# Documentation of Component 2:

# Code use in program:

|  |  |
| --- | --- |
|  |  |
| Form | using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Drawing.Drawing2D;  using System.IO;  using System.Linq;  using System.Text;  using System.Text.RegularExpressions;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace FinalAssignment  {  public partial class Form1 : Form  {  public Form1()  {  InitializeComponent();  g = Pnl\_Draw.CreateGraphics();  }  /// <summary>  /// variable to create triangle side  /// </summary>  Color btnBorderColor = Color.FromArgb(104, 162, 255);  Color mainColor = Color.Black;  int size = 2;  Graphics g;  int x, y = -1;  int mouseX, mouseY = 0;  Boolean moving = false;  Pen pen;  String active = "pen";  OpenFileDialog openFile = new OpenFileDialog();  String line = "";  Validation validate;  int loopCounter = 0;  Boolean hasDrawOrMoveValue = false;  public int radius = 0;  public int width = 0;  public int height = 0;  public int dSize = 0;  public int counter = 0;  string shape;  ShapeFactory shapeFactory = new ShapeFactory();  Shape shapes;  private void btn\_exec\_Click(object sender, EventArgs e)  {  hasDrawOrMoveValue = false;  if (txtCommand.Text != null && txtCommand.Text != "")  {  validate = new Validation(txtCommand);  if (!validate.isSomethingInvalid)  {  MessageBox.Show("Successful.... Click on OK to see the result!!");  loadCommand();  }  }  }  private void loadCommand()  {  int numberOfLines = txtCommand.Lines.Length;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  Boolean hasDrawto = Regex.IsMatch(oneLineCommand.ToLower(), @"\bdrawto\b");  Boolean hasMoveto = Regex.IsMatch(oneLineCommand.ToLower(), @"\bmoveto\b");  if (hasDrawto || hasMoveto)  {  String args = oneLineCommand.Substring(6, (oneLineCommand.Length - 6));  String[] parms = args.Split(',');  for (int j = 0; j < parms.Length; j++)  {  parms[j] = parms[j].Trim();  }  mouseX = int.Parse(parms[0]);  mouseY = int.Parse(parms[1]);  hasDrawOrMoveValue = true;  }  else  {  hasDrawOrMoveValue = false;  }  if (hasMoveto)  {  Pnl\_Draw.Refresh();  }  }  }  for (loopCounter = 0; loopCounter < numberOfLines; loopCounter++)  {  String oneLineCommand = txtCommand.Lines[loopCounter];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  RunCommand(oneLineCommand);  }  }  }  /\*\*  \* The code are executed when the button is clicked  \*/  private void RunCommand(String oneLineCommand)  {  Boolean hasPlus = oneLineCommand.Contains('+');  Boolean hasEquals = oneLineCommand.Contains("=");  if (hasEquals)  {  oneLineCommand = Regex.Replace(oneLineCommand, @"\s+", " ");  string[] words = oneLineCommand.Split(' ');  //removing white spaces in between words  for (int i = 0; i < words.Length; i++)  {  words[i] = words[i].Trim();  }  String firstWord = words[0].ToLower();  if (firstWord.Equals("if"))  {  Boolean loop = false;  if (words[1].ToLower().Equals("radius"))  {  if (radius == int.Parse(words[3]))  {  loop = true;  }  }  else if (words[1].ToLower().Equals("width"))  {  if (width == int.Parse(words[3]))  {  loop = true;  }  }  else if (words[1].ToLower().Equals("height"))  {  if (height == int.Parse(words[3]))  {  loop = true;  }  }  else if (words[1].ToLower().Equals("counter"))  {  if (counter == int.Parse(words[3]))  {  loop = true;  }  }  int ifStartLine = (GetIfStartLineNumber());  int ifEndLine = (GetEndifEndLineNumber() - 1);  loopCounter = ifEndLine;  if (loop)  {  for (int j = ifStartLine; j <= ifEndLine; j++)  {  string oneLineCommand1 = txtCommand.Lines[j];  oneLineCommand1 = oneLineCommand1.Trim();  if (!oneLineCommand1.Equals(""))  {  RunCommand(oneLineCommand1);  }  }  }  else  {  MessageBox.Show("If Statement is false");  }  }  else  {  string[] words2 = oneLineCommand.Split('=');  for (int j = 0; j < words2.Length; j++)  {  words2[j] = words2[j].Trim();  }  if (words2[0].ToLower().Equals("radius"))  {  radius = int.Parse(words2[1]);  }  else if (words2[0].ToLower().Equals("width"))  {  width = int.Parse(words2[1]);  }  else if (words2[0].ToLower().Equals("height"))  {  height = int.Parse(words2[1]);  }  else if (words2[0].ToLower().Equals("counter"))  {  counter = int.Parse(words2[1]);  }  }  }  else if (hasPlus)  {  oneLineCommand = System.Text.RegularExpressions.Regex.Replace(oneLineCommand, @"\s+", " ");  string[] words = oneLineCommand.Split(' ');  if (words[0].ToLower().Equals("repeat"))  {  counter = int.Parse(words[1]);  if (words[2].ToLower().Equals("circle"))  {  int increaseValue = GetSize(oneLineCommand);  radius = increaseValue;  for (int j = 0; j < counter; j++)  {  