# Java 8 Features with Examples

## Java 8 Features

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)
4. [Java Stream API for Bulk Data Operations on Collections](https://www.journaldev.com/2389/java-8-features-with-examples#java-stream-api)
5. [Java Time API](https://www.journaldev.com/2389/java-8-features-with-examples#java8-time)
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)
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)

Let’s have a brief look on these Java 8 features. I will provide some code snippets for better understanding, so if you want to run programs in Java 8, you will have to setup Java 8 environment by following steps.

* [Download JDK8](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) and install it. Installation is simple like other java versions. JDK installation is required to write, compile and run the program in Java.
* Download latest Eclipse IDE, it provides support for java 8 now. Make sure your projects build path is using Java 8 library.

### forEach() method in Iterable interface

Whenever we need to traverse through a Collection, we need to create an Iterator whose whole purpose is to iterate over and then we have business logic in a loop for each of the elements in the Collection. We might get [ConcurrentModificationException](https://www.journaldev.com/378/java-util-concurrentmodificationexception) if iterator is not used properly.

Java 8 has introduced forEach method in java.lang.Iterable interface so that while writing code we focus on business logic only. forEach method takes java.util.function.Consumer object as argument, so it helps in having our business logic at a separate location that we can reuse. Let’s see forEach usage with simple example.

package com.journaldev.java8.foreach;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.function.Consumer;

import java.lang.Integer;

public class Java8ForEachExample {

public static void main(String[] args) {

//creating sample Collection

List<Integer> myList = new ArrayList<Integer>();

for(int i=0; i<10; i++) myList.add(i);

//traversing using Iterator

Iterator<Integer> it = myList.iterator();

while(it.hasNext()){

Integer i = it.next();

System.out.println("Iterator Value::"+i);

}

//traversing through forEach method of Iterable with anonymous class

myList.forEach(new Consumer<Integer>() {

public void accept(Integer t) {

System.out.println("forEach anonymous class Value::"+t);

}

});

//traversing with Consumer interface implementation

MyConsumer action = new MyConsumer();

myList.forEach(action);

}

}

//Consumer implementation that can be reused

class MyConsumer implements Consumer<Integer>{

public void accept(Integer t) {

System.out.println("Consumer impl Value::"+t);

}

}

The number of lines might increase but forEach method helps in having the logic for iteration and business logic at separate place resulting in higher separation of concern and cleaner code.

### default and static methods in Interfaces

If you read forEach method details carefully, you will notice that it’s defined in Iterable interface but we know that interfaces can’t have method body. From Java 8, interfaces are enhanced to have method with implementation. We can use default and static keyword to create interfaces with method implementation. forEach method implementation in Iterable interface is:

default void forEach(Consumer<? super T> action) {

Objects.requireNonNull(action);

for (T t : this) {

action.accept(t);

}

}

We know that Java doesn’t provide [multiple inheritance in Classes](https://www.journaldev.com/1775/multiple-inheritance-in-java) because it leads to **Diamond Problem**. So how it will be handled with interfaces now, since interfaces are now similar to abstract classes. The solution is that compiler will throw exception in this scenario and we will have to provide implementation logic in the class implementing the interfaces.

package com.journaldev.java8.defaultmethod;

@FunctionalInterface

public interface Interface1 {

void method1(String str);

default void log(String str){

System.out.println("I1 logging::"+str);

}

static void print(String str){

System.out.println("Printing "+str);

}

//trying to override Object method gives compile time error as

//"A default method cannot override a method from java.lang.Object"

// default String toString(){

// return "i1";

// }

}

package com.journaldev.java8.defaultmethod;

@FunctionalInterface

public interface Interface2 {

void method2();

default void log(String str){

System.out.println("I2 logging::"+str);

}

}

Notice that both the interfaces have a common method log() with implementation logic.

package com.journaldev.java8.defaultmethod;

public class MyClass implements Interface1, Interface2 {

@Override

public void method2() {

}

@Override

public void method1(String str) {

}

//MyClass won't compile without having it's own log() implementation

@Override

public void log(String str){

System.out.println("MyClass logging::"+str);

Interface1.print("abc");

}

}

As you can see that Interface1 has static method implementation that is used in MyClass.log() method implementation. Java 8 uses **default** and **static** methods heavily in [**Collection API**](https://www.journaldev.com/1260/collections-in-java-tutorial) and default methods are added so that our code remains backward compatible.

If any class in the hierarchy has a method with same signature, then default methods become irrelevant. Since any class implementing an interface already has Object as superclass, if we have equals(), hashCode() default methods in interface, it will become irrelevant. Thats why for better clarity, interfaces are not allowed to have Object class default methods.

For complete details of interface changes in Java 8, please read [Java 8 interface changes](https://www.journaldev.com/2752/java-8-interface-changes-static-method-default-method).

### Functional Interfaces and Lambda Expressions

If you notice above interfaces code, you will notice @FunctionalInterface [annotation](https://www.journaldev.com/721/java-annotations). Functional interfaces are new concept introduced in Java 8. An interface with exactly one abstract method becomes Functional Interface. We don’t need to use @FunctionalInterface annotation to mark an interface as Functional Interface. @FunctionalInterface annotation is a facility to avoid accidental addition of abstract methods in the functional interfaces. You can think of it like [@Override annotation](https://www.journaldev.com/817/java-override-annotation) and it’s best practice to use it. java.lang.Runnable with single abstract method run() is a great example of functional interface.

One of the major benefits of functional interface is the possibility to use **lambda expressions** to instantiate them. We can instantiate an interface with [anonymous class](https://www.journaldev.com/996/java-inner-class) but the code looks bulky.

Runnable r = new Runnable(){

@Override

public void run() {

System.out.println("My Runnable");

}};

Since functional interfaces have only one method, lambda expressions can easily provide the method implementation. We just need to provide method arguments and business logic. For example, we can write above implementation using lambda expression as:

Runnable r1 = () -> {

System.out.println("My Runnable");

};

If you have single statement in method implementation, we don’t need curly braces also. For example above Interface1 anonymous class can be instantiated using lambda as follows:

Interface1 i1 = (s) -> System.out.println(s);

i1.method1("abc");

So lambda expressions are means to create anonymous classes of functional interfaces easily. There are no runtime benefits of using lambda expressions, so I will use it cautiously because I don’t mind writing few extra lines of code.

A new package java.util.function has been added with bunch of functional interfaces to provide target types for lambda expressions and method references. Lambda expressions are a huge topic, I will write a separate article on that in future.

# Functional Interfaces In Java

A functional interface is an interface that contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, [lambda expressions](https://www.geeksforgeeks.org/lambda-expressions-java-8/) can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. ***Runnable***, ***ActionListener***, ***Comparable*** are some of the examples of functional interfaces.  
Before Java 8, we had to create anonymous inner class objects or implement these interfaces.

*filter\_none*

*edit*  
*close*

*play\_arrow*

*link*  
*brightness\_4*  
*code*

|  |
| --- |
| // Java program to demonstrate functional interface   class Test  {      public static void main(String args[])      {          // create anonymous inner class object          new Thread(new Runnable()          {              @Override              public void run()              {                  System.out.println("New thread created");              }          }).start();      }  } |

***chevron\_right***

*filter\_none*

Output:

New thread created

Java 8 onwards, we can assign [lambda expression](https://www.geeksforgeeks.org/lambda-expressions-java-8/) to its functional interface object like this:

*filter\_none*

*edit*  
*close*

*play\_arrow*

*link*  
*brightness\_4*  
*code*

|  |
| --- |
| // Java program to demonstrate Implementation of  // functional interface using lambda expressions    class Test  {    public static void main(String args[])    {        // lambda expression to create the object      new Thread(()->         {System.out.println("New thread created");}).start();    }  } |

***chevron\_right***

*filter\_none*

New thread created

**@FunctionalInterface Annotation**  
@FunctionalInterface annotation is used to ensure that the functional interface can’t have more than one abstract method. In case more than one abstract methods are present, the compiler flags an ‘Unexpected @FunctionalInterface annotation’ message. However, it is not mandatory to use this annotation.

*filter\_none*

*edit*  
*close*

*play\_arrow*

*link*  
*brightness\_4*  
*code*

|  |
| --- |
| // Java program to demonstrate lamda expressions to implement  // a user defined functional interface.    @FunctionalInterface  interface Square  {      int calculate(int x);  }    class Test  {      public static void main(String args[])      {          int a = 5;            // lambda expression to define the calculate method          Square s = (int x)->x\*x;            // parameter passed and return type must be          // same as defined in the prototype          int ans = s.calculate(a);          System.out.println(ans);      }  } |

***chevron\_right***

*filter\_none*

Output:

25

**java.util.function Package:**  
The java.util.function package in Java 8 contains many builtin functional interfaces like-

* **Predicate:** The Predicate interface has an abstract method test which gives a Boolean value as a result for the specified argument. Its prototype is
* public Predicate
* {
* public boolean test(T t);

}

* **BinaryOperator:** The BinaryOperator interface has an abstract method apply which takes two argument and returns a result of same type. Its prototype is
* public interface BinaryOperator
* {
* public T apply(T x, T y);

}

* **Function:** The Function interface has an abstract method apply which takes argument of type T and returns a result of type R. Its prototype is
* public interface Function
* {
* public R apply(T t);

}

*filter\_none*

*edit*  
*close*

*play\_arrow*

*link*  
*brightness\_4*  
*code*

|  |
| --- |
| // A simple program to demonstrate the use  // of predicate interface  import java.util.\*;  import java.util.function.Predicate;    class Test  {      public static void main(String args[])      {            // create a list of strings          List<String> names =              Arrays.asList("Geek","GeeksQuiz","g1","QA","Geek2");            // declare the predicate type as string and use          // lambda expression to create object          Predicate<String> p = (s)->s.startsWith("G");            // Iterate through the list          for (String st:names)          {              // call the test method              if (p.test(st))                  System.out.println(st);          }      }  } |

***chevron\_right***

*filter\_none*

Output:

Geek

GeeksQuiz

Geek2

**Important Points/Observations:**

1. A functional interface has only one abstract method but it can have multiple default methods.
2. @FunctionalInterface annotation is used to ensure an interface can’t have more than one abstract method. The use of this annotation is optional.
3. The java.util.function package contains many builtin functional interfaces in Java 8.

### Java Stream API for Bulk Data Operations on Collections

A new java.util.stream has been added in Java 8 to perform filter/map/reduce like operations with the collection. Stream API will allow sequential as well as parallel execution. This is one of the best feature for me because I work a lot with Collections and usually with Big Data, we need to filter out them based on some conditions.

Collection interface has been extended with stream() and parallelStream() default methods to get the Stream for sequential and parallel execution. Let’s see their usage with simple example.

package com.journaldev.java8.stream;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Stream;

public class StreamExample {

public static void main(String[] args) {

List<Integer> myList = new ArrayList<>();

for(int i=0; i<100; i++) myList.add(i);

//sequential stream

Stream<Integer> sequentialStream = myList.stream();

//parallel stream

Stream<Integer> parallelStream = myList.parallelStream();

//using lambda with Stream API, filter example

Stream<Integer> highNums = parallelStream.filter(p -> p > 90);

//using lambda in forEach

highNums.forEach(p -> System.out.println("High Nums parallel="+p));

Stream<Integer> highNumsSeq = sequentialStream.filter(p -> p > 90);

highNumsSeq.forEach(p -> System.out.println("High Nums sequential="+p));

}

}

If you will run above example code, you will get output like this:

High Nums parallel=91

High Nums parallel=96

High Nums parallel=93

High Nums parallel=98

High Nums parallel=94

High Nums parallel=95

High Nums parallel=97

High Nums parallel=92

High Nums parallel=99

High Nums sequential=91

High Nums sequential=92

High Nums sequential=93

High Nums sequential=94

High Nums sequential=95

High Nums sequential=96

High Nums sequential=97

High Nums sequential=98

High Nums sequential=99

Notice that parallel processing values are not in order, so parallel processing will be very helpful while working with huge collections.  
Covering everything about Stream API is not possible in this post, you can read everything about Stream API at [Java 8 Stream API Example Tutorial](https://www.journaldev.com/2774/java-8-stream).

### Java Time API

It has always been hard to work with Date, Time and Time Zones in java. There was no standard approach or API in java for date and time in Java. One of the nice addition in Java 8 is the java.time package that will streamline the process of working with time in java.

Just by looking at Java Time API packages, I can sense that it will be very easy to use. It has some sub-packages java.time.format that provides classes to print and parse dates and times and java.time.zone provides support for time-zones and their rules.

The new Time API prefers enums over integer constants for months and days of the week. One of the useful class is DateTimeFormatter for converting datetime objects to strings.

For complete tutorial, head over to [Java Date Time API Example Tutorial](https://www.journaldev.com/2800/java-8-date-localdate-localdatetime-instant).

### Collection API improvements

We have already seen forEach() method and Stream API for collections. Some new methods added in Collection API are:

* + Iterator default method forEachRemaining(Consumer action) to perform the given action for each remaining element until all elements have been processed or the action throws an exception.
  + Collection default method removeIf(Predicate filter) to remove all of the elements of this collection that satisfy the given predicate.
  + Collection spliterator() method returning Spliterator instance that can be used to traverse elements sequentially or parallel.
  + Map replaceAll(), compute(), merge() methods.
  + Performance Improvement for HashMap class with Key Collisions

### Concurrency API improvements

Some important concurrent API enhancements are:

* + ConcurrentHashMap compute(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods.
  + CompletableFuture that may be explicitly completed (setting its value and status).
  + Executors newWorkStealingPool() method to create a work-stealing thread pool using all available processors as its target parallelism level.

### Java IO improvements

Some IO improvements known to me are:

* + Files.list(Path dir) that returns a lazily populated Stream, the elements of which are the entries in the directory.
  + Files.lines(Path path) that reads all lines from a file as a Stream.
  + Files.find() that returns a Stream that is lazily populated with Path by searching for files in a file tree rooted at a given starting file.
  + BufferedReader.lines() that return a Stream, the elements of which are lines read from this BufferedReader.

### Miscellaneous Core API improvements

Some misc API improvements that might come handy are:

* + [ThreadLocal](https://www.journaldev.com/1076/java-threadlocal-example) static method withInitial(Supplier supplier) to create instance easily.
  + [Comparator](https://www.journaldev.com/780/comparable-and-comparator-in-java-example) interface has been extended with a lot of default and static methods for natural ordering, reverse order etc.
  + min(), max() and sum() methods in Integer, Long and Double wrapper classes.
  + logicalAnd(), logicalOr() and logicalXor() methods in Boolean class.
  + [ZipFile](https://www.journaldev.com/957/java-zip-file-folder-example).stream() method to get an ordered Stream over the ZIP file entries. Entries appear in the Stream in the order they appear in the central directory of the ZIP file.
  + Several utility methods in Math class.
  + jjs command is added to invoke Nashorn Engine.
  + jdeps command is added to analyze class files
  + JDBC-ODBC Bridge has been removed.
  + PermGen memory space has been removed

That’s all for Java 8 features with example programs. If I have missed some important features of Java 8, please let me know through comments.

**Java8 features end---**

# ThreadPoolExecutor – Java Thread Pool Example: Start

**Java thread pool** manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed. We can use ThreadPoolExecutor to create thread pool in java.

Java thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue. **java.util.concurrent.Executors** provide implementation of **java.util.concurrent.Executor**interface to create the thread pool in java. Let’s write a simple program to explain it’s working.

First, we need to have a Runnable class, named WorkerThread.java

package com.journaldev.threadpool;

public class WorkerThread implements Runnable {

private String command;

public WorkerThread(String s){

this.command=s;

}

@Override

public void run() {

System.out.println(Thread.currentThread().getName()+" Start. Command = "+command);

processCommand();

System.out.println(Thread.currentThread().getName()+" End.");

}

private void processCommand() {

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public String toString(){

return this.command;

}

}

## ExecutorService Example

Here is the test program class SimpleThreadPool.java, where we are creating fixed thread pool from **Executors framework**.

package com.journaldev.threadpool;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class SimpleThreadPool {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(5);

for (int i = 0; i < 10; i++) {

Runnable worker = new WorkerThread("" + i);

executor.execute(worker);

}

executor.shutdown();

while (!executor.isTerminated()) {

}

System.out.println("Finished all threads");

}

}

In above program, we are creating fixed size thread pool of 5 worker threads. Then we are submitting 10 jobs to this pool, since the pool size is 5, it will start working on 5 jobs and other jobs will be in wait state, as soon as one of the job is finished, another job from the wait queue will be picked up by worker thread and get's executed.

Here is the output of the above program.

pool-1-thread-2 Start. Command = 1

pool-1-thread-4 Start. Command = 3

pool-1-thread-1 Start. Command = 0

pool-1-thread-3 Start. Command = 2

pool-1-thread-5 Start. Command = 4

pool-1-thread-4 End.

pool-1-thread-5 End.

pool-1-thread-1 End.

pool-1-thread-3 End.

pool-1-thread-3 Start. Command = 8

pool-1-thread-2 End.

pool-1-thread-2 Start. Command = 9

pool-1-thread-1 Start. Command = 7

pool-1-thread-5 Start. Command = 6

pool-1-thread-4 Start. Command = 5

pool-1-thread-2 End.

pool-1-thread-4 End.

pool-1-thread-3 End.

pool-1-thread-5 End.

pool-1-thread-1 End.

Finished all threads

The output confirms that there are five threads in the pool named from "pool-1-thread-1" to "pool-1-thread-5" and they are responsible to execute the submitted tasks to the pool.

## ThreadPoolExecutor Example

**Executors** class provide simple implementation of **ExecutorService** using **ThreadPoolExecutor** but ThreadPoolExecutor provides much more feature than that. We can specify the number of threads that will be alive when we create ThreadPoolExecutor instance and we can limit the size of thread pool and create our own **RejectedExecutionHandler** implementation to handle the jobs that can't fit in the worker queue.

Here is our custom implementation of RejectedExecutionHandler interface.

package com.journaldev.threadpool;

import java.util.concurrent.RejectedExecutionHandler;

import java.util.concurrent.ThreadPoolExecutor;

public class RejectedExecutionHandlerImpl implements RejectedExecutionHandler {

@Override

public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {

System.out.println(r.toString() + " is rejected");

}

}

ThreadPoolExecutor provides several methods using which we can find out the current state of executor, pool size, active thread count and task count. So I have a monitor thread that will print the executor information at certain time interval.

package com.journaldev.threadpool;

import java.util.concurrent.ThreadPoolExecutor;

public class MyMonitorThread implements Runnable

{

private ThreadPoolExecutor executor;

private int seconds;

private boolean run=true;

public MyMonitorThread(ThreadPoolExecutor executor, int delay)

{

this.executor = executor;

this.seconds=delay;

}

public void shutdown(){

this.run=false;

}

@Override

public void run()

{

while(run){

System.out.println(

String.format("[monitor] [%d/%d] Active: %d, Completed: %d, Task: %d, isShutdown: %s, isTerminated: %s",

this.executor.getPoolSize(),

this.executor.getCorePoolSize(),

this.executor.getActiveCount(),

this.executor.getCompletedTaskCount(),

this.executor.getTaskCount(),

this.executor.isShutdown(),

this.executor.isTerminated()));

try {

Thread.sleep(seconds\*1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

Here is the thread pool implementation example using **ThreadPoolExecutor**.

package com.journaldev.threadpool;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.Executors;

import java.util.concurrent.ThreadFactory;

import java.util.concurrent.ThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

public class WorkerPool {

public static void main(String args[]) throws InterruptedException{

//RejectedExecutionHandler implementation

RejectedExecutionHandlerImpl rejectionHandler = new RejectedExecutionHandlerImpl();

//Get the ThreadFactory implementation to use

ThreadFactory threadFactory = Executors.defaultThreadFactory();

//creating the ThreadPoolExecutor

ThreadPoolExecutor executorPool = new ThreadPoolExecutor(2, 4, 10, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2), threadFactory, rejectionHandler);

//start the monitoring thread

MyMonitorThread monitor = new MyMonitorThread(executorPool, 3);

Thread monitorThread = new Thread(monitor);

monitorThread.start();

//submit work to the thread pool

for(int i=0; i<10; i++){

executorPool.execute(new WorkerThread("cmd"+i));

}

Thread.sleep(30000);

//shut down the pool

executorPool.shutdown();

//shut down the monitor thread

Thread.sleep(5000);

monitor.shutdown();

}

}

Notice that while initializing the ThreadPoolExecutor, we are keeping initial pool size as 2, maximum pool size to 4 and work queue size as 2. So if there are 4 running tasks and more tasks are submitted, the work queue will hold only 2 of them and rest of them will be handled by RejectedExecutionHandlerImpl.

Here is the output of above program that confirms above statement.

pool-1-thread-1 Start. Command = cmd0

pool-1-thread-4 Start. Command = cmd5

cmd6 is rejected

pool-1-thread-3 Start. Command = cmd4

pool-1-thread-2 Start. Command = cmd1

cmd7 is rejected

cmd8 is rejected

cmd9 is rejected

[monitor] [0/2] Active: 4, Completed: 0, Task: 6, isShutdown: false, isTerminated: false

[monitor] [4/2] Active: 4, Completed: 0, Task: 6, isShutdown: false, isTerminated: false

pool-1-thread-4 End.

pool-1-thread-1 End.

pool-1-thread-2 End.

pool-1-thread-3 End.

pool-1-thread-1 Start. Command = cmd3

pool-1-thread-4 Start. Command = cmd2

[monitor] [4/2] Active: 2, Completed: 4, Task: 6, isShutdown: false, isTerminated: false

[monitor] [4/2] Active: 2, Completed: 4, Task: 6, isShutdown: false, isTerminated: false

pool-1-thread-1 End.

pool-1-thread-4 End.

[monitor] [4/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false

[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false

[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false

[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false

[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false

[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown: false, isTerminated: false

[monitor] [0/2] Active: 0, Completed: 6, Task: 6, isShutdown: true, isTerminated: true

[monitor] [0/2] Active: 0, Completed: 6, Task: 6, isShutdown: true, isTerminated: true

Notice the change in active, completed and total completed task count of the executor. We can invoke **shutdown()** method to finish execution of all the submitted tasks and terminate the thread pool.

**Why Do We Need ThreadPool ?**

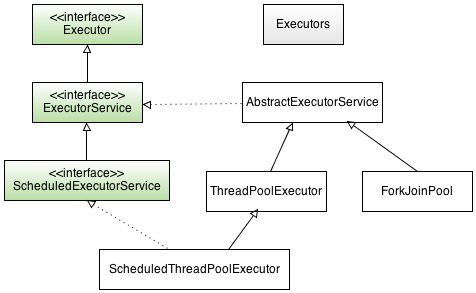
In a large scale web application, if only one thread / process is used to process all client requests, then only one client can access the server at a time.  In order to support large number of clients, we need to use one thread per request paradigm i.e. we need to use separate threads to process each client request. But this require new threads to be created, when request arrived.

Creating a thread in java is a very expensive process which includes memory overhead. So, it’s a good idea if we can re-use these threads once created, to run our future runnables. Thread pool solves this problem for us. It creates Thread and manages them. Instead of creating Thread and discarding them once task is done, thread-pool reuses threads in form of worker thread.

**Executor Framework**

Java 1.5 introduced Thread pool in Java in the form of Executor framework, which allows Java programmer to decouple submission of a task to execution of the task.

java.util.concurrent package hierarchy :



**1) Executor**

Executor interface is the core of executor framework. It has two sub interfaces ExecutorService and ScheduledExecutorService.  An object of type Executor can execute Runnable and Callable tasks.

The ExecutorService feature came with Java 5. It extends the Executor interface and provides a thread pool feature to execute asynchronous short tasks.   
  
There are five ways to execute the tasks asyncronously by using the ExecutorService interface provided Java 6.   
  
**ExecutorService execService = Executors.newCachedThreadPool();**   
  
This approach creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. These pools will typically improve the performance of programs that execute many short-lived asynchronous tasks. If no existing thread is available, a new thread will be created and added to the pool. Threads that have not been used for 60 seconds are terminated and removed from the cache.   
  
**ExecutorService execService = Executors.newFixedThreadPool(10);**   
  
This approach creates a thread pool that reuses a fixed number of threads. Created nThreads will be active at the runtime. If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available.   
  
**ExecutorService execService = Executors.newSingleThreadExecutor();**   
  
This approach creates an Executor that uses a single worker thread operating off an unbounded queue. Tasks are guaranteed to execute sequentially, and no more than one task will be active at any given time.   
  
**Methods of the ExecutorService :**   
  
**execute(Runnable) :** Executes the given command at some time in the future.   
  
**submit(Runnable) :** Submit method returns a Future Object which represents executed task. Future Object returns null if the task has finished correctly.   
  
**shutdown() :** Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted. Invocation has no additional effect if already shut down.   
  
**shutdownNow() :** Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.   
  
There are no guarantees beyond best-effort attempts to stop processing actively executing tasks. For example, typical implementations will cancel via Thread.interrupt, so any task that fails to respond to interrupts may never terminate.   
  
A sample application is below : 

## ****STEP 1 : CREATE MAVEN PROJECT****

A maven project is created as below. (It can be created by using Maven or IDE Plug-in). 

