Contents

Hands on Lab document for C#

Beginning with basic programs.

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

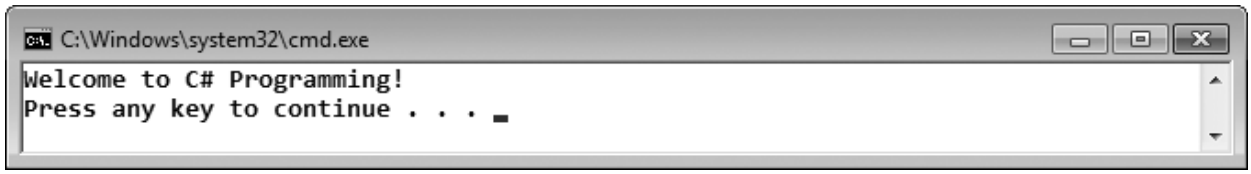
Steps: Create a new console application name Welcome1.

In the Welcome1.cs file update the Main method with the following code.

```
// Welcome1.cs
// Text-displaying application.
using System;

public class Welcome1
{
  // Main method begins execution of C# application
  public static void Main( string[] args )
  {
    Console.WriteLine( "Welcome to C# Programming!" );
    Console.ReadLine();
  } // end Main
} // end class Welcome1
```

Output :



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

Steps: Create a new console application name Welcome2.

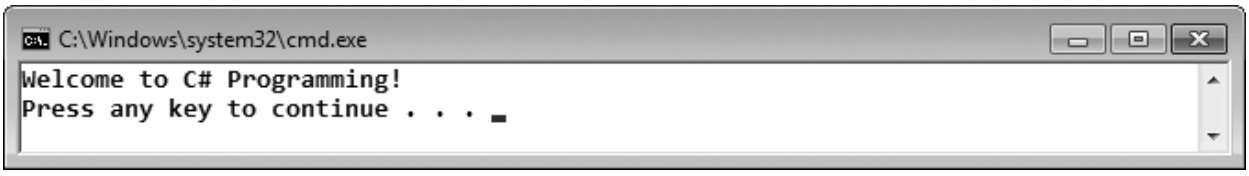
In the Welcome2.cs file update the Main method with the following code.

```
// Welcome2.cs
// Displaying one line of text with multiple statements.
using System;

public class Welcome2
{
  // Main method begins execution of C# application
```

```
  public static void Main(string[] args)
  {
    Console.Write("Welcome to ");
    Console.WriteLine("C# Programming!");
    Console.ReadLine();
  } // end Main
} // end class Welcome2
```

Output:



Lab 3: Display Welcome message as a multiple line with single statements.

Steps: Create a new console application name Welcome3.

In the Welcome3.cs file update the Main method with the following code.

```
// Welcome3.cs
// Displaying multiple lines with a single statement.
using System;

public class Welcome3
{
  // 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 Welcome3
```

Output:

```
Welcome
to
C#
Programming
```

Session 3: Data types and Control Flow in C#

Lab1: Reading inputs from the console. Below program takes input from the user for two numbers and displays the sum on to the console.

Steps : Create a new console application named Addition. Rename Program.cs to Addition.cs. In the Addition.cs file update the Main method with the following code.

```
// Addition.cs
// Displaying the sum of two numbers input from the keyboard.
using System;

public class Addition
{
  // Main method begins execution of C# application
  public static void Main( string[] args )
  {
    int number1; // declare first number to add
    int number2; // declare second number to add
    int sum; // declare sum of number1 and number2

    Console.Write( "Enter first integer: " ); // prompt user
    // read first number from user
    number1 = Convert.ToInt32( Console.ReadLine() );

    Console.Write( "Enter second integer: " ); // prompt user
    // read second number from user
    number2 = Convert.ToInt32( Console.ReadLine() );

    sum = number1 + number2; // add numbers

    Console.WriteLine( "Sum is {0}", sum ); // display sum
    Console.ReadLine();
  } // end Main
} // end class Addition
```

Output:

```
Enter first integer: 15
Enter second integer: 10
Sum is 25
```

Lab2 : Use of conditional,relational statements and equality

Steps : Create a new console application named Comparison. In the Comparison.cs file update the Main method with the following code.

```
 // Comparison.cs
// Comparing integers using if statements, equality operators,
// and relational operators.
using System;
public class Comparison
{
  // Main method begins execution of C# application
  public static void Main(string[] args)
  {
    int number1; // declare first number to compare
    int number2; // declare second number to compare

    // prompt user and read first number
    Console.Write("Enter first integer: ");
    number1 = Convert.ToInt32(Console.ReadLine());

    // prompt user and read second number
    Console.Write("Enter second integer: ");
    number2 = Convert.ToInt32(Console.ReadLine());

    if (number1 == number2)
      Console.WriteLine("{0} == {1}", number1, number2);

    if (number1 != number2)
      Console.WriteLine("{0} != {1}", number1, number2);

    if (number1 < number2)
      Console.WriteLine("{0} < {1}", number1, number2);

    if (number1 > number2)
      Console.WriteLine("{0} > {1}", number1, number2);

    if (number1 <= number2)
      Console.WriteLine("{0} <= {1}", number1, number2);

    if (number1 >= number2)
      Console.WriteLine("{0} >= {1}", number1, number2);
```

```
    Console.ReadLine();
  } // end Main
} // end class Comparison
```

Output:

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

Lab3 : Summing integers (2 to 20) using for statement

```
// Sum.cs
// Summing integers with the for statement.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SumUsingForLoop
{
        class Sum
        {

                public static void Main(string[] args)
                {

                        // total even integers from 2 through 20

                        int total = 0; // initialize total
                        for (int number = 2; number <= 20; number += 2)
                                total += number;


                        Console.WriteLine("Sum is {0}", total); // display results
                        Console.ReadLine();
                } // end Main
```

```
        } // end class Sum
}
```

Lab 4 : .do..while repetition statement

```
// DoWhileTest.cs
// do...while repetition statement.
using System;

public class DoWhileTest
{
        public static void Main(string[] args)
        {
                int counter = 1; // initialize counter
                do
                {
                        Console.Write("\n{0}", counter);
                        ++counter;
                } while (counter <= 10); // end do...while
                Console.WriteLine(); // outputs a newline
                Console.ReadLine();
        } // end Main
} // end class DoWhileTest
```

Lab 5 : Create GradeBook object, input grades and display grade report using Switch statement

```
// GradeBook.cs
// GradeBook class uses switch statement to count letter grades.

using System;

public class GradeBook
{
        private int total; // sum of grades
        private int gradeCounter; // number of grades entered
        private int aCount; // count of A grades
        private int bCount; // count of B grades
        private int cCount; // count of C grades
        private int dCount; // count of D grades
```

```
private int fCount; // count of F grades
// automatic property CourseName
public string CourseName { get; set; }

// constructor initializes automatic property CourseName;
// int instance variables are initialized to 0 by default
public GradeBook(string name)
{
        CourseName = name; // set CourseName to name
} // end constructor

// display a welcome message to the GradeBook user
public void DisplayMessage()
{
        // CourseName gets the name of the course
        Console.WriteLine("Welcome to the grade book for\n{0}!\n",
        CourseName);
} // end method DisplayMessage
// input arbitrary number of grades from user
public void InputGrades()
{
        int grade; // grade entered by user
        string input; // text entered by the user

        Console.WriteLine("{0}\n{1}",
        "Enter the integer grades in the range 0-100.",
        "Type <Ctrl> z and press Enter to terminate input:");

        input = Console.ReadLine(); // read user input

        // loop until user enters the end-of-file indicator (<Ctrl> z)
        while (input != null)
        {
                grade = Convert.ToInt32(input); // read grade off user input
                total += grade; // add grade to total
                ++gradeCounter; // increment number of grades

                // call method to increment appropriate counter
                IncrementLetterGradeCounter(grade);

                input = Console.ReadLine(); // read user input
        } // end while
} // end method InputGrades
```

```
// add 1 to appropriate counter for specified grade
private void IncrementLetterGradeCounter(int grade)
{
        // determine which grade was entered
        switch (grade / 10)
        {
                case 9: // grade was in the 90s
                case 10: // grade was 100
                        ++aCount; // increment aCount
                        break; // necessary to exit switch
                case 8: // grade was between 80 and 89
                        ++bCount; // increment bCount
                        break; // exit switch
                case 7: // grade was between 70 and 79
                        ++cCount; // increment cCount
                        break; // exit switch
                case 6: // grade was between 60 and 69
                        ++dCount; // increment dCount
                        break; // exit switch
                default: // grade was less than 60
                        ++fCount; // increment fCount
                        break; // exit switch
        } // end
} // end method IncrementLetterGradeCounter
// display a report based on the grades entered by the user
public void DisplayGradeReport()
{
        Console.WriteLine("\nGrade Report:");

        // if user entered at least one grade...
        if (gradeCounter != 0)
        {
                // calculate average of all grades entered
                double average = (double)total / gradeCounter;

                // output summary of results
                Console.WriteLine("Total of the {0} grades entered is {1}",
                gradeCounter, total);
                Console.WriteLine("Class average is {0:F}", average);
                Console.WriteLine("{0}A: {1}\nB: {2}\nC: {3}\nD: {4}\nF: {5}",
                "Number of students who received each grade:\n",
                aCount, // display number of A grades
                bCount, // display number of B grades
                cCount, // display number of C grades
```

```
                    dCount, // display number of D grades
                    fCount); // display number of F grades
                } // end if
                else // no grades were entered, so output appropriate message
                    Console.WriteLine("No grades were entered");
                    Console.ReadLine();
        } // end method DisplayGradeReport
} // end class GradeBook


// GradeBookTest.cs
// Create GradeBook object, input grades and display grade report.

using System;
public class GradeBookTest
{
        public static void Main(string[] args)
        {
                // create GradeBook object myGradeBook and
                // pass course name to constructor
                GradeBook myGradeBook = new GradeBook(
                 "CS101 Introduction to C# Programming");

                myGradeBook.DisplayMessage(); // display welcome message

                myGradeBook.InputGrades(); // read grades from user
                myGradeBook.DisplayGradeReport(); // display report based on grades
        } // end Main
} // end class GradeBookTest
```

Output

```
Welcome to the grade book for
CS101 Introduction to C# Programming!

Enter the integer grades in the range 0-100.
Type <Ctrl> z and press Enter to terminate input:
55
25
```

```
99
68
78
55
^Z

Grade Report:
Total of the 6 grades entered is 380
Class average is 63.33
Number of students who received each grade:
A: 1
B: 0
C: 1
D: 1
F: 3
```

Lab 6 : break statement exiting a for statement.

```
//  BreakTest.cs
// break statement exiting a for statement.
using System;

public class BreakTest
{
        public static void Main(string[] args)
        {
                int count; // control variable also used after loop terminates

                for (count = 1; count <= 10; count++) // loop 10 times
                {
                        if (count == 5) // if count is 5,

                                break; // terminate loop
                                Console.Write("{0} ", count);
                } // end for

                Console.WriteLine("\nBroke out of loop at count = {0}", count);
                Console.ReadLine();
```

```
        } // end Main
} // end class BreakTest
```

Lab 7 : continue statement terminating an iteration of a for statement

```
// ContinueTest.cs
// continue statement terminating an iteration of a for statement.
using System;

public class ContinueTest
{
        public static void Main(string[] args)
        {
                for (int count = 1; count <= 10; count++) // loop 10 times
                {
                        if (count == 5) // if count is 5,

                                continue; // skip remaining code in loop
                        Console.Write("{0} ", count);
                } // end for

                Console.WriteLine("\nUsed continue to skip displaying 5");
                Console.ReadLine();
        } // end Main
} // end class ContinueTest
```

Lab 8 : Program that uses while loop with condition

```
using System;

class Program
{
   static void Main()
   {
        // Continue in while loop until index is equal to ten.
        int i = 0;
        while (i < 10)
        {
           Console.Write("While statement ");
           // Write the index to the screen.
           Console.WriteLine(i);
```

```
            // Increment the variable.
            i++;
        }
        Console.ReadLine();
    }
}
```

