Sub:	Programming Using C#							
Date:	24/06/23		Duration:	90 min's	Max Marks:	50	Sem:	IV

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

PART I

1 **Make a short note on Inheritance giving example.**

Ans :

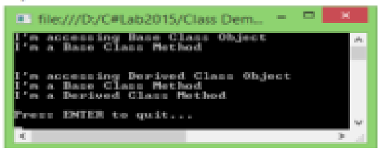
Inheritance provides you to reuse existing code and fast implementation time. The relationship between two or more classes is termed as Inheritance. In essence, inheritance allows to extend the behavior of a base (or parent/super) class by enabling a subclass to inherit core functionality (also called a derived class/child class). All public or protected variables andmethods in the base class can be called in the derived classes.
Inheritance and Constructors: As you know, a constructor is a special method of a class, which is used to initialize the members of the same class. A constructor is called by default whenever an object of a class is created. It is important to note that a Base class constructors are not inherited by derived classes. Thus cannot instantiate a base constructor using derived class object, so we need the “base” keyword to access constructor of the base class from within a derived class.

Inheritance is of four types, which are as follows: i. Single Inheritance: Refers to inheritance in which there is only one base class and one derived class. This means that a derived class inherits properties from single base class. ii. Hierarchical Inheritance: Refers to inheritance in which multiple derived classes are inherited from the same base class. iii. Multilevel Inheritance: Refers to inheritance in which a child class is derived from a class, which in turn is derived from another class. iv. Multiple Inheritance: Refers to inheritance in which a child class is derived from multiple baseclass.C# supports single, hierarchical, and multilevel inheritance because there is only a single base class. It does not support multiple inheritance directly.

InherDemo.cs

```
using System;
namespace Class_Demos{
  class BaseClass{
    public int dm;
    public void BCMethod(){
      Console.WriteLine("I'm a Base Class Method");
    }
  }
  class DerivedClass:BaseClass{
    public void DCMethod(){
      Console.WriteLine("I'm a Derived Class Method");
    }
  }
```

```
class InherDemo{
  static void Main(){
    //Create a Base Class Object
    Console.WriteLine("I'm accessing Base Class Object");
    BaseClass bc = new BaseClass();
    bc.dm = 10;
    bc.BCMethod();
    //Create a Derived Class Object
    Console.WriteLine("I'm accessing Derived Class Object");
    DerivedClass dc=new DerivedClass();
    dc.dm = 20;
    dc.BCMethod();
    dc.DCMethod();
    Console.WriteLine("\nPress ENTER to quit...");
    Console.Read();
  }
}
}
```



```
I'm accessing Base Class Object
I'm a Base Class Method

I'm accessing Derived Class Object
I'm a Base Class Method
I'm a Derived Class Method

Press ENTER to quit...
```

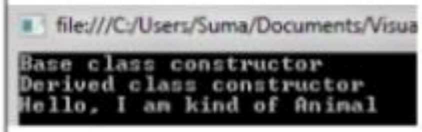
2 **Explain :**
Classical Inheritance   b) Containment / Delegation model

Classical inheritance ("is-a" relationship): When "is-a" relationship have established between classes, we are building a dependency between types. The basic idea behind classical inheritance is that new classes may extend the functionality of other classes.

Assume that we wish to define two additional classes to model Animal and Dog. The hierarchy looks like as shown below and we notice that Animal "is-a" Mammal, Dog IS-A Animal; Hence dog IS-A mammal as well. In "is-a" model, base classes are used to define general characteristics that are common to all subclasses and classes are extended by using ":" operator. The derived classes inherit the base class's properties and methods and clients of the derived class have no knowledge of the base class.

```
using System;
namespace Chapter4_Examples{
  class Animal {
   public Animal(){
     Console.WriteLine("Base class constructor");
   }
   public void Greet(){
     Console.WriteLine("Hello,I am kind of Animal");
   }
  }
  class Dog : Animal{
   public Dog(){
     Console.WriteLine("Derived class constructor");
   }
  }
```

```
class isademo{
 static void Main(){
    Dog d = new Dog();
    d.Greet();
    Console.ReadKey();
  }
}
}
```



```
Base class constructor
Derived class constructor
Hello, I am kind of Animal
```

Containment / Delegation model ("Has-A"): The "HAS-A" relationship specifies how one class is made up of other classes

Consider we have two different classes Engine and a Car when both of these entities share each other's object for some work and at the same time they can exists without each other's dependency(havingtheir own life time) and there should be no single owner both have to be an independent from each other than type of relationship is known as "has-a" relationship i.e. Association.

```
using System;
namespace Chapter4_Examples{
  class Engine{
    public int horsepower;
    public void start(){
     Console.WriteLine("Engine Started!");
  } }
  class Car{
    public string make;
    public Engine eng;   //Car has an Engine
    public void start(){
      eng.start();
  } }
```

```
class hasademo{
 static void Main(){
    Console.WriteLine("Manufacturing a Car");
    Car mycar = new Car();
    mycar.make = "Toyoto";
    Console.WriteLine("Manufacturing a Engine to start car");
    mycar.eng = new Engine();
    mycar.eng.horsepower =220;
    Console.WriteLine("\n***Car Details***");
    Console.WriteLine("Brand:"+ mycar.make);
    Console.WriteLine("Power: "+ mycar.eng.horsepower);
    mycar.start();
    Console.Read();
 }  }  }
```

PART II

3 **What are the two ways of enforcing encapsulation? Give examples for both the methods**

Encapsulation using accessors and mutators: Rather than defining the data in the form of public, we can

declare those fields as private so that we achieved encapsulation. The Private data are manipulated using accessor (get: store the data to private members) and mutator (set: to interact with the variable) methods. Syntax:

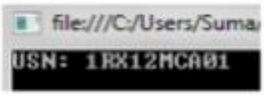
```
set { }
 get { }
```

. A property defined with both a getter and a setter is called a read-write property. A property defined with only a getter is called a read-only property. A property defined with only a setter is called a write-only property.

