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

TEACHER’S HANDBOOK

Created by Richard Pawson

[Introduction 3](#_Toc507581757)

[Model answers to Worksheet Questions 4](#_Toc507581758)

[Complete code at the end of Exercise 9 11](#_Toc507581759)

[Shape.cs 11](#_Toc507581760)

[Rectangle.cs 11](#_Toc507581761)

[EquilateralTriangle.cs 12](#_Toc507581762)

[House.cs 13](#_Toc507581763)

[Form1.cs 13](#_Toc507581764)

[Form1.Designer.cs 15](#_Toc507581765)

[Program.cs 17](#_Toc507581766)

[Project structure 17](#_Toc507581767)

# Introduction

OOPDraw is an extended exercise that will teach students the fundamental principles of object-oriented programming (OOP) including:

* Objects as custom data types
* Class and instance
* Objects encapsulating behaviour and state (methods and properties)
* Polymorphism, implemented initially via the use of interfaces
* Inheritance of implemented behaviour
* Abstract and concrete classes, and methods
* Information hiding
* Association, including both composition and aggregation
* Favouring delegation over inheritance

All of these principles are taught via the development of a single, substantial application - a simple drawing program. The program is developed incrementally, giving the student a sense of achievement at each stage.

Moreover, each of the principles is introduced as a solution to a need that has arisen - hopefully giving them a stronger understanding of the reason for the idea.

At each step the worksheet provides all, or almost all, of the code that will be required: the only code that the students need to write from scratch involves copying a coding pattern that has been provided into a slightly different context. While the most able students could probably figure out more of the code for themselves, the advantage of this storyboarded approach is that most students will be able to complete the exercise. They are likely to need live support, both to re-explain /reinforce some the principles and perhaps to help them identify the source of compile errors arising from not having copied the listed code correctly.

The more able students will finish sooner, but you can then encourage them to move straight on to the suggested optional extension exercises listed at the end (Exercise 10+) of the sheet.

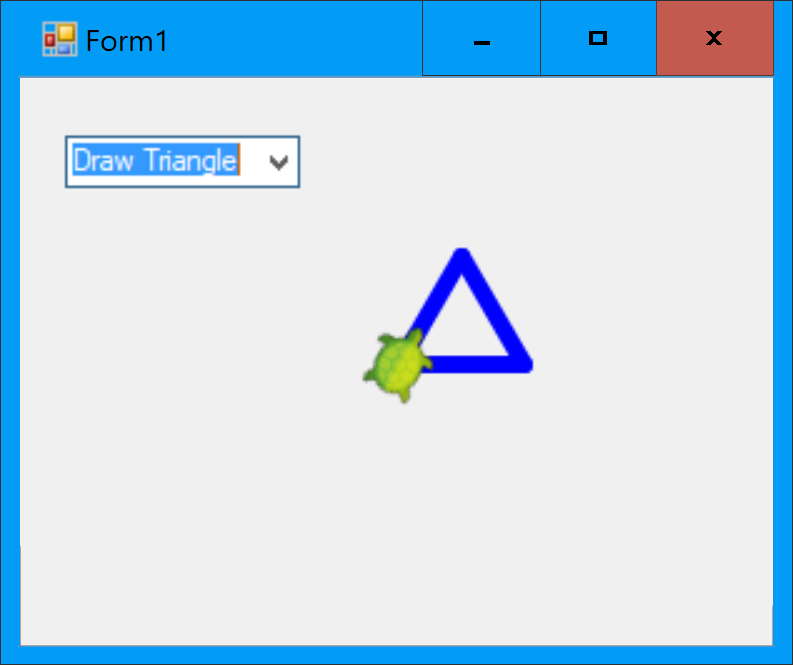
The choice of a Drawing program was deliberate. Apart from being a reasonably engaging idea (students enjoy using the program as well as writing it), it fits very well with the ideas behind OOP. This is not a coincidence: the very first drawing programs - at least, those using bit-mapped graphics - were developed by the same team that wrote the first pure OO Programming Language (SmallTalk) at Xerox PARC in the early 1970s. The team was called the Learning Research Group and although the ideas and techniques they developed would ultimately change the whole world of professional software development, their core interest lay in children’s learning development. Much of their inspiration came from the work of the great pedagogists such as Jean Piaget, Seymour Papert, and, particularly, Jerome Bruner. This worksheet was in large part inspired by the work of the LRG.

