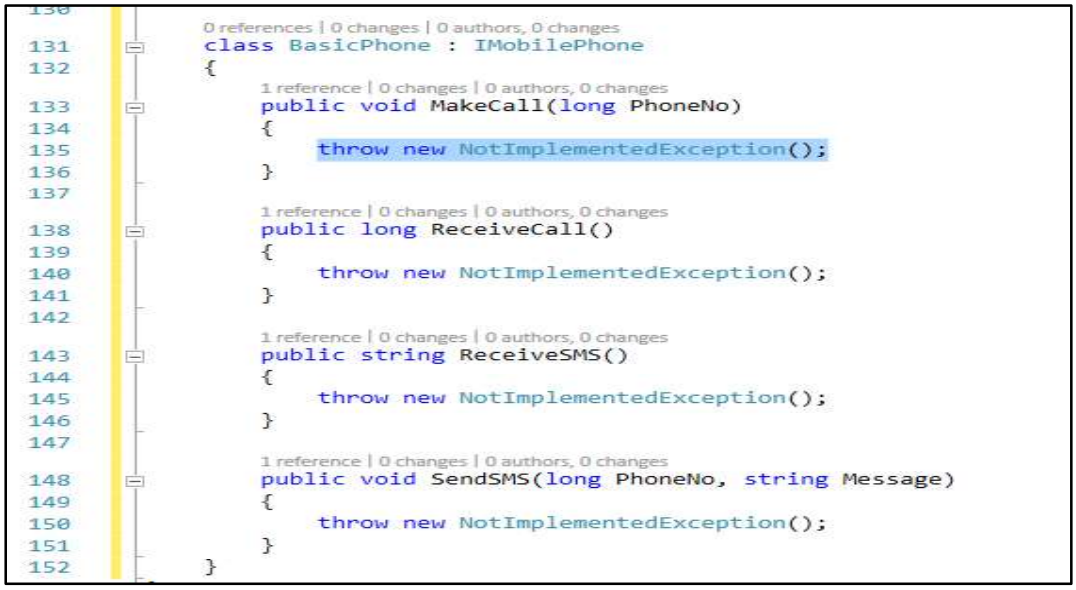


Figure-2.4 Default implementation of interface

Let’s inherit IMobilePhone interface in BasicPhone class and include following code.

```
public class BasicPhone : IMobilePhone
{
        public void MakeCall(long PhoneNo)
      {
        //Include code to make call
        Console.WriteLine(“Connecting to phone no – “ + PhoneNo);
      }
public long ReceiveCall()
      {
//Detect phone no from caller and return
        //For testing purpose use any number and return
        long phoneNo = 9999999999;
        return phoneNo;
      }
public string ReceiveSMS()
      {
string message = “This is test message”;
        return message;
      }

public void SendSMS(long PhoneNo, string Message)
      {
throw new NotImplementedException();
      }
}
```

In this example BasicPhone class implement basic functionality of IMobilePhone interface. To use functionality use BasicPhone class and make instance of it.

```
public static class Program
{
        public static void Main()
        {
        //Make instance of BasicPhone class
        BasicPhone obj = new BasicPhone();
        obj.MakeCall(9999999999);
        Console.WriteLine(obj.ReceiveCall());
        Console.WriteLine(obj.ReceiveSMS());
        //SendSMD throw exception as functionality not implemented
        obj.SendSMS(9999999999,”This is test message”);
        Console.ReadLine();
        }
}
```

When you are execute above code you will get runtime exception because SendSMS method is implemented by default and it throw exception. To overcome this problem modify functionality as follow

```
public void SendSMS(long PhoneNo, string Message)
      {
Console.WriteLine( “Message –“ + Message + “ sent to phone no – “ +  PhoneNo);
      }
```

You can try to implement IMobileInterface differently as you like but compulsory use same signature and return type for each method.

Multiple interface inheritance can be performed by providing “,” separated list of interfaces.

public class BasicPhone : IMobilePhone, IDisposable { ……. }

IDisplosable is inbuilt interface provided by .net framework to implement. It is simple interface contains only declaration of Dispose() method.

Check your Progress 3

1. What is use of shortcut key – “Ctrl+.” in visual studio?
   - A. Implement interface
   - B. Show potential fixies
   - C. Show Error List
   - D. None of Above

2.6LET US SUM UP

In this unit you learn about interface and implementation of interface. An Interface contains only declaration of methods, properties, events and indexers. Interface can be inherited same way as class inherited in C#. For example

public class BasicPhone : IMobilePhone { }

It is compulsory to implement all members of interface if interface inherited by any class. Multiple interface inheritance is possible in C#.
.Net Framework provides numbers of interfaces for various functionality implementations.

2.7CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

Answer – 1: C

In C# “interface” keyword is used to define interface.

Answer – 2: D

C# not allowed and access modifiers for members of interface.

Check your Progress 2

Answer – 1: A

You can set visibility of interface by using access modifier but not for members of interface.

Answer – 2: B

You cannot make instance of interface as it is only guideline for implementers.

Check your Progress 3

Answer – 1: B

Shows potential fixies by visual studio for code line where cursor is putted.

2.8 FURTHER READING

- Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner, Professional C# 2012 And .Net 4.5, Wrox Publication
- Interfaces (C# Programming Guide) https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/

2.9ASSIGNMENTS

- Create interface for contacts and implement it into BasicPhone class with IMobilePhone.

2.10ACTIVITIES

- **Activity-1**

  Make list of interfaces and its members defined in System interface.

Unit 3: Structures in C# 3

Unit Structure

3.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Create and use structure.
- Differentiate class and structure
- Improve performance for simple user define types

3.2 INTRODUCTION

There are two types in C#, Value types and reference types. Structure is value types and used to create user define composite types like class. Class is reference types.

In this unit you are going to learn how to create and use structure in C# and compare with class.

3.3 STRUCTURE

Structure is by nature value types. It is implicitly inherited from System.ValueType. It hold members like fields, properties, parameterized constructor and static parameter less constructor, methods, indexers, operators and events. Structure is used to create user define value types for related data. For example if you want to store information of student and make group of information than structure is helpful to create group of Student Id, Name, Program name and other information. Structure can be created by using “struct” keyword in C#.

Structure is useful to create simple light weight variables of related types. .Net framework use structure to store information of point, color etc… Structure is useful in situation like when create array of structure is more beneficial as compare to array of objects of any class because each element of array contains references of the objects and objects data while structure objects directly store value and save memory.

Syntax:*structStructName*

{

}

Example:

Make structure for student.

```
struct StudentStruct
{
int RollNo;
string StudentName;
string ProgramName;
}
```

Suppose you want to store cursor position on the screen or any point on chart by using X axis and Y axis. Structure can be created as follow.

```
struct MyPoint
{
int X;
int Y;
}
```

Structure can support only parameterized constructor and static parameter less constructor. Parameter less constructor for structure is not allowed. Try to create parameter less constructor in C#, Visual Studio shows you error – “Struct cannot contain explicit parameterless constructors”. Look in figure 3.1 for error information. Default parameter less constructor is supported by .net framework. Figure 3.2 shows static parameter less constructor.

```
6
7   namespace BAOUDemo
8   {
        0 references
9       class Program
10      {
            0 references
11          static void Main(string[] args)
12          {
13          }
14      }
15
            1 reference
16      struct DemoStruct
17      {
            0 references
18          public DemoStruct()
19          {
20
21          }
22      }
23  }
24
```

DemoStruct.DemoStruct()

Structs cannot contain explicit parameterless constructors

Figure 3.1 Parameter less constructor

Constructor is not allowed initialized instance member field in C# and if you try to

```
16      struct DemoStruct
17      {
18
19              public DemoClass obj1;
20              public DemoClass obj2 = new DemoClass();
            0 references
21          public DemoStruct()
22          {
23
24          }
            0 references
25          static DemoStruct()
26          {
27              Console.WriteLine("This is static parameterless constructor call");
28          }
29
30      }
31
            4 references
32      class DemoClass
33      {
                1 reference
34          public DemoClass()
35          {
36              Console.WriteLine("Demo Class Instance created");
37          }
38      }
39  }
```

(field) DemoClass DemoStruct.obj2

'DemoStruct': cannot have instance property or field initializers in structs

Figure 3.2 – Parameter less static constructor

declare visual studio shows error like "'StructName': cannot have instance property or field initializers in structs". Look in figure 3.2.

A structure cannot be created by using class inheritance but can be inherited from interface. A structure cannot become base for other class or structure.

Check your Progress1

1. Struct is _____ type in C#.
   A. reference
   B. value
   C. object
   D. none of these
2. Structure can be created by using class inheritance.
   A. True
   B. False

3.4 DIFFERENCE BETWEEN CLASS AND STRUCTURE

Sr. No	Class	Structure
1.	Class is reference type.	Structure is value type.
2.	We can declare parameter less constructor in class.	We cannot declare parameter less constructor in structure. It allows parameterized constructor and static parameter less constructor.
3.	Class must instantiated by new keyword.	Structure can be instantiated without new keyword but in this case you are not able to use all members.
4.	Class can be a base class of other class.	Structure cannot be a base for other structure or class.
5.	Class can be derived from other class or interfaces.	Structure only derived from interfaces.

Table-3.1Difference between Class and Structure

Check your Progress 2

1. Structure can be inherited from multiple interfaces.

A. True

B. False

3.5 CREATE STRUCTURE IN C#

Structure is used to create user defined value types which improve performance as compare to class. Let's consider example of colour, colour is a combination of RGB where R for red, G for Green and B for Blue. As the intensity of RGB changed the colour changed accordingly. To store any colour you need RGB values and values are between 0 and 255 for R, G and B. To represent colour create structure that hold values of R, G and B as fields and one parameterized constructor that initialize structure fields.

```
struct MyColour
  {
      byte R;
      byte G;
      byte B;
public MyColour(byte r, byte g, byte b)
    {
       R = r;
       G = g;
       B = b;
    }
  }
```

The above code creates structure with name MyColour with R, G and B fields and one constructor. The datatype of R, G and B is taken as byte because the range is between 0 and 255.

Let's make instance of structure in Main method by using new keyword same like we make instance of class.

```
static void Main(string[] args)
    {
        MyColour myColour = new MyColour(255, 255, 255);
    }
```

The constructor of MyColour structure initialize R, G and B fields. But our fields are private so we are not able to get values. Let's create properties for RGB and method GetRGB() for MyColour structure.

```
struct MyColour
  {
byte R;
byte G;
byte B;

public MyColour(byte r, byte g, byte b)
    {
        R = r;
        G = g;
        B = b;
    }

public byte Red
    {
set
```

```
            {
                R = value;
            }
get
            {
return R;
            }
        }

public byte Green
        {
set
            {
                G = value;
            }
get
            {
return G;
            }
        }

public byte Blue
        {
set
            {
                B = value;
            }
```

```
get
        {
return B;
        }
    }

public void GetRGB()
    {
Console.WriteLine("R = {0} , G = {1}, B = {2} ", R,G,B);
    }


  }
```

Now make changes in Main method and set values of RGB by using respective properties and get by using method.

```
static void Main(string[] args)
    {
        MyColour myColour = new MyColour();
        myColour.Red = 155;
        myColour.Green = 72;
        myColour.Blue = 180;

myColour.GetRGB();

Console.WriteLine();
    }
```

OUTPUT:

R = 155, G = 72, B = 180

In above code default constructor is used to make instance of structure MyColour and initialized RGB values by using properties. You can get value of R, G and B by using GetRGB method that display console message.

In case of only declaration of structure C# compiler shows you error like “Use of unassigned local variable” when trying to access members of structure. Look in figure 3.3. Structure is value type so you cannot initialize with null value.