DrawCircle(radius);  radius += increaseValue;  }  }  else if (words[2].ToLower().Equals("rectangle"))  {  int increaseValue = GetSize(oneLineCommand);  dSize = increaseValue;  for (int j = 0; j < counter; j++)  {  DrawRectangle(dSize, dSize);  dSize += increaseValue;  }  }  else if (words[2].ToLower().Equals("triangle"))  {  int increaseValue = GetSize(oneLineCommand);  dSize = increaseValue;  for (int j = 0; j < counter; j++)  {  DrawTriangle(dSize, dSize, dSize);  dSize += increaseValue;  }  }  }  else  {  string[] words2 = oneLineCommand.Split('+');  for (int j = 0; j < words2.Length; j++)  {  words2[j] = words2[j].Trim();  }  if (words2[0].ToLower().Equals("radius"))  {  radius += int.Parse(words2[1]);  }  else if (words2[0].ToLower().Equals("width"))  {  width += int.Parse(words2[1]);  }  else if (words2[0].ToLower().Equals("height"))  {  height += int.Parse(words2[1]);  }  }  }  else  {  sendDrawCommand(oneLineCommand);  }  }  /// <summary>  /// Returns the size of structure  /// </summary>  /// <param name="lineCommand"></param>  /// <returns></returns>  private int GetSize(string lineCommand)  {  int value = 0;  if (lineCommand.ToLower().Contains("radius"))  {  int pos = (lineCommand.IndexOf("radius") + 6);  int size = lineCommand.Length;  String tempLine = lineCommand.Substring(pos, (size - pos));  tempLine = tempLine.Trim();  String newTempLine = tempLine.Substring(1, (tempLine.Length - 1));  newTempLine = newTempLine.Trim();  value = int.Parse(newTempLine);  }  else if (lineCommand.ToLower().Contains("size"))  {  int pos = (lineCommand.IndexOf("size") + 4);  int size = lineCommand.Length;  String tempLine = lineCommand.Substring(pos, (size - pos));  tempLine = tempLine.Trim();  String newTempLine = tempLine.Substring(1, (tempLine.Length - 1));  newTempLine = newTempLine.Trim();  value = int.Parse(newTempLine);  }  return value;  }  /\*\*  \* Initiate shapes and figure to build shapes  \*/  private void sendDrawCommand(string lineOfCommand)  {  String[] shapes = { "circle", "rectangle", "triangle", "polygon" };  String[] variable = { "radius", "width", "height", "counter", "size" };  lineOfCommand = System.Text.RegularExpressions.Regex.Replace(lineOfCommand, @"\s+", " ");  string[] words = lineOfCommand.Split(' ');  //removing white spaces in between words  for (int i = 0; i < words.Length; i++)  {  words[i] = words[i].Trim();  }  String firstWord = words[0].ToLower();  Boolean firstWordShape = shapes.Contains(firstWord);  if (firstWordShape)  {  if (firstWord.Equals("circle"))  {  Boolean secondWordIsVariable = variable.Contains(words[1].ToLower());  if (secondWordIsVariable)  {  if (words[1].ToLower().Equals("radius"))  {  DrawCircle(radius);  }  }  else  {  DrawCircle(Int32.Parse(words[1]));  }  }  else if (firstWord.Equals("rectangle"))  {  String args = lineOfCommand.Substring(9, (lineOfCommand.Length - 9));  String[] parms = args.Split(',');  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  }  Boolean secondWordIsVariable = variable.Contains(parms[0].ToLower());  Boolean thirdWordIsVariable = variable.Contains(parms[1].ToLower());  if (secondWordIsVariable)  {  if (thirdWordIsVariable)  {  DrawRectangle(width, height);  }  else  {  DrawRectangle(width, Int32.Parse(parms[1]));  }  }  else  {  if (thirdWordIsVariable)  {  DrawRectangle(Int32.Parse(parms[0]), height);  }  else  {  DrawRectangle(Int32.Parse(parms[0]), Int32.Parse(parms[1]));  }  }  }  else if (firstWord.Equals("triangle"))  {  String args = lineOfCommand.Substring(8, (lineOfCommand.Length - 8));  String[] parms = args.Split(',');  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  }  DrawTriangle(Int32.Parse(parms[0]), Int32.Parse(parms[1]), Int32.Parse(parms[2]));  }  else if (firstWord.Equals("polygon"))  {  String args = lineOfCommand.Substring(8, (lineOfCommand.Length - 8));  String[] parms = args.Split(',');  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  }  if (parms.Length == 8)  {  DrawPolygon(Int32.Parse(parms[0]), Int32.Parse(parms[1]), Int32.Parse(parms[2]), Int32.Parse(parms[3]),  Int32.Parse(parms[4]), Int32.Parse(parms[5]), Int32.Parse(parms[6]), Int32.Parse(parms[7]));  }  else if (parms.Length == 10)  {  DrawPolygon(Int32.Parse(parms[0]), Int32.Parse(parms[1]), Int32.Parse(parms[2]), Int32.Parse(parms[3]),  Int32.Parse(parms[4]), Int32.Parse(parms[5]), Int32.Parse(parms[6]), Int32.Parse(parms[7]),  Int32.Parse(parms[8]), Int32.Parse(parms[9]));  }  }  }  else  {  if (firstWord.Equals("loop"))  {  counter = int.Parse(words[1]);  int loopStartLine = (GetLoopStartLineNumber());  int