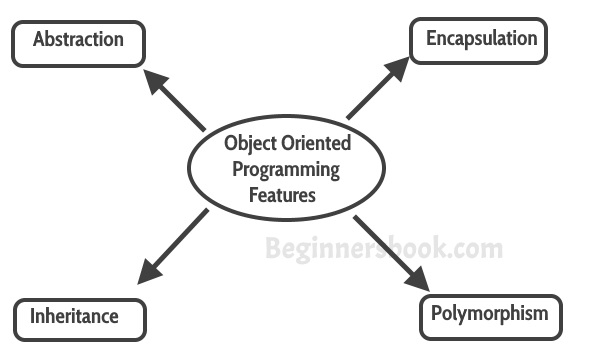
**What is OOPs Concepts in Java**

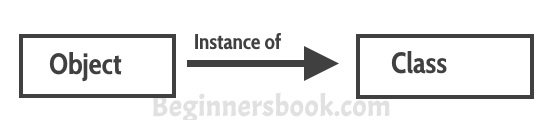
OOPs concepts includes following Object oriented programming concepts:

* Object
* Class
* Abstraction
* Encapsulation
* Inheritance
* Polymorphism



**1. Object**

An object can be represented as an entity that has state and behaviour. For example: A car is an object that has states such as color, model, price and behaviour such as speed, start, gear change, stop etc.



Let’s understand the **difference between state and behaviour.** The state of an object is a data item that can be represented in value such as price of car, color, consider them as variables in programming. The behaviour is like a method of the class, it is a group of actions that together can perform a task. For example, gear change is a behaviour as it involves multiple subtasks such as speed control, clutch, gear handle movement.  
Let’s take few more examples of Objects:

**Examples of states and behaviours**  
**Example 1:**  
**Class**: House  
**State**: address, color, area  
**Behaviour**: Open door, close door

Let’s see how can we write these state and behaviours in a java program. States can be represented as instance variables and behaviours as methods of the class.

class House {

String address;

String color;

double area;

void openDoor() {

//Write code here

}

void closeDoor() {

//Write code here

}

...

...

}

**Example 2:**  
**Class**: Car  
**State**: color, brand, weight, model  
**Behaviour**: Break,  Accelerate,  Slow Down,  Gear change.

**Note:** As we have seen in the above example, the states and behaviours of an object can be represented by variables and methods in the class.

**2. Class**

A class can be considered as a **blueprint** which **you can use to create as many objects as you like**. For example, here we have a class Website that has two data members. This is just a blueprint, it does not represent any website, however using this we can create Website objects that represents the websites. We have created two objects, while creating objects we provided separate properties to the objects using constructor.

public class Website {

//fields (or instance variable)

String webName;

int webAge;

// constructor

Website(String name, int age){

this.webName = name;

this.webAge = age;

}

public static void main(String args[]){

//Creating objects

Website obj1 = new Website("beginnersbook", 11);

Website obj2 = new Website("google", 28);

//Accessing object data through reference

System.out.println(obj1.webName+" "+obj1.webAge);

System.out.println(obj2.webName+" "+obj2.webAge);

}

}

**Output:**

beginnersbook 11

google 28

**3. 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. Read more about it here: [Abstraction in Java](https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/).

**Abstract Class Example:**

Here we have an abstract class Animal that has an abstract method animalSound(), since the animal sound differs from one animal to another, there is no point in giving the implementation to this method as every child class must override this method to give its own implementation details. That’s why we made it abstract.

Now each animal must have a sound, by making this method abstract we made it compulsory to the child class to give implementation details to this method. This way we ensures that every animal has a sound.

//abstract class

abstract class Animal{

//abstract method

public abstract void animalSound();

}

public class Dog extends Animal{

public void animalSound(){

System.out.println("Woof");

}

public static void main(String args[]){

Animal obj = new Dog();

obj.animalSound();

}

}

**Output:**

Woof

**4. Encapsulation**

[Encapsulation](https://beginnersbook.com/2013/05/encapsulation-in-java/) simply means binding object state(fields) and behaviour(methods) together. If you are creating class, you are doing encapsulation.

**Example**

1) Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.  
2) Have getter and setter methods in the class to set and get the values of the fields.

class EmployeeCount

{

private int numOfEmployees = 0;

public void setNoOfEmployees (int count)

{

numOfEmployees = count;

}

public double getNoOfEmployees ()

{

return numOfEmployees;

}

}

public class EncapsulationExample

{

public static void main(String args[])

{

EmployeeCount obj = new EmployeeCount ();

obj.setNoOfEmployees(5613);

System.out.println("No Of Employees: "+(int)obj.getNoOfEmployees());

}

}

**Output:**

No Of Employees: 5613

The class EncapsulationExample that is using the Object of class EmployeeCount will not able to get the NoOfEmployees directly. It has to use the setter and getter methods of the same class to set and get the value.

**What is the benefit of using encapsulation in java programming?**  
Well, at some point of time, if you want to change the implementation details of the class EmployeeCount, you can freely do so without affecting the classes that are using it.

