Table of content

S.N	NAME	DATE	REMARKS
1)	Introduction of C#	2080-10-7	
2)	WAP to demonstrate method overriding in C#	2080-10-7	
3)	WAP to demonstrate method hiding in C#	2080-10-9	
4)	WAP to demonstrate constructor in C#	2080-10-23	
5)	WAP to demonstrate inheritance in C#	2080-11-2	
6)	WAP to demonstrate struct in C#	2080-11-10	
7)	WAP to demonstrate Enum in C#	2080-11-20	
8)	WAP to demonstrate abstract classes in C#	2080-11-20	
9)	WAP to demonstrate Sealed class in C#	2080-11-27	
10)	WAP to demonstrate partial class in C#	2080-12-4	
11)	WAP to demonstrate Delegate & event in C#	2080-12-13	
12)	WAP to demonstrate Following in C# i) collection ii) Generic iii) Lambda Expression iv) LINQ v) Try statement and Exception.	2080-12-23	

INTRODUCTION OF C#:

introduction:

C# is general purpose object oriented programming language developed by microsoft. C# is designed for CLI (common language infrastructure), which consist of the executable code and runtime environment. that allows use of various High Level Language on different computer platform and architecture.

C# a widely used professional language due to:

→ It is object oriented and structural language.
→ It is component oriented.
→ It is easy to learn and produces efficient programs.
→ A part of .NET FrameWork.
→ It can be compiled on a variety of computer platforms.

syntax:

```
using system;
using system.collection.Generic;
using system.Linq;
using system.Text;
using system.Threading.Tasks;

namespace <_program name_>
{

   internal class <_class name_>
   {
```

```
    // some methods:
}
static void main(string[] arg)
{
    // some class object and
       calling method.
}
}.
```

WAP TO DEMONSTRATE METHOD OVERRIDING IN C#:

Introduction:

In order to accomplish method overriding technique, we need to implement a method with a virtual keyword in a parent class and some method again re-implement with some name but different in implementation in derived class with a override keyword.

Source code:

```
using system;
namespace methodoverriding
{
    class Over
    {
        public virtual void Display()
        {
            console.Writeline("parent class");
        }
    }
    class Down : Over
    {
        public override void Display()
        {
            console.Writeline("child class");
        }

    }
    class progrom
    {
```

```
    static void main(string[] arg)
    {
        Down D = new Down()
        D.Display();
    }
}
}
```

program output:

child class.

WAP TO DEMONSTRATE METHOD HIDING IN C#

Introduction:

Method hiding can be achieved using 'new' keyword. When a method is been declared as with new keyword that it mean it will automatically hide the existing method of a parent class.

Source code:

```
using system;
namespace methodhiding
{
  class Parent
  {
    public void display()
    {
      console.WriteLine("Hi Daddy");
    }
}
class Child
{
    public new void display()
    {
      console.WriteLine("My son");
    }
}
class Program
{
```

```
    static void main(string[] org)
    {
        child c = new child();
        c.display();
    }
}
}
```

program output:

my son

WAP TO DEMONSTRATE CONSTRUCTOR IN C#:

Introduction:

A constructor is a special method of the class which gets automatically involved. Whenever any instance of the class is created. like method a constructor also contain the collection of instruction that are executed at the time of object creation. It is used to assign initial value to the data member of the same class.

source code:

```
using system;
namespace constructor
{
   class Cons
   {
      string name;
        int id;
   }
   Cons(string name, int id) // parameterized
                                constructor.
   {
      this. name = name;
      this. id = id;
   }
   class program
   {
```

```
    public static void main(string[] arg)
    {
        Cons c = new Cons("James", 7);
        Console.WriteLine("Name = " + c.name + "and id = "
                                    + c.id);
    }
}
}
```

program output:

Name = James and id = 7

WAP TO DEMONSTRATE INHERITANCE IN C#:

Introduction:

Inheritance is an important pillar of OOP (object oriented programming). It is the mechanism in C# by which one class is allowed to inherit the features (field and method) of another class. It is the capacity of a class to derive properties and characteristics from another class is called inheritance the new class derived class are created from existing class called base class.

source code:

```
using system;
namespace Inheritance
{
  class Shape
  {
    public void setWidth(int w)
    {
      width = w;
    }
    public void setHeight(int h)
    {
      height = h;
    }
    protected int width;
    protected int height;

}
```

```
class Rectangle : Shape
{
  public int getArea()
  {
    return (width * height);
  }
}
class Program
{
  static void main(string[] arg)
  {
    Rectangle R = new Rectangle();
    R.setWidth(5);
    R.setHeight(5);
    Console.WriteLine("Area is: {0}", R.getArea());
  }
}
}
```

program output:

Area is: 25

WAP TO DEMONSTRATE STRUCT IN C#:

Introduction:

C# structs also known as C# structure is a simple user-define type, a lightweight alternative to a class. similar to classes, structures have behaviour and attributes. C# structs supports access modifiers, constructors, indexers, methods, fields, nested types, operators and properties.

source code:

```
using system;
struct Books
{
   public string title;
   public string author;
   public string subject;
   public int book-id;
};
public class TestStructure
{
   public static void main(string[] org)
   {
      Books B;
      B.title = "C#";
      B.author = "KEC";
      B.subject = "programming";
      B.book-id = "365";
```

```
        Console.WriteLine("Title : {0}", B.Title);
        Console.WriteLine("Author : {0}", B.author);
        Console.WriteLine("subject : {0}", B.subject);
        Console.WriteLine("Book id : {0}", B.Book_id);

    }

}
```

program output:

```
Title : C#
Author : KEC
subject : programming
Book id : 365
```