loopEndLine = (GetLoopEndLineNumber() - 1);  loopCounter = loopEndLine;  for (int i = 0; i < counter; i++)  {  for (int j = loopStartLine; j <= loopEndLine; j++)  {  String oneLineCommand = txtCommand.Lines[j];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  RunCommand(oneLineCommand);  }  }  }  }  else if (firstWord.Equals("if"))  {  Boolean loop = false;  if (words[1].ToLower().Equals("radius"))  {  if (radius == int.Parse(words[1]))  {  loop = true;  }  }  else if (words[1].ToLower().Equals("width"))  {  if (width == int.Parse(words[1]))  {  loop = true;  }  }  else if (words[1].ToLower().Equals("height"))  {  if (height == int.Parse(words[1]))  {  loop = true;  }  }  else if (words[1].ToLower().Equals("counter"))  {  if (counter == int.Parse(words[1]))  {  loop = true;  }  }  int ifStartLine = (GetIfStartLineNumber());  int ifEndLine = (GetEndifEndLineNumber() - 1);  loopCounter = ifEndLine;  if (loop)  {  for (int j = ifStartLine; j <= ifEndLine; j++)  {  String oneLineCommand = txtCommand.Lines[j];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  RunCommand(oneLineCommand);  }  }  }  }  }  }  /// <summary>  /// initiates loop  /// </summary>  /// <returns></returns>  private int GetEndifEndLineNumber()  {  int numberOfLines = txtCommand.Lines.Length;  int lineNum = 0;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = oneLineCommand.Trim();  if (oneLineCommand.ToLower().Equals("endif"))  {  lineNum = i + 1;  }  }  return lineNum;  }  /// <summary>  /// initiates if there is an if clause  /// </summary>  /// <returns></returns>  private int GetIfStartLineNumber()  {  int numberOfLines = txtCommand.Lines.Length;  int lineNum = 0;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = Regex.Replace(oneLineCommand, @"\s+", " ");  string[] words = oneLineCommand.Split(' ');  //removing white spaces in between words  for (int j = 0; j < words.Length; j++)  {  words[j] = words[j].Trim();  }  String firstWord = words[0].ToLower();  oneLineCommand = oneLineCommand.Trim();  if (firstWord.Equals("if"))  {  lineNum = i + 1;  }  }  return lineNum;  }  /// <summary>  /// Initiates loops  /// </summary>  /// <returns></returns>  private int GetLoopEndLineNumber()  {  try  {  int numberOfLines = txtCommand.Lines.Length;  int lineNum = 0;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = oneLineCommand.Trim();  if (oneLineCommand.ToLower().Equals("endloop"))  {  lineNum = i + 1;  }  }  return lineNum;  }  catch (Exception e)  {  return 0;  }  }  private int GetLoopStartLineNumber()  {  int numberOfLines = txtCommand.Lines.Length;  int lineNum = 0;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = Regex.Replace(oneLineCommand, @"\s+", " ");  string[] words = oneLineCommand.Split(' ');  //removing white spaces in between words  for (int j = 0; j < words.Length; j++)  {  words[j] = words[j].Trim();  }  String firstWord = words[0].ToLower();  oneLineCommand = oneLineCommand.Trim();  if (firstWord.Equals("loop"))  {  lineNum = i + 1;  }  }  return lineNum;  }  private void DrawPolygon(int v1, int v2, int v3, int v4, int v5, int v6, int v7, int v8)  {  Pen myPen = new Pen(mainColor);  Point[] pnt = new Point[5];  pnt[0].X = mouseX;  pnt[0].Y = mouseY;  pnt[1].X = mouseX - v1;  pnt[1].Y = mouseY - v2;  pnt[2].X = mouseX - v3;  pnt[2].Y = mouseY - v4;  pnt[3].X = mouseX - v5;  pnt[3].Y = mouseY - v6;  pnt[4].X = mouseX - v7;  pnt[4].Y = mouseY - v8;  g.DrawPolygon(myPen, pnt);  }  /\*\*  \* Draw Polygon  \*/  private void DrawPolygon(int v1, int v2, int v3, int v4, int v5, int v6, int v7, int v8, int v9, int v10)  {  Pen myPen = new Pen(mainColor);  Point[] pnt = new Point[6];  pnt[0].X = mouseX;  pnt[0].Y = mouseY;  pnt[1].X = mouseX - v1;  pnt[1].Y = mouseY - v2;  pnt[2].X = mouseX - v3;  pnt[2].Y = mouseY - v4;  pnt[3].X = mouseX - v5;  pnt[3].Y = mouseY - v6;  pnt[4].X = mouseX - v7;  pnt[4].Y = mouseY - v8;  pnt[5].X = mouseX - v9;  pnt[5].Y = mouseY - v10;  g.DrawPolygon(myPen, pnt);  }  /\*\*  \* Draws a triangle  \*/  private void DrawTriangle(int rBase, int adj, int hyp)  {  Pen myPen = new Pen(mainColor);  Point[] pnt = new Point[3];  pnt[0].X = mouseX;  pnt[0].Y = mouseY;  pnt[1].X = mouseX - rBase;  pnt[1].Y = mouseY;  pnt[2].X = mouseX;  pnt[2].Y = mouseY - adj;  g.DrawPolygon(myPen, pnt);  }  private void DrawRectangle(int width, int height)  {  Pen myPen = new Pen(mainColor);  g.DrawRectangle(myPen, mouseX - width / 2, mouseY - height / 2, width, height);  }  private void DrawCircle(int radius)  {  Pen myPen = new Pen(mainColor);  g.DrawEllipse(myPen, mouseX - radius, mouseY - radius, radius \* 2, radius \* 2);  }  /// <summary>  /// This function will load the text file from desired location  /// </summary>  /// <param name="sender"></param>  /// <param name="e"></param>  private void loadToolStripMenuItem\_Click(object sender, EventArgs e)  {  // txt\_cmd.Text = File.ReadAllText(OpenFileDialog.);  OpenFileDialog of = new OpenFileDialog();  of.Filter = "Text Document(\*.txt) | \*.txt";  if (of.ShowDialog() == DialogResult.OK)  {  txt\_cmd.Text = File.ReadAllText(of.FileName);  }  }  /// <summary>  /// to save running program to textfile.  /// </summary>  /// <param name="sender"></param>  /// <param name="e"></param>  private void saveToolStripMenuItem\_Click(object sender, EventArgs e)  {  SaveFileDialog sv = new SaveFileDialog();  sv.Filter = "Text Document(\*.txt)|\*.txt|All Files(\*.\*)|\*.\*";  if(sv.ShowDialog() == DialogResult.OK)  {  System.IO.File.WriteAllText(sv.FileName, txt\_cmd.Text);  }  }  /// <summary>  /// On clicking drawing panel, it displays x and y axis value  /// </summary>  /// <param name="sender"></param>  /// <param name="e"></param>  private void Pnl\_Draw\_MouseClick(object sender, MouseEventArgs e)  {  lbl\_StartPosX.Text = (e.X).ToString();  lbl\_StartPosY.Text = (e.Y).ToString();  }  private void groupBox1\_Enter(object sender, EventArgs e)  {  }  public int \_size1, \_size2, \_size3, \_size4, \_size5, \_size6, \_size7, \_size8, \_size9, \_size10, \_size11, \_size12;  private void Form1\_Load(object sender, EventArgs e)  {  }  private void Pnl\_Draw\_Paint(object sender, PaintEventArgs e)  {  }  private void helpToolStripMenuItem\_Click(object sender, EventArgs e)  {  MessageBox.Show("Version 1.0.0 || saurab adhikari ");  }  private void btnClear\_Click(object sender, EventArgs e)  {  txt\_cmd.Text = "";  Graphics g1 = Pnl\_Draw.CreateGraphics();  g1.Clear(Pnl\_Draw.BackColor);  txtCommand.Text = "";  }  private void button1\_Click(object sender, EventArgs e)  {  txt\_cmd.Text = "";  Graphics g1 = Pnl\_Draw.CreateGraphics();  g1.Clear(Pnl\_Draw.BackColor);  txtCommand.Text = "";  \_size1 = 0;  \_size2 = 0;  lbl\_StartPosX.Text = \_size1.ToString();  lbl\_StartPosY.Text = \_size2.ToString();  }  /// <summary>  /// for Triangle sides  /// </summary>  public int xi1, yi1, xi2, yi2, xii1, yii1, xii2, yii2, xiii1, yiii1, xiii2, yiii2;  Color paintcolor = Color.Blue;  Brush bb = new HatchBrush(HatchStyle.Wave, Color.Red, Color.FromArgb(255, 128, 255, 255));  int texturestyle = 5;  // Graphics g;  /// <summary>  /// all logic to run command in application  /// </summary>  /// <param name="sender"></param>  /// <param name="e"></param>  private void btn\_run\_Click(object sender, EventArgs e)  {  Regex regexDrRect = new Regex(@"drawto (.\*[\d])([,])(.\*[\d]) rectangle (.\*[\d])([,])(.\*[\d])");  Regex regexDrCircle = new Regex(@"drawto (.\*[\d])([,])(.\*[\d]) circle (.\*[\d])");  Regex regexDrTri = new Regex(@"drawto (.\*[\d])([,])(.\*[\d]) triangle (.\*[\d])([,])(.\*[\d])([,])(.\*[\d])");  Regex regexClear = new Regex(@"clear");  Regex regexReset = new Regex(@"reset");  Regex regexMT = new Regex(@"moveto (.\*[\d])([,])(.\*[\d])");  Regex regexR = new Regex(@"rectangle (.\*[\d])([,])(.\*[\d])");  Regex regexC = new Regex(@"circle (.\*[\d])");  Regex regexT = new Regex(@"triangle (.\*[\d])([,])(.\*[\d])([,])(.\*[\d])");    Match matchDrRect = regexDrRect.Match(txt\_cmd.Text.ToLower());  Match matchDrCircle = regexDrCircle.Match(txt\_cmd.Text.ToLower());  Match matchDrTri = regexDrTri.Match(txt\_cmd.Text.ToLower());  Match matchClear = regexClear.Match(txt\_cmd.Text.ToLower());  Match matchReset = regexReset.Match(txt\_cmd.Text.ToLower());  Match matchMT = regexMT.Match(txt\_cmd.Text.ToLower());  Match matchR = regexR.Match(txt\_cmd.Text.ToLower());  Match matchC = regexC.Match(txt\_cmd.Text.ToLower());  Match matchT = regexT.Match(txt\_cmd.Text.ToLower());    if (matchDrRect.Success || matchDrCircle.Success || matchDrTri.Success || matchClear.Success ||  matchReset.Success || matchMT.Success || matchR.Success || matchC.Success || matchT.Success)  {  //----------------RECTANGLE WITH DrawTo-----------------------//  if (matchDrRect.Success)  {  try  {  g = Pnl\_Draw.CreateGraphics();  \_size1 = int.Parse(matchDrRect.Groups[1].Value);  \_size2 = int.Parse(matchDrRect.Groups[3].Value);  \_size3 = int.Parse(matchDrRect.Groups[4].Value);  \_size4 = int.Parse(matchDrRect.Groups[6].Value);  ShapeFactory shapeFactory = new ShapeFactory();  Shape c = shapeFactory.GetShape("rectangle");  c.set(texturestyle, bb, paintcolor, \_size1, \_size2, \_size3, \_size4);  