JVM(Virtual Machine--Bytecode conversion),JRE(Runtime Environment--Library),JDK(Development Kit--Javac,jar archiver,jar doc)

Logs—ELK,Splunk

ELK stack(Elastic search,logstash,Kibana—3 separte tools—now one by elasticsearch company)

1.Elastic search(GET,PUT methods,UPDATE API’s are also there,can use it from KIBANA interface)

Searching

NoSQL

Restful API

INDEXING(makes it fast)

2.Logstash

Data pipeline

Centralize data

Collects,parses structured/unstructured data

Many plugins

3.Kibana

Charts,Reports,Graphs,Multiple dashboards

Timelion(status at a definite time)

Provide INDEX pattern to read data in a particular pattern(Patterns should be in elastic search---Then also it will support in kibana)

JAVA 8

Some of the important Java 8 features are;

1. [forEach() method in Iterable interface](https://www.journaldev.com/2389/java-8-features-with-examples#iterable-forEach)
2. [default and static methods in Interfaces](https://www.journaldev.com/2389/java-8-features-with-examples#interface-default-static-method)
3. [Functional Interfaces and Lambda Expressions](https://www.journaldev.com/2389/java-8-features-with-examples#functional-interface-lambdas) Interface1 i1 = (s) -> System.out.println(s);(Anonymous function)
4. [Java Stream API for Bulk Data Operations on Collections](https://www.journaldev.com/2389/java-8-features-with-examples#java-stream-api) Stream<Integer> sequentialStream = myList.stream();
5. [Java Time API](https://www.journaldev.com/2389/java-8-features-with-examples#java8-time)  java.time.format
6. [Collection API improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-collection)
7. [Concurrency API improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-concurrency) ConcurrentHashMap compute(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods
8. [Java IO improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-io)
9. [Miscellaneous Core API improvements](https://www.journaldev.com/2389/java-8-features-with-examples#java8-core)

Q1> why a local variable cannot be defined with static keyword?

A static variable means that this variable belongs to the entire class and not any particular instance.

Furthermore, inside a method is "local scope" to that method.

Q2> Diff between instance and static variable?

Instance

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

Static

A variable which is declared as static is called static variable. It cannot be local.

You can create a single copy of static variable and share among all the instances of the class.

Memory allocation for static variable happens only once when the class is loaded in the memory.

class A{

int data=50;//instance variable

static int m=100;//static variable

void method(){

int n=90;//local variable

}

}//end of class

Typecasting

float f=10.5f;

//int a=f;//Compile time error

int a=(int)f;

Operators

System.out.println(10<<2);//10\*2^2=10\*4=40

System.out.println(10>>2);//10/2^2=10/4=2

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

In real time development, we create classes and use it from another class.

It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

/Java Program to demonstrate having the main method in

//another class

//Creating Student class.

class Student{

int id;

String name;

}

//Creating another class TestStudent1 which contains the main method

class TestStudent1{

public static void main(String args[]){

Student s1=new Student();

System.out.println(s1.id);

System.out.println(s1.name);

}

}

0

null

//STATIC

The static keyword in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class.

The static can be:

Variable (also known as a class variable)

Method (also known as a class method)

Block

Nested class

//Restrictions for the static method

There are two main restrictions for the static method. They are:

The static method can not use non static data member or call non-static method directly.

this and super cannot be used in static context.

class A{

int a=40;//non static

public static void main(String args[]){

System.out.println(a);

}

}

Output:Compile Time Error

//this keyword in Java

Q3.> when formal arguments and instance variable having same name?

class Student{

int rollno;

String name;

float fee;

Student(int rollno,String name,float fee){

rollno=rollno;

name=name;

fee=fee;

}

void display(){System.out.println(rollno+" "+name+" "+fee);}

}

class TestThis1{

public static void main(String args[]){

Student s1=new Student(111,"ankit",5000f);

Student s2=new Student(112,"sumit",6000f);

s1.display();

s2.display();

}}

0 null 0.0

0 null 0.0

//Need to change formal name arguments(compiler purpose),if arguments and instance variable are having different name

then no problem

class Student{

int rollno;

String name;

float fee;

Student(int rollno,String name,float fee){

this.rollno=rollno;

this.name=name;

this.fee=fee;

}

void display(){System.out.println(rollno+" "+name+" "+fee);}

}

class TestThis2{

public static void main(String args[]){

Student s1=new Student(111,"ankit",5000f);

Student s2=new Student(112,"sumit",6000f);

s1.display();

s2.display();

}}

111 ankit 5000

112 sumit 6000

this() : to invoke current class constructor

class A{

A(){System.out.println("hello a");}

A(int x){

this();

System.out.println(x);

}

}

class TestThis5{

public static void main(String args[]){

A a=new A(10);

}}

VVI //Rule: Call to this() must be the first statement in constructor.

class Student{

int rollno;

String name,course;

float fee;

Student(int rollno,String name,String course){

this.rollno=rollno;

this.name=name;

this.course=course;

}

Student(int rollno,String name,String course,float fee){

this.fee=fee;

this(rollno,name,course);//C.T.Error

}

void display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}

}

class TestThis8{

public static void main(String args[]){

Student s1=new Student(111,"ankit","java");

Student s2=new Student(112,"sumit","java",6000f);

s1.display();

s2.display();

}}

Compile Time Error: Call to this must be first statement in constructor

//INHERITANCE

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

//Why use inheritance in java

For Method Overriding (so runtime polymorphism can be achieved).