Lab 9 : Program that uses true in while loop

```
//Program.cs
using System;

class Program
{
    static void Main()
    {
        int index = 0;
        //
        // Continue looping infinitely until internal condition is met.
        //
        while (true)
        {
            int value = ++index;
            //
            // Check to see if limit it hit.
            //
            if (value > 5)
            {
                Console.WriteLine("While loop break");
                break;
            }
            //
            // You can add another condition.
            //
            if (value > 100)
            {
                throw new Exception("Never hit");
            }
            //
            // Write to the screen on each iteration.
            //
            Console.WriteLine("While loop statement");
        }
```

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

Lab 10 : Program that assigns variable in while condition

```
using System;

class Program
{
        static void Main()
        {
                int value = 4;
                int i;
                // You can assign a variable in the while loop condition statement.
                while ((i = value) >= 0)
                {
                        // In the while loop body, both i and value are equal.
                        Console.WriteLine("While {0} {1}", i, value);
                        value--;
                }
                Console.Read();
        }
}
```

Lab 11: Program that uses loop and switch with break

```
using System;

class Program
{
        static void Main()
        {
                for (int i = 0; i < 5; i++) // Loop through five numbers.
                {
                        switch (i) // Use loop index as switch expression.
                        {
                                case 0:
                                case 1:
                                case 2:
```

```
                {
                    Console.WriteLine("First three");
                    break;
                }
            case 3:
            case 4:
                {
                    Console.WriteLine("Last two");
                    break;
                }
        }
    }
    Console.Read();
  }
}
```

Math Object

Lab 1 : Program that uses Math method

```
using System;

class Program
{
   static void Main()
   {
        // Use Math method.
        double sin = Math.Sin(2.5);
        Console.WriteLine(sin);
        Console.Read();
   }
}
```

Lab 2 : Program that computes absolute values

```
using System;

class Program
{
   static void Main()
   {
        //
```

```
        // Compute two absolute values.
        //
        int value1 = -1000;
        int value2 = 20;
        int abs1 = Math.Abs(value1);
        int abs2 = Math.Abs(value2);
        //
        // Write integral results.
        //
        Console.WriteLine(value1);
        Console.WriteLine(abs1);
        Console.WriteLine(value2);
        Console.WriteLine(abs2);
        //
        // Compute two double absolute values.
        //
        double value3 = -100.123;
        double value4 = 20.20;
        double abs3 = Math.Abs(value3);
        double abs4 = Math.Abs(value4);
        //
        // Write double results.
        //
        Console.WriteLine(value3);
        Console.WriteLine(abs3);
        Console.WriteLine(value4);
        Console.WriteLine(abs4);
        Console.Read();
    }
}
```

Lab 3 : Program that uses Ceiling

```
using System;

class Program
{
        static void Main()
        {
                // Get ceiling of double value.
                double value1 = 123.456;
                double ceiling1 = Math.Ceiling(value1);
```

```
            // Get ceiling of decimal value.
            decimal value2 = 456.789M;
            decimal ceiling2 = Math.Ceiling(value2);

            // Get ceiling of negative value.
            double value3 = -100.5;
            double ceiling3 = Math.Ceiling(value3);

            // Write values.
            Console.WriteLine(value1);
            Console.WriteLine(ceiling1);
            Console.WriteLine(value2);
            Console.WriteLine(ceiling2);
            Console.WriteLine(value3);
            Console.WriteLine(ceiling3);
            Console.Read();
        }
}
```

Lab 4 : Program that uses Math.Floor

```
using System;

class Program
{
        static void Main()
        {
            //
            // Two values.
            //
            double value1 = 123.456;
            double value2 = 123.987;
            //
            // Take floors of these values.
            //
            double floor1 = Math.Floor(value1);
            double floor2 = Math.Floor(value2);

            //
            // Write first value and floor.
            //
```

```
            Console.WriteLine(value1);
            Console.WriteLine(floor1);
            //
            // Write second value and floor.
            //
            Console.WriteLine(value2);
            Console.WriteLine(floor2);
            Console.Read();
        }
}
```

Lab 5 : Example program that uses Math .Max

```
using System;

class Program
{
        static int[] _array = new int[]
  {
        1,
        2,
        5,
        6,
        7
  };

        static void Main()
        {
            // Get 1 place from end.
            int i = GetLastIndex(1);
            Console.WriteLine(i);

            // Get 3 places from end.
            i = GetLastIndex(3);
            Console.WriteLine(i);

            // Get 10 places from end. Will not throw exception.
            i = GetLastIndex(10);
            Console.WriteLine(i);
            Console.Read();
```

```
        }

        static int GetLastIndex(int length)
        {
                int start = _array.Length - length - 1;
                start = Math.Max(0, start);
                return _array[start];
        }
}
```

Lab 6: Program that uses Math.Sqrt

```
using System;

class Program
{
        static double[] _lookup = new double[5];

        static void Main()
        {
                // Compute square roots by calling Math.Sqrt.
                double a = Math.Sqrt(1);
                double b = Math.Sqrt(2);
                double c = Math.Sqrt(3);
                double d = Math.Sqrt(4);

                // Store square roots in lookup table.
                var lookup = _lookup;
                lookup[1] = a;
                lookup[2] = b;
                lookup[3] = c;
                lookup[4] = d;

                Console.WriteLine(a);
                Console.WriteLine(b);
                Console.WriteLine(c);
                Console.WriteLine(d);

                Console.WriteLine(lookup[1]);
                Console.WriteLine(lookup[2]);
                Console.WriteLine(lookup[3]);
                Console.WriteLine(lookup[4]);
```

```
            Console.Read();
        }
}
```

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);

   }

}
```

Output

STATION

station

Lab2: Split the input string on spaces

```
using System;


class Program
{
   static void Main()
   {
            string s = "there is a cat";
            //
            // Split string on spaces.
            // This will separate all the words.
            //
            string[] words = s.Split(' ');
            foreach (string word in words)
            {
               Console.WriteLine(word);
            }
   }
}
```

Output

```
there
is
a
cat
```

Lab3: Display the length of a string.

```
Using system;

class MainClass
{
  static void Main()
  {
     string palindrome;

     System.Console.Write("Enter a palindrome: ");
     palindrome = System.Console.ReadLine();

     System.Console.WriteLine("The palindrome, \"{0}\" is {1} characters.",palindrome, palindrome.L
ength);
  }
}
```

Lab4: Add formatted string to StringBuilder

```
using System;
using System.Text;

class MainClass
{
   public static void Main()
   {
      StringBuilder sb = new StringBuilder();
      int number = 1;

      sb.AppendFormat("{0}: {1} ", number++, "another string");

      Console.WriteLine("{0}", sb);
   }
}
```

Output

```
1: another string
```

Lab5: Concatenate two specified strings and create a new string (WindowsApplication).

```
using System;
using System.Windows.Forms;

namespace WindowsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         string str1 = null;
         string str2 = null;

         str1 = "Concat() ";
         str2 = "Test";
         MessageBox.Show(string.Concat(str1, str2));
      }
   }
}
```

Output

When you run this C# source code you will get "Concat() Test " in message box.

Lab6: Use string.Format to combine three strings with formatting options.

```
using System;

class Program
```

```
{
   static void Main()
   {
            // Declare three variables that we will use in the format method.
            // The values they have are not important.

            string value1 = "Dot Net Manual";
            int value2 = 10000;
            DateTime value3 = new DateTime(2007, 11, 1);

            // Use string.Format method with four arguments.
            // The first argument is the formatting string.
            // It specifies how the next three arguments are formatted.

            string result = string.Format("{0}: {1:0.0} - {2:yyyy}",
               value1,
               value2,
               value3);

            // Write the result.

            Console.WriteLine(result);
   }
}
```

Output

```
Dot Net Manual: 10000.0 – 2007
```

Lab7: Use string.Format with a number

```
using System;

class Program
{
   static void Main()
   {
            // Format a ratio as a percentage string.
            // You must specify the percentage symbol in the format string.
            // It will multiply the value by 100 for you.
            double ratio = 0.76;
            string result = string.Format("string = {0:0.0%}",
               ratio);
            Console.WriteLine(result);
   }
}
```

Output

```
string = 76.0%
```

Lab8: Use string.Format for padding.

```
using System;

class Program
{
   static void Main()
   {
            // The constant formatting string.
            //  It specifies the padding.
            // A negative number means to left-align.
            // A positive number means to right-align.

            const string format = "{0,-10} {1,10}";

            // Construct the strings.

            string line1 = string.Format(format,
               100,
               5);
            string line2 = string.Format(format,
               "Carbon",
               "Paper");

            // Write the formatted strings.

            Console.WriteLine(line1);
            Console.WriteLine(line2);
   }
}
```

Output

```
100              5
Carbon       Paper
```

Lab9: Use StringBuilder in a foreach loop.

```
using System;
using System.Text;

class Program
{
   static void Main()
   {
            string[] items = { "Clay", "Marble", "Sand" };

            StringBuilder builder2 = new StringBuilder(
```

```
            "These items are required:").AppendLine();

        foreach (string item in items)
        {
            builder2.Append(item).AppendLine();
        }
        Console.WriteLine(builder2.ToString());
        Console.ReadLine();
    }
}
```

Output

```
These items are required:
Clay
Marble
Sand
```

Lab10: Use AppendFormat on StringBuilder type.

```
using System;
using System.Text;

class Program
{
    static int[] _v = new int[]
    {
        1,
        4,
        6
    };

    static void Main()
    {
        StringBuilder b = new StringBuilder();
        foreach (int v in _v)
        {
            b.AppendFormat("int: {0:0.0}{1}", v,
                Environment.NewLine);
        }
        Console.WriteLine(b.ToString());
    }
}
```

Output

```
int: 1.0
int: 4.0
int: 6.0
```

Session 6: Classes and Objects in C#

Lab1: Declaring a Class with a Method and Instantiating an Object of a Class

Steps : Create a new console application name GradeBook. Rename the Program.cs file to GradeBook.cs.

In the GradeBook.cs file write the following code.

```
//GradeBook.cs
// Class declaration with one method.
using System;
public class GradeBook
{
  // display a welcome message to the GradeBook user
  public void DisplayMessage()
  {
    Console.WriteLine( "Welcome to the Grade Book!" );
    Console.ReadLine();
  } // end method DisplayMessage
} // end class GradeBook
```

Add another class file name as GradeBookTest.cs. Add following code.

```
//GradeBookTest.cs
// Create a GradeBook object and call its DisplayMessage method.
public class GradeBookTest
{
  // Main method begins program execution
  public static void Main( string[] args )
  {
    // create a GradeBook object and assign it to myGradeBook
    GradeBook myGradeBook = new GradeBook();

    // call myGradeBook's DisplayMessage method
```

```
    myGradeBook.DisplayMessage();
  } // end Main
} // end class GradeBookTest
```

Output:

```
Welcome to the Grade Book!
```

Lab2 : Class declaration with a method that has a parameter

Steps : Create a new console application name GradeBook. Rename the Program.cs file to GradeBook.cs.

In the GradeBook.cs file write the following code.

```
// GradeBook.cs
// Class declaration with a method that has a parameter.
using System;
public class GradeBook
{
  // display a welcome message to the GradeBook user
  public void DisplayMessage( string courseName )
  {
    Console.WriteLine( "Welcome to the grade book for\n{0}!",  courseName );
   Console.ReadLine();
  } // end method DisplayMessage
} // end class GradeBook
```

Add another class file name as GradeBookTest.cs. Add following code.

```
// GradeBookTest.cs
// Create a GradeBook object and pass a string to
// its DisplayMessage method.
using System;
public class GradeBookTest
{
  // Main method begins program execution
  public static void Main( string[] args )
  {
```

```
    // create a GradeBook object and assign it to myGradeBook
    GradeBook myGradeBook = new GradeBook();

    // prompt for and input course name
    Console.WriteLine( "Please enter the course name:" );
    string nameOfCourse = Console.ReadLine(); // read a line of text
    Console.WriteLine(); // output a blank line

    // call myGradeBook's DisplayMessage method
    // and pass nameOfCourse as an argument
    myGradeBook.DisplayMessage( nameOfCourse );
  } // end Main
} // end class GradeBookTest
```

Output:

```
Please enter the course name:
C# Programming
Welcome to the grade book for
C# Programming!
```

Lab3 : Class with an Instance Variable and a Property

Steps : Create a new console application name GradeBook. Rename the Program.cs file to GradeBook.cs.

In the GradeBook.cs file write the following code.