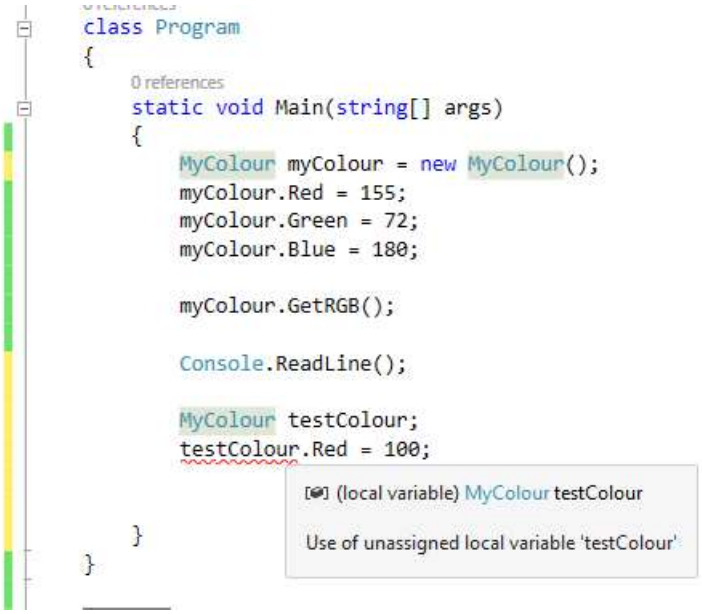


Figure 3.3 – Uninitialized structure variable

Check your Progress 3

1. “MyColour testColour = null;” statement is valid or not for MyColour structure.

A. Is Valid

B. Is Not Valid

3.6LET US SUM UP

In this unit you learn about structure. Structure is value type. Members of structure are fields, methods, properties, events, indexer and constructor.

C# automatically create default constructor for structure. Static parameter less and parameterized constructors are supported by C#.

Structure is useful for creating single variable that hold related data. For example position of cursor, colour, point etc.

Structure cannot be base of other structure or class. Structure only inherits from interface.

.Net Framework provides numbers of structures for various functionality implementations.

3.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

Answer – 1: B
Structure is value type.

Answer – 2: B
Structure can be inherited from interface only.

Check your Progress 2

Answer – 1: A
Structure supports multiple interface inheritance.

Check your Progress 3

Answer – 1: B
Structure is value type so you cannot initialize with null.

3.8 FURTHER READING

- Herbert Schildt, C# 4.0: The Complete Reference, Mc Graw Hill publication
- Using Structures (C# Programming Guide) https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-structs

3.9 ASSIGNMENTS

- Create structure for employee to store employee related information like Id, Name, Join date, basic salary.

3.10 ACTIVITIES

- **Activity-1**

  Make list of structure and its members defined in System interface

Unit 4: Operator Overloading and Generics in C#

4

Unit Structure

4.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Create and use user define operators
- Works with generics types
- Create generics types

4.2 INTRODUCTION

There are many operators defined in C# and classified as arithmetic operators, assignment operators, logical operators, relational operators, bitwise operators etc. An operator is a symbol or group of symbols (Characters) that apply on one or more operands in expression or statements.

An operator takes on operand is called unary operator. For example increment(++) or decrement(--) operators are unary operators.

An operator that takes two operands is called binary operator. For example arithmetic operators (+,-,*,/,%)

int SUM = 12+15;

An operator that takes three operands are called turnery operator. For Example conditional operator (? :). That takes three operands.

int A = 100, B = 200;

int LargeNo = (A>B) ? A : B;

In topic 4.3 you are going to learn operator overloading in C#. In C# the plus operatot(+) is used for two different type of operation like sum of numbers and it is also used for performing string concatenation operation.

In topic 4.4 you are going to learn about generics. Generic is a mechanism that provides type safety to user defined data structures. And it avoids boxing and unboxing when creates collection of generic types.

4.3 OPERATOR OVERLOADING IN C#

Overloading is types of polymorphism and it gives different meaning to operator as defined by user. Overloading is a technique used to define single identifier for performing multiple operations. C# supports two types of overloading.

1. Method overloading
   Two or more methods defined with same name and different parameters in same class
2. Operator overloading
   Operator overloading is the method to give distinct meaning to standard C# operators with user defined type such as class or structure.

C# provides supports to user defined types to overload operators by using special keyword “operator” and by defining static function. All standard operators are not support operator overloading but some supports like +, -, !, ~, ++,--, +, -, *, /, %, &, |, ^, <<, >>, Relational operators (==, !=, <, >, <=, >=), true, false must be overloaded in pairs. For Example (== and !=). Rest of the C# operators cannot overload.

To overload an operator on a user define class or structure, First declare operator in user define types and follow following rules.

1. Operator must be public and static
2. Must be attach method with name or symbol by using statement “operator XYZ”
3. For unary operator define one parameter
4. For binary operator define two parameters and any one parameter must be with same type as Class or Structure that defines operator.
5. The return type for binary operator can be any except void type.
6. For unary type operator return type can be any except void type but for true and false must be Boolean and overload in pair.
7. For ++ and – operator return type must be class type or structure type where operator declare.

Unary operators have one parameter, and binary operators have two parameters. In each case, at least one parameter must be the same type as the class or structure that declares the operator.

Syntax:

public static <return type> operator <op symbol> (parameters list){ ----}

Example:

The + symbol is used as plus operator for numeric operands and string concatenation operator for string type operand. Let's overload + symbol for sum of two matrix type object. To overload + symbol first create class with name Matrix that hold value of matrix and perform operator overloading for sum operation of two matrix and display elements of matrix.

```
class Matrix
  {
int A, B, C, D;
public Matrix(int R1E1, int R1E2, int R2E1, int R2E2)
    {
      A = R1E1;
      B = R1E2;
      C = R2E2;
      D = R2E2;
    }
      //overload + operator for sum of two matrixes
public static Matrix operator + (Matrix matrix1, Matrix matrix2)
    {
      //Make instance of Matrix class that hold sum of two matrix
      Matrix SumOfMatrix = new Matrix(matrix1.A + matrix2.A, matrix1.B+ matrix2.B, matrix1.C + matrix2.C, matrix1.D + matrix2.D);
return SumOfMatrix;
```

```
    }
public void GetMatrix()
    {
Console.WriteLine(A + "\t" + B);
        Console.WriteLine(C + "\t" + D);
    }
  }
```

Above code first create Matrix class with one constructor that initialize elements of 2X2 matrix and store in local variables A,B,C and D.

Operator + overload takes two argument both of matrix type and make sum of each element of both matrixes matrix1 and matrix2 and store in new matrix SumOfMatrix.

GetMatrix method return each element on console by using Console.WriteLine statement.

To test the functionality works create two instance of Matrix class and make sum of both as per below code.

```
class Program
  {
static void Main(string[] args)
    {
        Matrix M1 = new Matrix(1, 1, 1, 1);
        Matrix M2 = new Matrix(1, 1, 1, 1);
        Matrix M3 = M1 + M2;
M3.GetMatrix();
Console.ReadLine();
    }
  }
```

The output of above code is

2 2

2 2

The statement "Matrix M3 = M1 + M2;" demonstrate use of operator overloading where M1 and M2 both are Matrix type instance and the sum of this two matrix store in M3 matrix by using + operator. In this program + operator is used to make some of two matrixes.

Check your Progress1

1. + operator is _____ type of operator
   A. Unary
   B. Binary
   C. Turnery
   D. None of these
2. "operator" is keyword in c#.
   A. True
   B. False
3. For "-"operator, _____ is the return type.
   A. void
   B. int
   C. class type that declare "-" operator
   D. None of these

4.4 USING GENERICS IN C#

Generics is very powerful features of the C# programming language and it was in traduce when .Net framework 2.0 released. Before that programmers are not able to apply same logic on different data types by using single class implementation. For each type they need to write separate code and if object type collection object created than the boxing and unboxing process compulsory performed while storing and retrieving objects in and from collection object. For example working with ArrayList or HashTable objects where you are able to store any type of values.

Generics are helpful to create parameterized types for classes, structures, methods, interfaces etc. With the help of generics you are able to create generic classes, generic methods or generic interfaces.

Before generics C# depend on object type to create generalized code that is reusable with different data types but it required boxing and unboxing and it is not provides type safety. To provide type safety and avoid type casting Microsoft introduce generics. For example set and get value of specific type. Type may be int, float or any other users define type like Student or Employee. Let's create one class that provide facility to get or set value of integer number.

```
class DemoClass
{
        int number;
        public void SetNo(int no)
                {
                        number = no;
                }
        Public int GetNo()
                {
                        return number;
                }
}
```

The DemoClass is only capable to handle integer numbers and if you want to provide facility to handle other type of numbers you required to rewrite code for other numeric types. Now let's create same functionality with generics that provide parameterized types and generic class is capable to handle any type. You can create parameterized type class by appending "<T>" after name of class. You can use other character or name instead of "T" but compulsory enclosed between < and >.

```
class GenericDemoClass<T>
  {
     T number;
public void SetData(T no)
     {
number = no;
     }
public T GetData()
     {
return number;
     }
  }
```

In above code same logic used as DemoClass but “int” type is replaced with “T” type. T is parameterized type and when make instance of generic class provide required type. GenericDemoClass is capable to handle any types. Let’s use generic class in following code.

```
static void Main(string[] args)
     {
       // pass int as parameterized type for <T>
        GenericDemoClass<int> obj1 = new GenericDemoClass<int>();
obj1.SetData(100);
Console.WriteLine("The number is " + obj1.GetData());
Console.WriteLine("The type of data stored in GenericDemoClass object is " +
        obj1.GetData().GetType());
 // pass float as parameterized type for <T>
       GenericDemoClass<float> obj2 = new GenericDemoClass<float>();
obj2.SetData(98.1067f);
```

```
Console.WriteLine("The number is " + obj2.GetData());
Console.WriteLine("The type of data stored in GenericDemoClass object is " +
        obj2.GetData().GetType());
        //Use user define type Matrix
        //pass Matrix as parameterized type for <T>
        GenericDemoClass<Matrix> obj3 = new GenericDemoClass<Matrix>();
obj3.SetData(new Matrix(1,1,1,1));
Console.WriteLine("Matrix =");
obj3.GetData().GetMatrix();
Console.WriteLine("The type of data stored in GenericDemoClass object is " +
        obj3.GetData().GetType());
Console.ReadLine();
    }
```

Look in above code GenericDemoClass is capable to handle int, float and Matrix types and produce following output.

OUTPUT:

```
The number is 100
The type of data stored in GenericDemoClass object is System.Int32
The number is 98.1067
The type of data stored in GenericDemoClass object is System.Single
Matrix =
1    1
1    1
The type of data stored in GenericDemoClass object is BAOU_B3_U3_Operator_Overloading.Matrix
```

.Net framework provides number of generic collections. To use generic collections in C# include namespace System.Collection.Generics. List is an example of generic collection.

```
using System;
using System.Collections.Generic;
namespace BAOU_B3_U3_Generics_List
{
class Program
   {
static void Main(string[] args)
      {
          List<int> ls = new List<int>();
          //Add items in list
ls.Add(1);
ls.Add(2);
ls.Add(3);
          //Display items from list
Console.WriteLine("Items in list");
foreach(int no in ls)
          {
Console.WriteLine(no);
          }
Console.ReadLine();
}
   }
}
```

OUTPUT

Items in list

1

2

3

Check your Progress 2

1. Generic class can be capable to handle any type.

A. True

B. False

2. <T> is used to pass _____.

A. Data

B. Parameter

C. Parameterized Type

D. None of these

3. Select valid statement for declaration of generic class.

A. public class DemoClass {------}

B. public class DemoClass(T) {------}

C. public class DemoClass<type> {------}

D. None of these

4.5LET US SUM UP

In this unit you learn about operator overloading and generics.

Polymorphism can be achieved by using method overloading, operator overloading and method overriding. Operator overloading is used to give different meaning to standard C# operators for user defined class or structure.

In C# operators are classify as unary, binary or ternary operator. All standard operators are not support operator overloading but some supports like +, -, !, ~, ++,--, +, -, *, /, %, &, |, ^, <<, >>, Relational operators (==, !=, <, >, <=, >=), true, false must be overloaded in pairs. For Example (== and !=). Rest of the C# operators cannot overload.

Operator overloading can be performed by using “operator” keyword and syntax for operator overloading is

public static <return type> operator <op symbol> (parameters list){ ----}

.Net framework 2.0 introduce generics that provide type safety and facility to reuse common logic for different data type. To works with generics first we need to create generic class by using parameterized type and when make instance of generic class pass the required data type.