For Code Reusability.

Q> Why multiple inheritance is not supported in java?

class A{

void msg(){System.out.println("Hello");}

}

class B{

void msg(){System.out.println("Welcome");}

}

class C extends A,B{//suppose if it were

public static void main(String args[]){

C obj=new C();

obj.msg();//Now which msg() method would be invoked? Ambiguity

}

}

Q> Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading.

But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

class TestOverloading4{

public static void main(String[] args){System.out.println("main with String[]");}

public static void main(String args){System.out.println("main with String");}

public static void main(){System.out.println("main without args");}

}

main with String[]

Q> Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

SUPER keyword

class Person{

int id;

String name;

Person(int id,String name){

this.id=id;

this.name=name;

}

}

class Emp extends Person{

float salary;

Emp(int id,String name,float salary){

super(id,name);//reusing parent constructor

this.salary=salary;

}

void display(){System.out.println(id+" "+name+" "+salary);}

}

class TestSuper5{

public static void main(String[] args){

Emp e1=new Emp(1,"ankit",45000f);

e1.display();

}}

class Animal{

String color="white";

}

class Dog extends Animal{

String color="black";

void printColor(){

System.out.println(color);//prints color of Dog class

System.out.println(super.color);//prints color of Animal class

}

}

class TestSuper1{

public static void main(String args[]){

Dog d=new Dog();

d.printColor();

}}

Q> What is invoked first, instance initializer block or constructor?

class Bike8{

int speed;

Bike8(){System.out.println("constructor is invoked");}

{System.out.println("instance initializer block invoked");}

public static void main(String args[]){

Bike8 b1=new Bike8();

Bike8 b2=new Bike8();

}

}

Output:instance initializer block invoked

constructor is invoked

instance initializer block invoked

constructor is invoked

Final Keyword in Java

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

variable

method

class

The final keyword can be applied with the variables,

a final variable that have no value it is called blank final variable or uninitialized final variable.

It can be initialized in the constructor only.

The blank final variable can be static also which will be initialized in the static block only

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Polymorphism in Java

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.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Static vs Dynamic Binding

If there is any private, final or static method in a class, there is static binding.

Static

class Dog{

private void eat(){System.out.println("dog is eating...");}

public static void main(String args[]){

Dog d1=new Dog();

d1.eat();

}

}

next →← prev

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

//ABSTRACTION in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1.>Abstract class (0 to 100%)

2.>Interface (100%)

Points to Remember

An abstract class must be declared with an abstract keyword.

It can have abstract and non-abstract methods.

It cannot be instantiated.

It can have constructors and static methods also.

It can have final methods which will force the subclass not to change the body of the method.

interface A{

void a();

void b();

void c();

void d();

}

abstract class B implements A{

public void c(){System.out.println("I am c");}

}

class M extends B{

public void a(){System.out.println("I am a");}

public void b(){System.out.println("I am b");}

public void d(){System.out.println("I am d");}

}

class Test5{

public static void main(String args[]){

A a=new M();

a.a();

a.b();

a.c();

a.d();

}}

Output:I am a

I am b

I am c

I am d

//Interface--Multiple inheritance alternative

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

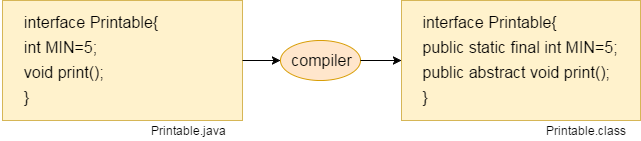
It is used to achieve abstraction.

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.

Internal addition by the compiler

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



## Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers.

1. //Interface declaration: by first user
2. **interface** Drawable{
3. **void** draw();
4. }
5. //Implementation: by second user
6. **class** Rectangle **implements** Drawable{
7. **public** **void** draw(){System.out.println("drawing rectangle");}
8. }
9. **class** Circle **implements** Drawable{
10. **public** **void** draw(){System.out.println("drawing circle");}
11. }
12. //Using interface: by third user
13. **class** TestInterface1{
14. **public** **static** **void** main(String args[]){
15. Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable()
16. d.draw();
17. }}

drawing circle

## Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable{
5. **void** print();
6. }
8. **class** TestInterface3 **implements** Printable, Showable{
9. **public** **void** print(){System.out.println("Hello");}
10. **public** **static** **void** main(String args[]){
11. TestInterface3 obj = **new** TestInterface3();
12. obj.print();
13. }
14. }

Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

## Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Interface inheritance

A class implements an interface, but one interface extends another interface.

1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable **extends** Printable{
5. **void** show();
6. }
7. **class** TestInterface4 **implements** Showable{
8. **public** **void** print(){System.out.println("Hello");}
9. **public** **void** show(){System.out.println("Welcome");}
11. **public** **static** **void** main(String args[]){
12. TestInterface4 obj = **new** TestInterface4();
13. obj.print();
14. obj.show();
15. }
16. }

