Lập trình C#

Mục tiêu:
Sau khi thực hành xong bạn nắm được các khái niệm sau:

Abstract classes and Interface

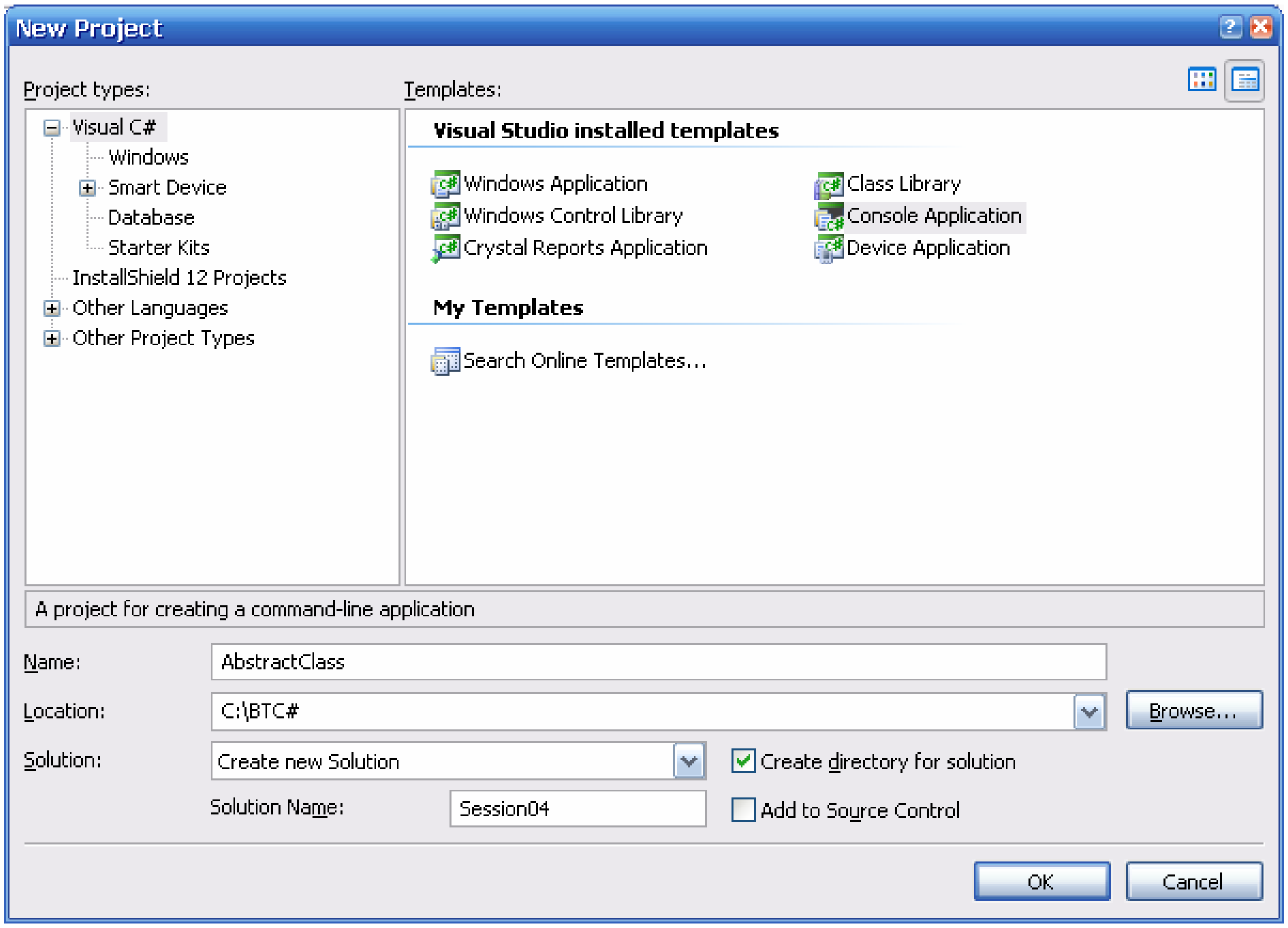
Properties and Indexers

Phần I: Thực hành theo các bước – 45 minutes

Bài tập 1.1: abstract class

Step 1: Open Visual Studio

Step 2: Select the menu File->New->Project to create console based project named 'AbstractClass' and Solution named Session04 as following