Declaration of generic class is like

public class GenericDemoClass <T> { ----- }

To make instance of generic class is

GenericDemoClass<int> obj = new GenericDemoClass<int>( );

.Net framework provide generic collection like List<>, Stack<>, Queue<>, Dictionary<> etc. To use generic collection include following namespace.

using System.Collections.Generic;

4.6CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

Answer – 1: B

+ Operator is binary operator and used like C = A+B

Answer – 2: A

“operator” is keyword used to overload operator in C#.

Answer – 3: C

"-" Operator is used return type as class or structure in which "-" operator declared.

Check your Progress 2

Answer – 1: A

Generic class is capable to works with any type which is passed as parameterized type

Answer – 2: C

Parameterized Type

Answer – 3: C

public class DemoClass<type> {------} is valid statement

4.7 FURTHER READING

- Herbert Schildt, C# 4.0: The Complete Reference, Mc Graw Hill publication
- Overloadable operators (C# Programming Guide) https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/overloadable-operators

4.8ASSIGNMENTS

1. State and classify all operators in C#
2. Create generic interface for Shape interface. Define Area and Volume methods that capable to calculate area and volume with any numeric type for shape.

4.9ACTIVITIES

- **Activity-1**

  Perform operator overloading for ++ and - - operators.

- **Activity-2**

  Perform push and pop operation on generic Stack<> collection.

Block-4

Threading, File handling, C# controls

Unit 1: Multithreading 1

Unit Structure

1.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Create threads
- Manage threads
- Understand thread life cycle
- Communicating data to a Thread

1.2 INTRODUCTION

Each application runs with at least one thread. Thread is like path of executing the application. There are two types of application. One is single thread application and second is multithreading application.

Single thread application is only create one thread. Example of single thread application is embedded system.

Multithreading application can create and control more than one thread. It starts with main thread and later on main thread creates other threads as per requirements.

This unit in details describes you working with threads with C# programming language. You will learn how to create and manage thread using C#.

1.3 GETTING STARTED WITH THREADS

Operating system executes multiple applications simultaneously by creating process for each application. In this way operating system provides multitasking and allocates processing time to each process. Each application at least creates one thread and the thread is called Primary Thread (Main Thread). Application may create many other threads for concurrent work. These threads are called worker threads (Other Threads).

Thread is defines execution path of application. By using multithreading application, application can define multiple execution paths. Multithread application works more efficiently and executes multiple part of application at same time as per allocated time slot.

The main advantages of multithreading are increase responsiveness of application and take advantages of multi core processor.For example, your application performs more than one operation and that can be done in parallel.The total execution time can be decreased by performing those operations in separate threads and running the application on a multicore processor. Multithreading might increase performance and responsiveness of the application.

C# has inbuilt support multithreading. C# provides robust facility for multithreading and eliminated problems associated with multithreading in older programming languages.

Microsoft has continually enhancing the functionality of .net framework. It include new features in parallel programming and multithreading like TPL (Task Parallel Library) and PLINQ (Parallel Language Integrated Query). TPL and PLINQ support multicore processors.

C# provides multithreading related functionality via System.Threading namespace. The System.Threading namespace contains classes and interfaces to provides facility for multithreading. The important class of the namespace is Thread and following are important properties the class

Property	**Description**
IsAlive	Returns true or false. If a thread has been started and not terminated normally or aborted then return true else return false
IsBackground	Returns true or false. If a thread is a background thread then return true else return false. Background threads don't prevent a process to stop by CLR while foreground threads prevent stopping.
Name	Gets or sets the name of a thread. Name property is very useful in debugging
Priority	Gets or sets a thread priority value that is used by the operating system to prioritize thread scheduling. You can set priority by using ThreadPriority enumerator. Possible values are AboveNormal,

	Normal, BelowNormal, Lowest and Highest.
ThreadState	Gets a ThreadState value that containing the current states of the thread. The list of ThreadState are Aborted, AbortRequested, Background, Running, Stopped, StopRequested, Suspended, SuspendedRequested, Unstarted, WaitSleepJoin

Table-1 Thread class property

Following example demonstrate how to works with current thread of the application and print values of above properties.

```
using System;
using System.Threading;
namespace MultiThreadingDemo
{
class Program
   {
static void Main(string[] args)
      {
         //Create object of thread and assign current thread to it
         Thread obj = Thread.CurrentThread;
         obj.Name = "Current Thread";
Console.WriteLine("Name of Thread is " + obj.Name);
Console.WriteLine("Current state of Thread is " + obj.ThreadState);
Console.WriteLine("Priority of the Thread is " + obj.Priority);
Console.WriteLine("The Thread is alive ==>  " + obj.IsAlive);
        Console.WriteLine("The Thread is background thread. ==> " +
        obj.IsBackground);
Console.ReadLine();
      }
```

```
    }
}
```

OUTPUT:

Name of Thread is **Current Thread**

Current state of Thread is **Running**

Priority of the Thread is **Normal**

The Thread is alive ==>**True**

The Thread is background thread. ==>**False**

There are several ways to create thread and start the tread. One of the ways is use ThreadStart delegate. ThreadStart delegate is defined by .net framework. To learn how two threads work and execute part of the application, let's create two methods in DemoClass.

One method print positive numbers between 1 to 20 and second method prints negative numbers between -1 to -20.

```
public class DemoClass
   {
public void PrintPositiveNos()
      {
for (int i = 1; i <= 20; i++)
Console.WriteLine("Positive No -" + i);
      }

public void PrintNegativeNos()
      {
for (int i = -1; i >= -20; i--)
Console.WriteLine("Negative No -" + i);
```

```
    }


  }
```

Now create two thread using ThreadStart delegate in Main method and start using Start() method of thread class.

```
using System;
using System.Threading;
namespace MultiThreadingDemo
{
class Program
  {
static void Main(string[] args)
    {
      DemoClass objDemo = new DemoClass();
//Create new thread for printing positive nos.
       //Thread class constructor argument type is ThreadStart delegate
       Thread ThreadPositiveNos = new Thread(objDemo.PrintPositiveNos);
       //Start thread
ThreadPositiveNos.Start();
//Create new thread for printing negative nos.
       Thread ThreadNegativeNos = new Thread(objDemo.PrintNegativeNos);
       //Start thread
ThreadNegativeNos.Start();
Console.ReadLine();
    }
  }
```

```
}
```

OUTPUT:

```
Positive No -1
Positive No -2
Positive No -3
Positive No -4
Positive No -5
Positive No -6
Positive No -7
Positive No -8
Positive No -9
Positive No -10
Positive No -11
Positive No -12
Positive No -13
Positive No -14
Negative No --1
Negative No --2
Negative No --3
Negative No --4
Negative No --5
Negative No --6
Negative No --7
Negative No --8
Negative No --9
Negative No --10
```

Positive No -15

Positive No -16

Positive No -17

Positive No -18

Positive No -19

Positive No -20

Negative No --11

Negative No --12

Negative No --13

Negative No --14

Negative No --15

Negative No --16

Negative No --17

Negative No --18

Negative No --19

Negative No –20

Look the output of above code. Both the methods concurrently execute and print positive or negative number as per time slot allot to each thread. Same way you can create multiple threads for application for complex operation and optimize performance of the application by multithreading programming.

Check your Progress1

1. What is the default priority of Thread.CurrentThread?
   D. AboveNormal
   E. BelowNormal
   F. Normal
   G. Highest

2. Select the namespace that support multithreading in .net framework.
   E. System
   F. System.Threading
   G. System.Threading.Tasks
   H. System.Linq

1.4 MANAGING THREAD LIFETIMES

The thread lifetime can be understood by using thread life cycle. You can calculate a time span from starting of the thread to ending of the thread. The lifetime of thread is started when instance of Thread class created and ended when execution of thread is completed or terminated.

The thread is passed in several states during its lifetime. Following is the list of thread states.

- Unstarted
- Running
- SuspendRequested
- Suspended
- WaitSleepJoin
- StopRequested (Internal Use Only)
- Stopped
- Background
- AbortRequested
- Aborted

When instance of Thread created and Start() is not called at that time thread instance has **Unstarted** thread state assigned.

When instance of Thread is started and not yet stop at that time thread instance has **Running** thread state assigned.

When instance of Thread is being requested to suspend at that time thread instance has **SuspendRequested** thread state assigned.

When instance of Thread has been suspend at that time thread instance has **Suspended** thread state assigned.

When instance of Thread has been blocked by Wait(), Join() or Sleep() method at that time thread instance has **WaitSleepJoin** thread state assigned.

When instance of Thread has been requested to stop at that time thread instance has **StopRequested** thread state assigned. This state is used by .net for internal use only.

When instance of Thread is stopped at that time thread instance has **Stopped** thread state assigned.

When instance of Thread is execute in background at that time thread instance has **Background**thread state assigned. You can change foreground thread into background thread by assigning “true” value to **IsBackground**property of the thread instance.

When instance of Thread is being requested to abort by calling Abort() method and not yet aborted at that time thread instance has **AbortRequested** thread state assigned.

When instance of Thread is aborted and the state is not yet changed to Stop at that time thread instance has **Aborted** thread state assigned.

Thread instance can be manage by using several method provided by Thread class in .Net framework. Following table describe few Methods of Thread class. Refer MSDN for all methods.

Method	**Description**
Start()	Change the current instance of thread into running state. Thread instance start execution. It has one overloaded method Start(Object). Start(Object) is used to pass data to thread.
Sleep(Int32)	Suspend the current thread for given time period in milliseconds. It has one overloaded method Sleep(TimeSpan).
Join()	The instance of thread is waiting till thread terminate. Block the calling thread. It has two overloaded method Join(Int32), Join(TimeSpan).

Abort()	Abort() method terminate the thread.
Interrupt()	Interrupt the thread that is in WaitSleepJoin state.

Table-2 Methods of Thread class

Check your Progress 2

1. Which method change thread state into running state?
   A. Join()
   B. Abort()
   C. Sleep()
   D. Start()
2. Sleep() method is permanently block thread.
   A. True
   B. False

1.5 DESTROYING THREADS

Thread instance is automatically stop execution when assigned method returns. In some situation you need to destroy running thread manually. You can destroy thread instance by calling Abort() method.

Following Example shows you how to start and destroy thread. The code use DemoClass for printing positive and negative numbers simultaneously.

```
static void Main(string[] args)
    {
        DemoClass objDemo = new DemoClass();
//Create new thread for printing positive nos.
        //Thread class constructor argument type is ThreadStart delegate
        Thread ThreadPositiveNos = new Thread(objDemo.PrintPositiveNos);
        //Start thread
ThreadPositiveNos.Start();
```

```
//Create new thread for printing negative nos.
        Thread ThreadNegativeNos = new Thread(objDemo.PrintNegativeNos);
        //Start thread
ThreadNegativeNos.Start();
Console.WriteLine("ThreadNegativeNos thread started");
Thread.Sleep(10);
ThreadNegativeNos.Abort();
Console.WriteLine("ThreadNegativeNos thread aborted");
Console.ReadLine();

    }
```

Output:

```
Positive No >>1
Positive No >>2
ThreadNegativeNos thread started
Positive No >>3
Negative No >>-1
Negative No >>-2
Negative No >>-3
Negative No >>-4
Negative No >>-5
Negative No >>-6
Negative No >>-7
Positive No >>4
Positive No >>5
Positive No >>6
```

Positive No >>7

Positive No >>8

Positive No >>9

Positive No >>10

Positive No >>11

Positive No >>12

Positive No >>13

Positive No >>14

Positive No >>15

Positive No >>16

Positive No >>17

Positive No >>18

Positive No >>19

Positive No >>20

Negative No >>-8

Negative No >>-9

Negative No >>-10

Negative No >>-11

Negative No >>-12

Negative No >>-13

Negative No >>-14

Negative No >>-15

ThreadNegativeNos thread aborted

In above example first “ThreadPositiveNos” started and start printing positive numbers. Than “ThreadNegativeNos” starts and start printing negative numbers. In main method 10 milliseconds delay applied after “ThreadNegativeNos” starts so few

negative numbers print but not all from -1 to -20 because by calling Abort() method called after 10 milliseconds and it terminate “ThreadNegativeNos”.

