OOP - Principles

# Data

Data is information that is broken down and can be stored as different types i.e. Strings, int, char,

floats and double

# Methods

Methods are functions declared by an object and therefore encapsulates like the shape drawing task

has the draw function the user just needs to call the function and not know how it works

# Classes

Classes are a blueprint that are used to make an object

# Abstraction

Abstraction is the concept of making code simple to understand by hiding the parts we don’t need to know about and only show the users the information they need to know to about in order to use the classes we create.  
In the Clock task (3.2) we have a tick method that can be called when the clock object is created, if a user was to use the clock class we create they would not need to know how the tick works just that they can call the method to increase the hours, mins and secs.

# Encapsulation

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. This reduces the dependencies among separate classes

When we create the counter class (2.1) we create private variables with public methods which when we transfer the class to our clock task (3.2) we can use with no change to the class.

# Inheritance

Allows related classes to acquire properties and fields from another and allows for reusability, by

allowing different classes to share common traits this is seen in 4.1 where the shape (parent class)

shares its information with its child class wither circle or rectangle because both circle and rectangle

are a shape and have the same properties as its position or colour

# Polymorphism

Polymorphism is through op principle of taking on many forms this can be seen when a class uses the same method name to perform different task depending on what parameters are passed (method overloading), or when a child class overrides a parent class depending on which child class makes the call  
method overloading can be seen in shape drawing task (2.2) the splashkit draw rectangle can be overloaded with different values like alpha depending on if the value is passed and method overriding is can be see used in 4.1 where we use the IsAt function where different shapes can call the isat function from the shape parent class depending on which child shape class calls it

# Responsibilities

they are related to an object being able to do something or know something. What does it need to do? What does it need to know?

This is an OOP design principle where an object created should know what it needs to do and has all the relevant information to do what it was created to do  
for example in iteration 3 (5.1) the bag class needs to be able to use the inventory class which store a list of items

# Collaborations

the idea that objects must communicate with one another example in iteration 3 (5.1) the bag class needs to be able to use the inventory class which store a list of items which shows bag, item and inventory collaborating with each other to achieve the desired results

# Coupling

Coupling is the idea of how much a class is dependent on another class the “Tightly” related class is

bad as this means a change in one class will have a higher impact on other code “loosely” related

class will have zero to no impact on classes  
any time we use the splashkit lib we create a highly coupled program as depending on if we use methods from splashkit like point2d or splashkit rectangle it means that the user will also need to have splashkit installed to use our classes

# Cohesion

Cohesion is the focus of a singular classes purpose, the higher the cohesion the better the classes is

as it can be easily transferred to another program which means it has higher maintainability which helps When we create the counter class (2.1) we transfer the class to our clock task (3.2) we can use with no change to the class.