```
// GradeBook.cs
// GradeBook class that contains a private instance variable, courseName,
// and a public property to get and set its value.
using System;
public class GradeBook
{
  private string courseName; // course name for this GradeBook

  // property to get and set the course name
  public string CourseName
  {
    get    {      return courseName;    } // end get
    set    {      courseName = value;    } // end set
  } // end property CourseName
```

```
  // display a welcome message to the GradeBook user
  public void DisplayMessage()
  {
    // use property CourseName to get the  name of the course that this GradeBook represents
    Console.WriteLine( "Welcome to the grade book for\n{0}!",   CourseName ); // display property
CourseName
    Console.ReadLine();
  } // end method DisplayMessage
} // end class GradeBook
```

Add another class file name as GradeBookTest.cs. Add following code.

```
// GradeBookTest.cs
// Create and manipulate a GradeBook object.
using System;
public class GradeBookTest
{
  // Main method begins program execution
  public static void Main( string[] args )
  {
    // create a GradeBook object and assign it to myGradeBook
    GradeBook myGradeBook = new GradeBook();

    // display initial value of CourseName
    Console.WriteLine( "Initial course name is: '{0}'\n",
      myGradeBook.CourseName );

    // prompt for and read course name
    Console.WriteLine( "Please enter the course name:" );
    myGradeBook.CourseName = Console.ReadLine(); // set CourseName
    Console.WriteLine(); // output a blank line

    // display welcome message after specifying course name
    myGradeBook.DisplayMessage();
  } // end Main
} // end class GradeBookTest
```

Output

```
Initial course name is: ''
Please enter the course name:
C# Programming
Welcome to the grade book for
C# Programming!
```

Lab 4 : Program that demonstrates interface

```
using System;

interface IPerl
{
   void Read();
}

class Test : IPerl
{
   public void Read()
   {
         Console.WriteLine("Read");
   }
}

class Program
{
   static void Main()
   {
         IPerl perl = new Test(); // Create instance.
         perl.Read(); // Call method on interface.
   }
}
```

Lab 5: Program that demonstrates abstract class

```
using System;

abstract class Test
{
```

```
        public int _a;
        public abstract void A();
}

class Example1 : Test
{
        public override void A()
        {
                Console.WriteLine("Example1.A");
                base._a++;
        }
}

class Example2 : Test
{
        public override void A()
        {
                Console.WriteLine("Example2.A");
                base._a--;
        }
}

class Program
{
        static void Main()
        {
                // Reference Example1 through Test type.
                Test test1 = new Example1();
                test1.A();

                // Reference Example2 through Test type.
                Test test2 = new Example2();
                test2.A();
                Console.ReadLine();
        }
}
```

Output:

Example1.A

Example2.A

Lab 6 : Program that demonstrates this as argument

```
using System;

class Net
{
   public string Name { get; set; }
   public Net(Perl perl)
   {
         // Use name from Perl instance.
         this.Name = perl.Name;
   }
}

class Perl
{
   public string Name { get; set; }
   public Perl(string name)
   {
         this.Name = name;
         // Pass this instance as a parameter!
         Net net = new Net(this);
         // The Net instance now has the same name.
         Console.WriteLine(net.Name);
         Console.ReadLine();
   }
}

class Program
{
   static void Main()
   {
         Perl perl = new Perl("C#");
   }

}
```

Lab 7 : Program that demonstrates sealed class

```
using System;

/// <summary>
/// Example interface.
```

```
/// </summary>
interface ITest
{
        /// <summary>
        /// Method required by the interface.
        /// </summary>
        int GetNumber();
}

/// <summary>
/// Non-sealed class that implements an interface.
/// </summary>
class TestA : ITest
{
        /// <summary>
        /// Interface implementation.
        /// </summary>
        public int GetNumber()
        {
                return 1;
        }
}

/// <summary>
/// Sealed class that implements an interface.
/// </summary>
sealed class TestB : ITest
{
        /// <summary>
        /// Interface implementation.
        /// </summary>
        public int GetNumber()
        {
                return 2;
        }
}

class Program
{
        static void Main()
        {
                ITest test1 = new TestA(); // Regular class
                ITest test2 = new TestB(); // Sealed instantiation
                Console.WriteLine(test1.GetNumber()); // TestA.GetNumber
```

```
            Console.WriteLine(test2.GetNumber()); // TestB.GetNumber
            Console.ReadLine();
        }
}
```

Lab 8 : Program that introduces virtual method

```
using System;

class A
{
        public virtual void Test()
        {
            Console.WriteLine("A.Test");
        }
}

class B : A
{
        public override void Test()
        {
            Console.WriteLine("B.Test");
        }
}

class Program
{
        static void Main()
        {
            // Compile-time type is A.
            // Runtime type is A as well.
            A ref1 = new A();
            ref1.Test();

            // Compile-time type is A.
            // Runtime type is B.
            A ref2 = new B();
            ref2.Test();
            Console.ReadLine();
        }
}
```

Session 7: Inheritance and Interfaces

Lab1: Inheritance and Interfaces.

This below code features a simple way to perform inheritance. It creates a class called Circle that contains calculations for a circle based on its radius. It calculates the diameter, the circumference, and the area. Then, a class called Sphere inherits from the Circle class.

Steps: Create a new console application name FlatShapes. Rename the Program.cs file to Circle.cs.

In the Circle.cs file write the following code.

```
//Circle.cs
namespace FlatShapes
{
        class Circle
        {
                private double _radius;
                public double Radius
                {
                        get { return (_radius < 0) ? 0.00 : _radius; }
                        set     { _radius = value; }
                }
                public double Diameter
                {
                        get     { return Radius * 2; }
                }
                public double Circumference
                {
                        get     { return Diameter * 3.14159; }
                }
                public double Area
```

```
                {
                        get     { return Radius * Radius * 3.14159; }
                }
        }
}
```

Add another class file with name Sphere.cs. Add following code.

```
//Sphere.cs
namespace Volumes
{
   class Sphere : FlatShapes.Circle
   {
      new public double Area
      {
         get { return 4 * Radius * Radius * 3.14159; }
      }

      public double Volume
      {
         get { return 4 * 3.14159 * Radius * Radius * Radius / 3; }
      }
   }}
```

Add one more class file with name Excercise.cs. Add following code.

```
//Excercise.cs
using System;
using Volumes;
using FlatShapes;
class Exercise
{
   static void Show(Circle round)
   {
      Console.WriteLine("Circle Characteristics");
      Console.WriteLine("Side:    {0}", round.Radius);
      Console.WriteLine("Diameter: {0}", round.Diameter);
      Console.WriteLine("Circumference: {0}", round.Circumference);
      Console.WriteLine("Area:    {0}", round.Area);
   }
```

```
    static void Show(Sphere ball)
    {
        Console.WriteLine("\nSphere Characteristics");
        Console.WriteLine("Side:    {0}", ball.Radius);
        Console.WriteLine("Diameter: {0}", ball.Diameter);
        Console.WriteLine("Circumference: {0}", ball.Circumference);
        Console.WriteLine("Area:    {0}", ball.Area);
        Console.WriteLine("Volume: {0}\n", ball.Volume);
        Console.ReadLine();
    }

    public static int Main()
    {
        FlatShapes.Circle c = new FlatShapes.Circle();
        Volumes.Sphere s = new Volumes.Sphere();

        c.Radius = 20.25;
        Show(c);

        s.Radius = 20.25;
        Show(s);

        return 0;
    }
}
```

Output:

Circle Characteristics Side: 25.55 Diameter: 51.1 Circumference: 160.535249 Area: 2050.837805975	Sphere Characteristics Side: 25.55 Diameter: 51.1 Circumference: 160.535249 Area: 8203.3512239 Volume: 69865.2079235483

Lab2 : Overriding Methods.

Create three classes shape, circle and rectangle where circle and rectangle are inherited from the class shape and overrides the methods Area() and Circumference() that are declared as virtual in Shape class.(Polymorphism)

Steps: Create a new console application name ProgramCall. Rename the Program.cs file to Shape.cs

In Shape.cs add following code.

```
//Shape.cs
using System;
namespace ProgramCall
{
   class Shape
   {
      protected float R, L, B;
      public virtual float Area()
      {
         return 3.14F * R * R;
      }
      public virtual float Circumference()
      {
         return 2 * 3.14F * R;
      }
   }
   }
```

Add one class file name Circle.cs. And add the following code.

```
//Circle.cs
using System;
namespace ProgramCall
{
   class Circle : Shape
   {
      public void GetRadius()
      {
         Console.Write("Enter  Radius  :  ");
         R = float.Parse(Console.ReadLine());
      }
   }
}
```

Add one more class file name Rectangle.cs. Add the below code.

```
//Rectangle.cs
using System;
namespace ProgramCall
{
   class Rectangle : Shape
   {
      public void GetLB()
      {
         Console.Write("Enter  Length   :  ");
         L = float.Parse(Console.ReadLine());
         Console.Write("Enter Breadth : ");
         B = float.Parse(Console.ReadLine());
      }
      public override float Area()
      {
         return L * B;
      }
      public override float Circumference()
      {
         return 2 * (L + B);
      }
   }

}
```

Add new class file name MainClass.cs file with below code.

```
//Rectangle.cs
using System;

namespace ProgramCall
{
   class Rectangle : Shape
   {
      public void GetLB()
      {
         Console.Write("Enter  Length   :  ");
         L = float.Parse(Console.ReadLine());
         Console.Write("Enter Breadth : ");
         B = float.Parse(Console.ReadLine());
      }
```

```
    public override float Area()
    {
      return L * B;
    }

    public override float Circumference()
    {
      return 2 * (L + B);
    }
  }
}
```

Output:

```
Enter Length : 10
Enter Breadth : 20
Area : 200
Circumference : 60
Enter Radius : 25
Area : 1962.5
Circumference : 157
```

Session 8: Generics and Collections

Lab1: Generic Methods.

The program demonstrates how Generic method returns the largest of three objects.

Steps: Create new console application name Generics.
Rename program.cs to MaximumTest.cs. Add below code.

```
//MaximumTest.cs    Generic method Maximum returns the largest of three objects.

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

```
        Console.WriteLine("Maximum of {0}, {1} and {2} is {3}\n",  3, 4, 5, Maximum(3, 4, 5));
        Console.WriteLine("Maximum of {0}, {1} and {2} is {3}\n", 6.6, 8.8, 7.7, Maximum(6.6, 8.8,
7.7));
        Console.WriteLine("Maximum of {0}, {1} and {2} is {3}\n", "pear", "apple", "orange",
        Maximum("pear", "apple", "orange"));
        Console.ReadLine();
      } // end Main

      // generic function determines the
      // largest of the IComparable objects
      private static T Maximum<T>(T x, T y, T z)
      where T : IComparable<T>
      {
         T max = x; // assume x is initially the largest

         // compare y with max
         if (y.CompareTo(max) > 0)
            max = y; // y is the largest so far

         // compare z with max
         if (z.CompareTo(max) > 0)
            max = z; // z is the largest

         return max; // return largest object
      } // end method Maximum
   } // end class MaximumTest
}
```

Output:

```
Maximum of 3, 4 and 5 is 5
Maximum of 6.6, 8.8 and 7.7 is 8.8
Maximum of pear, apple and orange is pear
```

Lab2: Working with ArrayList Class.