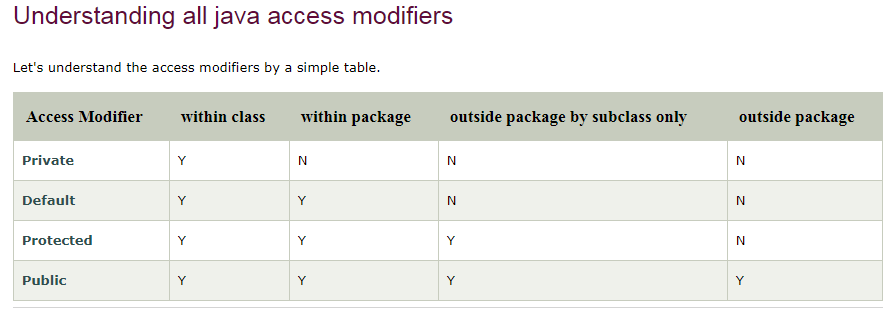
[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestInterface4)

Output:

Hello

Welcome

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/package)[**← prev**](https://www.javatpoint.com/interface-in-java) **Difference between abstract class and interface** Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.  But there are many differences between abstract class and interface that are given below.   |  |  | | --- | --- | | **Abstract class** | **Interface** | | 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. | | 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. | | 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. | | 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. | | 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. | | 6) An **abstract class**can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. | | 7) An **abstract class**can be extended using keyword "extends". | An **interface class**can be implemented using keyword "implements". | | 8) A Java**abstract class**can have class members like private, protected, etc. | Members of a Java interface are public by default. | | 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |   Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%). |



1. //save by A.java
2. **package** pack;
3. **public** **class** A{
4. **protected** **void** msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. **package** mypack;
8. **import** pack.\*;
10. **class** B **extends** A{
11. **public** **static** **void** main(String args[]){
12. B obj = **new** B();
13. obj.msg();
14. }
15. }

Output:Hello

CLONE

1. **class** Student18 **implements** Cloneable{
2. **int** rollno;
3. String name;
5. Student18(**int** rollno,String name){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. }
10. **public** Object clone()**throws** CloneNotSupportedException{
11. **return** **super**.clone();
12. }
14. **public** **static** **void** main(String args[]){
15. **try**{
16. Student18 s1=**new** Student18(101,"amit");
18. Student18 s2=(Student18)s1.clone();
20. System.out.println(s1.rollno+" "+s1.name);
21. System.out.println(s2.rollno+" "+s2.name);
23. }**catch**(CloneNotSupportedException c){}
25. }
26. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Student18)

Output:101 amit

101 amit

[download the example of object cloning](https://www.javatpoint.com/src/oops/clone.zip)

As you can see in the above example, both reference variables have the same value. Thus, the clone() copies the values of an object to another. So we don't need to write explicit code to copy the value of an object to another.

# **Wrapper class in Java**

**Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive.

1. **public** **class** WrapperExample1{
2. **public** **static** **void** main(String args[]){
3. //Converting int into Integer
4. **int** a=20;
5. Integer i=Integer.valueOf(a);//converting int into Integer
6. Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
7. System.out.println(a+" "+i+" "+j);
8. }}
9. Integer a=**new** Integer(3);
10. **int** i=a.intValue();//converting Integer to int

### **Example of call by value in java**

1. **class** Operation{
2. **int** data=50;
4. **void** change(**int** data){
5. data=data+100;//changes will be in the local variable only
6. }
8. **public** **static** **void** main(String args[]){
9. Operation op=**new** Operation();
11. System.out.println("before change "+op.data);
12. op.change(500);
13. System.out.println("after change "+op.data);
15. }
16. }

[download this example](https://www.javatpoint.com/src/oops/callbyvalue1.zip)

Output:before change 50

after change 50

|  |  |  |  |
| --- | --- | --- | --- |
| [**next>>**](https://www.javatpoint.com/strictfp-keyword)[**<<prev**](https://www.javatpoint.com/array-in-java) **Call by Value and Call by Reference in Java**  |  | | --- | | There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method. | |  |  **Example of call by value in java**  |  | | --- | | In case of call by value original value is not changed. Let's take a simple example: |  1. **class** Operation{ 2. **int** data=50; 4. **void** change(**int** data){ 5. data=data+100;//changes will be in the local variable only 6. } 8. **public** **static** **void** main(String args[]){ 9. Operation op=**new** Operation(); 11. System.out.println("before change "+op.data); 12. op.change(500); 13. System.out.println("after change "+op.data); 15. } 16. }   [download this example](https://www.javatpoint.com/src/oops/callbyvalue1.zip)  Output:before change 50  after change 50 **Another Example of call by value in java** In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:   1. **class** Operation2{ 2. **int** data=50; 4. **void** change(Operation2 op){ 5. op.data=op.data+100;//changes will be in the instance variable 6. }  9. **public** **static** **void** main(String args[]){ 10. Operation2 op=**new** Operation2(); 12. System.out.println("before change "+op.data); 13. op.change(op);//passing object 14. System.out.println("after change "+op.data); 16. } 17. }   [download this example](https://www.javatpoint.com/src/oops/callbyvalue2.zip)  Output:before change 50  after change 150 |

# **Java Strictfp Keyword**

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable.

### **Legal code for strictfp keyword**

The strictfp keyword can be applied on methods, classes and interfaces.

1. **strictfp** **class** A{}//strictfp applied on class
2. **strictfp** **interface** M{}//strictfp applied on interface
3. **class** A{
4. **strictfp** **void** m(){}//strictfp applied on method
5. }

### **Illegal code for strictfp keyword**

The strictfp keyword **cannot** be applied on abstract methods, variables or constructors.

