# Principles of Object Orientated Programming

OOP is a programming paradigm based around the use of Objects – which contain both data and the code for manipulating the data. An object is a data structure which has both fields (also known as members, attributes or properties) and methods (actions, subroutines or procedures).

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well suited for software that is large, complex and actively updated or maintained.

A **Class** is a detailed template and definition of what an object will be, but it is not the object itself. It is a user-defined data type, which contains all the data associated with it and all the functions to manipulate that data. To access that data we must instantiate a object from the class. It is the blueprint of any object and used to create them.

Once we have written a class and defined it, we can use it to create as many objects based on that class as we want. For example, consider the**Class** of**Animal**, with properties such as Name, Genus and Habitat. There will be many different animals, each with different values for these properties, each a different instance of the same class differing only by the data stored within the properties.

**Object** is an instance of a class. All the variables and functions of the class can be accessed through the object. A class does not take up memory until it is instantiated (i.e. an object is created). Following on with the example of an Animal class, example objects would be Cat, Dog, Whale or House Fly, each with different property values but still with the same overall structure.

## Key Features of OOP

*The following are the four fundamental concepts of Object-Oriented Programming.*

**Inheritance** – we can reuse code and properties from other objects. Related objects are able to use the properties of a parent object while extending or replacing other functionaliy.

**Polymorphism** – The word **polymorphism**means having **many forms**, and it comes from the Greek words **poly (many)** and **morph (forms).** Polymorphism allows us to make multiple classes conform to a set of requirements, allowing these classes to be used interchangeably in your code without having to know what the class will do. Each class provides its own internal behaviour to a given set of methods. This allows you to add new conforming classes in the future without having to change your existing code.

**Encapsulation** – All information is stored within the object, and only selected information is available externally to it. Other objects are not allowed to make changes to the internal data and must ask the object to make any changes through the use of public properties and methods. This provides greater data security and integrity as the object can determine if any data change is appropriate and allowable.

**Abstraction** – a technique where all but the required data about an object is hidden to reduce complexity and increase efficiency. For example, in order to store an object a unique identifier may be required, but this is not need to show the object to a user, so I can be hidden inside the object – used when needed but not used in any user interface.

## What are the benefits of OOP?

Benefits of OOP include the following:

* **Modularity.** Objects can be self-contained, making troubleshooting and collaborative development easier. There is no need to worry about the internal changes to an object requiring changes to other objects if the interface between them does not change.
* **Reusability.** Code can be reused. The nature of OOP means we can inherit functionality from parent objects without having to re-write it. Obviously, when we only have one set of code to complete one job, the developers can reduce the workload.
* **Productivity.** The features of OOP such as modularity and reuse can reduce the complexity of systems, the amount of code that needs to be written and cost of testing the reduced code base. This means programmers can be more productive.
* **Easily upgradable.**With well-defined interface between objects, we can upgrade a single module without effecting other modules. For example, we can upgrade to a more efficient search routine without changing the surrounding code.
* **Security.** Using encapsulation and abstraction, complex code is hidden and cannot be manipulated by buggy code. As access to an objects data is protected by its methods the integrity of its data can be protected by the object its self.
* **Flexibility.** Polymorphism enables a single function to adapt to the class it is placed in while the same interface as other classes. This allows us to define simpler interfaces that can be used for a greater range of situations instead of a series of more different, highly targeted interfaces.
* **Code maintenance.** Parts of a system can be updated and maintained without needing to make significant adjustments.