ADVANCED SOFTWARE

Project code

STUDENT ID

77275260

Circle class:

```
Using
System;
          using System.Collections.Generic;
          using System.Drawing;
          using System.Linq;
          using System.Text;
          using System.Threading.Tasks;


          namespace Assingment
          {
              public class Circle : Shape
              {
                  int radius;


                  public Circle() : base()
                  {



                  }
                  /// <summary>
                  ///
                  /// </summary>
                  /// <param name="c"></param>
                  /// <param name="x">20</param>
                  /// <param name="y">10</param>
                  /// <param name="radius">30</param>
                  public Circle(Color c, int x, int y, int radius) : base(x, y)
                  {
                      this.radius = radius;
                  }
                  /// <summary>
                  ///
                  /// </summary>
                  /// <param name="g"></param>
                  /// <param name="c"></param>
                  /// <param name="thickness"></param>
                  public override void draw(Graphics g, Color c, int thickness)
                  {
                      Pen p = new Pen(c, thickness);
```

```
                g.DrawEllipse(p, x, y, radius, radius);
            }
            /// <summary>
            ///
            /// </summary>
            /// <param name="g">4</param>
            /// <param name="c">4</param>
            public override void fill(Graphics g, Color c)
            {
                SolidBrush fill = new SolidBrush(c);
                g.FillEllipse(fill, x, y, radius, radius);
            }
            public void setRadius(int radius)
            {
                this.radius = radius;
            }
            /// <summary>
            ///
            /// </summary>
            /// <returns></returns>
            public int getRadius()
            {
                return radius;
            }
            /// <summary>
            ///
            /// </summary>
            /// <param name="color">red</param>
            /// <param name="list">xyz</param>
            public override void set(Color color, params int[] list)
            {
                base.set(color, list[0], list[1]);
                this.radius = list[2];
            }
        }
    }
```