## ****STEP 2 : CREATE A NEW TASK****

A new task is created by implementing the Runnable interface(creating Thread) as below. TestTask Class specifies business logic which will be executed. 

package com.otv.task;

import org.apache.log4j.Logger;

/\*\*

\* @author onlinetechvision.com

\* @since 24 Sept 2011

\* @version 1.0.0

\*

\*/

public class TestTask implements Runnable {

private static Logger log = Logger.getLogger(TestTask.class);

private String taskName;

public TestTask(String taskName) {

this.taskName = taskName;

}

public void run() {

try {

log.debug(this.taskName + " is sleeping...");

Thread.sleep(3000);

log.debug(this.taskName + " is running...");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

## ****STEP 3 : CREATE TestExecutorService by using newCachedThreadPool****

TestExecutorService is created by using the method newCachedThreadPool. In this case, created thread count is specified at the runtime. 

package com.otv;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import com.otv.task.TestTask;

/\*\*

\* @author onlinetechvision.com

\* @since 24 Sept 2011

\* @version 1.0.0

\*

\*/

public class TestExecutorService {

public static void main(String[] args) {

ExecutorService execService = Executors.newCachedThreadPool();

execService.execute(new TestTask("FirstTestTask"));

execService.execute(new TestTask("SecondTestTask"));

execService.execute(new TestTask("ThirdTestTask"));

execService.shutdown();

}

}

When TestExecutorService is run, the output will be seen as below : 

24.09.2011 17:30:47 DEBUG (TestTask.java:21) - SecondTestTask is sleeping...

24.09.2011 17:30:47 DEBUG (TestTask.java:21) - ThirdTestTask is sleeping...

24.09.2011 17:30:47 DEBUG (TestTask.java:21) - FirstTestTask is sleeping...

24.09.2011 17:30:50 DEBUG (TestTask.java:23) - ThirdTestTask is running...

24.09.2011 17:30:50 DEBUG (TestTask.java:23) - FirstTestTask is running...

24.09.2011 17:30:50 DEBUG (TestTask.java:23) - SecondTestTask is running...

## ****STEP 4 : CREATE TestExecutorService by using newFixedThreadPool****

TestExecutorService is created by using the method newFixedThreadPool. In this case, required thread count has to be set as the following :  

package com.otv;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import com.otv.task.TestTask;

/\*\*

\* @author onlinetechvision.com

\* @since 24 Sept 2011

\* @version 1.0.0

\*

\*/

public class TestExecutorService {

public static void main(String[] args) {

ExecutorService execService = Executors.newFixedThreadPool(2);

execService.execute(new TestTask("FirstTestTask"));

execService.execute(new TestTask("SecondTestTask"));

execService.execute(new TestTask("ThirdTestTask"));

execService.shutdown();

}

}

When TestExecutorService is run, ThirdTestTask is executed after FirstTestTask and SecondTestTask’ s executions are completed. The output will be seen as below: 

24.09.2011 17:33:38 DEBUG (TestTask.java:21) - FirstTestTask is sleeping...

24.09.2011 17:33:38 DEBUG (TestTask.java:21) - SecondTestTask is sleeping...

24.09.2011 17:33:41 DEBUG (TestTask.java:23) - FirstTestTask is running...

24.09.2011 17:33:41 DEBUG (TestTask.java:23) - SecondTestTask is running...

24.09.2011 17:33:41 DEBUG (TestTask.java:21) - ThirdTestTask is sleeping...

24.09.2011 17:33:44 DEBUG (TestTask.java:23) - ThirdTestTask is running...

## ****STEP 5 : CREATE TestExecutorService by using newSingleThreadExecutor****

TestExecutorService is created by using the method newSingleThreadExecutor. In this case, only one thread is created and tasks are executed sequentially. 

package com.otv;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import com.otv.task.TestTask;

/\*\*

\* @author onlinetechvision.com

\* @since 24 Sept 2011

\* @version 1.0.0

\*

\*/

public class TestExecutorService {

public static void main(String[] args) {

ExecutorService execService = Executors.newSingleThreadExecutor();

execService.execute(new TestTask("FirstTestTask"));

execService.execute(new TestTask("SecondTestTask"));

execService.execute(new TestTask("ThirdTestTask"));

execService.shutdown();

}

}

When TestExecutorService is run, SecondTestTask and ThirdTestTask is executed after FirstTestTask’ s execution is completed. The output will be seen as below : 

24.09.2011 17:38:21 DEBUG (TestTask.java:21) - FirstTestTask is sleeping...

24.09.2011 17:38:24 DEBUG (TestTask.java:23) - FirstTestTask is running...

24.09.2011 17:38:24 DEBUG (TestTask.java:21) - SecondTestTask is sleeping...

24.09.2011 17:38:27 DEBUG (TestTask.java:23) - SecondTestTask is running...

24.09.2011 17:38:27 DEBUG (TestTask.java:21) - ThirdTestTask is sleeping...

24.09.2011 17:38:30 DEBUG (TestTask.java:23) - ThirdTestTask is running...

**Benefits Of Using The Thread Pool**

1. **Better performance** : Creating a brand new thread each time a new task arrives is expensive. Threadpool saves time because it re-uses the threads once created, to run future runnables.
2. You do not have to create, manage, schedule, and terminate your thread, Executor framework does all of this for you. In other words, Thread Pool frees application developer from thread management stuff and allows to focus on business logic.
3. Use of Threadpool allows you to change your execution policy as you need. you can go from single thread to multiple threads by just replacing ExecutorService implementation.

**Limitations Of Using The Thread Pool**

While the thread pool is a powerful mechanism for structuring multithreaded applications, it is not without risk such as pool-related deadlock, resource thrashing and thread leakage.

1. **Resource thrashing**

If the thread pool size is not tuned properly threads consume numerous resources, including memory and other system resources. While the scheduling overhead of switching between threads is small, with many threads context switching can become a significant drag on your program’s performance.

1. **Thread leakage**

A significant risk in all kinds of thread pools is thread leakage, which occurs when a thread is removed from the pool to perform a task, but is not returned to the pool when the task completes.

One way this happens is when the task throws a RuntimeException or an Error. If the pool class does not catch these, then the thread will simply exit and the size of the thread pool will be permanently reduced by one.

When this happens enough times, the thread pool will eventually be empty, and the system will stall because no threads are available to process tasks.

While using Executor framework, we don't need to worry about thread leakages anymore.

# ThreadPoolExecutor – Java Thread Pool Example: End

# static keyword in java

[**2.3**](http://www.geeksforgeeks.org/easy/)

static is a non-access modifier in Java which is applicable for the following:

1. blocks
2. variables
3. methods
4. nested classes

To create a static member(block,variable,method,nested class), precede its declaration with the keyword static. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. For example, in below java program, we are accessing static method m1() without creating any object of Test class.

|  |
| --- |
| // Java program to demonstrate that a static member  // can be accessed before instantiating a class  class Test  {      // static method      static void m1()      {          System.out.println("from m1");      }        public static void main(String[] args)      {            // calling m1 without creating            // any object of class Test             m1();      }  } |

Run on IDE

Output:

from m1

**Static blocks**

If you need to do computation to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded. Consider the following java program demonstrating use of static blocks.

|  |
| --- |
| // Java program to demonstrate use of static blocks  class Test  {      // static variable      static int a = 10;      static int b;        // static block      static {          System.out.println("Static block initialized.");          b = a \* 4;      }        public static void main(String[] args)      {         System.out.println("from main");         System.out.println("Value of a : "+a);         System.out.println("Value of b : "+b);      }  } |

Run on IDE

Output:

Static block initialized.

from main

Value of a : 10

Value of b : 40

For Detailed article on static blocks, see [static blocks](http://www.geeksforgeeks.org/g-fact-79/)

**Static variables**

When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**Important points for static variables:-**

* We can create static variables at class-level only. See [here](http://www.geeksforgeeks.org/g-fact-47/)
* static block and static variables are executed in order they are present in a program.

Below is the java program to demonstrate that static block and static variables are executed in order they are present in a program.

|  |
| --- |
| // java program to demonstrate execution  // of static blocks and variables  class Test  {      // static variable      static int a = m1();        // static block      static {          System.out.println("Inside static block");      }        // static method      static int m1() {          System.out.println("from m1");          return 20;      }        // static method(main !!)      public static void main(String[] args)      {         System.out.println("Value of a : "+a);         System.out.println("from main");      }      } |

Run on IDE

Output:

from m1

Inside static block

Value of a : 20

from main

**Static methods**

When a method is declared with static keyword, it is known as static method. The most common example of a static method is main( ) method.As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object.Methods declared as static have several restrictions:

* They can only directly call other static methods.
* They can only directly access static data.
* They cannot refer to [this](http://www.geeksforgeeks.org/this-reference-in-java/) or [super](http://www.geeksforgeeks.org/super-keyword/) in any way.

Below is the java program to demonstrate restrictions on static methods.

|  |
| --- |
| // java program to demonstrate restriction on static methods  class Test  {      // static variable      static int a = 10;        // instance variable      int b = 20;        // static method      static void m1()      {          a = 20;          System.out.println("from m1");             // Cannot make a static reference to the non-static field b           b = 10; // compilation error             // Cannot make a static reference to the                   // non-static method m2() from the type Test           m2();  // compilation error             //  Cannot use super in a static context           System.out.println(super.a); // compiler error      }        // instance method      void m2()      {          System.out.println("from m2");      }            public static void main(String[] args)      {          // main method      }  } |

Run on IDE

**When to use static variables and methods?**

Use the static variable for the property that is common to all objects. For example, in class Student, all students shares the same college name. Use static methods for changing static variables.

Consider the following java program, that illustrate the use of static keyword with variables and methods.

|  |
| --- |
| // A java program to demonstrate use of  // static keyword with methods and variables    // Student class  class Student  {      String name;      int rollNo;        // static variable      static String cllgName;        // static counter to set unique roll no      static int counter = 0;          public Student(String name)      {          this.name = name;            this.rollNo = setRollNo();      }        // getting unique rollNo      // through static variable(counter)      static int setRollNo()      {          counter++;          return counter;      }        // static method      static void setCllg(String name){          cllgName = name ;      }        // instance method      void getStudentInfo(){          System.out.println("name : " + this.name);          System.out.println("rollNo : " + this.rollNo);            // accessing static variable          System.out.println("cllgName : " + cllgName);      }  }    //Driver class  public class StaticDemo  {      public static void main(String[] args)      {          // calling static method          // without instantiating Student class          Student.setCllg("XYZ");            Student s1 = new Student("Alice");          Student s2 = new Student("Bob");            s1.getStudentInfo();          s2.getStudentInfo();        }  } |

Run on IDE

Output:

name : Alice

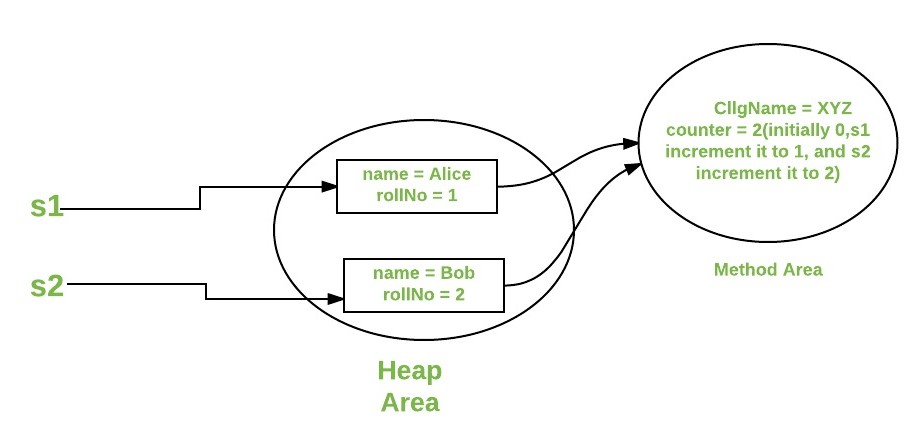
rollNo : 1

cllgName : XYZ

name : Bob

rollNo : 2

cllgName : XYZ

  
**Static nested classes :**We can not declare top-level class with a static modifier, but can declare [nested classes](http://www.geeksforgeeks.org/nested-classes-java/) as static. Such type of classes are called Nested static classes. For static nested class, see [static nested class in java](http://www.geeksforgeeks.org/static-class-in-java/)

# Static class in Java

[**3**](http://www.geeksforgeeks.org/medium/)

**Can a class be static in Java ?**  
The answer is YES, we can have static class in java. In java, we have [static instance variables](http://www.geeksforgeeks.org/static-keyword-in-java/) as well as [static methods](http://www.geeksforgeeks.org/static-keyword-in-java/) and also [static block](http://www.geeksforgeeks.org/g-fact-79/). Classes can also be made static in Java.

Java allows us to define a class within another class. Such a class is called a nested class. The class which enclosed nested class is known as Outer class. In java, we can’t make Top level class static. **Only nested classes can be static**.

**What are the differences between static and non-static nested classes?**  
Following are major differences between static nested class and non-static nested class. Non-static nested class is also called Inner Class.

**1)** Nested static class doesn’t need reference of Outer class, but Non-static nested class or Inner class requires Outer class reference.

**2)** Inner class(or non-static nested class) can access both static and non-static members of Outer class. A static class cannot access non-static members of the Outer class. It can access only static members of Outer class.

**3)**An instance of Inner class cannot be created without an instance of outer class and an Inner class can reference data and methods defined in Outer class in which it nests, so we don’t need to pass reference of an object to the constructor of the Inner class. For this reason Inner classes can make program simple and concise.

|  |
| --- |
| /\* Java program to demonstrate how to implement static and non-static     classes in a java program. \*/  class OuterClass{     private static String msg = "GeeksForGeeks";       // Static nested class     public static class NestedStaticClass{           // Only static members of Outer class is directly accessible in nested         // static class         public void printMessage() {             // Try making 'message' a non-static variable, there will be           // compiler error           System.out.println("Message from nested static class: " + msg);         }      }        // non-static nested class - also called Inner class      public class InnerClass{           // Both static and non-static members of Outer class are accessible in         // this Inner class         public void display(){            System.out.println("Message from non-static nested class: "+ msg);         }      }  }  class Main  {      // How to create instance of static and non static nested class?      public static void main(String args[]){           // create instance of nested Static class         OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();           // call non static method of nested static class         printer.printMessage();           // In order to create instance of Inner class we need an Outer class         // instance. Let us create Outer class instance for creating         // non-static nested class         OuterClass outer = new OuterClass();         OuterClass.InnerClass inner  = outer.new InnerClass();           // calling non-static method of Inner class         inner.display();           // we can also combine above steps in one step to create instance of         // Inner class         OuterClass.InnerClass innerObject = new OuterClass().new InnerClass();           // similarly we can now call Inner class method         innerObject.display();      }  } |

Run on IDE

Output:

Message from nested static class: GeeksForGeeks

Message from non-static nested class: GeeksForGeeks

Message from non-static nested class: GeeksForGeeks

# Nested Classes in Java

[**2.3**](http://www.geeksforgeeks.org/easy/)

In java, it is possible to define a class within another class, such classes are known as nested classes. They enable you to logically group classes that are only used in one place, thus this increases the use of [encapsulation](http://www.geeksforgeeks.org/encapsulation-in-java/), and create more readable and maintainable code.

* The scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class NestedClass does not exist independently of class OuterClass.
* A nested class has access to the members, including private members, of the class in which it is nested. However, reverse is not true i.e. the enclosing class does not have access to the members of the nested class.
* A nested class is also a member of its enclosing class.
* As a member of its enclosing class, a nested class can be declared private, public, protected, or package private(default).
* Nested classes are divided into two categories:
  1. **static nested class :**Nested classes that are declared static are called static nested classes.
  2. **inner class :**An inner class is a non-static nested class.

**Syntax:**

class OuterClass

{

...

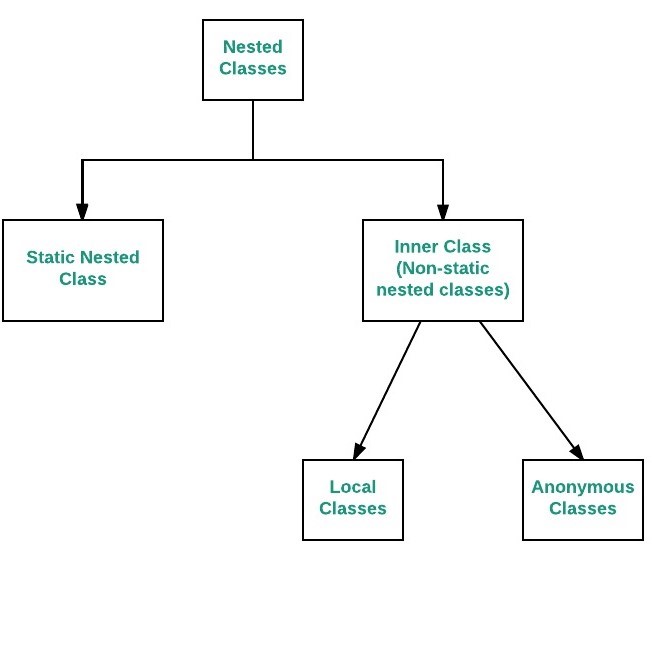
class NestedClass

{

...

}

}

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/d3.jpeg)

**Static nested classes**

As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.  
They are accessed using the enclosing class name.

OuterClass.StaticNestedClass

For example, to create an object for the static nested class, use this syntax:

OuterClass.StaticNestedClass nestedObject =

new OuterClass.StaticNestedClass();

|  |
| --- |
| // Java program to demonstrate accessing  // a static nested class    // outer class  class OuterClass  {      // static member      static int outer\_x = 10;        // instance(non-static) member      int outer\_y = 20;        // private member      private static int outer\_private = 30;        // static nested class      static class StaticNestedClass      {          void display()          {              // can access static member of outer class              System.out.println("outer\_x = " + outer\_x);                // can access display private static member of outer class              System.out.println("outer\_private = " + outer\_private);                // The following statement will give compilation error              // as static nested class cannot directly access non-static membera              // System.out.println("outer\_y = " + outer\_y);            }      }  }    // Driver class  public class StaticNestedClassDemo  {      public static void main(String[] args)      {          // accessing a static nested class          OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();            nestedObject.display();        }  } |

Run on IDE

Output:

outer\_x = 10

outer\_private = 30

**Inner classes**

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

OuterClass.InnerClass innerObject = outerObject.new InnerClass();

**There are two special kinds of inner classes :**

1. [Local inner classes](http://www.geeksforgeeks.org/local-inner-class-java/)
2. [Anonymous inner classes](http://www.geeksforgeeks.org/anonymous-inner-class-java/)

|  |
| --- |
| // Java program to demonstrate accessing  // a inner class    // outer class  class OuterClass  {      // static member      static int outer\_x = 10;        // instance(non-static) member      int outer\_y = 20;        // private member      private int outer\_private = 30;        // inner class      class InnerClass      {          void display()          {              // can access static member of outer class              System.out.println("outer\_x = " + outer\_x);                // can also access non-static member of outer class              System.out.println("outer\_y = " + outer\_y);                // can also access private member of outer class              System.out.println("outer\_private = " + outer\_private);            }      }  }    // Driver class  public class InnerClassDemo  {      public static void main(String[] args)      {          // accessing an inner class          OuterClass outerObject = new OuterClass();          OuterClass.InnerClass innerObject = outerObject.new InnerClass();            innerObject.display();        }  } |

Run on IDE

Output:

outer\_x = 10

outer\_y = 20

outer\_private = 30

We use inner class only when we are going to deal with the object that has no meaning outside of its outer class. You can consider an dummy, real example of inner class in programming context that is an Address class can have PhoneNumbers inner class which holds all the phone numbers associated with an address and may have some extra functionality, like returning the best phone number, the most used one, etc. You need a class to do all these extra things, but you don´t really need it in a separate class, since you do not need to deal with phone numbers outside the context of an Address.

**Difference between static and inner(non-static nested) classes**

* Static nested classes do not directly have access to other members(non-static variables and methods) of the enclosing class because as it is static, it must access the non-static members of its enclosing class through an object. That is, it cannot refer to non-static members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.
* Non-static nested classes (inner classes) has access to all members(static and non-static variables and methods, including private) of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.

# Can we Overload or Override static methods in java ?

[**3.5**](http://www.geeksforgeeks.org/medium/)

Let us first define Overloading and Overriding.

[**Overriding**](http://www.geeksforgeeks.org/overriding-in-java/): Overriding is a feature of OOP languages like Java that is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class).  
The implementation to be executed is decided at run-time and decision is made according to the object used for call. Note that signatures of both methods must be same. Refer [Overriding in Java](http://www.geeksforgeeks.org/overriding-in-java/) for details.

[**Overloading**](http://www.geeksforgeeks.org/overloading-in-java/): Overloading is also a feature of OOP languages like Java that is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters. Note that in both C++ and Java, [methods cannot be overloaded according to return type.](http://www.geeksforgeeks.org/g-fact-75/)

**Can we overload static methods?**  
The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |

Run on IDE

Output:

Test.foo() called

Test.foo(int) called

**Can we overload methods that differ only by static keyword?**  
We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example. This behaviour is same in C++ (See point 2 of [this](http://www.geeksforgeeks.org/function-overloading-in-c/)).

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public void foo() { // Compiler Error: cannot redefine foo()          System.out.println("Test.foo(int) called ");      }      public static void main(String args[]) {          Test.foo();      }  } |

Run on IDE

Output: Compiler Error, cannot redefine foo()

**Can we Override static methods in java?**  
We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’.  
If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

|  |
| --- |
| /\* Java program to show that if static method is redefined by     a derived class, then it is not overriding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // This method hides display() in Base      public static void display() {           System.out.println("Static or class method from Derived");      }        // This method overrides print() in Base      public void print() {           System.out.println("Non-static or Instance method from Derived");     }  }    // Driver class  public class Test {      public static void main(String args[ ])  {         Base obj1 = new Derived();           // As per overriding rules this should call to class Derive's static         // overridden method. Since static method can not be overridden, it         // calls Base's display()         obj1.display();           // Here overriding works and Derive's print() is called         obj1.print();      }  } |

Run on IDE

Output:

Static or class method from Base

Non-static or Instance method from Derived

Following are some important points for method overriding and static methods in Java.  
**1)** For class (or static) methods, the method according to the type of reference is called, not according to the abject being referred, which means method call is decided at compile time.

**2)** For instance (or non-static) methods, the method is called according to the type of object being referred, not according to the type of reference, which means method calls is decided at run time.

**3)** An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors.

|  |
| --- |
| /\* Java program to show that if static methods are redefined by     a derived class, then it is not overriding but hidding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // Static is removed here (Causes Compiler Error)      public void display() {          System.out.println("Non-static method from Derived");      }        // Static is added here (Causes Compiler Error)      public static void print() {          System.out.println("Static method from Derived");     }  } |

Run on IDE

**4)** In a subclass (or Derived Class), we can overload the methods inherited from the superclass. Such overloaded methods neither hide nor override the superclass methods — they are new methods, unique to the subclass.

**References:**

# Can we override private methods in Java?

[**3.3**](http://www.geeksforgeeks.org/medium/)

Let us first consider the following Java program as a simple example of Overriding or Runtime Polymorphism.

|  |
| --- |
| class Base {    public void fun() {       System.out.println("Base fun");    }  }    class Derived extends Base {    public void fun() {  // overrides the Base's fun()       System.out.println("Derived fun");    }    public static void main(String[] args) {        Base obj = new Derived();        obj.fun();    }  } |

Run on IDE

The program prints “Derived fun”.  
The Base class reference ‘obj’ refers to a derived class object (see expression “Base obj = new Derived()”). When fun() is called on obj, the call is made according to the type of referred object, not according to the reference.

**Is Overiding possible with private methods?**  
Predict the output of following program.

|  |
| --- |
| class Base {    private void fun() {       System.out.println("Base fun");    }  }    class Derived extends Base {    private void fun() {       System.out.println("Derived fun");    }    public static void main(String[] args) {        Base obj = new Derived();        obj.fun();    }  } |

Run on IDE

We get compiler error “fun() has private access in Base” (See [this](http://ideone.com/arKk3c)). So the compiler tries to call base class function, not derived class, means fun() is not overridden.

**An inner class can access private members of its outer class. What if we extend an inner class and create fun() in the inner class?**  
An Inner classes can access private members of its outer class, for example in the following program, fun() of Inner accesses private data member msg which is fine by the compiler.

|  |
| --- |
| /\* Java program to demonstrate whether we can override private method     of outer class inside its inner class \*/  class Outer {       private String msg = "GeeksforGeeks";       private void fun() {            System.out.println("Outer fun()");       }         class Inner extends Outer {           private void fun()  {                 System.out.println("Accessing Private Member of Outer: " + msg);           }       }         public static void main(String args[])  {              // In order to create instance of Inner class, we need an Outer            // class instance. So, first create Outer class instance and then            // inner class instance.            Outer o = new Outer();            Inner  i   = o.new Inner();              // This will call Inner's fun, the purpose of this call is to            // show that private members of Outer can be accessed in Inner.            i.fun();              // o.fun() calls Outer's fun (No run-time polymorphism).            o = i;            o.fun();       }  } |

Run on IDE

Output:

Accessing Private Member of Outer: GeeksforGeeks

Outer fun()

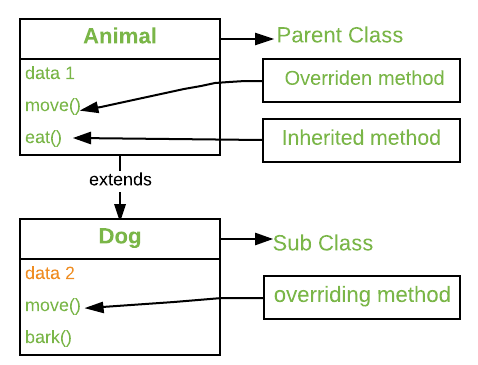
In the above program, we created an outer class and an inner class. We extended Inner from Outer and created a method fun() in both Outer and Inner. If we observe our output, then it is clear that the method fun() has not been overriden. It is so because ***private methods are bonded during compile time and it is the type of the reference variable – not the type of object that it refers to – that determines what method to be called.***. As a side note, private methods may be performance-wise better (compared to non-private and non-final methods) due to static binding.

