6.2P Key Object-Oriented Concepts

Four concepts:

Encapsulation

- Encapsulation can be referred to the use of public and private onto the classes. When declared as private the data may be not be seen by any other classes and only by itself meanwhile public classes allow us other classes to view and most importantly enable us to modify the data when in use of inheritance or polymorphism.

An example that I can take from what we learn so far would be the (shapesdrawer4): ShapeCount which in Drawing is a read-only and is calculated inside by the _shape.Count which prevent any external modification. In drawing class we have _shapes and _background are private fields in removeshapes and selectShapesAt. The list can be access internally or controlled methods.

Abstraction

- Abstraction, also known as data hiding in other words. Helps hide unnecessary information and only display information that we the users are interacting. It relates very much with encapsulation as it restricts direct contact with data meanwhile abstraction will control the interface interacted by the object or users.

An example from what I learnt (Shapedrawer4): in shape.cs we have private classes for _color, _x, _y, _width, _height, so on. These private properties are hidden to and cannot be accessed by an outside, each shape classes will internally manage and control how it is drawn or selected.

Inheritance

- As a sub class, Inheritance allows the user to gain access to methods and variables declared in mother class which is derived from. In development of software this method is useful as it saves memory and time.

An example from what I learnt (Shapedrawer4): In MyLine, MyRectangle, MyCircle we have inherited from Shape like for example

```
5 references
public class MyLine : Shape
```

this allow the derived classes like the one I stated above to have its mother classes properties and methods. As a derived class it can take it's own shape and the mother class which is the base class is able to retrieve any additional methods or properties call by it derived

Polymorphism

- Polymorphism only will occur when the inheritance is declared onto the classes because the with inheritance, we are able to override the methods that is once declared by the mother class. From what I understand of polymorphism is the use override function and methods which is derived from the mother class so that it can be its own identity. It stores its own variables inside its own class which can only be accessed by the mother class and not any other sub-classes derived from mother class

An example from what I learnt (Shapedrawer4): now for polymorphism to occur we also need inheritance as shown above we have successfully created a base and derived classes and now we want to access base class methods and modify it in each of our own derived class.

```
    }
    6 references
    public abstract void DrawOutline(); //draw outline to show that its highlighted
    7 references
    public virtual void SaveTo(StreamWriter _writer)
    {
        _writer.WriteColor(_color);
        _writer.WriteLine(X);
        _writer.WriteLine(Y);
    }

    7 references
    public virtual void LoadFrom(StreamReader _reader)
    {
        Color = _reader.ReadColor();
        X = _reader.ReadInteger();
        Y = _reader.ReadInteger();
    }
```

in our base class we have to set it to abstract (cannot have a body on it's own)or virtual

```
2 references
public override void Draw()
{
    if (Selected)
    {
        DrawOutline();
    }
    SplashKit.DrawLine(Color, X, Y, EndY, EndX);

}

2 references
public override void DrawOutline()
{
    SplashKit.DrawRectangle(SplashKitSDK.Color.Black, X - 2, Y - 2, EndY + 4, EndX + 4); // No need for SplashKit prefix
}

2 references
public override bool IsAt(Point2D point)
{
    return SplashKit.PointOnLine(point, SplashKit.LineFrom(X, Y, EndY, EndX));
}
5 references
public override void SaveTo(StreamWriter _writer)
{
    _writer.WriteLine("Line");
    base.SaveTo(_writer);
    _writer.WriteLine(EndY);
    _writer.WriteLine(EndX);
    _writer.WriteLine($"{(int)(Color.R * 255)},{(int)(Color.G * 255)},{(int)(Color.B * 255)}");
}
5 references
```

This will allow the derived classes to have the base class properties and we can have new set of method for each classes without affecting other classes (derived class cannot have contact with other derived class unless allowed) which then the base can access all of it

Concept Map