Form1 Designer Class:

```
namespace
Assingment
```

```
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;


        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }


        #region Windows Form Designer generated code


        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.outputbox = new System.Windows.Forms.PictureBox();
            this.cmdbox = new System.Windows.Forms.TextBox();
            this.cmdtext = new System.Windows.Forms.TextBox();
            this.button2 = new System.Windows.Forms.Button();
            this.openFileDialog1 = new System.Windows.Forms.OpenFileDialog();
            this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.homeToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
```

```
            this.saveToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.loadToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.exitToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.aboutToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.helpToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();

((System.ComponentModel.ISupportInitialize)(this.outputbox)).BeginInit();
            this.menuStrip1.SuspendLayout();
            this.SuspendLayout();
            //
            // outputbox
            //
            this.outputbox.BackColor = System.Drawing.SystemColors.Window;
            this.outputbox.Location = new System.Drawing.Point(0, 35);
            this.outputbox.Name = "outputbox";
            this.outputbox.Size = new System.Drawing.Size(386, 310);
            this.outputbox.TabIndex = 1;
            this.outputbox.TabStop = false;
            this.outputbox.Click += new
System.EventHandler(this.outputbox_Click);
            this.outputbox.Paint += new
System.Windows.Forms.PaintEventHandler(this.outputbox_Paint);
            //
            // cmdbox
            //
            this.cmdbox.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom
| System.Windows.Forms.AnchorStyles.Left)));
            this.cmdbox.Location = new System.Drawing.Point(0, 351);
            this.cmdbox.Multiline = true;
            this.cmdbox.Name = "cmdbox";
            this.cmdbox.Size = new System.Drawing.Size(640, 88);
            this.cmdbox.TabIndex = 3;
            this.cmdbox.TextChanged += new
System.EventHandler(this.cmdbox_TextChanged);
            this.cmdbox.Enter += new System.EventHandler(this.cmdbox_Enter);
            this.cmdbox.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.cmdbox_KeyDown);
```

```
            //
            // cmdtext
            //
            this.cmdtext.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Right)));
            this.cmdtext.Location = new System.Drawing.Point(388, 35);
            this.cmdtext.Margin = new System.Windows.Forms.Padding(4, 5, 4,
5);
            this.cmdtext.Multiline = true;
            this.cmdtext.Name = "cmdtext";
            this.cmdtext.Size = new System.Drawing.Size(411, 308);
            this.cmdtext.TabIndex = 8;
            this.cmdtext.TextChanged += new
System.EventHandler(this.textBox2_TextChanged);
            //
            // button2
            //
            this.button2.BackColor = System.Drawing.SystemColors.Highlight;
            this.button2.Location = new System.Drawing.Point(647, 365);
            this.button2.Margin = new System.Windows.Forms.Padding(4, 5, 4,
5);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(132, 70);
            this.button2.TabIndex = 11;
            this.button2.Text = "Run";
            this.button2.UseVisualStyleBackColor = false;
            this.button2.Click += new
System.EventHandler(this.button2_Click_1);
            //
            // openFileDialog1
            //
            this.openFileDialog1.FileName = "openFileDialog1";
            //
            // menuStrip1
            //
            this.menuStrip1.BackColor = System.Drawing.SystemColors.Highlight;
            this.menuStrip1.GripMargin = new System.Windows.Forms.Padding(2,
2, 0, 2);
            this.menuStrip1.ImageScalingSize = new System.Drawing.Size(24,
24);
            this.menuStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
```

```
            this.homeToolStripMenuItem,
            this.aboutToolStripMenuItem,
            this.helpToolStripMenuItem});
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(800, 33);
            this.menuStrip1.TabIndex = 13;
            this.menuStrip1.Text = "menuStrip1";
            //
            // homeToolStripMenuItem
            //
            this.homeToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.saveToolStripMenuItem,
            this.loadToolStripMenuItem,
            this.exitToolStripMenuItem});
            this.homeToolStripMenuItem.Name = "homeToolStripMenuItem";
            this.homeToolStripMenuItem.Size = new System.Drawing.Size(77, 29);
            this.homeToolStripMenuItem.Text = "Home";
            //
            // saveToolStripMenuItem
            //
            this.saveToolStripMenuItem.Name = "saveToolStripMenuItem";
            this.saveToolStripMenuItem.Size = new System.Drawing.Size(270,
34);
            this.saveToolStripMenuItem.Text = "Save";
            this.saveToolStripMenuItem.Click += new
System.EventHandler(this.saveToolStripMenuItem_Click_1);
            //
            // loadToolStripMenuItem
            //
            this.loadToolStripMenuItem.Name = "loadToolStripMenuItem";
            this.loadToolStripMenuItem.Size = new System.Drawing.Size(270,
34);
            this.loadToolStripMenuItem.Text = "Load";
            this.loadToolStripMenuItem.Click += new
System.EventHandler(this.loadToolStripMenuItem_Click_1);
            //
            // exitToolStripMenuItem
            //
            this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
            this.exitToolStripMenuItem.Size = new System.Drawing.Size(270,
34);
```

```
            this.exitToolStripMenuItem.Text = "Exit";
            this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click);
            //
            // aboutToolStripMenuItem
            //
            this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
            this.aboutToolStripMenuItem.Size = new System.Drawing.Size(78,
29);
            this.aboutToolStripMenuItem.Text = "About";
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(65, 29);
            this.helpToolStripMenuItem.Text = "Help";
            this.helpToolStripMenuItem.Click += new
System.EventHandler(this.helpToolStripMenuItem_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 20F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.SystemColors.MenuHighlight;
            this.ClientSize = new System.Drawing.Size(800, 449);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.cmdtext);
            this.Controls.Add(this.cmdbox);
            this.Controls.Add(this.outputbox);
            this.Controls.Add(this.menuStrip1);
            this.IsMdiContainer = true;
            this.Name = "Form1";
            this.Text = "Form1";
            this.Load += new System.EventHandler(this.Form1_Load);

((System.ComponentModel.ISupportInitialize)(this.outputbox)).EndInit();
            this.menuStrip1.ResumeLayout(false);
            this.menuStrip1.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();


        }
```

```
            #endregion
            private System.Windows.Forms.PictureBox outputbox;
            private System.Windows.Forms.TextBox cmdbox;
            private System.Windows.Forms.TextBox cmdtext;
            private System.Windows.Forms.Button button2;
            private System.Windows.Forms.OpenFileDialog openFileDialog1;
            private System.Windows.Forms.SaveFileDialog saveFileDialog1;
            private System.Windows.Forms.MenuStrip menuStrip1;
            private System.Windows.Forms.ToolStripMenuItem homeToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem saveToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem loadToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
            private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
        }
    }
```