***Comparison With C++***  
**1)**In Java, inner Class is allowed to access private data members of outer class. This behavior is same as C++ (See [this](http://www.geeksforgeeks.org/nested-classes-in-c/)).  
**2)** In Java, methods declared as private can never be overridden, they are in-fact bounded during compile time. This behavior is different from C++. In C++, we can have virtual private methods (See [this](http://ideone.com/6645Uz)).

# Overriding in Java

[**2.6**](http://www.geeksforgeeks.org/easy/)

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.



Method overriding is one of the way by which java achieve [Run Time Polymorphism](http://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/).The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

|  |
| --- |
| // A Simple Java program to demonstrate  // method overriding in java    // Base Class  class Parent  {      void show() { System.out.println("Parent's show()"); }  }    // Inherited class  class Child extends Parent  {      // This method overrides show() of Parent      @Override      void show() { System.out.println("Child's show()"); }  }    // Driver class  class Main  {      public static void main(String[] args)      {          // If a Parent type reference refers          // to a Parent object, then Parent's          // show is called          Parent obj1 = new Parent();          obj1.show();            // If a Parent type reference refers          // to a Child object Child's show()          // is called. This is called RUN TIME          // POLYMORPHISM.          Parent obj2 = new Child();          obj2.show();      }  } |

Run on IDE

Output:

Parent's show()

Child's show()

**Rules for method overriding:**

1. **Overriding and Access-Modifiers :**The [access modifier](http://www.geeksforgeeks.org/access-modifiers-java/) for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the super-class can be made public, but not private, in the subclass. Doing so, will generate compile-time error.

|  |
| --- |
| // A Simple Java program to demonstrate  // Overriding and Access-Modifiers    class Parent  {      // private methods are not overridden      private void m1() { System.out.println("From parent m1()");}        protected void m2() { System.out.println("From parent m2()"); }  }    class Child extends Parent  {      // new m1() method      // unique to Child class      private void m1() { System.out.println("From child m1()");}        // overriding method      // with more accessibility      @Override      public void m2() { System.out.println("From child m2()");}    }    // Driver class  class Main  {      public static void main(String[] args)      {          Parent obj1 = new Parent();          obj1.m2();          Parent obj2 = new Child();          obj2.m2();      }  } |

1. Run on IDE
2. Output :
3. From parent m2()
4. From child m2()

1. **Final methods can not be overridden :**If we don’t want a method to be overridden, we declare it as [final](http://www.geeksforgeeks.org/final-keyword-java/). Please see [Using final with Inheritance](http://www.geeksforgeeks.org/using-final-with-inheritance-in-java/).

|  |
| --- |
| // A Java program to demonstrate that  // final methods cannot be overridden    class Parent  {      // Can't be overridden      final void show() {  }  }    class Child extends Parent  {      // This would produce error      void show() {  }  } |

1. Run on IDE
2. Output :
3. 13: error: show() in Child cannot override show() in Parent
4. void show() { }
5. ^
6. overridden method is final
7. **Static methods can not be overridden(Method Overriding vs Method Hiding) :**When you defines a static method with same signature as a static method in base class, it is known as [method hiding](http://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/).

The following table summarizes what happens when you define a method with the same signature as a method in a super-class.

|  |  |  |
| --- | --- | --- |
|  | **SUPERCLASS INSTANCE METHOD** | **SUPERCLASS STATIC METHOD** |
| **SUBCLASS INSTANCE METHOD** | Overrides | Generates a compile-time error |
| **SUBCLASS STATIC METHOD** | Generates a compile-time error | Hides |

|  |
| --- |
| /\* Java program to show that if static method is redefined by  a derived class, then it is not overriding,it is hiding \*/    class Parent  {      // Static method in base class which will be hidden in subclass      static void m1() { System.out.println("From parent static m1()");}        // Non-static method which will be overridden in derived class      void m2() { System.out.println("From parent non-static(instance) m2()"); }  }    class Child extends Parent  {      // This method hides m1() in Parent      static void m1() { System.out.println("From child static m1()");}        // This method overrides m2() in Parent      @Override      public void m2() { System.out.println("From child non-static(instance) m2()");}    }    // Driver class  class Main  {      public static void main(String[] args)      {          Parent obj1 = new Child();            // As per overriding rules this should call to class Child static          // overridden method. Since static method can not be overridden, it          // calls Parent's m1()          obj1.m1();            // Here overriding works and Child's m2() is called          obj1.m2();      }  } |

Run on IDE

Output :

From parent static m1()

From child non-static(instance) m2()

1. **Private methods can not be overridden :**[Private methods](http://www.geeksforgeeks.org/can-override-private-methods-java/)cannot be overridden as they are bonded during compile time. Therefore we can’t even override private methods in a subclass.(See [this](http://www.geeksforgeeks.org/can-override-private-methods-java/) for details).

1. **The overriding method must have same return type (or subtype) :**From Java 5.0 onwards it is possible to have different return type for a overriding method in child class, but child’s return type should be sub-type of parent’s return type. This phenomena is known as [**covariant return type**](http://www.geeksforgeeks.org/covariant-return-types-java/).

1. **Invoking overridden method from sub-class :**We can call parent class method in overriding method using [super keyword](http://quiz.geeksforgeeks.org/super-keyword/).

|  |
| --- |
| // A Java program to demonstrate that overridden  // method can be called from sub-class    // Base Class  class Parent  {      void show()      {          System.out.println("Parent's show()");      }  }    // Inherited class  class Child extends Parent  {      // This method overrides show() of Parent      @Override      void show()      {          super.show();          System.out.println("Child's show()");      }  }    // Driver class  class Main  {      public static void main(String[] args)      {          Parent obj = new Child();          obj.show();      }  } |

1. Run on IDE
2. Output:
3. Parent's show()
4. Child's show()
5. **Overriding and constructor :**We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).

1. **Overriding and Exception-Handling :**Below are two rules to note when overriding methods related to exception-handling.
   * **Rule#1 :** If the super-class overridden method does not throws an exception, subclass overriding method can only throws the [unchecked exception](http://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/), throwing checked exception will lead to compile-time error.

|  |
| --- |
| /\* Java program to demonstrate overriding when    superclass method does not declare an exception  \*/    class Parent  {      void m1() { System.out.println("From parent m1()");}        void m2() { System.out.println("From parent  m2()"); }  }    class Child extends Parent  {      @Override      // no issue while throwing unchecked exception      void m1() throws ArithmeticException      { System.out.println("From child m1()");}        @Override      // compile-time error      // issue while throwin checked exception      void m2() throws Exception{ System.out.println("From child m2");}    } |

* + Run on IDE
  + Output:
  + error: m2() in Child cannot override m2() in Parent
  + void m2() throws Exception{ System.out.println("From child m2");}
  + ^
  + overridden method does not throw Exception
  + **Rule#2 :** If the super-class overridden method does throws an exception, subclass overriding method can only throw same, subclass exception. Throwing parent exception in [Exception hierarchy](http://www.geeksforgeeks.org/exceptions-in-java/) will lead to compile time error.Also there is no issue if subclass overridden method is not throwing any exception.

|  |
| --- |
| /\* Java program to demonstrate overriding when    superclass method does declare an exception  \*/    class Parent  {      void m1() throws RuntimeException      { System.out.println("From parent m1()");}    }    class Child1 extends Parent  {      @Override      // no issue while throwing same exception      void m1() throws RuntimeException      { System.out.println("From child1 m1()");}    }  class Child2 extends Parent  {      @Override      // no issue while throwing subclass exception      void m1() throws ArithmeticException      { System.out.println("From child2 m1()");}    }  class Child3 extends Parent  {      @Override      // no issue while not throwing any exception      void m1()      { System.out.println("From child3 m1()");}    }  class Child4 extends Parent  {      @Override      // compile-time error      // issue while throwing parent exception      void m1() throws Exception      { System.out.println("From child4 m1()");}    } |

* + Run on IDE
  + Output:
  + error: m1() in Child4 cannot override m1() in Parent
  + void m1() throws Exception
  + ^
  + overridden method does not throw Exception

1. **Overriding and abstract method :**Abstract methods in an interface or abstract class are meant to be overridden in derived concrete classes otherwise compile-time error will be thrown.

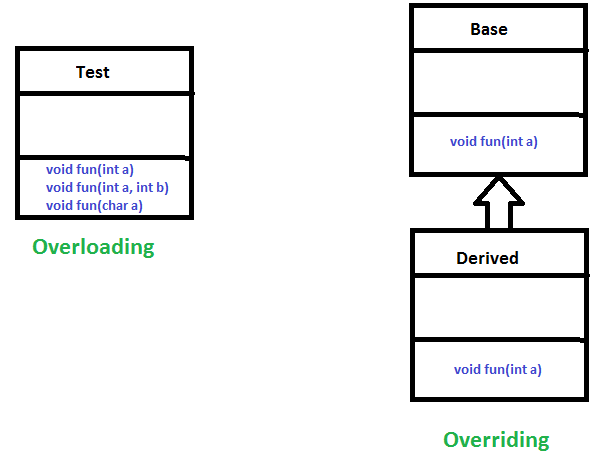
1. **Overriding and synchronized/stricfp method :**The presence of synchronized/stricfp modifier with method have no effect on the rules of overriding, i.e. it’s possible that a synchronized/stricfp method can override a non synchronized/stricfp one and vice-versa.

**Note :**

* 1. In C++, we need[virtual keyword](http://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-c-set-1-introduction/) to achieve overriding or [Run Time Polymorphism](http://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/). In Java, methods are virtual by default.
  2. We can have multilevel method-overriding.

|  |
| --- |
| // A Java program to demonstrate  //  multi-level overriding    // Base Class  class Parent  {      void show() { System.out.println("Parent's show()"); }  }    // Inherited class  class Child extends Parent  {      // This method overrides show() of Parent      void show() { System.out.println("Child's show()"); }  }    // Inherited class  class GrandChild extends Child  {      // This method overrides show() of Parent      void show() { System.out.println("GrandChild's show()"); }  }    // Driver class  class Main  {      public static void main(String[] args)      {          Parent obj1 = new GrandChild();          obj1.show();      }  } |

* 1. Run on IDE
  2. Output :
  3. GrandChild's show()
  4. **Overriding vs**[**Overloading**](http://www.geeksforgeeks.org/overloading-in-java/)**:**
     + 1. Overloading is about same method have different signatures. Overriding is about same method, same signature but different classes connected through inheritance.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/OverridingVsOverloading.png)

* + - 1. Overloading is an example of compiler-time polymorphism and overriding is an example of [run time polymorphism](http://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/).

**Why Method Overriding ?**

As stated earlier, overridden methods allow Java to support [run-time polymorphism](http://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/). Polymorphism is essential to object-oriented programming for one reason: it allows a general class to specify methods that will be common to all of its derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.Overridden methods are another way that Java implements the “one interface, multiple methods” aspect of polymorphism.

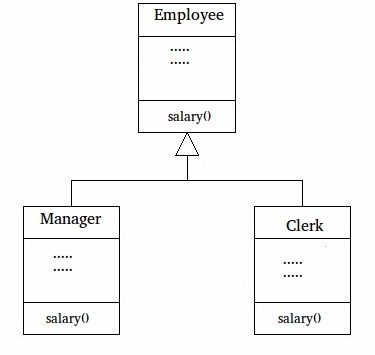
[Dynamic Method Dispatch](http://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/) is one of the most powerful mechanisms that object oriented design brings to bear on code reuse and robustness. The ability of existing code libraries to call methods on instances of new classes without recompiling while maintaining a clean abstract interface is a profoundly powerful tool.

Overridden methods allow us to call methods of any of the derived classes without even knowing type of derived class object.

**When to apply Method Overriding ?(with example)**

**Overriding and**[**Inheritance**](http://www.geeksforgeeks.org/inheritance-in-java/) : Part of the key to successfully applying polymorphism is understanding that the superclasses and subclasses form a hierarchy which moves from lesser to greaterspecialization. Used correctly, the superclass provides all elements that a subclass can use directly. It also defines those methods that the derived class must implement on its own. This allows the subclass the flexibility to define its own methods, yet still enforces a consistent interface. **Thus, by combining inheritance with overridden methods, a superclass can define the general form of the methods that will be used by all of its subclasses.**

Let’s look at a more practical example that uses method overriding. Consider a employee management software for an organization, let the code has a simple base class Employee, the class has methods like raiseSalary(), transfer(), promote(),.. etc. Different types of employees like Manager, Engineer, ..etc may have their own implementations of the methods present in base class Employee. In our complete software, we just need to pass a list of employees everywhere and call appropriate methods without even knowing the type of employee. For example, we can easily raise salary of all employees by iterating through list of employees. Every type of employee may have its own logic in its class, we don’t need to worry because if raiseSalary() is present for a specific employee type, only that method would be called.

[](http://www.geeksforgeeks.org/wp-content/uploads/JavaOVerriding.jpg)

|  |
| --- |
| // A Simple Java program to demonstrate application  // of overriding in Java    // Base Class  class Employee  {      public static int base = 10000;      int salary()      {          return base;      }  }    // Inherited class  class Manager extends Employee  {      // This method overrides show() of Parent      int salary()      {          return base + 20000;      }  }    // Inherited class  class Clerk extends Employee  {      // This method overrides show() of Parent      int salary()      {          return base + 10000;      }  }    // Driver class  class Main  {      // This method can be used to print salary of      // any type of employee using base class refernce      static void printSalary(Employee e)      {          System.out.println(e.salary());      }        public static void main(String[] args)      {          Employee obj1 = new Manager();            // We could also get type of employee using          // one more overridden method.loke getType()          System.out.print("Manager's salary : ");          printSalary(obj1);            Employee obj2 = new Clerk();          System.out.print("Clerk's salary : ");          printSalary(obj2);      }  } |

Run on IDE

Output:

Manager's salary : 30000

Clerk's salary : 20000

Related Article :  
[Dynamic Method Dispatch or Runtime Polymorphism in Java](http://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/)  
[Overriding equals() method of Object class](http://www.geeksforgeeks.org/overriding-equals-method-in-java/)  
[Overriding toString() method of Object class](http://www.geeksforgeeks.org/overriding-tostring-method-in-java/)  
[Overloading in java](http://www.geeksforgeeks.org/overloading-in-java/)  
[Output of Java program | Set 18 (Overriding)](http://www.geeksforgeeks.org/output-java-program-set-18-overriding/)

This article is contributed by **Twinkle Tyagi and Gaurav Miglani**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

# Java.util.TreeMap.pollFirstEntry() and pollLastEntry() in Java

Java.util.TreeMap also contains functions that support retrieval and deletion at both, high and low end of values and hence give a lot of flexibility in applicability and daily use. This function is poll() and has 2 variants discussed in this article.  
  
**1. pollFirstEntry() :**It**removes and retrieves a key-value pair** with the **least key value** in the map and **“null”** is map is empty.

**Syntax :**

public Map.Entry pollFirstEntry()

**Parameters:**

NA

**Return Value:**

Retrieves and removes the least key-value if map is filled else returns null.

**Exception:**

NA

|  |
| --- |
| // Java code to demonstrate the working  // of pollFirstEntry()  import java.io.\*;  import java.util.\*;  public class pollFirstEntry {      public static void main(String[] args)      {            // Declaring the tree map of String and Integer          TreeMap<String, Integer> tmp = new TreeMap<String, Integer>();            // Trying to retrieve and remove in empty map          // returns null          System.out.println          ("The smallest key value pair is : " + tmp.pollFirstEntry());            // assigning the values in the tree map          // using put()          tmp.put("Geeks", 1);          tmp.put("for", 4);          tmp.put("geeks", 1);            // Printing the initial map          System.out.println          ("The initial Map before deletion is : " + tmp);            // Use of pollFirstEntry()          // Removes the first entry and returns the least key          // lexicographically smallest in case of String          // prints Geeks-1          System.out.println          ("The smallest key value pair is : " + tmp.pollFirstEntry());            // Printing the map after deletion          System.out.println          ("The resultant Map after deletion is : " + tmp);      }  } |

Run on IDE

Output:

The smallest key value pair is : null

The initial Map before deletion is : {Geeks=1, for=4, geeks=1}

The smallest key value pair is : Geeks=1

The resultant Map after deletion is : {for=4, geeks=1}

**2. pollLastEntry() :**It**removes and retrieves a key-value pair** with the **largest key value** in the map and **“null”** is map is empty.

**Syntax :**

public Map.Entry pollLastEntry()

**Parameters:**

NA

**Return Value:**

Retrieves and removes the largest key-value if map is filled else returns null.

**Exception:**

NA

|  |
| --- |
| // Java code to demonstrate the working  // of pollLastEntry()  import java.io.\*;  import java.util.\*;  public class pollLastEntry {      public static void main(String[] args)      {            // Declaring the tree map of String and Integer          TreeMap<String, Integer> tmp = new TreeMap<String, Integer>();            // Trying to retrieve and remove in empty map          // returns null          System.out.println          ("The largest key value pair is : " + tmp.pollFirstEntry());            // assigning the values in the tree map          // using put()          tmp.put("Geeks", 1);          tmp.put("for", 4);          tmp.put("geeks", 1);            // Printing the initial map          System.out.println          ("The initial Map before deletion is : " + tmp);            // Use of pollLastEntry()          // Removes the last(max) entry and returns the max key          // lexicographically greatest in case of String          // prints geeks-1          System.out.println          ("The largest key value pair is : " + tmp.pollLastEntry());            // Printing the map after deletion          System.out.println          ("The resultant Map after deletion is : " + tmp);      }  } |

Run on IDE

Output:

The largest key value pair is : null

The initial Map before deletion is : {Geeks=1, for=4, geeks=1}

The largest key value pair is : geeks=1

The resultant Map after deletion is : {Geeks=1, for=4}

**Practical Application :**There are many applications that can be thought using the concept of deque or priority queueing. One such example is shown in the code below.

|  |
| --- |
| // Java code to demonstrate the application  // of pollLastEntry() and pollFirstEntry()  import java.io.\*;  import java.util.\*;  public class pollAppli {      public static void main(String[] args)      {            // Declaring the tree map of Integer and String          TreeMap<Integer, String> que = new TreeMap<Integer, String>();            // assigning the values in que          // using put()          que.put(10, "astha");          que.put(4, "shambhavi");          que.put(7, "manjeet");          que.put(8, "nikhil");            // Defining the priority          // takes highest value, if priority is high          // else takes lowest value          String prio = "high";            // Printing the initial queue          System.out.println("The initial queue is : " + que);          if (prio == "high") {              System.out.println              ("The largest valued person is : " + que.pollLastEntry());              System.out.println              ("The resultant queue after deletion is : " + que);          }          else {              System.out.println              ("The lowest valued person is : " + que.pollFirstEntry());              System.out.println              ("The resultant queue after deletion is : " + que);          }      }  } |

Run on IDE

Output:

The initial queue is : {4=shambhavi, 7=manjeet, 8=nikhil, 10=astha}

The largest valued person is : 10=astha

The resultant queue after deletion is : {4=shambhavi, 7=manjeet, 8=nikhil}

Coupling in Java

[**1**](http://www.geeksforgeeks.org/basic/)

In object oriented design, Coupling refers to the degree of direct knowledge that one element has of another. In other words, how often do changes in class A force related changes in class B.  
**There are two types of coupling:**

1. **Tight coupling :**In general, Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.  
   **Example :**If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI(Remote Method Invocation).

|  |
| --- |
| // Java program to illustrate  // tight coupling concept  class Subject {      Topic t = new Topic();      public void startReading()      {          t.understand();      }  }  class Topic {      public void understand()      {          System.out.println("Tight coupling concept");      }  } |

1. **Explanation:** In the above program the Subject class is dependents on Topic class. In the above program Subject class is tightly coupled with Topic class it means if any change in the Topic class requires Subject class to change. For example, if Topic class understand() method change to gotit() method then you have to change the startReading() method will call gotit() method instead of calling understand() method.

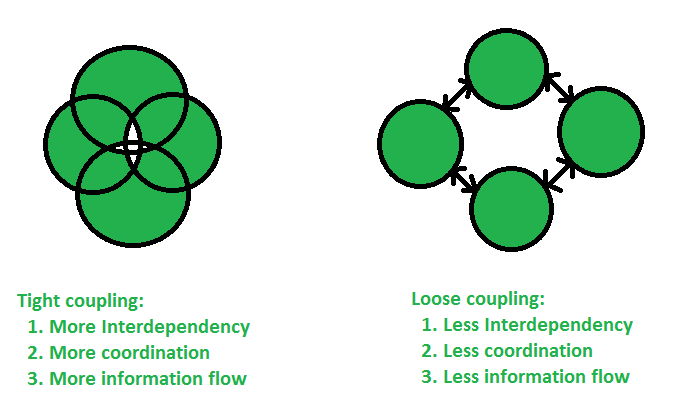
|  |
| --- |
| // Java program to illustrate  // tight coupling concept  class Volume  {       public static void main(String args[])       {           Box b = new Box(5,5,5);           System.out.println(b.volume);       }  }  class Box  {       public int volume;       Box(int length, int width, int height)       {           this.volume = length \* width \* height;       }  } |

1. Run on IDE
2. Output:
3. 125
4. **Explanation:**In the above example, there is a strong inter-dependency between both the classes. If there is any change in Box class then they reflects in the result of Class Volume.
5. **Loose coupling :**In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to over come from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling.  
   **Example :**If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. The examples of Loose coupling are Interface, JMS.

|  |
| --- |
| // Java program to illustrate  // loose coupling concept  public interface Topic  {      void understand();  }  class Topic1 implements Topic {  public void understand()      {          System.out.println("Got it");      }  } class Topic2 implements Topic {  public void unserstand()      {          System.out.println("understand");      }  } public class Subject {  public static void main(String[] args)      {          Topic t = new Topic1();          t.understand();      }  } |

1. **Explanation :**In the above example, Topic1 and Topic2 objects are loosely coupled. It means Topic is an interface and we can inject any of the implemented classes at run time and we can provide service to the end user.

|  |
| --- |
| // Java program to illustrate  // loose coupling concept  class Volume  {       public static void main(String args[])       {           Box b = new Box(5,5,5);           System.out.println(b.getVolume());       }  }  final class Box  {       private int volume;       Box(int length, int width, int height)       {           this.volume = length \* width \* height;       }       public int getVolume()       {           return volume;         }  } |

1. Run on IDE
2. Output:
3. 125
4. **Explanation :**In the above program, there is no dependency between both the classes. If we change anything in the Box classes then we dont have to change anything in Volume class.
5. **Which is better tight coupling or loose coupling?**
6. In general, Tight Coupling is bad in but most of the time, because it reduces flexibility and re-usability of code, it makes changes much more difficult, it impedes test ability etc. loose coupling is a better choice because A loosely coupled will help you when your application need to change or grow. If you design with loosely coupled architecture, only a few parts of the application should be affected when requirements change.  
   **Lets have a look on the pictorial view of tight coupling and loose coupling:**  
   
7. **Difference between tight coupling and loose coupling**
   * Tight coupling is not good at the test-ability. But loose coupling improves the test ability.
   * Tight coupling does not provide the concept of interface. But loose coupling helps us follow the GOF principle of program to interfaces, not implementations.
   * In Tight coupling, it is not easy to swap the codes between two classes. But it’s much easier to swap other pieces of code/modules/objects/components in loose coupling.
   * Tight coupling does not have the changing capability. But loose coupling is highly changeable.

Java – Read a file from resources folder

By [mkyong](http://www.mkyong.com/author/mkyong/) | July 30, 2014 | Viewed : 473,949 times +5,158 pv/w

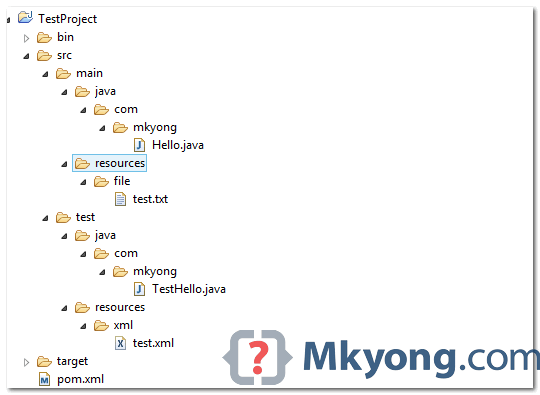
In this tutorial, we will show you how to read a file from a resources folder, in both Java and Unit Test environment. In simple, put files in a resources folder, and read the file with following code snippets :

ClassLoader classLoader = getClass().getClassLoader();

File file = new File(classLoader.getResource("file/test.xml").getFile());

1. Project Directory

Review a Maven project Structure.



