Hands on LAB document for Problem Solving (C#)

Contents

C# Basics 3

Lab 1: Display Welcome message on console with single code statement. 3

Lab 2: Display Welcome message in multiple lines on console with single code statement. 3

Lab3: Reading inputs from the console. 4

Working with Methods and its Parameters 6

Lab1: Methods with Different types of Parameters 6

Working with Arrays 8

Lab1: Working with a Single Dimensional Array 8

Lab2: Working with Multi-Dimensional Array 8

Working with Strings 10

Lab1: Convert a string to lowercase/ uppercase 10

Lab2: Split the input string on spaces 10

Lab3: Using StringBuilder for concatenation of strings 11

Working with Structures 13

Lab1: Declaring a structure with public members and accessing them in other classes using a structure variable 13

Working with Classes and Objects 14

Lab1: Declaring and Instantiating a class 14

Lab2: Declaring a class with Private fields, Creating methods to assign and retrieve values to/from private fields and Instantiating a class 14

Lab3: Declaring a class with Private Fields, Creating method with parameters to assign dynamic values, Return value from a method to display values of private fields and Instantiating a class ... 16

Lab4: Declaring a class with Private Fields, Creating Properties to assign / retrieve values into/from private fields and Instantiating a class ... 17

Lab5: Declaring a class with Private Fields, Creating Constructors (Default and Parameterized) to assign values into private fields, a method to return values from private fields and Instantiating a class ... 20

Inheritance and Polymorphism ... **22**

Lab1: Program that demonstrates Simple Inheritance with default and parameterised constructor ... 22

Lab2: Program that demonstrates Static Polymorphism using Method Overloading ... 24

Lab3: Program that demonstrates between Static Polymorphism using New Keyword and Dynamic Polymorphism using Virtual Keyword ... 25

Lab4: Program that demonstrates abstract class ... 28

Lab5: Program that demonstrates Interfaces ... 29

Lab6: Program that demonstrates Multiple Inheritance with Interfaces ... 30

Collections ... **32**

Lab1: Program that demonstrates ArrayList concepts**.** ... 32

Lab2: Program that demonstrates Hashtable concepts**.** ... 34

Generic Collections ... **36**

Lab1: Program that demonstrates List concepts. ... 36

Lab2: Program that demonstrates Dictionary concepts**.** ... 38

Scenario based Problem Statement ... **41**

C# Basics

Lab 1: Display Welcome message on console with single code statement.

Steps: Create a new console application name "ConsoleAppDemos". In the Program.cs file, update the Main method with the following code.

```
using System;

class Program
    {
        // Main method begins execution of C# application
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to C# Programming!");
            Console.ReadKey();
        } //end of Main
    } //end of class Program
```

Note: Press F5 to execute the program

Output:

```
Welcome to C# Programming
```

Lab 2: Display Welcome message in multiple lines on console with single code statement.

Steps: Add a new Item type "Class" to the project "ConsoleAppDemos", and name it as "Welcome.cs". In the Welcome.cs file, add Main method with the following code.

```
using System;

public class Welcome
    {
        // Main method begins execution of C# application
        public static void Main(string[] args)
        {
            Console.WriteLine("Welcome \nto \nC# \nProgramming!");
            Console.ReadLine();
        } // end Main
    } // end class Program
```

<u>Note:</u> Press F5 to execute the program.

Error Message will be displayed as two programs are defined with the entry point i.e. Main(). Now set the start-up program as Welcome.CS.

Steps to set the Start-up Program: Open Solution Explorer -> Select Project under Solution -> Double Click the properties icon under Project -> Click the Application Tab -> set the Start-up object as "Welcome.cs" -> Press F5 to execute.

Output:

```
Welcome

to

C#

Programming
```

Lab3: Reading inputs from the console.

Program to accept two numbers from input device and display the sum on to the console.

Steps : : Add a new Item type "Class" to the project "ConsoleAppDemos", and name it as "Addition.CS". In the Addition.cs file, add Main method with the following code.

```
using System;

class Addition
    {
        public static void Main(string[] args)
        {
            int number1, number2, sum;
            //Prompt User to input a number
            Console.Write("Enter first integer: ");
            // read first number from user
            number1 = Convert.ToInt32(Console.ReadLine());
            //Prompt User to input a number
            Console.Write("Enter second integer: ");
            // read second number from user
            number2 = Convert.ToInt32(Console.ReadLine());
            // add numbers
            sum = number1 + number2;
            // display sum
            Console.WriteLine("Sum is {0}", sum);
            Console.ReadLine();
        } // end Main
    } // end class Addition
```