From1.cs class:

```
using
Microsoft.CSharp;
                    using System;
                    using System.CodeDom.Compiler;
                    using System.Collections;
                    using System.Collections.Generic;
                    using System.ComponentModel;
                    using System.Data;
                    using System.Diagnostics;
                    using System.Drawing;
                    using System.IO;
                    using System.Linq;
                    using System.Text;
                    using System.Text.RegularExpressions;
                    using System.Threading.Tasks;
                    using System.Windows.Forms;


                    namespace Assingment
                    {
                        public partial class Form1 : Form
                        {
```

```
    Boolean paintTringle, fill;
    String syntax;
    String[] words;
    int moveX, moveY;
    int thickness;
    string actionCmd, syntaxCmd;
    ArrayList shapes = new ArrayList();
    Variables variable;
    List<Triangle> tringleObjects;
    List<Variables> variableObjects;
    Color c;
    Shape shape;
    ShapeFactory abstractFactory = new ShapeFactory();
    Triangle tringle;
    int counter;
    int loopCounter;
    string storeMethod;
    string methoName;




    private void cmdbox_TextChanged(object sender, EventArgs e)
    {
        actionCmd = cmdbox.Text.ToLower();
        syntaxCmd = cmdtext.Text;
    }


    private void button1_Click(object sender, EventArgs e)
    {








}
```

```
private void button2_Click(object sender, EventArgs e)
{
    // outputbox.InitialImage = null;
    cmdbox.Clear();


}


private void cmdbox_KeyDown(object sender, KeyEventArgs e)
{

}




private void cmdbox_Enter(object sender, EventArgs e)
{
}





private void textBox2_TextChanged(object sender, EventArgs e)
{

}



private void button1_Click_1(object sender, EventArgs e)
{
}



private void outputbox_Click(object sender, EventArgs e)
{

}
```

```
        private void saveToolStripMenuItem_Click_1(object sender,
EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                File.WriteAllText(saveFileDialog1.FileName,
cmdtext.Text);
            }
        }


        public Form1()
        {
            InitializeComponent();
        }


        private void button2_Click_1(object sender, EventArgs e)
        {
            if (cmdbox.Text == "" && cmdtext.Text == "")
            {
                MessageBox.Show("Both action command and syntax command
is empty! pl");
            }
            else
            {
                switch (actionCmd)
                {
                    case "run":
                        try
                        {
                            if (cmdtext.Text == "")
                            {
                                MessageBox.Show("Syntax and Parameter
is not Detected");
                            }
                            syntax = cmdtext.Text.ToLower();
                            //delimeters variables holds the array
                            char[] delimiters = new char[] { '\r', '\n'
};
                            //Holds invididuals column code line
                            string[] parts = syntax.Split(delimiters,
StringSplitOptions.RemoveEmptyEntries);


                            //loop through the whole row's code line
                            for (int i = 0; i < parts.Length; i++)
                            {
                                /* Hold single code line,
                                  for example at 0 position paint
circle, at 1 position color red 5
                                */
                                String code_line = parts[i];
                                //Splits the code when space
                                char[] value_code = new char[] { ' ' };
                                //Holds invididuals code line
                                words = code_line.Split(value_code,
StringSplitOptions.RemoveEmptyEntries);




                                //Calculation to add value to variable
                                if (Regex.IsMatch(words[0], @"^[a-zA-
Z]+$") && words[1].Equals("+"))
                                {
                                    //sets new incremented value to the
defined variable and puts it in vaiableObjects class

variableObjects[variableObjects.FindIndex(x =>
x.variable.Contains(words[0]))].

setValue(variableObjects[variableObjects.FindIndex(x =>
x.variable.Contains(words[0]))].
                                        getValue() +
Convert.ToInt32(words[2]));
                                }
                                if ((Regex.IsMatch(words[0], @"^[a-zA-
Z]+$") && words[1].Equals("=")))
                                {
                                    //add new variableObjects if
variableObject is empty
                                    if (variableObjects == null ||
variableObjects.Count == 0)
                                    {
                                        variable = new Variables();
                                        variable.setVariable(words[0]);
                                        int y =
Convert.ToInt32(words[2]);
                                        variable.setValue(y);
                                        variableObjects.Add(variable);
                                    }
                                    else
                                    {
                                        //else checks if variable
exists or not
                                        if (!variableObjects.Exists(x
=> x.variable == words[0]))
                                        {
                                            variable = new Variables();

variable.setVariable(words[0]);
                                            int y =
Convert.ToInt32(words[2]);
                                            variable.setValue(y);

