School of Science, Computing and Engineering Technologies

Object Oriented Programming

Pass Task 2.3: 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 the concept of abstraction.
Task:	Create a program that can draw rectangular shapes to the screen.
Deadline:	Due by the start of week three, Monday, 11 March 2024.

Submission Details

All students have access to the Adobe Acrobat tools. Please print your solution to PDF and combine it with the screenshots taken for this task.

- C# code files of the classes created.
- Screenshot of running program (i.e., SplashKit window).

Instructions

Over the course of the next few weeks you will develop a simple shape drawing program. You begin by creating a *Shape* class that allows you to draw a predefined shape to the screen. This task requires *SplashKit*, a third-party framework for 2D graphics, audio, physics, and animation support. The documentation can be found on the *SplashKit* website splashkit.io. Some elements of *SplashKit* can be overwhelming at first, but as you get more familiar with its features, tasks will become easier.

1. Install *SplashKit*. You can find guides for Windows and macOS installation on Canvas.

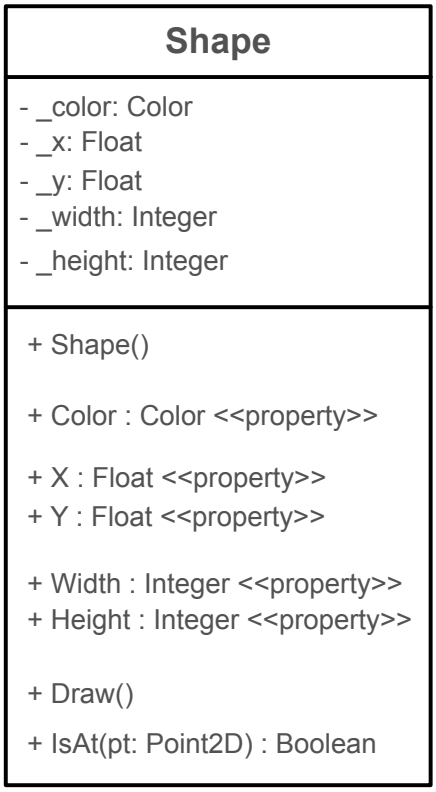
> **Note**: *SplashKit* is also available for Linux, but Linux is not supported in COS20007.

2. Start an MSYS2 MINGW64 terminal (or Terminal on macOS).
3. In your project directory, create a new folder called **ShapeDrawer** for your new project.
4. Change into directory (i.e., cd) **ShapeDrawer**.
5. Create a new *SplashKit* project in this folder using **skm dotnet new**.
6. Open *ShapeDrawer.csproj* in Visual Studio.
7. Change *Program.cs* as follows:

```
using SplashKitSDK;

namespace ShapeDrawer
{
   public class Program
   {
     public static void Main()
     {
       Window window = new Window("Shape Drawer", 800, 600);

       do
       {
         SplashKit.ProcessEvents();
         SplashKit.ClearScreen();

         SplashKit.RefreshScreen();
       } while (!window.CloseRequested);
     }
   }
}
```

Tip: Consult the *SplashKit* documentation. It contains guides for drawing and event handling. In addition, check the Developer Documentation sections on Graphics, Windows, and Input that explain the features used in the above code. Understanding what each line does in the above code is critical to mastering *SplashKit*.

8. Run the program (a window with white background appears on your screen). To close the application, click on × (Windows) or ● (macOS) in the window title.
9. Add a *Shape* class to your project. Use the following UML class diagram as a guide.



10. The type *Color* is defined in the *SplashKitSDK*. To use this type and the other drawing abstractions, add ***using*** SplashKitSDK; at the start of the file *Shape.cs*. The using statement makes all *SplashKit* features available in the current compilation unit.

```
using SplashKitSDK;

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

11. In the constructor, initialize ***_color*** to *Color.Green*, set ***_x*** and ***_y*** to 0.0f (the suffix f makes 0.0 a float value), and set both ***_width*** and ***_height*** to 100.

12. The ***Draw*** method will draw the shape as a filled rectangle using the shape's Color, position, and dimension.

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

Tip: Visual Studio can assist you in finding the right class or method, Start typing 'SplashKit' and Visual Studio will provide you with suggestions to complete a code line.

13. Add the ***IsAt*** method which takes a ***Point2D*** point (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 coordinates).

Tip: What does it mean for a point to be considered inside the area of a rectangle? Note, the boundary of a shape is part of the area. If you get stuck, do not hesitate to seek out help.

14. Add all properties (as defined in the UML diagram) to class *Shape*.
15. Return to the *Program.cs* file.
16. In ***Main***,
    - Add a ***myShape*** local variable of the type *Shape*.
    - Assign ***myShape***, a **new** *Shape* object.

17. Inside the ***do-while*** loop in **Main**:

    - Tell ***myShape*** to ***Draw*** itself - after clearing the screen.
    - Add an ***if***-statement to check whether the user has clicked the left mouse button. If this is the case, set ***myShapes***'s ***X*** and ***Y*** coordinates to the mouse position.

> **Hint**: You need to use the *SplashKit* functions ***MouseClicked***, ***MouseX***, and ***MouseY***, and use ***LeftButton*** as argument to function ***MouseClicked***.
>
> You need to write *SplashKit.MouseClicked(MouseButton.LeftButton)* to test if the user has clicked the left mouse button, and *SplashKit.MouseX()* to obtain the x-coordinate of the mouse position.

    - Add an ***if***-statement to test whether the user has pressed the spacebar (use function *SplashKit.KeyTyped* with *KeyCode.SpaceKey*). If, at the same time, the mouse pointer hovers over ***myShape*** (i.e. the mouse pointer is within the area of ***myShape***), then set ***myShape***'s ***Color*** property to a random color value.

> **Hint**: Check out SplashKit's ***KeyTyped***, ***MousePosition***, and ***RandomColor*** functions.

    - You need to tell ***myShape*** to ***Draw*** itself. This should be the last operation before the screen is refreshed. You may need to move the code that tells ***myShape*** to ***Draw***.

18. Compile and run your program. Try changing ***myShape***'s position and color.

Once your program is complete you can prepare it for your portfolio. This can be placed in your portfolio as evidence of what you have learnt.

1. Review your code and ensure it is formatted correctly.
2. Run the program and use your preferred screenshot program to take a screenshot of the Terminal showing the program's output.
3. Save and backup your work to multiple locations, if possible.
    - Once you your program is working you do not want to lose your work.
    - Work on your computer's storage device most of the time, but backup your work when you finish each task.
    - You may use a cloud storage provider to safely store your work.
    - USB and portable hard drives are good secondary backups, but there is a risk that the drive gets damaged or lost.

Assessment Criteria

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

- The "Universal Task Requirements" (see Canvas) have been met.
- Your program (as text or screenshot).
- Screenshot of application and shape at different positions and different color.