Encapsulation using Properties: i. Write-Only Property: Properties can be made write-only. This is

```
using System;
namespace Class_Demos{
 class Student{
   string studusn;
   public Student(){
     studusn="1RX12MCA01";
   }
   public string Studusn{
     get{
        return studusn;
     }
   }
 }
```

```
class ReadOnly{
  static void Main(){
     Student st1 = new Student();
     Console.WriteLine("USN: "
                          + st1.Studusn);
     Console.ReadKey();
  }
}
}
```



```
USN: 1RX12MCA01
```

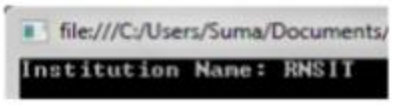
"static" property: C# also supports "*static*" properties. The ***static members*** are accessed at the class level, not from an instance (object) of that class. Static properties are manipulated in the same manner as static methods, as seen here:

Example 2.5: Assume that the Student type defines a point of static data to represent the name of the institution studying these students. You may define a static property as follows:

```
using System; namespace
Class Demos{ class
Student
  string name, branch, usn;
  static string instName
  public static string Institution

       set    instName     value   }
       get    return instName;
```

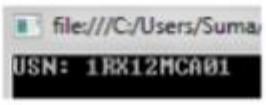
```
class StaticProperty{
  static void Main(){
     Student.Institution = "RNSIT";
     Console.WriteLine("InstitutionName:"
                      +Student.Institution);
     Console.ReadKey();
      }
  }
}
```



```
Institution Name: RNSIT
```

```
using System;
namespace Class_Demos{
 class Student{
   string studusn;
   public Student(){
     studusn="1RX12MCA01";
   }
   public string Studusn{
     get{
        return studusn;
     }
   }
 }
```

```
class ReadOnly{
  static void Main(){
     Student st1 = new Student();
     Console.WriteLine("USN: "
                    + st1.Studusn);
     Console.ReadKey();
  }
}
}
```



```
USN: 1RX12MCA01
```

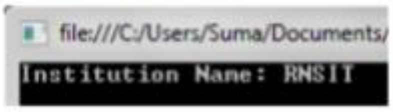
"static" property: C# also supports "*static*" properties. The ***static members*** are accessed at the class level, not from an instance (object) of that class. Static properties are manipulated in the same manner as static methods, as seen here:

Example 2.5: Assume that the Student type defines a point of static data to represent the name of the institution studying these students. You may define a static property as follows:

```
using System; namespace
Class Demos{ class
Student
  string name, branch, usn;
  static string instName
  public static string Institution

        set     instName     value    }
        get     return  instName;
```

```
class StaticProperty{
  static void Main(){
    Student.Institution = "RNSIT";
    Console.WriteLine("InstitutionName:"
                    +Student.Institution);
    Console.ReadKey();
       }
    }
}
```