A project for creating a command-line application

Name: AbstractClass

Location: C:\BTC#

Solution: Create new Solution

Solution Name: Session04

[✓] Create directory for solution

[ ] Add to Source Control

Step 3: Rename the class file ‘program.cs‘ to ‘AbstractClass.cs’

Step 4: Replace code in ‘AbstractClass.cs’ with given code

```
using System;
    // Declare an abstract
class abstract class clsBase
{
        // Declare an abstract method.
        abstract public void
        Describe();

        // Declare an abstract property that has only a get accessor.
        abstract public double DoubleProp
        {
            get;
        }
        // Declare an abstract property that has only a set accessor.
        abstract public int IntProp
        {
            set;
        }
        // Declare an abstract propety that has both get and set accessors.
        abstract public string StringProp
        {
            get;
            set;
        }
        // Declare a method that will access the abstract members.
        public void GetAbstract()
        {
            // Get the DoubleProp, which will be in the derived class.
            Console.WriteLine("DoubleProp = " + DoubleProp);
            // You can only set the IntProp value. The storage is in the
            // derived
            class. IntProp
            = 42;

            // Set the StringProp value
            StringProp = "StringProperty actually is stored in " +
                          "the derived class.";
            // Now show StringProp
            Console.WriteLine(StringProp);

            // Finally, call the abstract method
            Describe();
        }
}
    // Derive a class from clsBase. You must implement the abstract members
class clsDerived : clsBase
```

```
{
        // Declare a constructor to set the DoubleProp member
        public clsDerived(double val)
        {
            m_Double = val;
        }
    // When you implement an abstract member in a derived class, you may not
        // change the type or access
        level. override public void
        Describe()
        {
            Console.WriteLine("You called Describe() from the base " +
                                  "class but the code body is in the \r\n" +
                                  "derived class");
            Console.WriteLine("m_Int = " +
            m_Int);
        }

      // Implement the DoubleProp property. This is where you provide a body
        // for the accessors.
        override public double DoubleProp
        {
            get { return (m_Double); }
        }
        // Implement the set accessor for IntProp.
        override public int IntProp
        {
            set { m_Int = value; }
        }

        // Implement StringProp, providing a body for both the get
        // and set accessors.
        override public string StringProp
        {
            get { return (m_String); }
            set { m_String = value; }
        }
        // Declare fields to support the properties.
        private double m_Double;
        private int m_Int;
        private string m_String;
    }

class InterfaceDemo1
{
        static void Main(string[] args)
        {
            // Create an instance of the derived class.
            clsDerived derived = new clsDerived(3.14159);
            // Calling GetAbstract() actually calls the public method in the
```

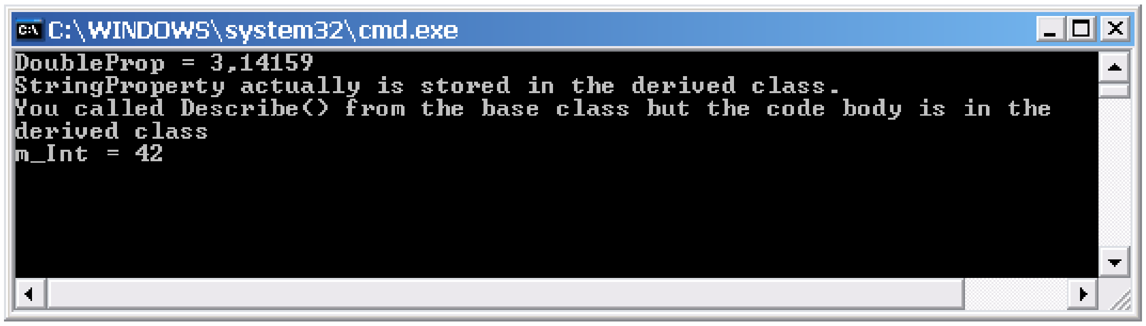
```
                // base class. There is no GetAbstract() in the derived class.
                derived.GetAbstract();
                Console.ReadLine();
            }
    }
```

Step 5: Select menu File -> Save to save the file

Step 6: Select Build -> Build AbstractClass option to build the project

Step 7: Select Debug -> Start without Debugging to execute the program

The output of the program as following



```
DoubleProp = 3,14159
StringProperty actually is stored in the derived class.
You called Describe() from the base class but the code body is in the
derived class
m_Int = 42
```