1. **class** B{
2. **strictfp** **abstract** **void** m();//Illegal combination of modifiers
3. }
4. **class** B{
5. **strictfp** **int** data=10;//modifier strictfp not allowed here
6. }
7. **class** B{
8. **strictfp** B(){}//modifier strictfp not allowed here

Collections

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();
5. ArrayList<String> list=**new** ArrayList<String>();//Creating arraylist
6. LinkedList<String> list=**new** LinkedList<String>();
7. Vector<String> list=**new** Vector<String>();
8. Stack<String> stack = **new** Stack<String>();
9. **import** java.util.\*;
10. **public** **class** TestJavaCollection3{
11. **public** **static** **void** main(String args[]){
12. Vector<String> v=**new** Vector<String>();
13. v.add("Ayush");
14. v.add("Amit");
15. v.add("Ashish");
16. v.add("Garima");
17. Iterator<String> itr=v.iterator();
18. **while**(itr.hasNext()){
19. System.out.println(itr.next());
20. }
21. }
22. }
23. **import** java.util.\*;
24. **public** **class** TestJavaCollection4{
25. **public** **static** **void** main(String args[]){
26. Stack<String> stack = **new** Stack<String>();
27. stack.push("Ayush");
28. stack.push("Garvit");
29. stack.push("Amit");
30. stack.push("Ashish");
31. stack.push("Garima");
32. stack.pop();
33. Iterator<String> itr=stack.iterator();
34. **while**(itr.hasNext()){
35. System.out.println(itr.next());
36. }
37. }
38. }

Ayush

Garvit

Amit

Ashish

import java.util.\*;

public class Java\_Test {

public static void main(String[] args) {

Map map=new HashMap();

//Adding elements to map

map.put("Kol","Hyd");

map.put("Del","Kol");

map.put("Ban","Del");

map.put("Koc","Mum");

//Traversing Map

Scanner in=new Scanner(System.in);

String source=in.nextLine();

String destination=in.nextLine();

Set set=map.entrySet();//Converting to Set so that we can traverse

Iterator itr=set.iterator();

String mid\_variable="";

//Finding Final Source

while(itr.hasNext()){

//Converting to Map.Entry so that we can get key and value separately

Map.Entry entry=(Map.Entry)itr.next();

//System.out.println(entry.getKey()+" "+entry.getValue().toString());

if(entry.getValue().toString().equals( source.toString().trim()) ){

source=entry.getKey().toString();

}

}

System.out.println(source);

Iterator itr2=set.iterator();

while(itr2.hasNext()){

//Converting to Map.Entry so that we can get key and value separately

Map.Entry entry=(Map.Entry)itr2.next();

// System.out.println(entry.getKey()+" "+entry.getValue().toString());

if(entry.getKey().toString().equals( destination.toString().trim()) ){

destination=entry.getValue().toString();

}

}

System.out.println(destination);

}

}

Del

Kol

Ban

Hyd

THREAD

/ Java program to illustrate

public class oneThread extends Thread {

    public void run()

    {

        System.out.println("geeks ");

        try {

            Thread.sleep(300);

        }

        catch (InterruptedException ie) {

        }

        System.out.println("forgeeks ");

    }

    public static void main(String[] args)

    {

        oneThread c1 = new oneThread();

        oneThread c2 = new oneThread();

        c1.start();

        try {

            c1.join(); // Waiting for c1 to finish

  c1.sleep(500);

System.out.println(c1.isAlive())

        }

        catch (InterruptedException ie) {

        }

        c2.start();