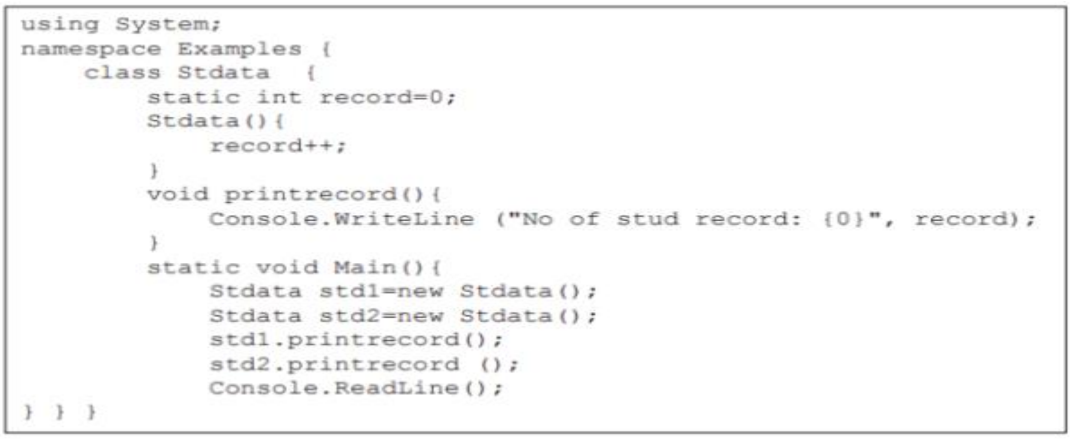


```
Institution Name: RNSIT
```

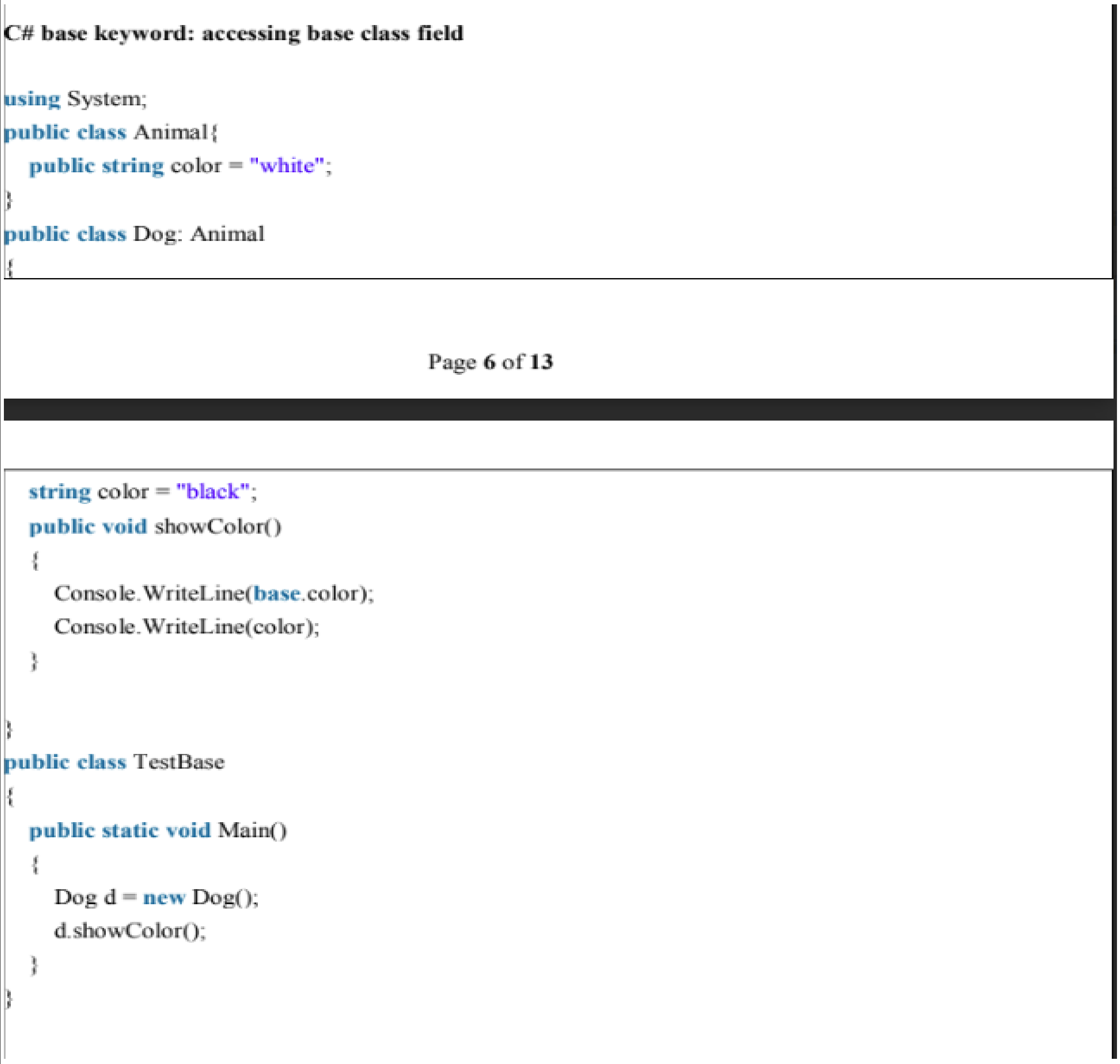
4 **Write a short note on:**
i. Static data ii. Virtual method iii. base keyword

Static data members:

When we declare a static field inside a class, it can be initialized with a value or all un-initialized static fields automatically get initialized to their default values when the class is loaded at first time. Characteristics: It is visible only within the class, but its lifetime is the entire program. Only one copy of static data member will exists for the entire class and is shared by all the objects of that class. No matter how many objects of a class are created. All static variables are initialized to zero when the first object of its class is created.



```
using System;
namespace Examples {
    class Stdata  {
        static int record=0;
        Stdata(){
            record++;
        }
        void printrecord(){
            Console.WriteLine ("No of stud record: {0}", record);
        }
        static void Main(){
            Stdata std1=new Stdata();
            Stdata std2=new Stdata();
            std1.printrecord();
            std2.printrecord ();
            Console.ReadLine();
} } }
```



C# base keyword: accessing base class field

```
using System;
public class Animal{
   public string color = "white";
}
public class Dog: Animal
{
```