2. Classic Example

Example to read a file “test.txt” from a resources folder.

main/resources/file/test.txt

This is line 1

This is line 2

This is line 3

This is line 4

This is line 5

Hello.java

package com.mkyong;

import java.io.File;

import java.io.IOException;

import java.util.Scanner;

public class Hello {

public static void main(String[] args) {

Hello obj = new Hello();

System.out.println(obj.getFile("file/test.txt"));

}

private String getFile(String fileName) {

StringBuilder result = new StringBuilder("");

//Get file from resources folder

ClassLoader classLoader = getClass().getClassLoader();

File file = new File(classLoader.getResource(fileName).getFile());

try (Scanner scanner = new Scanner(file)) {

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

result.append(line).append("\n");

}

scanner.close();

} catch (IOException e) {

e.printStackTrace();

}

return result.toString();

}

}

Output

This is line 1

This is line 2

This is line 3

This is line 4

This is line 5

3. IOUtils Example

This example uses IOUtils to parse a file.

pom.xml

<dependency>

<groupId>org.apache.directory.studio</groupId>

<artifactId>org.apache.commons.io</artifactId>

<version>2.4</version>

</dependency>

Hello.java

package com.mkyong;

import java.io.IOException;

import org.apache.commons.io.IOUtils;

public class Hello {

public static void main(String[] args) {

Hello obj = new Hello();

System.out.println(obj.getFileWithUtil("file/test.txt"));

}

private String getFileWithUtil(String fileName) {

String result = "";

ClassLoader classLoader = getClass().getClassLoader();

try {

result = IOUtils.toString(classLoader.getResourceAsStream(fileName));

} catch (IOException e) {

e.printStackTrace();

}

return result;

}

}

Output

This is line 1

This is line 2

This is line 3

This is line 4

This is line 5

4. Unit Test Example

A jUnit example.

test/resources/xml/test.xml

<test>

<case id=1>

<param>100</param>

<expected>mkyong</expected>

</case>

<case id=2>

<param>99</param>

<expected>mkyong</expected>

</case>

</test>

TestHello.java

package com.mkyong;

import java.io.IOException;

import org.apache.commons.io.IOUtils;

import org.junit.Test;

public class TestHello {

@Test

public void testHello() {

String result = getFile("xml/test.xml");

System.out.println(result);

}

private String getFile(String fileName){

String result = "";

ClassLoader classLoader = getClass().getClassLoader();

try {

result = IOUtils.toString(classLoader.getResourceAsStream(fileName));

} catch (IOException e) {

e.printStackTrace();

}

return result;

}

}

Output

<test>

<case id=1>

<param>100</param>

<expected>mkyong</expected>

</case>

<case id=2>

<param>99</param>

<expected>mkyong</expected>

</case>

</test>

Java – Convert String to int

By [mkyong](http://www.mkyong.com/author/mkyong/) | June 1, 2015 | Updated : July 1, 2015 | Viewed : 328,462 times +7,780 pv/w

In Java, you can use Integer.parseInt() to convert a String to int.

1. Integer.parseInt() Examples

Example to convert a String “10” to an primitive int.

String number = "10";

int result = Integer.parseInt(number);

System.out.println(result);

Output

10

2. Integer.valueOf() Examples

Alternatively, you can use Integer.valueOf(), it will returns an Integer object.

String number = "10";

Integer result = Integer.valueOf(number);

System.out.println(result);

Output

10

**Note**  
In summary, parseInt(String) returns a primitive int, whereas valueOf(String) returns a new Integer() object.

Java – Convert comma-separated String to a List

By [mkyong](http://www.mkyong.com/author/mkyong/) | April 14, 2017 | Viewed : 4,739 times +1,328 pv/w

Java examples to show you how to convert a comma-separated String into a List and vice versa.

1. Comma-separated String to List

TestApp1.java

package com.mkyong.utils;

import java.util.Arrays;

import java.util.List;

public class TestApp1 {

public static void main(String[] args) {

String alpha = "A, B, C, D";

//Remove whitespace and split by comma

List<String> result = Arrays.asList(alpha.split("\\s\*,\\s\*"));

System.out.println(result);

}

}

Output

[A, B, C, D]

2. List to Comma-separated String

No need to loop the List, uses the new Java 8 String.join

TestApp2.java

package com.mkyong.utils;

import java.util.Arrays;

import java.util.List;

public class TestApp2 {

public static void main(String[] args) {

List<String> list = Arrays.asList("A", "B", "C", "D");

String result = String.join(",", list);

System.out.println(result);

}

}

Output

A,B,C,D

How to sort a Map in Java

By [mkyong](http://www.mkyong.com/author/mkyong/) | July 7, 2010 | Updated : August 12, 2016 | Viewed : 381,166 times +1,946 pv/w

Few Java examples to sort a Map by its keys or values.

**Note**  
If you are using Java 8, refer to this article – [How to use Stream APIs to sort a Map](http://www.mkyong.com/java8/java-8-how-to-sort-a-map/)

1. Sort by Key

1.1 Uses java.util.TreeMap, it will sort the Map by keys automatically.

SortByKeyExample1.java

package com.mkyong.test;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

public class SortByKeyExample1 {

public static void main(String[] args) {

Map<String, String> unsortMap = new HashMap<String, String>();

unsortMap.put("Z", "z");

unsortMap.put("B", "b");

unsortMap.put("A", "a");

unsortMap.put("C", "c");

unsortMap.put("D", "d");

unsortMap.put("E", "e");

unsortMap.put("Y", "y");

unsortMap.put("N", "n");

unsortMap.put("J", "j");

unsortMap.put("M", "m");

unsortMap.put("F", "f");

System.out.println("Unsort Map......");

printMap(unsortMap);

System.out.println("\nSorted Map......By Key");

Map<String, String> treeMap = new TreeMap<String, String>(unsortMap);

printMap(treeMap);

}

//pretty print a map

public static <K, V> void printMap(Map<K, V> map) {

for (Map.Entry<K, V> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey()

+ " Value : " + entry.getValue());

}

}

}

Output

Unsort Map......

Key : A Value : a

Key : B Value : b

Key : C Value : c

Key : D Value : d

Key : E Value : e

Key : F Value : f

Key : Y Value : y

Key : Z Value : z

Key : J Value : j

Key : M Value : m

Key : N Value : n

Sorted Map......By Key

Key : A Value : a

Key : B Value : b

Key : C Value : c

Key : D Value : d

Key : E Value : e

Key : F Value : f

Key : J Value : j

Key : M Value : m

Key : N Value : n

Key : Y Value : y

Key : Z Value : z

1.2 Yet another java.util.TreeMap example, provide a custom Comparator to sort the key in descending order.

SortByKeyExample2.java

package com.mkyong.test;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

public class SortByKeyExample2 {

public static void main(String[] args) {

Map<Integer, String> unsortMap = new HashMap<Integer, String>();

unsortMap.put(10, "z");

unsortMap.put(5, "b");

unsortMap.put(6, "a");

unsortMap.put(20, "c");

unsortMap.put(1, "d");

unsortMap.put(7, "e");

unsortMap.put(8, "y");

unsortMap.put(99, "n");

unsortMap.put(50, "j");

unsortMap.put(2, "m");

unsortMap.put(9, "f");

System.out.println("Unsort Map......");

printMap(unsortMap);

System.out.println("\nSorted Map......By Key");

Map<Integer, String> treeMap = new TreeMap<Integer, String>(

new Comparator<Integer>() {

@Override

public int compare(Integer o1, Integer o2) {

return o2.compareTo(o1);

}

});

/\* For Java 8, try this lambda

Map<Integer, String> treeMap = new TreeMap<>(

(Comparator<Integer>) (o1, o2) -> o2.compareTo(o1)

);

\*/

treeMap.putAll(unsortMap);

printMap(treeMap);

}

public static <K, V> void printMap(Map<K, V> map) {

for (Map.Entry<K, V> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey()

+ " Value : " + entry.getValue());

}

}

}

Output

Unsort Map......

Key : 1 Value : d

Key : 50 Value : j

Key : 2 Value : m

Key : 99 Value : n

Key : 20 Value : c

Key : 5 Value : b

Key : 6 Value : a

Key : 7 Value : e

Key : 8 Value : y

Key : 9 Value : f

Key : 10 Value : z

Sorted Map......By Key

Key : 99 Value : n

Key : 50 Value : j

Key : 20 Value : c

Key : 10 Value : z

Key : 9 Value : f

Key : 8 Value : y

Key : 7 Value : e

Key : 6 Value : a

Key : 5 Value : b

Key : 2 Value : m

Key : 1 Value : d

### [Difference between List, Set and Map in java](http://www.javamadesoeasy.com/2016/02/difference-between-list-set-and-map-in.html)

*In this Collection framework tutorial we will learn Difference between List, Set and Map in java, it forms the base of java collection api. We will find out what are most important differences between java.util.List, java.util.Set and java.util.Map in java.*

*Read :* [**Collection - List, Set and Map all properties in tabular form in java**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

***10 Differences*** *between java.util.*[***List***](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html)*, java.util.*[***Set***](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) *and java,util.*[***Map***](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) *in java >*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Property | ***java.util.List*** | ***java.util.Set*** | ***java.util.Map*** |
| 1 | Duplicate elements | List **allows to store duplicate elements** in java. | *Set does* ***not allow to store duplicate elements*** in java*.* | *Map stores data in form of* ***key-value pair*** *it does not allow to store duplicate keys but allows duplicate values* in java*.* |
| 2 | Insertion order | java.util.List is ordered collection it **maintain insertion order** in java. | *Most of the java.util.Set implementation* does not **maintain insertion order**.  HashSet does not maintains insertion order in java.  Thought LinkedHashSet maintains insertion order in java.    TreeSet is sorted by natural order in java. | *Most of the java.util.Map implementation* does not **maintain insertion order**.  HashMap does not maintains insertion order in java.  Thought LinkedHashMap maintains insertion order of keys in java.    TreeMap is sorted by natural order of keys in java. |
|  |  |  |  |  |
| 3 | Null keys | List allows to store **many null keys** in java. | Most of the Set implementations allow to add only **one null** in java**.**  [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) and [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) does not allow to add null in java. | Lets look at Map implementations -  [HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) allows one null key and many null values.  [LinkedHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) allows one null key and many null values.  [TreeMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) doesn't allow null key but allow many null values.  [Hashtable](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) doesn't allow null key or null values.  [ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) doesn't allow null key or null values.  [ConcurrentSkipListMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) doesn't allow null key or null values. |
| 4 | Getting element on specific **index** | List implementations provide get method to get element on specific index in java.  ArrayList, Vector, copyOnWriteArrayList and LinkedList provides -  *get(int index)*  Method returns element on specified *index*.  **Get method directly gets element on specified index. Hence, offering O(1) complexity.** | Set implementations does not provide any such get method to get element on specified index in java. | Map implementations does not provide any such get method to get element on specified index in java. |
| 5 | Implementing classes | [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. | [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. | [HashMap, Hashtable, ConcurrentHashMap,  LinkedHashMap,  TreeMap,  ConcurrentSkipListMap,  IdentityHashMap,WeakHashMap,  EnumMap classes](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) implements [Map](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) interface in java. |
| 6 | listIterator | **listIterator** method returns listIterator to iterate over elements in List in java.  **listIterator provides** additional methods as compared to iterator like  **hasPrevious(), previous(), nextIndex(), previousIndex(), add(E element), set(E element)** | Set does not provide anything like listIterator. It simply return Iterator in java. | Map provides three type of iterators -  *map.keySet().iterator()* method returns iterator to iterate over keys in HashMap  *map.values().iterator()* method returns iterator to iterate over keys in HashMap in java.  *map.entrySet().iterator()* method returns iterator to iterate over keys in HashMap. |
| 7 | Structure and resizable | **List** are Resizable-array implementation of the java.util.**List** interface in java. | Set uses [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)for their implementation.  Hence, structure is map based and resizing depends on Map implementation.  *Example >* [***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *internally uses* [*HashMap*](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html)*.* | [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) **uses hashing technique for storing** key-value pairs. |
| 8 | Index based structure /RandomAccess | As **ArrayList** uses array for implementation it is index based structure, hence provides random access to elements.  But [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) is not indexed based structure in java. | Set is not index based structure at all in java. | Map is not index based structure at all in java. |
| 9 | unsynchronized implementations | [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) | [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) | [HashMap,  LinkedHashMap,  TreeMap,  IdentityHashMap, WeakHashMap,  EnumMap](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) |
| 10 | synchronized implementations | [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) | [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) | [Hashtable](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html), [ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html), [ConcurrentSkipListMap](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html), |

# HashMap and TreeMap in Java

[**2.2**](https://www.geeksforgeeks.org/easy/)

HashMap and TreeMap are part of [collection framework](https://docs.oracle.com/javase/tutorial/collections/intro/).

**HashMap**

java.util.HashMap class is a Hashing based implementation. In HashMap, we have a key and a value pair<Key, Value>.

HashMap<K, V> hmap = new HashMap<K, V>();

Let us consider below example where we have to count occurrences of each integer in given array of integers.

Input: arr[] = {10, 3, 5, 10, 3, 5, 10};

Output: Frequency of 10 is 3

Frequency of 3 is 2

Frequency of 5 is 2

|  |
| --- |
| /\* Java program to print frequencies of all elements using     HashMap \*/  import java.util.\*;    class Main  {      // This function prints frequencies of all elements      static void printFreq(int arr[])      {          // Creates an empty HashMap          HashMap<Integer, Integer> hmap =                       new HashMap<Integer, Integer>();            // Traverse through the given array          for (int i = 0; i < arr.length; i++)          {              Integer c = hmap.get(arr[i]);                // If this is first occurrence of element              if (hmap.get(arr[i]) == null)                 hmap.put(arr[i], 1);                // If elements already exists in hash map              else                hmap.put(arr[i], ++c);          }            // Print result          for (Map.Entry m:hmap.entrySet())            System.out.println("Frequency of " + m.getKey() +                               " is " + m.getValue());      }        // Driver method to test above method      public static void main (String[] args)      {          int arr[] = {10, 34, 5, 10, 3, 5, 10};          printFreq(arr);      }  } |

Run on IDE

Output:

Frequency of 34 is 1

Frequency of 3 is 1

Frequency of 5 is 2

Frequency of 10 is 3

**Key Points**

* HashMap does not maintain any order neither based on key nor on basis of value, If we want the keys to be maintained in a sorted order, we need to use TreeMap.
* **Complexity**: get/put/containsKey() operations are O(1) in average case but we can’t guarantee that since it all depends on how much time does it take to compute the hash.

**Application:**  
HashMap is basically an implementation of [hashing](http://geeksquiz.com/hashing-set-1-introduction/). So wherever we need hashing with key value pairs, we can use HashMap. For example, in Web Applications username is stored as a key and user data is stored as a value in the HashMap, for faster retrieval of user data corresponding to a username.

**TreeMap**

TreeMap can be a bit handy when we only need to store unique elements in a sorted order. Java.util.TreeMap uses a [red-black tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/) in the background which makes sure that there are no duplicates; additionally it also maintains the elements in a sorted order.

TreeMap<K, V> hmap = new TreeMap<K, V>();

Below is TreeMap based implementation of same problem. This solution has more time complexity O(nLogn) compared to previous one which has O(n). The advantage of this method is, we get elements in sorted order.

## [Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/java-collection-set-4-treemap/1)

|  |
| --- |
| /\* Java program to print frequencies of all elements using     TreeMap \*/  import java.util.\*;    class Main  {      // This function prints frequencies of all elements      static void printFreq(int arr[])      {          // Creates an empty TreeMap          TreeMap<Integer, Integer> tmap =                       new TreeMap<Integer, Integer>();            // Traverse through the given array          for (int i = 0; i < arr.length; i++)          {              Integer c = tmap.get(arr[i]);                // If this is first occurrence of element              if (tmap.get(arr[i]) == null)                 tmap.put(arr[i], 1);                // If elements already exists in hash map              else                tmap.put(arr[i], ++c);          }            // Print result          for (Map.Entry m:tmap.entrySet())            System.out.println("Frequency of " + m.getKey() +                               " is " + m.getValue());      }        // Driver method to test above method      public static void main (String[] args)      {          int arr[] = {10, 34, 5, 10, 3, 5, 10};          printFreq(arr);      }  } |

Run on IDE

Output:

Frequency of 3 is 1

Frequency of 5 is 2

Frequency of 10 is 3

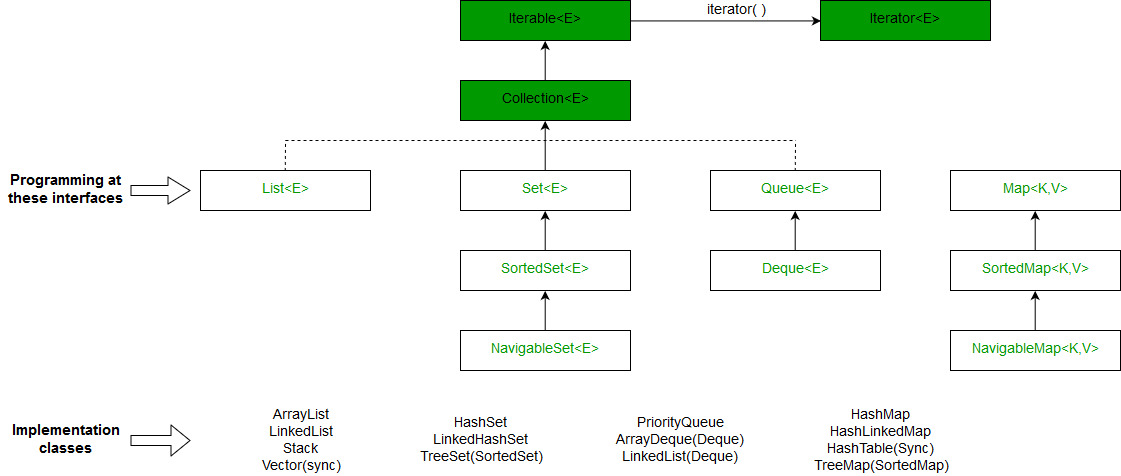
Frequency of 34 is 1

**Key Points**

* For operations like add, remove, containsKey, time complexity is O(log n where n is number of elements present in TreeMap.
* TreeMap always keeps the elements in a sorted(increasing) order, while the elements in a HashMap have no order. TreeMap also provides some cool methods for first, last, floor and ceiling of keys.

**Overview:**

1. HashMap implements Map interface while TreeMap implements SortedMap interface. A Sorted Map interface is a child of Map.
2. HashMap implements Hashing, while TreeMap implements Red-Black Tree(a Self Balancing Binary Search Tree). Therefore all[differences between Hashing and Balanced Binary Search Tree](https://www.geeksforgeeks.org/advantages-of-bst-over-hash-table/) apply here.
3. Both HashMap and TreeMap have their counterparts HashSet and TreeSet. HashSet and TreeSet implement [Set interface](http://geeksquiz.com/set-in-java/). In HashSet and TreeSet, we have only key, no value, these are mainly used to see presence/absence in a set. For above problem, we can’t use HashSet (or TreeSet) as we can’t store counts. An example problem where we would prefer HashSet (or TreeSet) over HashMap (or TreeMap) is to print all distinct elements in an array.



### [Differences and Similarities between ArrayList and LinkedList in java](http://www.javamadesoeasy.com/2015/04/arraylist-vs-linkedlist-similarity-and.html)

# How ArrayList Works Internally in Java

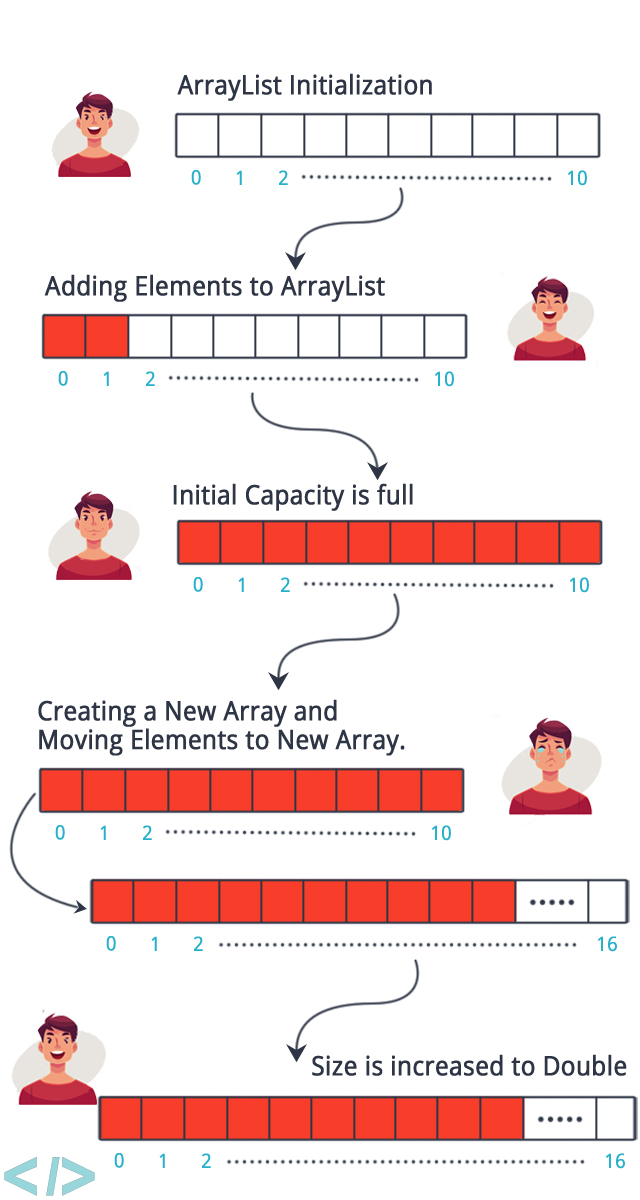
[December 1, 2017](http://www.codenuclear.com/how-arraylist-works-internally-java/) [codeNuclear](http://www.codenuclear.com/author/codenuclear/)

#### **Overview**

ArrayList is a resizable array implementation of the List interface i.e. ArrayList grows dynamically as the elements are added to it.

If the size of the current elements (including the new element to be added to the ArrayList) is greater than the maximum size of the array then increase the size of array. But the size of the array can not be increased dynamically. So, what happens internally is, a new Array is created and the old array is copied into the new array.

Let’s see how ArrayList is internally implemented; what is the backing data structure for an ArrayList, how it grows dynamically.

How ArrayList grows dynamically?

#### **ArrayList implementation in Java**

Internally an ArrayList uses an Object[] Array. All the addition, removal and traversal happens on this array.

In Java 7 or previous

Java



|  |  |
| --- | --- |
| 0  1  2  3  4  5  6 | /\*\*       \* The array buffer into which the elements of the ArrayList are stored.       \* The capacity of the ArrayList is the length of this array buffer.       \*/      private transient Object[] elementData; |

In Java 8 or later

*It’s very important to differentiate between java.util.****ArrayList*** *and java.util.****LinkedList****, so in this Collection framework tutorial we will learn what are differences and similarities between java.util.ArrayList and java.util.LinkedList in java.*

# How to remove an element from ArrayList in Java?

[**2.1**](https://www.geeksforgeeks.org/easy/)

There are two way to remove an element from ArrayList.

**1. By using remove() methods :**

ArrayList provides two overloaded remove() method.  
a. **remove(int index)** : Accept index of object to be removed.  
b. **remove(Obejct obj)** : Accept object to be removed.