    }

}

**yield()**basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.

## Java Thread class

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

## Java Thread Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Modifier and Type** | **Method** | **Description** |
| 1) | void | [start()](https://www.javatpoint.com/java-thread-start-method) | It is used to start the execution of the thread. |
| 2) | void | [run()](https://www.javatpoint.com/java-thread-run-method) | It is used to do an action for a thread. |
| 3) | static void | [sleep()](https://www.javatpoint.com/java-thread-sleep-method) | It sleeps a thread for the specified amount of time. |
| 4) | static Thread | [currentThread()](https://www.javatpoint.com/java-thread-currentthread-method) | It returns a reference to the currently executing thread object. |
| 5) | void | [join()](https://www.javatpoint.com/java-thread-join-method) | It waits for a thread to die. |
| 6) | int | [getPriority()](https://www.javatpoint.com/java-thread-getpriority-method) | It returns the priority of the thread. |
| 7) | void | [setPriority()](https://www.javatpoint.com/java-thread-setpriority-method) | It changes the priority of the thread. |
| 8) | String | [getName()](https://www.javatpoint.com/java-thread-getname-method) | It returns the name of the thread. |
| 9) | void | [setName()](https://www.javatpoint.com/java-thread-setname-method) | It changes the name of the thread. |
| 10) | long | [getId()](https://www.javatpoint.com/java-thread-getid-method) | It returns the id of the thread. |
| 11) | boolean | [isAlive()](https://www.javatpoint.com/java-thread-isalive-method) | It tests if the thread is alive. |
| 12) | static void | [yield()](https://www.javatpoint.com/java-thread-yield-method) | It causes the currently executing thread object to pause and allow other threads to execute temporarily. |
| 13) | void | [suspend()](https://www.javatpoint.com/java-thread-suspend-method) | It is used to suspend the thread. |
| 14) | void | [resume()](https://www.javatpoint.com/java-thread-resume-method) | It is used to resume the suspended thread. |
| 15) | void | [stop()](https://www.javatpoint.com/java-thread-stop-method) | It is used to stop the thread. |
| 16) | void | [destroy()](https://www.javatpoint.com/java-thread-destroy-method) | It is used to destroy the thread group and all of its subgroups. |
| 17) | boolean | [isDaemon()](https://www.javatpoint.com/java-thread-isdaemon-method) | It tests if the thread is a daemon thread. |
| 18) | void | [setDaemon()](https://www.javatpoint.com/java-thread-setdaemon-method) | It marks the thread as daemon or user thread. |
| 19) | void | [interrupt()](https://www.javatpoint.com/java-thread-interrupt-method) | It interrupts the thread. |
| 20) | boolean | [isinterrupted()](https://www.javatpoint.com/java-thread-isinterrupted-method) | It tests whether the thread has been interrupted. |
| 21) | static boolean | [interrupted()](https://www.javatpoint.com/java-thread-interrupted-method) | It tests whether the current thread has been interrupted. |
| 22) | static int | [activeCount()](https://www.javatpoint.com/java-thread-activecount-method) | It returns the number of active threads in the current thread's thread group. |
| 23) | void | [checkAccess()](https://www.javatpoint.com/java-thread-checkaccess-method) | It determines if the currently running thread has permission to modify the thread. |
| 24) | static boolean | [holdLock()](https://www.javatpoint.com/java-thread-holdlock-method) | It returns true if and only if the current thread holds the monitor lock on the specified object. |
| 25) | static void | [dumpStack()](https://www.javatpoint.com/java-thread-dumpstack-method) | It is used to print a stack trace of the current thread to the standard error stream. |
| 26) | StackTraceElement[] | [getStackTrace()](https://www.javatpoint.com/java-thread-getstacktrace-method)  //in catch | It returns an array of stack trace elements representing the stack dump of the thread. |
| 27) | static int | [enumerate()](https://www.javatpoint.com/java-thread-enumerate-method) | It is used to copy every active thread's thread group and its subgroup into the specified array. |
| 28) | Thread.State | [getState()](https://www.javatpoint.com/java-thread-getstate-method) | It is used to return the state of the thread. |
| 29) | ThreadGroup | [getThreadGroup()](https://www.javatpoint.com/java-thread-getthreadgroup-method) | It is used to return the thread group to which this thread belongs |
| 30) | String | [toString()](https://www.javatpoint.com/java-thread-tostring-method) | It is used to return a string representation of this thread, including the thread's name, priority, and thread group. |
| 31) | void | [notify()](https://www.javatpoint.com/java-thread-notify-method) | It is used to give the notification for only one thread which is waiting for a particular object. |
| 32) | void | [notifyAll()](https://www.javatpoint.com/java-thread-notifyall-method) | It is used to give the notification to all waiting threads of a particular object. |
| 33) | void | [setContextClassLoader()](https://www.javatpoint.com/java-thread-setcontextclassloader-method) | It sets the context ClassLoader for the Thread. |
| 34) | ClassLoader | [getContextClassLoader()](https://www.javatpoint.com/java-thread-getcontextclassloader-method) | It returns the context ClassLoader for the thread. |
| 35) | static Thread.UncaughtExceptionHandler | [getDefaultUncaughtExceptionHandler()](https://www.javatpoint.com/java-thread-getdefaultuncaughtexceptionhandler-method) | It returns the default handler invoked when a thread abruptly terminates due to an uncaught exception. |
| 36) | static void | [setDefaultUncaughtExceptionHandler()](https://www.javatpoint.com/java-thread-setdefaultuncaughtexceptionhandler-method) | It sets the default handler invoked when a thread abruptly terminates due to an uncaught exception. |

[**next>>**](https://www.javatpoint.com/thread-scheduler-in-java)[**<<prev**](https://www.javatpoint.com/life-cycle-of-a-thread)

# **How to create thread**

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### **1) Java Thread Example by extending Thread class**

1. **class** Multi **extends** Thread{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
5. **public** **static** **void** main(String args[]){
6. Multi t1=**new** Multi();
7. t1.start();
8. }
9. }

Output:thread is running...

### **2) Java Thread Example by implementing Runnable interface**

1. **class** Multi3 **implements** Runnable{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
6. **public** **static** **void** main(String args[]){
7. Multi3 m1=**new** Multi3();
8. Thread t1 =**new** Thread(m1);
9. t1.start();
10. }
11. }

Output:thread is running...

# **Sleep method in java**

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

## Syntax of sleep() method in java

The Thread class provides two methods for sleeping a thread:

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

## Example of sleep method in java

1. **class** TestSleepMethod1 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++){
4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}
5. System.out.println(i);
6. }
7. }
8. **public** **static** **void** main(String args[]){
9. TestSleepMethod1 t1=**new** TestSleepMethod1();
10. TestSleepMethod1 t2=**new** TestSleepMethod1();
12. t1.start();
13. t2.start();
14. }
15. }

Output:

1

1

2

2

3

3

4

4

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time,the thread shedular picks up another thread and so on.

# **Can we start a thread twice**

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown.

