Faculty of Science, Engineering and Technology

Object Oriented Programming

Pass Task 2.2: Drawing Program — A Basic Shape

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:	SplashKit SDK - documentation linked from Canvas

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. We will be using the SplashKit SDK to assist with graphics and basic user interaction. SplashKit provides a library of potentially useful methods for your tasks, and your Custom Program. Documentation for the library is linked online on the SplashKit website. We expect you will spend time getting familiar with it as you proceed through the lab tasks.

1. Setup and install SplashKit by following one of the video tutorials provided to you on Canvas that corresponds to your operating system.

> **Note**: We do not officially support Linux in this unit. If you want, you may still use Linux but no setup tutorial will be provided and support from our staff will be limited.

2. Create a folder called **ShapeDrawer** for your new project.
3. Create a new SplashKit project in this folder using the instructions provided in the tutorial video.

> **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.

4. Write the following code in the **Program** class.

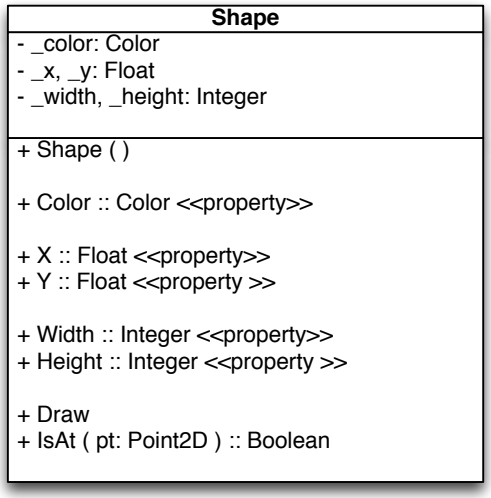
```
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Program
    {
        public static void Main()
        {
            new Window("Shape Drawer", 800, 600);
            do
            {
                SplashKit.ProcessEvents();
                SplashKit.ClearScreen();

                SplashKit.RefreshScreen();
            } while (!SplashKit.WindowCloseRequested("Shape
                Drawer"));
        }
    }
}
```

Tip: Understanding what each line does in the above code is critical to understanding the basics of how SplashKit works. Study the code above and if need be check the SplashKit documentation to gain a better understanding of what the code does.

5. Run the program and make sure you see the window.
6. Now it is time to add your Shape class. Use the following UML class diagram as a guide.



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

```
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Shape
    {
        private Color _color;
        …
```

8. In the constructor initialise the color to Color.Green, x, y to 0,0 and the size to 100 by 100.
9. The Draw method will draw the shape as a filled rectangle using the shape's Color, position, and size.

```
    public class Shape
    {
        …
        public void Draw()
        {
            SplashKit.FillRectangle (_color, _x, _y,
                                     _width, _height);
        }
```

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

10. 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**: You are going to have to do some math here. Think about what range the X and Y coordinates will need to be within for the point to be considered inside the rectangle. If you get stuck, don't hesitate to seek out help.

11. Switch back to **Program** so that you can test out your new shape class.
12. Add a **myShape** local variable of the Shape type.
13. Assign myShape, a **new** Shape object outside the loop.
14. Inside the loop:
    14.1. Tell **myShape** to **Draw** itself - after clearing the screen
    14.2. 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.

    14.3. 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.

    14.4. Make sure you have told **myShape** to **Draw** itself *after* you check and process the user input. If you haven't, move that line of code!

> **Hint**: It is best practice to leave drawing as the last operation before refreshing the screen. Otherwise your objects will be one frame behind.

15. 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.