.

Check your Progress 3

1. _____ method suspend the current thread for the specific milliseconds.

A. Abort

B. Start

C. Sleep

D. none of the all

1.6SCHEDULING THREADS

Operating system assign slice of time to execute each thread based on priority of thread. In .net threads are run under control of CLR perhaps operating system assign execution time to each thread. Each operating system use different scheduling algorithm. In CLR each thread starts with normal priority. During the execution you can change thread priority by changing Thread.Priority property. Available options for thread priority are AboveNormal, Normal, BelowNormal, Lowest and Highest.

Operating system can assign first priority to thread with “Highest” priority. Than “AboveNormal”, “Normal”, “BelowNormal” and “Lowest” sequentially. If multiple thread have same priority than operating system scheduler cycles through the threads at that priority.

Now change priority of “ThreadNegativeNos” with highest and run the example of 1.5 point again and check output.

```
static void Main(string[] args)
{
        DemoClass objDemo = new DemoClass();
```

```
//Create new thread for printing positive nos.
        //Thread class constructor argument type is ThreadStart delegate
        Thread ThreadPositiveNos = new Thread(objDemo.PrintPositiveNos);
        //Start thread
ThreadPositiveNos.Start();
//Create new thread for printing negative nos.
        Thread ThreadNegativeNos = new Thread(objDemo.PrintNegativeNos);
        //Start thread
ThreadNegativeNos.Start();
Console.WriteLine("ThreadNegativeNos thread started");
ThreadNegativeNos.Priority = ThreadPriority.Highest;
        //ThreadNegativeNos.Abort();
        //Console.WriteLine("ThreadNegativeNos thread aborted");
Console.ReadLine();
}
```

Output:

```
ThreadNegativeNos thread started
Positive No >>1
Negative No >>-1
Negative No >>-2
Negative No >>-3
Negative No >>-4
Negative No >>-5
Negative No >>-6
Negative No >>-7
Positive No >>2
```

Positive No >>3

Positive No >>4

Negative No >>-8

Negative No >>-9

Negative No >>-10

Negative No >>-11

Negative No >>-12

Negative No >>-13

Negative No >>-14

Negative No >>-15

Negative No >>-16

Negative No >>-17

Negative No >>-18

Negative No >>-19

Negative No >>-20

Positive No >>5

Positive No >>6

Positive No >>7

Positive No >>8

Positive No >>9

Positive No >>10

Positive No >>11

Positive No >>12

Positive No >>13

Positive No >>14

Positive No >>15

Positive No >>16

Positive No >>17

Positive No >>18

Positive No >>19

Positive No >>20

In above example first ThreadPositiveNos starts with normal priority, after that ThreadNegativeNos starts with normal priority but ThreadPositiveNos priority changes to highest so it complete its task before ThreadPositiveNos and print -1 to -20 numbers before positive numbers.

Check your Progress 4

1. _____ is not a thread priority value.

A. Normal

B. Highest

C. Average

D. Lowest

1.7COMMUNICATING DATA TO A THREAD

Sometime threads need to communicate with other threads or depends on task of other threads to complete to perform its own task in multithreading programming. For example thread T1 is running inside lock block and wait for resource R1 but at this time R1 is not available. T1 is blocking other threads access it till resource R1 not available. This situation impact performance of application because we are not taking full advantages of multithreading. The solution is T1 temporary release the lock and allow other thread to run. When R1 is available T1 can notified and resume the execution. This is achieved through by inter thread communication.

C# supports inter thread communication with Wait(), Pulse() and PulseAll() methods. This all methods are part of Monitor class.

The Wait() method waits till other thread to complete. It has two forms.

1. Wait(object obj)
2. Wait(object obj, int timeout)

Timeout is in milliseconds and thread wait till other thread complete or till timeout. Wait method is static and return type is bool.

Pulse and PulseAll method notify any waiting thread

1. Pulse(object obj)
2. PulseAll(object obj)

These two methods are static and return type is void.

Let understand use of these methods by changing over example of printing positive number and negative number in such a way that program print one positive number and one negative number in sequence.

```
public class DemoClass
  {
     //object used to apply lock
object locknos = new object();
public void PrintPositiveNos()
     {
for (int i = 1; i <= 5; i++)
        {
lock(locknos)
           {
Console.WriteLine("Positive No >>" + i);
Monitor.Pulse(locknos);// Notify any waiting thread
Monitor.Wait(locknos); //Wait for other thread to complete
           }
        }
     }
```

```
public void PrintNegativeNos()
      {
for (int i = -1; i >= -5; i--)
         {
lock (locknos)
            {
Console.WriteLine("Negative No >>" + i);
Monitor.Pulse(locknos);
Monitor.Wait(locknos);
}
         }
      }
   }
```

Code for main program

```
static void Main(string[] args)
      {
         DemoClass objDemo = new DemoClass();
//Create new thread for printing positive nos.
         //Thread class constructor argument type is ThreadStart delegate
         Thread ThreadPositiveNos = new Thread(objDemo.PrintPositiveNos);
         //Start thread
ThreadPositiveNos.Start();
//Create new thread for printing negative nos.
         Thread ThreadNegativeNos = new Thread(objDemo.PrintNegativeNos);
         //Start thread
ThreadNegativeNos.Start();
```

```
Console.ReadLine();
    }
```

OUTPUT:

```
Positive No >>1
Negative No >>-1
Positive No >>2
Negative No >>-2
Positive No >>3
Negative No >>-3
Positive No >>4
Negative No >>-4
Positive No >>5
Negative No >>-5
```

Look the output of this example and compare with other previous examples where sequence is maintain base on allocated time slot to threads. In this example both threads are communicate with each other and notify each other task is completed or not. If you run above example than notice that at last iteration both threads fall in wait state so program is not automatically stop. To overcome this problem use timeout in wait method with 100 milliseconds.

```
Monitor.Wait(locknos, 100); //Wait for other thread to complete or 100 milliseconds
```

Check your Progress 5

1. Which method provides notification to other thread?

A. Pulse

B. Wait

C. Sleep

D. Join

1.8 LET US SUM UP

This unit describe about multitasking and threading. By using threading application can create multiple execution path and execute them according to priority of each thread.

The heart of multithreading in C# is Thread class which is used to create and manage threads in application. It has bunch of properties and methods to achieve functionality of multitasking. Few important properties are Priority, IsAlive, Name, ThreadState etc.

The Thread class has a methods like Start, Join, Wait, Sleep, Abort etc.

Thread can be scheduled by assigning appropriate priority. You can set priority by using ThreadPriority enumerator. Possible values are AboveNormal, Normal, BelowNormal, Lowest and Highest.

Thread can be communicate with other threads by using functionality of Monitor class. The class provides signalling methods Pulse and PulseAll that notify other threads.

1.9CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

Answer – 1: C

The default priority for current thread is Normal

Answer – 2: B

System.Threading namespace supports multithreading.

Check your Progress 2

Answer – 1: D

Start() method change thread state into running state.

Answer – 2: B

False – Sleep method suspend the current thread for given time period in milliseconds.

Check your Progress 3

Answer – 1: C

Sleep method suspend the current thread for given time period in milliseconds.

Check your Progress 4

Answer – 1: C

Average is not a thread priority value.

Check your Progress 5

Answer – 1: A

Pulse method of monitor class notify other threads. It works with lock block

1.10 FURTHER READING

- Chapter-23 Multithreaded Programming,Part One
  Herbert Schildt, C# 4.0: The Complete Reference, The McGraw-Hill Companies
- Threading (Managed Threading)
  https://docs.microsoft.com/en-us/dotnet/standard/threading/

1.11ASSIGNMENTS

- Implement multithreading to print odd and even numbers by two separate thread. Also communicate with this two thread and print series of numbers 1,2,3,4,5,6,7,8,9,10.

1.12ACTIVITIES

- **Activity-1**
  Make list of classes available in System.threading Namespace and study its properties and methods.
- **Activity-2**
  Try to use Wait and Pulse method of monitor class without lock block and note what type of output or error you got and why.

Unit 2: File I/O With Streams 2

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Create new files on file system
- Read from files
- Write on the files
- Manage directories on file system

2.2 INTRODUCTION

There are basically two ways to store data by software. One is on database and second is on file system. The .net framework provides vast number of classes to perform read write operation on database by using ADO.NET. For file read and write operation on file system .net framework provides bunch of classes in System.IO namespace. On file system we are perform operation related to managing directories and files.

Read and write operations are performed by using stream. Stream is sequence of characters or bytes used to split large file into chunk of bytes. .Net framework provides classes like FileStream, StreamReader and StreamWriter to perform read and write operation.

Files are stored inside of directory or on hierarchical path of directories. To manage directories, files and path .net framework provides Directory and directoryInfo, File and FileInfo classes.

2.3 STREAM CLASSES – FILESTREAM, STREAMREADER, STREAMWRITER

Stream is sequence of bytes or characters are accessed in sequence one at a time. .net framework provide base class Stream and many other derived classes from Stream class. Stream class is abstract class so you cannot make instance of Stream class. Stream class provides methods and properties to perform read and write operation on stream of bytes.

2.3.1 FileStream

FileStream class is available in System.IO namespace and the class provides stream for file read and write operations. The FileStream class is used to read from and write to binary data on the file. FileStream class has several number of constructors are there for different functionality. Following table provide information of constructors.

FileStream(IntPtr, FileAccess)	Initializes a new instance of the FileStream class for the specified file handle, with the specified read/write permission.
FileStream(IntPtr, FileAccess, Boolean)	Initializes a new instance of the FileStream class for the specified file handle, with the specified read/write permission and FileStream instance ownership.
FileStream(IntPtr, FileAccess, Boolean, Int32)	Initializes a new instance of the FileStream class for the specified file handle, with the specified read/write permission, FileStream instance ownership, and buffer size.
FileStream(IntPtr, FileAccess, Boolean, Int32, Boolean)	Initializes a new instance of the FileStream class for the specified file handle, with the specified read/write permission, FileStream instance ownership, buffer size, and synchronous or asynchronous state.
FileStream(SafeFileHandle, FileAccess)	Initializes a new instance of the FileStream class for the specified file handle, with the specified read/write permission.
FileStream(SafeFileHandle, FileAccess, Int32)	Initializes a new instance of the FileStream class for the specified

	file handle, with the specified read/write permission, and buffer size.
FileStream(SafeFileHandle, FileAccess, Int32, Boolean)	Initializes a new instance of the FileStream class for the specified file handle, with the specified read/write permission, buffer size, and synchronous or asynchronous state.
FileStream(String, FileMode)	Initializes a new instance of the FileStream class with the specified path and creation mode.
FileStream(String, FileMode, FileAccess)	Initializes a new instance of the FileStream class with the specified path, creation mode, and read/write permission.
FileStream(String, FileMode, FileAccess, FileShare)	Initializes a new instance of the FileStream class with the specified path, creation mode, read/write permission, and sharing permission.
FileStream(String, FileMode, FileAccess, FileShare, Int32)	Initializes a new instance of the FileStream class with the specified path, creation mode, read/write and sharing permission, and buffer size.
FileStream(String, FileMode, FileAccess, FileShare, Int32, Boolean)	Initializes a new instance of the FileStream class with the specified path, creation mode, read/write and sharing permission, buffer size, and synchronous or asynchronous state.
FileStream(String, FileMode, FileAccess, FileShare, Int32, FileOptions)	Initializes a new instance of the FileStream class with the specified path, creation mode, read/write and sharing permission, the access other FileStreams can have to the same file, the buffer size, and additional file

	options.
FileStream(String, FileMode, FileSystemRights, FileShare, Int32, FileOptions)	Initializes a new instance of the FileStream class with the specified path, creation mode, access rights and sharing permission, the buffer size, and additional file options.
FileStream(String, FileMode, FileSystemRights, FileShare, Int32, FileOptions, FileSecurity)	Initializes a new instance of the FileStream class with the specified path, creation mode, access rights and sharing permission, the buffer size, additional file options, access control and audit security.

