Introduction to Object Oriented Programming

Classes and Objects are basic concepts of Object Oriented Programming which revolve around real-life entities.

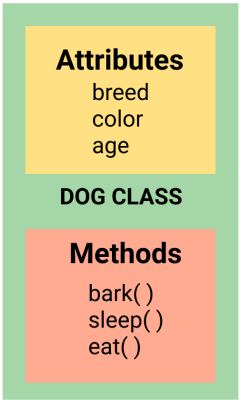
Class

A class is a user-defined blueprint or prototype from which real-world objects are created. It represents the set of properties or methods that are common to all objects of one type.

We can call class as a collection of objects, which is a logical entity and does not take space in the memory.

Example :

Dog Class can be represented as shown in the diagram below.



Object

It is a basic unit of Object Oriented Programming and represents the real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods.

Objects having memory addresses and take up some space. They can interact with each other without knowing the data and code.

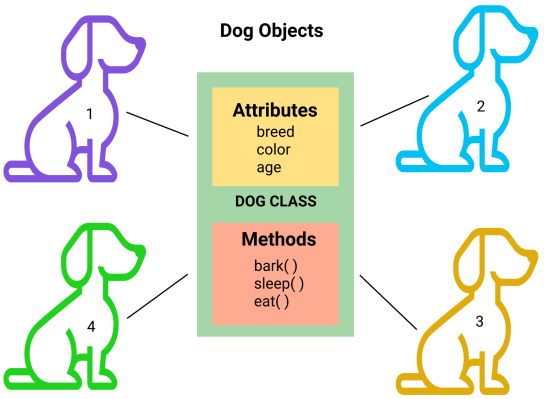
An object consists of:-

- **State** : It is represented by the attributes of an object. It also reflects the properties of an object.

- **Behavior** : It is represented by the methods of an object. It also reflects the response of an object with other objects.
- **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

Example :

Dog objects created from Dog Class



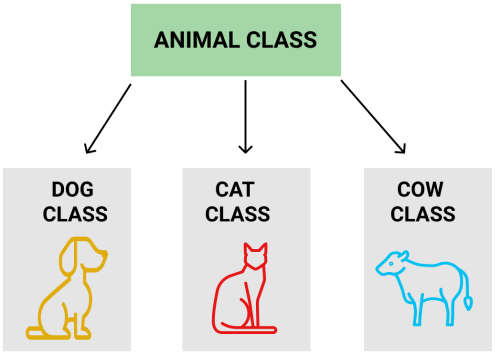
Inheritance

The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important feature of Object Oriented Programming.

- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:** The class whose properties are inherited by sub class is called Base Class or Super class.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example :

Dog, Cat, and Cow can the be derived classes of Animal base class.



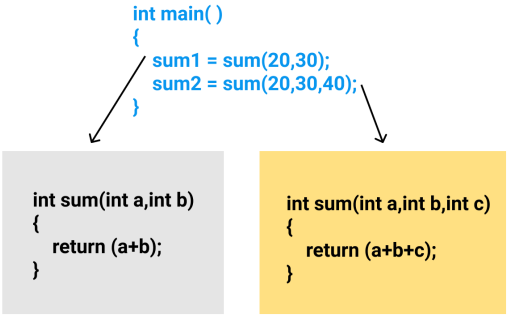
Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

A real-life example of polymorphism is that a person at the same time can have different characteristics. Like a man at the same time can be a father, a husband, and an employee. So the same person posses different behavior in different situations. This is called polymorphism.

Example:

Suppose we have to write a function to add some integers, some times there are 2 integers and some times there are 3 integers. We can write the Addition Method with the same name having different parameters, the concerned method will be called according to parameters.



Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. **Ex: A car is viewed as a car rather than its individual components.**

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object and ignoring the irrelevant details. The properties and behaviors of

an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a **real-life example** of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying the brakes will stop the car, but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes, etc in the car. This is what abstraction is.

Advantages of Abstraction

- It reduces the complexity of viewing things.
- Avoids code duplication and increases reusability.
- Helps to increase the security of an application or program as only the important details are provided to the user.

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together the code and the data it manipulates. Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from the other classes, so it is also known as **data-hiding**.
- Encapsulation can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of the variables.

Advantages of Encapsulation:

- **Data Hiding:** The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables. He only

knows that we are passing the values to a setter method and that the variables are getting initialized with that value.

- **Increased Flexibility:** We can make the variables of the class as read-only or write-only, depending on our requirements. If we wish to make the variables as read-only then we have to omit the setter methods such as setName( ), setAge( ) etc. or if we wish to make the variables as write-only then we have to omit the get methods such as getName( ), getAge( ), etc.
- **Reusability:** Encapsulation also improves the re-usability and makes the code easy to change with new requirements.
- **Testing code is easy:** Encapsulated code is easy to test for unit testing.

Encapsulation vs Data Abstraction

- Encapsulation is data hiding(information hiding) while Abstraction is detail hiding(implementation hiding).
- While encapsulation groups together the data and the methods that act upon the data, data abstraction deals with exposing the interface to the user and hiding the details of the implementation.