# Model answers to Worksheet Questions

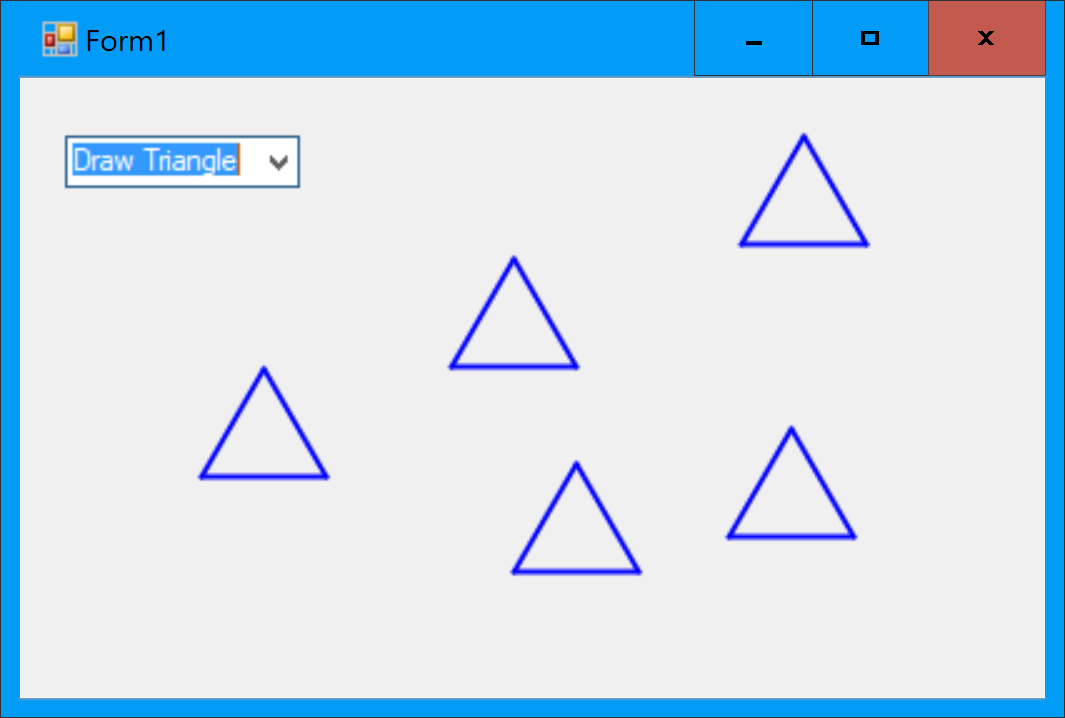
1. What is the value of SelectedItem now?

Null.

1. Paste in a screenshot of the resulting triangle



1. Paste in a screenshot showing several triangles drawn within the form.



1. Paste in your code for the DrawRectangle function

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);

}

}

1. Paste in your code for the modified Form1\_MouseClick function

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

{

DrawTriangle(turtleX, turtleY, 50);

}

else if (selectedItem == "Draw Rectangle")