variableObjects.Add(variable);
                                        }
                                        //else add new variable value
to variableObjects
                                        else
                                        {
                                            variable = new Variables();

variable.setVariable(words[0]);
                                            int y =
Convert.ToInt32(words[2]);
                                            variable.setValue(y);

variableObjects[variableObjects.FindIndex(x =>
x.variable.Contains(words[0]))] = variable;
                                        }
                                    }
```

```
                    }




                    //If the there is move word in syntax
                    if (words[0] == "move")
                    {
                        moveX = Convert.ToInt32(words[1]);
                        moveY = Convert.ToInt32(words[2]);
                    }


                    //If there is fill word in syntax
                    if (words[0] == "fill")
                    {
                        if (words[1] == "on")//checks if
the word[1] holds value'on'
                        {
                            fill = true;//sets fill ture
                        }
                        if (words[1] == "off")//checks if
the word[1] holds value 'off'
                        {
                            fill = false;//sets fill false
                        }
                    }


                    //Checks if syntax has color word of
not, if yes then
                    if (words[0] == "color")
                    {
                        //Convert string value to integer
value
                        thickness =
Convert.ToInt32(words[2]);



                        //If red color
                        if (words[1] == "red")
```

```
                                    {
                                        c = Color.Red;
                                    }
                                    //If blue color
                                    else if (words[1] == "blue")
                                    {
                                        c = Color.Blue;
                                    }
                                    //If green color
                                    else if (words[1] == "green")
                                    {
                                        c = Color.Green;
                                    }
                                    //If pink color
                                    else if (words[1] == "pink")
                                    {
                                        c = Color.Pink;
                                    }
                                    //If yellow color
                                    else if (words[1] == "yellow")
                                    {
                                        c = Color.Yellow;
                                    }
                                    //If purple color
                                    else if (words[1] == "purple")
                                    {
                                        c = Color.Purple;
                                    }
                                    //If brown color
                                    else if (words[1] == "brown")
                                    {
                                        c = Color.Brown;
                                    }
                                    //If not color then, set the deault
                black color

                                    else
                                    {
                                        c = Color.Red;
                                    }
                                }


                                //Check for 'paint' word
                                if (words[0].Equals("paint"))
                                {
                                    //Checks for 'circle' word
                                    if (words[1] == "circle")
                                    {
                                        if (words.Length != 3)
                                        {
                                            MessageBox.Show("!!!
Invalid syntax !!!\n eg.  'paint circle 150'");
                                        }
                                        else
                                        {
                                            if
(variableObjects.Exists(x => x.variable == words[2]) == true)
                                            //Assigns variable value to
paint code parameter value
                                            {
                                                words[2] =
Convert.ToString(variableObjects.ElementAt(variableObjects.
                                                    FindIndex(x =>
x.variable.Contains(words[2]))).getValue()); //variable value to radius
parameter
                                            }
                                            shape =
abstractFactory.getShape("circle");
                                            shape.set(c, moveX, moveY,
Convert.ToInt32(words[2]));
                                            shapes.Add(shape);
                                        }
                                    }



                                    //Check if the word is rectangle or
not
                                    else if
(words[1].Equals("rectangle"))
                                    {
                                        if (words.Length != 4)
                                        {
                                            MessageBox.Show("!!!
Invalid syntax !!!\n eg. 'paint rectangle 100 150'");
                                        }
                                        else