Note: Set the Start-up program as **Addition.CS** and **Press F5 to execute.**

Output:

```
Enter first integer: 15

Enter second integer: 10

Sum is 25
```

Working with Methods and its Parameters

Lab1: Methods with Different types of Parameters

```
using System;

namespace ConsoleAppDemos
{
    class MethodsWithParameterTypes_Demo
    {
        public int DisplayResult_WithValueTypeParams(int a, int b)
        {
            a = 100;
            b = 200;
            return a + b;
        }

        public int DisplayResult_WithRefTypeParams(ref int a, ref 
int b)
        {
            a = 100;
            b = 200;
            return a + b;
        }

        public void DisplayResult_WithOutTypeParams(ref int a, ref 
int b,out int Sum,out int Diff)
        {
            a = 100;
            b = 200;
            Sum = a + b;
            Diff = b - a;
        }

        static void Main()
        {
            int x=50, y=50;
            MethodsWithParameterTypes_Demo obj = new 
MethodsWithParameterTypes_Demo();
            Console.WriteLine("Before calling method, Values of x 
and y are:{0},{1}\n\n", x, y);
            int result=obj.DisplayResult_WithValueTypeParams(x, y);
            Console.WriteLine("After calling method with Value Type
Parameters, Values of x and y are:{0},{1} \nand the Result is:
{2}\n\n", x, y,result);
            result = obj.DisplayResult_WithRefTypeParams(ref x, ref
y);
            Console.WriteLine("After calling method with Refernce
Type Parameters, Values of x and y are:{0},{1} \nand the Result is:
{2}\n\n", x, y, result);

            int total = 0, diff = 0;
            obj.DisplayResult_WithOutTypeParams(ref x, ref y,out
total,out diff);
            Console.WriteLine("After calling method with Out Type
Parameters, Values of x & y are:{0},{1} \nand the sum is: {2},
\ndiff is: {3}", x, y, total, diff);
            Console.ReadKey();

        } // end Main
    } // end class MethodsWithParameterTypes_Demo
}// end namespace
```

Output:

```
Before calling methods, values of X and Y are : 50, 50

After calling method with Value Type Parameters, values of X and Y are: 50, 50

And the Result is: 300

After calling method with Reference Type Parameters, values of X and Y are:
100, 200

And the Result is: 300


After calling method with Out Type Parameters, values of X and Y are: 100, 200

And the sum is: 300

Diff is: 100
```

Working with Arrays

Lab1: Working with a Single Dimensional Array

```
using System;

namespace ConsoleAppDemos
{
    class SimpleArrayDemo
    {
        static void Main()
        {
            //Initializing an array at the time of declaration
            int[] arr = new int[] { 10, 20, 30, 40, 50, 60 };
            //Displaying the values of an array
            //array.Length property is used to know the size of an
array
            Console.WriteLine("Displaying the values of an array:");
            for (int i = 0; i < arr.Length; i++)
                Console.Write(arr[i] + ", ");
            Console.ReadKey();
        }
    }
}
```

Output:

```
Displaying the values of an array :

10, 20, 30, 40, 50
```

Lab2: Working with Multi-Dimensional Array

```
using System;

namespace ConsoleAppDemos
{
    {
```

```
    class MultiDimensionalArrayDemo
    {
        static void Main()
        {
            string[,] str = new string[2, 3] { { "C", "C++", "Java"
}, { "ABC", "XYZ", "OPR" } };
            Console.WriteLine("Displaying the values of an array in
rows and columns");
            for (int rowIndex = 0; rowIndex < 2; rowIndex++)
            {
                for (int colIndex = 0; colIndex < 3; colIndex++)
                {
                    Console.Write(str[rowIndex, colIndex] + "\t");
                }
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

Output:

```
Displaying the values of an array in rows and columns:

C       C++     Java

ABC     XYZ     OPR
```

Working with Strings

Lab1: Convert a string to lowercase/ uppercase

```
using System;

class Program
    {
        static void Main()
        {
            string value = "Station";
            // Convert to uppercase.
            value = value.ToUpper();
            Console.WriteLine(value);
            // Convert to lowercase.
            value = value.ToLower();
            Console.WriteLine(value);

        } //end of Main
    } //end class Program
```

Output:

```
STATION

station
```

Lab2: Split the input string on spaces

```
using System;

class Program
    {
        static void Main()
        {
            string s = "The cat sat on the mat";
            // Split string on spaces.
            string[] words = s.Split(' ');// This will separate all
the words.
```

```
            foreach (string word in words)
            {
                Console.WriteLine(word);
            }
            Console.ReadLine();
        } // end Main
    } // end class Program
```

Output:

```
The

cat

sat

on

the

mat
```

Lab3: Using StringBuilder for concatenation of strings

```
using System;
using System.Text;

namespace ConsoleAppDemos
{
    class StringBuilderDemo
    {
        static void Main()
        {
            string[] names = { "Ram", "Joseph", "Smith" };
            StringBuilder builder2 = new StringBuilder("These 
associates are required for Registration Process:").AppendLine();
            foreach (string item in names)
            {
                if(!string.IsNullOrEmpty(item))
                     builder2.Append(item).AppendLine();
            }
            Console.WriteLine(builder2.ToString());
            Console.ReadLine();
        } // end Main
    } // end class StringBuilderDemo
}
```

Output:

```
These associates are required for Registration Process:
Ram
Joseph
Smith
```

Working with Structures

Lab1: Declaring a structure with public members and accessing them in other classes using a structure variable

```
using System;

namespace ConsoleAppDemos
{
    struct student
    {
        public int rno;
        public string name;
        public float fee;
    }
    class program
    {
        static void Main()
        {
            student s1;
            s1.rno = 101;
            s1.name = "ABC";
            s1.fee = 1234f;
            CalculateFee_Display(s1);
            Console.ReadKey();
        }
        static void CalculateFee_Display(student s)
        {
            double totalFee = 0, annualFee = 1000, admissionFee =
20000;
            totalFee = annualFee + admissionFee + s.fee;
            Console.WriteLine("Rno={0}, Name={1}, TotalFee={2}",
s.rno, s.name, totalFee);
        }
    }
}
```

Working with Classes and Objects

Lab1: Declaring and Instantiating a class