WAP TO DEMONSTRATE ENUM IN C#:

Introduction:

An Enumeration is a set of named integer constant. An enumerated type is declared using the enum keyword. C# enumerations are valued data types. In other words, enumeration contain its own value and cannot inherit or cannot pass inheritance.

source code:

```
using system;
namespace Enumprogram
{ enum Days { sun, mon, tue, Wed, Thu, Fri, sat };
  static void Main(string[] args)
  { int Weekdaystart = (int) Days.mon;
    int WeekdayEnd = (int) Days.Fri;

    console.WriteLine("Monday: {0}", Weekdaystart);
    console.WriteLine("Friday: {0}", WeekdayEnd);
  }
}
```

program output:

Monday : 1

Friday : 5

WAP TO DEMONSTRATE ABSTRACT CLASS IN C#:

Introduction:

classes can be declared as abstract by using keyword abstract. If we like to make class that only represent base class and don't want any one to create object of these class type, then we implement this functionality. The derived class should implement the abstract class member.

source code:

```
using system;
abstract class AbsClass
{
  public abstract int Area();

}
class Square: AbsClass
{
    int side =0;
    public Square(int n)
    {
        side=n;
    }
    public override int Area()
    {
        return side*side;

    }

}
```

```
class program
{
    public static void Main(string[] args)
    {
        Square s = new Square(6);
        Console.WriteLine("Area = " + s.Area());
    }
}
```

program output:

Area : 36

WAP TO DEMONSTRATE SEALED CLASSES IN C#:

Introduction:

Sealed classes are used to restrict the inheritance feature of object oriented programming. once a class is defined as a sealed class, the class cannot be inherited. in c#, the sealed modifier is to define a class as sealed.

source code:

```
using system;
sealed class Seals
{  public void display()
   {  console.kliteline("this is sealed class");
   }
}
class program
{
   static void main(string[] args)
   {
      Seals s = new Seals();
      s.display();
   }
}
```

program output:

compilation error

WAP TO DEMONSTRATE PARTIAL CLASS IN C#:

introduction:

In c#, a class definition can be divided over multiple files. If class definition is divided into multiple files, each part is declared as a partial class. All part of partial class must be use the 'partial' keywords.

source code:

```
using system;
namespace partialclasses
{ public partial class cords
  {
     private int x;
     private int y;
     public cords(int x, int y)
     {
        this.x = x;
        this.y = y;
     }
  }
  public partial class cords
  {
     public void print(cords c)
     {
       console.writeline("cords:{0}{1}:", x, y);
     }
  }
```

```
class TestCords
{
    static void Main(string[] args)
    {
        Cords c = new Cords(10,15);
        c.printCords();
    }
}
}
```

program output:

```
Cords : 10,15
```

WAP TO DEMONSTRATE DELEGATE & EVENTS IN C#:

Introduction:

Delegate in c# are similar to the function pointers in c++/c. It provides a way which tells which method is to be called when an event is triggered. A delegate is a reference type of variable that hold the reference to a method. The reference can be changed at run time.

source code:

```
using system;
namespace delegate
{
  public delegate void SampleDel();
  class [illegible]
  {
    public static void MyFun()
    {
      console.WriteLine("call by delegate");
    }
    class DelTest
    {
      static void Main(string[] args)
      {
        sampleDel D = new Sample Del(MyFun());
        D();
      }
    }
}
```

program output:

call by delegate.

WAP TO DEMONSTRATE COLLECTION IN C#:

Introduction:

collection is a more flexible way to work group of object. unlike arrays, the group of objects can grow and shrink dynamically as the need of the application change. For example some collections we can assign a key to any object that we put into the collection so that we can quickly retrive the object by using the key. A collection is a class so we must declare an instance of the class before we can add elements to that collections.

source code:

```
using system;
namespace collection
{ internal class program
  { static void Main(string[], args)
    { ArrayList A = new ArrayList();
      console.WriteLine("defining some number");
      A.add(45);
      A.add(78);
      A.add(33);
      A.add(56);
      console.WriteLine("capacity:{0}", A.capacity);
      A.add(12);
      A.add(23);
      A.add(9);
      console.WriteLine("capacity:{0}", A.capacity);
      console.WriteLine("count:{0}", A.count);
      console.writeline("content");
```

```
foreach (int i in A)
{ console.WriteLine(i+" ");
}console.Writeline("sorted content");
A.short();
foreach (int i in A)
{ console.Writeline(i+" ");
}console.Writeline();
  }
 }
}
```

program output:

defining some number
capacity : 4
capacity : 8
count : 7
content
45
78
33
56
12
23
9
sorted content
9
12
23
33
45
56
78

WAP TO DEMONSTRATE GENERIC IN C#:

Introduction:

A generic collection enforces type safety so that another data type can be added to it. When we retrive an element from a generic collection. We do not have to determine it's data type or convert it. The generic collection classes are implemented which are present in this namespace are as follow:

stack<T>
queue<T>
Linkedlist<T>
storedlist<T>
list<T>
Dictionary<Tkey;Tvalue>

Here<T> refers to the type of value as we want to store under them.

source code:

```
using system;
namespace Generic
{
  public class Gen<T>
  {
    private T data;
    public T value
    {
      get
      {
        return this.data;
      }set{
        return this.data = value;
      }
    }
  }
}
```

```
public class Program
{
    static void Main(string[] args)
    {
        Gen<string> name = new Gen<string>();
        name.value = "My name is James";
        Gen<float> version = new Gen<float>();
        version.value = 5.6f;

        Console.WriteLine(name.value);
        Console.WriteLine(version.value);
    }
}
}
```

program output:

My name is James
5.6