Object oriented programming

OOPS concepts are as follows:

1. [Class](https://www.geeksforgeeks.org/classes-objects-java/)
2. [Object](https://www.geeksforgeeks.org/classes-objects-java/)
3. [Method](https://www.geeksforgeeks.org/methods-in-java/)and [method passing](https://www.geeksforgeeks.org/message-passing-in-java/)
4. Pillars of OOPs
   * [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)
   * [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)
   * [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)
   * [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-java/)
     + Compile-time polymorphism
     + Runtime polymorphism

OOPS concepts are as follows:

1. [Class](https://www.geeksforgeeks.org/classes-objects-java/)
2. [Object](https://www.geeksforgeeks.org/classes-objects-java/)
3. [Method](https://www.geeksforgeeks.org/methods-in-java/)and [method passing](https://www.geeksforgeeks.org/message-passing-in-java/)

Pillars of OOPs

* + [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)
  + [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)
  + [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)
  + [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-java/)
    - Compile-time polymorphism
    - Runtime polymorphism

1.Class :- A [class](https://www.geeksforgeeks.org/classes-objects-java/)is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

2. object:- n [object-oriented programming (OOP)](https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP), objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.

3. method:- A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything.

Pillar 1: Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or non-essential 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, 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 object.  
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the car speed or applying brakes will stop the car, but he does not know 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 the accelerators, brakes etc. in the car. This is what abstraction is.   
In Java, abstraction is achieved by [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) and [abstract classes](https://www.geeksforgeeks.org/abstract-classes-in-java/). We can achieve 100% abstraction using interfaces.

**Pillar 2:**[Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)

It 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 the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.
* In encapsulation, the data in a class is hidden from other classes, which is similar to what **data-hiding** does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.
* Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.

**Pillar 3:**[Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features (fields and methods) of another class.

Let us discuss some frequently used important terminologies:

* **Superclass:**The class whose features are inherited is known as superclass (also known as base or parent class).
* **Subclass:** The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **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.

**Pillar 4:**[**Polymorphism**](https://www.geeksforgeeks.org/polymorphism-in-java/)

It refers to the ability of object-oriented programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.