**Object Oriented Programming – Java OOPs Concepts With Examples**

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. We will cover each and every feature of OOPs in detail so that you won’t face any difficultly understanding **OOPs Concepts**.

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

## **Object**



Any entity that has state and behavior is known as an object. For example a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

## **Class**

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

### 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

{

}

#### Inheritance Example

In this example, we have a parent class Teacher and a child class MathTeacher. In the MathTeacherclass 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();

}

}

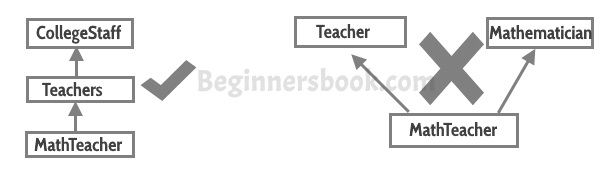
**Output:**

Beginnersbook

Teacher

Maths

Teaching

**Note: Multi-level inheritance** is allowed in Java but **not multiple inheritance**  


[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++.

### 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");

}

}

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 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

#### IS-A & HAS-A Relationships

A Car **IS-A** Vehicle and **HAS-A** License then the code would look like this:

public class Vehicle{ }

public class Car extends Vehicle{

private License myCarLicense;

}

## Abstract Class and methods in OOPs Concepts

**Abstract method:**  
1) A method that is declared but not defined. Only method signature no body.  
2) Declared using the abstract keyword  
3) Example :

abstract public void playInstrument();

5) Used to put some kind of compulsion on the class who inherits the class has abstract methods. The class that inherits must provide the implementation of all the abstract methods of parent class else declare the subclass as abstract.  
6) These cannot be abstract

* Constructors
* Static methods
* Private methods
* Methods that are declared “final”

**Abstract Class**  
An abstract class outlines the methods but not necessarily implements all the methods.

abstract class A{

abstract void myMethod();

void anotherMethod(){

//Does something

}

}

**Note 1:** There can be some scenarios where it is difficult to implement all the methods in the base class. In such scenarios one can define the base class as an abstract class which signifies that this base class is a special kind of class which is not complete on its own.

A class derived from the abstract base class must implement those methods that are not implemented(means they are abstract) in the abstract class.

**Note 2:** Abstract class cannot be instantiated which means you cannot create the object of abstract class. To use this class, you need to create another class that extends this abstract class provides the implementation of abstract methods, then you can use the object of that child class to call non-abstract parent class methods as well as implemented methods(those that were abstract in parent but implemented in child class).

**Note 3:** If a child does not implement all the abstract methods of parent class(the abstract class), then the child class must need to be declared abstract.

### Example of Abstract class and Methods

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

## 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.

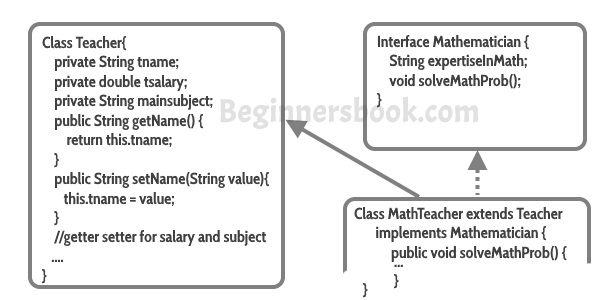
**Note:**

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](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.

------------------------------------------------------------------------------------------------------

**Java Object Oriented Programming concepts**

## Object Oriented Approach: An Introduction

Java is an object oriented language because it provides the features to implement an object oriented model.

These features includes

**Abstraction**, **Encapsulation**, **Inheritance** and **Polymorphism**.

**OOPS (above features)** is about developing an application around its data

Writing object-oriented programs involves creating classes, creating objects from those classes, and creating applications, which are stand-alone executable programs that use those objects. , i.e. objects which provides the access to their properties and the possible operations in their own way.

**Class**: A class is a template, blueprint, or contract that defines what an object’s data fields and methods will be. An object is an instance of a class. You can create many instances of a class. A Java class uses variables to define data fields and methods to define actions. Additionally, a class provides methods of a special type, known as constructors, which are invoked to create a new object. A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.

**Object**: Objects are made up of attributes and methods. Attributes are the characteristics that define an object; the values contained in attributes differentiate objects of the same class from one another. To understand this better let’s take the example of Mobile as an object. Mobile has characteristics like a model, manufacturer, cost, operating system etc. So if we create “Samsung” mobile object and “IPhone” mobile object we can distinguish them from characteristics. The values of the attributes of an object are also referred to as the object’s state.