1. **class** TestJoinMethod1 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<=5;i++){
4. **try**{
5. Thread.sleep(500);
6. }**catch**(Exception e){System.out.println(e);}
7. System.out.println(i);
8. }
9. }
10. **public** **static** **void** main(String args[]){
11. TestJoinMethod1 t1=**new** TestJoinMethod1();
12. TestJoinMethod1 t2=**new** TestJoinMethod1();
13. TestJoinMethod1 t3=**new** TestJoinMethod1();
14. t1.start();
15. **try**{
16. t1.join();
17. }**catch**(Exception e){System.out.println(e);}
19. t2.start();
20. t3.start();
21. }
22. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJoinMethod1)

Output:1

2

3

4

5

1

1

2

2

3

3

4

4

5

5

|  |
| --- |
| As you can see in the above example,when t1 completes its task then t2 and t3 starts executing. |

1. **class** TestJoinMethod3 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. //System.out.println(Thread.currentThread().getName());
5. }
6. **public** **static** **void** main(String args[]){
7. TestJoinMethod3 t1=**new** TestJoinMethod3();
8. TestJoinMethod3 t2=**new** TestJoinMethod3();
9. System.out.println("Name of t1:"+t1.getName());
10. System.out.println("Name of t2:"+t2.getName());
11. System.out.println("id of t1:"+t1.getId());
13. t1.start();  //all t1 will complete first
14. t2.start();
16. t1.setName("Sonoo Jaiswal");
17. System.out.println("After changing name of t1:"+t1.getName());
18. }
19. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestJoinMethod3)

Output:Name of t1:Thread-0

Name of t2:Thread-1

id of t1:8

running...

After changling name of t1:Sonoo Jaiswal

running...

|  |  |  |  |
| --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/daemon-thread)[**← prev**](https://www.javatpoint.com/naming-a-thread) **Priority of a Thread (Thread Priority):**  |  | | --- | | Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses. |  3 constants defined in Thread class:  |  | | --- | | 1. public static int MIN\_PRIORITY 2. public static int NORM\_PRIORITY 3. public static int MAX\_PRIORITY |  |  | | --- | | Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10. |  **Example of priority of a Thread:**  1. **class** TestMultiPriority1 **extends** Thread{ 2. **public** **void** run(){ 3. System.out.println("running thread name is:"+Thread.currentThread().getName()); 4. System.out.println("running thread priority is:"+Thread.currentThread().getPriority()); 6. } 7. **public** **static** **void** main(String args[]){ 8. TestMultiPriority1 m1=**new** TestMultiPriority1(); 9. TestMultiPriority1 m2=**new** TestMultiPriority1(); 10. m1.setPriority(Thread.MIN\_PRIORITY); 11. m2.setPriority(Thread.MAX\_PRIORITY); 12. m1.start(); 13. m2.start(); 15. } 16. }   [**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestMultiPriority1)  Output:running thread name is:Thread-0  running thread priority is:10  running thread name is:Thread-1  running thread priority is:1 |

# **Daemon Thread in Java**

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

## Points to remember for Daemon Thread in Java

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

### **Why JVM terminates the daemon thread if there is no user thread?**

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

### **Methods for Java Daemon thread by Thread class**

The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

### **Simple example of Daemon thread in java**

*File: MyThread.java*

1. **public** **class** TestDaemonThread1 **extends** Thread{
2. **public** **void** run(){
3. **if**(Thread.currentThread().isDaemon()){//checking for daemon thread
4. System.out.println("daemon thread work");
5. }
6. **else**{
7. System.out.println("user thread work");
8. }
9. }
10. **public** **static** **void** main(String[] args){
11. TestDaemonThread1 t1=**new** TestDaemonThread1();//creating thread
12. TestDaemonThread1 t2=**new** TestDaemonThread1();
13. TestDaemonThread1 t3=**new** TestDaemonThread1();
15. t1.setDaemon(**true**);//now t1 is daemon thread
17. t1.start();//starting threads
18. t2.start();
19. t3.start();
20. }
21. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestDaemonThread1)

#### **Output**

daemon thread work

user thread work

user thread work

#### **Note: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.**

*File: MyThread.java*

1. **class** TestDaemonThread2 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("Name: "+Thread.currentThread().getName());
4. System.out.println("Daemon: "+Thread.currentThread().isDaemon());
5. }
7. **public** **static** **void** main(String[] args){
8. TestDaemonThread2 t1=**new** TestDaemonThread2();
9. TestDaemonThread2 t2=**new** TestDaemonThread2();
10. t1.start();
11. t1.setDaemon(**true**);//will throw exception here
12. t2.start();
13. }
14. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestDaemonThread2)

Output:exception in thread main: java.lang.IllegalThreadStateException

***Program of performing single task by multiple threads***

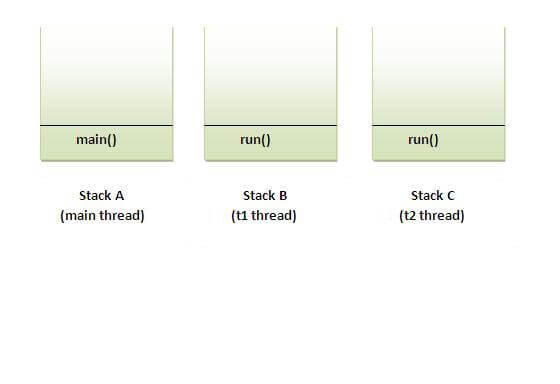
1. **class** TestMultitasking2 **implements** Runnable{
2. **public** **void** run(){
3. System.out.println("task one");
4. }
6. **public** **static** **void** main(String args[]){
7. Thread t1 =**new** Thread(**new** TestMultitasking2());//passing annonymous object of TestMultitasking2 class
8. Thread t2 =**new** Thread(**new** TestMultitasking2());
10. t1.start();
11. t2.start();
13. }
14. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestMultitasking2)