```
using System;

namespace ConsoleApplication1
{
    class Student
    {
        //class members are private by default
        int rno;
        string name;
        float fee;
    }
    class program
    {
        static void Main()
        {
            //Instantiating the class Student
            Student s1=new Student();
            //gives error because privaet members of a class can't
be accessed outside the class
            s1.rno = 101;
            s1.name = "ABC";
            s1.fee = 1234f;
            Console.ReadKey();
        }
    }
}
```

Lab2: Declaring a class with Private fields, Creating methods to assign and retrieve values to/from private fields and Instantiating a class

```
using System;

namespace ConsoleAppDemos
{
    class Student
    {
        //class members are private by default
        int rno;
        string name;
        float fee;

        //Method to assign values to private fields
        public void SetData()
        {
            rno = 101;
            name = "ABC";
            fee = 1234f;
        }

        //Method to retrieve values from private fields
        public void GetData()
        {
            Console.WriteLine("Roll No={0}, Name={1}, Fee={2}", rno,
name, fee);
        }
    }

    class program
    {
        static void Main()
        {
            //Instantiating the class Student
            Student s1=new Student();
            s1.GetData(); //Displays the default values
            s1.SetData();
            s1.GetData();
            Console.ReadKey();
        }
    }
}
```

Output:

```
Roll No=0, Name=, Fee=0

Roll No=101, Name=ABC, Fee=1234
```

Lab3: Declaring a class with Private Fields, Creating method with parameters to assign dynamic values, Return value from a method to display values of private fields and Instantiating a class

```
using System;

namespace ConsoleAppDemos
{
    class Student
    {
        //class members are private by default
        int rno;
        string name;
        float fee;

        //Method to assign values to private fields
        public void SetData(int no, string n, float fee)
        {
            rno = no;
            name = n;
            //this keyword is used to refer class-level variable.
When the local variable and class variable names are same, to
differentiate the class variable from local variable, we use ‘this’
            this.fee = fee;
        }


        //Method to retrieve values from private fields
        public string GetData()
        {
            //return "Rollno=" + rno + ",name=" + name + ",Fee=" +
tutionfee;
            return string.Format("Rollno={0}, Name={1}, Fee={2}",
rno, name, fee);
        }
    }

    class School
    {
        static void Main()
        {
            int no;
            string name;
            float fee;
            Console.WriteLine("Enter Rollnumber");
            no = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter name of the student");
            name = Console.ReadLine();
            Console.WriteLine("Enter fees");
            fee = Convert.ToSingle(Console.ReadLine());
            //Instantiating the class Student
            Student s1 = new Student();
            //Calling a method with parameters
            s1.SetData(no, name, fee);
            //Calling a method that returns value
            Console.WriteLine(s1.GetData());
            Console.ReadKey();
        }
    }
}
```

Lab4: Declaring a class with Private Fields, Creating Properties to assign / retrieve values into/from private fields and Instantiating a class

```
using System;

namespace ConsoleAppDemos
{
    class Student
    {
        //class members are private by default
        int rno;
        string name;
        float fee;

        public int RollNo
        {
            get
            {
                return rno;
            }
            set
            {
                rno = value;
            }
        }

        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }

        //validation in assigning value to private field in a
property
        public float Fee
        {
            get
            {
                return fee;
            }
            set
            {
                if (value >= 1000)
                    fee = value;
                else
                    fee = 0;
            }
        }
    }

    class School
    {
        static void Main()
        {
            //Instantiating the class Student
            Student s1 = new Student();
            Console.WriteLine("Enter Rollnumber");
            s1.RollNo = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter name of the student");
            s1.Name = Console.ReadLine();
            Console.WriteLine("Enter fees");
            s1.Fee = Convert.ToSingle(Console.ReadLine());
            Console.WriteLine("Rollno={0},name={1},fee={2}",
s1.RollNo, s1.Name, s1.Fee);
            Console.ReadKey();
        }
    }
}
```

Important points about Constructors:

1. Are used to initialize the data members of a class.

2. If no constructor is defined in a class, System will provide a default constructor.

3. The role of default constructor is to initialize the data members of a class with default values.

4. are special methods i.e. methods with no return type and its name should be same as **class name**.

5. are invoked automatically when an object is created for a class.

6. can be defined with / without parameters.

7. If a constructor is defined without any parameters, it is called as default constructor.

8. If it is defined with parameters, it is called as parameterised constructor.

9. Access level can be specified by adding the access modifier before constructor name. Default is private.

10. When a class is defined with the constructor, system will not provide the default constructor.

Lab5: Declaring a class with Private Fields, Creating Constructors (Default and Parameterized) to assign values into private fields, a method to return values from private fields and Instantiating a class

```
using System;

namespace ConsoleAppDemos
{
    class Student
    {
        //class members are private by default
        int rno;
        string name;
        float fee;

