**OOPDraw  
Learn the principles of OOP by writing a simple drawing program**

TEACHER’S GUIDE

Created by Richard Pawson

[Creative Commons License](https://creativecommons.org/licenses/by-sa/4.0/)  
This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

[Complete code after Exercise 1 4](#_Toc506562228)

[Complete code after Exercise 2 5](#_Toc506562229)

[Form1.cs 5](#_Toc506562230)

[Form1.Designer.cs 6](#_Toc506562231)

[Project view 7](#_Toc506562232)

[Complete code after Exercise 3 8](#_Toc506562233)

[Rectangle.cs 8](#_Toc506562234)

[EquliateralTriangle.cs 8](#_Toc506562235)

[Form1.cs 8](#_Toc506562236)

[Form1.Designer.cs 9](#_Toc506562237)

[Complete code after Exercise 4 10](#_Toc506562238)

[Rectangle.cs 10](#_Toc506562239)

[EquilateralTriangle.cs 10](#_Toc506562240)

[Form1.cs 11](#_Toc506562241)

[Form1.Designer.cs 11](#_Toc506562242)

[Complete code for Exercise 5 12](#_Toc506562243)

[Shape.cs 12](#_Toc506562244)

[Rectangle.cs 12](#_Toc506562245)

[EquilateralTriangle.cs 12](#_Toc506562246)

[Form1.cs 13](#_Toc506562247)

[Form1.Designer.cs 14](#_Toc506562248)

[Complete code after Exercise 6 16](#_Toc506562249)

[Shape.cs 16](#_Toc506562250)

[Rectangle.cs 17](#_Toc506562251)

[EquilateralTriangle.cs 17](#_Toc506562252)

[Complete code after Exercise 7 18](#_Toc506562253)

[Shape.cs 18](#_Toc506562254)

[Rectangle.cs 18](#_Toc506562255)

[EquilateralTriangle.cs 18](#_Toc506562256)

[Form1.cs 19](#_Toc506562257)

[Form1.Designer.cs 20](#_Toc506562258)

[Complete code for Exercise 8 22](#_Toc506562259)

[Shape.cs 22](#_Toc506562260)

[Rectangle.cs 22](#_Toc506562261)

[EquilateralTriangle.cs 23](#_Toc506562262)

[Form1.cs 24](#_Toc506562263)

[Form1.Designer.cs 25](#_Toc506562264)

[Complete code after Exercise 9 27](#_Toc506562265)

[Project structure 27](#_Toc506562266)

[Shape.cs 28](#_Toc506562267)

[Rectangle.cs 28](#_Toc506562268)

[EquilateralTriangle.cs 29](#_Toc506562269)

[House.cs 30](#_Toc506562270)

[Form1.cs 30](#_Toc506562271)

[Form1.Designer.cs 32](#_Toc506562272)

[Program.cs 33](#_Toc506562273)

# Complete code after Exercise 1

# Complete code after Exercise 2

## Form1.cs

using Nakov.TurtleGraphics;

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string) comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle")

{

DrawTriangle(turtleX, turtleY, 50);

}

else if (selectedItem == "Draw Rectangle")

{

DrawRectangle(turtleX, turtleY, 50, 100);

}

}

private static void DrawTriangle(float xOrigin, float yOrigin, float sideLength)

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = xOrigin;

Turtle.Y = yOrigin;

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(sideLength);

Turtle.Rotate(120);

}

}

private static void DrawRectangle(float xOrigin, float yOrigin, float height, float width)

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = xOrigin;

Turtle.Y = yOrigin;

for (int i = 0; i < 2; i++)

{

Turtle.Forward(height);

Turtle.Rotate(90);

Turtle.Forward(width);

Turtle.Rotate(90);

}

}

}

}

## Form1.Designer.cs

namespace OOPDraw

{

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.comboBox1 = new System.Windows.Forms.ComboBox();

this.SuspendLayout();

//

// comboBox1

//

this.comboBox1.Items.AddRange(new object[] {

"Draw Triangle",

"Draw Rectangle"});

this.comboBox1.Location = new System.Drawing.Point(13, 13);

this.comboBox1.Name = "comboBox1";

this.comboBox1.Size = new System.Drawing.Size(121, 21);

this.comboBox1.TabIndex = 0;