c.Draw(g);  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  }  //----------------RECTANGLE-----------------------//  else if (matchR.Success)  {  try  {  g = Pnl\_Draw.CreateGraphics();  \_size1 = int.Parse(lbl\_StartPosX.Text);  \_size2 = int.Parse(lbl\_StartPosY.Text);  \_size3 = int.Parse(matchR.Groups[1].Value);  \_size4 = int.Parse(matchR.Groups[3].Value);  ShapeFactory shapeFactory = new ShapeFactory();  Shape c = shapeFactory.GetShape("rectangle");  c.set(texturestyle, bb, paintcolor, \_size1, \_size2, \_size3, \_size4);  c.Draw(g);  }  catch (Exception ex)  {  MessageBox.Show("Error! Parameter should be in this form: \"rectangle width, height\"");  }  }  //----------------CIRCLE-----------------------//  else if (matchC.Success)  {  try  {  g = Pnl\_Draw.CreateGraphics();  \_size1 = int.Parse(lbl\_StartPosX.Text);  \_size2 = int.Parse(lbl\_StartPosY.Text);  \_size3 = int.Parse(matchC.Groups[1].Value);  ShapeFactory shapeFactory = new ShapeFactory();  Shape c = shapeFactory.GetShape("circle");  c.set(texturestyle, bb, paintcolor, \_size1, \_size2, \_size3 \* 2, \_size3 \* 2);  //c.draw(set);  c.Draw(g);  }  catch (Exception ex)  {  MessageBox.Show("Error! Parameter should be in this form: \"circle radius\"");  }  }  // ----------------TRIANGLE WITH DrawTo---------------------- -//  else if (matchDrTri.Success)  {  try  {  g = Pnl\_Draw.CreateGraphics();  \_size1 = int.Parse(matchDrTri.Groups[1].Value);  \_size2 = int.Parse(matchDrTri.Groups[3].Value);  \_size3 = int.Parse(matchDrTri.Groups[4].Value);  \_size4 = int.Parse(matchDrTri.Groups[6].Value);  \_size5 = int.Parse(matchDrTri.Groups[8].Value);  xi1 = \_size1;  yi1 = \_size2;  xi2 = Math.Abs(\_size3);  yi2 = \_size2;  xii1 = \_size1;  yii1 = \_size2;  xii2 = \_size1;  yii2 = Math.Abs(\_size4);  xiii1 = Math.Abs(\_size3);  yiii1 = \_size2;  xiii2 = \_size1;  yiii2 = Math.Abs(\_size4);  ShapeFactory shapeFactory = new ShapeFactory();  Shape c = shapeFactory.GetShape("triangle");  c.set(texturestyle, bb, paintcolor, xi1, yi1, xi2, yi2, xii1, yii1, xii2, yii2, xiii1, yiii1, xiii2, yiii2);  //===============================  c.Draw(g);  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  }  // ----------------TRIANGLE---------------------- -//  else if (matchT.Success)  {  try  {  g = Pnl\_Draw.CreateGraphics();  \_size1 = int.Parse(lbl\_StartPosX.Text);  \_size2 = int.Parse(lbl\_StartPosY.Text);  \_size3 = int.Parse(matchT.Groups[1].Value);  \_size4 = int.Parse(matchT.Groups[3].Value);  \_size5 = int.Parse(matchT.Groups[5].Value);  xi1 = \_size1;  yi1 = \_size2;  xi2 = Math.Abs(\_size3);  yi2 = \_size2;  xii1 = \_size1;  yii1 = \_size2;  xii2 = \_size1;  yii2 = Math.Abs(\_size4);  xiii1 = Math.Abs(\_size3);  yiii1 = \_size2;  xiii2 = \_size1;  yiii2 = Math.Abs(\_size4);  ShapeFactory shapeFactory = new ShapeFactory();  Shape c = shapeFactory.GetShape("triangle"); //new rectangles();  c.set(texturestyle, bb, paintcolor, xi1, yi1, xi2, yi2, xii1, yii1, xii2, yii2, xiii1, yiii1, xiii2, yiii2);  c.Draw(g);  }  catch (Exception ex)  {  MessageBox.Show("Error! Parameter should be in this form\"triangle side, side, side\"");  }  }  // ----------------CLEAR------------------------//  else if (matchClear.Success)  {  Pnl\_Draw.Refresh();  this.Pnl\_Draw.BackgroundImage = null;  }  // ----------------RESET------------------------//  else if (matchReset.Success)  {  \_size1 = 0;  \_size2 = 0;  lbl\_StartPosX.Text = \_size1.ToString();  lbl\_StartPosY.Text = \_size2.ToString();  }  // ----------------MOVETO------------------------//  else if (matchMT.Success)  {  try  {  \_size1 = int.Parse(matchMT.Groups[1].Value);  \_size2 = int.Parse(matchMT.Groups[3].Value);  lbl\_StartPosX.Text = \_size1.ToString();  lbl\_StartPosY.Text = \_size2.ToString();  }  catch (Exception ex)  {  MessageBox.Show(ex.Message);  }  }  }  else  {  MessageBox.Show("Invalid Syntax!!");  }  }  }  } |
| Circle | using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FinalAssignment  {  class Circle : Shape  {  /// <summary>  /// Getting value for ellipse  /// </summary>  int x, y, size, size1;  Color c1;  int texturestyle;  Brush bb;  public override void Draw(Graphics g)  {  Pen p = new Pen(c1, 5);  if(texturestyle == 0)  {  g.DrawEllipse(p, x, y, size, size1);  }  else  {  g.FillEllipse(bb, x, y, size, size1);  }  }  /// <summary>  /// Setting the required parameter for circle  /// </summary>  /// <param name="texturestyle">Define the texture</param>  /// <param name="bb">type of brush</param>  /// <param name="c1">color </param>  /// <param name="list">list of parameter that will be passed inside this function</param>  public override void set(int texturestyle, Brush bb, Color c1, params int[] list)  {  this.texturestyle = texturestyle;  this.bb = bb;  this.c1 = c1;  this.x = list[0];  this.y = list[1];  this.size = list[2];  this.size1 = list[3];  }  }  } |