        //Default Constructor - A constructor without parameters is
called Default Constructor
        public Student()
        {
            rno = 101;
            name = "ABC";
            fee = 1034.23f;
        }

        public Student(int rno, string name, float fee)
        {
            //when local variable and class variable names are same,
use this keyword to refer class variable in a method.
            this.rno = rno;
            this.name = name;
            this.fee = fee;
        }

        public string DisplayStudentDetails()
        {
            return string.Format("Rollno={0},Name={1},Fee={2}", rno,
name, fee);
        }
    }

    class School
    {
        static void Main()
        {
            //Instantiating the class Student
            Student s1 = new Student(); //invokes default 
constructor
            Console.WriteLine(s1.DisplayStudentDetails());
            Student s2 = new Student(102, "John", 3343.34f); 
//invokes parameterised constructor
            Console.WriteLine(s2.DisplayStudentDetails());
            Console.ReadKey();
        }
    }
}
```

Inheritance and Polymorphism

Lab1: Program that demonstrates Simple Inheritance with default and parameterised constructor

```
using System;

namespace ConsoleAppsDemo
{
    class Employee
    {
        int empno;
        string name;
        protected float sal;
        public Employee()
        {
            empno = 101;
            name = "Smith";
            sal = 10000f;
        }
        public Employee(int empno, string name, float sal)
//parameterised constructor
        {
            this.empno = empno;
            this.name = name;
            this.sal = sal;
        }
        public override string ToString()
        {
            return string.Format("Employee ID={0}, Name={1},
Salary={2}", empno, name, sal);
        }
        public double CalculateBonus()
        {
            return sal + (sal * .10);
        }
    }
    class Manager : Employee
    //Employee is called as base/parent/super class
    //Manager is called as derived/child/sub class
    {
        internal int deptno;
        //Child Class constructor by default invokes base class
default constructor.
        public Manager()
        {
            deptno = 10;
        }

        //invokes base class default constructor automatically
        public Manager(int deptno)
        {
            this.deptno = deptno;
        }

        //calling base class parameterised constructor by child
class constructor using base keyword
        public Manager(int empno, string name, float sal,int deptno)
:base(empno,name,sal)//default constructor
        {
            this.deptno = deptno;
        }

        public override string ToString()
        {
            return base.ToString() + " and Deptno=" + deptno;
        }
        public new double CalculateBonus()
        {
            return sal + (sal * .20);
        }
    }

    class SimpleInheritanceDemo
    {
        static void Main()
        {
            Employee e1 = new Employee(102,"John",5620f);
            Console.WriteLine(e1);

            //invokes child class parameterized constructor, which
internally invoke base class default constructor
            Manager m1 = new Manager(20);
            Console.WriteLine(m1);
            //invokes child class parameterized constructor, which
internally invokes base class parameterized constructor as per its
definition (because it is using Constructor-Call-Constructor)
technique
            Manager m2 = new Manager(103, "Jones", 43343f, 20);
            Console.WriteLine(m2);


            //parent class reference can hold the child class
objects
            Employee m3 = new Manager(); //invokes default
constructor in both the classes
            //m1.deptno = 10; //error, because m1 is a reference of
base class which don't have Deptno field definition

            //casting m1 to child class
            if (m3 is Manager)
            {
                Manager m = (Manager)m3;
                m.deptno = 20;
            }
            Console.WriteLine(m3);
            Console.WriteLine("Salary with bonus=" +
m1.CalculateBonus());
            Console.ReadKey();
        }
    }
}
```

Lab2: Program that demonstrates Static Polymorphism using Method Overloading

```
using System;

namespace ConsoleAppDemos
{
    class MethodOverloadingDemo
    {
        public int Addition(int a, int b)
        {
            return (a + b);
        }
        public int Addition(int a, int b, int c)
        {
            return (a + b + c);
        }
        public float Addition(float a, float b)
        {
            return (a + b);
        }
        public float Addition(float a, float b, float c)
        {
            return (a + b + c);
        }
    }

    //Now we can use those Addition method of 4 types
    class TestMethodOverloading
    {
        public static void Main()
        {
            MethodOverloadingDemo obj = new MethodOverloadingDemo();
            Console.WriteLine("Addition of two
integers::::::::::::::::" + obj.Addition(2, 5));
            Console.WriteLine("Addition of two double type
values::::::" + obj.Addition(0.40f, 0.50f));
            Console.WriteLine("Addition of three
integers::::::::::::::" + obj.Addition(2, 5, 5));
            Console.WriteLine("Addition of three double type
values::::" + obj.Addition(0.40f, 0.50f, 0.60f));
            Console.ReadKey();
        }
    }
}
```

Lab3: Program that demonstrates between Static Polymorphism using New Keyword and Dynamic Polymorphism using Virtual Keyword

```
using System;

namespace ConsoleAppDemos
{
    class BaseClass
    {
        public void StaticPolyMethod()
        {
            //body
            Console.WriteLine("Method is defined in base class - 
Static Polymorphism");
        }

        public virtual void DynamicPolyMethod()
        {
            //body
            Console.WriteLine("Method is defined in base class - 
Dynamic Polymorphism");
        }

