SIT771 Object Oriented Development

Pass Task 1.2: Working with Objects

Overview

Now that you have everything setup we can start to explore **objects**. Object oriented programming is all about working with objects that know and can do things.

Activity 3 in Course 1 should provide you with the details you need to complete this task.

Submission Details

For this task you need to create a program that opens two Windows and creates within these windows two different pictures using basic shape drawing methods on the Window object.

Submit the following files to OnTrack.

- Your program's source code (*Program.cs*)
- A screen shot of your program running

Check the following things before submitting:

- Your code layout matches the expected format.
- You are using the write case for Classes and Methods (both PascalCase) and variables (camelCase).

You want to focus on the following key ideas, and make sure you can explain them in relation to your program.

- Classes
- Methods
- Main
- Sequence
- Object creation

Instructions

For this task you will start to make use of the classes that we provide in SplashKit. We can use these classes to create and open a windows, show bitmaps, play sound effects and more.

1. Start by creating a new folder to store your project's code. The following shows the instructions for doing this in the terminal.

```
cd /c/Users/andrew/Documents/Code
mkdir WorkingWithObjects
cd WorkingWithObjects
```

2. Get **skm** to create a new .NET project for you:

```
skm dotnet new console
skm dotnet restore
```

3. Open **Visual Studio Code** and open the folder you created.

We are now ready to build the program.

Step 1: Creating a Window

In SplashKit you can open a Window to draw on and interact with. To open the window, create a *new* Window **object**, and pass it the window's title, its width and its height. For example `new Window("House Drawing", 800, 600);` will create a new Window object. When the object is created the Window will open and use the things the object knows to setup its layout. So the window will be 800 pixels wide and 600 pixels high with the title "House Drawing". We can then tell this window to draw some shapes to create the picture shown in the following image.

