VB Version

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

TEACHER’S HANDBOOK

Created by Richard Pawson

[Introduction 3](#_Toc508972356)

[Model answers to Worksheet Questions 4](#_Toc508972357)

[Complete code at the end of Exercise 9 11](#_Toc508972358)

[Shape.vb 11](#_Toc508972359)

[Rectangle.vb 12](#_Toc508972360)

[EquilateralTriangle.vb 13](#_Toc508972361)

[House.vb 14](#_Toc508972362)

[Form1.vb 15](#_Toc508972363)

[Form1.Designer.vb 16](#_Toc508972364)

[Project structure 17](#_Toc508972365)

# 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.

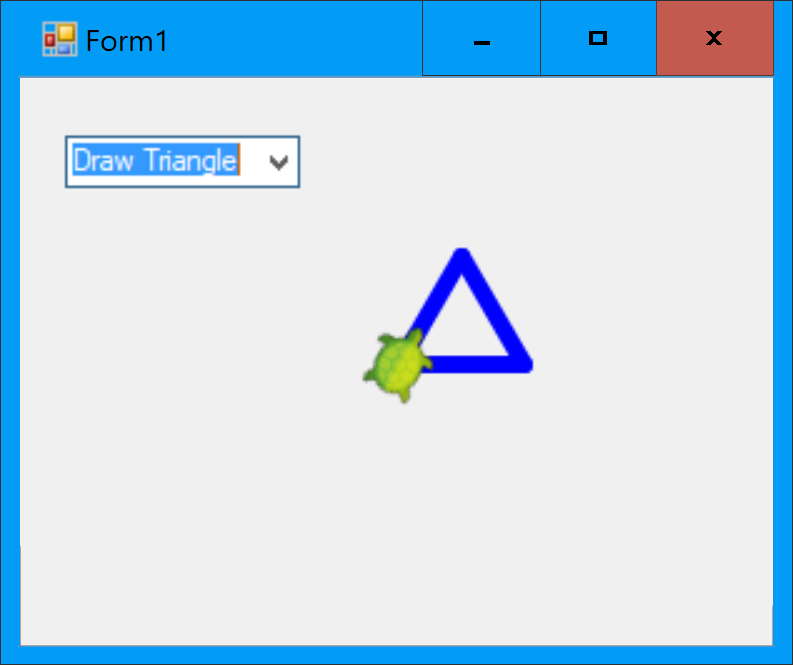
The choice of a Drawing program was deliberate. Apart from being a reasonably engaging idea (students enjoy using the program as well as writing it), it fits very well with the ideas behind OOP. This is not a coincidence: the very first drawing programs - at least, those using bit-mapped graphics - were 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

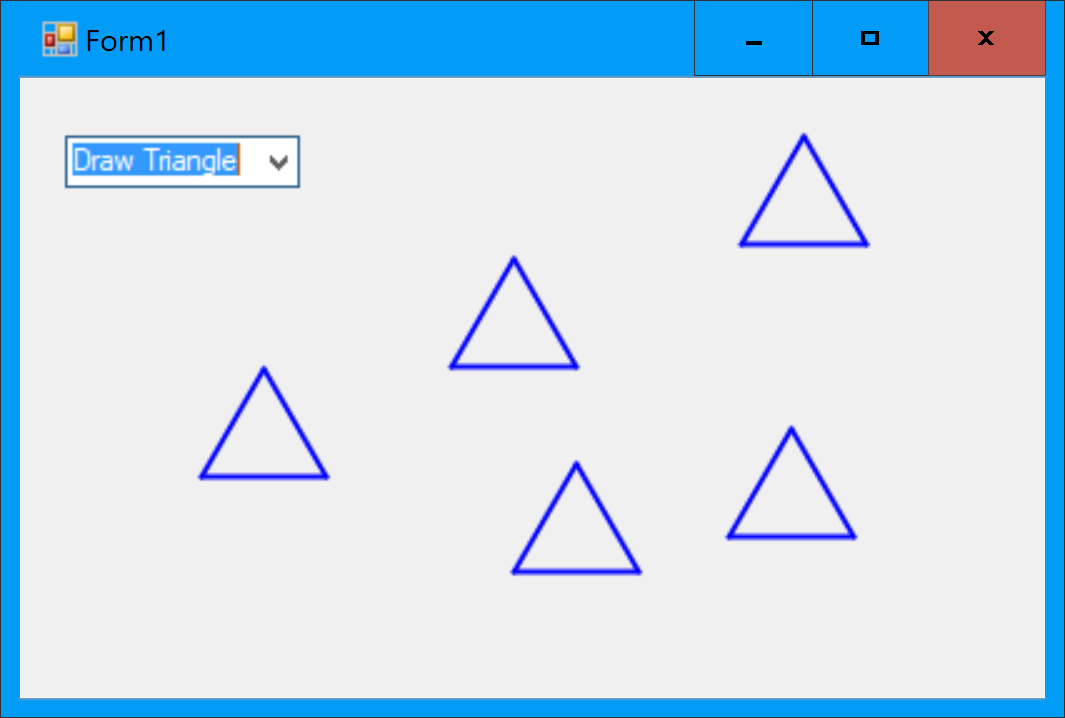
1. What is the value of SelectedItem now?

