**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/).

This Teacher Guide is to be used in conjunction with the Student Workbook of the same title.

## What this resource helps to teach

## How to use this resource

# Complete code after Exercise 1

### MyDrawing.cs

using Nakov.TurtleGraphics;

using System;

using System.Drawing;

namespace OOPDraw

{

public class MyDrawing

{

public static void Execute()

{

DrawSquare(0,0,Color.Blue, 100);

DrawSquare (100, 0, Color.Red, 50);

DrawCircle(125, -10, Color.Black, 10);

DrawCircle(25, -10, Color.Black, 10);

}

private static void DrawSquare(float centreX, float centreY,   
 Color lineColor, float sideLength)

{

Turtle.Angle = 0;

Turtle.X = centreX – sideLength/2;

Turtle.Y = centreY– sideLength/2;

Turtle.PenColor = lineColor;

for (int i = 0; i < 4; i++)

{

Turtle.Forward(sideLength);

Turtle.Rotate(90);

}

}

private static void DrawCircle(float centreX, float centreY,   
 Color lineColor, float radius)

{

Turtle.Angle = 0;

Turtle.X = centreX - radius;

Turtle.Y = centreY;

Turtle.PenColor = lineColor;

for (int i = 0; i < 360; i++)

{

Turtle.Forward((float) (2 \* Math.PI \* radius /360));

Turtle.Rotate(1);

}

}

}

}

# Complete code after Exercise 2

### MyDrawing.cs

using Nakov.TurtleGraphics;

using System;

using System.Drawing;

namespace OOPDraw

{

public class MyDrawing

{

public static void Execute()

{

var body = new Square(0,0,Color.Blue, 100);

DrawSquare(body);

var cab = new Square(100, 0, Color.Red, 50);

DrawSquare(cab);

var frontWheel = new Circle(125, -10, Color.Black, 10);

DrawCircle(frontWheel);

var rearWheel = new Circle(25, -10, Color.Black, 10);

DrawCircle(rearWheel);

}

private static void DrawSquare(Square sq)

{

Turtle.Angle = 0;

Turtle.X = sq.CentreX – sq.SideLength/2;

Turtle.Y = sq.CentreY– sq.SideLength/2;

Turtle.PenColor = sq.LineColor;

for (int i = 0; i < 4; i++)

{

Turtle.Forward(sq.SideLength);

Turtle.Rotate(90);

}

}

private static void DrawCircle(Circle c)

{

Turtle.Angle = 0;

Turtle.X =c.CentreX – c.Radius;

Turtle.Y = c.CentreY;

Turtle.PenColor = c.LineColor;

for (int i = 0; i < 360; i++)

{

Turtle.Forward((float) (2 \* Math.PI \* c.Radius /360));

Turtle.Rotate(1);

}

}

}

}

### Square.cs

using System.Drawing;

namespace OOPDraw

{

public class Square

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

public float SideLength { get; set; }

//The 'Constructor

public Square(float centreX, float centreY, Color lineColor, float sideLength)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

SideLength = sideLength;

}

}

}

### Circle.cs

using System.Drawing;

namespace OOPDraw

{

public class Circle

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

public float Radius { get; set; }

//The 'Constructor

public Circle(float centreX, float centreY, Color lineColor, float radius)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

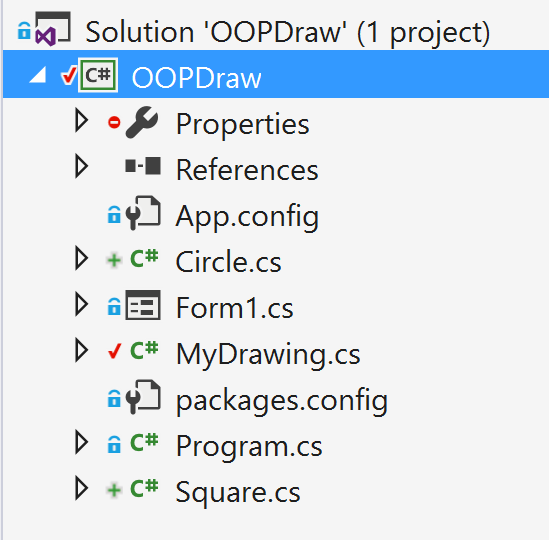
Radius = radius;