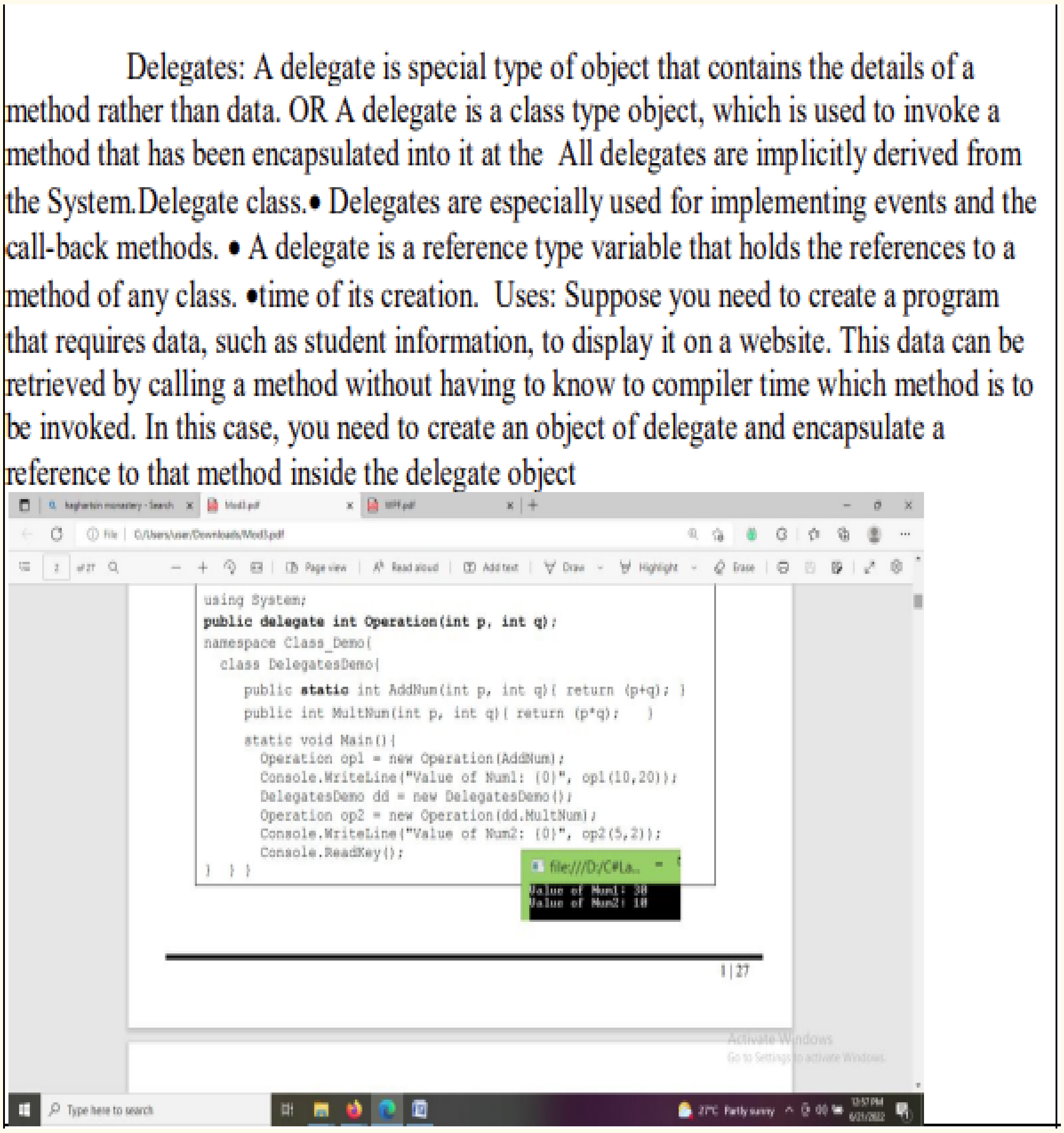
Page 6 of 13

```
   string color = "black";
   public void showColor()
   {
      Console.WriteLine(base.color);
      Console.WriteLine(color);
   }


}
public class TestBase
{
   public static void Main()
   {
      Dog d = new Dog();
      d.showColor();
   }
}
```

PART III

5 **What is meant by the .Net delegate type? Explain the concept with example code.**



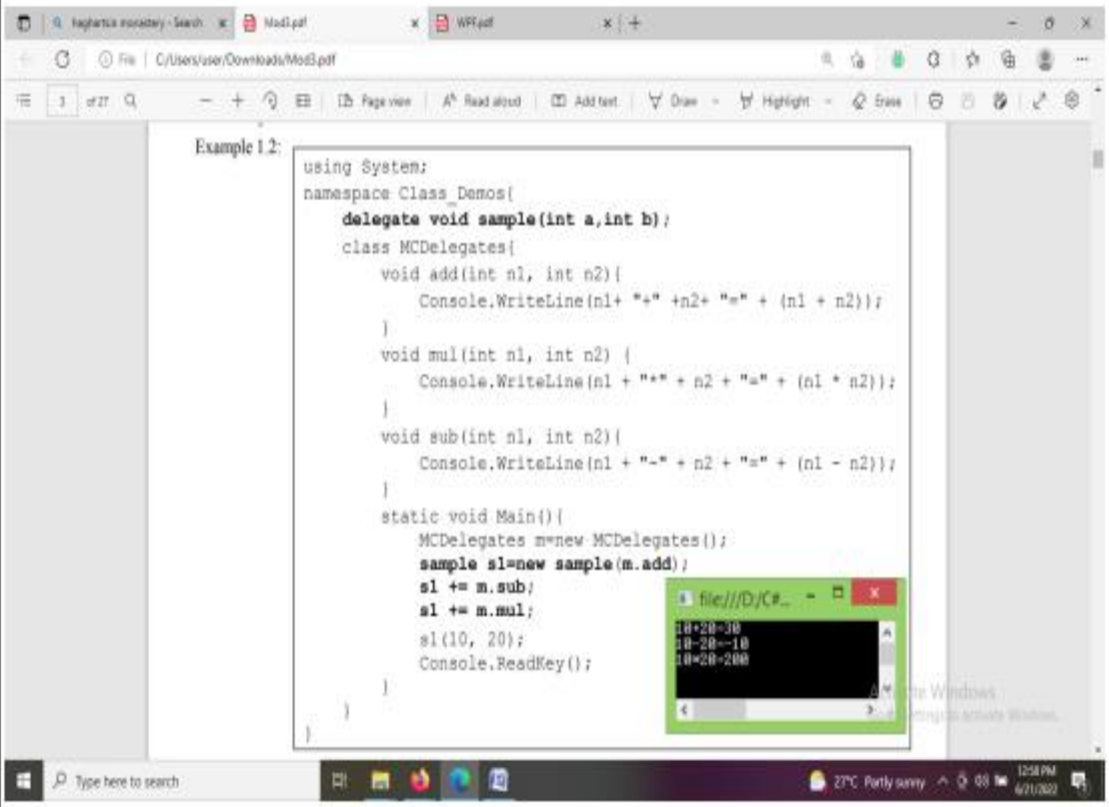
Delegates: A delegate is special type of object that contains the details of a method rather than data. OR A delegate is a class type object, which is used to invoke a method that has been encapsulated into it at the All delegates are implicitly derived from the System.Delegate class.• Delegates are especially used for implementing events and the call-back methods. • A delegate is a reference type variable that holds the references to a method of any class. •time of its creation. Uses: Suppose you need to create a program that requires data, such as student information, to display it on a website. This data can be retrieved by calling a method without having to know to compiler time which method is to be invoked. In this case, you need to create an object of delegate and encapsulate a reference to that method inside the delegate object

```
using System;
public delegate int Operation(int p, int q);
namespace Class_Demo{
  class DelegatesDemo{
     public static int AddNum(int p, int q){ return (p+q); }
     public int MultNum(int p, int q){ return (p*q);    }
     static void Main(){
       Operation op1 = new Operation(AddNum);
       Console.WriteLine("Value of Num1: {0}", op1(10,20));
       DelegatesDemo dd = new DelegatesDemo();
       Operation op2 = new Operation(dd.MultNum);
       Console.WriteLine("Value of Num2: {0}", op2(5,2));
       Console.ReadKey();
}  } }
```

```
Value of Num1: 30
Value of Num2: 10
```