Steps: Create new console application ArrayList. Rename Program.cs to ArraylistTest.cs & add below code.

```
// ArrayListTest.cs
// Using class ArrayList.
using System;
```

```
using System.Collections;
public class ArrayListTest
{
    private static readonly string[] colors = { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
   private static readonly string[] removeColors = { "RED", "WHITE", "BLUE" };

   // create ArrayList, add colors to it and manipulate it
   public static void Main(string[] args)
   {// initial capacity of 1. add the elements of the colors array to the ArrayList list
      ArrayList list = new ArrayList(1);
      foreach (var color in colors)
         list.Add(color); // add color to the ArrayList list add elements in the removeColors array to /
          /the ArrayList removeList with the ArrayList constructor
      ArrayList removeList = new ArrayList(removeColors);

      Console.WriteLine("ArrayList: ");
      DisplayInformation(list); // output the list

      // remove from ArrayList list the colors in removeList
      RemoveColors(list, removeList);

      Console.WriteLine("\nArrayList after calling RemoveColors: ");
      DisplayInformation(list); // output list contents
      Console.Read();
   } // end Main

   // displays information on the contents of an array list
   private static void DisplayInformation(ArrayList arrayList)
   {
      // iterate through array list with a foreach statement
      foreach (var element in arrayList)
         Console.Write("{0} ", element); // invokes ToString


      // display the size and capacity
      Console.WriteLine("\nSize = {0}; Capacity = {1}", arrayList.Count, arrayList.Capacity);
      int index = arrayList.IndexOf("BLUE");
      if (index != -1)
         Console.WriteLine("The array list contains BLUE at index {0}.",
         index);
      else
         Console.WriteLine("The array list does not contain BLUE.");
      //Console.ReadLine();
   } // end method DisplayInformation
```

```
   // remove colors specified in secondList from firstList
   private static void RemoveColors(ArrayList firstList,
   ArrayList secondList)
   {
      // iterate through second ArrayList like an array
      for (int count = 0; count < secondList.Count; count++)
         firstList.Remove(secondList[count]);
   } // end method RemoveColors
} // end class ArrayListTest
```

Output:

```
ArrayList:
MAGENTA RED WHITE BLUE CYAN
Size = 5; Capacity = 8
The array list contains BLUE at index 3.
ArrayList after calling RemoveColors:
MAGENTA CYAN
Size = 2; Capacity = 8
The array list does not contain BLUE.
```

Lab3: Using Generic Collections with an Example of the List<T> and Dictionary<TKey,TValue> Generic Collections.

Steps: Crate a new console application name CollectionSample. Add the following code in Program.cs.

```
//Program.cs
using System;
using System.Collections.Generic;
namespace CollectionSample
```

```
{
  public class Customer
 {
            public Customer(int id, string name)
            {
               ID = id;
               Name = name;
            }
            private int m_id;
            public int ID
            {
               get { return m_id; }
               set { m_id = value; }
            }
            private string m_name;

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

class Program
{
   static void Main(string[] args)
   {
      List<int> myInts = new List<int>();

      myInts.Add(1);
      myInts.Add(2);
      myInts.Add(3);

      for (int i = 0; i < myInts.Count; i++)
      {
         Console.WriteLine("MyInts: {0}", myInts[i]);
      }

      Dictionary<int, Customer> customers = new Dictionary<int, Customer>();

      Customer cust1 = new Customer(1, "Cust 1");
      Customer cust2 = new Customer(2, "Cust 2");
      Customer cust3 = new Customer(3, "Cust 3");
```

```
        customers.Add(cust1.ID, cust1);
        customers.Add(cust2.ID, cust2);
        customers.Add(cust3.ID, cust3);

        foreach (KeyValuePair<int, Customer> custKeyVal in customers)
        {
          Console.WriteLine(
            "Customer ID: {0}, Name: {1}",
            custKeyVal.Key,
            custKeyVal.Value.Name);
        }

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

Output:

```
MyInts: 1
MyInts: 2
MyInts: 3
Customer ID: 1, Name: Cust 1
Customer ID: 2, Name: Cust 2
Customer ID: 3, Name: Cust 3
```

Session 9: Exception Handling

Lab1: Divide By Zero and Format Exceptions.

Steps: Create a new console application name ExceptionHandling. Rename Program.cs to DivideByZeroExceptionHandling.cs. Add the below code.

```
//DivideByZeroExceptionHandling.cs
// FormatException and DivideByZeroException handlers.

using System;
namespace ExceptionHandling
```

```
{
   class DivideByZeroExceptionHandling
      {
         static void Main(string[] args)
         {
            bool continueLoop = true; // determines whether to keep looping
            do
            {
               // retrieve user input and calculate quotient
               try
               {
                  // Convert.ToInt32 generates FormatException
                  // if argument cannot be converted to an integer
                  Console.Write("Enter an integer numerator: ");
                  int numerator = Convert.ToInt32(Console.ReadLine());
                  Console.Write("Enter an integer denominator: ");
                  int denominator = Convert.ToInt32(Console.ReadLine());
                  // division generates DivideByZeroException
                  // if denominator is 0
                  int result = numerator / denominator;
                  // display result
                  Console.WriteLine("\nResult: {0} / {1} = {2}",
                  numerator, denominator, result);
                  continueLoop = false;
               } // end try
               catch (FormatException formatException)
               {
                  Console.WriteLine("\n" + formatException.Message);
                  Console.WriteLine(
                  "You must enter two integers. Please try again.\n");
               } // end catch
               catch (DivideByZeroException divideByZeroException)
               {
                  Console.WriteLine("\n" + divideByZeroException.Message);
                  Console.WriteLine(
                  "Zero is an invalid denominator. Please try again.\n");
               } // end catch
               Console.ReadLine();
            } while (continueLoop); // end do...while
         } // end Main
      } // end class DivideByZeroExceptionHandling
   }
```

Output:

```
Enter an integer numerator: 100
Enter an integer denominator: 0
Attempted to divide by zero.
Zero is an invalid denominator. Please try again.
Enter an integer numerator: 100
Enter an integer denominator: 7
Result: 100 / 7 = 14
```

```
Enter an integer numerator: 100
Enter an integer denominator: hello
Input string was not in a correct format.
You must enter two integers. Please try again.
Enter an integer numerator: 100
Enter an integer denominator: 7
Result: 100 / 7 = 14
```

Lab2: Using Finally Block.

The program demonstrates that the finally block always executes, regardless of whether an exception occurs in the corresponding try block. The program consists of method Main and four other methods that Main invokes to demonstrate finally. These methods are DoesNotThrowException, ThrowExceptionWithCatch, ThrowExceptionWithoutCatch and ThrowExceptionCatchRethrow

Steps: Create a new console application name FinallyBlockHandling. Rename Program.cs to UsingExceptions.cs. Add the below code.

```
//UsingExceptions.cs
// Using finally blocks.
// finally blocks always execute, even when no exception occurs.
using System;
namespace FinallyBlockHandling
{
   class UsingExceptions
   {
      static void Main()
      {
         // Case 1: No exceptions occur in called method
```

```
   Console.WriteLine("Calling DoesNotThrowException");
   DoesNotThrowException();

   // Case 2: Exception occurs and is caught in called method
   Console.WriteLine("\nCalling ThrowExceptionWithCatch");
   ThrowExceptionWithCatch();
   // Case 3: Exception occurs, but is not caught in called method
   // because there is no catch block.
   Console.WriteLine("\nCalling ThrowExceptionWithoutCatch");

   // call ThrowExceptionWithoutCatch
   try
   {
      ThrowExceptionWithoutCatch();
   } // end try
   catch
   {
      Console.WriteLine("Caught exception from " +
      "ThrowExceptionWithoutCatch in Main");
   } // end catch

   // Case 4: Exception occurs and is caught in called method,
   // then rethrown to caller.
   Console.WriteLine("\nCalling ThrowExceptionCatchRethrow");

   // call ThrowExceptionCatchRethrow
   try
   {
      ThrowExceptionCatchRethrow();
   } // end try
   catch
   {
      Console.WriteLine("Caught exception from " +
      "ThrowExceptionCatchRethrow in Main");
      Console.ReadLine();

   } // end catch
} // end method Main

// no exceptions thrown
static void DoesNotThrowException()
{
   // try block does not throw any exceptions
   try
```

```
   {
      Console.WriteLine("In DoesNotThrowException");
   } // end try
   catch
   {
      Console.WriteLine("This catch never executes");
   } // end catch
   finally
   {
      Console.WriteLine("finally executed in DoesNotThrowException");
   } // end finally
   Console.WriteLine("End of DoesNotThrowException");
   // Console.ReadLine();
} // end method DoesNotThrowException
// throws exception and catches it locally
static void ThrowExceptionWithCatch()
{
   // try block throws exception
   try
   {
      Console.WriteLine("In ThrowExceptionWithCatch");

   } // end try
   catch (Exception exceptionParameter)
   {
      Console.WriteLine("Message: " + exceptionParameter.Message);
   } // end catch
   finally
   {
      Console.WriteLine(
      "finally executed in ThrowExceptionWithCatch");
   } // end finally
   Console.WriteLine("End of ThrowExceptionWithCatch");
} // end method ThrowExceptionWithCatch

// throws exception and does not catch it locally
static void ThrowExceptionWithoutCatch()
{
   // throw exception, but do not catch it
   try
   {
      Console.WriteLine("In ThrowExceptionWithoutCatch");

   } // end try
```

```
      finally
      {
         Console.WriteLine("finally executed in " +
         "ThrowExceptionWithoutCatch");
      } // end finally
      // unreachable code; logic error
      Console.WriteLine("End of ThrowExceptionWithoutCatch");
   } // end method ThrowExceptionWithoutCatch

   // throws exception, catches it and rethrows it
   static void ThrowExceptionCatchRethrow()
   {
      // try block throws exception
      try
      {
         Console.WriteLine("In ThrowExceptionCatchRethrow");
         throw new Exception("Exception in ThrowExceptionCatchRethrow");
      } // end try
      catch (Exception exceptionParameter)
      {
         Console.WriteLine("Message: " + exceptionParameter.Message);
         // rethrow exception for further processing
         throw;
         // unreachable code; logic error
      } // end catch
      finally
      {
         Console.WriteLine("finally executed in " +
         "ThrowExceptionCatchRethrow");
      } // end finally
      // any code placed here is never reached
      Console.WriteLine("End of ThrowExceptionCatchRethrow");
   } // end method ThrowExceptionCatchRethrow
} // end class UsingExceptions

}
```

Output:

```
Calling DoesNotThrowException
In DoesNotThrowException
finally executed in DoesNotThrowException
End of DoesNotThrowException
Calling ThrowExceptionWithCatch
In ThrowExceptionWithCatch
Message: Exception in ThrowExceptionWithCatch
finally executed in ThrowExceptionWithCatch
End of ThrowExceptionWithCatch
Calling ThrowExceptionWithoutCatch
In ThrowExceptionWithoutCatch
```

Session 10: File Handling in C# 3.5

Lab1:Using FileWriter class.

Steps: Create a console application name FileWriter. Rename Program.cs to FileWrite.cs and add following code.

```
// FileWrite.cs
//Program to write user input to a file using StreamWriter Class
using System;
using System.Text;
using System.IO;

namespace FileWriter
{
      class FileWrite
      {
         public void WriteData()
         {
            FileStream fs = new FileStream("c:\\test.txt", FileMode.Append, FileAccess.Write);
            StreamWriter sw = new StreamWriter(fs);
            Console.WriteLine("Enter the text which you want to write to the file");
           string str = Console.ReadLine();
            sw.WriteLine(str);
            sw.Flush();
            sw.Close();
            fs.Close();
```

```
    }

  static void Main(string[] args)

  {
    FileWrite wr = new FileWrite();
    wr.WriteData();

  }   }}
```

Output: User Entered text will get write in a c:\\test.txt file**.**

Lab2: Read from a file using StreamReader Class.

Steps: Create a console application name FileReader. Rename Program.cs to FileRead.cs and add following code.