What happens when we have an **integer** arrayList and we want to remove an item? For example consider below program.

|  |
| --- |
| // Java program to demonstrate working of remove  // on an integer arraylist  import java.util.List;  import java.util.ArrayList;    public class GFG  {      public static void main(String[] args)      {          List al = new ArrayList();          al.add(10);          al.add(20);          al.add(30);          al.add(1);          al.add(2);            // This makes a call to remove(int) and          // removes element 20.          al.remove(1);            // Now element 30 is moved one position back          // So element 30 is removed this time          al.remove(1);            System.out.println("Modified ArrayList : " + al);      }  } |

Run on IDE

Output :

Modified ArrayList : [10, 1, 2]

We can see that the passed parameter is considered as index. How to remove elements by value.

|  |
| --- |
| // Java program to demonstrate working of remove  // on an integer arraylist  import java.util.List;  import java.util.ArrayList;    public class GFG  {      public static void main(String[] args)      {          List al = new ArrayList();          al.add(10);          al.add(20);          al.add(30);          al.add(1);          al.add(2);            // This makes a call to remove(Object) and          // removes element 1          al.remove(new Integer(1));            // This makes a call to remove(Object) and          // removes element 2          al.remove(new Integer(2));            System.out.println("Modified ArrayList : " + al);      }  } |

Run on IDE

Output :

Modified ArrayList : [10, 20, 30]

**2. Using Iterator.remove() method :**

It is not recommended to use ArrayList.remove() when iterating over elements. This may lead to [ConcurrentModificationException](https://docs.oracle.com/javase/7/docs/api/java/util/ConcurrentModificationException.html) (Refer [this](https://ide.geeksforgeeks.org/uP1Tc5) for a sample program with this exception). When iterating over elements, it is recommended to use [Iterator.remove()](https://www.geeksforgeeks.org/iterators-in-java/) method .

|  |
| --- |
| // Java program to demonstrate working of  // Iterator.remove() on an integer arraylist  import java.util.List;  import java.util.ArrayList;  import java.util.Iterator;    public class GFG  {      public static void main(String[] args)      {          List al = new ArrayList();          al.add(10);          al.add(20);          al.add(30);          al.add(1);          al.add(2);            // Remove elements smaller than 10 using          // Iterator.remove()          Iterator itr = al.iterator();          while (itr.hasNext())          {              int x = (Integer)itr.next();              if (x < 10)                  itr.remove();          }            System.out.println("Modified ArrayList : "                                             + al);      }  } |

Run on IDE

Output :

Modified ArrayList : [10, 20, 30]

*Also Read :* [**Collection - List, Set and Map all properties in tabular form in java**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

[**java.util.List hierarchy in java**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html)

[**java.util.Set hierarchy in java**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)

[**java.util.Map hierarchy in java**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)

*Differences between java.util.*[*ArrayList*](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) *and java.util.*[*LinkedList*](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | **java.util.LinkedList** |
| 1 | Structure | java.util.ArrayList is index based structure in java.  [https://lh6.googleusercontent.com/f8eJoX-FNynX9xx_rTl8IqaAGjMMtDYnZVgg6Hkco7C7uYfoHpCk-5-EjhsyIPm5OxjMI7d0JkALWUYnXP7tgqWTUrYCoFQBBbjQrg6fRHdCNyeglftSiWg-tB0R1-SOWfE1tw0](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html) | A java.util.**LinkedList** is a data structure consisting of a group of **nodes** which together represent a sequence.  node is composed of a data and a reference (in other words, a **link**) to the next node in the sequence in java.  [https://lh3.googleusercontent.com/iS8cWH4MmnPfJHvBJRj9EonXJ2d2NQXuZf6_5vA8n9DzFWouOJI0O7xB0Ed115OpPhGvSXIx5X7BU28PXg4YFrhkp6RHX0RtwwlcboamgUM3_YaBSQlN2VmjYzt-KfY8lEvTnNk](http://www.javamadesoeasy.com/2015/01/doublylinkedlist-insert-and-delete-at.html) |
| 2 | **Resizable** | **ArrayList is Resizable-array in java.** | New node is created for storing new element in LinkedList in java. |
| 3 | **Initial capacity** | java.util.ArrayList is created with initial capacity of 10 in java. | For storing every element node is created in LinkedList, so linkedList’s initial capacity is 0 in java. |
| 4 | Ensuring **Capacity**/ resizing. | ArrayList is created with initial capacity of 10.  ArrayList’s size is **increased by 50%** i.e. after resizing it’s size become 15 in java. | For storing every element node is created, so linkedList’s initial capacity is 0, it’s size grow with addition of each and every element in java. |
| 5 | RandomAccess interface | ArrayList implements RandomAccess(Marker interface) to indicate that they support fast random access (i.e. index based access) in java. | LinkedList does not implement RandomAccess interface in java. |
| 6 | AbstractList and AbstractSequentialList | ArrayList extends AbstractList (abstract class) which provides implementation to  List interface to minimize the effort required to implement this interface backed by RandomAccess interface. | LinkedList extends AbstractSequentialList (abstract class), AbstractSequentialList extends AbstractList.  In LinkedList, data is accessed sequentially, so for obtaining data at specific index, iteration is done on nodes sequentially in java. |
| 7 | How **get(index)** method works?  (Though difference has been discussed briefly in above 2 points but in this in point we will figure difference in detail.) | Get method of ArrayList directly gets element on specified index. Hence, offering O(1) complexity in java. | Get method of LinkedList iterates on nodes sequentially to get element on specified index. Hence, offering O(n) complexity in java. |
| **8** | **When to use** | **Use ArrayList when get operations is more frequent than add and remove operations in java.** | **Use LinkedList when add and remove operations are more frequent than get operations in java.** |
| **9** | **Complexity** offered by methods are different | |  |  |  | | --- | --- | --- | | Operation/ method | Worst case | Best case | | *add* | O(n), when array is full it needs restructuring,  operation runs in *amortized constant time in java.* | O(1), when array does not need any restructuring in java. | | *remove* | O(n), when removal is done from between restructuring is needed. | O(1), when removal is done at last position, no restructuring is needed. | | *get* | O(1), it is index based structure. So, complexity of  get operation is always done in O(1) in java. | O(1) it is index based structure. So, complexity of  get operation is always done in O(1) in java. | | *set* | O(1), it is index based structure, no restructuring is needed in set operation. So, complexity of operation is always O(1) | O(1), it is index based structure, no restructuring is needed in set operation. So, complexity of operation is always O(1) | | *iterator* | O(n), because iteration is done over each and every element in java. | O(n), because iteration is done over each and every element in java. | | *listIterator* | O(n), its same as iterator in java. | O(n), its same as iterator in java. | | *enumeration* | O(n), its same as iterator in java. | O(n), its same as iterator in java. | | |  |  |  | | --- | --- | --- | | Operation/ method | Worst case | Best case | | *add(E element)* | **O(1),** Adds specified *element* to the end of LinkedList in java. | **O(1),** Adds specified *element* to the end of LinkedList in java. | | *add(int index, E element)* | **O(n)**, because iteration is done on all elements one by one to find out specified index.  Current *element* is placed at specified *index* and one is added to indices of subsequent elements on right in java. | **O(n)** | | *addFirst(E element)* | **O(1)** | **O(1)** | | *addLast(E element)* | **O(1)** | **O(1)** | |  |  |  | | *remove()* | **O(1),** Method retrieves and removes the first element (head) of this list in java. | **O(1)** | | *remove(int index)* | **O(n)**, because iteration is done on all elements one by one to find out specified index.  one is subtracted from indices of subsequent elements on right. | **O(n)** | | *remove(Object object)* | **O(n)**, because iteration is done on all elements one by one to find out specified object.  one is subtracted from indices of subsequent elements on right. | **O(n)** | | *removeFirst()* | O(1) | O(1) | | *removeLast()* | O(1) | O(1) | | iterator | O(n), because iteration is done over each and every element. | O(n), because iteration is done over each and every element. | | listIterator | O(n), its same as iterator in java. | O(n), its same as iterator in java. | | enumeration | O(n), its same as iterator in java. | O(n), its same as iterator in java. | |

*So far we have learned what are differences between ArrayList and LinkedList in java. Now we will learn similarities in ArrayList and LinkedList in Collection framework in java.*

*Read :* [**Collection - List, Set and Map all properties in tabular form in java**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

***Similarity*** *in java.util.****ArrayList*** *and java.util.****LinkedList*** *in java >*

|  |  |  |
| --- | --- | --- |
|  | Property | *java.util.****ArrayList*** *and java.util.***LinkedList** |
| 1 | synchronization | ***ArrayList*** *and* **LinkedList both** are **not synchronized**  (because 2 threads on same ArrayList/LinkedList object can access it at same time) in java.  I have created **program** to show see consequence of using ArrayList in multithreading environment.  In the program i will implement our own arrayList in java. |
| 2 | Iterator and listIterator are Fail-fast | Iterator and listIterator returned by ***ArrayList*** *and* **LinkedList both** are [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) in java. |
| 3 | Enumeration is fail-fast | **Enumeration** of ***ArrayList*** *and* **LinkedList both** is **fail-fast**, means any modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException in java.  Example/Code to throw ConcurrentModificationException in ArrayList (we may replace arrayList with linkedList )-   |  | | --- | | Enumeration<String> listEnum= Collections.*enumeration*(arrayList);  **while**(listEnum.hasMoreElements()){  //adding element will throw   ConcurrentModificationException        System.*out*.println(listEnum.nextElement());  } | |
| 4 | Insertion order | ***ArrayList*** *and* **LinkedList both maintains insertion order** in java. |
| 5 | Allows null | ***ArrayList*** *and* **LinkedList** both **allows to store null** in java. |
| 6 | Implements java.util.List | ***ArrayList*** *and* **LinkedList** both are implementation of the java.util.**List** interface. |
| 7 | Introduced in which java version | ***ArrayList*** *and* **LinkedList** both were introduced in second version of java (1.2) i.e. **JDK 2.0** |

### [Differences and Similarities between ArrayList and vector in java](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)

*You are here :* [***Home***](http://www.javamadesoeasy.com/) ***/*** [***Core Java Tutorials***](http://www.javamadesoeasy.com/2015/04/core-java-tutorial.html) ***/*** [*Collection framework Tutorial in java*](http://www.javamadesoeasy.com/2015/05/collection-framework-tutorial-in-java.html)

*It’s very important to differentiate between ArrayList and Vector, so in this Collection framework tutorial we will learn what are differences and similarities between java.util.ArrayList and java.util.Vector in java.*

*Also Read :*[**java.util.List hierarchy in java**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html)

[**java.util.Set hierarchy in java**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)

[**java.util.Map hierarchy in java**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)

***Differences*** *between java.util.*[***ArrayList***](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) *and java.util.****Vector*** *in java>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | ***java.util.Vector*** |
| 1 | synchronization | java.util.ArrayList is **not synchronized**  (because 2 threads on same ArrayList object can access it at same time).  I have created [**program**](http://www.javamadesoeasy.com/2015/05/consequence-of-using-arraylist-in.html)to show consequence of using ArrayList in multithreading environment.  In the program we will implement our own arrayList in java. | java.util.Vector is **synchronized** (because 2 threads on same Vector object cannot  access it at same time).  I have created [**program**](http://www.javamadesoeasy.com/2015/05/advantage-of-using-vector-in.html)to show advantage of using Vector in multithreading environment.  In the program we will implement our own vector in java. |
| 2 | Performance | ArrayList is not synchronized, hence its operations are **faster** as compared to Vector in java. | Vector is synchronized, hence its operations are **slower** as compared to ArrayList in java.  If we are working not working in multithreading environment jdk recommends us to use ArrayList. |
| 3 | Enumeration | **Enumeration** is [**fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), means any modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException in java. | **Enumeration** is **fail-safe**, means any modification made to Vector during iteration using Enumeration don’t throw any exception in java. |
| 4 | Introduced  in which java version | ArrayList was introduced in second version of java i.e. **JDK 2.0** | Vector was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the List interface, hence making it a member of member of the [Java Collections Framework](http://www.javamadesoeasy.com/2015/04/collection-in-java.html). |
| 5 | Ensuring Capacity/ resizing. | ArrayList is created with initial capacity of 10.  When its full size is **increased by 50%** i.e. after resizing it’s size become 15 in java. | Vector is created with initial capacity of 10.  Vector’s size is **increased by 100%** i.e. after resizing it’s size become 20 in java. |
| 6 | Custom implementation | [https://lh4.googleusercontent.com/FxlXU1zjlhbC03GLS5VMI9ubd2k_1ToIAsVmXXFYwEzF93-gTYCcJUkmkikYTeLiApsdx62z2NZXn4CAeTBdaAGSWDZZe_TPO9Pyz3GtW-PdbqkTQv8ZwoOR3rFSEb_CHbBZVng](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)Read : [ArrayList custom implementation](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html) | [https://lh3.googleusercontent.com/-RcOB6IdgA9EdNDV1BU77aAYgZ29vPcz59SYkwvjtapRDlEvmOVny8YU-sjAtqFr1HBoZNk8qJShPDxlzmKb-MrhLKQQuyfZOMqp_OxeWni3x8qzY1M4V9XxOnMpd195WoCLqCY](http://javamadesoeasy.com/2015/02/vector-custom-implementation.html)Read :  [Vector custom implementation](http://javamadesoeasy.com/2015/02/vector-custom-implementation.html) |

*So far we have learned what are differences between ArrayList and Vector in java. Now we will learn similarities in ArrayList and Vector in Collection framework in java.*

*Read :* [**Collection - List, Set and Map all properties in tabular form in java**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

***Similarity*** *in java.util.****ArrayList*** *and java.util.****Vector*** *in java >*

|  |  |  |
| --- | --- | --- |
|  | Property | *java.util.****ArrayList and*** *java.util.****Vector*** |
| 1 | Iterator and listIterator are Fail-fast | Iterator and listIterator returned by *ArrayList and Vector* both are **Fail-fast** in java. |
| 2 | Insertion order | *ArrayList and Vector* both **maintains insertion order** in java. |
| 3 | Duplicate | *ArrayList and Vector* both **allow to store duplicate elements** in java**.** |
| 4 | Allows null | *ArrayList and Vector* both **allows to store null** in java. |
| 5 | Implements java.util.List | *ArrayList and Vector* both are implementation of the java.util.**List** interface. |
| 6 | Index based structure | *ArrayList and Vector* both are **index based structures** in java. |
| 7 | Complexity | Complexity offered by **methods of** *ArrayList and Vector* **both of these is same** in java.   |  |  |  | | --- | --- | --- | | Operation/ method | ***Worst case*** | ***Best case*** | | add | O(n), when array is full it needs restructuring,  operation runs in *amortized constant time.* | O(1), when array does not need any restructuring. | | remove | O(n), when removal is done from between restructuring is needed. | O(1), when removal is done at last position, no restructuring is needed. | | get | O(1), it is index based structure. So, complexity of  get operation is always done in O(1). | O(1) it is index based structure. So, complexity of  get operation is always done in O(1). | | set | O(1), it is index based structure, no restructuring is needed in set operation. So, complexity of operation is always O(1) | O(1), it is index based structure, no restructuring is needed in set operation. So, complexity of operation is always O(1) | | iterator | O(n), because ireation is done over each and every element. | O(n), because ireation is done over each and every element. | | listIterator | O(n), its same as iterator. | O(n), its same as iterator. | | enumeration | O(n), its same as iterator. | O(n), its same as iterator. | |

### [Differences and Similarities between Iterator and ListIterator in java](http://www.javamadesoeasy.com/2015/04/iterator-vs-listiterator-similarity-and.html)

*In this Collection framework tutorial we will learn what are differences and similarities between java.util.****Iterator*** *and java.util.****ListIterator*** *in java.*

***Differences*** *between java.util.****Iterator*** *and java.util.****ListIterator*** *>*

|  |  |  |
| --- | --- | --- |
|  | ***java.util.ListIterator*** | ***java.util.Iterator*** |
| 1 | **hasPrevious()**  method returns true if this listIterator has more elements when traversing the list in the reverse direction. | **No such method** in java.util.Iterator. |
| 2 | **previous()**  returns previous element in iteration (traversing in backward direction).  if the iteration has no previous elements than NoSuchElementException is thrown. | **No such method** in java.util.Iterator. |
| 3 | **nextIndex()**  method returns the index of the element that would be returned by a subsequent call to next() method. If listIterator is at the end of the list than method returns size of list. | **No such method** in java.util.Iterator. |
| 4 | **previousIndex()**  method returns the index of the element that would be returned by a subsequent call to previous() method. If listIterator is at the start of the list than method returns -1. | **No such method** in java.util.Iterator. |
| 5 | **add(E element)**  Method inserts the specified **element** into the list.  The element is inserted immediately before the element that would be returned by next (So, subsequent call to next would be unaffected), if any, and after the element that would be returned by previous (So,subsequent call to previous would return the new **element**), if any.  If the list does not contain any element than new **element** will be the sole element in the list. | **No such method** in java.util.Iterator. |
| 6 | **set(E element)**  Method replaces the last element returned by next() or previous() method with the specified **element**. This call can be made only if neither remove nor add have been called after the last call to next or previous.  If call to set() method is followed up by any call made to remove() or add() method after next() or previous() than UnsupportedOperationException is thrown. | **No such method** in java.util.Iterator. |
| 7 | All the implementations of [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) interface like [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html), [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes returns listIterator. | All Implementation classes of [**Collection**](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) interface’s subinterfaces like [Set and List](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) return iterator. |

*Also Read :* [**Collection - List, Set and Map all properties in tabular form in java**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

*So far we have learned what are differences between java.util.****Iterator*** *and java.util.****ListIterator*** *in java.*

*Now we will learn similarities between java.util.****Iterator*** *and java.util.****ListIterator*** *in Collection framework in java.*

***Similarity*** *between java.util.****Iterator*** *and java.util.****ListIterator*** *>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | **java.util.Iterator** | **java.util.ListIterator** |
| 1 | Definition | *Iterator* method returns iterator to iterate over elements in ArrayList. | *ListIterator* method returns listIterator to iterate over elements in ArrayList. |
| 2 | Common important methods | Iterator important methods :  **hasNext()** method returns true if the iteration has more elements. (Traversing/Iteration is  done in forward direction).  **next()** method - returns next element in iteration.  if the iteration has no more elements than NoSuchElementException is thrown.  **remove()** method - removes last element returned by iterator.  Method must always be called after call to next() method else IllegalStateException is thrown. | ListIterator provide all 3 methods.  Apart from hasNext(), next() and remove() methods provided by Iterator, **ListIterator additionally provides(We will discuss that in differences).** |
| 3 | **Complexity** | Complexity of Iterator is **O(n)**, because iteration/traversal is done over each and every element. | Complexity of ListIterator is also **O(n)**, because iteration/traversal is done over each and every element. |

### [Differences between Collection and Collections in java](http://www.javamadesoeasy.com/2015/04/collection-vs-collections-differences.html)

*In this Collection framework tutorial we will learn what are differences between java.util.****Collection*** *and java.util.****Collections*** *in java.*

java.util.[***Collection***](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) ***​*** *is the* root **interface** in the ​*hierarchy of Java Collection framework​*.

The JDK does not provide any classes which directly implements java.util.Collection interface, but it  provides classes such as [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html), [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html), [**vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html), [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html), [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [CopyOnWriteArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html), [CopyOnWriteArraySet](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html), [ConcurrentSkipListSet](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)  which implements more specific subinterfaces like ​[Set and List​](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) in java.

java.util.**Collections** is a utility **class** which **consists** of **static methods** that **operate on** or return **Collection** in java.

**java.util.Collections provides method like >**

* **reverse** method for reversing [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) in java.
* **shuffle** method for shuffling elements of **List** in java.
* **unmodifiableCollection**, [**unmodifiableSet**](http://www.javamadesoeasy.com/2015/04/hashset-making-set-unmodifiable-using.html), [**unmodifiableList**](http://www.javamadesoeasy.com/2015/04/arraylist-making-list-unmodifiable.html), [**unmodifiableMap**](http://www.javamadesoeasy.com/2015/04/hashmap-making-map-unmodifiable-using.html) methods for making **List**, [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) and [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) unmodifiable in java.
* **min** method to return smallest element in **Collection** in java.
* **max** method to return smallest element in **Collection**.
* **sort** method for sorting **List**.
* **synchronizedCollection**, [**synchronizedSet**](http://www.javamadesoeasy.com/2015/04/hashset-synchronizing-using.html), [**synchronizedList**](http://www.javamadesoeasy.com/2015/04/arraylist-synchronizing-using.html), [**synchronizedMap**](http://www.javamadesoeasy.com/2015/04/hashmap-synchronizing-map-using.html)methods for synchronizing **List**, **Set** and **Map** respectively in java**.**

Additionally you must know that *java.util.Collection and java.util.Collections both were introduced in* ***second version of java i.e. in JDK 2.0.***

*So in this Collection framework tutorial we learned what are differences between java.util.****Collection*** *and java.util.****Collections*** *in java.*

### [How to synchronize arraylist in java to make it completely thread safe in java](http://www.javamadesoeasy.com/2015/12/how-to-synchronize-arraylist-in-java-to.html)

In this Collection framework tutorial we will learn How to synchronize arraylist in java to make it completely thread safe with examples and programs in java.

Contents of page >

* *1) First let’s understand problem statement >*
* *2) So, How to synchronize arraylist in java to make it completely thread safe in java >*
* *3) Example/ Program 1* ***(elaborate problem statement)*** *to show that iterator returned by synchronizedArrayList won't be synchronized, it can* ***throw*** [***ConcurrentModificationException***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) *in java>*
* *4) Example/ Program 2* ***(elaborate Solution 1)*** *to show that iterator returned by synchronizedArrayList won't be synchronized, so during iteration we must keep synchronizedArrayList in* [*synchronization block*](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) *to* ***avoid*** [***ConcurrentModificationException***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) *in java >*
* *5) Example/ Program 3* ***(elaborate Solution 2)*** *to show that iterator returned by* [***CopyOnWriteArrayList***](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) *will be thread-safe, so during iteration we don’t need any synchronization block and*[***ConcurrentModificationException***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) *will* ***not*** *be thrown in java >*

1) First let’s understand problem statement >

Is synchronizing [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) using [Collections](http://www.javamadesoeasy.com/2015/04/collection-vs-collections-differences.html).synchronizedList is completely [thread-safe](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) in java?

Answer is **No.**

Because when we synchronize [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) using [Collections](http://www.javamadesoeasy.com/2015/04/collection-vs-collections-differences.html).synchronizedList in java, Iterator on synchronizedArrayList won't be synchronized in java.

2) So, How to synchronize arraylist in java to make it completely thread safe in java >

There are 2 solutions

**Solution 1 =** iterator returned by synchronizedArrayList won't be synchronized, so during iteration we must keep synchronizedArrayList in [synchronization block](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) to avoid [**ConcurrentModificationException**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) in java.

**Solution 2 =** Use [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html), it is completely thread-safe, also Iterator & listIterator returned by [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) are completely thread-safe in java.

3) Example/ Program 1 **(elaborate problem statement)** to show that iterator returned by synchronizedArrayList won't be synchronized, it can **throw** [**ConcurrentModificationException**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)  in java>

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Iterator;  **import** java.util.List;  **public** **class** ArrayListSynchronizationFailsExample {  **public** **static** **void** main(String[] args) {  **final** List<Integer> arrayList = **new** ArrayList<Integer>();  **final** List<Integer> synchronizedArrayList;            // Let's make arrayList synchronized            synchronizedArrayList = Collections.*synchronizedList*(arrayList);              //Thread 1 will add elements in synchronizedArrayList            Thread t1 = **new** Thread(**new** Runnable() {  **public** **void** run() {  **for** (**int** i = 0; i <= 3; i++) {                                synchronizedArrayList.add(i);  **try** {                                       Thread.*sleep*(100);                                } **catch** (InterruptedException e) {                                       e.printStackTrace();                                }                         }                   }            }, "thread-1");              t1.start();            //Thread 2 will iterate on synchronizedArrayList            Thread t2 = **new** Thread(**new** Runnable() {  **public** **void** run() {                         Iterator<Integer> it = synchronizedArrayList.iterator();  **while** (it.hasNext()) {  **try** {                                       Thread.*sleep*(100);                                } **catch** (InterruptedException e) {                                       e.printStackTrace();                                }                                System.***out***.println(it.next());//throws ConcurrentModificationException                         }                   }            }, "thread-2");            t2.start();     }  }  /\*OUTPUT  Exception in thread "thread-2" [java.util.ConcurrentModificationException](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)     at java.util.ArrayList$Itr.checkForComodification(Unknown Source)     at java.util.ArrayList$Itr.next(Unknown Source)     at ArrayListSynchronizationFails$2.run(ArrayListSynchronizationFails.java:43)     at java.lang.Thread.run(Unknown Source)  \*/ |

Let’s discuss what happened in above program >

main thread started thread-1, thread-1 added 0 (i.e. i=0) in **synchronizedArrayList**, then thread-1 went for sleep(100), meanwhile

main thread started thread-2, thread-2 obtained iterator on

**synchronizedArrayList**, then thread-2 entered while loop, then thread-2 went for sleep(100),

sleep time was up and **thread-1 again entered running state and added 1** (i.e. i=1) **in synchronizedArrayList**, then thread-1 went for sleep(100),

sleep time was up and thread-2 again entered running state and **it.next()** throws [ConcurrentModificationException](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) , because arraylist was modified by thread-1 after obtaining iterator on synchronizedArrayList.

4) Example/ Program 2 **(elaborate Solution 1)** to show that iterator returned by synchronizedArrayList won't be synchronized, so during iteration we must keep synchronizedArrayList in [synchronization block](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) to **avoid** [**ConcurrentModificationException**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) in java >

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Iterator;  **import** java.util.List;  **public** **class** ArrayListSynchronizationSucceedsExample {  **public** **static** **void** main(String[] args) {  **final** List<Integer> arrayList = **new** ArrayList<Integer>();  **final** List<Integer> synchronizedArrayList;            // Let's make arrayList synchronized            synchronizedArrayList = Collections.*synchronizedList*(arrayList);            //Thread 1 will add elements in synchronizedArrayList            Thread t1 = **new** Thread(**new** Runnable() {  **public** **void** run() {  **for** (**int** i = 0; i <= 3; i++) {                                synchronizedArrayList.add(i);  **try** {                                       Thread.*sleep*(100);                                } **catch** (InterruptedException e) {                                       e.printStackTrace();                                }                         }                   }            }, "thread-1");            t1.start();            //Thread 2 will iterate on synchronizedArrayList            Thread t2 = **new** Thread(**new** Runnable() {  **public** **void** run() {                         Iterator<Integer> it = synchronizedArrayList.iterator();  **synchronized (synchronizedArrayList) {** //synchronization block  **while** (it.hasNext()) {  **try** {                                              Thread.*sleep*(100);                                       } **catch** (InterruptedException e) {                                              e.printStackTrace();                                       }                                       System.***out***.println(it.next());                                }  **}**                     }            }, "thread-2");            t2.start();     }  }  /\*output  0  \*/ |

Let’s discuss what happened in above program >

main thread started thread-1, thread-1 added 0 (i.e. i=0) in **synchronizedArrayList**, then thread-1 went for sleep(100), meanwhile

main thread started thread-2, it obtained lock on **synchronizedArrayList object**, then thread-2 obtained iterator on **synchronizedArrayList**, then thread-2 entered while loop, then thread-2 went for sleep(100),

sleep time was up and **thread-1 entered running state but add(i) method on synchronizedArrayList was synchronized(you must know that only iterator returned on synchronizedArrayList is not thread safe, rest of the methods are completely thread-safe), so thread-1 waited for thread-2 to release lock on synchronizedArrayList object,**

**as soon as thread-2 released lock on synchronizedArrayList object thread-1 added** 1 (i.e. i=1) **in synchronizedArrayList, it further completed for loop and we didn’t encountered**  [ConcurrentModificationException](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html).

5) Example/ Program 3 **(elaborate Solution 2)** to show that iterator returned by [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) will be thread-safe, so during iteration we don’t need any synchronization block and  [**ConcurrentModificationException**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) will **not** be thrown in java >

**CopyOnWriteArrayList** **is** completely thread-safe in java. We does not need to synchronize it explicitly.

Iterator & listIterator returned by **CopyOnWriteArrayList** are also completely thread-safe in java.

|  |
| --- |
| **import** java.util.Iterator;  **import** java.util.List;  **import** java.util.concurrent.CopyOnWriteArrayList;  **public** **class** CopyOnWriteArrayListSynchronizationSucceedsExample {  **public** **static** **void** main(String[] args) {  **final** List<Integer> copyOnWriteArrayList = **new** CopyOnWriteArrayList<Integer>();            //Thread 1 will add elements in CopyOnWriteArrayList            Thread t1 = **new** Thread(**new** Runnable() {  **public** **void** run() {  **for** (**int** i = 0; i <= 3; i++) {                                copyOnWriteArrayList.add(i);  **try** {                                       Thread.*sleep*(100);                                } **catch** (InterruptedException e) {                                       e.printStackTrace();                                }                         }                   }            }, "thread-1");              t1.start();            //Thread 2 will iterate on CopyOnWriteArrayList            Thread t2 = **new** Thread(**new** Runnable() {  **public** **void** run() {                         Iterator<Integer> it = copyOnWriteArrayList.iterator();  **while** (it.hasNext()) {  **try** {                                       Thread.*sleep*(100);                                } **catch** (InterruptedException e) {                                       e.printStackTrace();                                }                                System.***out***.println(it.next());                         }                   }            }, "thread-2");            t2.start();     }  }  /\*OUTPUT  0  \*/ |

### [COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-3 > Q75- Q100](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview45.html)

### *You are here :* [*Home*](http://www.javamadesoeasy.com/) */* [*Core Java Tutorials*](http://www.javamadesoeasy.com/2015/04/core-java-tutorial.html) */* [*Collection framework Tutorial in java*](http://www.javamadesoeasy.com/2015/05/collection-framework-tutorial-in-java.html) */* [*Java Interview Questions and answers*](http://www.javamadesoeasy.com/p/interview-questions.html)

### [COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

### [COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

### These very important questions have been framed to test your basic and in depth knowledge of Collection, these questions covers vast variety of questions that touches almost all concepts of Collection in java. Questions ranges from easy to hard for fresher/beginner to experienced developers . Go, have a crack on these!!