6 **What are delegates? Explain with code example, the concept of multicasting with delegates.**

Delegates: A delegate is special type of object that contains the details of a method rather than data. OR A delegate is a class type object, which is used to invoke a method that has been encapsulated into it at the All delegates are implicitly derived from Delegates are especially used for implementing events and the•the System.Delegate class. A delegate is a reference type variable that holds the references to a•call-back methods. time of its creation. Uses: Suppose you need to create a program•method of any class. that requires data, such as student information, to display it on a website. This data can be retrieved by calling a method without having to know to compiler time which method is to be invoked. In this case, you need to create an object of delegate and encapsulate a reference to that method inside the delegate object



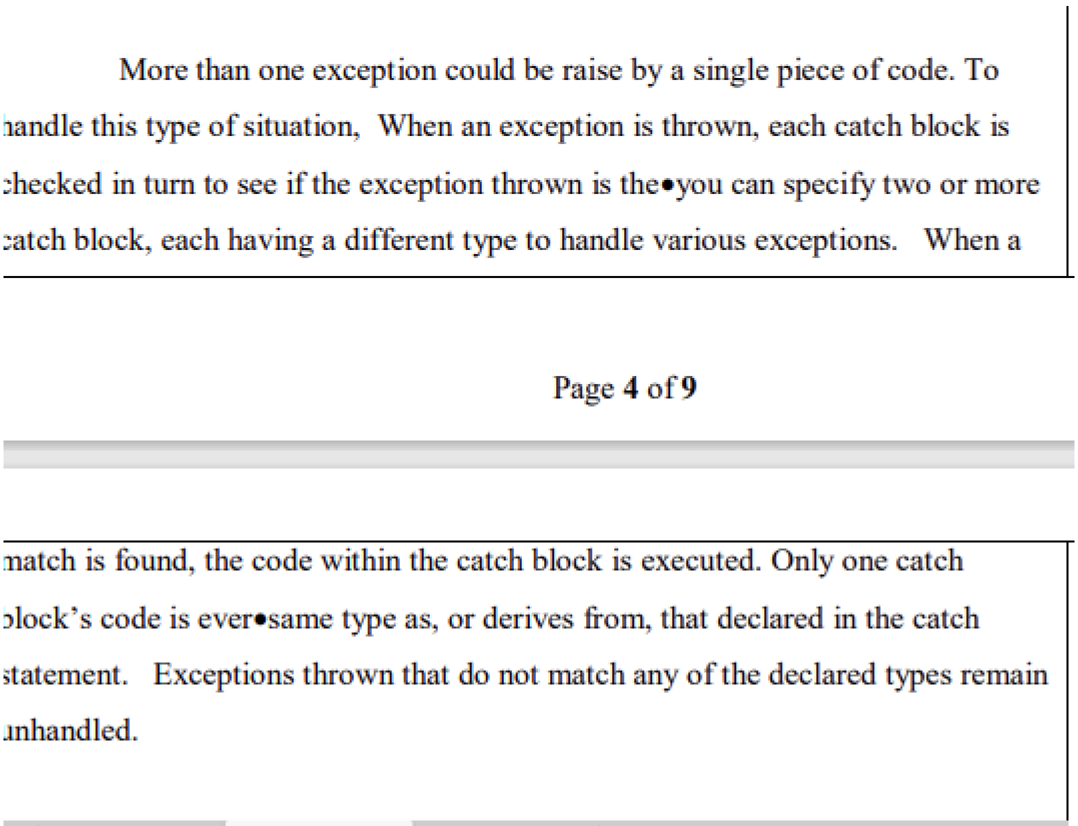
Example 1.2:

```
using System;
namespace Class_Demos{
    delegate void sample(int a,int b);
    class MCDelegates{
        void add(int n1, int n2){
            Console.WriteLine(n1+ "+" +n2+ "=" + (n1 + n2));
        }
        void mul(int n1, int n2) {
            Console.WriteLine(n1 + "*" + n2 + "=" + (n1 * n2));
        }
        void sub(int n1, int n2){
            Console.WriteLine(n1 + "-" + n2 + "=" + (n1 - n2));
        }
        static void Main(){
            MCDelegates m=new MCDelegates();
            sample s1=new sample(m.add);
            s1 += m.sub;
            s1 += m.mul;
            s1(10, 20);
            Console.ReadKey();
        }
    }
}
```

```
10+20=30
10-20=-10
10*20=200
```