                                        {
                                            if
(variableObjects.Exists(x => x.variable == words[2] == true))
                                            {
                                                //Variable value to
height parameter
                                                words[2] =
Convert.ToString(variableObjects.ElementAt(variableObjects.
                                                    FindIndex(x =>
x.variable.Contains(words[2]))).getValue());
                                            }
                                            if
(variableObjects.Exists(x => x.variable == words[3]) == true)
                                            {
                                                //Variable value to
width parameter
                                                words[3] =
Convert.ToString(variableObjects.ElementAt(variableObjects.
                                                    FindIndex(x =>
x.variable.Contains(words[3]))).getValue());
                                            }
                                            shape =
abstractFactory.getShape("rectangle");
                                            shape.set(c, moveX, moveY,
Convert.ToInt32(words[2]), Convert.ToInt32(words[3]));
                                            shapes.Add(shape);
                                        }
                                    }



                                    //Check if the word is tringle or
not
                                    if (words[1].Equals("triangle"))
                                    {
                                        if (words.Length != 2)
                                        {
                                            MessageBox.Show("!!!
Invalid syntax !!!\n eg. 'paint tringle'");
                                        }
                                        else
                                        {
                                            if
(variableObjects.Exists(x => x.variable == words[2]) == true)
```

```
                                            //Assigns variable value to
paint code parameter value
                                            {
                                                words[2] =
Convert.ToString(variableObjects.ElementAt(variableObjects.
                                                    FindIndex(x =>
x.variable.Contains(words[2]))).getValue()); //variable value to side
parameter
                                            }
                                            Triangle tringle = new
Triangle();
                                            PointF[] points = { new
PointF(100, 100), new PointF(200, 100), new PointF(150, 10) };
                                            tringle.setPoints(points);

tringleObjects.Add(tringle);
                                            paintTringle = true;
                                        }
                                    }
                                }
                                if (words[0] == "loop")
                                {
                                    //Store value of words[1] into
loopCounter
                                    loopCounter =
Convert.ToInt32(words[1]);
                                }
                                //Checks if syntax have 'endloop' word
or not, then yes
                                if (parts[i] == "end loop") // code for
end loop statement
                                {
                                    //If counter to paint is not less
than loop counter
                                    if (counter < loopCounter)
                                    {
                                        i = Array.IndexOf(parts, "loop
" + loopCounter);
                                        //Value to increment paint
circle method
                                        counter += 1;
                                    }
                                    //Keep painting
```

```
                                        else
                                        {
                                            i = Array.IndexOf(parts, "end 
loop");
                                        }
                                    }




                                    //Function
                                    if (words[0] == "method")
                                    {
                                        storeMethod = words[0];
                                        methoName = words[1];
                                    }


                                    if (storeMethod == "method" && 
methoName == "myMethod")
                                    {




                                    }



                                    //If condition
                                    //Check wheather syntax contain 'if' 
word or not, if yes then
                                    //Code for if statement
                                    if (words[0] == "if")
                                    {
                                        //Declared string variable with 
varibale_name and store the value of 'word[1]' into it
                                        string variable_name = words[1];
                                        //Declared integer variable and 
store the value of of word[3]
                                        int value = 
Convert.ToInt32(words[3]);
                                        //Checks if condition defined in if 
condition matches with variable objects list
```

```
                                    if (variableObjects.Exists(x =>
x.variable == words[1]) == true
                                        && variableObjects.Exists(x =>
x.value == Convert.ToInt32(words[3])) == true)
                                    {
                                        Console.WriteLine("Entered