Nothinhg.

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 Shared Sub DrawRectangle(xOrigin As Single, yOrigin As Single, height As Single, width As Single)

Turtle.ShowTurtle = False

Turtle.PenSize = 2

Turtle.Angle = 0 'Always start from North

Turtle.X = xOrigin

Turtle.Y = yOrigin

For i = 1 To 10

Turtle.Forward(height)

Turtle.Rotate(90)

Turtle.Forward(width)

Turtle.Rotate(90)

Next

End Sub

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

Private Sub Form1\_MouseClick(sender As Object, e As MouseEventArgs) Handles MyBase.MouseClick

'Transform windows coordinates to Turtle coordinates

Dim turtleX As Single = e.X - Width / 2 + 8

Dim turtleY As Single = Height / 2 - e.Y - 19

Dim selectedItem As String = ComboBox1.SelectedItem

If selectedItem = "Draw Triangle" Then 'We Then will add more options later

DrawTriangle(turtleX, turtleY, 50)

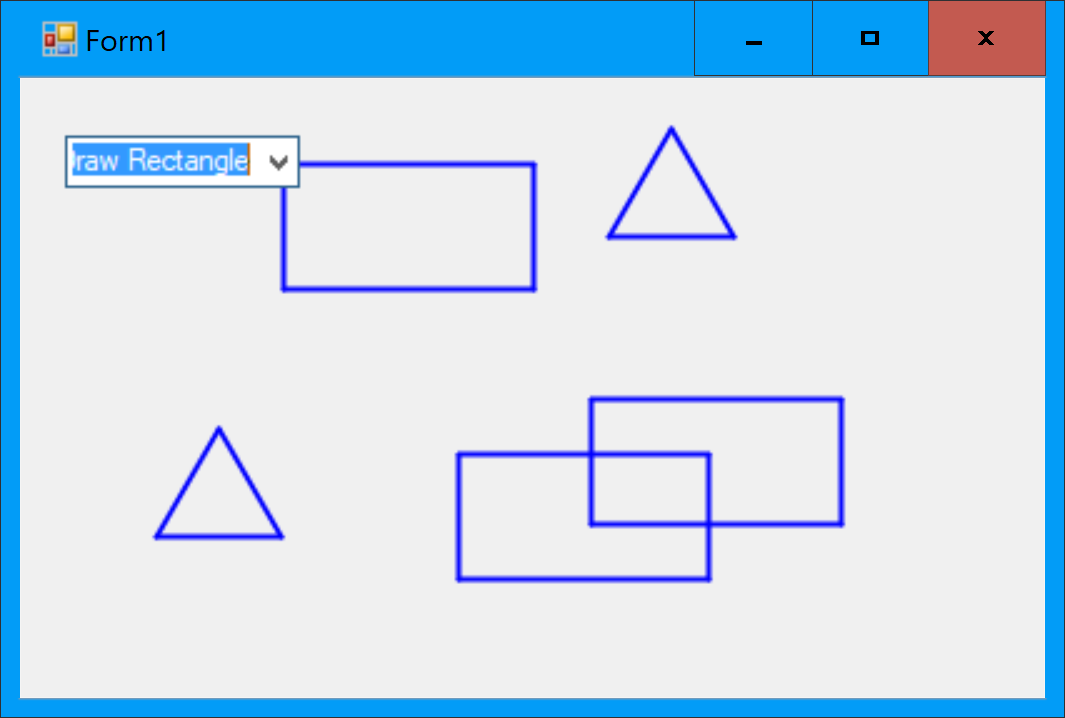
ElseIf selectedItem = "Draw Rectangle" Then

DrawRectangle(turtleX, turtleY, 50, 100)

End If

End Sub

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.