```
\\FileRead.cs
using System;
using System.IO;
namespace FileReader
{
  class FileRead
  {
    public void ReadData()
      {
        FileStream fs = new FileStream("c:\\test.txt", FileMode.Open, FileAccess.Read);
        StreamReader sr = new StreamReader(fs);
        Console.WriteLine("Program to show content of test file");
        sr.BaseStream.Seek(0, SeekOrigin.Begin);
        string str = sr.ReadLine();
        while (str != null)
        {
          Console.WriteLine(str);
          str = sr.ReadLine();
        }
        Console.ReadLine();
        sr.Close();
        fs.Close();
      }

  static void Main(string[] args)
  {
```

```
        FileRead wr = new FileRead();
        wr.ReadData();
    }
}

}
```

Output:

C:\\test.txt file content will get shown on console.

Lab 3 : Program that shows how to read & write in file.

```
using System;
using System.IO;

namespace FileHandlingApp
{
        class Program
        {
                static void Main(string[] args)
                {
                        if (File.Exists("D:\\test.txt"))
                        {
                                string content = File.ReadAllText("test.txt");
                                Console.WriteLine("Current content of file:");
                                Console.WriteLine(content);
                        }
                        Console.WriteLine("Please enter new content for the file:");
                        string newContent = Console.ReadLine();
                        File.WriteAllText("D:\\test.txt", newContent);
                }
        }
}
```

Lab 4 : Program opens a file or creates it if it does not already exist, and appends information to the end of the file

```
using System;
using System.IO;
using System.Text;

class FSOpenWrite
```

```
{
        public static void Main()
        {
                FileStream fs = new FileStream("c:\\Variables.txt", FileMode.Append,
FileAccess.Write, FileShare.Write);
                fs.Close();
                StreamWriter sw = new StreamWriter("c:\\Variables.txt", true, Encoding.ASCII);
                string NextLine = "This is the appended line.";
                sw.Write(NextLine);
                sw.Close();
        }
}
```

Lab 5 : Program that demonstrates some of the FileStream constructors.

```
using System;
using System.IO;
using System.Text;

class Test
{

        public static void Main()
        {
                string path = @"c:\Test.txt";

                // Delete the file if it exists.
                if (File.Exists(path))
                {
                        File.Delete(path);
                }

                //Create the file.
                using (FileStream fs = File.Create(path))
                {
                        AddText(fs, "This is some text");
                        AddText(fs, "This is some more text,");
                        AddText(fs, "\r\nand this is on a new line");
                        AddText(fs, "\r\n\r\nThe following is a subset of characters:\r\n");
```

```
                for (int i = 1; i < 120; i++)
                {
                        AddText(fs, Convert.ToChar(i).ToString());

                }
        }

        //Open the stream and read it back.
        using (FileStream fs = File.OpenRead(path))
        {
                byte[] b = new byte[1024];
                UTF8Encoding temp = new UTF8Encoding(true);
                while (fs.Read(b, 0, b.Length) > 0)
                {
                        Console.WriteLine(temp.GetString(b));
                }
        }
        Console.Read();
    }

    private static void AddText(FileStream fs, string value)
    {
        byte[] info = new UTF8Encoding(true).GetBytes(value);
        fs.Write(info, 0, info.Length);
    }
}
```

Session 11: Xml in .NET Framework

Lab1: Using XMLTextReader to read an XML file.

Read an XML file using XmlTextReader and call Read method to read its node one by one until end of the file.

Steps:

Create a sample xml file name books.xml on C:\ with following content.

```
<?xml version="1.0" encoding="utf-8"?>

<bookshop>
  <book>
      <title>The Autobiography of Benjamin Franklin</title>
      <author>
        <first-name>Benjamin</first-name>
        <last-name>Franklin</last-name>
      </author>
      <price>8.99</price>
  </book>
      <book>
      <title>The Confidence Man</title>
      <author>
        <first-name>Herman</first-name>
        <last-name>Melville</last-name>
      </author>
      <price>11.99</price>
  </book>
  <book>
      <title>The Gorgias</title>
      <author>
      <name>Plato</name>
      </author>
      <price> 9.99</price>
      </book>
</bookshop>
```

Create a new console application name ReadXMLSample. Add following code in Program.cs

```
//Program.cs
 using System;
using System.Xml;

namespace  ReadXMLSample
{
   /// <summary>
   /// Summary description for Class1.
   /// </summary>
   class Program
   {
      static void Main(string[] args)
      {
         XmlTextReader reader = new XmlTextReader("c:\\books.xml");
         while (reader.Read())
         {
            switch (reader.NodeType)
            {
```

```
                case XmlNodeType.Element: // The node is an element.
                    Console.Write("<" + reader.Name);
                    Console.WriteLine(">");
                    break;
                case XmlNodeType.Text: //Display the text in each element.
                    Console.WriteLine(reader.Value);
                    break;
                case XmlNodeType.EndElement: //Display the end of the element.
                    Console.Write("</" + reader.Name);
                    Console.WriteLine(">");
                    break;
            }
        }
        Console.ReadLine();
    }
  }
}
```

Output:

C:\books.xml files content will get shown on console.

Lab2: XmlTextWriter and StringWriter.

The program creates an array of four Tuple instances. These represent employee data in a very small company.

Steps: Create new console application. Add the following code in Program.cs.

```
//Program.cs
using System;
using System.IO;
using System.Xml;

class Program
{
  static void Main()
  {
    // Create an array of four Tuple instances containing employee data.
    var array = new Tuple<int, string, string, int>[4];
    array[0] = new Tuple<int, string, string, int>(1, "Dave", "Smith", 10000);
    array[1] = new Tuple<int, string, string, int>(3, "Mac", "Drinkwater", 30000);
    array[2] = new Tuple<int, string, string, int>(4, "Norman", "Miller", 20000);
    array[3] = new Tuple<int, string, string, int>(12, "Cecee", "Walker", 120000);
```

```
    // Use StringWriter as backing for XmlTextWriter.
    using (StringWriter str = new StringWriter())
    using (XmlTextWriter xml = new XmlTextWriter(str))
    {
      // Root.
      xml.WriteStartDocument();
      xml.WriteStartElement("List");
      xml.WriteWhitespace("\n");

      // Loop over Tuples.
      foreach (var element in array)
      {
        // Write Employee data.
        xml.WriteStartElement("Employee");
        {
          xml.WriteElementString("ID", element.Item1.ToString());
          xml.WriteElementString("First", element.Item2);
          xml.WriteWhitespace("\n  ");
          xml.WriteElementString("Last", element.Item3);
          xml.WriteElementString("Salary", element.Item4.ToString());
        }
        xml.WriteEndElement();
        xml.WriteWhitespace("\n");
      }

      // End.
      xml.WriteEndElement();
      xml.WriteEndDocument();

      // Result is a string.
      string result = str.ToString();
      Console.WriteLine("Length: {0}", result.Length);
      Console.WriteLine("Result: {0}", result);
      Console.Read();
    }
  }
}
```

Output:

```
Length: 441
```

Result:

```
<?xml version="1.0" encoding="utf-16"?>
<List>
  <Employee>
    <ID>1</ID>
    <First>Dave</First>
    <Last>Smith</Last>
    <Salary>10000</Salary>
  </Employee>
  <Employee>
    <ID>3</ID>
    <First>Mac</First>
    <Last>Drinkwater</Last>
    <Salary>30000</Salary>
  </Employee>
  <Employee>
    <ID>4</ID>
    <First>Norman</First>
    <Last>Miller</Last>
    <Salary>20000</Salary>
  </Employee>
  <Employee>
    <ID>12</ID>
    <First>Cecee</First>
    <Last>Walker</Last>
    <Salary>120000</Salary>
  </Employee>
</List>
```

Lab3: XmlAttributeCollection containing the list of attributes.

Steps: Create a new console application & add the below code in Program.cs

```
using System;
using System.IO;
using System.Xml;

public class Program
{
   public static void Main()
   {
      XmlDocument doc = new XmlDocument();
      doc.LoadXml("<book genre='Computer' ISBN='1-111111-11-1'>" +
                  "<title>C#</title>" +
                  "</book>");

      XmlElement root = doc.DocumentElement;
```

```
        root.Attributes[0].Value = "fiction";

        Console.WriteLine("Display the modified XML...");
        Console.WriteLine(doc.InnerXml);
        Console.Read();
    }
}
```

Output:

Display the modified XML...

```
<book genre="fiction" ISBN="1-111111-11-1"><title>C#</title></book>
```

Lab4: Using LoadXML method.

Steps: Create a new console application & add the below code in BookListing.cs

```
using System;
using System.IO;
using System.Xml;
public class BookListing
{
    public static void Main()
    {
        XmlDocument doc = new XmlDocument();
        String entry = "<book genre='biography'" +
         " ISBN='1111111111'><title>my title</title>" +
         "</book>";
        doc.LoadXml(entry);

        StringWriter writer = new StringWriter();
        doc.Save(writer); // to StringWriter
        String strXML = writer.ToString(); // to String
        Console.WriteLine(strXML);
        Console.Read();
    }
}
```

Output: Run the program & check the contents displayed on console.

Lab5: Creates a new XmlWriter instance using the specified stream.

Steps: Create a new console application & add the below code in Sample.cs

```
using System;
using System.IO;
using System.Xml;
using System.Text;

public class Sample
{
   public static void Main()
   {
      XmlWriter writer = null;
      try
      {
         XmlWriterSettings settings = new XmlWriterSettings();
         settings.Indent = true;
         settings.IndentChars = ("\t");
         settings.OmitXmlDeclaration = true;
         writer = XmlWriter.Create("data.xml", settings);
         writer.WriteStartElement("book");
         writer.WriteElementString("item", "tesing");
         writer.WriteEndElement();
         writer.Flush();
      }
      finally
      {
         if (writer != null)
            writer.Close();
      }
   }
}
```

Lab6: XmlWriter writes XML data out.

Steps: Create a new console application & add the below code in XMLSchemaExamples.cs

```
using System;
using System.IO;
```

```
using System.Text;
using System.Xml;
using System.Xml.Schema;

class XMLSchemaExamples
{
   public static void Main()
   {

      StringBuilder output = new StringBuilder();

      String xmlString =
            @"<?xml version='1.0'?>
              <!-- This is a sample XML document -->
              <Items>
               <Item>test with a child element <more/> stuff</Item>
              </Items>";
      using (XmlReader reader = XmlReader.Create(new StringReader(xmlString)))
      {
         XmlWriterSettings ws = new XmlWriterSettings();
         ws.Indent = true;
         using (XmlWriter writer = XmlWriter.Create(output, ws))
         {
            while (reader.Read())
            {
               switch (reader.NodeType)
               {
                  case XmlNodeType.Element:
                     writer.WriteStartElement(reader.Name);
                     break;
                  case XmlNodeType.Text:
                     writer.WriteString(reader.Value);
                     break;
                  case XmlNodeType.XmlDeclaration:
                  case XmlNodeType.ProcessingInstruction:
                     writer.WriteProcessingInstruction(reader.Name, reader.Value);
                     break;
                  case XmlNodeType.Comment:
                     writer.WriteComment(reader.Value);
                     break;
                  case XmlNodeType.EndElement:
                     writer.WriteFullEndElement();
                     break;
```

```
            }
          }

        }
      }
      Console.WriteLine(output.ToString());
      Console.Read();
    }
}
```

Lab7: Using XmlTextReader for parsing

Steps: Create a new console application & add the below code in Sample.cs

```
using System;
using System.IO;
using System.Xml;

public class Sample
{

   public static void Main()
   {

      XmlTextReader reader = new XmlTextReader("books.xml");
      reader.WhitespaceHandling = WhitespaceHandling.None;
      reader.MoveToContent();
      reader.Read();

      XmlTextWriter writer = new XmlTextWriter(Console.Out);
      writer.Formatting = Formatting.Indented;
      writer.WriteStartElement("myBooks");
      writer.WriteNode(reader, false);
      reader.Skip();
      writer.WriteNode(reader, false);
      writer.WriteEndElement();
      writer.Close();
      reader.Close();
   }
}
```

Lab8:Using XMLDocument for Parsing .

Steps: Create a new console application & add the below code in Sample.cs