PART IV

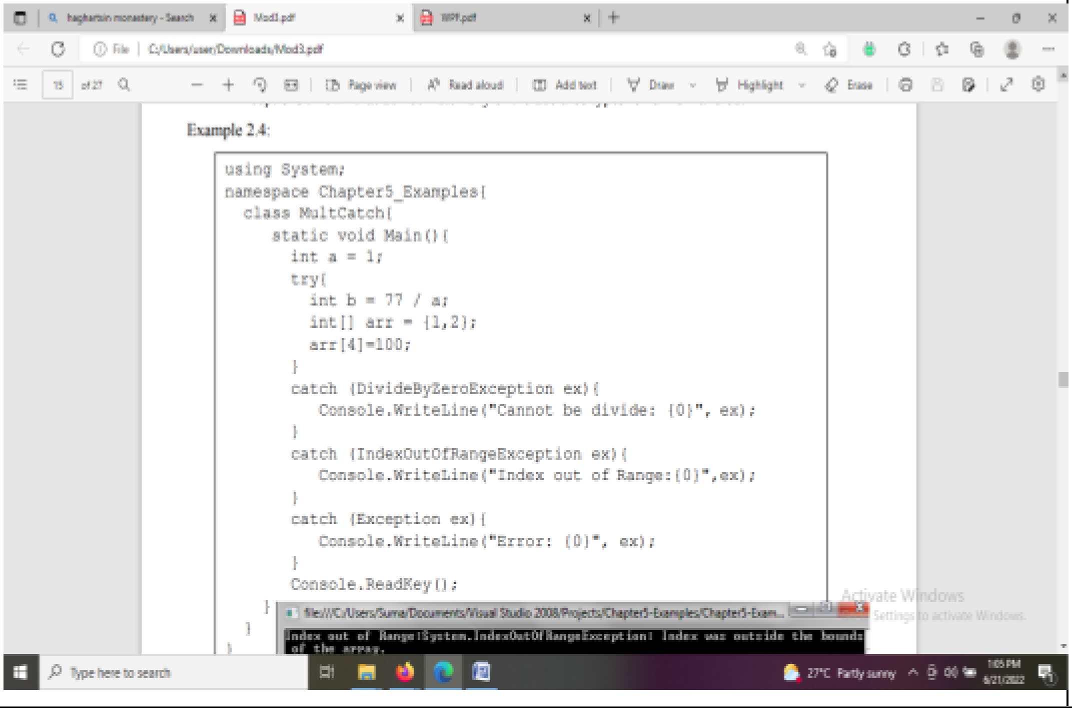
7 **Explain exception handling with a sample program to handle multiple exceptions.**



More than one exception could be raise by a single piece of code. To handle this type of situation, When an exception is thrown, each catch block is checked in turn to see if the exception thrown is the•you can specify two or more catch block, each having a different type to handle various exceptions. When a

Page **4** of **9**

match is found, the code within the catch block is executed. Only one catch block's code is ever•same type as, or derives from, that declared in the catch statement. Exceptions thrown that do not match any of the declared types remain unhandled.



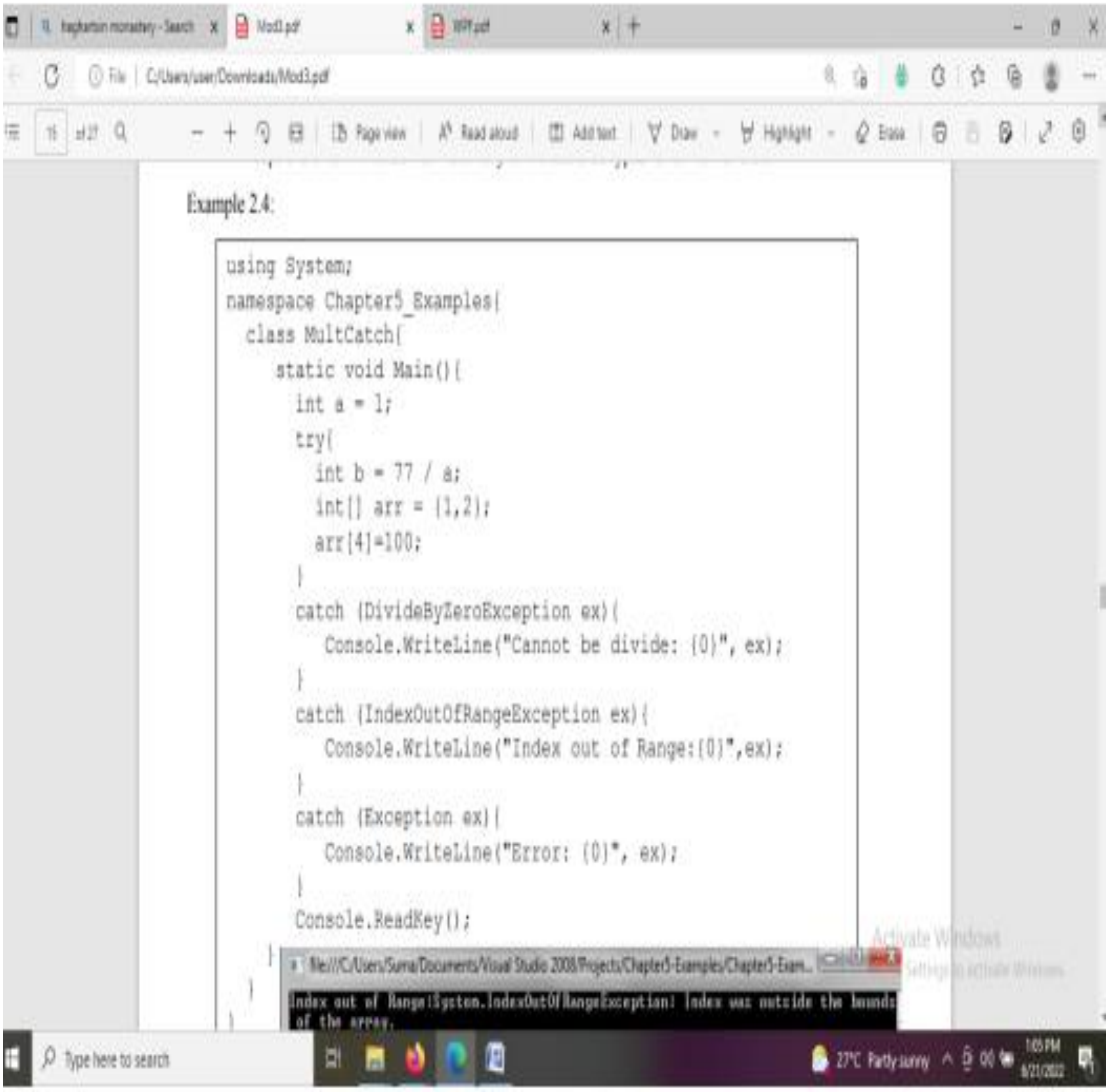
Example 2.4:

```
using System;
namespace Chapter5_Examples{
  class MultCatch{
     static void Main(){
       int a = 1;
       try{
         int b = 77 / a;
         int[] arr = {1,2};
         arr[4]=100;
       }
       catch (DivideByZeroException ex){
          Console.WriteLine("Cannot be divide: {0}", ex);
       }
       catch (IndexOutOfRangeException ex){
          Console.WriteLine("Index out of Range:{0}",ex);
       }
       catch (Exception ex){
          Console.WriteLine("Error: {0}", ex);
       }
       Console.ReadKey();
     }
  }
}
```

```
Index out of Range:System.IndexOutOfRangeException: Index was outside the bounds
 of the array.
```