Public Class EquliateralTriangle

'Properties

Public Property XOrigin As Single

Public Property YOrigin As Single

Public Property SideLength As Single

'The 'Constructor'

Public Sub New(xOrigin As Single, yOrigin As Single, sideLength As Single)

Me.XOrigin = xOrigin

Me.YOrigin = yOrigin

Me.SideLength = sideLength

End Sub

End Class

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

If selectedItem = "Draw Triangle" Then

Dim tri = New EquliateralTriangle(turtleX, turtleY, 50)

DrawTriangle(tri)

Private Shared Sub DrawTriangle(tri As EquliateralTriangle)

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 i = 1 To 3

Turtle.Forward(tri.SideLength)

Turtle.Rotate(120)

Next

End Sub

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

Cannot refer to an instance member of a class from within a shared method or shared member initializer without an explicit instance of the class.

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

If selectedItem = "Draw Triangle" Then 'We Then will add more options later

Dim tri = New EquliateralTriangle(turtleX, turtleY, 50)

tri.Draw()

Imports Nakov.TurtleGraphics

Public Class EquliateralTriangle

'Properties

Private Property XOrigin As Single

Private Property YOrigin As Single

Private Property SideLength As Single

'The 'Constructor'

Public Sub New(xOrigin As Single, yOrigin As Single, sideLength As Single)

Me.XOrigin = xOrigin

Me.YOrigin = yOrigin

Me.SideLength = sideLength

End Sub

Public Sub Draw()

Turtle.ShowTurtle = False

Turtle.PenSize = 2

Turtle.Angle = 0 'Always start from North

Turtle.X = XOrigin

Turtle.Y = YOrigin

Turtle.Rotate(30)

For i = 1 To 3

Turtle.Forward(SideLength)

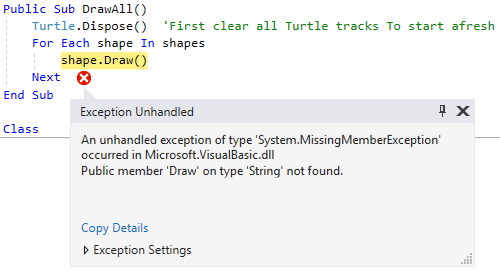
Turtle.Rotate(120)

Next

End Sub

End Class

1. What is the run-time error and where does it occur in the code? Either paste in a screenshot that shows the error, or describe it.



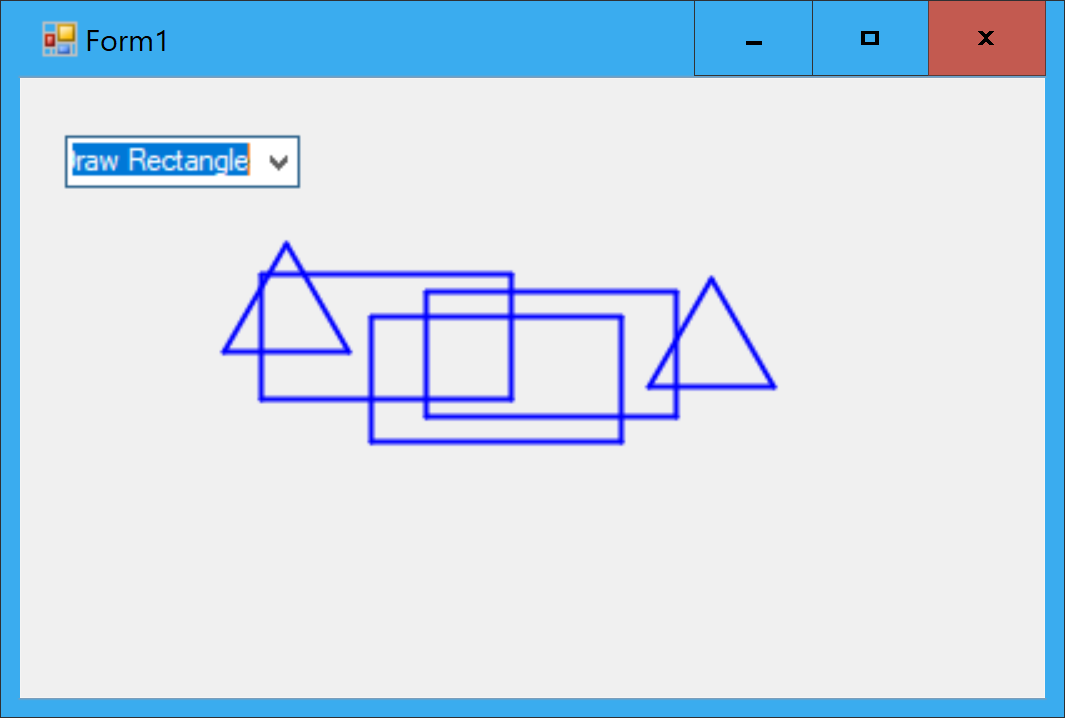
1. What compile error messages do you get?

