Java Notes

### OOPs (Object-Oriented Programming System)

**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

## A class is defined as a blueprint from which you can create an individual object.

## Class doesn't consume any space.

### Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

### Polymorphism

If one task is performed by different ways, it is known as polymorphism. For example: to convince the customer differently,

## In Java, we use method overloading and method overriding to achieve polymorphism.

### Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

### Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

## Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier whereas in a procedure-oriented programming language it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding whereas in a procedure-oriented programming language a global data can be accessed from anywhere.

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution for a real word problem if we are using the Object-Oriented Programming language.

## What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

### Java Naming conventions

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

## Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

The following are the key rules that must be followed by every identifier:

* The name must not contain any white spaces.
* The name should not start with special characters like & (ampersand), $ (dollar), \_ (underscore).

Let's see some other rules that should be followed by identifiers.

### Class

* It should start with the uppercase letter.
* It should be a noun such as Color, Button, System, Thread, etc.
* Use appropriate words, instead of acronyms.

**Example:**

**public** **class** Employee

{

//code snippet

}

### Interface

* It should start with the uppercase letter.
* It should be an adjective such as Runnable, Remote, ActionListener.
* Use appropriate words, instead of acronyms.

**interface** Printable

{

//code snippet

}

### Method

* It should start with lowercase letter.
* It should be a verb such as main(), print(), println().
* If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().

**Example:-**

**class** Employee

{

//method

**void** draw()

{

//code snippet

}

}

### Variable

It should start with a lowercase letter such as id, name.

* It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore).
* If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
* Avoid using one-character variables such as x, y, z.

**Example :-**

1. **class** Employee
2. {
3. //variable
4. **int** id;
5. //code snippet
6. }

### Package

* It should be a lowercase letter such as java, lang.
* If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.

**Example :-**

**package** com.javatpoint; //package

**class** Employee

{

//code snippet

}

### Constant

* It should be in uppercase letters such as RED, YELLOW.
* If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY.
* It may contain digits but not as the first letter.

**Example :-**

**class** Employee

{

//constant

**static** **final** **int** MIN\_AGE = 18;

//code snippet

}

**Objects and Classes in Java**

### What is an object in Java

An entity that has state and behavior is known as an object

e.g. chair, bike, marker, pen, table, car etc.

It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

### What is a class in Java

It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

**class** <class\_name>{

    field;

   method;

}

**Object Definitions:**

* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

### Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

### Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

#### Advantage of Method

* Code Reusability
* Code Optimization

### new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

3 Ways to initialize object

There are 3 ways to initialize object in java.

1. By reference variable
2. By method
3. By constructor

*File: TestStudent1.java*

1. **class** Student{
2. **int** id;
3. String name;
4. }
5. //Creating another class TestStudent1 which contains the main method
6. **class** TestStudent1{
7. **public** **static** **void** main(String args[]){
8. Student s1=**new** Student();
9. System.out.println(s1.id);
10. System.out.println(s1.name);
11. }
12. }

Output:

0

null

### 1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

*File: TestStudent2.java*

1. **class** Student{
2. **int** id;
3. String name;
4. }
5. **class** TestStudent2{
6. **public** **static** **void** main(String args[]){
7. Student s1=**new** Student();
8. s1.id=101;
9. s1.name="Sonoo";
10. System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12. }

### 2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

*File: TestStudent4.java*

1. **class** Student{
2. **int** rollno;
3. String name;
4. **void** insertRecord(**int** r, String n){
5. rollno=r;
6. name=n;
7. }
8. **void** displayInformation(){System.out.println(rollno+" "+name);}
9. }
10. **class** TestStudent4{
11. **public** **static** **void** main(String args[]){
12. Student s1=**new** Student();
13. Student s2=**new** Student();
14. s1.insertRecord(111,"Karan");
15. s2.insertRecord(222,"Aryan");
16. s1.displayInformation();
17. s2.displayInformation();
18. }
19. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestStudent4)

Output:

111 Karan

222 Aryan

Id=222 name=Aryan

Stack Memory

Id=111 name=Karan

S1

S2

### 3) Object and Class Example: Initialization through a constructor

We will learn about constructors in java later.

What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

## Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object.

If you have to use an object only once, an anonymous object is a good approach. For example:

1. **new** Calculation();//anonymous object

Calling method through a reference:

1. Calculation c=**new** Calculation();
2. c.fact(5);

Calling method through an anonymous object

1. **new** Calculation().fact(5);

Let's see the full example of an anonymous object in Java.

1. **class** Calculation{
2. **void** fact(**int**  n){
3. **int** fact=1;
4. **for**(**int** i=1;i<=n ;i++){
5. fact=fact\*i;
6. }
7. System.out.println("factorial is "+fact);
8. }
9. **public** **static** **void** main(String args[]){

**new** Calculation().fact(5);//calling method with anonymous object

}

}

Output:

Factorial is 120

### Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

1. **int** a=10, b=20;

Initialization of refernce variables:

1. **int** a=10, b=20;
2. Rectangle r1=**new** Rectangle(), r2=**new** Rectangle();//creating two objects

# Multithreading in Java

1. [Multithreading](https://www.javatpoint.com/multithreading-in-java)
2. [Multitasking](https://www.javatpoint.com/multithreading-in-java#multitasing)
3. [Process-based multitasking](https://www.javatpoint.com/multithreading-in-java#multiprocessing)
4. [Thread-based multitasking](https://www.javatpoint.com/multithreading-in-java#multithreading)
5. [What is Thread](https://www.javatpoint.com/multithreading-in-java#thread)

**Multithreading in java** is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

1. **class** Account{
2. **int** acc\_no;
3. String name;
4. **float** amount;
5. //Method to initialize object
6. **void** insert(**int** a,String n,**float** amt){
7. acc\_no=a;
8. name=n;
9. amount=amt;
10. }
11. //deposit method
12. **void** deposit(**float** amt){
13. amount=amount+amt;
14. System.out.println(amt+" deposited");
15. }
16. //withdraw method
17. **void** withdraw(**float** amt){
18. **if**(amount<amt){
19. System.out.println("Insufficient Balance");
20. }**else**{
21. amount=amount-amt;
22. System.out.println(amt+" withdrawn");
23. }
24. }
25. //method to check the balance of the account
26. **void** checkBalance(){System.out.println("Balance is: "+amount);}
27. //method to display the values of an object
28. **void** display(){System.out.println(acc\_no+" "+name+" "+amount);}
29. }
30. //Creating a test class to deposit and withdraw amount
31. **class** TestAccount{
32. **public** **static** **void** main(String[] args){
33. Account a1=**new** Account();
34. a1.insert(832345,"Ankit",1000);
35. a1.display();
36. a1.checkBalance();
37. a1.deposit(40000);
38. a1.checkBalance();
39. a1.withdraw(15000);
40. a1.checkBalance();
41. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAccount)

Output:

832345 Ankit 1000.0

Balance is: 1000.0

40000.0 deposited

Balance is: 41000.0

15000.0 withdrawn

Balance is: 26000.0