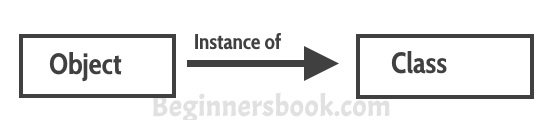
**OOPs concepts in Java**

Object-oriented programming System(OOPs) is a programming paradigm based on the concept of “objects” that contain data and methods. The primary purpose of object-oriented programming is to increase the flexibility and maintainability of programs. Object oriented programming brings together data and its behaviour(methods) in a single location(object) makes it easier to understand how a program works.

## What is an Object

  
**Object:** is a bundle of data and its behaviour(often known as methods).

Objects have two characteristics: They have **states** and **behaviors**.

**Examples of states and behaviors**  
**Example 1:**  
**Object**: House  
**State**: Address, Color, Area  
**Behavior**: Open door, close door

class House {

String address;

String color;

double are;

void openDoor() {

//Write code here

}

void closeDoor() {

//Write code here

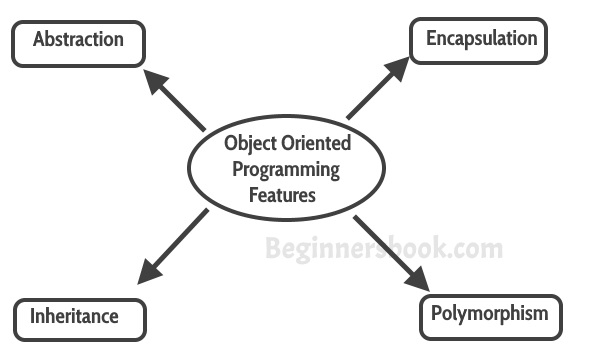
}

...

...

}

## Object Oriented Programming features

  
These four features are the main OOPs Concepts

### Abstraction

Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user. For example, when you login to your bank account online, you enter your user\_id and password and press login, what happens when you press login, how the input data sent to server, how it gets verified is all abstracted away from the you.

### Encapsulation

Encapsulation simply means binding object state(fields) and behavior(methods) together. If you are creating class, you are doing encapsulation.

### Inheritance

The process by which one class acquires the properties and functionalities of another class is called [inheritance](https://beginnersbook.com/2013/03/inheritance-in-java/). Inheritance provides the idea of reusability of code and each sub class defines only those features that are unique to it, rest of the features can be inherited from the parent class.

1. Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.
2. Inheritance allows us to reuse of code, it improves reusability in your java application.
3. The parent class is called the **base class** or **super class**. The child class that extends the base class is called the derived class or **sub class** or **child class**.

**Note:** The biggest advantage of Inheritance is that the code in base class need not be rewritten in the child class.  
The **variables** and **methods** of the base class can be used in the **child class** as well.

#### Syntax: Inheritance in Java

To inherit a class we use extends keyword. Here class A is child class and class B is parent class.

class A extends B

{

}

[Types of Inheritance](https://beginnersbook.com/2013/05/java-inheritance-types/):  
**Single Inheritance**: refers to a child and parent class relationship where a class extends the another class.

**Multilevel inheritance**: refers to a child and parent class relationship where a class extends the child class. For example class A extends class B and class B extends class C.

**Hierarchical inheritance**: refers to a child and parent class relationship where more than one classes extends the same class. For example, class B extends class A and class C extends class A.

**Multiple Inheritance**: refers to the concept of one class extending more than one classes, which means a child class has two parent classes. Java doesn’t support multiple inheritance.

Most of the new **OO** **languages** like Small Talk, Java, C# do not support Multiple inheritance. Multiple Inheritance is supported in C++.

### Polymorphism

[Polymorphism](https://beginnersbook.com/2013/03/polymorphism-in-java/) is a object oriented programming feature that allows us to perform a single action in different ways. For example, lets say we have a class Animal that has a method animalSound(), here we cannot give implementation to this method as we do not know which Animal class would extend Animal class. So, we make this method abstract like this:

public abstract class Animal{

...

public abstract void animalSound();

}

Now suppose we have two Animal classes Dog and Lion that extends Animal class. We can provide the implementation detail there.

public class Lion extends Animal{

...

@Override

public void animalSound(){

System.out.println("Roar");

}

}

and

public class Dog extends Animal{

...