```
using System;
using System.IO;
using System.Xml;

public class Sample
{
   public static void Main()
   {

      XmlDocument doc = new XmlDocument();
      doc.LoadXml("<book genre='novel' ISBN='1-111111-11-1'>" +
                  "<title>C#</title>" +
                  "</book>");

      XmlElement root = doc.DocumentElement;

      // Create a new element.
      XmlElement elem = doc.CreateElement("price");
      elem.InnerText = "19.95";

      Console.WriteLine(elem.OwnerDocument.OuterXml);

      // Add the new element into the document.
      root.AppendChild(elem);

      Console.WriteLine(doc.InnerXml);
      Console.Read();
   }
}
```

Output:

```
<book genre="novel" ISBN="1-111111-11-1"><title>C#</title></book>
<book genre="novel" ISBN="1-111111-11-1"><title>C#</title><price>19.95</price>
</book>
```

Session 12: Delegates in C#

Lab1: Using Delegates

The following example illustrates declaring, instantiating, and using a delegate. The BookDB class encapsulates a bookstore database that maintains a database of books. It exposes a method ProcessPaperbackBooks, which finds all paperback books in the database and calls a delegate for each one. The **delegate** type used is called ProcessBookDelegate. The Test class uses this class to print out the titles and average price of the paperback books.

Steps:

Create a new console application name Bookstore. Rename Program.cs to book.cs. Add the following code in book.cs

```
// book.cs
using System;

// A set of classes for handling a bookstore:
namespace Bookstore
{
   using System.Collections;

   // Describes a book in the book list:
   public struct Book
   {
      public string Title;       // Title of the book.
      public string Author;      // Author of the book.
      public decimal Price;      // Price of the book.
      public bool Paperback;     // Is it paperback?

      public Book(string title, string author, decimal price, bool paperBack)
      {
         Title = title;
         Author = author;
         Price = price;
         Paperback = paperBack;
      }
```

```
    }

    // Declare a delegate type for processing a book:
    public delegate void ProcessBookDelegate(Book book);

    // Maintains a book database.
    public class BookDB
    {
        // List of all books in the database:
        ArrayList list = new ArrayList();

        // Add a book to the database:
        public void AddBook(string title, string author, decimal price, bool paperBack)
        {
            list.Add(new Book(title, author, price, paperBack));
        }

        // Call a passed-in delegate on each paperback book to process it:
        public void ProcessPaperbackBooks(ProcessBookDelegate processBook)
        {
            foreach (Book b in list)
            {
                if (b.Paperback)
                    // Calling the delegate:
                    processBook(b);
            }
        }
    }
}
```

Create another class test.cs & add below code.

```
// Using the Bookstore classes:
namespace BookTestClient
{
    using Bookstore;
    using System;

    // Class to total and average prices of books:
    class PriceTotaller
    {
        int countBooks = 0;
        decimal priceBooks = 0.0m;
```

```
    internal void AddBookToTotal(Book book)
    {
       countBooks += 1;
       priceBooks += book.Price;
    }

    internal decimal AveragePrice()
    {
       return priceBooks / countBooks;
    }
 }

 // Class to test the book database:
class Test
{
  // Print the title of the book.
  static void PrintTitle(Book b)
  {
    Console.WriteLine("   {0}", b.Title);
  }

  // Execution starts here.
  static void Main()
  {
    BookDB bookDB = new BookDB();

    // Initialize the database with some books:
    AddBooks(bookDB);

    // Print all the titles of paperbacks:
    Console.WriteLine("Paperback Book Titles:");
    // Create a new delegate object associated with the static
    // method Test.PrintTitle:
    bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(PrintTitle));

    // Get the average price of a paperback by using
    // a PriceTotaller object:
    PriceTotaller totaller = new PriceTotaller();
    // Create a new delegate object associated with the nonstatic
    // method AddBookToTotal on the object totaller:
    bookDB.ProcessPaperbackBooks(new ProcessBookDelegate(totaller.AddBookToTotal));
    Console.WriteLine("Average Paperback Book Price: ${0:#.##}",
      totaller.AveragePrice());
```

```
      Console.Read();
    }

    // Initialize the book database with some test books:
    static void AddBooks(BookDB bookDB)
    {
      bookDB.AddBook("The C Programming Language",
        "Brian W. Kernighan and Dennis M. Ritchie", 19.95m, true);
      bookDB.AddBook("The Unicode Standard 2.0",
        "The Unicode Consortium", 39.95m, true);
      bookDB.AddBook("The MS-DOS Encyclopedia",
        "Ray Duncan", 129.95m, false);
      bookDB.AddBook("Dogbert's Clues for the Clueless",
        "Scott Adams", 12.00m, true);
    }
  }
}
```

Output:

```
Paperback Book Titles:
  The C Programming Language
  The Unicode Standard 2.0
  Dogbert's Clues for the Clueless
Average Paperback Book Price: $23.97
```

Lab3: The below program demonstrates Generic delegate Example.

Steps: Create a new console application name Generic_Delegate. Add the following code in Program.cs.

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

namespace Generic_Delegate
{
  class Program
  {
    public delegate void MyGenericDelegate<T>(T arg);
```

```
    static void Main(string[] args)
    {
      Console.WriteLine
      ("***** Generic Delegates *****\n");

     MyGenericDelegate<string> strTarget =
     new MyGenericDelegate<string>(StringTarget);
     strTarget("Some string data");

     MyGenericDelegate<int> intTarget = IntTarget;
     intTarget(9);
     Console.ReadLine();
   }

   static void StringTarget(string arg)
   {
     Console.WriteLine("arg in uppercase is: {0}", arg.ToUpper());
   }

   static void IntTarget(int arg)
   {
     Console.WriteLine("++arg is: {0}", ++arg);
   }

    }
}
```

Output:

```
***** Generic Delegates *****

arg in uppercase is: SOME STRING DATA

++arg is: 10
```

Lab1:Threads and Joins The below program creates two instances of the Thread type and uses the ThreadStart class to specify their target methods A and B. The threads are started and then they are joined.

Steps: Create a new console application name ThreadingSample. Add the below code in Program.cs.

```
//Program.cs
//Program that uses threading constructs
using System;
using System.Threading;

namespace ThreadingSample
{
  class Program
   {
      static void Main()
      {
         Thread thread1 = new Thread(new ThreadStart(A));
         Thread thread2 = new Thread(new ThreadStart(B));
         thread1.Start();
         thread2.Start();
         thread1.Join();
         thread2.Join();
         Console.Read();
      }

      static void A()
      {
         Thread.Sleep(100);
         Console.WriteLine('A');
      }

      static void B()
      {
         Thread.Sleep(1000);
         Console.WriteLine('B');
      }
   }
}
```

Output:(The threads terminate after 0.1 and 1.0 seconds.)

A
B

Lab2: Creating, starting, and interacting between threads.

Steps: Create a new console application name MultithreadingSample. Rename program.cs to StopJoin.cs & add the below code.

```
// StopJoin.cs
using System;
using System.Threading;

public class Alpha
{

   // This method that will be called when the thread is started
   public void Beta()
   {
      while (true)
      {
         Console.WriteLine("Alpha.Beta is running in its own thread.");
      }
   }
};

public class StopJoin
{
   public static int Main()
   {
      Console.WriteLine("Thread Start/Stop/Join Sample");

      Alpha oAlpha = new Alpha();

      // Create the thread object, passing in the Alpha.Beta method
      // via a ThreadStart delegate. This does not start the thread.
      Thread oThread = new Thread(new ThreadStart(oAlpha.Beta));

      // Start the thread
      oThread.Start();

      // Spin for a while waiting for the started thread to become
```

```
        // alive:
        while (!oThread.IsAlive) ;

        // Put the Main thread to sleep for 1 millisecond to allow oThread
        // to do some work:
        Thread.Sleep(1);

        // Request that oThread be stopped
        oThread.Abort();

        // Wait until oThread finishes. Join also has overloads
        // that take a millisecond interval or a TimeSpan object.
        oThread.Join();

        Console.WriteLine();
        Console.WriteLine("Alpha.Beta has finished");

        try
        {
           Console.WriteLine("Try to restart the Alpha.Beta thread");
           oThread.Start();
        }
        catch (ThreadStateException)
        {
           Console.Write("ThreadStateException trying to restart Alpha.Beta. ");
           Console.WriteLine("Expected since aborted threads cannot be restarted.");
        }
        Console.Read();
        return 0;
     }
}
```

```
Thread Start/Stop/Join Sample

Alpha.Beta has finished
Try to restart the Alpha.Beta thread
ThreadStateException trying to restart Alpha.Beta. Expected since aborted threads cannot be
restarted.
```

Lab3: Monitor object and Lock keyword.

The following example shows how synchronization can be accomplished using the C# lock keyword and the Pulse method of the Monitor object. The Pulse method Notifies a thread in the waiting queue of a change in the object's state

Steps: Create new console application with name MonitorSample. Rename Program.cs to MonitorSample.cs & add the following code.

```
// MonitorSample.cs
// This example shows use of the following methods of the C# lock keyword
// and the Monitor class
// in threads:
//      Monitor.Pulse(Object)
//      Monitor.Wait(Object)
using System;
using System.Threading;

public class MonitorSample
{
   public static void Main(String[] args)
   {
      int result = 0;   // Result initialized to say there is no error
      Cell cell = new Cell();

      CellProd prod = new CellProd(cell, 5);  // Use cell for storage,
      // produce 5 items
      CellCons cons = new CellCons(cell, 5);  // Use cell for storage,
      // consume 5 items

      Thread producer = new Thread(new ThreadStart(prod.ThreadRun));
      Thread consumer = new Thread(new ThreadStart(cons.ThreadRun));
      // Threads producer and consumer have been created,
      // but not started at this point.

      try
      {
         producer.Start();
         consumer.Start();

         producer.Join();   // Join both threads with no timeout
         // Run both until done.
         consumer.Join();
         // threads producer and consumer have finished at this point.
      }
      catch (ThreadStateException e)

      {
         Console.WriteLine(e);  // Display text of exception
```

```
            result = 1;          // Result says there was an error
        }
        catch (ThreadInterruptedException e)
        {
            Console.WriteLine(e);  // This exception means that the thread
            // was interrupted during a Wait
            result = 1;          // Result says there was an error
        }
        // Even though Main returns void, this provides a return code to
        // the parent process.
        Environment.ExitCode = result;
    }
}

public class CellProd
{
    Cell cell;        // Field to hold cell object to be used
    int quantity = 1;  // Field for how many items to produce in cell

    public CellProd(Cell box, int request)
    {
        cell = box;         // Pass in what cell object to be used
        quantity = request;  // Pass in how many items to produce in cell
    }
    public void ThreadRun()
    {
        for (int looper = 1; looper <= quantity; looper++)
            cell.WriteToCell(looper);  // "producing"
    }}
public class CellCons
{
    Cell cell;        // Field to hold cell object to be used
    int quantity = 1;  // Field for how many items to consume from cell

    public CellCons(Cell box, int request)
    {
        cell = box;         // Pass in what cell object to be used
        quantity = request;  // Pass in how many items to consume from cell
    }
    public void ThreadRun()
    {
        int valReturned;
        for (int looper = 1; looper <= quantity; looper++)
            // Consume the result by placing it in valReturned.
```

```
            valReturned = cell.ReadFromCell();
        }
    }

    public class Cell
    {
        int cellContents;       // Cell contents
        bool readerFlag = false;  // State flag
        public int ReadFromCell()
        {
            lock (this)   // Enter synchronization block
            {
                if (!readerFlag)
                {          // Wait until Cell.WriteToCell is done producing
                    try
                    {
                        // Waits for the Monitor.Pulse in WriteToCell
                        Monitor.Wait(this);
                    }
                    catch (SynchronizationLockException e)
                    {
                        Console.WriteLine(e);
                    }
                    catch (ThreadInterruptedException e)
                    {
                        Console.WriteLine(e);
                    }
                }
                Console.WriteLine("Consume: {0}", cellContents);
                Console.ReadLine();
                readerFlag = false;    // Reset the state flag to say consuming
                // is done.
                Monitor.Pulse(this);   // Pulse tells Cell.WriteToCell that
                // Cell.ReadFromCell is done.
            }  // Exit synchronization block
            return cellContents;
        }

        public void WriteToCell(int n)
        {
            lock (this)  // Enter synchronization block
            {
                if (readerFlag)
                {     // Wait until Cell.ReadFromCell is done consuming.
```

```
            try
            {
                Monitor.Wait(this);   // Wait for the Monitor.Pulse in
                // ReadFromCell
            }
            catch (SynchronizationLockException e)
            {
                Console.WriteLine(e);
            }
            catch (ThreadInterruptedException e)
            {
                Console.WriteLine(e);
            }
        }
        cellContents = n;
        Console.WriteLine("Produce: {0}", cellContents);
        readerFlag = true;    // Reset the state flag to say producing
        // is done
        Monitor.Pulse(this);  // Pulse tells Cell.ReadFromCell that
        // Cell.WriteToCell is done.
    }  // Exit synchronization block
}}
```

Output:

```
Produce: 1
Consume: 1
Produce: 2
Consume: 2
Produce: 3
Consume: 3
Produce: 4
Consume: 4
Produce: 5
Consume: 5
```

Lab4:Using Locks

Steps: Create a new console application name Lock_Threading . Add the below code in Program.cs.

```
//Lock in Threading
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Lock_Threading
{
   class Program
   {
      static readonly object _object = new object();
      static void A()
      {
         // Lock on the readonly object.
         // ... Inside the lock, sleep for 100 milliseconds.
         // ... This is thread serialization.
         lock (_object)
         {
            Thread.Sleep(100);
            Console.WriteLine("hello...");
         }
      }
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            ThreadStart start = new ThreadStart(A);
            new Thread(start).Start();
         }

          Console.ReadLine();
    }

      }
   }
```

Output: Run the program & check how hello is getting displayed on console.