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(385, 323);

this.Controls.Add(this.comboBox1);

this.Name = "Form1";

this.Text = "Form1";

this.MouseClick += new System.Windows.Forms.MouseEventHandler(this.Form1\_MouseClick);

this.ResumeLayout(false);

}

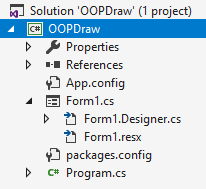
#endregion

private System.Windows.Forms.ComboBox comboBox1;

}

}

## Project view



# Complete code after Exercise 3

## Rectangle.cs

namespace OOPDraw

{

public class Rectangle

{

//Properties

public float XOrigin { get; set; }

public float YOrigin { get; set; }

public float Width { get; set; }

public float Height { get; set; }

//The 'Constructor'

public Rectangle(float xOrigin, float yOrigin, float width, float height)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

Width = width;

Height = height;

}

}

}

## EquliateralTriangle.cs

namespace OOPDraw

{

public class EquilateralTriangle

{

//Properties

public float XOrigin { get; set; }

public float YOrigin { get; set; }

public float SideLength { get; set; }

//The 'Constructor'

public EquilateralTriangle(float xOrigin, float yOrigin, float sideLength)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

SideLength = sideLength;

}

}

}

## Form1.cs

using Nakov.TurtleGraphics;

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string) comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle") //We will add more options later

{

var tri = new EquilateralTriangle(turtleX, turtleY, 50);

DrawTriangle(tri);

}

else if (selectedItem == "Draw Rectangle")

{

var rec = new Rectangle(turtleX, turtleY, 100, 50);

DrawRectangle(rec);

}

}

private static void DrawRectangle(Rectangle rec)

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = rec.XOrigin;

Turtle.Y = rec.YOrigin;

for (int i = 0; i < 2; i++)

{

Turtle.Forward(rec.Height);

Turtle.Rotate(90);

Turtle.Forward(rec.Width);

Turtle.Rotate(90);

}

}

private static void DrawTriangle(EquilateralTriangle tri)

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = tri.XOrigin;

Turtle.Y = tri.YOrigin;

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(tri.SideLength);

Turtle.Rotate(120);

}

}

}

}

## Form1.Designer.cs

Unchanged from Exercise 3

# Complete code after Exercise 4

## Rectangle.cs

using Nakov.TurtleGraphics;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OOPDraw

{

public class Rectangle

{

//Properties

private float XOrigin { get; set; }

private float YOrigin { get; set; }

private float Width { get; set; }

private float Height { get; set; }

//The 'Constructor'

public Rectangle(float xOrigin, float yOrigin, float width, float height)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

Width = width;

Height = height;

}

public void Draw()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

for (int i = 0; i < 2; i++)

{

Turtle.Forward(Height);

Turtle.Rotate(90);

Turtle.Forward(Width);

Turtle.Rotate(90);

}

}

}

}

## EquilateralTriangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class EquilateralTriangle

{

//Properties

private float XOrigin { get; set; }

private float YOrigin { get; set; }

private float SideLength { get; set; }

//The 'Constructor'

public EquilateralTriangle(float xOrigin, float yOrigin, float sideLength)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

SideLength = sideLength;

}

public void Draw()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

}

}

## Form1.cs

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string) comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle") //We will add more options later

{

var tri = new EquilateralTriangle(turtleX, turtleY, 50);

tri.Draw(); ;

}

else if (selectedItem == "Draw Rectangle")

{

var rec = new Rectangle(turtleX, turtleY, 100, 50);

rec.Draw();

}

}

}

}

## Form1.Designer.cs

Unchanged from Exercise 3

# Complete code for Exercise 5

## Shape.cs

namespace OOPDraw

{

public interface Shape

{

void Draw();

void MoveTo(float x, float y);

}

}

## Rectangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class Rectangle : Shape

{

//Properties

private float XOrigin { get; set; }

private float YOrigin { get; set; }

private float Width { get; set; }

private float Height { get; set; }

//The 'Constructor'

public Rectangle(float xOrigin, float yOrigin, float width, float height)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

Width = width;

Height = height;

}

public void Draw()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

for (int i = 0; i < 2; i++)

{

Turtle.Forward(Height);

Turtle.Rotate(90);

Turtle.Forward(Width);

Turtle.Rotate(90);

}

}

public void MoveTo(float x, float y)