Table 2.1List of FileStream Constructor(Source : https://docs.microsoft.com)

FileStream class provide following list of properties.

CanRead	Gets a value that indicates whether the current stream supports reading.
CanSeek	Gets a value that indicates whether the current stream supports seeking.
CanTimeout	Gets a value that determines whether the current stream can time out. (Inherited from Stream)
CanWrite	Gets a value that indicates whether the current stream supports writing.
Handle	Gets the operating system file handle for the file that the current FileStream object encapsulates.
IsAsync	Gets a value that indicates whether the FileStream was opened asynchronously or synchronously.
Length	Gets the length in bytes of the stream.
Name	Gets the absolute path of the file opened in the FileStream.
Position	Gets or sets the current position of this stream.
ReadTimeout	Gets or sets a value, in milliseconds, that determines how long the stream will attempt to read before timing out.

	(Inherited from Stream)
SafeFileHandle	Gets a SafeFileHandle object that represents the operating system file handle for the file that the current FileStream object encapsulates.
WriteTimeout	Gets or sets a value, in milliseconds, that determines how long the stream will attempt to write before timing out. (Inherited from Stream)

Table 2.2 Properties of FileStream Class(Source : https://docs.microsoft.com)

The FileStream class provides following methods for read and write operation on file.

BeginRead(Byte[], Int32, Int32, AsyncCallback, Object)	Begins an asynchronous read operation
BeginWrite(Byte[], Int32, Int32, AsyncCallback, Object)	Begins an asynchronous write operation.
EndRead(IAsyncResult)	Waits for the pending asynchronous read operation to complete. (Consider using ReadAsync(Byte[], Int32, Int32, CancellationToken) instead.)
EndWrite(IAsyncResult)	Ends an asynchronous write operation and blocks until the I/O operation is complete. (Consider using WriteAsync(Byte[], Int32, Int32, CancellationToken) instead.)
Flush()	Clears buffers for this stream and causes any buffered data to be written to the file.
Flush(Boolean)	Clears buffers for this stream and causes any buffered data to be written to the file, and also clears all intermediate file buffers.
Lock(Int64, Int64)	Prevents other processes from reading

	from or writing to the FileStream.
Read(Byte[], Int32, Int32)	Reads a block of bytes from the stream and writes the data in a given buffer.
ReadAsync(Byte[], Int32, Int32)	Asynchronously reads a sequence of bytes from the current stream and advances the position within the stream by the number of bytes read. (Inherited from Stream)
ReadByte()	Reads a byte from the file and advances the read position one byte.
Seek(Int64, SeekOrigin)	Sets the current position of this stream to the given value.
SetLength(Int64)	Sets the length of this stream to the given value.
Unlock(Int64, Int64)	Allows access by other processes to all or part of a file that was previously locked.
Write(Byte[], Int32, Int32)	Writes a block of bytes to the file stream.
WriteAsync(Byte[], Int32, Int32)	Asynchronously writes a sequence of bytes to the current stream and advances the current position within this stream by the number of bytes written. (Inherited from Stream)
WriteAsync(Byte[], Int32, Int32, CancellationToken)	Asynchronously writes a sequence of bytes to the current stream, advances the current position within this stream by the number of bytes written, and monitors cancellation requests.

WriteByte(Byte)	Writes a byte to the current position in the file stream.

Table 2.3 Methods of FileStream Class (Source :https://docs.microsoft.com)

Lets instantiate FileStream class with constructor FileStream(String, FileMode, FileAccess). This constructor take three arguments.

First argument is string type and take file name with full path. For example "D:\DemoFolder\Demo.txt".

Second argument is FileMode. FileMode is enumerator and allow you to select any one option from available file modes. FileMode enumerator provides following values.

FileMode.Append	Open the file if exists and seek at end of file. If file not exists create new file.
FileMode.Create	Create new file. If file exists overwrite existing file.
FileMode.CreateNew	Create new file. If file exists overwrite IOException thrown
FileMode.Open	Open existing file. If file not found FileNotFoundException thrown
FileMode.OpenOrCreate	Open file if exists else create new file
FileMode.Truncate	Open the file and truncate to file size zero byte(Delete content of the file)

Table 2.4 FileMode enumerator

Third argument is FileAccess. FileAccess is also enumerator used to open file for read operation or write operation. FileAccess enumerator provides following values.

FileAccess.Read	Assign write access to the file. Data can be write on the file
FileAccess.Write	Assign read access to the file. Data can be read from the file
FileAccess.ReadWrite	Assign read and write access to the file.

	Data can be read from and write to the file

Table 2.5 FileAccess enumerator

Following example demonstrate you to open “BeReady.txt” file and write data on the file.

```
static void Main(string[] args)
    {
        FileStream fs = new FileStream("E:/BeReady.txt", FileMode.OpenOrCreate, FileAccess.Write);
string fileData = "Hello, This is test to write string on file Beready.txt";
        //Convert string into array of bytes
byte[] bytesData = Encoding.ASCII.GetBytes(fileData);
        //Write bytes to the file stream
fs.Write(bytesData, 0, bytesData.Length);
        //Clear buffer for the stream and write all buffered data to the file
fs.Flush();
        //Close the current stream and release any resources
fs.Close();
    }
```

This example create file stream with file mode open or create and with write access. Means if BeReady.txt file exists on “E:/” drive than open it else create new. To write on the file first need to create string and convert the string into array of bytes by using Encoding.ASCII.GetBytes(fileData). The write() method of the FileStream class write bytesto the file stream.

```
fs.Write(bytesData, 0, bytesData.Length);
```

Write method take three arguments bytes array, offset – index number from which start writing to the stream, length – how many bytes write to stream start from offset to given length.

fs.Flush();

Flush() method clear buffer for the stream and write all buffered data to the file.

fs.Close();

fs.Close() method close the current stream and release any resources.

Following code block shows how to read from BeReady.txt file.

```
static void Main(string[] args)
    {
      FileStream fs = new FileStream("E:/BeReady.txt", FileMode.Open,
      FileAccess.Read);
byte[] BytesData = new byte[fs.Length];
int result = fs.Read(BytesData, 0, BytesData.Length);
string str = Encoding.ASCII.GetString(BytesData);
Console.WriteLine("The information on File BeReady.txt -");
Console.WriteLine(str);
fs.Close();
Console.ReadLine();
    }
```

OUTPUT:

```
The information on File BeReady.txt –
Read Bytes = 55
Hello, This is test to write string on file Beready.txt
```

Above example create file stream for read operation. FileStream class Read() method reads bytes from file stream and add to the byte array. Read method return number of bytes read from file stream.

We need to convert bytes into string by using Encoding.ASCII.GetString(BytesData) method.

Check your Progress 1

1. _____ is not FileMode.
   E. Create
   F. Open
   G. Truncate
   H. Write
2. _____ method converts string into array of bytes.
   E. Read()
   F. Write()
   G. GetBytes()
   H. None of these

2.3.2 StreamReader

The StreamReader class is used to read from text file. As the use of StreamReader class is prefix to read text file so use of StreamReader class is very easy as compare to FileStream where you need to open or create file with specific file access mode. Also you need to use ASCII, UTF8, UTF16 etc. encoding for text file to read or write operation by using FileStream class. Here in StreamReader class only did read operation on text files with multiple options.

StreamReader class can be used to read from another stream. This class can be instantiated by using several constructor as per your requirement. You can directly make instance of StreamReader class to read from text file or you can use FileInfo create StreamReader instance. FileStream class help you to set file share permission while StreamReader not offer file share permission.

Following is list of few important constructors of the StreamReader class.

StreamReader(Stream)	Initializes a new instance of the StreamReader class for the specified stream.
StreamReader(Stream,	Initializes a new instance of the StreamReader class

Boolean)	for the specified stream, with the specified byte order mark detection option.
StreamReader(Stream, Encoding)	Initializes a new instance of the StreamReader class for the specified stream, with the specified character encoding.
StreamReader(String)	Initializes a new instance of the StreamReader class for the specified file name.
StreamReader(String, Boolean)	Initializes a new instance of the StreamReader class for the specified file name, with the specified byte order mark detection option.
StreamReader(String, Encoding)	Initializes a new instance of the StreamReader class for the specified file name, with the specified character encoding.

Table 2.6 StreamReader Constructor (Source: https://docs.microsoft.com)

Properties

BaseStream	Returns the underlying stream.
CurrentEncoding	Gets the current character encoding that the current StreamReader object is using.
EndOfStream	Gets a value that indicates whether the current stream position is at the end of the stream.

Table 2.7 StreamReader Properties (Source: https://docs.microsoft.com)

Following is the list of few important methods of the StreamReader class.

Close()	Closes the StreamReader object and the underlying stream, and releases any system resources associated with the reader.
DiscardBufferedData()	Clears the internal buffer.
Dispose()	Releases all resources used by the TextReader object. (Inherited from TextReader)
Peek()	Returns the next available character but does not consume it.
Read()	Reads the next character from the input stream and advances the character position by one character.
Read(Char[], Int32, Int32)	Reads a specified maximum of characters from the current stream into a buffer, beginning at the specified index.
ReadBlock(Char[], Int32, Int32)	Reads a specified maximum number of characters from the current stream and writes the data to a buffer, beginning at the specified index.
ReadLine()	Reads a line of characters from the current stream and returns the data as a string.

ReadToEnd()	Reads all characters from the current position to the end of the stream.

Table 2.8 StreamReader Methods (Source: https://docs.microsoft.com)

Following example shows you read operation from “BeReady.txt” file. Read the code and compare with the read operation of FileStream class yourself.

```
static void Main(string[] args)
      {
  StreamReader streamReader = new StreamReader(@"E:\BeReady.txt");
int i = 0;
while (!streamReader.EndOfStream)
          {
i++;
Console.WriteLine("Line No -" + i);
Console.WriteLine(streamReader.ReadLine());
          }
streamReader.Close();
Console.ReadLine();
      }
```

OUTPUT

```
Line No -1
Hello, This is test to write string on file Beready.txt by using FileStream.
Line No -2


Line No -3
This is test of reading from file Beready.txt by using StreamReader.
```

This example perform read operation on BeReady.txt file. Create and few lines on this file before executing the previous code. Above code use EndOfStream property to check read pointer at end of stream or not. If pointer is note at end of stream while loop continue read line by line from BeReady.txt file by using StreamReader's ReadLine() method. This method return string. Always close the stream before move to perform other operation in the application so other resources can use this file.

Check your Progress2

1. StreamReader is able to read character by character from text file.
   A. True
   B. False
2. You can make instance of StreamReader class by using _____ .
   A. DirectoryInfo
   B. FileStream
   C. String
   D. None of the above

2.3.3 StreamWriter

StreamWriter class used to write to the text file or another stream. It works almost same as StreamReader class to perform write operation. This class provides facility to write text in specific encoding.

Following is list of few important constructors of the StreamWriter class.

StreamWriter(Stream)	Initializes a new instance of the StreamWriter class for the specified stream by using UTF-8 encoding and the default buffer size.
StreamWriter(Stream, Encoding)	Initializes a new instance of the StreamWriter class for the specified stream by using the specified encoding and the default buffer size.

StreamWriter(Stream, Encoding, Int32)	Initializes a new instance of the StreamWriter class for the specified stream by using the specified encoding and buffer size.
StreamWriter(String)	Initializes a new instance of the StreamWriter class for the specified file by using the default encoding and buffer size.
StreamWriter(String, Boolean)	Initializes a new instance of the StreamWriter class for the specified file by using the default encoding and buffer size. If the file exists, it can be either overwritten or appended to. If the file does not exist, this constructor creates a new file.
StreamWriter(String, Boolean, Encoding)	Initializes a new instance of the StreamWriter class for the specified file by using the specified encoding and default buffer size. If the file exists, it can be either overwritten or appended to. If the file does not exist, this constructor creates a new file.