| Triangle | using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FinalAssignment  {  public class Triangle : Shape  {  /// <summary>  /// Triangle side  /// </summary>  public int xi1, yi1, xi2, yi2, xii1, yii1, xii2, yii2, xiii1, yiii1, xiii2, yiii2;  Color c1;  int texturestyle;  Brush bb;  public override void Draw(Graphics g)  {  Pen p = new Pen(c1, 5);  //----------------------------------------------------------------------------------------------------------------------  g.DrawLine(p, xi1, yi1, xi2, yi2);  g.DrawLine(p, xii1, yii1, xii2, yii2);  g.DrawLine(p, xiii1, yiii1, xiii2, yiii2);  //---------------------------------------------------------------------------------------------------------------------  }  /// <summary>  /// geting value of coordinates for drawing triangle  /// </summary>  /// <param name="texturestyle"></param>  /// <param name="bb"></param>  /// <param name="c1"></param>  /// <param name="list"></param>  public override void set(int texturestyle, Brush bb, Color c1, params int[] list)  {  //\_size1, \_size2, xi1,yi1,xi2,yi2,xii1,yii1,xii2,yii2,xiii1,yiii1,xiii2,yiii2  this.texturestyle = texturestyle;  this.bb = bb;  this.c1 = c1;  this.xi1 = list[0];  this.yi1 = list[1];  this.xi2 = list[2];  this.yi2 = list[3];  this.xii1 = list[4];  this.yii1 = list[5];  this.xii2 = list[6];  this.yii2 = list[7];  this.xiii1 = list[8];  this.yiii1 = list[9];  this.xiii2 = list[10];  this.yiii2 = list[11];  }  }  } |
| Rectangle | using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FinalAssignment  {  public class Rectangle : Shape  {  public int x, y, size, size1;  public int texturestyle;    public Brush bb;  public Color c1;  /// <summary>  /// this function draws rectangle  /// </summary>  /// <param name="g"></param>  public override void Draw(Graphics g)  {  Pen p = new Pen(c1, 5);  if (texturestyle == 0)  {  g.DrawRectangle(p, x, y, size, size1);  }  else  {  g.FillRectangle(bb, x, y, size, size1);  }  }  /// <summary>  /// Setting the required properties for rectangle  /// </summary>  /// <param name="texturestyle"></param>  /// <param name="bb"></param>  /// <param name="c1"></param>  /// <param name="list"></param>  public override void set(int texturestyle, Brush bb, Color c1, params int[] list)  {  this.texturestyle = texturestyle;  this.bb = bb;  this.c1 = c1;  this.x = list[0];  this.y = list[1];  this.size = list[2];  this.size1 = list[3];  }  }  } |
| Shape | using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FinalAssignment  {  public abstract class Shape  {  /// <summary>  /// Passing Graphic Value  /// </summary>  /// <param name="g"></param>  public abstract void Draw(Graphics g);  /// <summary>  /// passing the value from button click of form to the shape  /// </summary>  /// <param name="texturestyle">define texture</param>  /// <param name="bb">define properties of brush</param>  /// <param name="c">define color</param>  /// <param name="list">list of parameter</param>  public abstract void set(int texturestyle, Brush bb, Color c, params int[] list);  }  } |
| Shape Factory | using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace FinalAssignment  {  class ShapeFactory  {  /// <summary>  /// checking the type of shape  /// </summary>  /// <param name="shapeType"></param>  /// <returns>return shape object </returns>  public Shape GetShape(string shapeType)  {  if (shapeType == "circle")  {  return new Circle();  }  else if (shapeType == "rectangle")  {  return new Rectangle();  }    else if (shapeType == "triangle")  {  return new Triangle();  }  return null;  }  }  } |