Class 'Rectangle' must implement 'Sub MoveTo(x As Single, y As Single)' for interface 'Shape'.OOPDraw

Class 'EquliateralTriangle' must implement 'Sub MoveTo(x As Single, y As Single)' for interface 'Shape'. OOPDraw

UP TO HERE

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 Overrides Sub Draw()

ResetTurtle()

Turtle.Rotate(30)

For i = 1 To 3

Turtle.Forward(SideLength)

Turtle.Rotate(120)

Next

End Sub

1. Why is the program not user-friendly?

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

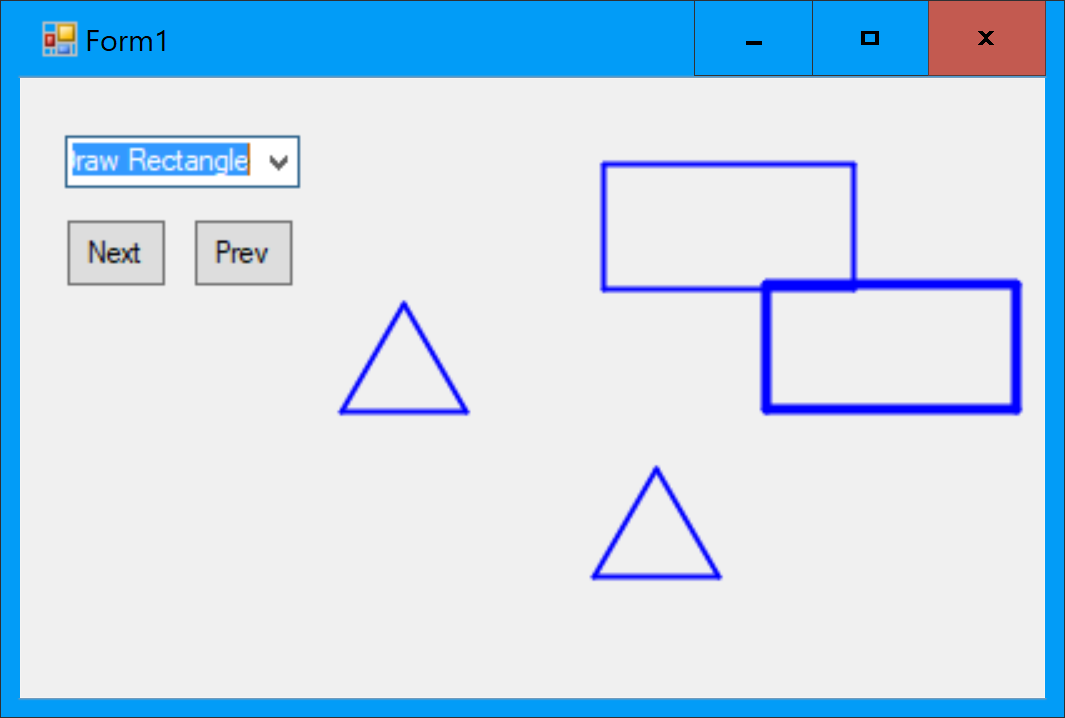
1. What is the bug?

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.