Table 2.9 StreamWriter Constructors (Source: https://docs.microsoft.com)

Properties

AutoFlush	Gets or sets a value indicating whether the StreamWriter will flush its buffer to the underlying stream after every call to Write(Char).
BaseStream	Gets the underlying stream that interfaces with a backing store.
Encoding	Gets the Encoding in which the output is written.

NewLine	Gets or sets the line terminator string used by the current TextWriter. (Inherited from TextWriter)

Table 2.10 StreamWriter Properties (Source: https://docs.microsoft.com)

Following is list of few important methods of the StreamWriter class.

Close()	Closes the current StreamWriter object and the underlying stream.
Dispose()	Releases all resources used by the TextWriter object. (Inherited from TextWriter)
Flush()	Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
Write(String)	Writes a string to the stream. Write method can also be used to write text representation of any type.
WriteLine()	Writes a line terminator to the text stream. (Inherited from TextWriter)
WriteLine(String)	Writes a string to the stream witha line terminator to the text stream. WriteLine method can also be used to write text representation of any type followed by line terminator.

Table 2.10 StreamWriter Methods (Source: https://docs.microsoft.com)

Following example shows you how to write to the file.

```
static void Main(string[] args)
    {
StreamWriter streamWriter = new StreamWriter(@"E:\BeReady.txt");
```

```
streamWriter.Write("This is Write operation");
streamWriter.WriteLine("done by using StreamWriter class");
streamWriter.Flush();

streamWriter.WriteLine("You can write any type using Write method.");
streamWriter.WriteLine(DateTime.Now);
streamWriter.Flush();
Console.WriteLine("File write operation completed successfully");
Console.ReadLine();
    }
```

OUTPUT

File write operation completed successfully

This example open BeReady.txt file if exists and overwrite or create new one on given path and write on the file. You can check the file on given location.

Wrie() and WriteLine() methods write text on the stream and Flush() methods apply changes to physical file or stream and clear all buffer data. Do not forget to close the stream after completion of write operation.

To append the existing file use following constructor in above example.

```
StreamWriter streamWriter = new StreamWriter(@"E:\BeReady.txt", true);
```

To write by using specific encoding use following constructore.This constructor take three argument

1. File Path with file name as string
2. Append (True/False)
3. Encoding (ASCII,UTF8,UTF16 etc..)

```
StreamWriter streamWriter = new StreamWriter(@"E:\BeReady.txt", false, Encoding.ASCII);
```

Check your Progress3

1. Is StreamWriter class's Write() method able to write DateTime type on the file?

A. Yes

B. No

2. Flush method is Clears all buffers for the current writer.

A. True

B. False

2.4 DIRECTORY AND DIRECTORYINFO

The Directory and DirectoryInfo classes are represent folder on the file system. Directory class is only contains static methods and DirectoryInfo class contains all the methods of Directory class, constructors and properties. To use DirectoryInfo class you need to make instance of the DirectoryInfo class.

2.4.1 Directory

The Directory class is typically performoperations like copying, moving, renaming, creating, and deleting directories.

Directory class has bunch of static methods to create new directory, delete, copy, rename or move directory. You can also get list of files and sub directories of selected directory by using enumerable collection.

Following is list of important static methods of Directory class.

CreateDirectory(String)	Creates all directories and subdirectories in the specified path unless they already exist.
Delete(String)	Deletes an empty directory from a

	specified path.
Delete(String, Boolean)	Deletes the specified directory and, if indicated, any subdirectories and files in the directory.
EnumerateDirectories(String)	Returns an enumerable collection of directory names in a specified path.
EnumerateDirectories(String, String)	Returns an enumerable collection of directory names that match a search pattern in a specified path.
EnumerateDirectories(String, String, SearchOption)	Returns an enumerable collection of directory names that match a search pattern in a specified path, and optionally searches subdirectories.
EnumerateFiles(String, String, SearchOption)	Returns an enumerable collection of file names that match a search pattern in a specified path, and optionally searches subdirectories.
EnumerateFiles(String)	Returns an enumerable collection of file names in a specified path.
EnumerateFiles(String, String)	Returns an enumerable collection of file names that match a search pattern in a specified path.
EnumerateFileSystemEntries(String)	Returns an enumerable collection of file names and directory names in a

	specified path.
Exists(String)	Determines whether the given path refers to an existing directory on disk.
GetAccessControl(String)	Gets a DirectorySecurity object that encapsulates the access control list (ACL) entries for a specified directory.
GetCreationTime(String)	Gets the creation date and time of a directory.
GetCreationTimeUtc(String)	Gets the creation date and time, in Coordinated Universal Time (UTC) format, of a directory.
GetCurrentDirectory()	Gets the current working directory of the application.
GetDirectories(String, String, SearchOption)	Returns the names of the subdirectories (including their paths) that match the specified search pattern in the specified directory, and optionally searches subdirectories.
GetDirectories(String)	Returns the names of subdirectories (including their paths) in the specified directory.
GetDirectories(String, String)	Returns the names of subdirectories (including their paths) that match the specified search pattern in the

	specified directory.
GetDirectoryRoot(String)	Returns the volume information, root information, or both for the specified path.
GetFiles(String)	Returns the names of files (including their paths) in the specified directory.
GetFiles(String, String)	Returns the names of files (including their paths) that match the specified search pattern in the specified directory.
GetFiles(String, String, SearchOption)	Returns the names of files (including their paths) that match the specified search pattern in the specified directory, using a value to determine whether to search subdirectories.
GetFileSystemEntries(String)	Returns the names of all files and subdirectories in a specified path.
GetLastAccessTime(String)	Returns the date and time the specified file or directory was last accessed.
GetLastAccessTimeUtc(String)	Returns the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last accessed.

GetLastWriteTime(String)	Returns the date and time the specified file or directory was last written to.
GetLastWriteTimeUtc(String)	Returns the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last written to.
GetLogicalDrives()	Retrieves the names of the logical drives on this computer in the form "<drive letter>:\".
GetParent(String)	Retrieves the parent directory of the specified path, including both absolute and relative paths.
Move(String, String)	Moves a file or a directory and its contents to a new location.
SetCreationTime(String, DateTime)	Sets the creation date and time for the specified file or directory.
SetCreationTimeUtc(String, DateTime)	Sets the creation date and time, in Coordinated Universal Time (UTC) format, for the specified file or directory.
SetCurrentDirectory(String)	Sets the application's current working directory to the specified directory.
SetLastAccessTime(String,	Sets the date and time the specified

DateTime)	file or directory was last accessed.
SetLastAccessTimeUtc(String, DateTime)	Sets the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last accessed.
SetLastWriteTime(String, DateTime)	Sets the date and time a directory was last written to.
SetLastWriteTimeUtc(String, DateTime)	Sets the date and time, in Coordinated Universal Time (UTC) format, that a directory was last written to.

Table 2.11 Static Methods of Directory class (Source: https://docs.microsoft.com)

Following example shows how to use Directory class to manage directories and get list of subdirectories.

```
static void Main(string[] args)
    {
        //Create new directory on G: drive

Directory.CreateDirectory("G:\\Courses");
Console.WriteLine("Directory created...");

        //Create sub directories of Courses
Directory.CreateDirectory("G:\\Courses\\MCA");
Console.WriteLine("Directory created...");

Directory.CreateDirectory("G:\\Courses\\M.Sc. IT");
```

```
Console.WriteLine("Directory created...");


Directory.CreateDirectory("G:\\Courses\\B. Sc. IT");
Console.WriteLine("Directory created...");


        //Print name of sub directories of Courses directory
String[] DirNames = Directory.GetDirectories("G:\\Courses");
Console.WriteLine("Sub directories of Courses directory are... ");
foreach(var dir in DirNames )
        {
Console.WriteLine(dir);
        }
Console.ReadLine();
    }
```

OUTPUT

```
Directory created...
Directory created...
Directory created...
Directory created...
Sub directories of Courses directory are...
G:\Courses\B. Sc. IT
G:\Courses\M.Sc. IT
G:\Courses\MCA
```

In above example by using Directory.CreateDirectory() method create new directory. CreateDirectory() method creates new directory if directory with specified name and path not exists.

Directory.GetDirectories("G:\\Courses") methods return collections of sub directories of specified directory as array of string.

You can also get list of files from specified directory. To test this first create one or two files in M.Sc. IT directory and execute following code.

```
//Print name of files of M.Sc. IT directory
String[] FileNames = Directory.GetFiles("G:\\Courses\\M.Sc. IT");
Console.WriteLine("Files of directory are... ");
foreach (var file in FileNames)
        {
Console.WriteLine(file);
        }
Console.ReadLine();
```

OUTPUT

```
Files of directory are...
G:\Courses\M.Sc. IT\M.Sc(IT) Doc File.docx
```

2.4.2 DirectoryInfo

DirectoryInfo provides instance methods for creating, moving, and enumerating through directories and subdirectories. You cannot inherit DirectoryInfo class.

DirectoryInfo class have advantages over Directory class if you are perform many operations on same directory. Directory class perform security check every time you use its method. While in DirectoryInfo class security check performed only when you make instance of DirectoryInfo.

DirectoryInfo class have only one constructor.

```
DirectoryInfo dirInfo = new DirectoryInfo("G:\\Courses");
```

Following is list of properties.

Attributes	Gets or sets the attributes for the current file or directory. (Inherited from FileSystemInfo)
CreationTime	Gets or sets the creation time of the current file or directory. (Inherited from FileSystemInfo)
CreationTimeUtc	Gets or sets the creation time, in coordinated universal time (UTC), of the current file or directory. (Inherited from FileSystemInfo)
Exists	Gets a value indicating whether the directory exists.
Extension	Gets the string representing the extension part of the file. (Inherited from FileSystemInfo)
FullName	Gets the full path of the directory.
LastAccessTime	Gets or sets the time the current file or directory was last accessed. (Inherited from FileSystemInfo)
LastAccessTimeUtc	Gets or sets the time, in coordinated universal time (UTC), that the current file or directory was last accessed.

	(Inherited from FileSystemInfo)
LastWriteTime	Gets or sets the time when the current file or directory was last written to. (Inherited from FileSystemInfo)
LastWriteTimeUtc	Gets or sets the time, in coordinated universal time (UTC), when the current file or directory was last written to. (Inherited from FileSystemInfo)
Name	Gets the name of this DirectoryInfo instance.
Parent	Gets the parent directory of a specified subdirectory.
Root	Gets the root portion of the directory.

Table 2.12 Properties of DirectoryInfo class (Source: https://docs.microsoft.com)

Following is list of important methods of DirectoryInfo class

Create()	Creates a directory.
CreateSubdirectory(String)	Creates a subdirectory or subdirectories on the specified path. The specified path can be relative to this instance of the DirectoryInfo class.
Delete()	Deletes this DirectoryInfo if it is empty.

Delete(Boolean)	Deletes this instance of a DirectoryInfo, specifying whether to delete subdirectories and files.
EnumerateDirectories()	Returns an enumerable collection of directory information in the current directory.
EnumerateDirectories(String)	Returns an enumerable collection of directory information that matches a specified search pattern.
EnumerateDirectories(String, SearchOption)	Returns an enumerable collection of directory information that matches a specified search pattern and search subdirectory option.
EnumerateFiles()	Returns an enumerable collection of file information in the current directory.
EnumerateFiles(String)	Returns an enumerable collection of file information that matches a search pattern.
EnumerateFiles(String, SearchOption)	Returns an enumerable collection of file information that matches a specified search pattern and search subdirectory option.