**5. 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

{

}

**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.

**Inheritance Example**

In this example, we have a parent class Teacher and a child class MathTeacher. In the MathTeacher class we need not to write the same code which is already present in the present class. Here we have college name, designation and does() method that is common for all the teachers, thus MathTeacher class does not need to write this code, the common data members and methods can inherited from the Teacher class.

class Teacher {

String designation = "Teacher";

String college = "Beginnersbook";

void does(){

System.out.println("Teaching");

}

}

public class MathTeacher extends Teacher{

String mainSubject = "Maths";

public static void main(String args[]){

MathTeacher obj = new MathTeacher();

System.out.println(obj.college);

System.out.println(obj.designation);

System.out.println(obj.mainSubject);

obj.does();

}

}

**Output:**

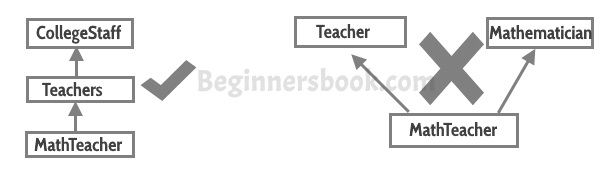
Beginnersbook

Teacher

Maths

Teaching

**Note: Multi-level inheritance** is allowed in Java but**multiple inheritance** is not allowed as shown in the following diagram.



[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, read more about it [here](https://beginnersbook.com/2013/05/java-multiple-inheritance/).

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

**6. 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, let’s 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");

}

}

As you can see that although we had the common action for all subclasses animalSound() but there were different ways to do the same action. This is a perfect example of polymorphism (feature that allows us to perform a single action in different ways).

[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](https://beginnersbook.com/2013/05/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.

**Dynamic Polymorphism**

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime rather, thats why it is called runtime polymorphism.

**Example**

class Animal{

public void animalSound(){

System.out.println("Default Sound");

}

}

public class Dog extends Animal{

public void animalSound(){

System.out.println("Woof");

}

public static void main(String args[]){

Animal obj = new Dog();

obj.animalSound();

}

}

**Output:**

Woof

Since both the classes, child class and parent class have the same method animalSound. Which of the method will be called is determined at runtime by JVM.

**Few more overriding examples:**

Animal obj = new Animal();

obj.animalSound();

// This would call the Animal class method

Dog obj = new Dog();

obj.animalSound();

// This would call the Dog class method

Animal obj = new Dog();

obj.animalSound();

// This would call the Dog class method

**Frequently asked questions on OOPs concepts in Java**

**1. What are four basic OOPs concepts in Java?**

The four basic features of Object Oriented programming are abstraction, encapsulation, inheritance and polymorphism.

**2. Explain oops concepts in Java with realtime examples**

**Encapsulation real time example:** A person class has a variable Aadhar number which is declared private so the other class extending this class won’t have access to the Aadhar number. This is the perfect example of encapsulation where hiding sensitive information from the outside classes.

**Abstraction real time example:**  
When you are performing online transaction on the bank website, the page shows you relevant information such as bank account number, amount entered etc. However it hides the unnecessary details from the user such as how the bank is handling the transaction at the backend.

**Inheritance real time example:**  
Consider a class Human, and subclasses such as Male, Female etc. Now these classes are extending properties such as skin color, hair, eyes, teeth etc from Human class and behaviours such as walking, talking, eating etc.

**Polymorphism real time example:**  
Let’s say an Animal class has animalSound() method. The class extending this Animal class overrides this method. For example: Lion class overrides this method with “Roar” sound, while Cat class overrides this method “Meow” sound.

**3. What are the OOPS concepts in Java with examples?**

The following are the some of the popular OOPs concepts in Java with examples:  
1. Class – Blueprint used to create object.  
2. Object – An entity that represents state and behaviour.  
3. [Inheritance](https://beginnersbook.com/2013/03/inheritance-in-java/)  
4. [Polymorphism](https://beginnersbook.com/2013/03/polymorphism-in-java/)  
5. [Abstraction](https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/)  
6. [Encapsulation](https://beginnersbook.com/2013/05/encapsulation-in-java/)  
7. Association – Association establishes relationship between two separate classes through their objects. The relationship can be one to one, One to many, many to one and many to many. [example here](https://beginnersbook.com/2013/05/association/)  
8. Aggression – It represents HAS-A relationship between two classes. It is strictly a one way association. example [here](https://beginnersbook.com/2013/05/aggregation/)  
9. Composition

**4. What are the advantages of OOPs?**

Object oriented programming has following advantages:

**Code Re-usability**

A class, variables and methods can be reused in other classes.

**Easier Code maintenance**

It is easier to do maintenance as the code is easier to troubleshoot.

**Security**

Encapsulation and abstraction features of OOPs concepts allows better security and prevents potential data leaks.

**Flexible**

It is easier to add more code to existing programs and remove or update existing code.

**5. Define access Specifiers in OOPs Concepts in Java**

Well, you must have seen public, private keyword in the examples I have shared above. They are called [access specifiers](https://beginnersbook.com/2013/05/java-access-modifiers/) 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.