| Validation | using System;  using System.Linq;  using System.Text.RegularExpressions;  using System.Windows.Forms;  namespace Paint  {  /// <summary>  /// class to validate user input  /// </summary>  public class Validation  {  private TextBox txtCommand;  public Boolean isValidCommand = true;  public Boolean isSomethingInvalid = false;  public int Raduis = 0;  public int Width = 0;  public int Height = 0;  public int counter = 0;  public int LoopCounter = 0;  public int lineNumber = 0;  public Boolean hasLoop = false;  public Boolean hasEndLoop = false;  public Boolean hasIf = false;  public Boolean hasEndif = false;    public int loopLineNo = 0, endLoopLineNo = 0, ifLineNo = 0, endIfLineNo = 0;  public Validation(TextBox txtCommand)  {  this.txtCommand = txtCommand;  int numberOfLines = txtCommand.Lines.Length;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  checkLineValidation(oneLineCommand);  lineNumber = (i + 1);  if (!isValidCommand)  {  MessageBox.Show("Error in line " + lineNumber);  isValidCommand = true;  }  }  }  checkLoopAndIfValidation();  if (!isValidCommand)  {  isSomethingInvalid = true;  }  }  /// <summary>  /// function to check loop and if validation  /// </summary>  public void checkLoopAndIfValidation()  {  int numberOfLines = txtCommand.Lines.Length;  for (int i = 0; i < numberOfLines; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  hasLoop = Regex.IsMatch(oneLineCommand.ToLower(), @"\bloop\b");  if (hasLoop)  {  loopLineNo = (i + 1);  }  hasEndLoop = oneLineCommand.ToLower().Contains("endloop");  if (hasEndLoop)  {  endLoopLineNo = (i + 1);  }  hasIf = Regex.IsMatch(oneLineCommand.ToLower(), @"\bif\b");  if (hasIf)  {  ifLineNo = (i + 1);  }  hasEndif = oneLineCommand.ToLower().Contains("endif");  if (hasEndif)  {  endIfLineNo = (i + 1);  }  }  }  if (loopLineNo > 0)  {  hasLoop = true;  }  if (endLoopLineNo > 0)  {  hasEndLoop = true;  }  if (ifLineNo > 0)  {  hasIf = true;  }  if (endIfLineNo > 0)  {  hasEndif = true;  }  if (hasLoop)  {  if (hasEndLoop)  {  if (loopLineNo < endLoopLineNo)  {  }  else  {  isValidCommand = false;  MessageBox.Show("'ENDLOOP' must be after loop start");  }  }  else  {  isValidCommand = false;  MessageBox.Show("Loop Not Ended with 'ENDLOOP'");  }  }  if (hasIf)  {  if (hasEndif)  {  if (ifLineNo < endIfLineNo)  {  }  else  {  isValidCommand = false;  MessageBox.Show("'ENDIF' must be after IF");  }  }  else  {  isValidCommand = false;  MessageBox.Show("IF Not Ended with 'ENDIF'");  }  }  }  public void checkLineValidation(string lineOfCommand)  {  String[] keyword = { "circle", "rectangle", "triangle", "polygon", "drawto", "moveto", "repeat", "if", "endif", "loop", "endloop" };  String[] shapes = { "circle", "rectangle", "triangle", "polygon" };  String[] variable = { "radius", "width", "height", "counter", "size" };  lineOfCommand = Regex.Replace(lineOfCommand, @"\s+", " ");  string[] words = lineOfCommand.Split(' ');  //removing white spaces in between words  for (int i = 0; i < words.Length; i++)  {  words[i] = words[i].Trim();  }  String firstWord = words[0].ToLower();  Boolean firstWordIsKeyword = keyword.Contains(firstWord);  if (firstWordIsKeyword)  {  Boolean firstWordIsShape = shapes.Contains(words[0].ToLower());  if (firstWordIsShape)  {  if (words[0].ToLower().Equals("circle"))  {  if (words.Length == 2)  {  Boolean isInt = words[1].All(char.IsDigit);  if (!isInt)  {  //if it isnot variable then invalid  Boolean isVariable = variable.Contains(words[1].ToLower());  if (isVariable)  {  checkIfVariableDefined(words[1]);  }  else  {  isValidCommand = false;  }  //throw new NonDigitValueException("The value is not numerical \r\n It is not an error but just showing custom made exception.");  }  }  else  {  isValidCommand = false;  }  }  else if (words[0].ToLower().Equals("rectangle"))  {  String args = lineOfCommand.Substring(9, (lineOfCommand.Length - 9));  String[] parms = args.Split(',');  if (parms.Length == 2)  {  Boolean isInt = false;  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  isInt = parms[i].All(char.IsDigit);  if (!isInt)  {  //if it isnot variable then invalid  Boolean isVariable = variable.Contains(parms[i].ToLower());  if (!isVariable)  {  isValidCommand = false;  }  }  }  }  else  {  isValidCommand = false;  }  }  else if (words[0].ToLower().Equals("triangle"))  {  String args = lineOfCommand.Substring(8, (lineOfCommand.Length - 8));  String[] parms = args.Split(',');  if (parms.Length == 3)  {  Boolean isInt = false;  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  isInt = parms[i].All(char.IsDigit);  if (!isInt)  {  isValidCommand = false;  }  }  }  else  {  isValidCommand = false;  }  }  else if (words[0].ToLower().Equals("polygon"))  {  String args = lineOfCommand.Substring(7, (lineOfCommand.Length - 7));  String[] parms = args.Split(',');  if (parms.Length == 8 || parms.Length == 10)  {  Boolean isInt = false;  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  isInt = parms[i].All(char.IsDigit);  if (!isInt)  {  isValidCommand = false;  }  }  }  else  {  isValidCommand = false;  }  }  else { }  }  else if (firstWord.Equals("loop"))  {  if (words.Length == 2)  {  Boolean isInt = words[1].All(char.IsDigit);  if (!isInt)  {  