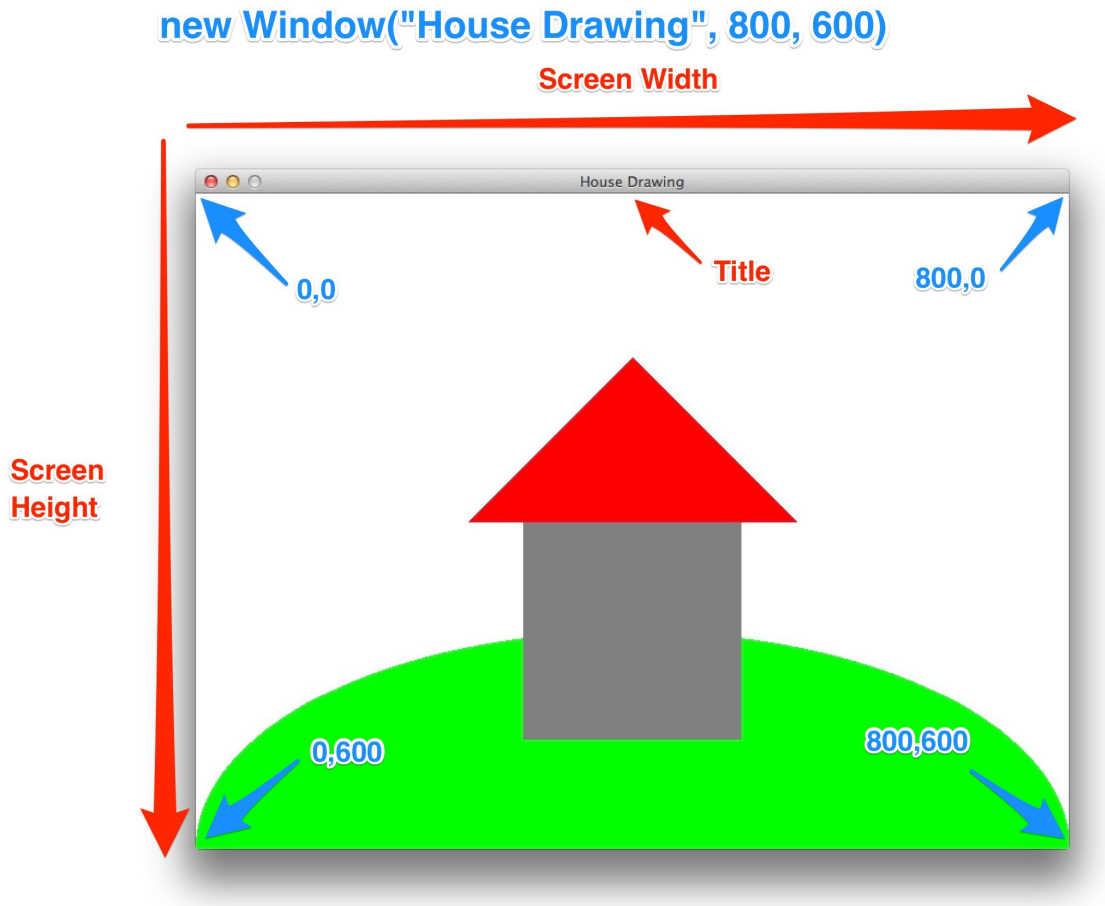

Figure: Window with dimensions illustrated

1. Lets get this started by creating a new Window, and using SplashKit to delay us for a few seconds. Give the following code a try:

```
using SplashKitSDK;

public class Program
{
    public static void Main()
    {
        new Window("Window Title... to change", 800, 600);

        SplashKit.Delay(5000);
    }
}
```

In this code `Window` is the name of a class, and we are using this class to create an **object**. The object knows to make a Window appear when it is created, so this is what we should see.

`SplashKit` is also a class - and classes can be used in a similar way to objects at runtime. The `SplashKit` class object knows how to `Delay`, so here we are asking it to delay the program for 5000 milliseconds (5 seconds).

2. Compile and run the program from the terminal.

```
skm dotnet run
```

You should see the window open, and the program delay for 5 seconds.

3. Change the window title to "Shapes by " and your name. For example, `"Shapes by Andrew"`.

Switch back to the terminal to compile and run your program.

Step 2: Drawing to a Window

Now that we have created an object, we need a way of accessing it. For this we can use a **variable** to remember the object we created. The code `Window shapesWindow;` creates a new variable called `shapesWindow` that can *refer* to a `Window` object. The assignment statement (i.e. `variable = value;`) is used to **assign** a value to the variable. In this case we can assign the variable a *reference* to the object we created using the `new Window(...)` code.

Once we have a reference to the object, via the variable, we can use the variable to tell the object to do things. In this case the Window object is able to do a range of drawing operations for us. So lets use it to draw some shapes to the window.

1. Adjust your `Main` method so that it now draws the house shown above.

```
public static void Main(string[] args)
{
    Window shapesWindow;
    shapesWindow = new Window("Shapes by ...", 800, 600);

    shapesWindow.Clear(Color.White);
    shapesWindow.FillEllipse(Color.BrightGreen, 0, 400, 800, 400);
    shapesWindow.FillRectangle(Color.Gray, 300, 300, 200, 200);
    shapesWindow.FillTriangle(Color.Red, 250, 300, 400, 150, 550, 300);
    shapesWindow.Refresh();

    SplashKit.Delay(5000);
}
```

2. Build and run the program.

```
skm dotnet run
```

   Make sure it all looks correct before moving on.

Step 3: You own drawing

Now that you have the house drawing, its time to try this out yourself.

1. Add some code, after the call to `Refresh` and before the `Delay` that will create a **new Window**. You come up with the title and size.
2. Use `Clear`, the shape drawing methods, and `Refresh` to draw your own custom picture.
3. Compile and run to test as you go.

```
skm dotnet run
```

4. When you are happy with your program, login to OnTrack and submit your code and screenshot to Pass Task 1.2.

Remember to save and backup your work! Storing your work in multiple locations will help ensure that you do not lose anything if one of your computers fails, or you lose a USB Key.

Tip:
Add a long delay so you can get a good screen shot!

> **Note:**
>
> This is another tasks you need to submit to OnTrack. Check the assessment criteria for the important aspect your tutor will check.

Graphics Primer

The images you see on your computer's screen are made up of dots called pixels: picture elements.

The screen has many pixels arranged into a grid (columns and rows), with each pixel having its own unique location (a combination of an `x` and `y` value, where `x` indicates the column and `y` the row), and color.