{

XOrigin = x;

YOrigin = y;

}

}

}

## EquilateralTriangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class EquilateralTriangle : Shape

{

//Properties

private float XOrigin { get; set; }

private float YOrigin { get; set; }

private float SideLength { get; set; }

//The 'Constructor'

public EquilateralTriangle(float xOrigin, float yOrigin, float sideLength)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

SideLength = sideLength;

}

public void Draw()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

public void MoveTo(float x, float y)

{

XOrigin = x;

YOrigin = y;

}

}

}

## Form1.cs

using Nakov.TurtleGraphics;

using System.Collections.Generic;

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private List<Shape> shapes = new List<Shape>();

private Shape mostRecent;

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string) comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle") //We will add more options later

{

var tri = new EquilateralTriangle(turtleX, turtleY, 50);

shapes.Add(tri);

mostRecent = tri;

}

else if (selectedItem == "Draw Rectangle")

{

var rec = new Rectangle(turtleX, turtleY, 100, 50);

shapes.Add(rec);

mostRecent = rec;

}

else if (selectedItem == "Move Shape")

{

mostRecent.MoveTo(turtleX, turtleY);

}

Turtle.Dispose(); //Clear all Turtle tracks

DrawAll();

}

public void DrawAll()

{

foreach (var shape in shapes)

{

shape.Draw();

}

}

}

}

## Form1.Designer.cs

namespace OOPDraw

{

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.comboBox1 = new System.Windows.Forms.ComboBox();

this.SuspendLayout();

//

// comboBox1

//

this.comboBox1.Items.AddRange(new object[] {

"Draw Triangle",

"Draw Rectangle",

"Move Shape"});

this.comboBox1.Location = new System.Drawing.Point(13, 13);

this.comboBox1.Name = "comboBox1";

this.comboBox1.Size = new System.Drawing.Size(121, 21);

this.comboBox1.TabIndex = 0;

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(385, 323);

this.Controls.Add(this.comboBox1);

this.Name = "Form1";

this.Text = "Form1";

this.MouseClick += new System.Windows.Forms.MouseEventHandler(this.Form1\_MouseClick);

this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.ComboBox comboBox1;

}

}

# Complete code after Exercise 6

## Shape.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public abstract class Shape

{

protected float XOrigin { get; set; }

protected float YOrigin { get; set; }

//The 'Constructor'

public Shape(float xOrigin, float yOrigin)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

}

//Abstract methods

public abstract void Draw();

//Concrete methods

public void MoveTo(float x, float y)

{

XOrigin = x;

YOrigin = y;

}

protected void ResetTurtle()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = 2;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

}

}

}

## Rectangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class Rectangle : Shape

{

//Properties

private float Width { get; set; }

private float Height { get; set; }

//The 'Constructor'

public Rectangle(float xOrigin, float yOrigin, float width, float height) : base(xOrigin, yOrigin)

{

Width = width;

Height = height;

}

public override void Draw()

{

ResetTurtle();

for (int i = 0; i < 2; i++)

{

Turtle.Forward(Height);

Turtle.Rotate(90);

Turtle.Forward(Width);

Turtle.Rotate(90);

}

}

}

}

## EquilateralTriangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class EquilateralTriangle : Shape

{

//Properties

private float SideLength { get; set; }

//The 'Constructor'

public EquilateralTriangle(float xOrigin, float yOrigin, float sideLength) : base(xOrigin, yOrigin)

{

SideLength = sideLength;

}

public override void Draw()

{

ResetTurtle();

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

}

}

All other code unchanged from previous exercise.

# Complete code after Exercise 7

## Shape.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public abstract class Shape

{

protected float XOrigin { get; set; }

protected float YOrigin { get; set; }

private float LineWidth { get; set; }

//The 'Constructor'

public Shape(float xOrigin, float yOrigin)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

}

//Abstract methods

public abstract void Draw();

//Concrete methods

public void MoveTo(float x, float y)

{

XOrigin = x;

YOrigin = y;

}

public void Select()

{

LineWidth = 4;

}

public void Unselect()

{

LineWidth = 2;

}

protected void ResetTurtle()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = LineWidth;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

}

}

}

## Rectangle.cs

Unchanged from previous exercise

## EquilateralTriangle.cs

Unchanged from previous exercise

## Form1.cs

using Nakov.TurtleGraphics;

using System.Collections.Generic;

