Basic concepts of OOP: Encapsulation, Inheritance and Polymorphism

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

Benefits of Encapsulation

* The fields of a class can be made read-only or write-only.
* A class can have total control over what is stored in its fields.

**public** **class** EncapTest {

 **private** String name;

 **private** **int** age;

 **public** **int** getAge() {

 **return** age;

 }

 **public** String getName() {

 **return** name;

 }

 **public** **void** setAge( **int** newAge) {

 **if**(newAge > 20){

 age = newAge;

 }

 }

 **public** **void** setName(String newName) {

 name = newName;

 }

}

**public** **class** EncapMain {

 **public** **static** **void** main(String args[]) {

 EncapTest encap = **new** EncapTest();

 encap.setName("James");

 encap.setAge(22);

 System.***out***.print("Name is: " + encap.getName() + " Age is : " + encap.getAge());

 }

 }

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

## extends Keyword

**extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

### Why use inheritance in java

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).
* For Code Reusability.

**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");}

}

**class** TestInheritance{

**public** **static** **void** main(String args[]){

Dog d=**new** Dog();

d.bark();

d.eat();

}}

# Polymorphism in Java

**Polymorphism in Java** is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

 There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Why we need polymorphism?

A parent class reference is used to refer to a child class object.

**class** Bike{

 **void** run(){System.***out***.println("bike running safely with 60km");}

}

**class** Splendor **extends** Bike{

 **void** run(){System.***out***.println("splendor running safely with 60km");}

 **public** **static** **void** main(String args[]){

 Bike b = **new** Splendor();//upcasting

 b.run();

 }

}

Output: splendor running safely with 60km