endside the if statement statement");
                                    }
                                    else
                                    {
                                        //Directed to end if line
                                        i = Array.IndexOf(parts, "end
if");

                                    }
                                }


                            }
                        }
                        catch (IndexOutOfRangeException ex)
                        {
                            Console.WriteLine("Error" + " " + ex);
                        }
                        catch (FormatException ex)
                        {
                            Console.WriteLine("Enter the correct
parameter" + " " + ex);


                        }
                        catch (ArgumentOutOfRangeException ex)
                        {
                            Console.WriteLine("Enter the correct
parameter" + " " + ex);
                        }
                        outputbox.Refresh();
                        break;
                    case "clear":
                        shapes.Clear();
                        tringleObjects.Clear();
                        cmdtext.Clear();
                        outputbox.Refresh();
                        break;
                    case "reset":
                        moveX = 0;
                        moveY = 0;
                        outputbox.Refresh();
                        break;
                    default:
                        MessageBox.Show("The action command is empty\n"
+
                            "\n" +
                            "Must be: 'run' for Execuit the app\n" +
                            "Must be: 'clear' for Fresh Start"
                            );
                        break;
                }
            }


        }


        private void outputbox_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;




            //Paint shapes
            for (int i = 0; i < shapes.Count; i++)
            {
                shape = (Shape)shapes[i];
                if (shape != null)
                {
                    shape.draw(g, c, thickness);
                    //check if fill is one or not
                    if (fill == true)
                    {
                        shape.fill(g, c);
                    }
                }
                else
                {
                    Console.WriteLine("Invalid Shape in array");
                }
            }




            //If paintTringle condition is true then
            if (paintTringle == true)// paint condition is true then
            {
                foreach (Triangle trangleObject in tringleObjects)
                {
                    //If fill is on then fill the shape with color
                    if (fill == true)
                    {
                        trangleObject.fill(g, c);
                    }
                    //If fill is off, then
                    else
                    {
                        trangleObject.draw(g, c, thickness);
                    }
                }
            }
        }


        private void helpToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            MessageBox.Show(
            "-------------------HINTS------------------\n" +
            "COMMANDS TO DISPLAY THE SHAPES \n" +
            "-----------------------------\n" +
            "Example :- \n" +
            "paint rectangle 100 150\n" +
            "paint circle 150 \n" +
            "paint tringle \n" +
            "-----------------------------------------\n" +
            "TO CHANGE THE CO-ORDINATE OF THE SHAPES \n" +
            "-----------------------------\n" +
            "Example :- \n" +
            "move 50 50\n" +
            "---------------------------------------\n" +
            "TO CHANGE THE COLOR OF SHAPES \n" +
            "--------------------------------\n" +
            "Example :- \n" +
            "color red 10\n " +
            "-----------------------------------------------\n" +
            "TO FILL AND UNFILL COLOR \n" +
            "--------------------------------\n" +
            "Example :- \n" +
            "fill on \n" +
            "fill off \n" +
            "-----------------------------------------------\n" +
            "TO PAINT THE SAHPES USING VARIABLES \n" +
            "------------------------------------------\n" +
            "Example :- \n" +
            "radius = 150\n" +
            "paint circle radius\n" +
            "-------------------------------------------------\n" +
            "IF STATEMENT:\n" +
            "--------------------------------\n" +
            "Example :- \n" +
            "a = 5 \n if a = 5 then \n paint circle 100 \n end if \n" +
            "--------------------------------------------\n" +
            "FOR LOOPING: \n" +
            "--------------------------------\n" +
            "Example :- \n" +
            "r = 5 \n loop 3 \n r + 50 \n paint circle r \n endloop \n
" +
            "--------------------------------\n"
            );


        }


        private void exitToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Application.Exit();
        }


        private void listBox1_SelectedIndexChanged(object sender,
EventArgs e)
```

```
        {

        }


        private void Form1_Load(object sender, EventArgs e)
        {
            tringleObjects = new List<Triangle>(); //creates array of
new polygon object
            variableObjects = new List<Variables>();//creates array of
new variables objects
            //Sets the color on startUp
            c = Color.DarkCyan;
        }


        private void loadToolStripMenuItem_Click_1(object sender,
EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                cmdtext.Text =
File.ReadAllText(openFileDialog1.FileName);
            }
        }
    }
    }
```

