School of Science, Computing and Engineering Technologies

Object-Oriented Programming

Week 4: Drawing Program and SwinAdventure Iteration 3

Overview

In this week, there are **two assessable tasks** 4.1 and 4.2. **Each task contributes 2%** to your final grade. Noting that you need to complete these tasks before coming to your allocated lab. In the lab, there will be verification tasks and short interview to verify your understanding.

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. We will use it to develop a drawing program with SplashKit SDK. In addition, we will keep developing a new iteration 3 of the SwinAdventure game. The game will be your opportunity to create a larger program and better understand how the OOP can make it easier to create complex software solutions.

Purposes Task 4.1 **(pages 2-5)**	Learn to apply OOP techniques related to the concept of abstraction and get familiar with using third-party SDK. You can also interpret UML class diagram. **The task contains personalized requirements.**
Task 4.2 (pages 5-7)	Learn to apply object-oriented programming techniques related to the concept of abstraction and conduct the NUnit test. Understand the case study program and implement iteration 3. **The task contains personalized requirements**

Note: If you are unable to complete theses tasks, you will struggle in this unit. Your tutor may be able assist you with suggestions on how increase your knowledge. You can also participate in HelpDesk sessions

SWINBURNE UNIVERSITY OF TECHNOLOGY

Task 3.1. Instructions - Iteration 1: Identifiable Object

1. Review the ***SplashKit Installation Instruction*** document in Week 0. It outlines what you need to create a sample project and can display a graphic window on your screen.

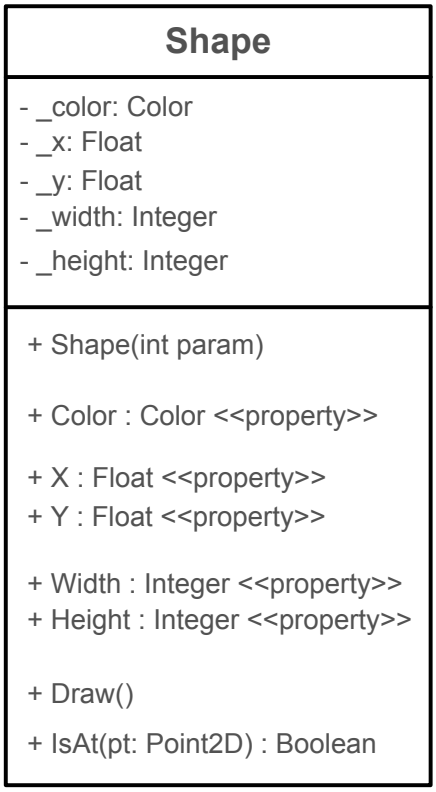
2. In this week, we will develop a simple shape drawing program. You begin by creating a Shape class that allows you to draw a predefine shape to the screen with SplashKit.
3. You can utilize the Shape.cs class that you have developed in Week 2 for this task. Noting that there is a slight difference in the UML diagram of the Shape class for this week. Specifically, we will use the Color and Point2D datatypes of SplashKit so that we can utilize existing methods developed by SplashKit to save implementation time and reduce potential bugs/errors.
4. Start an MSYS2 MINGW64 terminal on Window or Terminal on macOS.
5. Create a folder called **ShapeDrawer** for your new project
6. In the terminal, change current working directory to the **ShapeDrawer** using *cd* command
7. Create a new SplashKit project in this folder using **skm dotnet new**
8. Open the project with Visual Studio Code and 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* API documentation https://splashkit.io/api/ . 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*.

9. 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.

10. Migrate the Shape.cs from Week 2 and modify the class using the following UML class diagram as a guide.



11. The type *Color* is now 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;
        …
```

12. In the constructor, initialize ***_color*** to *Color.Azure* if the first letter of your first name is from A to K. Otherwise, set by your *Color.Chocolate*. In that constructor, set ***_x*** and ***_y*** to 0.0f (the suffix f makes 0.0 a float value), and both the _width and _height are assigned to the value of a given parameter called param. When you later create objects based on this Shape class, please specify the parameter to be 1**XX**, where **XX** is the last two digits of your student ID.

13. 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);
    }
```

14. The ***IsAt*** method 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).

15. Add all properties (as defined in the UML diagram) to class *Shape*.

16. Return to the *Program.cs* file.

17. In ***Main*** method,

- Add a ***myShape*** local variable of the type *Shape*.
- Assign ***myShape***, a **new** *Shape* object.

18. 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***.
>
> See sample usage of https://splashkit.io/api/input/#mouse-clicked
>
> 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.

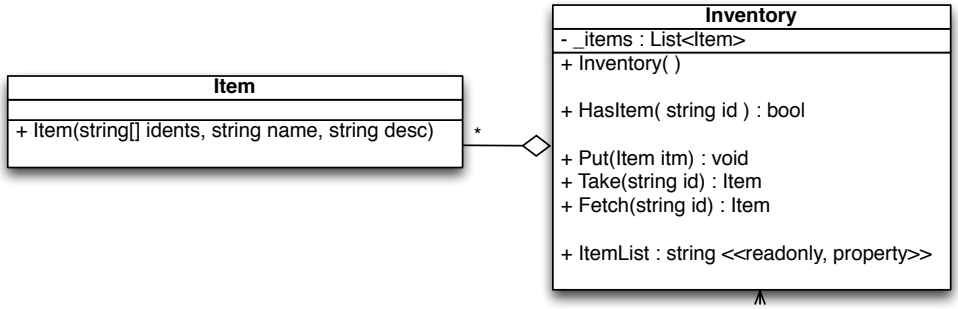
- You need to tell ***myShape*** to ***Draw*** itself. This should be the last operation before the screen is refreshed. You may need to move the code that tells ***myShape*** to ***Draw***.

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

Task 4.2. Instructions - SwinAdventure Iteration 3: Inventory

1. Review the ***Case Study Requirements*** document. It outlines what you need to create.

2. In this Iteration 3 you will create an Inventory class which maintains the list of Item objects. Noting that this Iteration reuses the class Item developed in the Iteration 2.
3. The UML diagram for the Inventory is as follows, and the unit tests to create are shown on the following page.



- Inventory
  - The **constructor** initializes the list of Items
  - **HasItem** checks if an input identifier exists in any existing Item (using AreYou)
  - Items can be added using Put, or removed by id using Take
  - Fetch locates an item by id (using AreYou) and returns it

Inventory Unit Tests	
Test Find Item	The Inventory has items that are put in it.
Test No Item Find	The Inventory does not have items it does not contain.
Test Fetch Item	Returns items it has, and the item remains in the inventory.
Test Take Item	Returns the item, and the item is no longer in the inventory.
Test Item List	Returns a string containing multiple lines. Each line contains a tab-indented short description of an item in the Inventory.

- Create the above unit test cases for the Inventory.

When you arrive at your lab, you will receive the verification tasks.
See you very soon.