}

}

}

## Project view



# Complete code after Exercise 3

### MyDrawing

using System.Drawing;

namespace OOPDraw

{

public class MyDrawing

{

public static void Execute()

{

var body = new Square(0,0,Color.Blue, 100);

body.Draw();

var cab = new Square(100, 0, Color.Red, 50);

cab.Draw();

var frontWheel = new Circle(125, -10, Color.Black, 10);

frontWheel.Draw();

var rearWheel = new Circle(25, -10, Color.Black, 10);

rearWheel.Draw();

}

}

}

### Square

using Nakov.TurtleGraphics;

using System.Drawing;

namespace OOPDraw

{

public class Square

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

public float SideLength { get; set; }

//The 'Constructor

public Square(float centreX, float centreY, Color lineColor, float sideLength)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

SideLength = sideLength;

}

public void Draw()

{

Turtle.Angle = 0;

Turtle.X = CentreX – SideLength/2;

Turtle.Y = CentreY – SideLength/2;

Turtle.PenColor = LineColor;

for (int i = 0; i < 4; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(90);

}

}

}

}

### Circle

using System.Drawing;

using Nakov.TurtleGraphics;

using System;

namespace OOPDraw

{

public class Circle

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

public float Radius { get; set; }

//The 'Constructor

public Circle(float centreX, float centreY, Color lineColor, float radius)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

Radius = radius;

}

public void Draw()

{

Turtle.Angle = 0;

Turtle.X = CentreX - Radius;

Turtle.Y = CentreY;

Turtle.PenColor = LineColor;

for (int i = 0; i < 360; i++)

{

Turtle.Forward((float)(2 \* Math.PI \* Radius / 360));

Turtle.Rotate(1);

}

}

}

}

# Complete code after Exercise 4

### MyDrawing

using System.Collections.Generic;

using System.Drawing;

namespace OOPDraw

{

public class MyDrawing

{

public static void Execute()

{

var list = new List<Shape>();

var body = new Square(50, 50, Color.Blue, 100);

list.Add(body);

var cab = new Square(125, 25, Color.Red, 50);

list.Add(cab);

var frontWheel = new Circle(125, -10, Color.Black, 10);

list.Add(frontWheel);

var rearWheel = new Circle(25, -10, Color.Black, 10);

list.Add(rearWheel);

foreach (var item in list)

{

item.GrowBy(2);

item.Draw();

}

}

}

}

### Shape

namespace OOPDraw

{

public abstract class Shape

{

public abstract void Draw();

public abstract void GrowBy(float factor);

}

}

### Square

using Nakov.TurtleGraphics;

using System.Drawing;

using System;

namespace OOPDraw

{

public class Square : Shape

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

public float SideLength { get; set; }

//The 'Constructor

public Square(float centreX, float centreY, Color lineColor, float sideLength)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

SideLength = sideLength;

}

public override void Draw()

{

Turtle.Angle = 0;

Turtle.X = CentreX-SideLength/2; //To ensure shape is centred correctly

Turtle.Y = CentreY-SideLength/2;

Turtle.PenColor = LineColor;

for (int i = 0; i < 4; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(90);

}

}

public override void GrowBy(float factor)

{

CentreX \*= factor;

CentreY \*= factor;

SideLength \*= factor;

}

}

}

### Circle

using System.Drawing;

using Nakov.TurtleGraphics;

using System;

namespace OOPDraw

{

public class Circle : Shape

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

public float Radius { get; set; }

//The 'Constructor

public Circle(float centreX, float centreY, Color lineColor, float radius)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

Radius = radius;

}

public override void Draw()

{

Turtle.Angle = 0;

Turtle.X = CentreX - Radius;

Turtle.Y = CentreY;

Turtle.PenColor = LineColor;

for (int i = 0; i < 360; i++)