    }

    class ChildClass : BaseClass
    {
        public new void StaticPolyMethod()
        {
            //body
            Console.WriteLine("Method is defined in child class - 
Static Polymorphism");
        }

        public override void DynamicPolyMethod()
        {
            //body
            Console.WriteLine("Method is defined in child class - 
Dynamic Polymorphism");
        }

    }
    class Test
    {
        static void Main()
        {
            ChildClass c1 = new ChildClass();

Console.WriteLine("==================================================
========================");
            Console.WriteLine("Direct reference - Child Class object 
pointed by Child class reference");
Console.WriteLine("===================================================
==========================");
            c1.StaticPolyMethod();
            c1.DynamicPolyMethod();

            Console.WriteLine("\n");
            BaseClass c2 = new ChildClass();

Console.WriteLine("===================================================
==========================");
            Console.WriteLine("Indirect reference - Child Class
object pointed by Base class refernce");
Console.WriteLine("===================================================
==========================");
            c2.StaticPolyMethod();
            c2.DynamicPolyMethod();
            Console.ReadKey();
        }
    }
}
```

Output:

```
======================================================================
Direct reference - Child Class object pointed by Child class reference
======================================================================

Method is defined in child class - Static Polymorphism

Method is defined in child class - Dynamic Polymorphism



======================================================================
Indirect reference - Child Class object pointed by Base class reference
======================================================================

Method is defined in base class - Static Polymorphism

Method is defined in child class - Dynamic Polymorphism
```

Lab4: Program that demonstrates abstract class

```
using System;

namespace ConsoleAppDemos
{

    //Abstract class can consist of one or more abstract methods
    abstract class AbstractClass
    {
        public void DefinedMethod()
        {
            //body
            Console.WriteLine("Method is defined in abstract 
class");
        }
        //abstract method - method with no definition, only 
declaration
        //abstract methods should be marked with abstract keyword
        public abstract void UndefinedMethod();
    }


    //To provide the definitions for abstract class methods, a class 
has to be created which should inherit from abstract class
    //A class which inherits from abstract class, should provide 
definitions for all of its abstract methods using Override keyword.
    class ChildClass : AbstractClass
    {
        public override void UndefinedMethod()
        {
            Console.WriteLine("Undefined method of abstract class is 
defined in child class");
        }

    }
    class abstractDemo
    {
        static void Main()
        {
            //AbstractClass obj=new AbstractClass();
            AbstractClass obj = new ChildClass();
            obj.DefinedMethod();
```

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

Output:

```
Method is defined in abstract class

Undefined method of abstract class is defined in child class
```

Lab5: Program that demonstrates Interfaces

```
using System;

namespace ConsoleAppDemos
{
    interface ICalc
    {
        int add(int a, int b);
        int diff(int a, int b);
    }
    interface ICalculator : ICalc
    {
        int product(int a, int b);
        int division(int a, int b);
    }
    class Impl_Interface : ICalculator
    {

        public int add(int a, int b)
        {
            return a + b;
        }

        public int diff(int a, int b)
        {
            return a - b;
        }

        public int product(int a, int b)
        {
            return a * b;
        }

        public int division(int a, int b)
        {
            return a / b;
        }
    }
    class InterfaceDemo
    {
        static void Main()
        {
            ICalculator i1 = new Impl_Interface();
            Console.WriteLine("{0} + {1} = {2}", 10, 20, i1.add(10,
20));
            Console.WriteLine("{1} - {0} = {2}", 10, 20, i1.diff(20,
10));
            Console.WriteLine("{0} * {1} = {2}", 10, 20,
i1.product(10, 20));
            Console.WriteLine("{1} / {0} = {2}", 10, 20,
i1.division(20, 10));
            Console.ReadKey();
        }
    }
}
```

Lab6: Program that demonstrates Multiple Inheritance with Interfaces

```
using System;

namespace ConsoleApplication1
{
    interface IFather
    {
        void MyProperty();
    }
    interface IMother
    {
        void MyProperty();
    }
    class Child : IFather, IMother
    {
        //explicit method definition for a specific interface
        void IFather.MyProperty()
        {
            Console.WriteLine("Use property for Charity");
        }
        void IMother.MyProperty()
        {
            Console.WriteLine("Use property for Business");
        }
        //Own class method
        public void SayHello()
        {
            Console.WriteLine("Hello World");
        }
    }
    class InterfaceDemo2
    {
        static void Main()
        {
            //Child c1 = new Child();

            IMother i2 = new Child();
            i2.MyProperty();

            IFather i1;
            i1 = new Child();
            i1.MyProperty();
            //i1.SayHello(); error because i1 is a reference of
IFather interface which don't have definition for SayHello().
            //casting base reference to child type to invoke Child
class method
            Child c1 = (Child)i1;
            c1.SayHello();
            Console.ReadKey();
        }
    }
}
```

Collections

Lab1: Program that demonstrates ArrayList concepts.