Session 14: Reflection and Serialization in C#

Lab1:Using Type and FieldInfo classes.

This program uses the System.Reflection namespace to access the metadata of itself. It searches for the public field of the name "_number" and then acquires its value. You can see how a string literal is used to find a variable with that same name.

Steps: Create new console application with name ReflectionSample. Add the below code in Program.cs

```
//Program.cs
//Program that uses reflection
using System;
using System.Reflection;

namespace ReflectionSample
{
   class Program
   {
      public static int _number = 7;
      static void Main()
      {
         Type type = typeof(Program);
         FieldInfo field = type.GetField("_number");
         object temp = field.GetValue(null);
         Console.WriteLine(temp);
             Console.Read();

      }

   }
}
```

Output:

```
7
```

Lab2: Using Invoke Method.

UsingThe GetMethods method on the Type class returns all the public instance methods on a type by default. Here, we get an array of MethodInfo instances of all the public Program class methods. We print their names, but also test for the Win method. If we find this method, we call Invoke on it.

Steps: Create a new console application with name ReflectionInvokeMethod. Add the below code in Program.cs.

```
//Program.cs
//Program that calls Invoke on MethodInfo

using System;
using System.Reflection;

namespace ReflectioInvokeMethod
{
  class Program
  {
    public void Win()
    {
      Console.WriteLine("{You're a winner}");
    }

    public void Lose()
    {
    }

    public void Draw()
    {
    }

    static void Main()
    {
      // Instance used for Invoke. [1]
      Program program = new Program();

      // Get methods.
      MethodInfo[] methods = typeof(Program).GetMethods();
      foreach (MethodInfo info in methods)
      {
        Console.WriteLine(info.Name);

        // Call Win method.
        if (info.Name == "Win")
        {
          info.Invoke(program, null); // [2]
        }
      }
      Console.Read();
    }
  }
```

```
}
```

Notice. Please note how the Program class is instantiated inside the Main method [1]. This instance is then passed as the first argument to the Invoke method [2]. This is not the most natural way to use an object-oriented system, but it can get the job done if necessary.

Output:

```
Win
{You're a winner}
Lose
Draw
ToString
Equals
GetHashCode
GetType
```

Lab3: Looping over Properties

This program uses the System.Reflection namespace. The Program class has two instance properties: the Awesome property and the Perls property. One is of type int and the second is of type string. In the Main entry point, we create an instance of Program and assign the properties.

Steps: Create a new console application with name ReflectionInvokeMethod. Add the below code in Program.cs.

```
//Program.cs
//Program that uses reflection on properties
using System;
using System.Reflection;

namespace ReflectionPropertiesSample
{
  class Program
   {
      public int Awesome { get; set; }
      public string Perls { get; set; }
```

```
    static void Main()
    {
      // Create an instance of Program.
      Program programInstance = new Program();
      programInstance.Awesome = 7;
      programInstance.Perls = "Hello";

      // Get type.
      Type type = typeof(Program);

      // Loop over properties.
      foreach (PropertyInfo propertyInfo in type.GetProperties())
      {
        // Get name.
        string name = propertyInfo.Name;

        // Get value on the target instance.
        object value = propertyInfo.GetValue(programInstance, null);

        // Test value type.
        if (value is int)
        {
          Console.WriteLine("Int: {0} = {1}", name, value);
        }
        else if (value is string)
        {
          Console.WriteLine("String: {0} = {1}", name, value);
        }
      }
      Console.Read();
    }
  }
}
```

Output:

```
Int: Awesome = 7
String: Perls = Hello
```

Lab4: Using GetType() method.

This program uses the GetType method on a base type reference to more derived objects. Also, it shows how transitive closure can be applied on the class derivation system to understand how base classes relate to derived classes. The program shows that the GetType method prints the most derived object type of the instance.

Steps: Create a new console application with name ReflectionGetTypeMethod. Add the below code in Program.cs.

```
//Program.cs
// Program that uses GetType on base type instance
using System;

namespace ReflectionGetTypeMethod
{
   class A
   {
   }

   class B : A
   {
   }

   class C : B
   {
   }

   class Program
   {
      static void Main()
      {
         A a1 = new A();
         A a2 = new B();
         A a3 = new C();

         Console.WriteLine(a1.GetType());
         Console.WriteLine(a2.GetType());
         Console.WriteLine(a3.GetType());
         Console.Read();
      }   }}
```

Output:

```
ReflectionGetTypeMethod.A
ReflectionGetTypeMethod.B
ReflectionGetTypeMethod.C
```

Lab5:Exploring int Type.

Steps: Create a new console application with name ReflectionSample3. Add the below code in Program.cs.

```
//Program.cs
using System;
using System.Reflection;

namespace ReflectionSample3
{
   class Program
   {
      static void Main(string[] args)
      {
          int n = 45;

         System.Type t = n.GetType();
         Console.WriteLine(t.ToString());//Display type of varible
         Console.WriteLine(t.Name);//Display name of variable
         Console.WriteLine(t.Namespace);//Display namespace by name
         Console.WriteLine(t.IsArray.ToString());//Display boolean value
         Console.WriteLine(t.IsClass.ToString());//Display boolean value
         Console.WriteLine(t.Module.ToString());//Display module(library) of assembly
         Console.WriteLine(t.MemberType.ToString());//Display membertype by name
         Console.WriteLine(t.GetProperties().ToString());//Display property by name
         Console.WriteLine(t.GetType().ToString());//Display type defined in assembly
         Console.WriteLine(t.GetMethods().ToString());//Display method by name
         Console.WriteLine(t.GetInterfaces().ToString());//Display interface by name
         Console.WriteLine(t.FullName.ToString());//Display type of variable by name
         Console.WriteLine(t.BaseType.ToString());//Display base type
         Console.WriteLine(t.Attributes.ToString());//Display attribute by name
         Console.WriteLine(t.Assembly.ToString());//Display assembly by name
         Console.WriteLine(t.AssemblyQualifiedName.ToString());//Display AssemblyQualifiedName
         System.Reflection.Assembly o = System.Reflection.Assembly.Load("mscorlib.dll");
```

```
        Console.WriteLine(o.GetName().ToString());//Display information about  mscorlib assembly
        object obj = o.GetType();
        Console.WriteLine(obj.ToString());//Display name of runtime assembly
        ConstructorInfo[] ci = t.GetConstructors();//Display information about constructors
        foreach (ConstructorInfo i in ci)
        {
            Console.WriteLine(i.ToString());
        }
        MethodInfo[] mi = t.GetMethods();//Display information about methods
        foreach (MethodInfo i in mi)
        {
            Console.WriteLine(i.ToString());
        }
        PropertyInfo[] pi = t.GetProperties();//Display information about properties
        foreach (PropertyInfo i in pi)
        {
            Console.WriteLine(i.ToString());
        }
        MemberInfo[] mf = t.GetMembers();//Display information about members
        foreach (MemberInfo i in mf)
        {
            Console.WriteLine(i.ToString());
        }
        EventInfo[] ei = t.GetEvents();//Display information about events
        foreach (EventInfo i in ei)
        {
            Console.WriteLine(i.ToString());
        }
        FieldInfo[] fi = t.GetFields();//Display information about fields
        foreach (FieldInfo i in fi)
        {
            Console.WriteLine(i.ToString());
        }
        Console.Read();
    }

    }
}
```

Output: Run the program & check console for output.

Session 16: Remoting in C# 3.5

Lab1: TicketServer holds information about the ticket status of a movie theater. A client needs to know the status of a ticket. Establish a connection between the client and he server so that client gets the information needed.

Steps: A method called GetTicketStatus is defined in the server space. This method returns the status of the ticket. The server publishes this method which can be used by any client. The server listens to port 9998 over TCP. The client invokes the published method and gets the ticket status.

Server part:

1. Create a Console Application named TicketServer.
2. Add System.Runtime.Remoting as a reference to the project.
3. Replace the existing code in Program.cs with the following code and build the project.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

class Program
{
   static void Main(string[] args)
   {
      TicketServer();
   }

   static void TicketServer()
   {
      Console.WriteLine("Ticket Server started...");

      TcpChannel tcpChannel = new TcpChannel(9998);
```

```
        ChannelServices.RegisterChannel(tcpChannel);

        Type commonInterfaceType = Type.GetType("MovieTicket");

        RemotingConfiguration.RegisterWellKnownServiceType(commonInterfaceType,
        "MovieTicketBooking", WellKnownObjectMode.SingleCall);

        System.Console.WriteLine("Press ENTER to quitnn");
        System.Console.ReadLine();

    }

}

public interface MovieTicketInterface
{
    string GetTicketStatus(string stringToPrint);
}

public class MovieTicket : MarshalByRefObject, MovieTicketInterface
{
    public string GetTicketStatus(string stringToPrint)
    {
        string returnStatus = "Ticket Confirmed";
        Console.WriteLine("Enquiry for {0}", stringToPrint);
        Console.WriteLine("Sending back status: {0}", returnStatus);

        return returnStatus;
    }
}
```

Client side:

1. Create a Console Application named Client.
2. Add System.Runtime.Remoting and TicketServer.exe as references to the project.
3. Replace the existing code in Program.cs with the following code and build the project.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

class MyClient
```

```
{
   public static void Main()
   {
      TcpChannel tcpChannel = new TcpChannel();
      ChannelServices.RegisterChannel(tcpChannel);

      Type requiredType = typeof(MovieTicketInterface);

      MovieTicketInterface remoteObject =
(MovieTicketInterface)Activator.GetObject(requiredType,
      "tcp://localhost:9998/MovieTicketBooking");

      Console.WriteLine(remoteObject.GetTicketStatus("Ticket No: 3344"));
   }
}
```

Execution:

1. Execute TicketServer.exe (folder path \bin)
2. Execute Client.exe (folder path \bin)

You will see the appropriate messages on the client side and the remote side.

Output:

Ticket Server started...

Press ENTER to quitnn

Enquiry for Ticket No: 3344

Sending back status: Ticket Confirmed

Session 17: Garbage Collection in C#

Lab1: This example demonstrates the effect of invoking the GC.Collect method. Three calls to get the total memory usage on the system are present: they occur before the allocation, after the allocation, and after the forced garbage collection. You can see that memory returns to its low level after the garbage collection.