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private List<Shape> shapes = new List<Shape>();

private Shape mostRecent;

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string)comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle") //We will add more options later

{

AddShape(new EquilateralTriangle(turtleX, turtleY, 50));

}

else if (selectedItem == "Draw Rectangle")

{

AddShape(new Rectangle(turtleX, turtleY, 100, 50));

}

else if (selectedItem == "Move Shape")

{

ActiveShape().MoveTo(turtleX, turtleY);

}

DrawAll();

}

private void AddShape(Shape shape)

{

if (shapes.Count > 0) //i.e. this isn't the first shape

{

ActiveShape().Unselect();

}

shapes.Add(shape);

activeShapeNumber = shapes.Count - 1; //i.e. the shape just added

ActiveShape().Select();

}

public void DrawAll()

{

Turtle.Dispose(); //First clear all Turtle tracks to start afresh

foreach (var shape in shapes)

{

shape.Draw();

}

}

private int activeShapeNumber = 0;

private Shape ActiveShape()

{

return shapes[activeShapeNumber]; //List elements can be accessed like an array

}

private void Next\_Click(object sender, System.EventArgs e)

{

ActiveShape().Unselect();

activeShapeNumber = activeShapeNumber + 1;

if (activeShapeNumber >= shapes.Count) activeShapeNumber = 0;

ActiveShape().Select();

DrawAll();

}

private void Prev\_Click(object sender, System.EventArgs e)

{

ActiveShape().Unselect();

activeShapeNumber = activeShapeNumber - 1;

if (activeShapeNumber < 0) activeShapeNumber = shapes.Count - 1;

ActiveShape().Select();

DrawAll();

}

}

}

## Form1.Designer.cs

namespace OOPDraw

{

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.comboBox1 = new System.Windows.Forms.ComboBox();

this.Next = new System.Windows.Forms.Button();

this.Prev = new System.Windows.Forms.Button();

this.SuspendLayout();

//

// comboBox1

//

this.comboBox1.Items.AddRange(new object[] {

"Draw Triangle",

"Draw Rectangle",

"Move Shape"});

this.comboBox1.Location = new System.Drawing.Point(13, 13);

this.comboBox1.Name = "comboBox1";

this.comboBox1.Size = new System.Drawing.Size(121, 21);

this.comboBox1.TabIndex = 0;

//

// Next

//

this.Next.Location = new System.Drawing.Point(13, 51);

this.Next.Name = "Next";

this.Next.Size = new System.Drawing.Size(48, 22);

this.Next.TabIndex = 1;

this.Next.Text = "Next";

this.Next.UseVisualStyleBackColor = true;

this.Next.Click += new System.EventHandler(this.Next\_Click);

//

// Prev

//

this.Prev.Location = new System.Drawing.Point(79, 51);

this.Prev.Name = "Prev";

this.Prev.Size = new System.Drawing.Size(55, 22);

this.Prev.TabIndex = 2;

this.Prev.Text = "Prev";

this.Prev.UseVisualStyleBackColor = true;

this.Prev.Click += new System.EventHandler(this.Prev\_Click);

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(385, 323);

this.Controls.Add(this.Prev);

this.Controls.Add(this.Next);

this.Controls.Add(this.comboBox1);

this.Name = "Form1";

this.Text = "Form1";

this.MouseClick += new System.Windows.Forms.MouseEventHandler(this.Form1\_MouseClick);

this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.ComboBox comboBox1;

private System.Windows.Forms.Button Next;

private System.Windows.Forms.Button Prev;

}

}

# Complete code for Exercise 8

## Shape.cs

using System;

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public abstract class Shape

{

protected float XOrigin { get; set; }

protected float YOrigin { get; set; }

private float LineWidth { get; set; }

//The 'Constructor'

public Shape(float xOrigin, float yOrigin)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

}

//Abstract methods

public abstract void Draw();

//Concrete methods

public void MoveTo(float x, float y)

{

XOrigin = x;

YOrigin = y;

}

public void ResizeAbsolute(float turtleX, float turtleY)

{

Resize(Math.Abs(turtleX - XOrigin), Math.Abs(turtleY - YOrigin));

}

public abstract void Resize(float x, float y);

public void Select()

{

LineWidth = 4;

