Faculty of Science, Engineering and Technology

Object Oriented Programming

Pass Task 2.1: Drawing Program — Shapes

Overview

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. In this task you will start to create an object oriented drawing program.

Purpose:	Learn to apply object oriented programming techniques related to abstraction.
Task:	Create a program that can draw rectangular shapes to the screen.
Time:	Aim to complete this task by the start of week 3
Resources:	

Submission Details

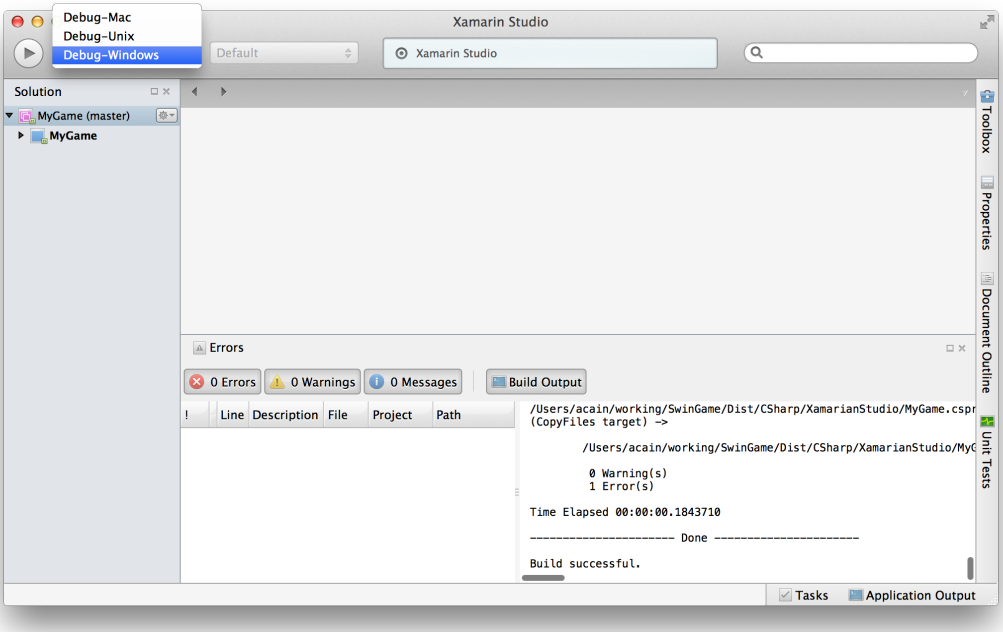
You must submit the following files to Doubtfire:

- Program source code
- Screenshot of program execution

Instructions

Over the next few weeks you will develop a simple shape drawing program. Users will be able to select between different shapes and draw them to the canvas. In this stage you will start by creating a Shape class and drawing it to the screen.

1. Download the starter code for this project from Doubtfire. This contains the code to use SwinGame using C# code. This file contains a project that you can open with Xamarin Studio (MonoDevelop).
2. Open the **MyGame.sln** using Xamarin Studio.
3. Choose your operating system from the configurations drop down.

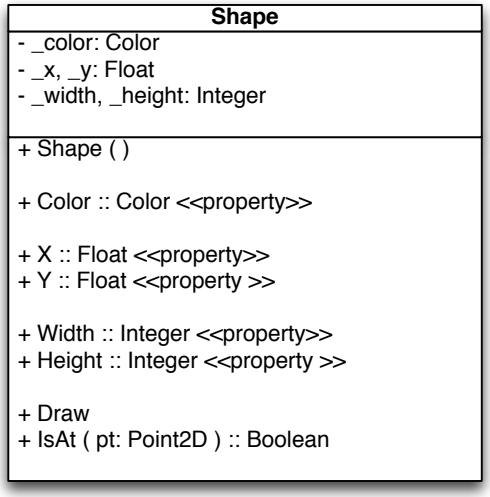


4. Right click the appropriate project (if you are using OS X then select the Mac version, otherwise use the other version) and **Set** it as the **Active Project**.
5. Review the code in the **GameMain** class.
6. Run the program and make sure you see the window and can hear the startup sound from the splash screen.

Note: If you have issues at this point let us know. Error messages can help identify problems. The text from the **Build Output** should help identify the issue.

Note: On Linux you may need to compile and install the backend library (see Source project on website) and run the program from the Terminal after compiling from the IDE.

7. Now it is time to add your Shape class. Use the following UML class diagram as a guide.



8. To get the Color type you will need to add the following code, but the other fields and properties should be straight forward.

```
using SwinGameSDK;

namespace MyGame
{
    public class Shape
    {
        private Color _color;
        ...
```

9. In the constructor initialise the color to Color.Green, x, y to 0,0 and the size to 100 by 100.

10. The Draw method will draw the shape as a filled rectangle using the shape's Color, position, and size.

```
namespace MyGame
{
    public class Shape
    {
        ...
        public void Draw()
        {
            SwinGame.FillRectangle ( _clr,
                                     _x, _y,
                                     _width, _height);
        }
```

Tip: Type **SwinGame.** to get the IDE to list *all* of the SwinGame functions for you!

11. Add a **IsAt** method that takes in a **Point2D** (a struct that contains an X and Y value representing a point in 2d space - like a point on the screen) and returns a boolean to indicate if the shape is at that point. You need to return true if the point (pt) is within the shape's area (as defined by the shape's fields).

Tip: SwinGame has a PointInRect function that can do the hard lifting for you here. When the function is selected in the editor press down to switch between the different **overloads**.

```
▲ 3 of 4 ▼
public static bool PointInRect (
  Point2D pt,
  float x,
  float y,
  float w,
  float h
)
```

Note: Method names can be **overloaded**, this means that multiple methods can use the one method name as long as they have different parameters.

12. Switch back to **GameMain** so that you can test out your new shape class.
13. Add a **myShape** local variable of the Shape type.
14. Assign myShape, a **new** Shape object outside the loop.
15. Inside the loop:
    15.1. Tell **myShape** to **Draw** itself - after clearing the screen
    15.2. If the user clicks the LeftButton on their mouse, set the shapes x, y to be at the mouse's position.

Hint: Check out **MouseClicked**, **MouseX**, and **MouseY** functions.

    15.1. If the mouse is over the shape (i.e. it **is at** the same point as the **mouse position**) and the user presses the spacebar, then change the Color of the shape to a random color.

Hint: Check out **KeyTyped, MousePosition** and **RandomRGBColor** - set the Alpha to 255 so it is opaque.

16. Compile and run your program.

Tip: It is probably best to make these changes one at a time, and compile and run as you go. It is easier to build a little at a time.

Once your program is working correctly you can prepare it for your portfolio. Take a screenshot of the program running, and ensure your code is well formatted.

Assessment Criteria

Make sure that your task has the following in your submission:

- The program is implemented correctly with a Shape class as indicated
- Code must follow the C# coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show it outputting the correct details.