EnumerateFileSystemInfos()	Returns an enumerable collection of file system information in the current directory.
EnumerateFileSystemInfos(String)	Returns an enumerable collection of file system information that matches a specified search pattern.
EnumerateFileSystemInfos(String, SearchOption)	Returns an enumerable collection of file system information that matches a specified search pattern and search subdirectory option.
GetAccessControl()	Gets a DirectorySecurity object that encapsulates the access control list (ACL) entries for the directory described by the current DirectoryInfo object.
GetDirectories()	Returns the subdirectories of the current directory.
GetDirectories(String)	Returns an array of directories in the current DirectoryInfo matching the given search criteria.
GetDirectories(String, SearchOption)	Returns an array of directories in the current DirectoryInfo matching the given search criteria and using a value to determine whether to search

	subdirectories.
GetFiles()	Returns a file list from the current directory.
GetFiles(String)	Returns a file list from the current directory matching the given search pattern.
GetFiles(String, SearchOption)	Returns a file list from the current directory matching the given search pattern and using a value to determine whether to search subdirectories.
GetFileSystemInfos()	Returns an array of strongly typed FileSystemInfo entries representing all the files and subdirectories in a directory.
GetFileSystemInfos(String)	Retrieves an array of strongly typed FileSystemInfo objects representing the files and subdirectories that match the specified search criteria.
GetFileSystemInfos(String, SearchOption)	Retrieves an array of FileSystemInfo objects that represent the files and subdirectories matching the specified search criteria.

MoveTo(String)	Moves a DirectoryInfo instance and its contents to a new path.
Refresh()	Refreshes the state of the object. (Inherited from FileSystemInfo)

Table-2.13 Methods of DirectoryInfo (Source:https://docs.microsoft.com)

Following example demonstrate use of the DirectoryInfo class.

```
DirectoryInfo dirInfo = new DirectoryInfo("G:\\Courses\\M.Sc. IT Subjects");
        //Create Folder on file system
dirInfo.Create();
Console.WriteLine("Directory created...");

        //Print name of sub directories of Courses directory
String[] DirNames = Directory.GetDirectories("G:\\Courses");
Console.WriteLine("Sub directories of Courses directory are... ");
foreach (var dir in DirNames)
        {
Console.WriteLine(dir);
        }
Console.ReadLine();

        //Move directory info instance  from G:\Courses\M.Sc. IT Subjects to G:\Courses\M.Sc. IT directory

try
        {
dirInfo.MoveTo("G:\\Courses\\M.Sc. IT\\M.Sc. IT Subjects");
Console.WriteLine("DirectoryInfo moved to G:\\Courses\\M.Sc. IT\\ ");
        }
catch(IOException ex)
        {
Console.WriteLine(ex.Message);
        }
```

```
foreach(DirectoryInfo dir in dirInfo.Parent.GetDirectories())
        {
Console.WriteLine(dir.FullName);
        }

Console.ReadLine();

        //Print name of sub directories of Courses directory
Console.WriteLine("Root of the directory is " + dirInfo.Root.ToString());
Console.WriteLine("Parent of the directory is " + dirInfo.Parent.ToString());
Console.ReadLine();

    }
```

OUTPUT

```
Directory created...
Sub directories of Courses directory are...
G:\Courses\B. Sc. IT
G:\Courses\M.Sc. IT
G:\Courses\M.Sc. IT Subjects
G:\Courses\MCA

DirectoryInfo moved to G:\Courses\M.Sc. IT\
G:\Courses\M.Sc. IT\M.Sc. IT Subjects

Root of the directory is G:\
Parent of the directory is M.Sc. IT
```

Above code first make instance of DirectoryInfo class with path “G:\Courses\M.Sc. IT Subjects”.

```
DirectoryInfo dirInfo = new DirectoryInfo("G:\\Courses\\M.Sc. IT Subjects");
```

Next step is to create supplied directory by using Create() method.

```
dirInfo.Create();
```

To check directory is created or not print all sub directories of “G:\Courses” directory by using Directory class.

By using directory class print all sub directories of “G:\Courses” directory.

```
        //Print name of sub directories of Courses directory
String[] DirNames = Directory.GetDirectories("G:\\Courses");
Console.WriteLine("Sub directories of Courses directory are... ");
foreach (var dir in DirNames)
        {
Console.WriteLine(dir);
        }
Console.ReadLine();
```

Now suppose we want to move our instance from “M.Sc. IT Subjects” directory to “G:\ Courses\M.Sc. IT\” directory. Use MoveTo() method with new directory path.

```
try
        {
dirInfo.MoveTo("G:\\Courses\\M.Sc. IT\\M.Sc. IT Subjects");
Console.WriteLine("DirectoryInfo moved to G:\\Courses\\M.Sc. IT\\ ");
        }
catch(IOException ex)
        {
Console.WriteLine(ex.Message);
        }
```

MoveTo() methods move all sub directories of instantiated directories to new location and remove instantiated directory. It assign reference of newly created directory to existing object of DirectoryInfo class.

By using Parent and Root property you can get name of parent directory and drive name.

There are many other properties and methods given in table no – 2.12 and 2.13.

Check your Progress 4

1. Can we use both the classes Directory and DirectoryInfo to move directory from one location to another location?

A. Yes

B. No

2. Is GetParent() methods of Directory class is similar to Parent properties of DirectoryInfo instance?

C. Yes

D. No

2.5 FILE AND FILEINFO

File and FileInfo classes are used to perform create, open, read, write, copy and move operation on single file. File class provide static methods while FileInfo provides instance methods. In many cases File class static methods faster for single operation on file. FileInfo class is basically more used to perform multiple operations on specific file. File.Exist() method is faster than FileInfo instance's Exist() method.

2.5.1 File

The File class provides static methods to perform create, open, read, write, copy, delete and move operations. As all methods are static so no need to make instance of File class to use these methods.

Following table describe important static methods of File class.

AppendAllLines(String, IEnumerable<String>)	Appends lines to a file, and then closes the file. If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.

AppendAllText(String, String)	Opens a file, appends the specified string to the file, and then closes the file. If the file does not exist, this method creates a file, writes the specified string to the file, then closes the file.
AppendAllText(String, String, Encoding)	Appends the specified string to the file using the specified encoding, creating the file if it does not already exist.
AppendText(String)	Creates a StreamWriter that appends UTF-8 encoded text to an existing file, or to a new file if the specified file does not exist.
Copy(String, String)	Copies an existing file to a new file. Overwriting a file of the same name is not allowed.
Copy(String, String, Boolean)	Copies an existing file to a new file. Overwriting a file of the same name is allowed.
Create(String)	Creates or overwrites a file in the specified path.
CreateText(String)	Creates or opens a file for writing UTF-8 encoded text. If the file already exists, its contents are overwritten.

Decrypt(String)	Decrypts a file that was encrypted by the current account using the Encrypt(String) method.
Delete(String)	Deletes the specified file.
Encrypt(String)	Encrypts a file so that only the account used to encrypt the file can decrypt it.
Exists(String)	Determines whether the specified file exists.
GetAttributes(String)	Gets the FileAttributes of the file on the path.
GetCreationTime(String)	Returns the creation date and time of the specified file or directory.
GetLastAccessTime(String)	Returns the date and time the specified file or directory was last accessed.
GetLastWriteTime(String)	Returns the date and time the specified file or directory was last written to.
Move(String, String)	Moves a specified file to a new location, providing the option to specify a new file name.
Open(String, FileMode)	Opens a FileStream on the specified path with read/write access with no sharing.

Open(String, FileMode, FileAccess)	Opens a FileStream on the specified path, with the specified mode and access with no sharing.
Open(String, FileMode, FileAccess, FileShare)	Opens a FileStream on the specified path, having the specified mode with read, write, or read/write access and the specified sharing option.
OpenRead(String)	Opens an existing file for reading.
OpenText(String)	Opens an existing UTF-8 encoded text file for reading.
OpenWrite(String)	Opens an existing file or creates a new file for writing.
ReadAllBytes(String)	Opens a binary file, reads the contents of the file into a byte array, and then closes the file.
ReadAllLines(String)	Opens a text file, reads all lines of the file, and then closes the file.
ReadAllText(String)	Opens a text file, reads all the text in the file, and then closes the file.
ReadLines(String)	Reads the lines of a file.

Replace(String, String, String)	Replaces the contents of a specified file with the contents of another file, deleting the original file, and creating a backup of the replaced file.
Replace(String, String, String, Boolean)	Replaces the contents of a specified file with the contents of another file, deleting the original file, and creating a backup of the replaced file and optionally ignores merge errors.
WriteAllBytes(String, Byte[])	Creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is overwritten.
WriteAllLines(String, String[])	Creates a new file, write the specified string array to the file, and then closes the file.
WriteAllText(String, String)	Creates a new file, writes the specified string to the file, and then closes the file. If the target file already exists, it is overwritten.

Table-2.14 Static Methods of File (Source:https://docs.microsoft.com)

In Table 2.14 many methods have several overload methods to perform same operation with different arguments.

Following example check given text file is exist or not. If file is exist than open for read operation and if not exist than create new file for write operation. In same example after writing to the file perform read operation and display file content on console.

```
static void Main(string[] args)
      {
         String FilePath = "G:\\Courses\\FileMethods.txt";

if(!File.Exists(FilePath))
         {
Console.WriteLine("New File created with name - " + FilePath);
String[] fileContent = new String[5];
fileContent[0] = "Following are few methods of File class to open file";
fileContent[1] = "File.Open";
fileContent[2] = "File.OpenRead";
fileContent[3] = "File.OpenText";
fileContent[4] = "File.OpenWrite";
            //Create new file, write lines and close the file
File.WriteAllLines(FilePath, fileContent);
         }

Console.WriteLine("Content of " + FilePath + " file");
            //open, read and close an existing text file.
String[] fileTextContent = File.ReadAllLines(FilePath);
foreach(string s in fileTextContent)
            {
               Console.WriteLine(s);
            }

Console.ReadLine();
      }
```

OUTPUT

New File created with name - G:\Courses\FileMethods.txt

Content of G:\Courses\FileMethods.txt file

Following are few methods of File class to open file

File.Open

File.OpenRead

File.OpenText

File.OpenWrite

This example demonstrate the way to create file and write multiple lines by using WriteAllLines() method if file is not exists. This example also demonstrate read operation on same file by using ReadAllLines() method.

The File class has many methods that open file for read or write operation. These methods create FileStream or StreamWriter or StreamReader instance. You can use previously learned methods of respective instance to perform operations on stream.

2.5.2 FileInfo

FileInfo class is used in case of you want to perform several operations on same file. It provide better performance as compare to File class as security check performed only once when instance of FileInfo class created.

FileInfo class provides properties and instance methods for create, copy, delete, open and move operation on files. By using FileInfo class you can create object of FileStream objects. This class also part of System.IO namespace. FileInfo class is not inheritable. You can use FileAccess, FileMode and FileSharing options while opening file using FileInfo class.

FileInfo class has only one constructor and it take file path string as argument.

FileInfo fobj = new FileInfo("G:\\Courses\\TestFile.txt");

Properties of FileInfo class

Attributes	Gets or sets the attributes for the current file or directory. (Inherited from FileSystemInfo)
CreationTime	Gets or sets the creation time of the current file or directory. (Inherited from FileSystemInfo)
CreationTimeUtc	Gets or sets the creation time, in coordinated universal time (UTC), of the current file or directory. (Inherited from FileSystemInfo)
Directory	Gets an instance of the parent directory.
DirectoryName	Gets a string representing the directory's full path.
Exists	Gets a value indicating whether a file exists.
Extension	Gets the string representing the extension part of the file. (Inherited from FileSystemInfo)
FullName	Gets the full path of the directory or file. (Inherited from FileSystemInfo)
IsReadOnly	Gets or sets a value that determines if the current file is read only.
LastAccessTime	Gets or sets the time the current file or directory was last accessed.

	(Inherited from FileSystemInfo)
LastAccessTimeUtc	Gets or sets the time, in coordinated universal time (UTC), that the current file or directory was last accessed. (Inherited from FileSystemInfo)
LastWriteTime	Gets or sets the time when the current file or directory was last written to. (Inherited from FileSystemInfo)
LastWriteTimeUtc	Gets or sets the time, in coordinated universal time (UTC), when the current file or directory was last written to. (Inherited from FileSystemInfo)
Length	Gets the size, in bytes, of the current file.
Name	Gets the name of the file.

Table-2.15 Properties of FileInfo class (Source: https://docs.microsoft.com)

Methods of FileInfo class

AppendText()	Creates a StreamWriter that appends text to the file represented by this instance of the FileInfo.
CopyTo(String)	Copies an existing file to a new file, disallowing the overwriting of an existing file.