Bài tập 1.2: Properties

Step 1: Add a console based project '**PropertiesDemo**' to the solution

Step 2: Right click on project **PropertiesDemo** -> set as Startup project

Step 3: Rename the class file 'Program.cs' to '**PropertiesDemo**.cs'

Step 4: Replace the code in '**PropertiesDemo**.cs' with the given code

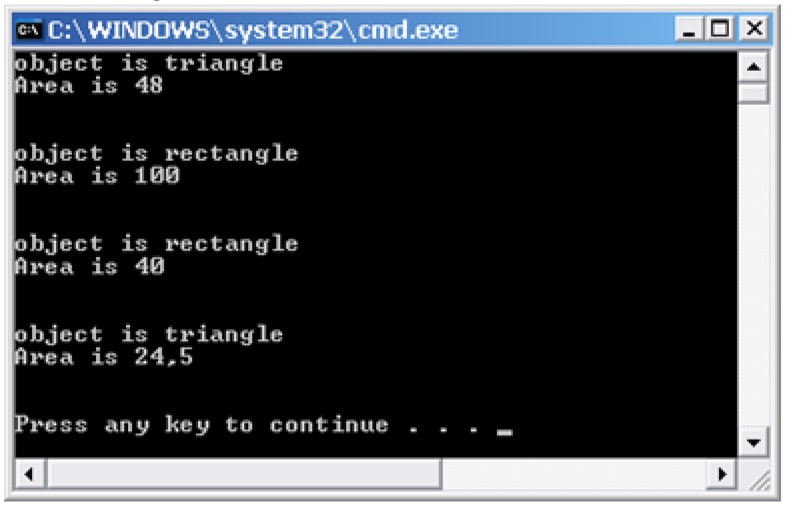
```
using System;
abstract class TwoDShape
{
        double pri_width;   // private
        double pri_height; // private
        string pri_name;   // private

        // A default constructor.
        public TwoDShape()
        {
            width = height = 0.0;
            name = "null";
        }

        // Parameterized constructor.
        public TwoDShape(double w, double h, string n)
        {
            width = w;
            height =
            h; name =
            n;
        }

        // Construct object with equal width and height.
        public TwoDShape(double x, string n)
        {
            width = height = x;
            name = n;
        }

        // Construct an object from an object.
        public TwoDShape(TwoDShape ob)
        {
            width = ob.width;
            height =
            ob.height; name =
            ob.name;
        }
        // Properties for width, height, and name
        public double width
        {
            get { return pri_width; }
            set { pri_width = value; }
        }

        public double height
        {
            get { return pri_height; }
            set { pri_height = value; }
        }

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

        public void showDim()
        {
            Console.WriteLine("Width and height are " +
                           width + " and " + height);
        }

        // Now, area() is abstract.
        public abstract double area();
}

    // A derived class of TwoDShape for triangles.
class Triangle : TwoDShape
{
        string style; // private

        // A default constructor.
        public Triangle()
        {
            style = "null";
        }

        // Constructor for Triangle.
        public Triangle(string s, double w, double h) :
        base(w, h, "triangle")
        {
            style = s;
        }

        // Construct an isosceles triangle.
        public Triangle(double x) : base(x, "triangle")
        {
            style = "isosceles";
        }

        // Construct an object from an object.
        public Triangle(Triangle ob) : base(ob)
        {
            style = ob.style;
        }

        // Override area() for Triangle.
        public override double area()
        {
             return width * height / 2;
        }

        // Display a triangle's style.
        public void showStyle()
        {
            Console.WriteLine("Triangle is " + style);
        }
}

    // A derived class of TwoDShape for rectangles.
class Rectangle : TwoDShape
{
      // Constructor for Rectangle.
      public Rectangle(double w, double h) :
        base(w, h, "rectangle"){ }

      // Construct a square.
      public Rectangle(double x) :
        base(x, "rectangle") { }

      // Construct an object from an object.
      public Rectangle(Rectangle ob) : base(ob) { }

      // Return true if the rectangle is square.
      public bool isSquare()
      {
        if(width == height) return true;
        return false;
      }

      // Override area() for Rectangle.
      public override double area()
      {
        return width * height;
        }
}
class PropertiesDemo
    {
        static void Main(string[] args)
        {
            TwoDShape[] shapes = new TwoDShape[4];

            shapes[0] = new Triangle("right", 8.0, 12.0);
            shapes[1] = new Rectangle(10);
            shapes[2] = new Rectangle(10, 4);
            shapes[3] = new Triangle(7.0);

            for(int i=0; i < shapes.Length; i++) {
            Console.WriteLine("object is " + shapes[i].name);
            Console.WriteLine("Area is " + shapes[i].area());
            Console.WriteLine();
            Console.ReadLine();
        }
    }
}
```