> **Note:**
>
> Programming languages and terminology tend to use American spelling. So we use `Color`, not `Colour`.

The following image shows an example of two rectangles (one filled, one outlined). The top left corner of the screen is at row (x) 0 and column (y) 0, and these numbers increase as you go to the right and down the screen. The call to `FillRectangle` and `DrawRectangle` take in a color, a `x` value, a `y` value, and a `width` and a `height`. So the blue rectangle is filled at x 1, y 1, is 7 pixels wide, and 3 pixels high.

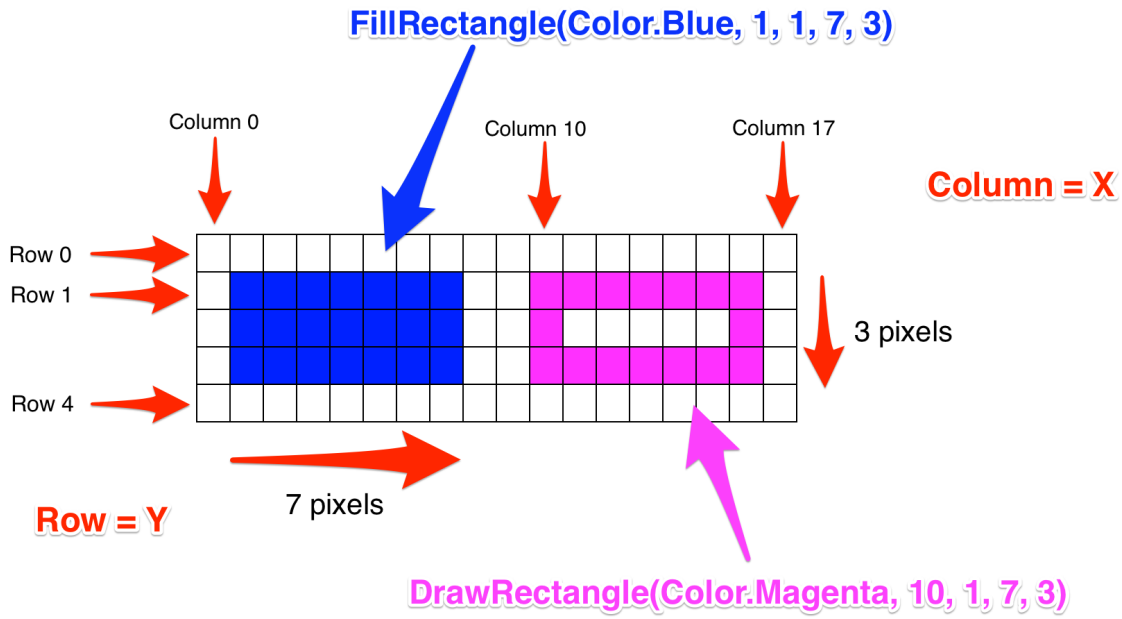

Figure: Pixel locations are based on x and y locations.

Positions on the screen are determined using two values, one for x and the other for y. The x value determines the number of pixels from the left side of the screen. The y value determines the number of pixels from the top of the screen.

For example: the magenta rectangle is drawn at 10, 1. This means its x value is 10 and its y is 1. This rectangle is drawn 10 pixels from the left of the screen, and its 1 pixel from the top.

To draw a shape on a Window with SplashKit you need start by telling the Window what color to draw it and where to draw it (i.e., the x y coordinates). Different shapes will require additional information such as width and height for rectangles and radius for circles.

All of the shape drawing operations in SplashKit take a number of parameter values:

- The **color** to draw the shape. SplashKit has a number of colors you can use. The Color class object knows a wide range of colors, so you can access them from the Color class, for example: `Color.White`, `Color.Green`, `Color.Blue`, `Color.Black`, `Color.Red`, and many others. VS Code should show you a list once you type `Color.`

- An **x** value, representing the x position of the shape (column).

  This is a number of pixels from the left edge of the screen. Larger values are further to the right.
- A **y** value, representing the y position of the shape (row).

  This is a number of pixels from the top edge of the screen. Larger values are further down the screen.

