# Unit 3: Fundamentals of Object-Oriented Design

What does Object Orientated Programming (OOP) mean? An object in OOP is a grouping of data an behaviour. An object is a model of a thing.

Object orientated programming is therefore the process of using the concept of objects to model a software system.

## Object orientated analysis and object orientated design

As well as OOP there are other OO activity that can be used to help design a system, two of these are object orientated analysis and object orientated design.

The goal of object orientated analysis (OOA) is to create a set of requirements for the system. In OOA this is done by identifying the objects in the domain and how they interact.

The goal of object -orientated design is to turn a set of requirements into an implementation specification. The specification will describe the classes required to implement the system and the ways in which they interact.

OOP is the final stage in this process during which the design is implemented as code. In reality, the three-process described here are not executed perfectly sequentially instead overlap. This is because often during later stages we realize further investigation was required in earlier stages.

## Objects

### Data

The data of an object defines the individual characteristics of an object. A class defines what characteristics an object will have and an instance of the class (an object) will have specific values.

## Behavior

An object's behavior is something that it can do. It is through an object's behavior that various objects can interact with each other.

### Public interface

The public interface of an object is the set of attributes and methods that are available to interact with the object. The public interface does not expose everything that the object does it merely provides a model for interactions.

The hidden behaviors, for example, calculations to perform methods of the public interface, are hidden. This is known by the name's information hiding or encapsulation.

### Abstraction

Abstraction is the act of hiding the irrelevant low-level detail of a class. Abstraction is used when a public interface hides the inner workings. Abstractions help to reduce the cognitive load required when interacting with an object. To abstract is to create a model of a “real concept”.

## Composition and Inheritance

Objects in an OOP system can represent more concepts than objects in the real world. Abstract concepts can be modelled using OOP. For example, a real-world concept is a phone, but in OOP an abstract concept we could model would be a “phone call”.

A composite object is an object composed of multiple other types of objects. On object that “has-a” another object is composed of that object.

### *Composition vs aggregation*

Composition is a strong relationship. The parent object is in full control of the composed object. The composed object cannot exist by itself.

An aggregate relationship is a weak relationship. For example, a bus has people on board, but it does not own them. The people on the bus can exist without the bus. An aggregate relationship can be temporary and could be stored as a dynamic list of references.

### *Inheritance*

An inheritance relationship is an “is-a” relationship. A class that inherits from another class is of the same type as the base class. The derived class must be able to do everything its parent class could do (this idea is described by Liskov’s substitution principle). As such the public interface of the derived class must contain at least the public interface of the base class and optionally more. A derived class inherits the data and behavior of the base class.

### *Multiple inheritance*

Inheriting from multiples classes can be tricky. Problems can occur if multiple bases classes share identical method or attribute signatures. Languages that offer multiple inheritance provide mechanisms to specify which method/attribute the developer wants to call. Multiple inheritance if not offered by many languages, notably C#, which goes to show it is not needed, and in my opinion, should normally be avoided.

### *Abstract classes*

An abstract class is a class that defines what attributes and behaviors must be implemented by a derived class but does not itself have to implement them. Abstract classes cannot be instantiated.

### *Interfaces*

An interface is similar to an abstract class, but whereas the abstract class can supply default implementations of methods, an interface cannot implement any methods.

### *Polymorphism*

Polymorphism is the ability to call the same function of different objects and get different results. As long as all objects implement the interface of the base class then a function could invoke a method on the objects without knowing how it was implemented.

### *Duck-typing*

Python uses ducking, where different classes don’t have to be defined upfront. With duck-typing behavior can be added to objects during execution. Arguments in python cannot require specific types, as such anything which implements the required interface will be accepted even if it was not the type anticipated? Duck-typing allows for great software extensibility.