}

public void Unselect()

{

LineWidth = 2;

}

protected void ResetTurtle()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = LineWidth;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

}

}

}

## Rectangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class Rectangle : Shape

{

//Properties

private float Width { get; set; }

private float Height { get; set; }

//The 'Constructor'

public Rectangle(float xOrigin, float yOrigin, float width, float height) : base(xOrigin, yOrigin)

{

Width = width;

Height = height;

}

public override void Draw()

{

ResetTurtle();

for (int i = 0; i < 2; i++)

{

Turtle.Forward(Height);

Turtle.Rotate(90);

Turtle.Forward(Width);

Turtle.Rotate(90);

}

}

public override void Resize(float x, float y)

{

Width = x;

Height = y;

}

}

}

## EquilateralTriangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class EquilateralTriangle : Shape

{

//Properties

private float SideLength { get; set; }

//The 'Constructor'

public EquilateralTriangle(float xOrigin, float yOrigin, float sideLength) : base(xOrigin, yOrigin)

{

SideLength = sideLength;

}

public override void Draw()

{

ResetTurtle();

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

public override void Resize(float x, float y)

{

//Ignore Y

SideLength = x;

}

}

}

## Form1.cs

using Nakov.TurtleGraphics;

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private List<Shape> shapes = new List<Shape>();

private Shape mostRecent;

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string)comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle") //We will add more options later

{

AddShape(new EquilateralTriangle(turtleX, turtleY, 50));

}

else if (selectedItem == "Draw Rectangle")

{

AddShape(new Rectangle(turtleX, turtleY, 100, 50));

}

else if (selectedItem == "Move Shape")

{

ActiveShape().MoveTo(turtleX, turtleY);

}

else if (selectedItem == "Resize Shape")

{

ActiveShape().ResizeAbsolute(turtleX, turtleY);

}

DrawAll();

}

private void AddShape(Shape shape)

{

if (shapes.Count > 0) //i.e. this isn't the first shape

{

ActiveShape().Unselect();

}

shapes.Add(shape);

activeShapeNumber = shapes.Count - 1; //i.e. the shape just added

ActiveShape().Select();

}

public void DrawAll()

{

Turtle.Dispose(); //First clear all Turtle tracks to start afresh

foreach (var shape in shapes)

{

shape.Draw();

}

}

private int activeShapeNumber = 0;

private Shape ActiveShape()

{

return shapes[activeShapeNumber]; //List elements can be accessed like an array

}

private void Next\_Click(object sender, System.EventArgs e)

{

ActiveShape().Unselect();

activeShapeNumber = activeShapeNumber + 1;

if (activeShapeNumber >= shapes.Count) activeShapeNumber = 0;

ActiveShape().Select();

DrawAll();

}

private void Prev\_Click(object sender, System.EventArgs e)

{

ActiveShape().Unselect();

activeShapeNumber = activeShapeNumber - 1;

if (activeShapeNumber < 0) activeShapeNumber = shapes.Count - 1;

ActiveShape().Select();

DrawAll();

}

}

}

## Form1.Designer.cs

namespace OOPDraw

{

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.comboBox1 = new System.Windows.Forms.ComboBox();

this.Next = new System.Windows.Forms.Button();

this.Prev = new System.Windows.Forms.Button();

this.SuspendLayout();

//

// comboBox1

//

this.comboBox1.Items.AddRange(new object[] {

"Draw Triangle",

"Draw Rectangle",

"Move Shape",

"Resize Shape"});

this.comboBox1.Location = new System.Drawing.Point(13, 13);

this.comboBox1.Name = "comboBox1";

this.comboBox1.Size = new System.Drawing.Size(121, 21);

this.comboBox1.TabIndex = 0;

//

// Next

//

this.Next.Location = new System.Drawing.Point(13, 51);

this.Next.Name = "Next";

this.Next.Size = new System.Drawing.Size(48, 22);

this.Next.TabIndex = 1;

this.Next.Text = "Next";

this.Next.UseVisualStyleBackColor = true;

this.Next.Click += new System.EventHandler(this.Next\_Click);

//

// Prev

//

this.Prev.Location = new System.Drawing.Point(79, 51);

this.Prev.Name = "Prev";

this.Prev.Size = new System.Drawing.Size(55, 22);

this.Prev.TabIndex = 2;

this.Prev.Text = "Prev";

this.Prev.UseVisualStyleBackColor = true;

this.Prev.Click += new System.EventHandler(this.Prev\_Click);