isValidCommand = false;  }  }  else  {  isValidCommand = false;  }  }  else if (firstWord.Equals("end"))  {  if (words.Length == 2)  {  if (!words[1].Equals("loop"))  {  isValidCommand = false;  }  }  else  {  isValidCommand = false;  }  }  else if (firstWord.Equals("repeat"))  {  if (words.Length >= 4 && words.Length <= 6)  {  Boolean isInt = words[1].All(char.IsDigit);  if (isInt)  {  if (shapes.Contains(words[2].ToLower()))  {  Boolean hasPlus = words[3].Contains('+');  if (hasPlus)  {  string[] words2 = words[3].Split('+');  for (int i = 0; i < words2.Length; i++)  {  words2[i] = words2[i].Trim();  }  Boolean firstWordIsVariable = variable.Contains(words2[0].ToLower());  if (!firstWordIsVariable)  {  isValidCommand = false;  }  else  {  if (words2.Length != 2)  {  isValidCommand = false;  }  else  {  //third char should be int to be valid  Boolean isInt2 = words2[1].All(char.IsDigit);  if (!isInt2)  {  isValidCommand = false;  }  }  }  }  else  {  if (variable.Contains(words[3].ToLower()))  {  if (words[4].Trim().Equals("+"))  {  Boolean isInt3 = words[5].All(char.IsDigit);  if (!isInt3)  {  isValidCommand = false;  }  }  else  {  Boolean hasPlus2 = words[4].Contains('+');  if (hasPlus2)  {  string[] words2 = words[4].Split('+');  for (int i = 0; i < words2.Length; i++)  {  words2[i] = words2[i].Trim();  }  if (words2.Length == 2)  {  Boolean isInt2 = words2[1].All(char.IsDigit);  if (!isInt2)  {  isValidCommand = false;  }  }  else  {  isValidCommand = false;  }  }  }  }  }  }  }  }  else  {  isValidCommand = false;  }  }  else if (firstWord.Equals("if"))  {  if (words.Length == 5)  {  if (variable.Contains(words[1].ToLower()))  {  if (words[2].Equals("="))  {  Boolean isInt = words[3].All(char.IsDigit);  if (isInt)  {  if (words[4].ToLower().Equals("then"))  {  }  else { isValidCommand = false; }  }  else { isValidCommand = false; }  }  else { isValidCommand = false; }  }  else { isValidCommand = false; }  }  else  {  isValidCommand = false;  }  }  else if (firstWord.Equals("endif"))  {  if (words.Length != 1)  {  isValidCommand = false;  }  }  else if (firstWord.Equals("drawto") || firstWord.Equals("moveto"))  {  String args = lineOfCommand.Substring(6, (lineOfCommand.Length - 6));  String[] parms = args.Split(',');  if (parms.Length == 2)  {  Boolean isInt = false;  for (int i = 0; i < parms.Length; i++)  {  parms[i] = parms[i].Trim();  isInt = parms[i].All(char.IsDigit);  if (!isInt)  {  isValidCommand = false;  }  }  }  else  {  isValidCommand = false;  }  }  }  else  {  Boolean hasPlus = lineOfCommand.Contains('+');  Boolean hasEquals = lineOfCommand.Contains("=");  if (!hasEquals && !hasPlus)  {  isValidCommand = false;  }  else  {  if (hasEquals)  {  string[] words2 = lineOfCommand.Split('=');  for (int i = 0; i < words2.Length; i++)  {  words2[i] = words2[i].Trim();  }  Boolean firstWordIsVariable = variable.Contains(words2[0].ToLower());  if (!firstWordIsVariable)  {  isValidCommand = false;  }  else  {  if (words2.Length != 2)  {  isValidCommand = false;  }  else  {  //third char should be int to be valid  Boolean isInt = words2[1].All(char.IsDigit);  if (!isInt)  {  isValidCommand = false;  }  }  }  }  if (hasPlus)  {  string[] words2 = lineOfCommand.Split('+');  for (int i = 0; i < words2.Length; i++)  {  words2[i] = words2[i].Trim();  }  Boolean firstWordIsVariable = variable.Contains(words2[0].ToLower());  if (!firstWordIsVariable)  {  isValidCommand = false;  }  else  {  if (words2.Length != 2)  {  isValidCommand = false;  }  else  {  //third char should be int to be valid  Boolean isInt = words2[1].All(char.IsDigit);  if (!isInt)  {  isValidCommand = false;  }  }  }  }  }  }  if (!isValidCommand)  {  isSomethingInvalid = true;  }  }  /// <summary>  /// to check if the variable is defined or not  /// </summary>  /// <param name="variable"></param>  public void checkIfVariableDefined(string variable)  {  Boolean isVaraibleFound = false;  if (txtCommand.Lines.Length > 1)  {  if (lineNumber > 0)  {  for (int i = 0; i < lineNumber; i++)  {  String oneLineCommand = txtCommand.Lines[i];  oneLineCommand = oneLineCommand.Trim();  if (!oneLineCommand.Equals(""))  {  Boolean isVariableDefined = oneLineCommand.ToLower().Contains(variable.ToLower());  if (isVariableDefined)  {  isVaraibleFound = true;  }  }  }  if (!isVaraibleFound)  {  MessageBox.Show("Variable is not defined");  isValidCommand = false;  }  }  else  {  MessageBox.Show("Variable is not defined");  isValidCommand = false;  }  }  else  {  MessageBox.Show("Varaible is not defined");  isValidCommand = false;  }  }  }  } |