Mod1.pdf
WPf.pdf
C:/Users/user/Downloads/Mod3.pdf
Page view
Read aloud
Add text
Draw
Highlight
Erase
file:///C:/Users/Suma/Documents/Visual Studio 2008/Projects/Chapter5-Examples/Chapter5-Exam...
Activate Windows
Type here to search
27°C Partly sunny

8 **Explain .NET Exception handling with valid example code**



Example 2.4:

```
using System;
namespace Chapter5_Examples{
  class MultCatch{
     static void Main(){
       int a = 1;
       try{
         int b = 77 / a;
         int[] arr = {1,2};
         arr[4]=100;
       }
       catch (DivideByZeroException ex){
          Console.WriteLine("Cannot be divide: {0}", ex);
       }
       catch (IndexOutOfRangeException ex){
          Console.WriteLine("Index out of Range:{0}",ex);
       }
       catch (Exception ex){
          Console.WriteLine("Error: {0}", ex);
       }
       Console.ReadKey();
     }
   }
}
```

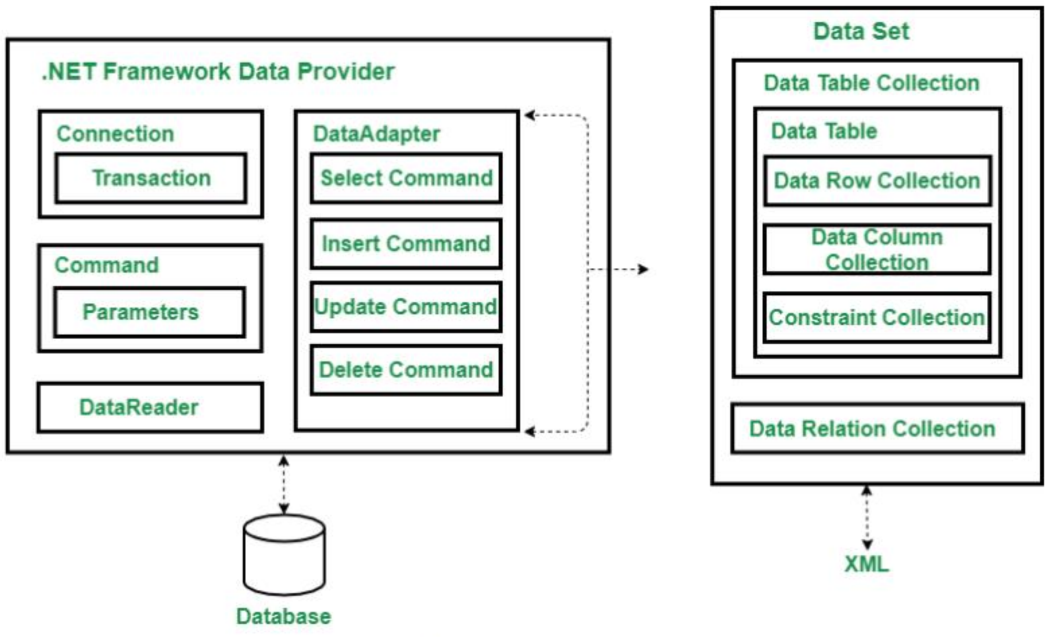
```
Index out of Range:System.IndexOutOfRangeException: Index was outside the bounds
 of the array.
```