```
using System;
using System.Collections;

namespace ConsoleAppDemos
{
    class Product
    {
        int _productID;
        string _name;
        float _price;

        public int ProductID
        {
            get { return _productID; }
            set { _productID = value; }
        }

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        public float Price
        {
            get { return _price; }
            set { _price = value; }
        }

        public Product()
        {
            _productID = 101;
            _name = "Toys";
            _price = 10.34f;
        }

        public Product(int pid, string name,float cost)
        {
            _productID = pid;
            _name = name;
            _price = cost;
        }

    }
    class ArrayListDemo
    {
        static void Main()
        {
            //ArrayList is a class presented in System.Collections
namespace
            ArrayList al = new ArrayList(10);
            al.Add(new Product
{ProductID=10,Name="Soaps",Price=42f}); //Properties are called on
Product object
            al.Add(new Product()); //calls the default constructor
of Product class
            al.Add(new Product(11,"Key Chain",20f)); //calls the
parameterised constructor of Product class
            al.Add(3);
            al.Add("ABC");
            al.Add(false);
            Console.WriteLine("\nCapacity of the arraylist={0}",
al.Capacity);
            Console.WriteLine("No. of items in arraylist =" +
al.Count);

            /*
             foreach (Product p in al)
                Console.WriteLine("Product ID={0}, Name={1},
Price={2}", p.ProductID, p.Name, p.Price);

             * Displays InvalidCastException at 4th item, as this
item can't converted into Product.
             *
             * ArrayList stores all the values into Object, so it
can store any type of value in it. If the data to be stored in type-
specific format, use GenericCollections.

            */

            //Referring the items using foreach loop
            foreach (object o in al)
                Console.Write(o + "  ");
            Console.WriteLine("\n\n");

            //Referring the items using index with for loop
            for (int i = 0; i < al.Count; i++)
                Console.Write(al[i] + "  ");
            Console.WriteLine("\n\n");

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

Lab2: Program that demonstrates Hashtable concepts.

```
using System;
using System.Collections;

namespace ConsoleAppDemos
{
    class HashtableDemo
    {
        static void Main()
        {
            //Hashtable is a class presented in System.Collections
namespace. It is a collection of key-and-value pairs. The key is
used to access the items in the collection
            Hashtable ht = new Hashtable(10);
            ht.Add("001", "Zara Ali");
            ht.Add("002", "Abida Rehman");
            ht.Add("003", "Joe Holzner");
            ht.Add("004", "Mausam Benazir Nur");
            ht.Add("005", "M. Amlan");
            ht.Add("006", "M. Arif");
            ht.Add("007", "Ritesh Saikia");
           // ht.Add("001", "Amit"); Error because Key is duplicated

            //Referring the items using foreach loop
            Console.WriteLine("Displaying the items of Hashtable");
            foreach (DictionaryEntry de in ht)
            {
                Console.WriteLine(de.Key + " - " + de.Value);
            }

            Console.WriteLine("\n\n");
            //Referring the values using Keys
            Console.WriteLine("Displaying the items of Hashtable 
using Keys");
            foreach (string k in ht.Keys)
            {
                Console.WriteLine(k + " - " + ht[k]);
            }

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

Generic Collections

Lab1: Program that demonstrates List concepts.

```
using System;
using System.Collections.Generic;

namespace ConsoleApplication1
{
    class Product
    {
        int _productID;
        string _name;
        float _price;

        public int ProductID
        {
            get { return _productID; }
            set { _productID = value; }
        }

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        public float Price
        {
            get { return _price; }
            set { _price = value; }
        }

        public Product()
        {
            _productID = 101;
            _name = "Toys";
            _price = 10.34f;
        }

        public Product(int pid, string name,float cost)
        {
            _productID = pid;
            _name = name;
            _price = cost;
        }

    }
    class ListDemo
    {
        static void Main()
        {
            //List is a class presented in
System.Collections.Generic namespace
            List<Product> productsList = new List<Product>(10);
            productsList.Add(new Product { ProductID = 10, Name =
"Soaps", Price = 42f }); //Properties are called on Product object
            productsList.Add(new Product()); //calls the default
constructor of Product class
            productsList.Add(new Product(11, "Key Chain", 20f));
//calls the parameterised constructor of Product class
            //productsList.Add(3); Compile time error, as the list
is defined to store only Products. If any other datatype value is
provided, displays an error message

            Console.WriteLine("\nCapacity of the List={0}",
productsList.Capacity);
            Console.WriteLine("No. of items in List =" +
productsList.Count);

            //Referring the items using foreach loop
            Console.WriteLine("Displaying the items of List with the
help of foreach");
            foreach (Product p in productsList)
                Console.WriteLine("Product ID={0}, Name={1},
Price={2}", p.ProductID, p.Name, p.Price);



            Console.WriteLine("\n\n");
            Console.WriteLine("Displaying the items of List using
Index");
            //Referring the items using index with for loop
            for (int i = 0; i < productsList.Count; i++)
```

```
                Console.WriteLine("Product ID={0}, Name={1}, 
Price={2}", productsList[i].ProductID, productsList[i].Name, 
productsList[i].Price);
            Console.WriteLine("\n\n");