Output:task one

task one

#### **Note: Each thread run in a separate callstack.**



[**next →**](https://www.javatpoint.com/java-runtime-class)[**← prev**](https://www.javatpoint.com/multitasking-in-multithreading)

# **Java Garbage Collection**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

### **Advantage of Garbage Collection**

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

### **1) By nulling a reference:**

1. Employee e=**new** Employee();
2. e=**null**;

### **2) By assigning a reference to another:**

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

### **3) By anonymous object:**

1. **new** Employee();

## finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. **protected** **void** finalize(){}

#### **Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).**

## gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

1. **public** **static** **void** gc(){}

#### **Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.**

[**next →**](https://www.javatpoint.com/java-runtime-class)[**← prev**](https://www.javatpoint.com/multitasking-in-multithreading)

# **Java Garbage Collection**

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

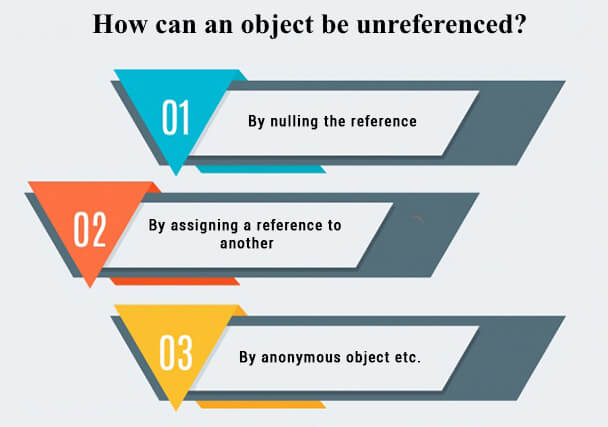
### **Advantage of Garbage Collection**

* It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.



### **1) By nulling a reference:**

1. Employee e=**new** Employee();
2. e=**null**;

### **2) By assigning a reference to another:**

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

### **3) By anonymous object:**

1. **new** Employee();

## finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. **protected** **void** finalize(){}

#### **Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).**

## gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

1. **public** **static** **void** gc(){}

#### **Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.**

### **Simple Example of garbage collection in java**

1. **public** **class** TestGarbage1{
2. **public** **void** finalize(){System.out.println("object is garbage collected");}
3. **public** **static** **void** main(String args[]){
4. TestGarbage1 s1=**new** TestGarbage1();
5. TestGarbage1 s2=**new** TestGarbage1();
6. s1=**null**;
7. s2=**null**;
8. System.gc();
9. }
10. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestGarbage1)

object is garbage collected

object is garbage collected

**Note: Neither finalization nor garbage collection is guaranteed.**

# **Java Runtime class**

**Java Runtime** class is used to interact with java runtime environment. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.

The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

## Important methods of Java Runtime class

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public static Runtime getRuntime() | returns the instance of Runtime class. |
| 2) | public void exit(int status) | terminates the current virtual machine. |
| 3) | public void addShutdownHook(Thread hook) | registers new hook thread. |
| 4) | public Process exec(String command)throws IOException | executes given command in a separate process. |
| 5) | public int availableProcessors() | returns no. of available processors. |
| 6) | public long freeMemory() | returns amount of free memory in JVM. |
| 7) | public long totalMemory() | returns amount of total memory in JVM. |

## Java Runtime exec() method

1. **public** **class** Runtime1{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("notepad");//will open a new notepad
4. }
5. }

## How to shutdown system in Java

You can use shutdown -s command to shutdown system. For windows OS, you need to provide full path of shutdown command e.g. c:\\Windows\\System32\\shutdown.

Here you can use -s switch to shutdown system, -r switch to restart system and -t switch to specify time delay.

1. **public** **class** Runtime2{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("shutdown -s -t 0");
4. }
5. }

## How to shutdown windows system in Java

1. **public** **class** Runtime2{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("c:\\Windows\\System32\\shutdown -s -t 0");
4. }
5. }

## How to restart system in Java

1. **public** **class** Runtime3{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime.getRuntime().exec("shutdown -r -t 0");
4. }
5. }

## Java Runtime availableProcessors()

1. **public** **class** Runtime4{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. System.out.println(Runtime.getRuntime().availableProcessors());
4. }
5. }

## Java Runtime freeMemory() and totalMemory() method

In the given program, after creating 10000 instance, free memory will be less than the previous free memory. But after gc() call, you will get more free memory.

1. **public** **class** MemoryTest{
2. **public** **static** **void** main(String args[])**throws** Exception{
3. Runtime r=Runtime.getRuntime();
4. System.out.println("Total Memory: "+r.totalMemory());
5. System.out.println("Free Memory: "+r.freeMemory());
7. **for**(**int** i=0;i<10000;i++){
8. **new** MemoryTest();
9. }
10. System.out.println("After creating 10000 instance, Free Memory: "+r.freeMemory());
11. System.gc();
12. System.out.println("After gc(), Free Memory: "+r.freeMemory());
13. }
14. }