Ifactory.cs Inteface:

```
using
System;
          using System.Collections.Generic;
          using System.Drawing;
          using System.Linq;
          using System.Text;
          using System.Threading.Tasks;


          namespace Assingment
          {
            public  interface Ifactory
              {
                  void draw(Graphics g, Color c, int thickness);
                  void fill(Graphics g, Color c);
              }
          }
```

Move Direction.cs class:

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


        namespace Assingment
        {
           public class MoveDirection
            {
                public MoveDirection()
                {


                }
                public int x { get; set; }

                public int y { get; set; }
            }
        }
```

Program.cs class:

```
using
System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Threading.Tasks;
        using System.Windows.Forms;


        namespace Assingment
```

```
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Rectangle class:

```
using
System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace Assingment
{
    public class Rectangle: Shape
    {
        int height, width;
        public Rectangle()
        {
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="color"></param>
        /// <param name="x"></param>
        /// <param name="y"></param>
        /// <param name="3"></param>
        /// <param name="8"></param>
        public Rectangle(Color color, int x, int y, int height, int width) : 
base(x, y)
        {
            this.height = height;
            this.width = width;
        }
        public override void draw(Graphics g, Color c, int thickness)
        {
            Pen p = new Pen(c, thickness);
            g.DrawRectangle(p, x, y, height, width);
        }
        public override void fill(Graphics g, Color c)
        {
            SolidBrush brush = new SolidBrush(c);
            g.FillRectangle(brush, x, y, height, width);
        }
        public void setHeight(int height)
        {
            this.height = height;
        }
        public void setWidth(int width)
        {
            this.width = width;
        }
        public int getHeight()
        {
            return height;
        }
        public int getWidth()
        {
            return width;
        }
        public override void set(Color color, params int[] list)
        {
            base.set(color, list[0], list[1]);
            this.height = list[2];
            this.width = list[3];
        }
    }
}
```

Shape class:

```
using
System;
          using System.Collections.Generic;
          using System.Drawing;
          using System.Linq;
          using System.Text;
          using System.Threading.Tasks;


          namespace Assignment
          {
              public abstract class Shape : Ifactory
              {
                  protected int x = 0, y = 0, z = 0;
                  protected Color color;


                  public Shape()
                  {


                  }
                  /// <summary>
                  ///
                  /// </summary>
                  /// <param name="x">10</param>
                  /// <param name="y">10</param>
                  public Shape(int x, int y)
                  {
                      this.x = x;
                      this.y = y;
                  }
                  /// <summary>
                  ///
                  /// </summary>
                  /// <param name="x">5</param>
                  /// <param name="y">10</param>
                  /// <param name="z">15</param>
                  public Shape(int x, int y, int z)
                  {
                      this.x = x;
                      this.y = y;
                      this.z = z;
```

```
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="x">5</param>
        public void setX(int x)
        {
            this.x = x;
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="y">10</param>
        public void setY(int y)
        {
            this.y = y;
        }
        /// <summary>
        ///
        /// </summary>
        /// <returns></returns>
        public int getX()
        {
            return x;
        }
        public int getY()
        {
            return y;
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="color">green</param>
        /// <param name="list">no of list</param>
        public virtual void set(Color color, params int[] list)
        {
            this.color = color;
            this.x = list[0];
            this.y = list[1];
        }
        public abstract void draw(Graphics g, Color c, int thickness);
        public abstract void fill(Graphics g, Color c);
    }
```

```
            }
```