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

Lab2: Program that demonstrates Dictionary concepts.

```
using System;
using System.Collections.Generic;

namespace ConsoleAppDemos
{
    class Product
    {
        int _productID;
        string _name;
        float _price;

        public int ProductID
        {
            get { return _productID; }
            set { _productID = value; }
        }

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        public float Price
        {
            get { return _price; }
            set { _price = value; }
        }
        public Product()
        {
            _productID = 101;
            _name = "Toys";
            _price = 10.34f;
        }

        public Product(int pid, string name,float cost)
        {
            _productID = pid;
            _name = name;
            _price = cost;
        }

    }
    class DictionaryDemo
    {
        static void Main()
        {
            //Dictionary is a class presented in
System.Collections.Generic namespace
            Dictionary<int, Product> productsList = new
Dictionary<int, Product>();
            productsList.Add(10, new Product { ProductID = 10, Name
= "Soaps", Price = 42f }); //Properties are called on Product object
            productsList.Add(101, new Product()); //calls the
default constructor of Product class
            productsList.Add(11, new Product(11, "Key Chain", 20f));
//calls the parameterised constructor of Product class
            //productsList.Add(101, new Product()); // Run time
error, as the key is already existed

            Console.WriteLine("No. of items in Dictionary =" +
productsList.Count);

            //Referring the items using foreach loop
            Console.WriteLine("Displaying the items of Dictionary
with the help of foreach");
            foreach (KeyValuePair<int,Product> p in productsList)
                Console.WriteLine("Key:{0}, - Product Details:
Product ID={1}, Name={2}, Price={3}", p.Key, p.Value.ProductID,
p.Value.Name, p.Value.Price);
```

```
            Console.WriteLine("\n\n");
            Console.WriteLine("Displaying the items of Dictionary
using Keys");
            //Referring the items using index with for loop
            foreach(int k in productsList.Keys)
                Console.WriteLine("Key:{0}, - Product Details:
Product ID={1}, Name={2}, Price={3}", k, productsList[k].ProductID,
productsList[k].Name, productsList[k].Price);
            Console.WriteLine("\n\n");

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

Scenario based Problem Statement

Food Panda Company is looking for an e-commerce Order Management System to allow the user to perform below mentioned operations:

1. **Place Order**
2. **View Order Details for the Placed Order**
3. **Count total no. of orders placed for each variety of food**
4. **Modify Order Details**
5. **Change Order Status**
6. **Cancel the Order**

So, Design an Order class with following fields and define Properties, Default Constructor and Parameterised Constructors to access them.

- **OrderID (Auto-generated Field)**
- **OrderedBy**
- **Quantity**
- **Order Status (Placed/Shipping/Delivered)**
- **ProductID**

1. **Place Order – Collect the Order Details from the User and print the Order ID generated to hold these details in the system.**
2. **View Order Details – Accept the OrderID from the User and print the details of that Order if existed, Otherwise display error message**
3. **Modify Order Details - Accept OrderID and Quantity from the User and modify the details of that Order if existed and if the status of the Order is Placed, Otherwise display error message**
4. **Cancel Order – Accept Order ID from the user and Cancel the order if existed, otherwise display error message.**
5. **Count Orders of a product – Accept Product ID from the user and display the total number of orders placed for that food if existed, Otherwise display error message**
6. **Change Order Status – Used by Admin to change the status of Order if order is in placed or shipping. Placed – Shipping – Delivered.**

Steps:

1. **Create a class to design Order Type**
2. **Create a class to design OrderManagement Type**
3. **Create a class to test OrderManagement**

Order.CS:

```
using System;

namespace ConsoleApplication1
{
    public enum OrderStatus { Placed, Shipping, Delivered };
    public class Order
    {
        static int _oid = 1000;
        int _pid, _qty;
        string _orderedBy;
        OrderStatus _status;

        public Order()
        {
            _oid++;
            _pid = 10;
            _qty = 5;
            _orderedBy = "Kethi";
            _status = OrderStatus.Placed;
        }

        public Order(int pid, int qty, string orderedBy)
        {
            _oid++;
            _pid = pid;
            _qty = qty;
            _orderedBy = orderedBy;
            _status = OrderStatus.Placed;
        }

        //Order ID should be a read-only property as Order ID value
will be generated automatically
```

```
public int OrderID
{
    get
    {
        return _oid;
    }
}

public int ProductID
{
    get
    {
        return _pid;
    }
    set
    {
        _pid = value;
    }
}

public int Quantity
{
    get
    {
        return _qty;
    }
    set
    {
        _qty = value;
    }
}

public string OrderedBy
{
    get
    {
        return _orderedBy;
    }
    set
    {
        _orderedBy = value;
    }
}
```

```
        public OrderStatus Status
        {
            get
            {
                return _status;
            }
            set
            {
                _status = value;
            }
        }

        public override string ToString()
        {
            return string.Format("Order Details:\nOrder
ID={0},\nProduct ID={1},\nOrdered By={2},\nQuantity
Ordered={3},\nOrder Status={4}", _oid, _pid, _orderedBy, _qty,
_status);