Total Memory: 100139008

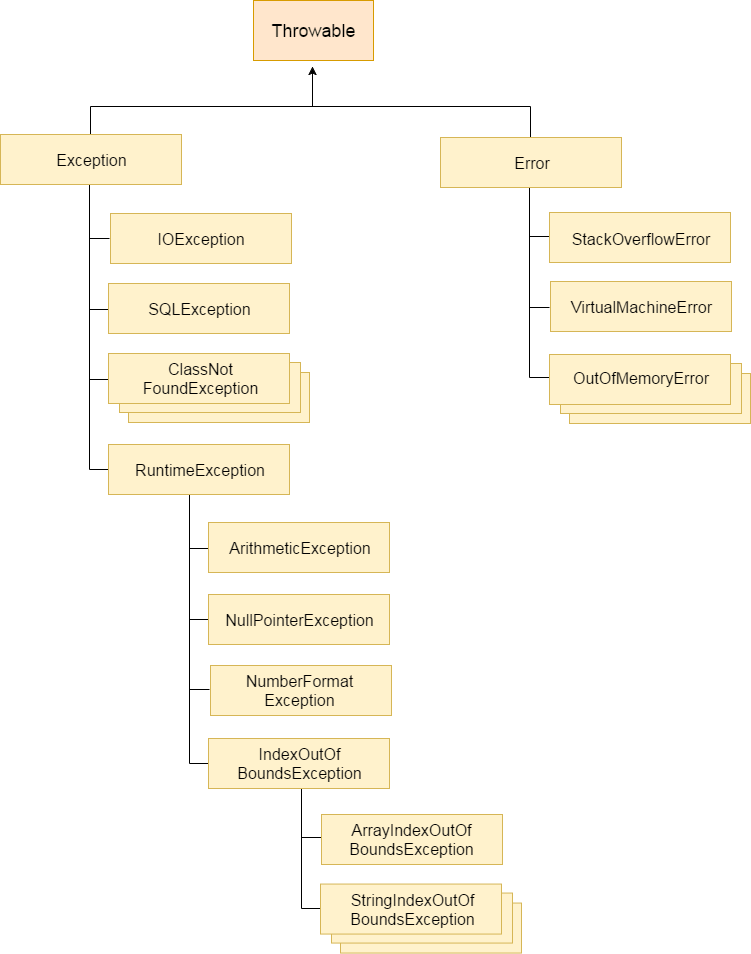
Free Memory: 99474824

After creating 10000 instance, Free Memory: 99310552

After gc(), Free Memory: 100182832

## Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

1. **import** java.io.\*;
2. **class** M{
3. **void** method()**throws** IOException{
4. **throw** **new** IOException("device error");
5. }
6. }
7. **public** **class** Testthrows2{
8. **public** **static** **void** main(String args[]){
9. **try**{
10. M m=**new** M();
11. m.method();
12. }**catch**(Exception e){System.out.println("exception handled");}
14. System.out.println("normal flow...");
15. }
16. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testthrows2)

Output:exception handled

normal flow...

1. **import** java.io.\*;
2. **class** M{
3. **void** method()**throws** IOException{
4. System.out.println("device operation performed");
5. }
6. }
7. **class** Testthrows3{
8. **public** **static** **void** main(String args[])**throws** IOException{//declare exception
9. M m=**new** M();
10. m.method();
12. System.out.println("normal flow...");
13. }
14. }

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next →**](https://www.javatpoint.com/difference-between-final-finally-and-finalize)[**← prev**](https://www.javatpoint.com/exception-propagation) **Difference between throw and throws in Java** There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:   |  |  |  | | --- | --- | --- | | **No.** | **throw** | **throws** | | 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used  to declare an exception. | | 2) | Checked exception cannot be propagated using throw only. | Checked exception can be  propagated with throws. | | 3) | Throw is followed by an instance. | Throws is followed by class. | | 4) | Throw is used within the method. | Throws is used with the  method signature. | | 5) | You cannot throw multiple exceptions. | You can declare multiple  exceptions e.g. public void method()throws IOException,SQLException. | |

# **Difference between final, finally and finalize**

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **final** | **finally** | **finalize** |
| 1) | Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed. | Finally is used to place important code, it will be executed whether exception is handled or not. | Finalize is used to perform clean up processing just before object is garbage collected. |
| 2) | Final is a keyword. | Finally is a block. | Finalize is a method. |

# [How HashMap works in Java](https://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html)

HashMap internally stores mapping in the form of **Map.Entry** object which contains both key and value object. When you want to retrieve the object, you call [the get() method](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) and again pass the key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve the object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location.

**What will happen if two different objects have the same hashcode?**

Since hashcode is same, bucket location would be same and collision will occur in HashMap Since HashMap uses LinkedList to store object, this entry (object of Map.Entrycomprise key and value )  will be stored in [LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html).

"**What happens On HashMap in Java if the size of the HashMap  exceeds a given threshold defined by load factor ?"**.

Java HashMap re-size itself by creating a new bucket array of size twice of the previous size of HashMap and then start putting every old element into that new bucket array. This process is called rehashing because it also applies the hash function to find new bucket location.

**3) Can we use ConcurrentHashMap in place of Hashtable?**

This is another question which getting popular due to increasing popularity ofConcurrentHashMap. Since we know Hashtable is synchronized but ConcurrentHashMapprovides better concurrency by only locking portion of map determined by concurrency level.ConcurrentHashMap is certainly introduced as Hashtable and can be used in place of it, butHashtable provides stronger thread-safety than ConcurrentHashMap.

Diff between implements and extends?

Because a class can **implement** multiple interfaces but **extend** only one class. ... **Difference**:**implements** means you are using the elements of a**Java** Interface in your class. **extends** means that you are creating a subclass of the base class you are**extending**.

IMPORTANT

All **synchronized are from legacy class(Vector,LinkeyList) and they can iterate through iterator and enumeration.**

**All un- synchronized are new(ArrayList) and they can iterate through iterator only.**