Shape Factory class:

```
 using
 System;
            using System.Collections.Generic;
            using System.Linq;
            using System.Text;
            using System.Threading.Tasks;
            using System.Windows.Forms;


            namespace Assingment
            {
                public class ShapeFactory
                {
                    public Shape getShape(string shapeType)
                    {
                        shapeType = shapeType.ToLower().Trim();


                        if (shapeType == null)
                        {
                            return null;
                        }
                        else if (shapeType.Equals("circle"))
                        {
                            return new Circle();
                        }
                        else if (shapeType.Equals("rectangle"))
                        {
                            return new Rectangle();
                        }
                        else if (shapeType.Equals("triangle"))
                        {
                            return new Triangle();
                        }
                        else
                        {
                            MessageBox.Show("Factory error: " + shapeType + " does not
            exist");
```

```
                        return null;
                    }


                }
            }
        }
```

Shape factory Def class:

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


          namespace Assingment
          {
              class ShapeFactoryDef
              {
                  /// <summary>
                  ///
                  /// </summary>
                  /// <param name="shape">circle</param>
                  /// <returns></returns>
                  public bool isCircle(string shape)
                  {
                      if (shape == "circle")
                      {
                          return true;
                      }
                      return false;
                  }
                  public bool isRectangle(string shape)
                  {
                      if (shape == "rectangle")
                      {
                          return true;
                      }
```

```
                return false;
            }
            public bool isTriangle(string shape)
            {
                if (shape == "triangle")
                {
                    return true;
                }
                return false;
            }
        }
    }
```

Triangle class:

```
using
System;
          using System.Collections.Generic;
          using System.Drawing;
          using System.Linq;
          using System.Text;
          using System.Threading.Tasks;


          namespace Assingment
          {
             class Triangle: Shape
              {
                  PointF[] point;


                  public Triangle()
                  {


                  }
                  /// <summary>
                  ///
                  /// </summary>
                  /// <param name="point">50</param>
                  public Triangle(PointF[] point)
                  {
```

```
                this.point = point;
            }
            public Triangle(Color color, int x, int y, PointF[] point) : base(x, y)
            {
                this.point = point;
            }
             public override void draw(Graphics g, Color c, int thickness)
            {
                Pen p = new Pen(c);
                g.DrawPolygon(p, point);
            }
            public override void fill(Graphics g, Color c)
            {
                SolidBrush fill = new SolidBrush(c);
                g.FillPolygon(fill, point);
            }
            public void setPoints(PointF[] point)
            {
                this.point = point;
            }
            public PointF[] getPoint()
            {
                return this.point;
            }
        }
    }
```

Variable Class:

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


    namespace Assignment
    {
       public class Variables
        {
            //String variable set and get
            public string variable { get; set; }
```

```
//Float value get and set
public float value { get; set; }




//set and get mehtod of variable
/// <summary>
/// setVariable method to set varible
/// </summary>
/// <param name="variable">return varible</param>
public void setVariable(string variable)
{
    this.variable = variable;
}


/// <summary>
/// This method get the variable
/// </summary>
/// <returns></returns>
public string getVariable()
{
    return this.variable;
}




/// <summary>
/// Set and get method of value
/// </summary>
/// <param name="value">return the value of set</param>
public void setValue(float value)
{
    this.value = value;
}


/// <summary>
/// method to retunrn the value
```

```
        /// </summary>
        /// <returns>return value</returns>


        public float getValue()
        {
            return this.value;
        }
    }
}
```