### Collection interview Question 76.

### Collection Output interview question 26.

|  |
| --- |
| import **java.util.HashMap;**  **import** java.util.Map;  /\*\* \*/  **public** **class** MyClass {  **public** **static** **void** main(String args[]) {            Map<String, String> hashMap = **new** HashMap<String, String>();            hashMap.put(**new** String("a"), "audi");            hashMap.put(**new** String("a"), "ferrari");            System.*out*.println(hashMap);     }  }  /\*OUTPUT  {a=ferrari}  \*/ |

### Answer. HashMap does not allow duplicate keys. HashMap when comparing keys (and values) performs object-equality not reference-equality. In an HashMap, two keys k1 and k2 are equal if and only if (k1==null ? k2==null : k1.equals(k2))

### new String("a") & new String("a") are different by reference but equal by value.

### 

### Collection interview Question 77.

### Collection Output interview question 27.

|  |
| --- |
| import **java.util.IdentityHashMap;**  **import** java.util.Map;  /\*\* \*/  **public** **class** MyClass {  **public** **static** **void** main(String args[]) {            Map<String, String> identityHashMap = **new** IdentityHashMap<String, String>();            identityHashMap.put(**new** String("a"), "audi");            identityHashMap.put(**new** String("a"), "ferrari");            System.*out*.println(identityHashMap);     }  }  /\*OUTPUT  {a=audi, a=ferrari}  \*/ |

### Answer.

### [IdentityHashMap](http://www.javamadesoeasy.com/2015/04/identityhashmap-in-java.html) when comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2). (In normal Map implementations (like HashMap) two keys k1 and k2 are considered equal if and only if (k1==null ? k2==null : k1.equals(k2)).)

### new String("a") & new String("a") are different by reference.

### Must read : [Differences and Similarities between HashMap and IdentityHashMap with program in java](http://www.javamadesoeasy.com/2015/04/hashmap-vs-identityhashmap-similarity.html)

### 

### Collection interview Question 78.

### Collection Output interview question 28.

|  |
| --- |
| import **java.util.Map;**  **import** java.util.TreeMap;  /\*\* \*/  **public** **class** TreeMapTest {  **public** **static** **void** main(String args[]) {            Map<Integer, String> m = **new** TreeMap<Integer, String>();            m.put(11, "audi");            m.put(**null**, **null**);            m.put(11, "bmw");            m.put(**null**, "fer");            System.*out*.println(m.size());            System.*out*.println(m);     }  } |

### Answer.  NullPointerException

### [TreeMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) does not any null key or null value.

### 

### Collection interview Question 79.

### Collection Output interview question 29.

|  |
| --- |
| import **java.util.Arrays;**  **import** java.util.Comparator;  /\*\* \*/  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            String[] ar = { "c", "d", "b", "a", "e" };            NestedClass in = **new** NestedClass();            Arrays.*sort*(ar, in);  **for** (String str : ar)                   System.*out*.print(str + " ");            System.*out*.println(Arrays.*binarySearch*(ar, "b"));     }  **static** **class** NestedClass **implements** Comparator<String> {  **public** **int** compare(String s1, String s2) {  **return** s2.compareTo(s1);            }     }  } |

### Answer.

### /\*

### e d c b a -1

### \*/

### >compareTo() method will do the reverse sorting.

### >binarySearch() gives –1 because it should have been invoked using the same Comparator as was used during reverse sorting of the array.

### 

### Read:

### [COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

### [COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

### 

### Collection interview Question 80.

### Collection Output interview question 30.

|  |
| --- |
| import **java.util.EnumSet;**  **import** java.util.Set;  /\*\* \*/  **public** **class** EnumSetTest {  **private** **enum** Days {  *Monday*, *Tuesday*, *Wednesday*, *Thursday*, *Friday*, *Saturday*, *Sunday*;  **public** **static** Set<Days> *allDays* = EnumSet.*allOf*(Days.**class**);    **public** **static** Set<Days> *weekDays* = EnumSet.*range*(*Monday*, *Friday*);    **public** **boolean** isWeekDay() {  **return** *weekDays*.contains(**this**);     }     }     /\*\* Main \*/  **public** **static** **final** **void** main(**final** String args[]) {            System.*out*.println(Days.*weekDays*.size());              Days day=Days.*Monday*;            System.*out*.println( (day.isWeekDay() ? "is WeekDay" : "is weekEnd"));              day=Days.*Sunday*;            System.*out*.println( (day.isWeekDay() ? "is WeekDay" : "is weekEnd"));         day=Days.*Monday*;     System.*out*.println(Days.*allDays*.contains(day));     System.*out*.println(day.ordinal());     }  } |

### Answer.

### /\*OUTPUT

### 5

### is WeekDay

### is weekEnd

### true

### 0

### \*/

### *Read :* [*EnumSet in java*](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html)

### 

### Collection interview Question 81.

### Collection Output interview question 31.

|  |
| --- |
| import **java.util.LinkedHashSet;**  **import** java.util.Set;  /\*\* \*/  **public** **class** LinkedHashSetTest {  **public** **static** **void** main(String args[]) {            Set s = **new** LinkedHashSet();            s.add("1");            s.add(1);            s.add(3);            s.add(2);            System.*out*.println(s);     }  } |

### Answer.

### /\* OUTPUT

### [1, 1, 3, 2]

### \*/

### [LinkedHashSet](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) maintains insertion order and does not allow duplicates.

### *Read :* [*HashSet vs LinkedHashSet vs TreeSet - Similarity and Differences*](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)

### 

### Collection interview Question 82.

### Collection Output interview question 32.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.Collections;  **class** Employee **implements** Comparable<Employee>{     String name;     String id;  **public** Employee(String name, String id) {  **this**.name = name;  **this**.id = id;     }     @Override  **public** **int** compareTo(Employee otherEmployee) {  **return** **this**.name.compareTo(otherEmployee.name);     }     @Override  **public** String toString() {  **return** "{" + "name=" + name + ", id=" + id  + '}';     }  }  /\*\* \*/  **public** **class** ComparableUsage {  **public** **static** **void** main(String[] args) {         Employee emp1=**new** Employee("sam","4");         Employee emp2=**new** Employee("amy","2");         ArrayList<Employee> list=**new** ArrayList<Employee>();         list.add(emp1);         list.add(emp2);         Collections.*sort*(list);         System.*out*.println(list);       }  } |

### Answer.

### /\*OUTPUT

### [{name=amy, id=2}, {name=sam, id=4}]

### \*/

### compareTo method of Comparable has been implemented properly and will sort Employee class on basis of name in ascending order.

### Read : [Comparable vs Comparator - differences](http://www.javamadesoeasy.com/2015/04/comparable-vs-comparator-differences.html) for more.

### 

### Collection interview Question 83.

### Collection Output interview question 33.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.Collections;  **import** java.util.Comparator;  **class** Employee **implements** Comparator<Employee>{     String name;     String id;    **public** Employee() {}    **public** Employee(String name, String id) {  **this**.name = name;  **this**.id = id;     }       @Override  **public** **int** compare(Employee obj1, Employee obj2) {  **return** obj2.name.compareTo(obj1.name);     }     @Override  **public** String toString() {  **return** "{" + "name=" + name + ", id=" + id  + '}';     }  }  /\*\* \*/  **public** **class** ComparatorUsage {  **public** **static** **void** main(String[] args) {         Employee emp1=**new** Employee("sam","4");         Employee emp2=**new** Employee("amy","2");        ArrayList<Employee> list=**new** ArrayList<Employee>();         list.add(emp1);         list.add(emp2);         Collections.*sort*(list,**new** Employee());         System.*out*.println(list);     }  } |

### Answer.

### /\*OUTPUT

### [{name=sam, id=4}, {name=amy, id=2}]

### \*/

### compare method of Comparator has been implemented properly and will sort Employee class on basis of name in descending order.

### 

### Collection interview Question 84.

### Collection Output interview question 34.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.Collections;  **import** java.util.Comparator;  **class** Employee{     String name;  **public** Employee() {}  **public** Employee(String name) {  **this**.name = name;     }  **public** String toString() {  **return** "name=" + name;     }  **static** **class** ComparatorName **implements** Comparator<Employee>{  **public** **int** compare(Employee obj1, Employee obj2) {  **return** obj1.name.compareTo(obj2.name);         }     }  }  /\*\* \*/  **public** **class** ComparatorUsage {  **public** **static** **void** main(String[] args) {         Employee emp1=**new** Employee("ankit");         Employee emp2=**new** Employee("brad");          ArrayList<Employee> list=**new** ArrayList<Employee>();         list.add(emp1);         list.add(emp2);         Collections.*sort*(list,**new** Employee.ComparatorName());         System.*out*.println(list);     }  } |

### Answer.

### /\*OUTPUT

### [name=ankit, name=brad]

### \*/

### compare method of Comparator has been implemented properly by static class ComparatorName and will sort Employee class on basis of name in ascending order.

### 

### Collection interview Question 85.

### Collection Output interview question 35.

|  |
| --- |
| import **java.util.Arrays;**  **import** java.util.Comparator;  **class** Sort **implements** Comparator<Integer> {  **public** **int** compare(Integer o1, Integer o2) {  **return** o2.compareTo(o1);     }  }  /\*\* \*/  **public** **class** MyClass {  **public** **static** **void** main(String...a){         Integer intArray[]={2,3,1};         Arrays.*sort*(intArray, **new** Sort());  **for**(**int** i: intArray){            System.*out*.print(i+" ");         }     }  } |

### Answer.

### /\*OUTPUT

### 3 2 1

### \*/

### In program, we sort Integer array by using Arrays.sort (we will define Comparator to sort elements in descending order)

### *Read :* [*Arrays.sort to sort arrays by implementing Comparator and how Comparator of superclass can be used by subclasses*](http://www.javamadesoeasy.com/2015/04/arrayssort-to-sort-arrays-by.html)

### 

### Read:

### [COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

### [COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

### 

### Collection interview Question 86.

### Collection Output interview question 36.

|  |
| --- |
| import **java.util.Comparator;**  **import** java.util.Set;  **import** java.util.TreeSet;  /\*\* \*/  **public** **class** SortSet {  **public** **static** **void** main(String...a){         Set<Integer> treeSet = **new** TreeSet<Integer>(**new** Comparator<Integer>() {  **public** **int** compareTo(Integer o1, Integer o2) {  **return** o2.compareTo(o1);                   }            });         treeSet.add(3);         treeSet.add(1);         treeSet.add(2);         System.*out*.println(treeSet);     }  } |

### Answer.

### /\*OUTPUT

### compile time exception

### \*/

### We haven’t implemented compare method of Comparator. If compare would have been there in place of compareTo program would have compiled and executed properly, hence would have sorted elements of treeSet in reverse order..

### Read : [Sort Set by using TreeSet and by implementing Comparator and Comparable interface](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

### 

### Collection interview Question 87.

### Collection Output interview question 37.

|  |
| --- |
| import **java.util.Collection;**  **import** java.util.HashSet;  **import** java.util.Set;  **import** java.util.TreeSet;  /\*\* \*/  **public** **class** SortSet {  **public** **static** **void** main(String...a){         Collection<Integer> collection = **new** HashSet<Integer>();         collection.add(3);         collection.add(1);         collection.add(2);         Set<Integer> treeSet = **new** TreeSet<Integer>(collection);         System.*out*.println(treeSet);     }  } |

### Answer.

### /\*OUTPUT

### [1, 2, 3]

### \*/

### *If elements are stored in stored in* [*HashSet*](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*/*[*ArrayList*](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) *or any other class that implements* [*Collection*](http://www.javamadesoeasy.com/2015/04/collection-in-java.html)*, then we can use TreeSet’s addAll method or constructor for sorting.*

### *Read :* [*Sort Set by using TreeSet and by implementing Comparator and Comparable interface in java*](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

### 

### Collection interview Question 88.

### Collection Output interview question 38.

|  |
| --- |
| import **java.util.Collection;**  **import** java.util.HashSet;  **import** java.util.Set;  **import** java.util.TreeSet;  /\*\* \*/  **public** **class** SortSet {  **public** **static** **void** main(String...a){         Collection<Integer> collection = **new** HashSet<Integer>();         collection.add(3);         collection.add(1);         collection.add(2);         collection.add(**null**);         Set<Integer> treeSet = **new** TreeSet<Integer>();         treeSet.addAll(collection);         System.*out*.println(treeSet);     }  } |

### Answer.

### /\*OUTPUT

### Runtime Exception - NullPointerException

### \*/

### *If elements are stored in stored in* [*HashSet*](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*, then we can use TreeSet’s addAll method for sorting, but* [*TreeSet*](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) *does not allow null.*

### Collection interview Question 89.

### Collection Output interview question 39.

|  |
| --- |
| import **java.util.Comparator;**  **import** java.util.Map;  **import** java.util.TreeMap;  /\*\* \*/  **public** **class** SortMap {  **public** **static** **void** main(String...a){         Map<Integer, Integer> treeMap = **new** TreeMap<Integer, Integer>(**new** Comparator<Integer>(){  **public** **int** compare(Integer o1, Integer o2) {  **return** o2.compareTo(o1);                   }            });         treeMap.put(4, 1);         treeMap.put(2, 1);         treeMap.put(3, 1);           System.*out*.println(treeMap);     }  } |

### Answer.

### /\*OUTPUT

### {4=1, 3=1, 2=1}

### \*/

### *TreeMap* is sorted by natural order of keys, but we will implement Comparator interface to change the behaviour to sort TreeMap in descending order of keys.

### Here, [Comparator interface has been implemented in form of anonymous inner class.](http://www.javamadesoeasy.com/2015/04/sort-map-by-key-in-ascending-and.html)

### 

### Read:

### [COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50](http://www.javamadesoeasy.com/2015/05/collection-top-50-interview-questions.html)

### [COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

### 

### Collection interview Question 90.

### Collection Output interview question 40.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **class** Sort **implements** Comparator<Map.Entry<Integer, Integer>>{     @Override  **public** **int** compare( Map.Entry<Integer, Integer> entry1, Map.Entry<Integer, Integer> entry2 ){  **return** (entry2.getValue()).compareTo( entry1.getValue() );     }  }  /\*\* \*/  **public** **class** MyClass {  **public** **static** **void** main(String...a){         Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();         map.put(1, 2);         map.put(2, 1);         map.put(4, 8);           Set<Entry<Integer, Integer>> set = map.entrySet();         List<Entry<Integer, Integer>> list = **new** ArrayList<Entry<Integer, Integer>>(set);         Collections.*sort*(list, **new** Sort());  **for**(Map.Entry<Integer, Integer> entry:list)          System.*out*.print(entry.getKey());     }  } |

### Answer.

### /\*OUTPUT

### 412

### \*/

### Read : [Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method](http://www.javamadesoeasy.com/2015/04/sort-map-by-value-in-ascending-and.html)

### 

### Collection interview Question 91.

### Collection Output interview question 41.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.LinkedHashMap;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **class** Sort **implements** Comparator<Map.Entry<Integer, Integer>>{     @Override  **public** **int** compare( Map.Entry<Integer, Integer> entry1, Map.Entry<Integer, Integer> entry2 ){  **return** (entry2.getKey()).compareTo( entry1.getKey() );     }  }  /\*\* \*/  **public** **class** SortMap {  **public** **static** **void** main(String...a){         Map<Integer, Integer> map = **new** LinkedHashMap<Integer, Integer>();         map.put(4, 1);         map.put(2, 6);         map.put(5, 1);           Set<Entry<Integer, Integer>> entrySet = map.entrySet();         List<Entry<Integer, Integer>> listOfentrySet = **new** ArrayList<Entry<Integer, Integer>>(entrySet);           Collections.*sort*(listOfentrySet, **new** Sort());    **for**(Map.Entry<Integer, Integer> entry:listOfentrySet)          System.*out*.print(entry.getKey());     }  } |

### Answer.

### /\*OUTPUT

### 542

### \*/

### Read : [Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method](http://www.javamadesoeasy.com/2015/04/sort-map-by-value-in-ascending-and.html)

### 

### Collection interview Question 92.

### Collection Output interview question 42.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.List;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Number> numberList = **new** ArrayList<Number>();            numberList.add(2);            numberList.add(3);  *m*(numberList);     }  **static** **void** m(List<? **super** Double> l) {            System.*out*.print(l.get(0));            System.*out*.print(l.get(1));     }  } |

### Answer.

### /\*

### 23

### \*/

### 

### Collection interview Question 93.

### Collection Output interview question 43.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.List;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> l = **new** ArrayList<Integer>();            l.add(2);  *m*(l);       }  **static** **void** m(List<? **super** Double> l) {            System.*out*.println(l.get(0));            System.*out*.println(l.get(1));     }  } |

### Answer.  Program won’t compile.

### List<? super Double> can not accept List<Integer>, it can accept list of anySuperClassOfDouble i.e. List<Number> or List<Object>

### 

### Collection interview Question 94.

### Collection Output interview question 44.

|  |
| --- |
| class **Abc {**     <t> **void** display(t obj[]) {  **for** (t i : obj) {                   System.*out*.print(i + "  ");            }     }  }  **class** MyClass {  **public** **static** **void** main(String... args) {            Abc o = **new** Abc();              Integer i[] = { 1, 2 };            o.display(i);            Double d[] = { 1.1, 2.2 };            o.display(d);     }  } |

### Answer.

### /\*

### 1  2  1.1  2.2

### \*/

### because t can of any type may be Integer or double

### 

### Collection interview Question 95.

### Collection Output interview question 45.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.List;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> list = **new** ArrayList<Integer>();            list.add(2);            list.add(3);            System.*out*.println(*sum*(list));     }  **public** **static** **double** sum(List<? **extends** Number> list) {  **double** sum = 0;  **for** (Number num : list) {                   sum += num.doubleValue();            }  **return** sum;     }  } |

### Answer.

### /\*

### 5.0

### \*/

### List<? super Number> can accept List<Integer>, it can accept list of anySubClassOfNumber i.e. List<Double>, List<Float>, etc.

### 

### Collection interview Question 96.

### Collection Output interview question 46.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.List;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> list = **new** ArrayList<Integer>();            list.add(2);            list.add(3);  *m*(list);     }  **public** **static** **void** m(List<Number> list) {            System.*out*.println(list);     }  } |

### Answer.  Program won’t compile.

### List<Number> cannot accept List<Integer>, to avoid compilation error we must use List<? extends Number>

### 

### Collection interview Question 97.

### Collection Output interview question 47.

|  |
| --- |
| import **java.util.ArrayList;**  **import** java.util.List;  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            List<Integer> list = **new** ArrayList<Integer>();            list.add(1);            list.add(2);            System.*out*.println(*sum*(list));     }  **public** **static** **double** sum(List<? **extends** Number> list) {            list.add(4);  **double** sum = 0;  **for** (Number num : list) {                   sum += num.doubleValue();            }  **return** sum;     }  } |

### Answer.  Program won’t compile.

### List<? extends Number> cannot add or remove elements from list. So, list.add(4) will cause compilation error.

### 

### Collection interview Question 98.  Output

### Collection Output interview question question 48.

|  |
| --- |
| import **java.util.PriorityQueue;**  **public** **class** MyClass {  **public** **static** **void** main(String[] args) {            PriorityQueue<Integer> q = **new** PriorityQueue<Integer>();            q.add(1);            q.add(2);            q.add(3);            System.*out*.println(q.poll());            System.*out*.println(q.offer(4));            q.add(1);            q.remove(2);            System.*out*.println(q.peek());            System.*out*.println(q);     }  } |

### Answer.

### /\* OUTPUT

### 1

### true

### 1

### [1, 3, 4]

### \*/

### 

### [ArrayList custom implementation - add, get, remove Employee object in java](http://www.javamadesoeasy.com/2015/02/arraylist-custom-implementation-add-get.html)

*Contents of page :*

* [***1) Methods used in custom ArrayList in java****-*](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation-add-get.html#1)
* [***2) Full Program/SourceCode to add, get, remove Employee object***](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation-add-get.html#2) ***in*** [***custom ArrayList in java.***](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)

In this post i will be explaining how to **add, get, remove Employee object in** [**custom ArrayList.**](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)

**1) Methods used in custom ArrayList in java**-

|  |  |
| --- | --- |
| public void **add**(E value) | Add objects in **ArrayListCustom** |
| public E **get**(int index) | Method returns element on specific index. |
| public Object **remove**(int index) | Method returns removedElement on specific index, else it throws IndexOutOfBoundException if index is negative or greater than size of size. |
| public void **display**() | -Method displays all objects in **ArrayListCustom**.  **-Insertion order is guaranteed**. |
| private void **ensureCapacity**() | Method increases capacity of list by making it double. |

**2) Full Program/SourceCode to add, get, remove Employee object in** [**custom ArrayList in java.**](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)

|  |
| --- |
| **package** com.ankit;  **import** java.util.Arrays;  **class** **Employee** {  **private** String id;  **private** String name;       /\*\*     \* Employee constructor     \*/  **public** Employee(String id, String name) { // constructor  **this**.id = id;  **this**.name = name;     }     @Override  **public** String toString() {  **return** "Employee[id=" + id + ", name=" + name + "] ";     }  }  /\*\*   \*/  /\*\*  \* **@author** AnkitMittal  \* . All Contents are copyrighted and must not be reproduced in any form.  \* This class provides custom implementation of ArrayList(without using java api's)  \* Insertion order of objects is maintained.  \* Implementation allows you to store null as well.  \* **@param** <E>  \*/  **class** **ArrayListCustom**<E> {    **private** **static** **final** **int** *INITIAL\_CAPACITY* = 10;  **private** Object elementData[]={};  **private** **int** size = 0;   /\*\*   \* constructor.   \*/  **public** **ArrayListCustom**() {     elementData = **new** Object[*INITIAL\_CAPACITY*];   }   /\*\*    \* method adds elements in ArrayListCustom.    \*/  **public** **void** **add**(E e) {  **if** (size == elementData.length) {       ensureCapacity(); //increase current capacity of list, make it double.     }     elementData[size++] = e;   }   /\*\*    \* method returns element on specific index.    \*/   @SuppressWarnings("unchecked")  **public** E **get**(**int** index) {  **if** ( index <0 || index>= size) { //if index is negative or greater than size of size, we throw Exception.  **throw** **new** IndexOutOfBoundsException("Index: " + index + ", Size " + index);     }  **return** (E) elementData[index]; //return value on index.   }   /\*\*    \* method returns removedElement on specific index.    \* else it throws IndexOutOfBoundException if index is negative or greater than size of size.    \*/  **public** Object **remove**(**int** index) {  **if** ( index <0 || index>= size) { //if index is negative or greater than size of size, we throw Exception.  **throw** **new** IndexOutOfBoundsException("Index: " + index + ", Size " + index);     }       Object removedElement=elementData[index];  **for**(**int** i=index;i<size - 1;i++){        elementData[i]=elementData[i+1];     }     size--;   //reduce size of ArrayListCustom after removal of element.    **return** removedElement;   }   /\*\*    \* method increases capacity of list by making it double.    \*/  **private** **void** **ensureCapacity**() {  **int** newIncreasedCapacity = elementData.length \* 2;     elementData = Arrays.*copyOf*(elementData, newIncreasedCapacity);   }   /\*\*    \* method displays all the elements in list.    \*/  **public** **void** **display**() {      System.*out*.print("Displaying list : ");  **for**(**int** i=0;i<size;i++){             System.*out*.print(elementData[i]+" ");      }   }  }  /\*\*   \*/  /\*\*  \* Main class to test ArrayListCustom functionality.  \*/  **public** **class** **ArrayListEmployee** {    **public** **static** **void** main(String...a) {            ArrayListCustom<Employee> list = **new** ArrayListCustom<Employee>();     list.add(**new** Employee("1", "sam"));     list.add(**new** Employee("2", "amy"));     list.add(**new** Employee("3", "wil"));     list.add(**new** Employee("4", "cat"));     list.add(**new** Employee("1", "sam"));     list.add(**new** Employee("2", "amy"));     list.add(**null**);       list.display();     System.*out*.println("\nelement at index "+1+" = "+list.get(1));     System.*out*.println("element removed from index "+1+" = "+list.remove(1));       System.*out*.println("\nlet's display list again after removal at index 1");       list.display();       //list.remove(11); //will throw IndexOutOfBoundsException, because there is no element to remove on index 11.     //list.get(11);   //will throw IndexOutOfBoundsException, because there is no element to get on index 11.        }    }  /\***Output**  Displaying list : Employee[id=1, name=sam]  Employee[id=2, name=amy]  Employee[id=3, name=wil]  Employee[id=4, name=cat]  Employee[id=1, name=sam]  Employee[id=2, name=amy]  null  element at index 1 = Employee[id=2, name=amy]  element removed from index 1 = Employee[id=2, name=amy]  let's display list again after removal at index 1  Displaying list : Employee[id=1, name=sam]  Employee[id=3, name=wil]  Employee[id=4, name=cat]  Employee[id=1, name=sam]  Employee[id=2, name=amy]  null  \*/ |

**Collection interview Question 1. What is Collection framework in java?**

**Answer**. It’s the basic Collection framework interview question. Freshers must know about this.  [***java.util.Collection***](http://www.javamadesoeasy.com/2015/04/collection-in-java.html)*is the* root interface in the *hierarchy of Java Collection framework in java*.