PARTV

9 **Explain the architecture of ADO.NET with a neat diagram**

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

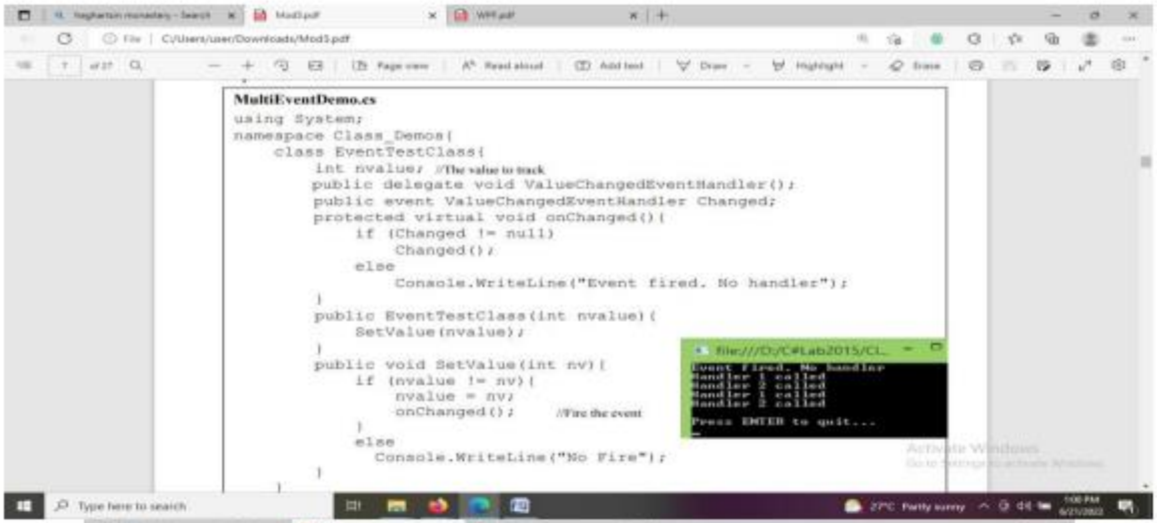
Refer figure which shows the ADO.NET objects at



Architecture of ADO.NET

10 **Describe how to implement the multiple events. Give example**

Like delegates, events can be multicast. This enables multiple objects to respond to an event notification, you used the += operator to add the event handler

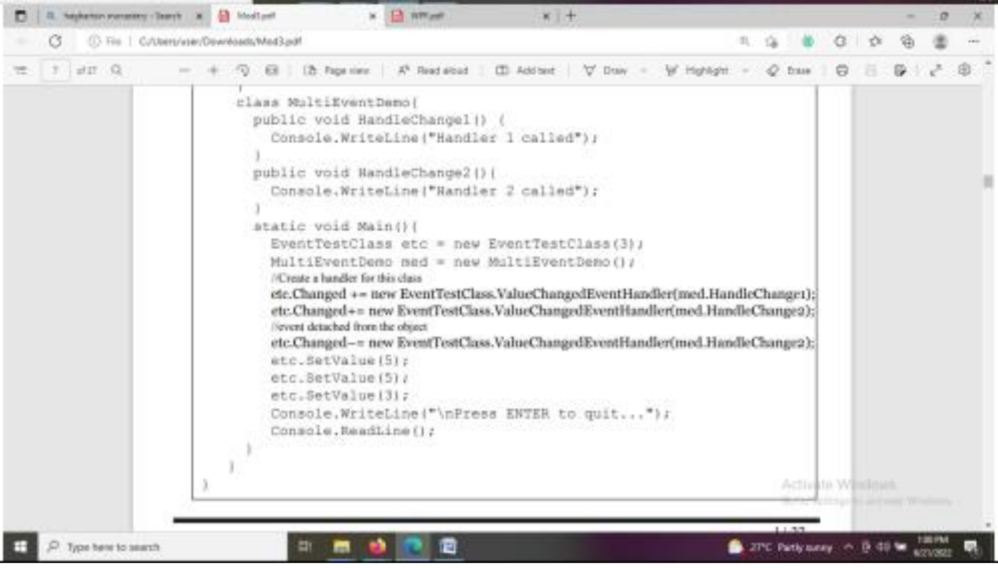


MultiEventDemo.cs

```
using System;
namespace Class_Demos{
    class EventTestClass{
        int nvalue; //The value to track
        public delegate void ValueChangedEventHandler();
        public event ValueChangedEventHandler Changed;
        protected virtual void onChanged(){
            if (Changed != null)
                Changed();
            else
                Console.WriteLine("Event fired. No handler");
        }
        public EventTestClass(int nvalue){
            SetValue(nvalue);
        }
        public void SetValue(int nv){
            if (nvalue != nv){
                nvalue = nv;
                onChanged();        //Fire the event
            }
            else
              Console.WriteLine("No Fire");
        }
    }
```

```
Event Fired. No handler
Handler 1 called
Handler 2 called
Handler 1 called
Handler 2 called

Press ENTER to quit...
```



```
    class MultiEventDemo{
      public void HandleChange1() {
        Console.WriteLine("Handler 1 called");
      }
      public void HandleChange2(){
        Console.WriteLine("Handler 2 called");
      }
      static void Main(){
        EventTestClass etc = new EventTestClass(3);
        MultiEventDemo med = new MultiEventDemo();
        //Create a handler for this class
        etc.Changed += new EventTestClass.ValueChangedEventHandler(med.HandleChange1);
        etc.Changed+= new EventTestClass.ValueChangedEventHandler(med.HandleChange2);
        //event detached from the object
        etc.Changed-= new EventTestClass.ValueChangedEventHandler(med.HandleChange2);
        etc.SetValue(5);
        etc.SetValue(5);
        etc.SetValue(3);
        Console.WriteLine("\nPress ENTER to quit...");
        Console.ReadLine();
      }
    }
}
```