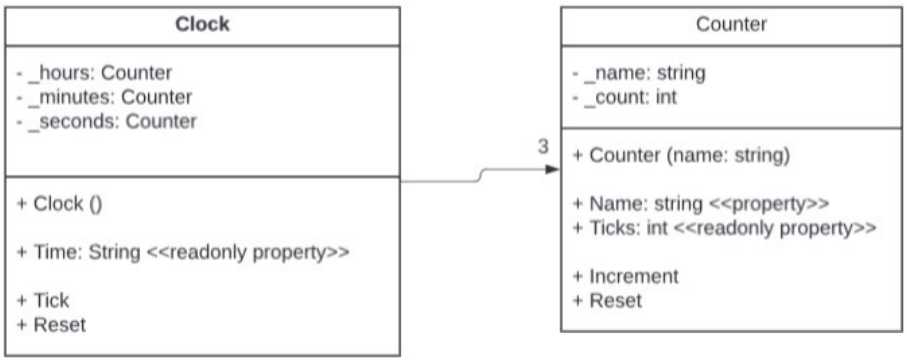
6.2P - Key Object Oriented Concepts

Jayden Kong, 104547242

1. Abstraction

Abstraction is breaking down a system into different roles, each with distinct “attributes and interactions” (fields and methods), to define classes that make up the system [1].

This can be shown in the clock task (3.1P):

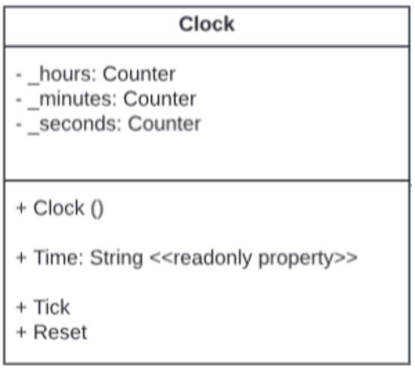


The clock program can be broken down into a Clock class and a Counter class. The Clock class has the role of coordinating three Counter objects to keep track of time. It knows the values of its Counter objects, hours, minutes, and seconds, and can be told to Tick (increment time by one second) or to Reset (reset all Counter objects to zero) The Counter class has the role of representing a generic counter the is able to count upwards in increments of one. It knows its own name, and its count, and can be told to Increment (increase its count by one) or to Reset (reset its count to zero).

2. Encapsulation

Encapsulation refers to the grouping of data and functions that operate on the data into individual units (a class). The data is stored in the class’s fields, while the functions that operate on the data are defined as the class’s methods. These data or methods can be indicated as accessible or not accessible by declaring them as public or private respectively [1]. In this way, encapsulation allows for the grouping of fields and methods into classes, whilst also controlling how accessible the data and methods of the class are to the rest of the program.

This can be shown with the Clock class created for the clock task (3.1P):

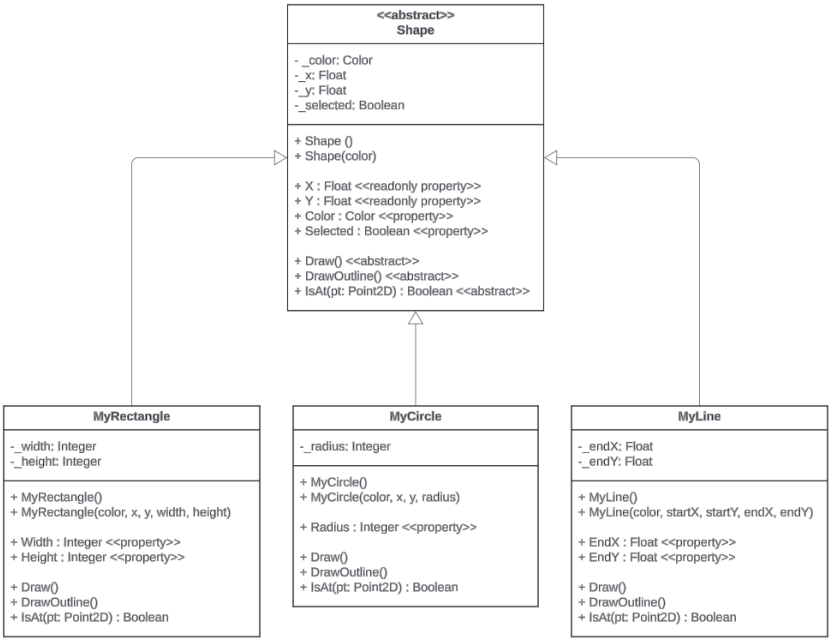


The Clock class groups together data in the form of Counter objects for its hours, minutes, and seconds, along with functionality that works with the data in the methods, Tick and Reset. The hours, minutes and seconds fields have also been declared as private, whilst the Tick and Reset methods have been declared as public. This means that the hours, minutes, and seconds fields are not accessible outside the Clock class, whereas the Tick and Reset methods can be accessed outside the Clock class.

3. Inheritance

Inheritance allows for the creation of new abstractions by utilising a generic foundational class (the parent class) as the base [1]. This enables the development of a hierarchy of more specialised classes (child classes) where each shares common characteristics inherited from the generic parent class, while being able to have their own unique attributes and behaviours.