Step 5: Select menu File -> Save to save the file

Step 6: Select Build -> Build '**PropertiesDemo**' option to build the project

Step 7: Select Debug -> Start without Debuging to execute the program

The output of program as following



```
object is triangle
Area is 48


object is rectangle
Area is 100


object is rectangle
Area is 40


object is triangle
Area is 24,5


Press any key to continue . . .
```

Bài tập 1.3: Creating and Using Interface

Step 1: Add a console based project ‘**InterfaceDemo**’ to the solution

Step 2: Right click on project **InterfaceDemo ->** set as Startup project

Step 3: Rename the class file ‘Program.cs’ to ‘**InterfaceDemo**.cs’

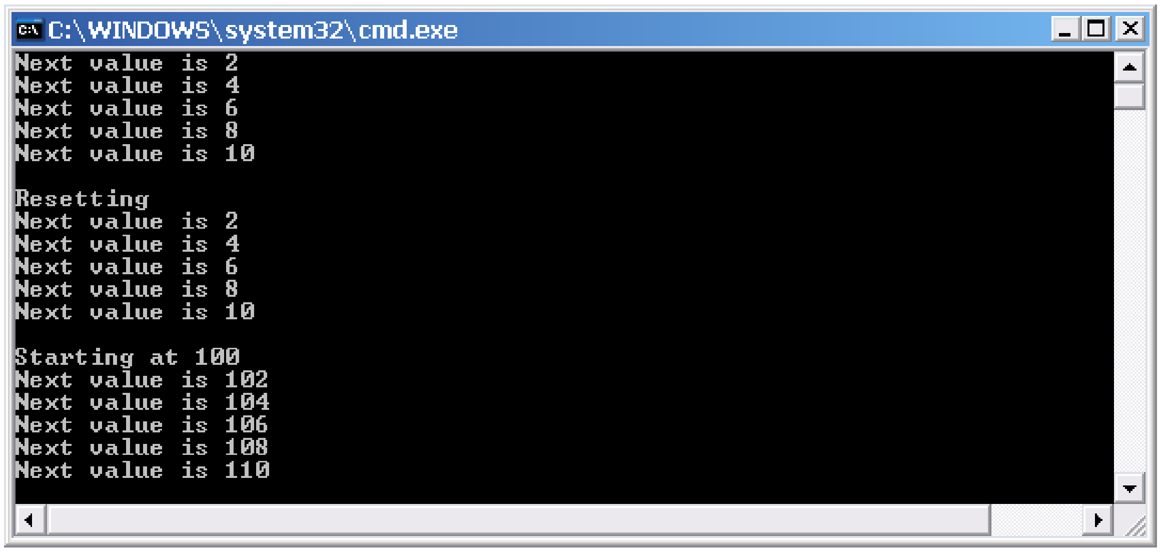
Step 4: Replace the code in ‘**InterfaceDemo**.cs’ with the given code

```
using System;
public interface ISeries
{
        int getNext(); // return next number in series
        void reset(); // restart
        void setStart(int x); // set starting value
}
// Implement
ISeries. class
ByTwos : ISeries
{
        int start;
        int val;
        public ByTwos()
        {
            start = 0;
            val = 0;
        }
        public int getNext()
        {
            val += 2;
            return val;
        }

        public void reset()
        {
            val = start;
        }
        public void setStart(int x)
        {
            start = x;
            val = start;
        }
}
class InterfaceDemo3
{
        static void Main(string[] args)
        {
            ByTwos ob = new ByTwos();

            for (int i = 0; i < 5; i++)
                Console.WriteLine("Next value is "
                +
                                    ob.getNext()
                                         );

            Console.WriteLine("\nResetting");
            ob.reset();
            for (int i = 0; i < 5; i++) Console.WriteLine("Next value is " +
                                                ob.getNext());

            Console.WriteLine("\nStarting at 100");
            ob.setStart(100);
            for (int i = 0; i < 5; i++) Console.WriteLine("Next value is " +
                                                     ob.getNext());
            Console.ReadLine();
        }
}
```