Steps: Create a new console application with name GarbageCollectionSample. Add the below code in Program.cs.

```
//Program that uses GC.Collect
using System;

namespace GarbageCollectionSample
{
   class Program
   {
      static void Main()
      {
         long mem1 = GC.GetTotalMemory(false);
         {
            // Allocate an array and make it unreachable.
            int[] values = new int[50000];
            values = null;
         }
         long mem2 = GC.GetTotalMemory(false);
         {
            // Collect garbage.
            GC.Collect();
         }
         long mem3 = GC.GetTotalMemory(false);
         {
            Console.WriteLine(mem1);
            Console.WriteLine(mem2);
            Console.WriteLine(mem3);
            Console.Read();
         }
      }
   }
}
```

Output: (May vary based on your system.)

```
45664
245696
33244
```

Lab2: The WeakReference type is created using a constructor call. You must pass the object reference you want to point to to the constructor; here we use a StringBuilder object. In the middle of the program, the garbage collector is run using GC.Collect. After this call, the object pointed to by the WeakReference no longer exists. If you don't call GC.Collect, the object will almost certainly still exist.

Steps: Create a new console application with name WeakReferenceSample. Add the below code in Program.cs.

```
// Program that uses WeakReference
using System;
using System.Text;

namespace WeakReferenceSample
{
   class Program
   {
      /// <summary>
      /// Points to data that can be garbage collected any time.
      /// </summary>
      static WeakReference _weak;

      static void Main()
      {
         // Assign the WeakReference.
         _weak = new WeakReference(new StringBuilder("Ruby"));

         // See if weak reference is alive.
         if (_weak.IsAlive)
         {
            Console.WriteLine((_weak.Target as StringBuilder).ToString());
         }

         // Invoke GC.Collect.
         // ... If this is commented out, the next condition will evaluate true.
         GC.Collect();

         // Check alive.
```

```
        if (_weak.IsAlive)
        {
            Console.WriteLine("IsAlive");
        }

        // Finish.
        Console.WriteLine("[Done]");
        Console.Read();
    }
  }
}
```

Output:

```
Ruby
[Done]
```

Lab3: Working with Using Clause.

This program defines a class called SystemResource. The class implements the IDisposable interface and the required Dispose method. In the Main method, we wrap the SystemResource instance inside a using statement.

Steps: Create a new console application with name usingStatementSample . Add the below code in Program.cs.

```
// Program that demonstrates using statement which automatically disposes objects when it goes beyond its scope.
using System;
using System.Text;
namespace usingStatementSample
{

class Program
{
  static void Main()
  {
    // Use using statement with class that implements Dispose.
    using (SystemResource resource = new SystemResource())
    {
      Console.WriteLine(1);
    }
```

```
        Console.WriteLine(2);
        Console.Read();
    }
}

class SystemResource : IDisposable
{
    public void Dispose()
    {
        // The implementation of this method not described here.
        // ... For now, just report the call.
        Console.WriteLine(0);
    }
}
}
```

Output:

```
1
0
2
```

LINQ TO Objects

Highlighted in yellow is applicable all the labs given below.

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

static class Samples {
    static int[] numbers = new [] { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    static string[] strings = new [] { "zero", "one", "two", "three", "four", "five", "six",
"seven", "eight", "nine" };

    class Person {
      public string Name {get; set;}
      public int Level {get; set;}
    }

    static Person[] persons = new Person[] {
        new Person {Name="Jesper", Level=3},
        new Person {Name="Lene", Level=3},
```

```
        new Person {Name="Jonathan", Level=5},
        new Person {Name="Sagiv", Level=3},
        new Person {Name="Jacqueline", Level=3},
        new Person {Name="Ellen", Level=3},
        new Person {Name="Gustavo", Level=9}
        };
```

LAB 1 : Using Where Extension Method

```
public static void Sample1() {
    // use Where() to filter out elements matching a particular condition
    var fnums = numbers.Where(n => n < 5);

    Console.WriteLine("Numbers < 5");
    foreach(int x in fnums) {
        Console.WriteLine(x);
    }
}
```

LAB 1 A : Using Where in Query Expression

```
public static void Sample1A()
{
    // use Where() to filter out elements matching a particular condition
    var fnums = from n in numbers
                where n < 5
                select n;

    Console.WriteLine("Numbers < 5");
    foreach (int x in fnums)
    {
        Console.WriteLine(x);
    }
}
```

LAB 2 : Using First() with Extension Method syntax

```
public static void Sample2() {
    // use First() to find the one element matching a particular condition
    string v = strings.First(s => s[0] == 'o');

    Console.WriteLine("string starting with 'o': {0}", v);
}
```

LAB 2A : Using Where condition with query expression

```
public static void Sample2A()
{
           var v = from s in strings
               where s[0] == 'o'
               select s;

    foreach ( string  str in v)
```

```
                Console.WriteLine("string starting with 'o': {0}", str);
    }
```

LAB 3 : use Select() to convert each element into a new value – Extension Method syntax

```
    public static void Sample3() {
        // use Select() to convert each element into a new value
        var snums = numbers.Select(n => strings[n]);

        Console.WriteLine("Numbers");
        foreach(string s in snums) {
            Console.WriteLine(s);
        }
    }
```

LAB 3 A : use Select() to convert each element into a new value – Expression syntax

```
    public static void Sample3A()
    {
        // use Select() to convert each element into a new value
        var snums = from n in numbers
                    select strings[n];

        Console.WriteLine("Numbers");
        foreach (string s in snums)
        {
            Console.WriteLine(s);
        }
    }
```

LAB 4 : use Ananymous Types with LINQ – Extension Method

```
    public static void Sample4()
    {
        // use Anonymous Type constructors to construct multi-valued results on the fly
        var q = strings.Select(s => new {Head = s.Substring(0,1), Tail = s.Substring(1)});
        foreach(var p in q) {
            Console.WriteLine("Head = {0}, Tail = {1}", p.Head, p.Tail);
        }
    }
```

LAB 4 A : use Ananymous Types with LINQ – Expression syntax

```
    public static void Sample4A()
    {
        // use Anonymous Type constructors to construct multi-valued results on the fly
        var q = from s in strings
                select new { Head = s.Substring(0, 1), Tail = s.Substring(1) };
```

```
        foreach (var p in q)
        {
            Console.WriteLine("Head = {0}, Tail = {1}", p.Head, p.Tail);
        }
    }
```

LAB 5 : Select() and Where() to make a complete query- Extension Method

```
    public static void Sample5() {
        // Combine Select() and Where() to make a complete query
        var q = numbers.Where(n => n < 5).Select(n => strings[n]);

        Console.WriteLine("Numbers < 5");
        foreach(var x in q) {
            Console.WriteLine(x);
        }
    }
```

LAB 5A : Select() and Where() to make a complete query- Expressive syntax

```
    public static void Sample5A()
    {
        // Combine Select() and Where() to make a complete query
        var q = from n in numbers
                where n > 5
                select  strings[n] ;

        Console.WriteLine("Numbers < 5");
        foreach (var x in q)
        {
            Console.WriteLine(x);
        }
    }
```

LAB 6 : Sequence Operator- Extension Method

```
    public static void Sample6() {
        // Sequence operators form first-class queries are not executed until you enumerate
them.
        int i = 0;
        var q = numbers.Select(n => ++i);
        // Note, the local variable 'i' is not incremented until each element is evaluated
(as a side-effect).
        foreach(var v in q) {
          Console.WriteLine("v = {0}, i = {1}", v, i);
        }
        Console.WriteLine();

        // Methods like ToList() cause the query to be executed immediately, caching the
results
        int i2 = 0;
        var q2 = numbers.Select(n => ++i2).ToList();
        // The local variable i2 has already been fully incremented before we iterate the
results
```

```
        foreach(var v in q2) {
          Console.WriteLine("v = {0}, i2 = {1}", v, i2);
        }
    }
```

LAB 6A : Query Execution- Expressive Syntax

```
    public static void Sample6A()
    {
        // Sequence operators form first-class queries are not executed until you enumerate
them.
        int i = 0;
        var q =  from n in numbers
                 select ++i;
        // Note, the local variable 'i' is not incremented until each element is evaluated
(as a side-effect).
        foreach (var v in q)
        {
            Console.WriteLine("v = {0}, i = {1}", v, i);
        }
        Console.WriteLine();

        // Methods like ToList() cause the query to be executed immediately, caching the
results
        int i2 = 0;
        var q2 =   from n in numbers
                 select ++i;
        List<int> q3 = q2.ToList();

        // The local variable i2 has already been fully incremented before we iterate the
results
        foreach (var v in q3)
        {
            Console.WriteLine("v = {0}, i2 = {1}", v, i2);
        }
    }
```

LAB 7 : Group by - Extension Method

```
    public static void Sample7() {
        // use GroupBy() to construct group partitions out of similar elements
        var q = strings.GroupBy(s => s[0]); // <- group by first character of each string

        foreach(var g in q) {
          Console.WriteLine("Group: {0}", g.Key);
          foreach(string v in g) {
            Console.WriteLine("\tValue: {0}", v);
          }
        }
    }
```

LAB 7 : Group by – Expression Syntax

```
    public static void Sample7A()
    {
        // use GroupBy() to construct group partitions out of similar elements
        //var q = strings.GroupBy(s => s[0]); // <- group by first character of each string
        var q = from s in strings
                group s by s[0];
        foreach (var g in q)
        {
            Console.WriteLine("Group: {0}", g.Key);
            foreach (string v in g)
            {
                Console.WriteLine("\tValue: {0}", v);
            }
        }
    }
```

LAB 8 : Group by with Count, Min, Max, Sum - Extension Method

```
    public static void Sample8() {
        // use GroupBy() and aggregates such as Count(), Min(), Max(), Sum(), Average() to
compute values over a partition
        var q = strings.GroupBy(s => s[0])
                       .Select(g => new {FirstChar = g.Key, Count = g.Count()});

        foreach(var v in q) {
          Console.WriteLine("There are {0} string(s) starting with the letter {1}", v.Count,
v.FirstChar);
        }
    }
```

LAB 8A : Group by with Count, Min, Max, Sum – Expression Syntax

```
    public static void Sample8A()
    {
        // use GroupBy() and aggregates such as Count(), Min(), Max(), Sum(), Average() to
compute values over a partition


        var q = from s in strings
                group s by s[0] into g
                select new {FirstChar=g.Key, Count=g.Count()};

        foreach (var v in q)
        {
            Console.WriteLine("There are {0} string(s) starting with the letter {1}",
v.Count, v.FirstChar);
        }
    }
```

LAB 9 : OrderByDescending – Extension Method

```
    public static void Sample9() {
        // use OrderBy()/OrderByDescending() to give order to your resulting sequence
        var q = strings.OrderBy(s => s);  // order the strings by their name

        foreach (string s in q)
        {
            Console.WriteLine(s);
        }

         var q1 = persons.OrderBy(p => p.Level).ThenBy(p => p.Name);

         foreach (var p in q1)
         {
             Console.WriteLine("{0}  {1}", p.Level, p.Name);

        }
    }
```

LAB 9A : OrderByDescending – Expression Syntax

```
    public static void Sample9A()
    {
        // if you use query expression syntax, you can use the order by like in SQL Syntax.

        var q = from s in strings
                orderby s
                select s;

        foreach (string s in q)
        {
            Console.WriteLine(s);
        }

        var q1 = from p in persons
                 orderby p.Level, p.Name descending  // descending key word can be used if
required.
                 select p;

        foreach (var p1 in q1)
        {
            Console.WriteLine("{0}  {1}", p1.Level, p1.Name);
        }
    }
```

LAB 10 : Using GroupBy and OrderBy together

```
    public static void Sample10() {
        // use query expressions to simplify
        var q = from p in persons
                orderby p.Level, p.Name
                group p.Name by p.Level into g
                select new {Level = g.Key, Persons = g};

        foreach(var g in q) {
            Console.WriteLine("Level: {0}", g.Level);
            foreach(var p in g.Persons) {
                Console.WriteLine("Person: {0}", p);
            }
        }
    }
}
```