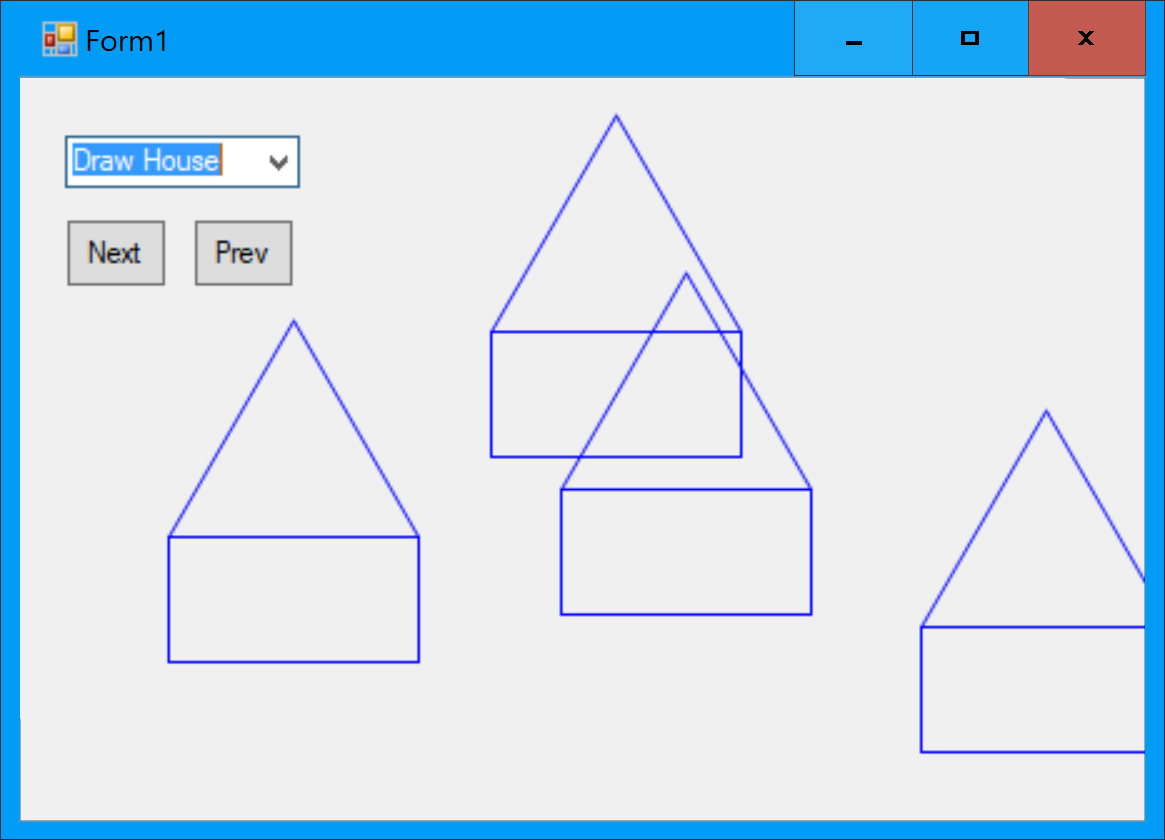
1. What happens, for example if you resize a triangle, and click somewhere low on the screen?

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.

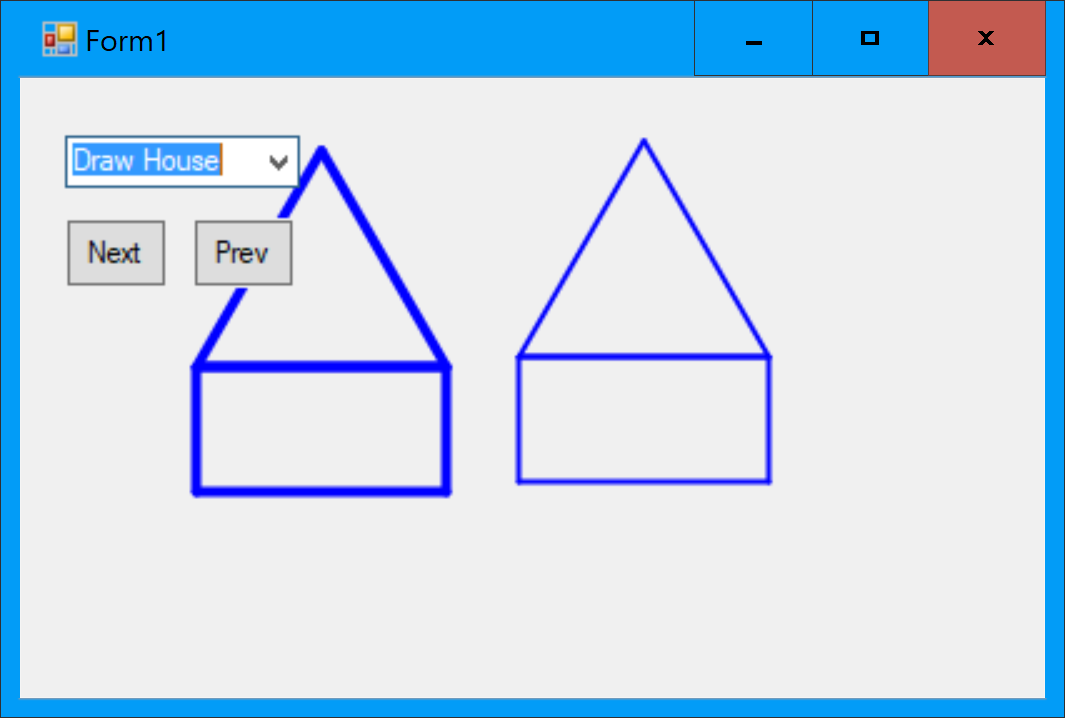
1. What happens if you draw a house, then select Resize and click somewhere? Is this expected?