Step 5: Select menu File -> Save to save the file

Step 6: Select Build -> Build '**InterfaceDemo**' option to build the project

Step 7: Select Debug -> Start without Debuging to execute the program The output of program as following


C:\WINDOWS\system32\cmd.exe

```
Next value is 2
Next value is 4
Next value is 6
Next value is 8
Next value is 10

Resetting
Next value is 2
Next value is 4
Next value is 6
Next value is 8
Next value is 10

Starting at 100
Next value is 102
Next value is 104
Next value is 106
Next value is 108
Next value is 110
```

Bài tập 1.4: Deriving an interface from multiple interfaces

Step 1: Add a console based project '**MultiInterface**' to the solution

Step 2: Right click on project **MultiInterface** -> set as Startup project

Step 3: Rename the class file 'Program.cs' to '**MultiInterface**.cs'

Step 4: Replace the code in '**MultiInterface**.cs' with the given code

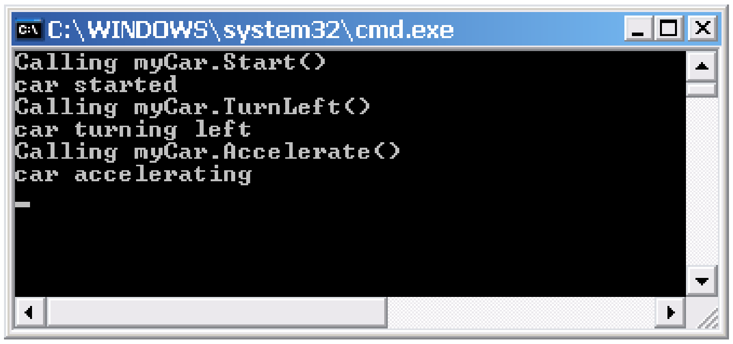
```
using System;
// define the IDrivable
interface public interface
IDrivable
{
        // method
        declarations void
        Start();
        void Stop();
        // property
        declaration bool
        Started
        {
            get;
        }
}
// define the ISteerable
interface public interface
ISteerable
{
        // method
        declarations void
        TurnLeft();
        void TurnRight();
}
// define the IMovable interface (derived from IDrivable and ISteerable)
public interface IMovable : IDrivable, ISteerable
    {
        // method
        declarations void
        Accelerate();
        void Brake();
    }

// Car class implements the IMovable interface
public class Car : IMovable
{
        // declare the underlying field used by the
        // Started property of the IDrivable interface
        private bool started = false;

        // implement the Start() method of the IDrivable interface
        public void Start()
        {
            Console.WriteLine("car started");
            started = true;
        }
        // implement the Stop() methodof the IDrivable interface
        public void Stop()
        {
            Console.WriteLine("car stopped");
            started = false;
        }
        // implement the Started property of the IDrivable interface
        public bool Started
        {
          get
          {
                 return started;     }
        }



        // implement the TurnLeft() method of the ISteerable interface
        public void TurnLeft()
        {
            Console.WriteLine("car turning left");
        }

        // implement the TurnRight() method of the ISteerable interface
        public void TurnRight()
        {
            Console.WriteLine("car turning right");
        }
        // implement the Accelerate() method of the IMovable interface
        public void Accelerate()
        {
            Console.WriteLine("car accelerating");
        }

        // implement the Brake() method of the IMovable interface
        public void Brake()
        {
            Console.WriteLine("car braking");
        }
}
class InterfaceDemo5
{
        static void Main(string[] args)
        {
            // create a Car object
            Car myCar = new Car();
            // call myCar.Start()
            Console.WriteLine("Calling myCar.Start()");
```

```
                myCar.Start();
                // call myCar.TurnLeft()
                Console.WriteLine("Calling myCar.TurnLeft()");
                myCar.TurnLeft();
                // call myCar.Accelerate()
                Console.WriteLine("Calling myCar.Accelerate()");
                myCar.Accelerate();
                Console.ReadLine();
            }
    }
```