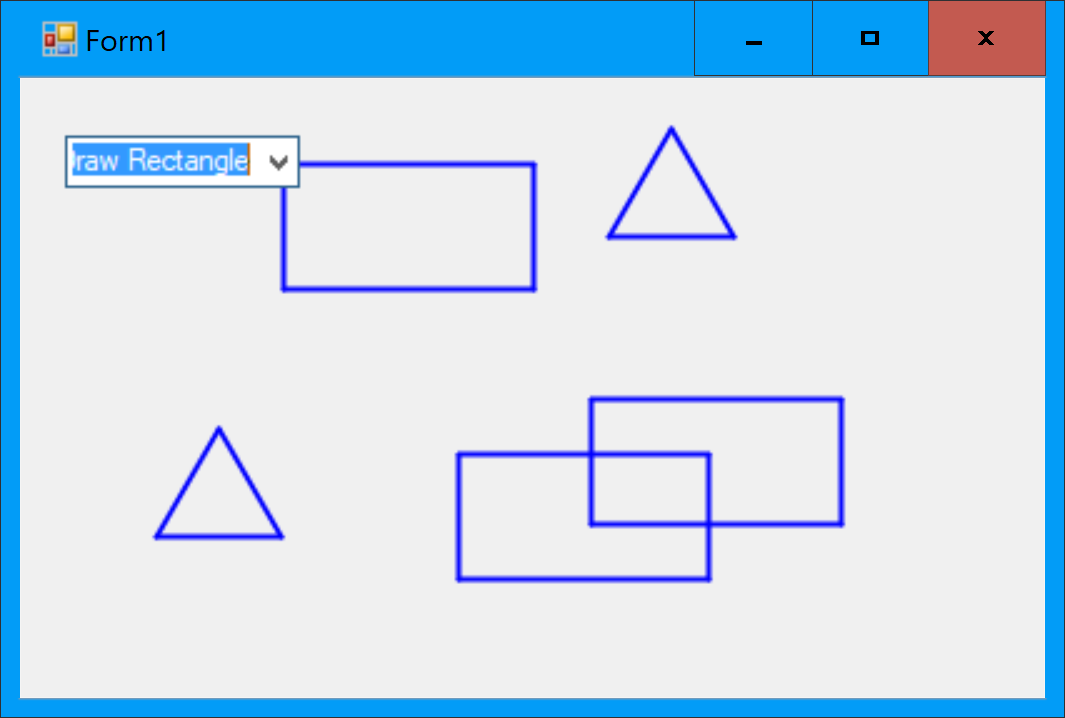
{

DrawRectangle(turtleX, turtleY, 50, 100);

}

}

1. Paste in a screenshot showing that you have drawn a triangle and a rectangle in different places on the same screen.



1. Paste your code for the EquilateralTriangle class.

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;

}

}

}

1. Paste in your equivalent code changes for drawing the EquilateralTriangle.

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

{

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

DrawTriangle(tri);

}

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);

}

}

1. What compile error messages appear within the Draw method?

An object reference is required for the non-static field, method, or property 'Rectangle.Height'.

An object reference is required for the non-static field, method, or property 'Rectangle.Width.

1. Paste in the sections of code you changed, equivalent to those shown above for Rectangle.

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

{

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

tri.Draw();

}

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);

}

}

}

1. What compile error message do you get?

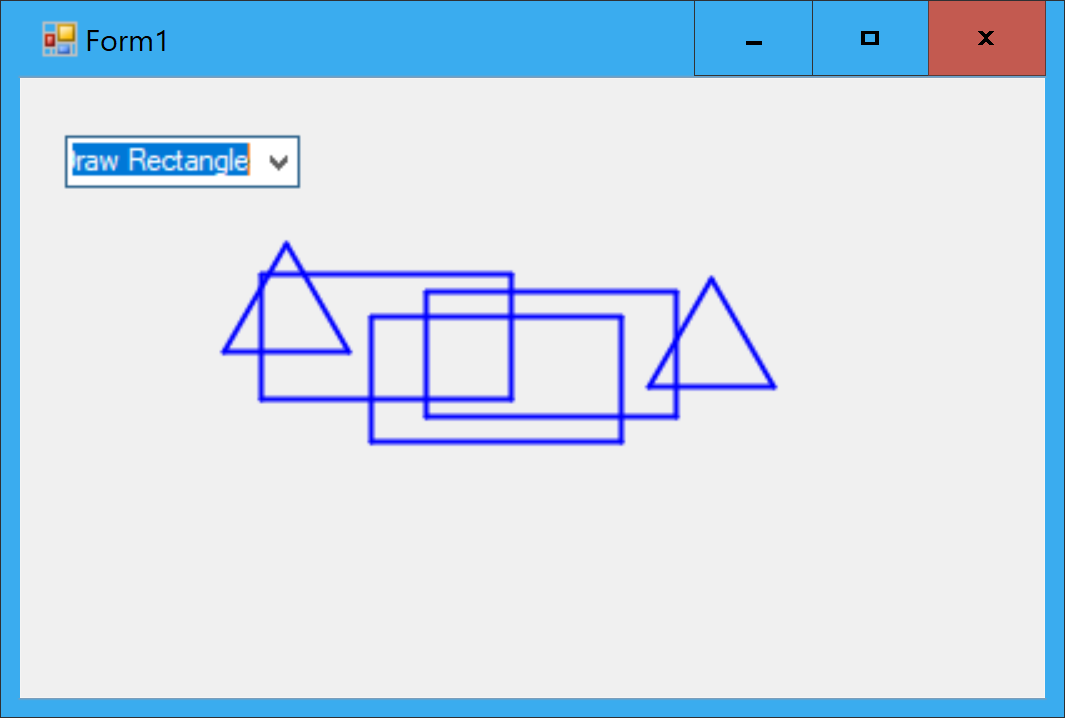
'object' does not contain a definition for 'Draw' and no extension method 'Draw' accepting a first argument of type 'object' could be found (are you missing a using directive or an assembly reference?)