F,{33b35e15-ac80-4191-bcbf-67c683f5e2a7}{211},12.54167,8.875

Consider a mobile phone. We generally use a mobile phone for multimedia purposes which include: texting, recording, calls, audio, video, photographs, etc.

* **Mobile phone is an object** because it is a real time entity and it has its own state, behavior and identity. Mobile phones are manufactured in surplus by various companies and each of the brands have distinct features.
* Classes in java are logical entities which have methods, fields, constructors and a lot more. **In a mobile phone, you can consider its specifications as classes,** be it processor/battery/sensors/basic features like: call, text, record.

**Abstraction:**

* Revealing relevant/necessary information and hiding the unwanted information is abstraction.
* In a mobile phone, dialing a number would call some method internally which will concatenate the numbers and displays it on screen but, we don’t know what is happening internally.
* When you tap on the call option, it is sending signals to other person’s mobile (whom you are trying to call) but we are unaware of its implementation.
* So it is hiding the implementation and showing you the functionality only.

For example, when you login to your Amazon account online, you enter your user\_id and password and press login, what happens when you press login, how the input data sent to amazon server, how it gets verified is all abstracted away from the you.

Another example of abstraction: A car in itself is a well-defined object, which is composed of several other smaller objects like a gearing system, steering mechanism, engine, which are again have their own subsystems. But for humans car is a one single object, which can be managed by the help of its subsystems, even if their inner details are unknown.

package oopsconcept;

public abstract class VehicleAbstract {

public abstract void start();

public void stop(){

System.out.println("Stopping Vehicle in abstract class");

}

}

class TwoWheeler extends VehicleAbstract{

@Override

public void start() {

System.out.println("Starting Two Wheeler");

}

}

class FourWheeler extends VehicleAbstract{

@Override

public void start() {

System.out.println("Starting Four Wheeler");

}

}

package oopsconcept;

public class VehicleTesting {

public static void main(String[] args) {

VehicleAbstract my2Wheeler = new TwoWheeler();

VehicleAbstract my4Wheeler = new FourWheeler();

my2Wheeler.start();

my2Wheeler.stop();

my4Wheeler.start();

my4Wheeler.stop();

}

}

Output :

F,{2885a284-326d-4197-ac92-c8ec7a0cd049}{116},7.4375,2.395833

**Encapsulation:**

* Enclosing one or more details from outside world through access rights.
* We have Bluetooth in our mobiles to transfer data between two mobile phones within a limited distance.
* If I turn on Bluetooth in both of the mobiles, I can transfer the desired data, but I can’t access the other mobile’s information like gallery until and unless I get a permission from the other mobile.

Encapsulation means putting together all the variables (instance variables) and the methods into a single unit called Class. It also means hiding data and methods within an Object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes.

Looking at the example of a power steering mechanism of a car. Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. It even controls the power delivered by the engine to the steering wheel. But to the external world there is only one interface is available and rest of the complexity is hidden. Moreover, the steering unit in itself is complete and independent. It does not affect the functioning of any other mechanism.

Similarly, same concept of encapsulation can be applied to code. Encapsulated code should have following characteristics:

· Everyone knows how to access it.

· Can be easily used regardless of implementation details.

· There shouldn’t any side effects of the code, to the rest of the application.

The idea of encapsulation is to keep classes separated and prevent them from having tightly coupled with each other.

package oopsconcept;

public class Mobile {

private String manufacturer;

private String operating\_system;

public String model;

private int cost;

//Constructor to set properties/characteristics of object

Mobile(String man, String o,String m, int c){

this.manufacturer = man;

this.operating\_system=o;

this.model=m;

this.cost=c;

}

//Method to get access Model property of Object

public String getModel(){

return this.model;

}

// We can add other method to get access to other properties

}

**Polymorphism:**

* Performing a job in different ways.(One to many)
* We click photographs and record videos using the camera in a mobile phone.
* Various brands have a plethora of camera modes in a mobile like: split camera/panorama/slow-mo.
* The basic purpose is to click/record, so the functionality of camera is the same in every mobile, whereas the modes enable you to use your camera in different ways.