@Override

public void animalSound(){

System.out.println("Woof");

}

}

[Types of Polymorphism](https://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/)  
1) Static Polymorphism  
2) Dynamic Polymorphism

#### Static Polymorphism:

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading can be considered as static polymorphism example.  
**Method Overloading**: This allows us to have more than one methods with same name in a class that differs in signature.

class DisplayOverloading

{

public void disp(char c)

{

System.out.println(c);

}

public void disp(char c, int num)

{

System.out.println(c + " "+num);

}

}

public class ExampleOverloading

{

public static void main(String args[])

{

DisplayOverloading obj = new DisplayOverloading();

obj.disp('a');

obj.disp('a',10);

}

}

**Output:**

a

a 10

When I say method signature I am not talking about return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

**Method overriding in java with example**

Declaring a method in **sub class** which is already present in **parent class** is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Method Overriding Example

Let’s take a simple example to understand this. We have two classes: A child class Boy and a parent class Human. The Boy class extends Human class. Both the classes have a common method void eat(). Boy class is giving its own implementation to the eat() method or in other words it is overriding the eat() method.

The purpose of Method Overriding is clear here. Child class wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Human is eating.

class Human{

//Overridden method

public void eat()

{

System.out.println("Human is eating");

}

}

class Boy extends Human{

//Overriding method

public void eat(){

System.out.println("Boy is eating");

}

public static void main( String args[]) {

Boy obj = new Boy();

//This will call the child class version of eat()

obj.eat();

}

}

Output:

Boy is eating

## Advantage of method overriding

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method **without even modifying the parent class code**.

This is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code.

## Interfaces in Java

An interface is a blueprint of a class, which can be declared by using **interface** keyword. Interfaces can contain only constants and abstract methods (methods with only signatures no body).Like abstract classes, Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces. Interface is a common way to achieve full abstraction in Java.

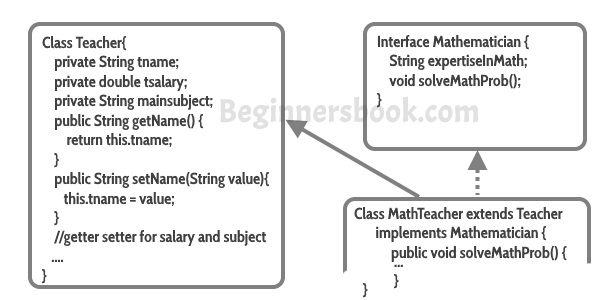
**Note:**

1. Java does not support Multiple Inheritance, however a class can implement more than one interfaces
2. Interface is similar to an abstract class but it contains only abstract methods.
3. Interfaces are created by using interface keyword instead of the keyword class
4. We use implements keyword while implementing an interface(similar to extending a class with extends keyword)

**Interface: Syntax**

class ClassName extends Superclass implements Interface1, Interface2, ....

### Example of Interface:



**Note**:

1. All **methods in an interface** are implicitly public and abstract. Using the keyword abstractbefore each method is optional.
2. An **interface** may contain final variables.
3. A class can **extend only one other class**, but it can **implement any number of interfaces.**
4. When a class implements an interface it has to give the definition of all the abstract methods of interface, else it can be declared as abstract class
5. An interface reference can point to **objects** of its implementing classes.

**Generalization and Specialization:**  
In order to implement the concept of inheritance in an OOPs, one has to first identify the similarities among different classes so as to come up with the base class.

This process of identifying the similarities among different classes is called **Generalization**. Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass. Shared characteristics can be attributes or methods.

In contrast to generalization, specialization means creating new subclasses from an existing class. If it turns out that certain attributes or methods only apply to some of the objects of the class, a subclass can be created.

#### Access Specifiers

Well, you must have seen public, private keyword in the examples I have shared above. They are called access specifiers as they decide the scope of a data member, method or class.

There are **four types** of access specifiers in java:  
**public:** Accessible to all. Other objects can also access this member variable or function.  
**private:** Not accessible by other objects. Private members can be accessed only by the methods in the same class. **Object accessible only in class in which they are declared.**  
**protected:** The scope of a protected variable is within the class which declares it and in the class which inherits from the class (Scope is class and subclass).  
**Default:** Scope is Package Level. We do not need to explicitly mention default as when we do not mention any access specifier it is considered as default.