1. What compile error messages do you get?

'EquilateralTriangle' does not implement interface member 'Shape.MoveTo(float, float)'

'Rectangle' does not implement interface member 'Shape.MoveTo(float, float)'

1. Paste in a screenshot showing that you can still draw rectangles and triangles.



1. Paste your new code for the Draw method on EquilateralTriangle

public override void Draw()

{

ResetTurtle();

Turtle.Rotate(30);

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

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

1. Why is the program not user-friendly?

Because there is no visual indication of which shape is currently selected.

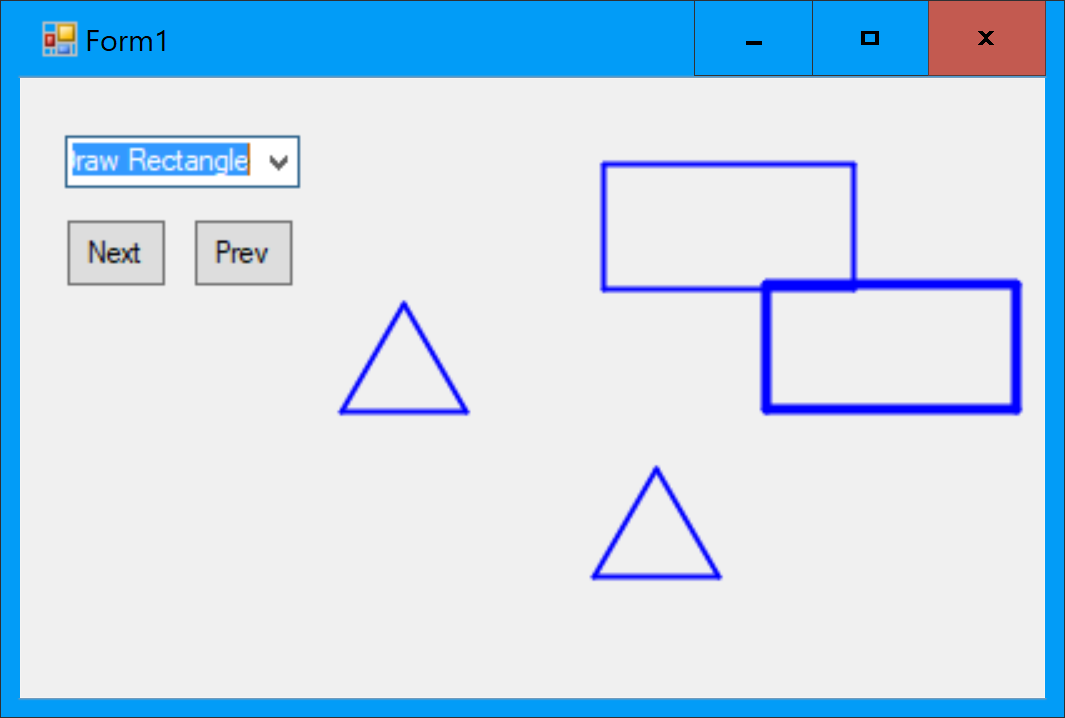
1. What is the bug?

All the shapes are draw in bold.

1. What error arises, and why?

Index was out of range. Must be non-negative and less than the size of the collection. Because when we draw the first shape, there is no previous shape to unselect.