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

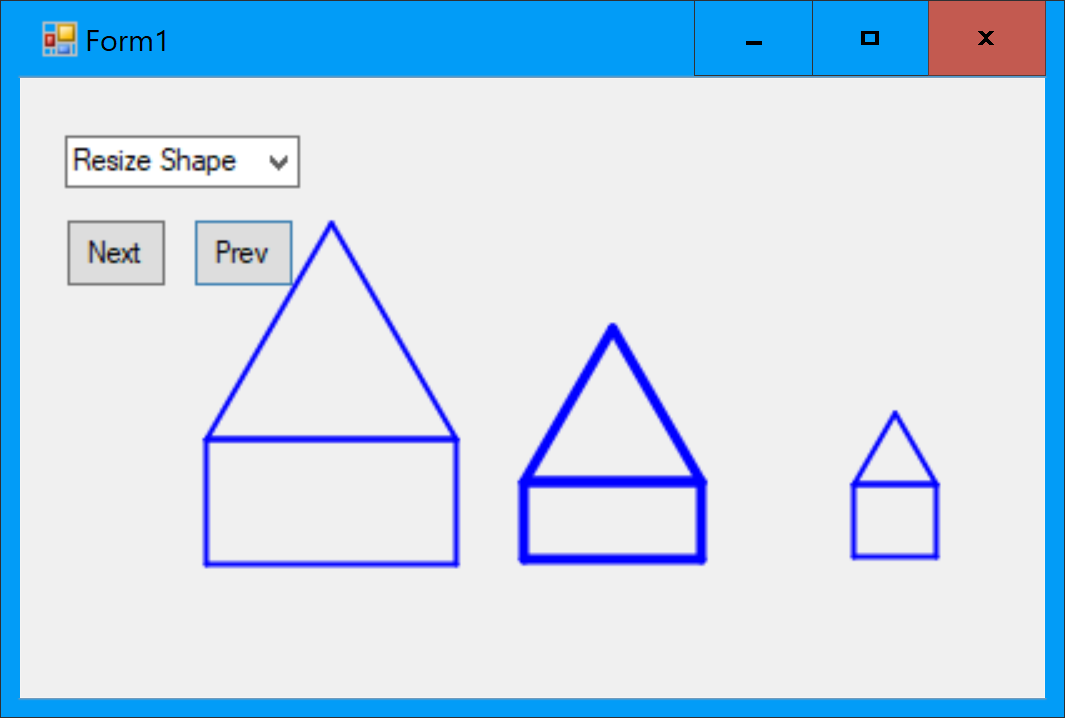
1. What compile error message do you get if you *temporarily* remove the keyword Overridable from either of the inherited methods on Shape?

e.g. 'Public Overrides Sub SelectShape()' cannot override 'Public Sub SelectShape()' because it is not declared 'Overridable'.