The JDK does not provide any classes which directly implements this interface, but it provides classes which are implementations of more specific subinterfaces like [Set and List](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) in java.

java.util.[**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) extends java.util.Collection interface in java.

[**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

java.util.[**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) extends java.util.Collection interface in java.

[**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

***Also read >***

[COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

### [COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-3 > Q75- Q100](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview45.html)

**Collection interview Question 2. Which interfaces and classes are most frequently used in Collection framework in java?**

**Answer**. This collection framework interview question will test your practical knowledge. Freshers may get away by answering few interface and classes but experienced developers must answer this question in detail.

**Most frequently used interface in Collection framework are >**

[**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html), [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) and [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html).

**Most frequently used classes in Collection framework are >**

[**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

[**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

[**HashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)***,*** [**Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)***,*** [**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)***,*** [**LinkedHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html), [**TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html), [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) classes implements **Map** interface.

**Collection interview Question 3. What are subinterfaces of Collection interface in java? Is Map interface also a subinterface of Collection interface in java?**

**Answer**. [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) and [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) are subinterfaces of java.util.[**Collection**](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) in java.

*It’s important to note* [***Map***](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) *interface is a member of the Java Collections Framework, but it does not implement Collection interface in java.*

**Collection interview Question 4. What are differences between** [**ArrayList and LinkedList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-linkedlist-similarity-and.html) **in java?**

**Answer**. This is very important collection framework interview question in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | **java.util.LinkedList** |
| 1 | Structure | java.util.ArrayList is index based structure in java.  [https://lh3.googleusercontent.com/wCfo_q1uxCzZZCgGpetqEQYMeVj9YMJokT9-WJ7QY4jxCF11u5-WVIjVheBCfKlPJtQ9Bp5zzxTJcPgYLMr0N3n6PvjXPzd-7O-FJr2KoW7qrUjERB-yXK2YxFkH6qrLAX6hvdg5](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html) | A java.util.**LinkedList** is a data structure consisting of a group of **nodes** which together represent a sequence.  node is composed of a data and a reference (in other words, a **link**) to the next node in the sequence in java.  [https://lh3.googleusercontent.com/ykSE04usYkDTj50vuGVTWKtVGJootTOKa07Eub-E6D5KkOCNAb399G4agtbSKOyeaPAUvAngY6JjDMs-SBNmblDOXLv62eHNVIwEuGD5-GNXTP45Ubtyp0BYg0seOxGSpXHatWJP](http://www.javamadesoeasy.com/2015/01/doublylinkedlist-insert-and-delete-at.html) |
| 2 | **Resizable** | **ArrayList is Resizable-array in java.** | New node is created for storing new element in LinkedList in java. |
| 3 | **Initial capacity** | java.util.ArrayList is created with initial capacity of 10 in java. | For storing every element node is created in LinkedList, so linkedList’s initial capacity is 0 in java. |
| 4 | Ensuring **Capacity**/ resizing. | ArrayList is created with initial capacity of 10.  ArrayList’s size is **increased by 50%** i.e. after resizing it’s size become 15 in java. | For storing every element node is created, so linkedList’s initial capacity is 0, it’s size grow with addition of each and every element in java. |
| 5 | RandomAccess interface | ArrayList implements RandomAccess(Marker interface) to indicate that they support fast random access (i.e. index based access) in java. | LinkedList does not implement RandomAccess interface in java. |
| 6 | AbstractList and AbstractSequentialList | ArrayList extends AbstractList (abstract class) which provides implementation to  List interface to minimize the effort required to implement this interface backed by RandomAccess interface. | LinkedList extends AbstractSequentialList (abstract class), AbstractSequentialList extends AbstractList.  In LinkedList, data is accessed sequentially, so for obtaining data at specific index, iteration is done on nodes sequentially in java. |
| 7 | How **get(index)** method works?  (Though difference has been discussed briefly in above 2 points but in this in point we will figure difference in detail.) | Get method of ArrayList directly gets element on specified index. Hence, offering O(1) complexity in java. | Get method of LinkedList iterates on nodes sequentially to get element on specified index. Hence, offering O(n) complexity in java. |
| **8** | **When to use** | **Use ArrayList when get operations is more frequent than add and remove operations in java.** | **Use LinkedList when add and remove operations are more frequent than get operations in java.** |

For more detail like complexity comparison of method please read : [**ArrayList vs LinkedList in java**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-linkedlist-similarity-and.html)

**Collection interview Question 5. What are differences between** [**ArrayList and Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html) **in java?**

**Answer**. Another very important collection framework interview question to differentiate between ArrayList and Vector in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | ***java.util.Vector*** |
| 1 | synchronization | java.util.ArrayList is **not synchronized**  (because 2 threads on same ArrayList object can access it at same time).  I have created [**program**](http://www.javamadesoeasy.com/2015/05/consequence-of-using-arraylist-in.html)to show consequence of using ArrayList in multithreading environment.  In the program we will implement our own arrayList in java. | java.util.Vector is **synchronized** (because 2 threads on same Vector object cannot  access it at same time).  I have created [**program**](http://www.javamadesoeasy.com/2015/05/advantage-of-using-vector-in.html)to show advantage of using Vector in multithreading environment.  In the program we will implement our own vector in java. |
| 2 | Performance | ArrayList is not synchronized, hence its operations are **faster** as compared to Vector in java. | Vector is synchronized, hence its operations are **slower** as compared to ArrayList in java.  If we are working not working in multithreading environment jdk recommends us to use ArrayList. |
| 3 | Enumeration | **Enumeration** is [**fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), means any modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException in java. | **Enumeration** is **fail-safe**, means any modification made to Vector during iteration using Enumeration don’t throw any exception in java. |
| 4 | Introduced  in which java version | ArrayList was introduced in second version of java i.e. **JDK 2.0** | Vector was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the List interface, hence making it a member of member of the [Java Collections Framework](http://www.javamadesoeasy.com/2015/04/collection-in-java.html). |
| 5 | Ensuring Capacity/ resizing. | ArrayList is created with initial capacity of 10.  When its full size is **increased by 50%** i.e. after resizing it’s size become 15 in java. | Vector is created with initial capacity of 10.  Vector’s size is **increased by 100%** i.e. after resizing it’s size become 20 in java. |
| 6 | Custom implementation | [https://lh3.googleusercontent.com/2yHNtovknpsdxOKpK4Sd3oFiHP3fKhhrMsZDH3DJaRNsvWB7RnEqtXjyS5yrk6175OwELqF6-viscZQxK8uMK58-gmsz1tN0sHmVSJBEwKJ1UZwle61DItNZeF8MDwyFx-NUXHDU](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)Read : [ArrayList custom implementation](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html) | [https://lh5.googleusercontent.com/Gm_SkaJR6TXlRpDt3ipjw739Gfcg4b2V-pMT4WWrw0cMVTsUhDvApjrvMATYF4XiBotrk8O0Sbc7kwi_v0V7SrrY_cfTUU5dR0_rEogBBG34UB1IviiWwIHcpc5XuH_k5KmOsnmP](http://javamadesoeasy.com/2015/02/vector-custom-implementation.html)Read :  [Vector custom implementation](http://javamadesoeasy.com/2015/02/vector-custom-implementation.html) |

For more detail like complexity comparison of method please read: [**ArrayList vs Vector- Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)

**Collection interview Question 6. What are differences between** [**List and Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) **interface in java?**

**Answer**. Another very very important collection framework interview question to differentiate between **List and Set** in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.List*** | ***java.util.Set*** |
| 1 | Insertion order | java.util.List is ordered collection it **maintain insertion order** in java. | *Most of the java.util.Set implementation* does not **maintain insertion order**.  HashSet does not maintains insertion order in java.  Thought LinkedHashSet maintains insertion order in java.    TreeSet is sorted by natural order in java. |
| 2 | Duplicate elements | List **allows to store duplicate elements** in java. | *Set does* ***not allow to store duplicate elements*** in java*.* |
| 3 | Null keys | List allows to store **many null keys** in java. | Most of the Set implementations allow to add only **one null** in java**.**  TreeSet does not allow to add null in java. |
| 4 | Getting element on specific **index** | List implementations provide get method to get element on specific index in java.  ArrayList, Vector, copyOnWriteArrayList and LinkedList provides -  *get(int index)*  Method returns element on specified *index*.  **Get method directly gets element on specified index. Hence, offering O(1) complexity.** | Set implementations does not provide any such get method to get element on specified index in java. |
| 5 | Implementing classes | [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. | [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. |
| 6 | listIterator | **listIterator** method returns listIterator to iterate over elements in List in java.  **listIterator provides** additional methods as compared to iterator like  **hasPrevious(), previous(), nextIndex(), previousIndex(), add(E element), set(E element)** | Set does not provide anything like listIterator. It simply return Iterator in java. |
| 7 | Structure and resizable | **List** are Resizable-array implementation of the java.util.**List** interface in java. | Set uses [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)for their implementation.  Hence, structure is map based and resizing depends on Map implementation.  *Example >* [***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *internally uses* [*HashMap*](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html)*.* |
| 8 | Index based structure /RandomAccess | As **ArrayList** uses array for implementation it is index based structure, hence provides random access to elements.  But [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) is not indexed based structure in java. | Set is not index based structure at all in java. |

For more detail read : [**List vs Set - Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html)

**Collection interview Question 7. What are differences between** [**Iterator and ListIterator**](http://www.javamadesoeasy.com/2015/04/iterator-vs-listiterator-similarity-and.html)**? in java**

**Answer**. This collection framework interview question is tests your knowledge of iterating over different collection framework classes in java.

|  |  |  |
| --- | --- | --- |
|  | ***java.util.ListIterator*** | ***java.util.Iterator*** |
| 1 | **hasPrevious()**  method returns true if this listIterator has more elements when traversing the list in the reverse direction. | **No such method** in java.util.Iterator. |
| 2 | **previous()**  returns previous element in iteration (traversing in backward direction).  if the iteration has no previous elements than NoSuchElementException is thrown. | **No such method** in java.util.Iterator. |
| 3 | **nextIndex()**  method returns the index of the element that would be returned by a subsequent call to next() method. If listIterator is at the end of the list than method returns size of list. | **No such method** in java.util.Iterator. |
| 4 | **previousIndex()**  method returns the index of the element that would be returned by a subsequent call to previous() method. If listIterator is at the start of the list than method returns -1. | **No such method** in java.util.Iterator. |
| 5 | **add(E element)**  Method inserts the specified **element** into the list.  The element is inserted immediately before the element that would be returned by next (So, subsequent call to next would be unaffected), if any, and after the element that would be returned by previous (So,subsequent call to previous would return the new **element**), if any.  If the list does not contain any element than new **element** will be the sole element in the list. | **No such method** in java.util.Iterator. |
| 6 | **set(E element)**  Method replaces the last element returned by next() or previous() method with the specified **element**. This call can be made only if neither remove nor add have been called after the last call to next or previous.  If call to set() method is followed up by any call made to remove() or add() method after next() or previous() than UnsupportedOperationException is thrown. | **No such method** in java.util.Iterator. |
| 7 | All the implementations of [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) interface like [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html), [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes returns listIterator. | All Implementation classes of [**Collection**](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) interface’s subinterfaces like [Set and List](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) return iterator. |

For more detail read : [**Iterator vs ListIterator - Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/iterator-vs-listiterator-similarity-and.html)

**Collection interview Question 8. What are differences between** [**Collection and Collections**](http://www.javamadesoeasy.com/2015/04/collection-vs-collections-differences.html) **in java?**

**Answer**.  This is another very important collection framework interview question.In real projects you must have used both Collection and Collections but what is the difference between two of them in java?

java.util.[***Collection***](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) ***​*** *is the* root **interface** in the ​*hierarchy of Java Collection framework​*.

The JDK does not provide any classes which directly implements java.util.Collection interface, but it  provides classes such as [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html), [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html), [**vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html), [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html), [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [CopyOnWriteArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html), [CopyOnWriteArraySet](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html), [ConcurrentSkipListSet](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)  which implements more specific subinterfaces like ​[Set and List​](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) in java.

java.util.**Collections** is a utility **class** which **consists** of **static methods** that **operate on** or return **Collection** in java.

**java.util.Collections provides method like >**

* **reverse** method for reversing [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) in java.
* **shuffle** method for shuffling elements of **List** in java.
* **unmodifiableCollection**, [**unmodifiableSet**](http://www.javamadesoeasy.com/2015/04/hashset-making-set-unmodifiable-using.html), [**unmodifiableList**](http://www.javamadesoeasy.com/2015/04/arraylist-making-list-unmodifiable.html), [**unmodifiableMap**](http://www.javamadesoeasy.com/2015/04/hashmap-making-map-unmodifiable-using.html) methods for making **List**, [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) and [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) unmodifiable in java.
* **min** method to return smallest element in **Collection** in java.
* **max** method to return smallest element in **Collection**.
* **sort** method for sorting **List**.
* **synchronizedCollection**, [**synchronizedSet**](http://www.javamadesoeasy.com/2015/04/hashset-synchronizing-using.html), [**synchronizedList**](http://www.javamadesoeasy.com/2015/04/arraylist-synchronizing-using.html), [**synchronizedMap**](http://www.javamadesoeasy.com/2015/04/hashmap-synchronizing-map-using.html)methods for synchronizing **List**, **Set** and **Map** respectively in java**.**

Additionally you must know that *java.util.Collection and java.util.Collections both were introduced in* ***second version of java i.e. in JDK 2.0.***

**Collection interview Question 9. What are core classes and interfaces in java.util.List hierarchy in java?**

**Answer**. Freshers must know core classes in List hierarchy but experienced developers must be able to explain this java.util.List hierarchy in detail.

[](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html)

java.util.[**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface extends java.util.Collection interface.

java.util.[**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***, java.util.***[**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***, java.util.***[**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***, java.util.concurrent.***[**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)classes implements java.util.[**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

Also some abstract classes like java.util.**AbstractCollection**, java.util.**AbstractList** and java.util.**AbstractSequentialList** have been mentioned in hierarchy.

**Collection interview Question 10. What are core classes and interfaces in java.util.Set hierarchy?**

**Answer**. Freshers must know core classes in Set hierarchy but experienced developers must be able to explain this java.util.Set hierarchy in detail.

[](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)

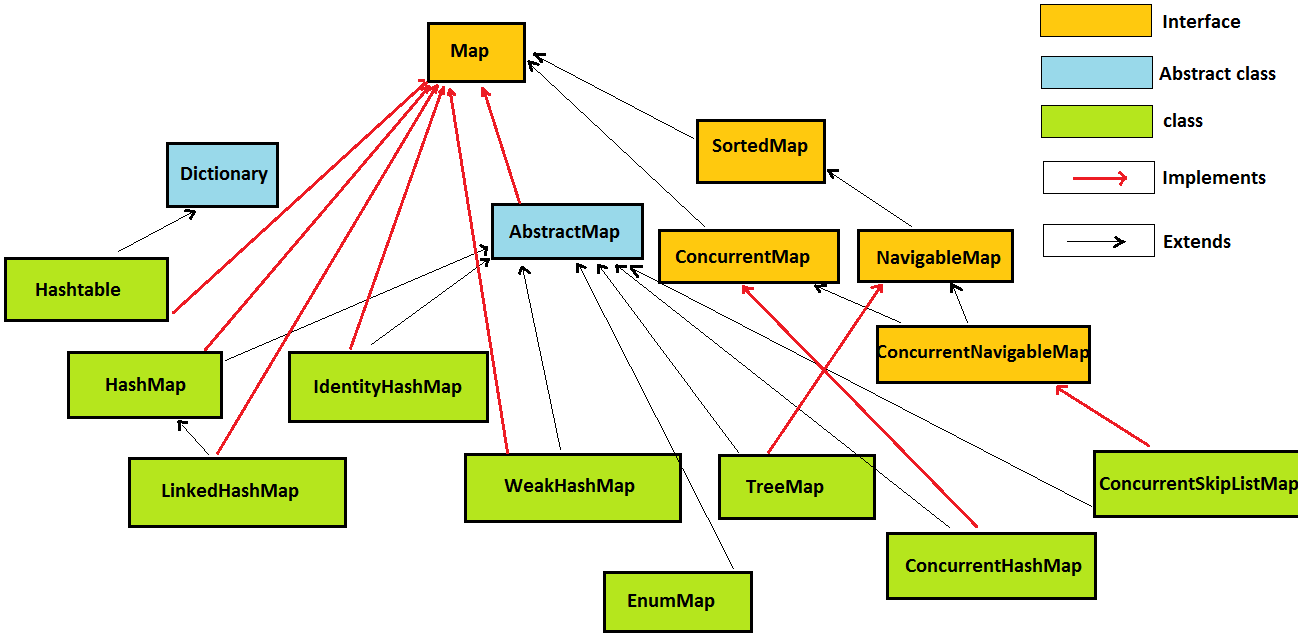
java.util.[**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface extends java.util.Collection interface.

java.util.[**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***, java.util.concurrent.***[**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***, java.util.***[**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***, java.util.***[**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), java.util.concurrent.[**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), java.util.[**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements java.util.[**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

Also some abstract classes like java.util.**Dictionary** and java.util.**AbstractSet** and java.util.**AbstractCollection** have been mentioned in hierarchy.

**Collection interview Question 11. What are core classes and interfaces in java.util.Map hierarchy?**

**Answer**. Freshers must know core classes in Map hierarchy but experienced developers must be able to explain this java.util.Map hierarchy in detail.

[](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)

**java.util.Map** interface extends java.util.Collection interface.

java.util.[**HashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)***, java.util.***[**Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)***, java.util.concurrent.***[**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)***, java.util.***[**LinkedHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html), java.util.[**TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html), java.util.concurrent.[**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html), java.util.[**IdentityHashMap**](http://www.javamadesoeasy.com/2015/04/identityhashmap-in-java.html), java.util.[**WeakHashMap**](http://www.javamadesoeasy.com/2015/04/weakhashmap-in-java.html), java.util.[**EnumMap**](http://www.javamadesoeasy.com/2015/04/enummap-in-java-with-program.html) classes implements java.util.**Map** interface.

Also some abstract classes like java.util.**Dictionary** and java.util.**AbstractMap** have been mentioned in hierarchy.

**Collection interview Question 12.  What are differences between** [**Iterator and Enumeration**](http://www.javamadesoeasy.com/2015/04/iterator-vs-enumeration-differences-and.html) **in java?**

**Answer**. Experienced developers must be well versed to answer this collection framework interview question in java.

***Differences*** *between java.util.****Iterator*** *and java.util.****Enumeration*** *in java**>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Property*** | ***java.util.Enumeration*** | ***java.util.Iterator*** |
| 1 | Remove elements during iteration | java.util.Enumeration **doesn’t allows** to remove elements from collection during iteration in java. | java.util.Iterator **allows** to remove elements from collection during iteration by using **remove()** method in java. |
| 2 | Improved naming conventions in Iterator | **nextElement()**  Method Returns the next element of this enumeration if this enumeration object has at least one more element to provide.  **hasMoreElements()**  returns true if enumeration contains more elements. | **nextElement()** has been changed to **next()** in Iterator.  And  **hasMoreElements()** has been changed to **hasNext()** in Iterator. |
| 3 | Introduced in  which java  version | Enumeration was introduced in first version  of java i.e. ​**JDK 1.0** | Iterator was introduced in second version  of java i.e. ​**JDK 2.0**  Iterator was introduced to replace Enumeration in the Java Collections Framework. |
| 4 | Recommendation | **Java docs** recommends iterator over enumeration**.** | **Java docs** recommends iterator over enumeration**.** |
| 5 | Enumeration and Iterator over [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html) | **Enumeration** returned by Vector is [**fail-safe**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), means any modification made to Vector during iteration using Enumeration don’t throw any exception in java. | **Iterator** returned by Vector are [**fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to ArrayList during iteration will throw ConcurrentModificationException  in java. |

### For more detail read : [Iterator vs Enumeration - Differences and similarities in java](http://www.javamadesoeasy.com/2015/04/iterator-vs-enumeration-differences-and.html)

**Collection interview Question 13. How do we override equals and hashcode method in java, write a code to use Employee as key in HashMap in java? (Important)**

**Answer**.  This is one of the most important collection framework interview question in java. Prepare for this question properly. Freshers must know the concept how to override equals and hashcode method but experienced developers must be able to write the java code to override equals and hashcode neatly. We will override equals() and hashCode() like this -

By overriding equals() and hashCode() method we could use custom object as key in HashMap.

1)  Check whether obj is null or not.

**if(obj==null) //If obj is null, return without comparing obj & Employee class.**

2)  check whether  obj is instance of Employee class or not.

**if(this.getClass()!=obj.getClass()) //identifies whether obj is instance of Employee class or not.**

3) Then, type cast obj into employee instance.

**Employee emp=(Employee)obj;  //type cast obj into employee instance.**

|  |
| --- |
| **@Override**  **public boolean equals(Object obj){**    **if(obj==null)**  **return false;**    **if(this.getClass()!=obj.getClass())**  **return false;**    **Employee emp=(Employee)obj;**  **return (emp.id==this.id || emp.id.equals(this.id))**  **&& (emp.name==this.name || emp.name.equals(this.name));**  **}**    **@Override**  **public int hashCode(){**  **int hash=(this.id==null ? 0: this.id.hashCode() ) +**  **(this.name==null ? 0: this.name.hashCode() );**  **return hash;**  **}** |

Let’s say in an organisation there exists a employee with **id=1 and name=’sam’**     and **some data** is stored corresponding to him, but if modifications have to be made in data, **previous data must be overridden**.

[DETAILED DESCRIPTION : Override equals() and hashCode() method](http://www.javamadesoeasy.com/2015/02/override-equals-and-hashcode-method.html).

**Must read :** [**Overriding equals and hashcode method - Top 18 Interview questions in java**](http://www.javamadesoeasy.com/2015/02/overriding-equals-and-hashcode-method.html)

**Collection interview Question 14. What classes should i prefer to use a key in HashMap in java? (Important)**

**Answer**. This collection framework interview question will check your in depth knowledge of Java’s Collection Api’s. we should prefer **String, Integer, Long, Double, Float, Short and any other wrapper class.** Reason behind using them as a key is that they override equals() and hashCode() method, we need not to write any explicit code for overriding equals() and hashCode() method in java.

Let’s use Integer class as key in HashMap(Example) -

|  |
| --- |
| **import** java.util.HashMap;  **import** java.util.Map;  **public** **class** StringInMapExample {  **public** **static** **void** main(String...a){             //HashMap's key=Integer class  (Integer’s api has already overridden hashCode() and equals() method for us )            Map<Integer, String> hm=**new** HashMap<Integer, String>();            hm.put(1, "data");            hm.put(1, "data OVERRIDDEN");              System.*out*.println(hm.get(1));       }  }  /\*OUTPUT  data OVERRIDDEN  \*/ |

If, we note above program, what we will see is we didn’t override equals() and hashCode() method, but still we were able to store data in HashMap, override data and retrieve data using get method.

>Let’s check in **Integer’s API**, how Integer class has overridden equals() and hashCode() method :

|  |
| --- |
| **public** **int** **hashCode**() {  **return** value;  }  **public** **boolean** **equals**(Object obj) {  **if** (obj **instanceof** Integer) {  **return** value == ((Integer)obj).intValue();         }  **return** **false**;  } |

**Collection interview Question 15. What are differences between** [**HashMap and Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) **in java?**

**Answer**. Fresher and Experienced developers must answer this important collection framework interview question in detail in java.

***Differences*** *between java.util.*[***HashMap***](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) *and java.util.****Hashtable*** *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.HashMap*** | ***java.util.Hashtable*** |
| 1 | synchronization | java.util.HashMap is **not synchronized**  (because 2 threads on same HashMap object can access it at same time) in java. | java.util.Hashtable is **synchronized** (because 2 threads on same Hashtable object cannot access it at same time) in java. |
| 2 | Performance | HashMap is not synchronized, hence its operations are **faster** as compared to Hashtable in java. | Hashtable is synchronized, hence its operations are **slower** as compared to HashMap in java.  If we are working not working in multithreading environment jdk recommends us to use HashMap. |
| 3 | Null keys and values | HashMap allows to store **one null key** and **many null values** i.e. many keys can have null value in java. | Hashtable does **not allow to store null key or null value**.  Any attempt to store null key or value throws runtimeException (NullPointerException) in java. |
| 4 | Introduced  in which java version | HashMap was introduced in second version of java i.e. **JDK 2.0** | Hashtable was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the Map interface, hence making it a member of member of the [Java Collections Framework](http://download.oracle.com/javase/7/docs/technotes/guides/collections/index.html). |
| 5 | Recommendation | In non-multithreading environment it is recommended to use HashMap than using Hashtable in java. | I**n java 5 i.e. JDK 1.5**, it is **recommended** to use [ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/concurrenthashmap-in-java.html) than using Hashtable. |
| 6 | Extends Dictionary (Abstract class, which is obsolete) | HashMap does not extends Dictionary in java. | Hashtable extends Dictionary (which maps non-null keys to values. In a given Dictionary we can look up value corresponding to key) in java. |

For more detail read : [**HashMap and Hashtable - Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)

**Collection interview Question 16. when to use** [**HashSet vs LinkedHashSet vs TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) **in java?**

**Answer**. Another very important collection framework interview question to differentiate between **following Set implementations** in java.

