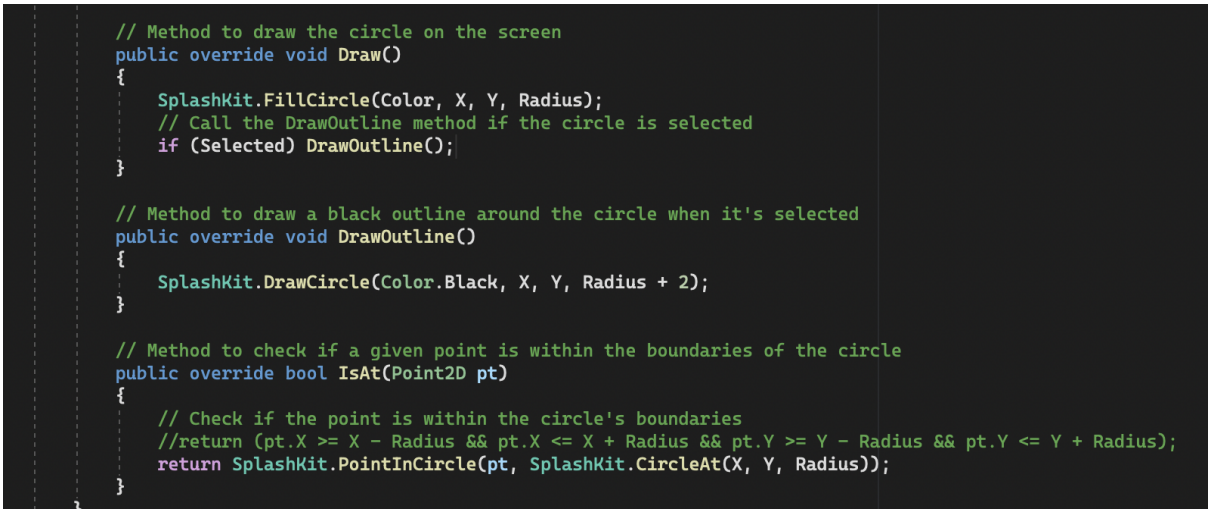
The 4 core concepts of Object Oriented Programming

1. Abstraction - It involves recognising patterns and structures as classes and objects then designing a simplified code representation that reflects what it knows and what it should be able to do in real life hence hoping that only essential features/details are left. Using abstraction, we hope to reduce complexity by breaking down the code into classes and objects that can be managed individually, while still being detailed enough to perform their designated functions.

Example of Abstraction -
In the ShapeDrawer, I have different classes representing different shapes (circles, rectangles, and lines). Each of the classes abstracts the idea of a shape. They encapsulate the properties and behaviour of those shapes.

In the MyCircle class, these methods abstract the drawing and interaction behaviour of a circle without specifying how they are implemented in detail. They provide a view of how the circle should be drawn , when an outline is drawn and when checking if a point is within its boundaries



```
// Method to draw the circle on the screen
public override void Draw()
{
    SplashKit.FillCircle(Color, X, Y, Radius);
    // Call the DrawOutline method if the circle is selected
    if (Selected) DrawOutline();
}

// Method to draw a black outline around the circle when it's selected
public override void DrawOutline()
{
    SplashKit.DrawCircle(Color.Black, X, Y, Radius + 2);
}

// Method to check if a given point is within the boundaries of the circle
public override bool IsAt(Point2D pt)
{
    // Check if the point is within the circle's boundaries
    //return (pt.X >= X - Radius && pt.X <= X + Radius && pt.Y >= Y - Radius && pt.Y <= Y + Radius);
    return SplashKit.PointInCircle(pt, SplashKit.CircleAt(X, Y, Radius));
}
```

---

2. Encapsulation - Encapsulation wraps up data under a single unit. It is the mechanism that binds together code and the data it manipulates. It is a protective shield that prevents the data from being accessed by the code outside this shield. In encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding. It can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

Example of encapsulation -
An example from the ShapeDrawer assignment.
Private fields: _color, _x, _y and _selected. These fields hold the internal state of a shape.
Public fields: X, Y, Color and Selected. They provide controlled access to the private fields and allow the external code to get or set their values.

```
// Property to get or set the X-coordinate of the shape
public float X
{
    get { return _x; }
    set { _x = value; }
}
```

---

3. **Inheritance** - The child class inherits all of the features of the parent. It can inherit the position, size and can be drawn. It helps bring flexibility, extensibility and adaptability to OOP programs. The child class can't access private or protected fields of the parents.

Example of inheritance:
In the ShapeDrawer assignment, the MyCircle class is a derived class of the Shape class. MyCircle inherits all the properties and methods defined in the Shape class

```
namespace ShapeDrawer
{
    public class MyCircle : Shape
    {
        private int _radius;
```

---

4. **Polymorphism** - Poly Morph = Many Forms. This concept relates to a single object with many forms; one way to achieve this is using inheritance.This means the child class can also be handled as its parent type in the program, and programmers can use it in place where functions accept the parent type.

Example of polymorphism:

In the ShapeDrawer assignment, in Drawing.cs, it calls the ‘Draw’ method on each shape in the ‘_shapes’ list. Polymorphism is achieved in this example through inheritance and method overriding.

```
// Tells SplashKit to clear the screen to the background color
public void Draw()
{
    SplashKit.ClearScreen(_background);
    foreach (Shape shape in _shapes)
    {
        shape.Draw();
    }
}
```