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.vb

Imports Nakov.TurtleGraphics

Public MustInherit Class Shape

Protected Property XOrigin As Single

Protected Property YOrigin As Single

Private Property LineWidth As Single

Public Sub New(xOrigin As Single, yOrigin As Single)

Me.XOrigin = xOrigin

Me.YOrigin = yOrigin

End Sub

Public Overridable Sub SelectShape()

LineWidth = 4

End Sub

Public Overridable Sub UnselectShape()

LineWidth = 2

End Sub

Public MustOverride Sub Draw()

Public Overridable Sub MoveTo(x As Single, y As Single)

XOrigin = x

YOrigin = y

End Sub

Public Sub MoveBy(x As Single, y As Single)

XOrigin += x

YOrigin += y

End Sub

Public Sub ResizeAbsolute(turtleX As Single, turtleY As Single)

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

End Sub

Public MustOverride Sub Resize(x As Single, y As Single)

Protected Sub ResetTurtle()

Turtle.ShowTurtle = False

Turtle.PenSize = LineWidth

Turtle.Angle = 0 'Always start from North

Turtle.X = XOrigin

Turtle.Y = YOrigin

End Sub

End Class

## Rectangle.vb

Imports Nakov.TurtleGraphics

Public Class Rectangle

Inherits Shape

'Properties

Private Property Width As Single

Private Property Height As Single

'The 'Constructor'

Public Sub New(xOrigin As Single, yOrigin As Single, width As Single, height As Single)

MyBase.New(xOrigin, yOrigin)

Me.Width = width

Me.Height = height

End Sub

Public Overrides Sub Draw()

ResetTurtle()

For i = 1 To 10

Turtle.Forward(Height)

Turtle.Rotate(90)

Turtle.Forward(Width)

Turtle.Rotate(90)

Next

End Sub

Public Overrides Sub Resize(x As Single, y As Single)

Width = x

Height = y

End Sub

End Class

## EquilateralTriangle.vb

Imports Nakov.TurtleGraphics

Public Class EquliateralTriangle

Inherits Shape

'Properties

Private Property SideLength As Single

'The 'Constructor'

Public Sub New(xOrigin As Single, yOrigin As Single, sideLength As Single)

MyBase.New(xOrigin, yOrigin)

Me.SideLength = sideLength

End Sub

Public Overrides Sub Draw()

ResetTurtle()

Turtle.Rotate(30)

For i = 1 To 3

Turtle.Forward(SideLength)

Turtle.Rotate(120)

Next

End Sub

Public Overrides Sub Resize(x As Single, y As Single)

'Ignore y

SideLength = x

End Sub

End Class

## House.vb

Public Class House

Inherits Shape

Private Property Width As Single

Private Property WallHeight As Single

Private Property Walls As Rectangle

Private Property Roof As EquliateralTriangle

Public Sub New(originX As Single, originY As Single, width As Single, wallHeight As Single)

MyBase.New(originX, originY)

Me.Width = width

Me.WallHeight = wallHeight

Walls = New Rectangle(originX, originY, width, wallHeight)

Roof = New EquliateralTriangle(originX, originY + wallHeight, width)

End Sub

Public Overrides Sub Draw()

Walls.Draw()

Roof.Draw()

End Sub

Public Overrides Sub SelectShape()

Walls.SelectShape()

Roof.SelectShape()

End Sub

Public Overrides Sub UnselectShape()

Walls.UnselectShape()

Roof.UnselectShape()

End Sub

Public Overrides Sub MoveTo(x As Single, y As Single)

MyBase.MoveTo(x, y)

Walls.MoveTo(x, y)

Roof.MoveTo(x, y + WallHeight)

End Sub

Public Overrides Sub Resize(x As Single, y As Single)

Width = x

Dim yDiff = y - WallHeight

WallHeight = y

Walls.Resize(x, y)

Roof.Resize(x, 0)

Roof.MoveBy(0, yDiff)

End Sub

End Class

## Form1.vb

Imports Nakov.TurtleGraphics

Public Class Form1

Private shapes As List(Of Shape) = New List(Of Shape)()

Private Sub Form1\_MouseClick(sender As Object, e As MouseEventArgs) Handles MyBase.MouseClick

Dim turtleX As Single = CType(e.X - Width / 2 + 8, Single)

Dim turtleY As Single = CType(Height / 2 - e.Y - 19, Single)

Dim selectedItem As String = CType(ComboBox1.SelectedItem, String)

If selectedItem = "Draw Triangle" Then 'We Then will add more options later

AddShape(New EquliateralTriangle(turtleX, turtleY, 50))

ElseIf selectedItem = "Draw Rectangle" Then

AddShape(New Rectangle(turtleX, turtleY, 100, 50))

ElseIf selectedItem = "Draw House" Then

AddShape(New House(turtleX, turtleY, 100, 50))