Step 5: Select menu File -> Save to save the file

Step 6: Select Build -> Build ‘**MultiInterface**’ option to build the project

Step 7: Select Debug -> Start without Debuging to execute the program The output of program as following



```
Calling myCar.Start()
car started
Calling myCar.TurnLeft()
car turning left
Calling myCar.Accelerate()
car accelerating
_
```

Bài tập 1.5: Indexers

Step 1: Add a console based project ‘**IndexersDemo**’ to the solution

Step 2: Right click on project **IndexersDemo** -> set as Startup project

Step 3: Rename the class file ‘Program.cs’ to ‘**IndexersDemo**.cs’

Step 4: Replace the code in ‘**IndexersDemo**.cs’ with the given code

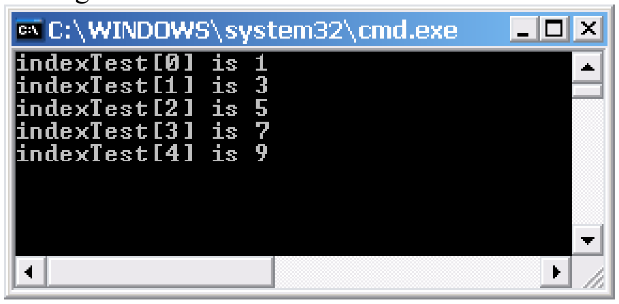
```
using System;
class IndexerExample
{
    public int[] intList = new int[10];
    public int this[int index]
    {
       get
            {
                return intList[index];

            }
             set
           {
               intList[index] = value;
           }
    }
}

class IndexerDemo
{
    static void Main()
    {
        int i, j = 0;
        IndexerExample indexTest = new IndexerExample();
        for (i = 1; i < 10; i += 2)
        {
            indexTest[j] = i;
            j++;
        }
        for (i = 0; i < 5; i++)
            Console.WriteLine("indexTest[{0}]  is {1}", i, indexTest[i]);
        Console.ReadLine();
    }
}
```

Step 5: Select menu File -> Save to save the file

Step 6: Select Build -> Build ‘**IndexersDemo**’ option to build the project

Step 7: Select Debug -> Start without Debuging to execute the program

The output of program as following



```
indexTest[0] is 1
indexTest[1] is 3
indexTest[2] is 5
indexTest[3] is 7
indexTest[4] is 9
```

Bài tập 1.6: Overloaded Indexers

Step 1: Add a console based project '**OverloadedIndexers**' to the solution

Step 2: Right click on project **OverloadedIndexers** -> set as Startup project

Step 3: Rename the class file 'Program.cs' to '**OverloadedIndexers**.cs'

Step 4: Replace the code in '**OverloadedIndexers**.cs' with the given code

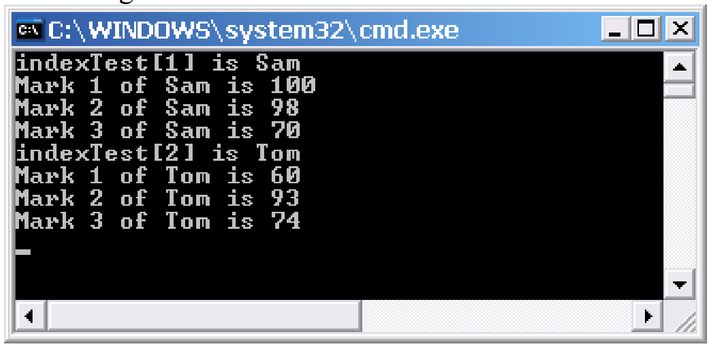
```
using System;
class IndexerExample
{
    public string[] StringList = new string[10];

    public string this[int index]
    {
           get{
                return StringList[index];
                }
          set{
                    StringList[index] = value;
           }
    }

    public int[,] intList = new int[10, 3];

    public int this[int index1, int index2]
    {
           get{ return intList[index1, index2];
                }
          set{intList[index1, index2] = value;
           }
    }


class OvrldIndexers
{
    static void Main()
    {
           IndexerExample indexTest = new IndexerExample();
           indexTest[0] =
           "Sam"; indexTest[0,
           0] = 100;
           indexTest[0, 1] =
           98; indexTest[0, 2]
           = 70; indexTest[1]
           = "Tom";
           indexTest[1, 0] =
           60; indexTest[1, 1]
           = 93; indexTest[1,
           2] = 74;
           Console.WriteLine("indexTest[1] is {0}", indexTest[0]);
           Console.WriteLine("Mark 1 of {0} is {1}", indexTest[0],

           indexTest[0,0]); Console.WriteLine("Mark 2 of {0} is {1}",

           indexTest[0], indexTest[0,1]);

           Console.WriteLine("Mark   3   of   {0}   is   {1}",   indexTest[0],
           indexTest[0,2]);

           Console.WriteLine("indexTest[2] is {0}", indexTest[1]);
           Console.WriteLine("Mark 1 of {0} is {1}", indexTest[1],
           indexTest[1,0]);

           Console.WriteLine("Mark 2 of {0} is {1}", indexTest[1],

           indexTest[1,1]);

           Console.WriteLine("Mark 3 of {0} is {1}", indexTest[1],

           indexTest[1,2]);

           Console.ReadLine();

     }

}
```