CopyTo(String, Boolean)	Copies an existing file to a new file, allowing the overwriting of an existing file.
Create()	Creates a file.
CreateText()	Creates a StreamWriter that writes a new text file.
Decrypt()	Decrypts a file that was encrypted by the current account using the Encrypt() method.
Delete()	Permanently deletes a file.
Encrypt()	Encrypts a file so that only the account used to encrypt the file can decrypt it.
MoveTo(String)	Moves a specified file to a new location, providing the option to specify a new file name.
Open(FileMode)	Opens a file in the specified mode.
Open(FileMode, FileAccess)	Opens a file in the specified mode with read, write, or read/write access.
Open(FileMode, FileAccess, FileShare)	Opens a file in the specified mode with read, write, or read/write access and the specified sharing

	option.
OpenRead()	Creates a read-only FileStream.
OpenText()	Creates a StreamReader with UTF8 encoding that reads from an existing text file.
OpenWrite()	Creates a write-only FileStream.
Refresh()	Refreshes the state of the object. (Inherited from FileSystemInfo)
Replace(String, String)	Replaces the contents of a specified file with the file described by the current FileInfo object, deleting the original file, and creating a backup of the replaced file.
Replace(String, String, Boolean)	Replaces the contents of a specified file with the file described by the current FileInfo object, deleting the original file, and creating a backup of the replaced file. Also specifies whether to ignore merge errors.
ToString()	Returns the path as a string. Use the Name property for the full path.

Table-2.14 Instance Methods of FileInfo (Source: https://docs.microsoft.com)

Let’s open existing file - G:\Courses\FileMethods.txt and append the text of this file. Append option is live old content of the file as it is and add new content at the end of file content.

```
static void Main(string[] args)
      {
         String FilePath = "G:\\Courses\\FileMethods.txt";
          FileInfo FileObj = new FileInfo(FilePath);
          //Write on to file using StreamWriter
          StreamWriter sw = FileObj.AppendText();
sw.WriteLine("Following are few methods of FileInfo class to open file");
sw.WriteLine("Open(FileMode)");
sw.WriteLine("OpenRead()");
sw.WriteLine("OpenWrite()");
sw.WriteLine("OpenText()");
sw.Flush();
sw.Close();
          //Open to read from file using StreamReader
          StreamReader sr = FileObj.OpenText();
Console.WriteLine("Content of File - " + FilePath);
Console.WriteLine(sr.ReadToEnd());
sr.Close();

Console.ReadLine();
      }
```

OUTPUT:

```
Content of File - G:\Courses\FileMethods.txt
Following are few methods of File class to open file
```

File.Open

File.OpenRead

File.OpenText

File.OpenWrite

Following are few methods of FileInfo class to open file

Open(FileMode)

OpenRead()

OpenWrite()

OpenText()

Check your Progress 5

1. _____ property of FileInfo class used to get file extension.

A. GetExtension

B. SetExtension

C. Extension

D. None of Above

2. _____ is return type of Open() method of File class.

A. FileStream

B. StreamWriter

C. StreamReader

D. None of Above

3. FileInfo class provides methods to write on to file.

A. True

B. False

2.6PARSING PATHS

In windows and other operating system file system is used to access file or directory. Each operating system has specific format for path to access file or directory. Windows use following format to represent path of file or directory.

Drive letter followed by volume separator (:)

Example: C: , D:

Directory Separator (\)

Example: C:\TempDir\

Path can be location of directory or location of file. File can be identify by using extension. Extension is used to identify file type. Many operating system limit extension size three character but it can be different base on operating system to operating system.

File Extension Separator (.)

Example: C:\TempDir\MyFile.txt

Path can be relative or absolute. Absolute path start with volume or drive character. Contains drive separator, directory name and drive separator.

Example : C:\TempDir\

Relative path starts with directory name or with current directory.

Example : Subjects\SubjectNames.txt

.Net framework provide**static class Path** to parse file or directory path. Path class is part of System.IO namespace. Path class works with instance of string that contains path. Path class has following static methods.

ChangeExtension(String, String)	Changes the extension of a path string.
Combine(String[])	Combines an array of strings into a path.
Combine(String, String)	Combines two strings into a path.
Combine(String, String, String)	Combines three strings into a path.
Combine(String, String, String, String)	Combines four strings into a path.
GetDirectoryName(String)	Returns the directory information for the specified path string.
GetExtension(String)	Returns the extension (including the period ".") of the specified path string.
GetFileName(String)	Returns the file name and extension of the specified path string.
GetFileNameWithoutExtension(String)	Returns the file name of the specified path string without the extension.
GetFullPath(String)	Returns the absolute path for the specified path string.

GetInvalidFileNameChars()	Gets an array containing the characters that are not allowed in file names.
GetInvalidPathChars()	Gets an array containing the characters that are not allowed in path names.
GetPathRoot(String)	Gets the root directory information of the specified path.
GetRandomFileName()	Returns a random folder name or file name.
GetTempFileName()	Creates a uniquely named, zero-byte temporary file on disk and returns the full path of that file.
GetTempPath()	Returns the path of the current user's temporary folder.
HasExtension(String)	Determines whether a path includes a file name extension.
IsPathRooted(String)	Returns a value indicating whether the specified path string contains a root.

Table-2.15 Static Methods of Path class (Source: https://docs.microsoft.com)

Path class provides fields to set directory separator character, path separator character and volume separator character. All this fields have default values so only useful if platform specific separator you want to set.

AltDirectorySeparatorChar	Provides a platform-specific alternate character used to separate directory levels in a path string that reflects a hierarchical file system organization.
DirectorySeparatorChar	Provides a platform-specific character used to separate directory levels in a path string that reflects a hierarchical file system organization.
InvalidPathChars	Provides a platform-specific array of characters that cannot be specified in path string arguments passed to members of the Path class.
PathSeparator	A platform-specific separator character used to separate path strings in environment variables.
VolumeSeparatorChar	Provides a platform-specific volume separator character.

Table-2.16 Fields of Path class (Source: https://docs.microsoft.com)

Following example parse given path into Volume name, Full Path, File Name and File Extension.

```
static void Main(string[] args)

    {
string path = @"G:\Courses\M.Sc. IT\M.Sc. IT Subjects\Subjects.txt";
```

```
        //Parse Drive name from path
Console.WriteLine("Volume / Drive name :" + Path.GetPathRoot(path));
        //Parse direcory path from path
Console.WriteLine("Directory : " + Path.GetDirectoryName(path));
        //Parse file name from path
Console.WriteLine("File name : " + Path.GetFileName(path));
        //Parse extension of file
Console.WriteLine("Extension of File : " + Path.GetExtension(path));

        Console.ReadLine();
    }
```

OUTPUT

```
Volume / Drive name :G:\
Directory : G:\Courses\M.Sc. IT\M.Sc. IT Subjects
File name : Subjects.txt
Extension of File : .txt
```

Above example provides us drive name, directory name with full path, file name from path and extensions of file. This example not check for actually the directory and file exists on file system. Path class is useful to create valid path for file system.

Check your Progress 6

1. Path class performs operations on _____ instances that contain file or directory path information.

A. FileInfo

B. DirectoryInfo

C. String.

D. None of Above

2. _____ method return absolute path.

A. GetDirectoryName()

B. GetFullPath()

C. GetPathRoot()

D. None of Above

2.7LET US SUM UP

In this unit you are learn about managing files and directories on file system using different static and instance classes provided by System.IO name space. To perform read or write operation on file first you need to create stream. Stream is a intermediator between physical file and application.

There are three classes to create stream. FileStream, StreamWriter and StreamReader are these classes. FileStream works with all types file as it is works with bytes. It preferable when you are working with binary data.

StreamWriter and StreamReader is useful to works with text file with specific encoding or default encoding. To perform write operation on text file use StreamWriter class. To perform read operation use StreamReader class. Both classes provides many useful methods for write and read operation.

To create, delete, move or copy directory on file system System.IO namespace provides Directory and DirectoryInfo class. For single operation on the directory use static class Directory. For multiple operation on same directory use DirectoryInfo class.

To manage files on file system System.IO namespace provides static File class and FileInfo instance class. Same as Directory and DirectoryInfo class for single operation use File class and for multiple operation on same file use FileInfo class.

DirecoryInfo and FileInfo class provides better performance in case of multiple operation you want to perform on same directory and file respectively.

To parse path or create path .net framework provide static class Path. It is useful to validate the path, retrieve volume, directory, filename and file extension from given path as string.

The scope of this material is very limited but you can experiment yourself all the methods described in respective classes.

2.8CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Check your Progress 1

Answer – 1: D

Write is not a FileMode

Answer – 2: C

Encoding.ASCII.GetBytes(fileData)

Check your Progress 2

Answer – 1: A

Read() method read next character from stream.

Answer – 2: D

None of Above

Check your Progress 3

Answer – 1: A

YES, Write() method able to works with DateTime type.

Answer – 2: B

True, Flush() method clean all buffer data and writes to physical file.

Check your Progress 4

Answer – 1: A

Yes Directory class has Move() method and DirectoryInfo class has MoveTo() method.

Answer – 2: A

Yes, GetParent() of Directory class and Parent property of DirectoryInfo class return DirectoryInfo object

Check your Progress 5

Answer – 1: C

Extension property is used to get extension of file.

Answer – 2: A

The return type of Open() method is FileStream.

Answer – 3: B

False, To perform write operation you need to use StreamWriter or FileStream.

Check your Progress 6

Answer – 1: C

Path class works with instance of String that represent relative or absolute path.

Answer – 2: B

GetFullPath() method return absolute path.

2.9 FURTHER READING

- Chapter 24: Manipulating Files and the Registry
  Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner, Professional C# 2012 And .Net 4.5, Wrox Publication

- Chapter 14: Using IO
  Herbert Schildt, C# 4.0: The Complete Reference, The McGraw-Hill Companies

- System.IO Namespace
  https://docs.microsoft.com/en-us/dotnet/api/system.io?view=netframework-4.6.2

2.10ASSIGNMENTS

- Make method list of FileStream, StreamWriter, StreamReader with parameter and return type.

2.11ACTIVITIES

- **Activity-1**

  Develop console application that perform directory manipulation (Create, Delete, Copy, Move with sub directory and files) on file system using Directory and DirectoryInfo class.

યુનિવર્સિટી ગીત

સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ
સ્વાધ્યાયઃ પરમં તપઃ

શિક્ષણ, સંસ્કૃતિ, સદ્‌ભાવ, દિવ્યબોધનું ધામ
ડૉ. બાબાસાહેબ આંબેડકર ઓપન યુનિવર્સિટી નામ;
સૌને સૌની પાંખ મળે, ને સૌને સૌનું આભ,
દશે દિશામાં સ્મિત વહે હો દશે દિશે શુભ-લાભ.

અભણ રહી અજ્ઞાનના શાને, અંધકારને પીવો ?
કહે બુદ્ધ આંબેડકર કહે, તું થા તારો દીવો;
શારદીય અજવાળા પહોંચ્યાં ગુર્જર ગામે ગામ
ધ્રુવ તારકની જેમ ઝળહળે એકલવ્યની શાન.

સરસ્વતીના મયૂર તમારે ફળિયે આવી ગહેકે
અંધકારને હડસેલીને ઉજાસના ફૂલ મહેંકે;
બંધન નહીં કો સ્થાન સમયના જવું ન ઘરથી દૂર
ઘર આવી મા હરે શારદા દૈન્ય તિમિરના પૂર.

સંસ્કારોની સુગંધ મહેંકે, મન મંદિરને ધામે
સુખની ટપાલ પહોંચે સૌને પોતાને સરનામે;
સમાજ કેરે દરિયે હાંકી શિક્ષણ કેરું વહાણ,
આવો કરીયે આપણ સૌ
ભવ્ય રાષ્ટ્ર નિર્માણ...
દિવ્ય રાષ્ટ્ર નિર્માણ...
ભવ્ય રાષ્ટ્ર નિર્માણ

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY
(Established by Government of Gujarat)
'Jyotirmay' Parisar,
Sarkhej-Gandhinagar Highway, Chharodi, Ahmedabad-382 481
Website : www.baou.edu.in