        }
    }
}
```

<u>OrderManagement.CS</u>

```
using System;
using System.Collections.Generic;


namespace ConsoleApplication1
{
    public class OrderManagement
    {
        static List<Order> OrderDB = new List<Order>();

        public int PlaceOrder(Order o)
        {
            foreach (Order or in OrderDB)
            {
                if (or.OrderID == o.OrderID)
                    return 0;
            }
            OrderDB.Add(o);
            return o.OrderID;
        }

        public bool DeleteOrder(int orderID)
        {
            if (OrderDB.Count == 0)
                return false;
            else
            {
                foreach (Order or in OrderDB)
                {
                    if (or.OrderID == orderID)
                    {
                        OrderDB.Remove(or);
                        return true;
                    }
                }
                return false;
            }
        }
        public int CountOrders_ProductID(int pID)
        {
            int count = 0;
            foreach (Order or in OrderDB)
            {
                if (or.ProductID == pID)
                    count++;
            }
            return count;
        }

        public Order ViewOrderDetails(int orderID)
        {
            Order obj = null;
            foreach (Order or in OrderDB)
            {
                if (or.OrderID == orderID)
                {
                    obj = or;
                }
            }
            return obj;
        }
        public bool ModifyOrderDetails(int orID, int qty)
        {
            if (OrderDB.Count == 0)
                return false;
            else
            {
                foreach (Order or in OrderDB)
                {
                    if (or.OrderID == orID && or.Status ==
OrderStatus.Placed)
                    {
                        or.Quantity = qty;
                        return true;
                    }
                }
                return false;
            }
        }

        public bool ChangeOrderStatus()
        {
            if (OrderDB.Count == 0)
                return false;
            else
            {
                foreach (Order or in OrderDB)
                {
                    if (or.Status == OrderStatus.Placed)
                    {
                        or.Status = OrderStatus.Shipping;
                    }
                    else if (or.Status == OrderStatus.Shipping)
                    {
                        or.Status = OrderStatus.Delivered;
                    }
                }
                return true;
            }
        }
    }
}
```

TestOrderManagementApplication.CS

```
using System;

namespace ConsoleApplication1
{
    class TestOrderMgmtSystem
    {
        static void Main(string[] args)
        {
            string action;
            do
            {
                Console.Clear();
                Order obj;

Console.WriteLine("=================================================
===========================");
                Console.WriteLine("********** Welcome to Food Panda
Order Management System **********");

Console.WriteLine("=================================================
===========================");
                 Console.WriteLine("\n\n We serve the following
food:\n1.Veg.Manchuria \n2.Chicken Manchuria \n3.Veg. Biryani
\n4.Chicken Biryani \n5.Fish Curry \n6.Mutton Biryani\n\n");
                Console.WriteLine("Enter Your choice:\n1.Place Order
\n2.Remove Order \n3.View Order Details of a particular Order
\n4.View total no. of Orders placed for a particular Product
\n5.Modify Order Details of a particular Order \n6.Change Order
Status\n");
                int ch = Convert.ToInt32(Console.ReadLine());
                OrderManagement ormg = new OrderManagement();
                switch (ch)
                {
                    case 1:
                        obj=new Order();
                        Console.WriteLine("Enter your Name");
                        obj.OrderedBy = Console.ReadLine();
                        Console.WriteLine("Enter food ID to be
ordered");
                        obj.ProductID =
Convert.ToInt32(Console.ReadLine());
                        Console.WriteLine("Enter quantity");
                        obj.Quantity = 
Convert.ToInt32(Console.ReadLine());
                        int id= ormg.PlaceOrder(obj);
                        if (id>0)
                            Console.WriteLine("Order Placed 
Successfully with the ID: {0}", id);
                        else
                            Console.WriteLine("Sorry, Can't place an 
order right now..");
                        break;

                    case 2:
                        Console.WriteLine("Enter Order ID to cancel 
the Order");
                        int 
orID=Convert.ToInt32(Console.ReadLine());
                        bool result = ormg.DeleteOrder(orID);
                        if (result)
                            Console.WriteLine("Order is Cancelled. 
You can place another Order");
                        else
                            Console.WriteLine("Sorry, Order details 
are not existed");
                        break;

                    case 3:
                        Console.WriteLine("Enter Order ID to view 
the Order details");
                        orID=Convert.ToInt32(Console.ReadLine());
                        obj = ormg.ViewOrderDetails(orID);
                        if (obj == null)
                            Console.WriteLine("Sorry, Order details 
are not existed");
                        else
                            Console.WriteLine(obj);
                        break;

                    case 4:
                        Console.WriteLine("Enter Product ID to view 
its total no. of orders placed");
                        int prID = 
Convert.ToInt32(Console.ReadLine());
                        int total=ormg.CountOrders_ProductID(prID);
                        if (total == 0)
```

```
                            Console.WriteLine("No orders placed yet
for {0} product",prID);
                        else
                            Console.WriteLine("Total No. of Orders
placed for {0} is {1}", prID, total);
                        break;
                    case 5:
                        Console.WriteLine("Enter Order ID to modify
its Order details");
                        orID=Convert.ToInt32(Console.ReadLine());
                        Console.WriteLine("Enter quantity to be
ordered");
                        int qty =
Convert.ToInt32(Console.ReadLine());
                        result = ormg.ModifyOrderDetails(orID, qty);
                        if (result)
                            Console.WriteLine("Order details
modified successfully");
                        else
                            Console.WriteLine("Sorry, Details can't
be modified because either the Order is shipped/Delivered or the
Order might have been Cancelled. Please check and Try Again...");
                        break;
                    case 6:
                        ormg.ChangeOrderStatus();
                        break;
                    default:
                        Console.WriteLine("Invalid Choice");
                        break;
                }
                Console.WriteLine("Do you want to continue (Y/N)?");
                action = Console.ReadLine().ToUpper();
            } while (action[0] == 'Y');

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