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(385, 323);

this.Controls.Add(this.Prev);

this.Controls.Add(this.Next);

this.Controls.Add(this.comboBox1);

this.Name = "Form1";

this.Text = "Form1";

this.MouseClick += new System.Windows.Forms.MouseEventHandler(this.Form1\_MouseClick);

this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.ComboBox comboBox1;

private System.Windows.Forms.Button Next;

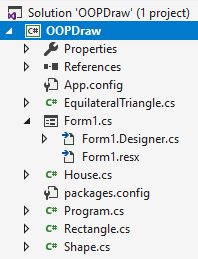
private System.Windows.Forms.Button Prev;

}

}

# Complete code after Exercise 9

## Project structure



## Shape.cs

using System;

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public abstract class Shape

{

protected float XOrigin { get; set; }

protected float YOrigin { get; set; }

private float LineWidth { get; set; }

//The 'Constructor'

public Shape(float xOrigin, float yOrigin)

{

XOrigin = xOrigin;

YOrigin = yOrigin;

}

//Abstract methods

public abstract void Draw();

//Concrete methods

public virtual void MoveTo(float x, float y)

{

XOrigin = x;

YOrigin = y;

}

public virtual void MoveBy(float x, float y)

{

XOrigin += x;

YOrigin += y;

}

public void ResizeAbsolute(float turtleX, float turtleY)

{

Resize(Math.Abs(turtleX - XOrigin), Math.Abs(turtleY - YOrigin));

}

public abstract void Resize(float x, float y);

public virtual void Select()

{

LineWidth = 4;

}

public virtual void Unselect()

{

LineWidth = 2;

}

protected void ResetTurtle()

{

Turtle.ShowTurtle = false;

Turtle.PenSize = LineWidth;

Turtle.Angle = 0; //Always start from North

Turtle.X = XOrigin;

Turtle.Y = YOrigin;

}

}

}

## Rectangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class Rectangle : Shape

{

//Properties

private float Width { get; set; }

private float Height { get; set; }

//The 'Constructor'

public Rectangle(float xOrigin, float yOrigin, float width, float height) : base(xOrigin, yOrigin)

{

Width = width;

Height = height;

}

public override void Draw()

{

ResetTurtle();

for (int i = 0; i < 2; i++)

{

Turtle.Forward(Height);

Turtle.Rotate(90);

Turtle.Forward(Width);

Turtle.Rotate(90);

}

}

public override void Resize(float x, float y)

{

Width = x;

Height = y;

}

}

}

## EquilateralTriangle.cs

using Nakov.TurtleGraphics;

namespace OOPDraw

{

public class EquilateralTriangle : Shape

{

//Properties

private float SideLength { get; set; }

public EquilateralTriangle(float xOrigin, float yOrigin, float sideLength) : base(xOrigin, yOrigin)

{

SideLength = sideLength;

}

public override void Draw()

{

ResetTurtle();

Turtle.Rotate(30);

for (int i = 0; i < 3; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

public override void Resize(float x, float y)

{

//Ignore Y

SideLength = x;

}

}

}

## House.cs

namespace OOPDraw

{

public class House : Shape

{

private float Width { get; set; }

private float WallHeight { get; set; }

private Rectangle Walls { get; set; }

private EquilateralTriangle Roof { get; set; }

public House(float originX, float originY, float width, float wallHeight) : base(originX, originY)

{

Width = width;

WallHeight = wallHeight;

Walls = new Rectangle(originX, originY, width, wallHeight);

Roof = new EquilateralTriangle(originX, originY + wallHeight, width);

}

public override void Draw()

{

Walls.Draw();

Roof.Draw();

}

public override void Resize(float x, float y)

{

Width = x;

var yDiff = y - WallHeight;

WallHeight = y;

Walls.Resize(x, y);

Roof.Resize(x, 0);

Roof.MoveBy(0, yDiff);

}

public override void Select()

{

Walls.Select();

Roof.Select();

}

public override void Unselect()

{

Walls.Unselect();

Roof.Unselect();

}

public override void MoveTo(float x, float y)

{

base.MoveTo(x, y);

Walls.MoveTo(x, y);