This can be shown through the shape drawer task (4.1P):



In the shape drawer, the Shape class acts as the general parent class to the child classes: MyRectangle, MyCircle, and MyLine. Each child class inherits certain attributes such as an x- and y-coordinate, and behaviours such as being able to be told to Draw. However, each child class can introduce more specialised attributes that relate to their specific shape, such as a width and height for MyRectangle, a radius for MyCircle, or ending x- and y- coordinates for MyLine.

4. Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass. This allows for child objects to implement the methods inherited from a parent class in different ways. This is achieved through method overriding in the child classes [1].

This can be shown through the shape drawer task (4.1P):

In parent Shape Class:

```
74          public abstract void Draw();
```

In child MyRectangle:

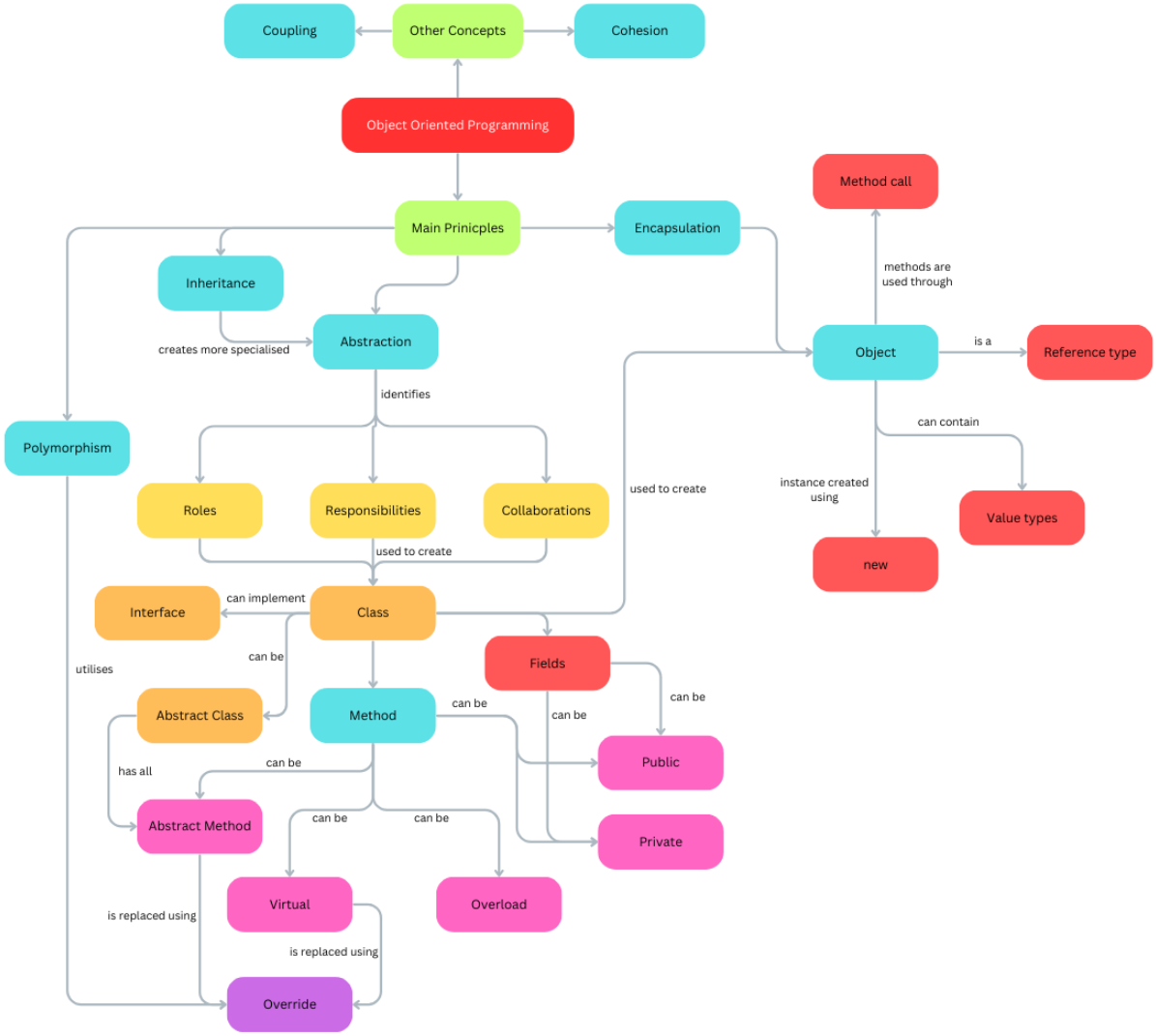
```
48          public override void Draw()
49          {
50              if (base.Selected)
51              {
52                  DrawOutline();
53              }
54
55              SplashKit.FillRectangle(base.Color, X, Y, _width, _height);
56          }
```

In child MyCircle:

```
35          public override void Draw()
36          {
37              if (Selected)
38              {
39                  DrawOutline();
40              }
41
42              SplashKit.FillCircle(base.Color, X, Y, _radius);
43          }
```

The child classes, MyRectangle and MyCircle, both inherit the Draw method from the parent Shape class. However, MyRectangle and MyCircle have both used method overriding to change the Draw method to be able to draw their respective shapes. In this way, MyRectangle and MyCircle can be treated uniformly as Shape objects, where when the Draw method is called, they each perform their specific implementation of the Draw method to draw either a rectangle or a circle.

Concept Map



References

[1] Microsoft, "*Object-Oriented Programming (C#)*", learn.microsoft.com, https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop (accessed Apr. 11, 2024).