1. Paste in a screenshot showing one of several shapes selected.



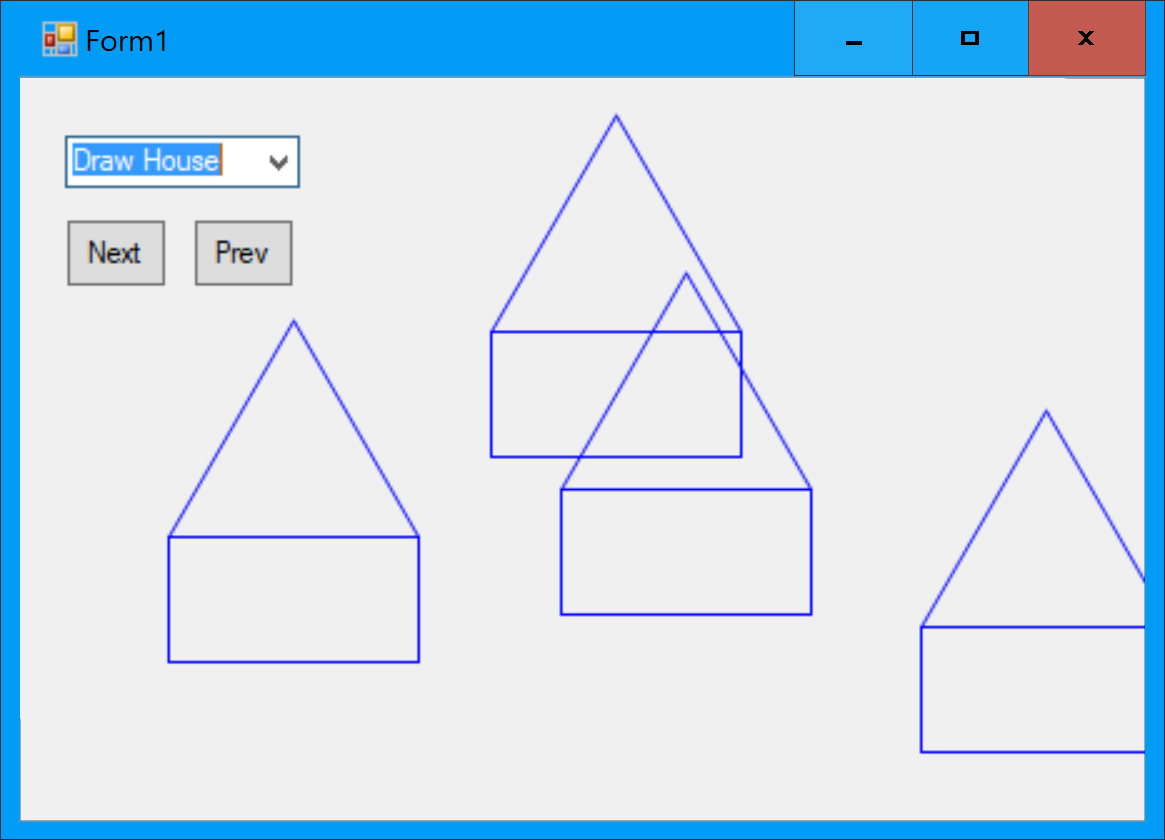
1. What happens, for example if you resize a triangle, and click somewhere low on the screen?

The triangle is inverted (drawn upside-down).

1. Paste in a screenshot of your running program showing the house.



1. Paste in a screenshot:



1. Can you figure out why not?

Because even though the LineWidth on the House has been changed, the drawing is being delegated to the Walls and Roof, and they have not had their LineWidth changed.

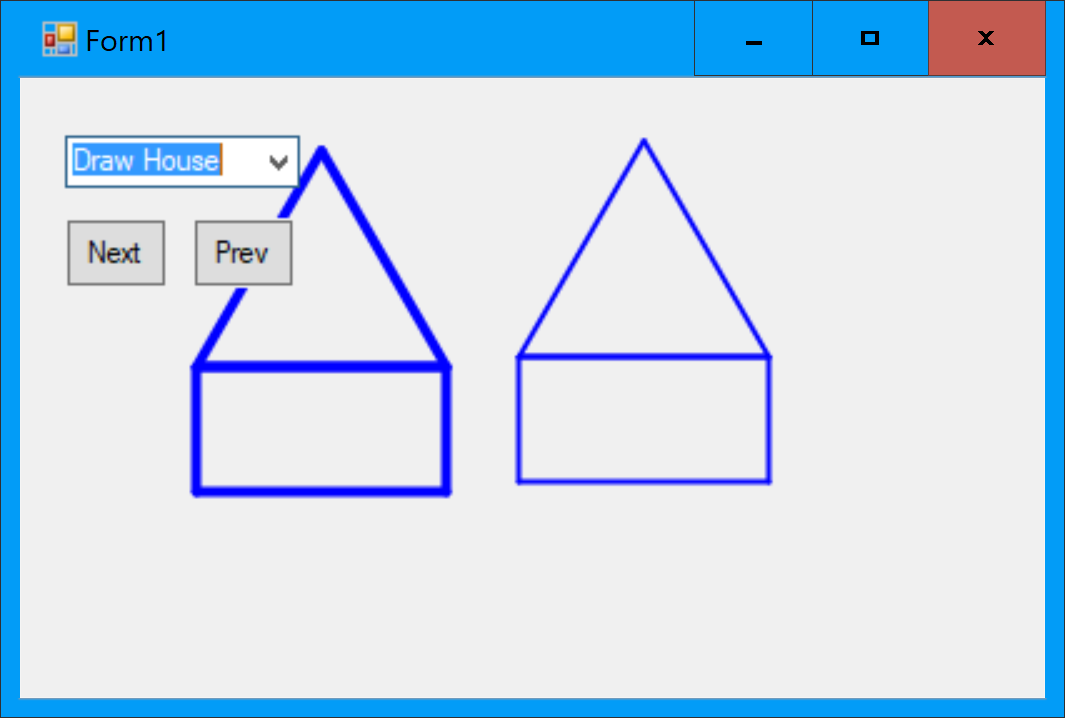
1. What happens if you draw a house, then select Resize and click somewhere? Is this expected?