Roof.MoveTo(x, y + WallHeight);

}

}

}

## Form1.cs

using Nakov.TurtleGraphics;

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace OOPDraw

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private List<Shape> shapes = new List<Shape>();

private Shape mostRecent;

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

//Transform windows coordinates to Turtle coordinates

float turtleX = e.X - Width / 2 + 8;

float turtleY = Height / 2 - e.Y - 19;

string selectedItem = (string)comboBox1.SelectedItem;

if (selectedItem == "Draw Triangle") //We will add more options later

{

AddShape(new EquilateralTriangle(turtleX, turtleY, 50));

}

else if (selectedItem == "Draw Rectangle")

{

AddShape(new Rectangle(turtleX, turtleY, 100, 50));

}

else if (selectedItem == "Draw House")

{

AddShape(new House(turtleX, turtleY, 100, 50));

}

else if (selectedItem == "Move Shape")

{

ActiveShape().MoveTo(turtleX, turtleY);

}

else if (selectedItem == "Resize Shape")

{

ActiveShape().ResizeAbsolute(turtleX, turtleY);

}

DrawAll();

}

private void AddShape(Shape shape)

{

if (shapes.Count > 0) //i.e. this isn't the first shape

{

ActiveShape().Unselect();

}

shapes.Add(shape);

activeShapeNumber = shapes.Count - 1; //i.e. the shape just added

ActiveShape().Select();

}

public void DrawAll()

{

Turtle.Dispose(); //First clear all Turtle tracks to start afresh

foreach (var shape in shapes)

{

shape.Draw();

}

}

private int activeShapeNumber = 0;

private Shape ActiveShape()

{

return shapes[activeShapeNumber]; //List elements can be accessed like an array

}

private void Next\_Click(object sender, System.EventArgs e)

{

ActiveShape().Unselect();

activeShapeNumber = activeShapeNumber + 1;

if (activeShapeNumber >= shapes.Count) activeShapeNumber = 0;

ActiveShape().Select();

DrawAll();

}

private void Prev\_Click(object sender, System.EventArgs e)

{

ActiveShape().Unselect();

activeShapeNumber = activeShapeNumber - 1;

if (activeShapeNumber < 0) activeShapeNumber = shapes.Count - 1;

ActiveShape().Select();

DrawAll();

}

}

}

## Form1.Designer.cs

namespace OOPDraw

{

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.comboBox1 = new System.Windows.Forms.ComboBox();

this.Next = new System.Windows.Forms.Button();

this.Prev = new System.Windows.Forms.Button();

this.SuspendLayout();

//

// comboBox1

//

this.comboBox1.Items.AddRange(new object[] {

"Draw Triangle",

"Draw Rectangle",

"Draw House",

"Move Shape",

"Resize Shape"});

this.comboBox1.Location = new System.Drawing.Point(13, 13);

this.comboBox1.Name = "comboBox1";

this.comboBox1.Size = new System.Drawing.Size(121, 21);

this.comboBox1.TabIndex = 0;

//

// Next

//

this.Next.Location = new System.Drawing.Point(13, 51);

this.Next.Name = "Next";

this.Next.Size = new System.Drawing.Size(48, 22);

this.Next.TabIndex = 1;

this.Next.Text = "Next";

this.Next.UseVisualStyleBackColor = true;

this.Next.Click += new System.EventHandler(this.Next\_Click);

//

// Prev

//

this.Prev.Location = new System.Drawing.Point(79, 51);

this.Prev.Name = "Prev";

this.Prev.Size = new System.Drawing.Size(55, 22);

this.Prev.TabIndex = 2;

this.Prev.Text = "Prev";

this.Prev.UseVisualStyleBackColor = true;

this.Prev.Click += new System.EventHandler(this.Prev\_Click);

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(385, 323);

this.Controls.Add(this.Prev);

this.Controls.Add(this.Next);

this.Controls.Add(this.comboBox1);

this.Name = "Form1";

this.Text = "Form1";

this.MouseClick += new System.Windows.Forms.MouseEventHandler(this.Form1\_MouseClick);

this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.ComboBox comboBox1;

private System.Windows.Forms.Button Next;

private System.Windows.Forms.Button Prev;

}

}

## Program.cs

using System;

using System.Windows.Forms;

namespace OOPDraw

{

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());

}

}

}