Step 5: Select menu File -> Save to save the file

Step 6: Select Build -> Build '**IndexersDemo**' option to build the project

Step 7: Select Debug -> Start without Debuging to execute the program

The output of program as following



```
indexTest[1] is Sam
Mark 1 of Sam is 100
Mark 2 of Sam is 98
Mark 3 of Sam is 70
indexTest[2] is Tom
Mark 1 of Tom is 60
Mark 2 of Tom is 93
Mark 3 of Tom is 74
```

Phần II: Tự thực hành – 60 phút

Bài tập 2.1: Circle Class

Cho đoạn mã nguồn sau

```
public abstract class GeometricObject
{
       protected string color;
       protected double weight;
        // Default construct
        protected GeometricObject()
        {
               color = "white";
               weight = 1.0;
        }
        // Construct a geometric object
        protected GeometricObject(string color, double weight)
        {
               this.color = color;
               this.weight = weight;
        }
        public string color
        {
              get
              {
                     return color;
              }
              set{
                     color = value;
               }
        }
        public double Weight
        {
              get
              {
                     return weight;

              }
              set
              {
                     weight = value;
               }
        }
        // Abstract method
        public abstract double findArea();
        // Abstract method
        public abstract double findPerimeter();
}
```

1. Viết một lớp tên là **`Circle`** kế thừa từ lớp **`GeometricObject`**. Circle có một trường tên là **`radius`**.

   Viết hai hàm khởi tạo

   - Một là với tham số đơn (for radius)
   - và một hàm khác với 3 tham số (với radius, color, weight)

   Override the **`ToString`**() để in đối tượng

2. Viết một chương trình Test để kiểm tra tất cả các hành vi của `Circle class`

3. Viết một chương trình test đọc radius, color và weight từ dòng lệnh và bắt lỗi trong các tình huống sau:

   Khi các tham số dòng lệnh không được cung cấp hoặc không đủ

   Khi giá trị bán kính không phải là giá trị số

Bài tập 2: Person Class

Xây dựng một lớp tên là **`Person`** và hai lớp con của `Person` có tên là **`Student`** và **`Employee`**. Cho **`Faculty`** và **`Staff`** là hai lớp con của `Employee`. Một người (Person) có thông tin `name`, `phone number` và `email address`. Một sinh viên (Student) có một chương trình mà bạn đó tham gia (Business, Computer Science...) . Một nhân viên (Employee) có thông tin `department`, `salary` và `date hired`. Một thành viên của khoa (Faculty) có thông tin `office hours` và `rank`. Một nhân viên có 1 chức danh (`Title`). Bạn cần làm:

1. Override the **`ToString`**() để hiển thị tên lớp, tên người và địa chỉ email.

2. Cung cấp các properties ở mỗi lớp để đọc và ghi các trường của lớp đó

3. Định nghĩa **`CalculateBonus`** và **`CalculateVacation`** như là các phương thức abstract trong lớp `Employee` và thực thi nó trong các lớp `Faculty` và `Staff` như sau

   o `Faculty` nhận 1000 + 0.05 x Salary và Staff get 0.06 x Salary

   o `Faculty` nhận 5 tuần họ được tuyển dụng trên 3 năm và thêm một nếu bạn đó là

   "Senior Lecturer". ngược lại là 4 tuần. Nhân viên nhận 4 tuần cho 5 năm phục vụ. Ngược lại nhận 3 tuần