ElseIf selectedItem = "Move Shape" Then

ActiveShape().MoveTo(turtleX, turtleY)

ElseIf selectedItem = "Resize Shape" Then

ActiveShape().ResizeAbsolute(turtleX, turtleY)

End If

DrawAll()

End Sub

'ActiveShape().Resize(turtleX, turtleY)

Private Sub AddShape(shape As Shape)

If shapes.Count > 0 Then

ActiveShape().UnselectShape()

End If

shapes.Add(shape)

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

ActiveShape().SelectShape()

End Sub

Public Sub DrawAll()

Turtle.Dispose() 'First clear all Turtle tracks To start afresh

For Each shape In shapes

shape.Draw()

Next

End Sub

Private activeShapeNumber As Integer = 0

Private Function ActiveShape() As Shape

Return shapes(activeShapeNumber) 'List elements can be accessed Like an array

End Function

Private Sub NextShape\_Click(sender As Object, e As EventArgs) Handles NextShape.Click

ActiveShape().UnselectShape()

activeShapeNumber = activeShapeNumber + 1

If (activeShapeNumber >= shapes.Count) Then activeShapeNumber = 0

ActiveShape().SelectShape()

DrawAll()

End Sub

Private Sub PrevShape\_Click(sender As Object, e As EventArgs) Handles PrevShape.Click

ActiveShape().UnselectShape()

activeShapeNumber = activeShapeNumber - 1

If (activeShapeNumber < 0) Then activeShapeNumber = shapes.Count - 1

ActiveShape().SelectShape()

DrawAll()

End Sub

End Class

## Form1.Designer.vb

<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> \_

Partial Class Form1

Inherits System.Windows.Forms.Form

'Form overrides dispose to clean up the component list.

<System.Diagnostics.DebuggerNonUserCode()> \_

Protected Overrides Sub Dispose(ByVal disposing As Boolean)

Try

If disposing AndAlso components IsNot Nothing Then

components.Dispose()

End If

Finally

MyBase.Dispose(disposing)

End Try

End Sub

'Required by the Windows Form Designer

Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer

'It can be modified using the Windows Form Designer.

'Do not modify it using the code editor.

<System.Diagnostics.DebuggerStepThrough()> \_

Private Sub InitializeComponent()

Me.ComboBox1 = New System.Windows.Forms.ComboBox()

Me.NextShape = New System.Windows.Forms.Button()

Me.PrevShape = New System.Windows.Forms.Button()

Me.SuspendLayout()

'

'ComboBox1

'

Me.ComboBox1.FormattingEnabled = True

Me.ComboBox1.Items.AddRange(New Object() {"Draw Triangle", "Draw Rectangle", "Move Shape", "Resize Shape", "Draw House"})

Me.ComboBox1.Location = New System.Drawing.Point(13, 13)

Me.ComboBox1.Name = "ComboBox1"

Me.ComboBox1.Size = New System.Drawing.Size(121, 21)

Me.ComboBox1.TabIndex = 0

'

'NextShape

'

Me.NextShape.Location = New System.Drawing.Point(13, 42)

Me.NextShape.Margin = New System.Windows.Forms.Padding(1, 1, 1, 1)

Me.NextShape.Name = "NextShape"

Me.NextShape.Size = New System.Drawing.Size(55, 29)

Me.NextShape.TabIndex = 1

Me.NextShape.Text = "Next"

Me.NextShape.UseVisualStyleBackColor = True

'

'PrevShape

'

Me.PrevShape.Location = New System.Drawing.Point(85, 42)

Me.PrevShape.Margin = New System.Windows.Forms.Padding(1, 1, 1, 1)

Me.PrevShape.Name = "PrevShape"

Me.PrevShape.Size = New System.Drawing.Size(47, 29)

Me.PrevShape.TabIndex = 2

Me.PrevShape.Text = "Prev"

Me.PrevShape.UseVisualStyleBackColor = True

'

'Form1

'

Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)

Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font

Me.ClientSize = New System.Drawing.Size(393, 311)

Me.Controls.Add(Me.PrevShape)

Me.Controls.Add(Me.NextShape)

Me.Controls.Add(Me.ComboBox1)

Me.Name = "Form1"

Me.Text = "Form1"

Me.ResumeLayout(False)

End Sub

Friend WithEvents ComboBox1 As ComboBox

Friend WithEvents NextShape As Button

Friend WithEvents PrevShape As Button

End Class

## Project structure