A System.NotImplementedException is thrown. Yes, this is expected, because we haven’t implemented the Resize method on House.

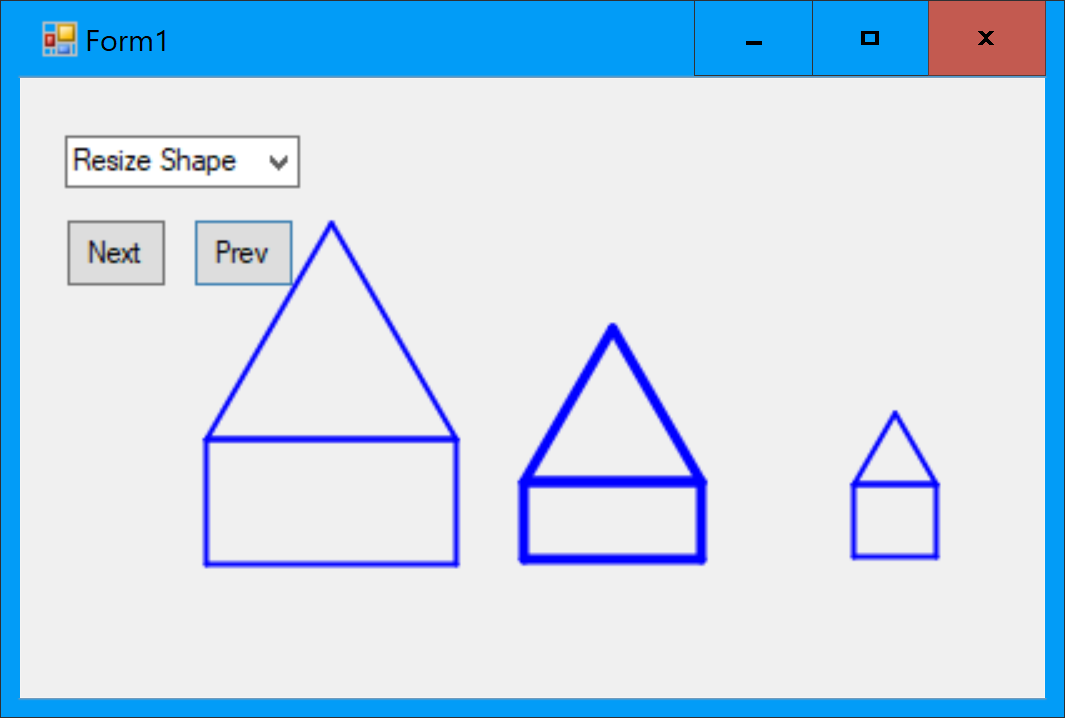
1. What compi;e error message do you get if you *temporarily* remove the keyword virtual from either of the inherited methods on Shape?

'House.Select()': cannot override inherited member 'Shape.Select()' because it is not marked virtual, abstract, or override

1. Paste in a screenshot.



1. Paste in a screenshot showing several houses of different sizes



# Complete code at the end of Exercise 9

## 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());

}

}

}

## Project structure