- and other values for the size of the shape, these will differ depending on the kind of shape being drawn (e.g., rectangle has a width and height, as does ellipse).

The following Table shows the parameters for the different shapes you can draw with SlashKit, an example method call is also shown. You should be able to get Visual Studio Code to show you to list of parameters. There are usually a couple of different ways to draw each shape, by pressing the up down arrows you should be able to explore this list.

Method	Example
Clear(color)	`w.Clear(Color.White)`
DrawCircle(color, x, y, radius)	`w.DrawCircle(Color.Red, 50, 100, 25);`
FillCircle(color, x, y, radius)	`w.FillCircle(Color.Red, 70, 50, 20);`
DrawRectangle(color, x, y, width, height)	`w.DrawRectangle(Color.Green, 100, 150, 30, 60);`
FillRectangle(color, x, y, width, height)	`w.FillRectangle(ColorGreen, 100, 160, 10, 40);`
DrawTriangle(color, x1, y1, x2, y2, x3, y3)	`w.DrawTriangle(ColorBlue, 150, 100, 150, 200, 175, 200);`
FillTriangle(color, x1, y1, x2, y2, x3, y3)	`w.FillTriangle(ColorBlue, 155, 135, 155, 195, 170, 195);`
DrawEllipse(color, x, y, width, height)	`w.DrawEllipse(Color.Black, 200, 100, 50, 160);`
FillEllipse(color, x, y, width, height)	`w.FillEllipse(ColorBlack, 210, 110, 30, 140);`
DrawLine(color, x1, y1, x2, y2)	`w.DrawLine(ColorMagenta, 0, 300, 800, 300);`

To draw a picture, like the house shown above, the computer executes the code to draw the individual shapes one at a time in the order they appear in the code. However, we don't want each element to appear individually, we just want the whole picture to appear at once, so the whole house should show all at once. SplashKit uses a technique called **Double Buffering** to enable this. When double buffering, the computer first draws the shapes, then waits for a command to display the shapes to the user. With

SplashKit, the shapes are all shown together when you tell the Window to Refresh. This is illustrated below.

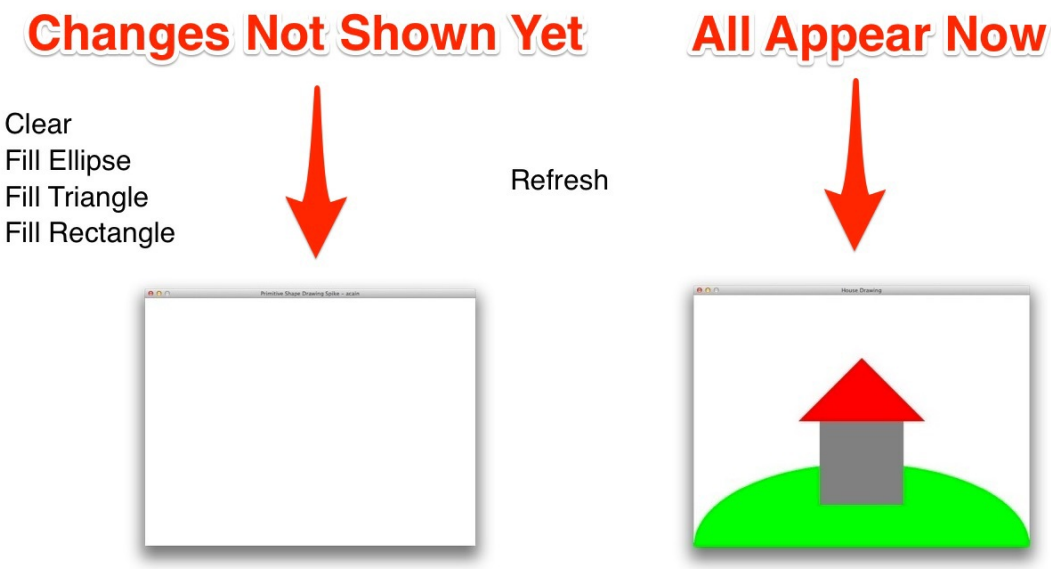

Figure: Illustration of double buffering, adn the need to refresh screen