Lets us look at same example of a car. A car have a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered to the car. The action is same applying gear but based on the type of gear the action behaves differently or you can say that it shows many forms (polymorphism means many forms)

In Core, Java Polymorphism is one of easy concept to understand. Polymorphism definition is that Poly means many and morphos means forms. It describes the feature of languages that allows the same word or symbol to be interpreted correctly in different situations based on the context. There are two types of Polymorphism available in Java. For example, in English, the verb “run” means different things if you use it with “a footrace,” a “business,” or “a computer.” You understand the meaning of “run” based on the other words used with it. Object-oriented programs are written so that the methods having the same name works differently in different context. Java provides two ways to implement polymorphism.

**Static Polymorphism (compile time polymorphism/ Method overloading):**

The ability to execute different method implementations by altering the argument used with the method name is known as method overloading. In below program, we have three print methods each with different arguments. When you properly overload a method, you can call it providing different argument lists, and the appropriate version of the method executes.

package oopsconcept;

class Overloadsample {

public void print(String s){

System.out.println("First Method with only String- "+ s);

}

public void print (int i){

System.out.println("Second Method with only int- "+ i);

}

public void print (String s, int i){

System.out.println("Third Method with both- "+ s + "--" + i);

}

}

public class PolymDemo {

public static void main(String[] args) {

Overloadsample obj = new Overloadsample();

obj.print(10);

obj.print("Amit");

obj.print("Hello", 100);

}

}

Output:

F,{77c5ce1d-d7d6-4447-b413-56a764928a8e}{51},11.04167,2.458333

**Dynamic Polymorphism (run time polymorphism/ Method Overriding)**

When you create a subclass by extending an existing class, the new subclass contains data and methods that were defined in the original superclass. In other words, any child class object has all the attributes of its parent. Sometimes, however, the superclass data fields and methods are not entirely appropriate for the subclass objects; in these cases, you want to override the parent class members

package oopsconcept;

public class OverridingDemo {

public static void main(String[] args) {

//Creating Object of SuperClass and calling getModel Method

Mobile m = new Mobile("Nokia", "Win8", "Lumia",10000);

System.out.println(m.getModel());

//Creating Object of Sublcass and calling getModel Method

Android a = new Android("Samsung", "Android", "Grand",30000);

System.out.println(a.getModel());

//Creating Object of Sublcass and calling getModel Method

Blackberry b = new Blackberry("BlackB", "RIM", "Curve",20000);

System.out.println(b.getModel());

}

}

**Inheritance:**

* One class(sub/child class) is deriving all properties from other class.(parent class)
* The basic purpose of using a mobile phone is communication.
* There are several brands in mobiles. So, the brands of a mobile are using this basic functionality (communication) by extending the mobile class functionality and adding their own new features to their respective brands.

For example: Car is a four wheeler vehicle so assume that we have a class FourWheeler and a sub class of it named Car. Here Car acquires the properties of a class FourWheeler. Other classifications could be a jeep, tempo, van etc. FourWheeler defines a class of vehicles that have four wheels, and specific range of engine power, load carrying capacity etc. Car (termed as a sub-class) acquires these properties from FourWheeler, and has some specific properties, which are different from other classifications of FourWheeler, such as luxury, comfort, shape, size, usage etc.

A car can have further classification such as an open car, small car, big car etc, which will acquire the properties from both Four Wheeler and Car, but will still have some specific properties. This way the level of hierarchy can be extended to any level.

An important feature of object-oriented programs is inheritance—the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability. So you are making use of already written the classes and further extending on that. That why we discussed the code reusability the concept. In general one line definition, we can tell that deriving a new class from existing class, it’s called as Inheritance. You can look into the following example for inheritance concept. Here we have Mobile class extended by other specific class like Android and Blackberry.

package oopsconcept;

public class Mobile {

private String manufacturer;

private String operating\_system;

public String model;

private int cost;

//Constructor to set properties/characteristics of object

Mobile(String man, String o,String m, int c){

this.manufacturer = man;

this.operating\_system=o;

this.model=m;

this.cost=c;

}

//Method to get access Model property of Object

public String getModel(){

return this.model;

}

// We can add other method to get access to other properties

}

package oopsconcept;

public class Android extends Mobile{

//Constructor to set properties/characteristics of object

Android(String man, String o,String m, int c){

super(man, o, m, c);

}

//Method to get access Model property of Object

public String getModel(){

return "This is Android Mobile- " + model;

}

}

package oopsconcept;

public class Blackberry extends Mobile{

//Constructor to set properties/characteristics of object

Blackberry(String man, String o,String m, int c){

super(man, o, m, c);

}

public String getModel(){

return "This is Blackberry-"+ model;

}

}