{

Turtle.Forward((float)(2 \* Math.PI \* Radius / 360));

Turtle.Rotate(1);

}

}

public override void GrowBy(float factor)

{

CentreX \*= factor;

CentreY \*= factor;

Radius \*= factor;

}

}

}

# Complete code after Exercise 5

### MyDrawing

using System.Collections.Generic;

using System.Drawing;

namespace OOPDraw

{

public class MyDrawing

{

public static void Execute()

{

var list = new List<Shape>();

var body = new Square(50, 50, Color.Blue, 100);

list.Add(body);

var cab = new Square(125, 25, Color.Red, 50);

list.Add(cab);

var frontWheel = new Circle(125, -10, Color.Black, 10);

list.Add(frontWheel);

var rearWheel = new Circle(25, -10, Color.Black, 10);

list.Add(rearWheel);

var warning = new EquilateralTriangle(-30, -40, Color.Red, 20);

list.Add(warning);

foreach (var item in list)

{

item.MoveCentreBy(20, 0);

item.GrowBy(2);

item.Draw();

}

}

}

}

### Shape

using System.Drawing;

namespace OOPDraw

{

public abstract class Shape

{

//Properties

public float CentreX { get; set; }

public float CentreY { get; set; }

public Color LineColor { get; set; }

//Abstract methods - to be implemeted in sub-types

public abstract void Draw();

public abstract void GrowBy(float factor);

//Concrete methods

public void MoveCentreBy(float xDiff, float yDiff)

{

CentreX += xDiff;

CentreY += yDiff;

}

}

}

### Square

using Nakov.TurtleGraphics;

using System.Drawing;

using System;

namespace OOPDraw

{

public class Square : Shape

{

//Properties

public float SideLength { get; set; }

//The 'Constructor

public Square(float centreX, float centreY, Color lineColor, float sideLength)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

SideLength = sideLength;

}

public override void Draw()

{

Turtle.Angle = 0;

Turtle.X = CentreX-SideLength/2; //To ensure shape is centred correctly

Turtle.Y = CentreY-SideLength/2;

Turtle.PenColor = LineColor;

for (int i = 0; i < 4; i++)

{

Turtle.Forward(SideLength);

Turtle.Rotate(90);

}

}

public override void GrowBy(float factor)

{

CentreX \*= factor;

CentreY \*= factor;

SideLength \*= factor;

}

}

}

### Circle

using System.Drawing;

using Nakov.TurtleGraphics;

using System;

namespace OOPDraw

{

public class Circle : Shape

{

//Properties

public float Radius { get; set; }

//The 'Constructor

public Circle(float centreX, float centreY, Color lineColor, float radius)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

Radius = radius;

}

public override void Draw()

{

Turtle.Angle = 0;

Turtle.X = CentreX - Radius;

Turtle.Y = CentreY;

Turtle.PenColor = LineColor;

for (int i = 0; i < 360; i++)

{

Turtle.Forward((float)(2 \* Math.PI \* Radius / 360));

Turtle.Rotate(1);

}

}

public override void GrowBy(float factor)

{

CentreX \*= factor;

CentreY \*= factor;

Radius \*= factor;

}

}

}

### EquilateralTriangle

using Nakov.TurtleGraphics;

using System;

using System.Drawing;

namespace OOPDraw

{

public class EquilateralTriangle : Shape

{

//Properties

public float SideLength { get; set; }

//The 'Constructor

public EquilateralTriangle(float centreX, float centreY, Color lineColor, float sideLength)

{

LineColor = lineColor;

CentreX = centreX;

CentreY = centreY;

SideLength = sideLength;

}

public override void Draw()

{

Turtle.Angle = -90;

Turtle.X = CentreX - SideLength / 2; //To ensure shape is centred correctly

Turtle.Y = CentreY - (float) (SideLength \* Math.Sin(60) /3); //Centre is at a third of the height

Turtle.PenColor = LineColor;

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

{

Turtle.Forward(SideLength);

Turtle.Rotate(120);

}

}

public override void GrowBy(float factor)

{

SideLength \*= factor;

}

}

}