***Differences*** *between java.util.*[***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *vs java.util.****LinkedHashSet*** *vs java.util.****TreeSet*** *in java>*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Property | *java.util.HashSet* | *java.util.LinkedHashSet* | *java.util.TreeSet* |
| 1 | Insertion order | java.util.HashSet does not maintains insertion order in java.  Example in java >  **set.add("b");**  **set.add("c");**  **set.add("a");**  Output >  **No specific order** | java.util.LinkedHashSet maintains insertion order in java.  Example in java >  **set.add("b");**  **set.add("c");**  **set.add("a");**  Output >  **b**  **c**  **a** | java.util.TreeSet is sorted by natural order in java.  Example in java >  **set.add("b");**  **set.add("c");**  **set.add("a");**  Output >  **a**  **b**  **c** |
| 2 | Null elements | HashSet allows to store **one null** in java**.** | LinkedHashSet allows to store **one null** in java. | TreeSet does **not** allows to store **any null** in java.  Any attempt to add null throws runtimeException (NullPointerException). |
| 3 | Data structure internally used for storing data | For storing elements HashSet internally uses HashMap. | For storing elements LinkedHashSet internally uses  LinkedHashMap. | For storing elements TreeSet internally uses TreeMap. |
| 4 | Introduced  in which java version | java.util.HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | java.util.LinkedHashSet was introduced in second version of java (1.4) i.e. **JDK 4.0** | java.util.TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0** |
| 5 | Implements which interface | HashSet implements **java.util.**[**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)interface. | LinkedHashSet implements **java.util.Set** interface. | TreeSet implements **java.util.Set**  **java.util.SortedSet**  **java.util.NavigableSet** interface. |

For more detail read : [**HashSet vs LinkedHashSet vs TreeSet in java**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)

# LinkedHashSet class in Java with Examples

[**2.2**](https://www.geeksforgeeks.org/easy/)

A LinkedHashSet is an ordered version of [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/) that maintains a doubly-linked List across all elements. When the iteration order is needed to be maintained this class in used. When iterating through a [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/) the order is unpredictable, while a LinkedHashSet lets us iterate through the elements in the order in which they were inserted.when cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

**Syntax:**

LinkedHashSet<String> hs = new LinkedHashSet<String>();

* Contains unique elements only like [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/). It extends [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/) class and implements Set interface.
* Maintains insertion order.

Basic **Operations** of LinkedHashSet:

|  |
| --- |
| import java.util.LinkedHashSet;  public class Demo  {      public static void main(String[] args)      {          LinkedHashSet<String> linkedset =                             new LinkedHashSet<String>();            // Adding element to LinkedHashSet          linkedset.add("A");          linkedset.add("B");          linkedset.add("C");          linkedset.add("D");            //This will not add new element as A already exists          linkedset.add("A");          linkedset.add("E");            System.out.println("Size of LinkedHashSet = " +                                      linkedset.size());          System.out.println("Original LinkedHashSet:" + linkedset);          System.out.println("Removing D from LinkedHashSet: " +                              linkedset.remove("D"));          System.out.println("Trying to Remove Z which is not "+                              "present: " + linkedset.remove("Z"));          System.out.println("Checking if A is present=" +                              linkedset.contains("A"));          System.out.println("Updated LinkedHashSet: " + linkedset);      }  } |

Run on IDE

**Output:**

Size of LinkedHashSet=5

Original LinkedHashSet:[A, B, C, D, E]

Removing D from LinkedHashSet: true

Trying to Remove Z which is not present: false

Checking if A is present=true

Updated LinkedHashSet: [A, B, C, E]

[LinkedHashmap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/) vs LinkedHashset

* [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/) does a mapping of keys to values whereas a LinkedHashSet simply stores a collection of things with no duplicates.
* LinkedHashMap extends HashMap and LinkedHashSet extends HashSet.

**Important :** Keeping the insertion order in both LinkedHashmap and LinkedHashset have additional associated costs, both in terms of spending additional CPU cycles and needing more memory. If you do not need the insertion order maintained, it is recommended to use the lighter-weight [HashSet](http://quiz.geeksforgeeks.org/hashset-in-java/) and [HashMap](https://www.geeksforgeeks.org/hashmap-treemap-java/) instead.

**Collection interview Question 17. What are differences between** [**HashMap and ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) **in java?**

**Answer**. Take my words java developers won’t be able to get away from this very important collection framework interview question.

***Differences*** *between java.util.*[***HashMap***](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) *and java.util.concurrent.*[*ConcurrentHashMap*](http://www.javamadesoeasy.com/2015/04/concurrenthashmap-in-java.html) *in java >*

|  |  |  |
| --- | --- | --- |
| Property | *java.util.****HashMap*** | *java.util.concurrent.* ***ConcurrentHashMap*** |
| synchronization | HashMap is **not synchronized.** | ConcurrentHashMap is **synchronized**. |
| 2 threads on same Map object can access it at concurrently? | Yes, because HashMap is not synchronized**.** | Yes.  But how despite of being synchronized, 2 threads on same *ConcurrentHashMap* object can access it at same time?  *ConcurrentHashMap* is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently. |
| Performance | We will **synchronize HashMap and then compare its performance with ConcurrentHashMap**.  *We can synchronize hashMap by using Collections’s class* ***synchronizedMap*** *method.*   |  | | --- | | *Map synchronizedMap = Collections.****synchronizedMap****(hashMap);* |   *Now, no 2 threads can access same instance of map concurrently.*  **Hence synchronized HashMap’s performance is slower as compared to ConcurrentHashMap.**  But why we didn’t compared HashMap (unSynchronized) with ConcurrentHashMap?  Because performance of unSynchronized collection is always better than some synchronized collection. As, default (unSynchronized) hashMap didn’t cause any locking. | **ConcurrentHashMap’s performance is faster as compared to HashMap (**because it is divided into segments, as discussed in above point**).**  [*Read this post for performance comparison between HashMap and ConcurrentHashMap.*](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) |
| Null keys and values | HashMap allows to store **one null key** and **many null values** i.e. any key can have null value. | ConcurrentHashMap does **not allow to store null key or null value**.  Any attempt to store null key or value throws runtimeException (NullPointerException). |
| iterators | The iterators returned by the iterator() method of HashMap are [***fail-fast***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) *>*  *hashMap.keySet().iterator()*  *hashMap.values().iterator()*  *hashMap.entrySet().iterator()*  all three iterators are ***fail-fast*** | iterators are [***fail-safe***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*.*  *concurrentHashMap.keySet().iterator()*  *concurrentHashMap.values().iterator()*  *concurrentHashMap.entrySet().iterator()*  all three iterators are ***fail-safe.*** |
| **putIfAbsent** | HashMap does not contain putIfAbsent method.  ***putIfAbsent*** *method is equivalent to writing following code >*   |  | | --- | | **synchronized** (map){  **if** (!*map*.containsKey(key))  **return** *map*.put(key, value);  **else**  **return** *map*.get(key);  } |   [**Program to create method that provides functionality similar to putIfAbsent method of ConcurrentHashMap and to be used with HashMap**](http://www.javamadesoeasy.com/2015/04/program-to-create-method-that-provides.html) | If map does not contain specified **key**, put specified **key-value** pair in map and return null.  If map already contains specified **key**, return value corresponding to specified **key**.    [**Program to use ConcurrentHashMap’s putIfAbsent method**](http://www.javamadesoeasy.com/2015/04/program-to-use-concurrenthashmaps.html) |
| Introduced  in which java version | HashMap was introduced in **java 2 i.e. JDK 1.2**, | ConcurrentHashMap was introduced in **java 5** i.e. **JDK 1.5**, since its introduction Hashtable has become obsolete, because of concurrency level its performance is better than Hashtable. |
| Implements which interface | HashMap implements **java.util.**[**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | ConcurrentHashMap implements  **java.util.Map** and  **java.util.concurrent.ConcurrentMap** |
| Package | HashMap is in **java.util** package | ConcurrentHashMap is in **java.util.concurrent** package. |

For more detail read : [**HashMap and ConcurrentHashMap in java**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)

**Collection interview Question 18. When to use** [**HashMap vs Hashtable vs LinkedHashMap vs TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) **in java?**

**Answer**. Another important collection framework interview question

to differentiate between **following Map implementations** in java.

***Differences*** *between java.util.*[***HashMap***](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) *vs java.util.*[***Hashtable***](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)*vs java.util.****LinkedHashMap*** *vs java.util.*[***TreeMap***](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) ***>***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Property** | ***HashMap*** | ***Hashtable*** | ***LinkedHashMap*** | ***TreeMap*** |
| 1 | Insertion order | HashMap does not maintains insertion order in java. | Hashtable does not maintains insertion order in java. | LinkedHashMap  maintains insertion order in java. | TreeMap is sorted by natural order of keys in java. |
| 2 | Performance | HashMap is not synchronized, hence its operations are **faster** as compared to Hashtable. | Hashtable is synchronized, hence its operations are **slower** as compared HashMap.  If we are working not working in multithreading environment jdk recommends us to use HashMap. | LinkedHashMap must be used only when we want to maintain insertion order. **Time and space overhead** is there because for maintaining order it internally uses **Doubly Linked list**. | TreeMap must be used only when we want sorting based on natural order. Otherwise sorting operations cost performance. (Comparator is called for sorting purpose) |
| 3 | Null keys and values | HashMap allows to store **one null key** and **many null values** i.e. many keys can have null value in java. | Hashtable does **not allow to store null key or null value**.  Any attempt to store null key or value throws runtimeException (NullPointerException) in java. | LinkedHashMap allows to store **one null key** and **many null values** i.e. any key can have null value in java. | TreeMap does **not allow to store null key but allow many null values**.  Any attempt to store null key throws runtimeException (NullPointerException) in java. |
| 4 | Implements which interface | HashMap implements **java.util.**[**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | Hashtable implements **java.util.Map** | LinkedHashMap implements **java.util.Map** | TreeMap implements  **java.util.Map**  **java.util.SortedMap**  **java.util.NavigableMap** |
| 5 | Implementation uses? | HashMap use [**buckets**](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html) | Hashtable use **buckets** | LinkedHashMap uses [**doubly linked lists**](http://www.javamadesoeasy.com/2015/02/linkedhashmap-custom-implementation.html) | TreeMap uses **Red black tree** |
| 6 | Complexity of put, get and remove methods | O(1) | O(1) | O(1)  **overhead** of updating **Doubly Linked list** for maintaining order it internally uses. | O(log(n)) |
| 7 | Extends java.util.**Dictionary** (Abstract class, which is obsolete) | HashMap **doesn’t** extends Dictionary. | Hashtable **extends** Dictionary (which maps non-null keys to values. In a given Dictionary we can look up value corresponding to key) | LinkedHashMap **doesn’t** extends Dictionary. | TreeMap **doesn’t** extends Dictionary. |
| 8 | Introduced in which java version? | HashMap was introduced in second version of java i.e. **JDK 2.0** | Hashtable was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the Map interface, hence making it a member of member of the [Java Collections Framework](http://download.oracle.com/javase/7/docs/technotes/guides/collections/index.html). | LinkedHashMap was introduced in fourth version of java i.e. **JDK 4.0** | TreeMap was introduced in second version of java i.e. **JDK 2.0** |

For more detail read : [**HashMap vs Hashtable vs LinkedHashMap vs TreeMap in java**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)

**Collection interview Question 19. What are differences between** [**HashMap vs IdentityHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-identityhashmap-similarity.html) **in java?**

**Answer**. This is tricky and complex collection framework interview question for experienced developers in java.

***Differences*** *between java.util.*[***HashMap***](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)*and java.util.*[***IdentityHashMap***](http://www.javamadesoeasy.com/2015/04/identityhashmap-in-java.html) *in java**>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.HashMap*** | ***java.util.IdentityHashMap*** |
| 1 | **Keys comparison***object-equality  vs reference-equality* | **HashMap** when comparing keys (and values) performs object-equality not reference-equality. In an HashMap, two keys k1 and k2 are equal if and only if (k1==null ? k2==null : k1.equals(k2)) | **IdentityHashMap** when comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2) |
| 2 | Initial size | Constructs a new HashMap, Its initial capacity is 16 in java.   |  | | --- | | **new** HashMap(); | | Constructs a new IdentityHashMap, with maximum size of 21 in java.   |  | | --- | | **new** IdentityHashMap(); | |
| 3 | Introduced in which java version | HashMap was introduced in second version of java i.e. **JDK 2.0** | IdentityHashMap was introduced in fourth version of java i.e. **JDK 4.0** |
| 4 | *Program* | Program 1 shows > *comparing keys (and values) performs object-equality in place of reference-equality . In an HashMap, two keys k1 and k2 are equal if and only if* **(k1==null ? k2==null : k1.equals(k2)).** | Program 2 shows >  *comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if* **(k1==k2).** |
| 5 | overridden equals() and hashCode() method call? | [*overridden equals() and hashCode() method*](http://www.javamadesoeasy.com/2015/02/override-equals-and-hashcode-method.html)are called when put, get methods are called in ***HashMap***.  As shown in Program 3. | *overridden equals() and hashCode() method* are not called when put, get methods are called in ***IdentityHashMap***.  *Because IdentityHashMap implements equals() and hashCode() method by itself and checks for reference-equality of keys.*  As shown in Program 4. |
| 6 | Application - can maintain *proxy object* | HashMap cannot be used to maintain *proxy object.* | IdentityHashMap can be used to maintain *proxy objects*. For example, we might need to maintain proxy object for each object debugged in the program. |

For more detail read : [**HashMap vs IdentityHashMap - Similarity and Differences with program in java**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-identityhashmap-similarity.html)

**Collection interview Question 20. What is WeakHashMap in java?**

**Answer**.  Another tricky collection framework interview question for experienced developers in java.

java.util.[WeakHashMap](http://www.javamadesoeasy.com/2015/04/weakhashmap-in-java.html) is hash table based implementation of the Map interface, with *weak keys*.

An entry in a WeakHashMap will be automatically removed by garbage collector when its key is no longer in ordinary use. Mapping for a given key will not prevent the key from being discarded by the garbage collector, (i.e. made finalizable, finalized, and then reclaimed). When a key has been discarded its entry is removed from the map in java.

java.util.**WeakHashMap** is implementation of the java.util.[**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) interface in java.

*The behavior of the java.util.WeakHashMap class depends upon garbage collector*

The behavior of the WeakHashMap class depends upon garbage collector in java. Because the garbage collector may discard keys at any time, in WeakHashMap it may look like some unknown thread is silently removing entries. Even if you synchronize WeakHashMap instance and invoke none of its methods,

* it is possible for the **size** method to return smaller values over time,
* for **isEmpty** method to return false and then true,
* for **containsKey** method to return true and later false for a given key,
* for **get** method to return a value for a given key but later return null,
* for **put** method to return null, and
* for **remove** method to return false for a key that previously existed in the WeakHashMap.

Each key object in a WeakHashMap is stored indirectly as the referent of a weak reference. Therefore a key will be removed automatically only after the weak references to it, both inside and outside of the map, have been cleared by the garbage collector.

# Hashmap vs WeakHashMap in Java

[**2.7**](https://www.geeksforgeeks.org/easy/)

[**HashMap**](https://www.geeksforgeeks.org/hashmap-treemap-java/)

Java.util.HashMap class is a Hashing based implementation. In HashMap, we have a key and a value pair.  
Even though the object is specified as key in hashmap, it does not have any reference and it is **not**eligible for garbage collection if it is associated with HashMap i.e. HashMap dominates over Garbage Collector.

|  |
| --- |
| // Java program to illustrate  // Hashmap  import java.util.\*;  class HashMapDemo  {      public static void main(String args[])throws Exception      {          HashMap m = new HashMap();          Demo d = new Demo();            // puts an entry into HashMap          m.put(d," Hi ");            System.out.println(m);          d = null;            // garbage collector is called          System.gc();            //thread sleeps for 4 sec          Thread.sleep(4000);            System.out.println(m);          }      }      class Demo      {          public String toString()          {              return "demo";          }            // finalize method          public void finalize()          {              System.out.println("Finalize method is called");          }  } |

Run on IDE

Output:

{demo=Hi}

{demo=Hi}

**WeakHashMap**

WeakHashMap is an implementation of the Map interface. WeakHashMap is almost same as HashMap except in case of WeakHashMap, if object is specified as key doesn’t contain any references- it is eligible for garbage collection even though it is associated with WeakHashMap. i.e Garbage Collector dominates over WeakHashMap.

|  |
| --- |
| // Java program to illustrate  // WeakHashmap  import java.util.\*;  class WeakHashMapDemo  {      public static void main(String args[])throws Exception      {          WeakHashMap m = new WeakHashMap();          Demo d = new Demo();            // puts an entry into WeakHashMap          m.put(d," Hi ");          System.out.println(m);            d = null;            // garbage collector is called          System.gc();            // thread sleeps for 4 sec          Thread.sleep(4000); .            System.out.println(m);      }  }    class Demo  {      public String toString()      {          return "demo";      }        // finalize method      public void finalize()      {          System.out.println("finalize method is called");      }  } |

Run on IDE

Output:

{demo = Hi}

finalize method is called

{ }

**Some more important differences between Hashmap and WeakHashmap:**

1. [**Strong vs Weak References**](https://www.geeksforgeeks.org/types-references-java/): Weak Reference Objects are not the default type/class of Reference Object and they should be explicitly specified while using them. This type of reference is used in WeakHashMap to reference the entry objects.  
   Strong References: This is the default type/class of Reference Object. Any object which has an active strong reference are not eligible for garbage collection. In HashMap, key objects have strong reference.
2. **Role of Garbage Collector:** Garbage Collected : In HashMap , entry object(entry object stores key-value pairs) is not eligible for garbage collection i.e Hashmap is dominant over Garbage Collector.  
   In WeakHashmap, wWhen a key is discarded then its entry is automatically removed from the map , in other words, garbage collected.
3. [**Clone method**](https://www.geeksforgeeks.org/clone-method-in-java-2/)**Implementation:**HashMap implements Cloneable interface .  
   WeakHashMap does not implement Cloneable interface , it only implements Map interface. Hence , there is no clone() method in the WeakHashMap class.

This article is contributed by **Rishabh Patel**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

**Collection interview Question 21. What is EnumSet in java?**

**Answer**. Freshers must know about EnumMap in java.

A java.util.[EnumSet](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) is specialized [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) implementation for use with enum types in java.

EnumSet all elements comes from a single enum type that is specified when the set is created in java.

*Order of elements in EnumSet in java*

The java.util.EnumSet maintains ***natural order*** (the order in which the enum constants are declared) of elements in java.

*Iterator on EnumSet in java*

The iterator returned by the iterator method traverses the elements in their ***natural order*** (the order in which the enum constants are declared).

iterator never throw ConcurrentModificationException and it may or may not show the effects of any modifications to the set that occur while the iteration is in progress.

*Null elements in EnumSet in java*

Null elements are not allowed in EnumSet in java. Attempts to insert a null element will throw NullPointerException in java.

**Collection interview Question 22. What is EnumMap in java?**

**Answer**. Freshers must be able to answer this collection framework interview question in java. A java.util.[EnumMap](http://www.javamadesoeasy.com/2015/04/enummap-in-java-with-program.html) is specialized [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) implementation for use with enum type keys.

EnumMap all keys comes from a single enum type that is specified when the set is created in java.

*Order of keys in EnumMap in java*

The EnumMap maintains ***natural order*** (the order in which the enum constants are declared) of keys in java.

*Iterator on EnumMap in java*

The iterator returned by the iterator method in EnumMap traverses the elements in their ***natural order* of keys**(the order in which the enum constants are declared).

iterator never throw ConcurrentModificationException and it may or may not show the effects of any modifications to the map that occur while the iteration is in progress in java.

*Null allowed in EnumMap in java?*

**Null keys are not allowed** in EnumMap. Attempts to insert a null key will throw NullPointerException.

**But, Null values are allowed** in EnumMap in java.

# Differences between TreeMap, HashMap and LinkedHashMap in Java

[**3.6**](https://www.geeksforgeeks.org/medium/)

Prerequisite : [HashMap and TreeMap in Java](https://www.geeksforgeeks.org/hashmap-treemap-java/)

**TreeMap, HashMap and LinkedHashMap: What’s Similar?**

**TreeMap, HashMap and LinkedHashMap: What’s Similar?**

* All offer **a key->value** map and a way to iterate through the keys. The most important distinction between these classes is the time guarantees and the ordering of the keys.
* All three classes HashMap, TreeMap and LinkedHashMap implements [**java.util.Map**](https://www.geeksforgeeks.org/map-interface-java-examples/) interface, and represents mapping from unique key to values.

**Key Points**

1. **HashMap:** HashMap offers **0(1)** lookup and insertion. If you iterate through the keys, though, the ordering of the keys is essentially arbitrary. It is implemented by an array of linked lists.  
   **Syntax:**
2. **public class HashMap extends AbstractMap**

**implements Map,Cloneable, Serializable**

* + A HashMap contains values based on the key.
  + It contains only unique elements.
  + It may have one null key and multiple null values.
  + It maintains **no order**.

1. **LinkedHashMap:**LinkedHashMap offers **0(1)** lookup and insertion. Keys are ordered by their insertion order. It is implemented by doubly-linked buckets.  
   **Syntax:**
2. **public class LinkedHashMap extends HashMap**

**0implements Map**

* + A LinkedHashMap contains values based on the key.
  + It contains only unique elements.
  + It may have one null key and multiple null values.
  + It is same as HashMap instead **maintains insertion order**.

1. **TreeMap:** TreeMap offers **O(log N)** lookup and insertion. Keys are ordered, so if you need to iterate through the keys in sorted order, you can. This means that keys must implement the Comparable interface. TreeMap is implemented by a Red-Black Tree.  
   **Syntax:**
2. **public class TreeMap extends AbstractMap implements**

**NavigableMap, Cloneable, Serializable**

* + A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
  + It contains only unique elements.
  + It cannot have null key but can have multiple null values.
  + It is same as HashMap instead **maintains ascending order(Sorted using the natural order of its key**).

1. **Hashtable:**“Hashtable” is the generic name for hash-based maps.  
   **Syntax:**
2. **public class Hashtable extends Dictionary implements**

**Map, Cloneable, Serializable**

* + A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
  + It contains only unique elements.
  + It may have not have any null key or value.
  + It is synchronized.
  + It is a legacy class.
* HashMap
* LinkedHashMap
* TreeMap

|  |
| --- |
| // Java program to print ordering  // of all elements using HashMap  import java.util.\*;  import java.lang.\*;  import java.io.\*;  class Main  {      // This function prints ordering of all elements      static void insertAndPrint(AbstractMap<Integer, String> map)      {          int[] array= {1, -1, 0, 2,-2};          for (int x: array)          {              map.put(x, Integer.toString(x));          }          for (int k: map.keySet())          {              System.out.print(k + ", ");          }      }        // Driver method to test above method      public static void main (String[] args)      {          HashMap<Integer, String> map = new HashMap<Integer, String>();          insertAndPrint(map);      }  } |

Run on IDE

Output of HashMap:

-1, 0, 1, -2, 2,

// ordering of the keys is essentially arbitrary (any ordering)

Output of LinkedHashMap:

1, -1, 0, 2, -2,

// Keys are ordered by their insertion order

Output of TreeMap:

-2, -1, 0, 1, 2,

// Keys are in sorted order

**Comparison Table**

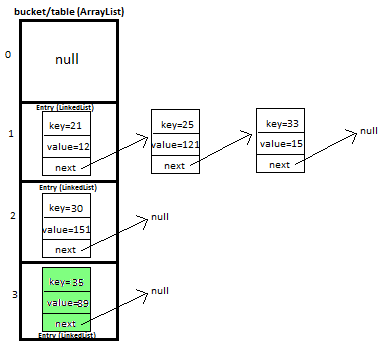


**Real Life Applications**

1. Suppose you were creating a mapping of names to Person objects. You might want to periodically output the people in alphabetical order by name. A TreeMap lets you do this.
2. A TreeMap also offers a way to, given a name, output the next 10 people. This could be useful for a “More”function in many applications.
3. A LinkedHashMap is useful whenever you need the ordering of keys to match the ordering of insertion. This might be useful in a caching situation, when you want to delete the oldest item.
4. Generally, unless there is a reason not to, you would use HashMap. That is, if you need to get the keys back in insertion order, then use LinkedHashMap. If you need to get the keys back in their true/natural order, then use TreeMap. Otherwise, HashMap is probably best. It is typically faster and requires less overhead.

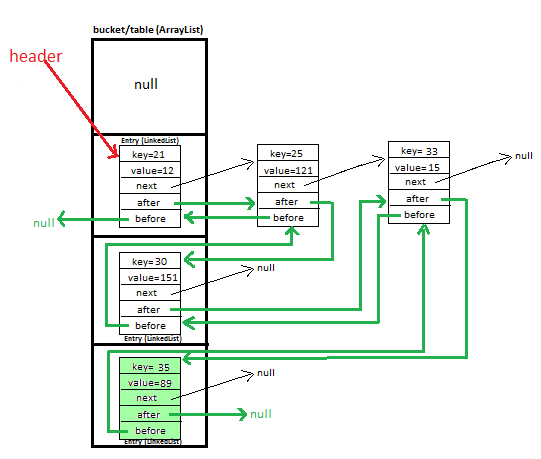
**Collection interview Question 23. How to implement own/custom HashMap in java? Or How HashMap works in java?**

**Answer**.

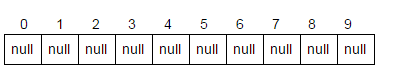
[HashMap Custom implementation/ HashMap works in java](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html)

**Collection interview Question 24. How to implement own LinkedHashMap in java? Or LinkedHashMap works in java?**

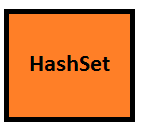
**Answer**.

[LinkedHashMap Custom implementation/How LinkedHashMap works in java](http://javamadesoeasy.com/2015/02/linkedhashmap-custom-implementation.html)

**Collection interview Question 25. How to implement own ArrayList in java?Or How ArrayList works in java ?**

**Answer**. [ArrayList custom implementation /  How ArrayList works in java](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)

**Collection interview Question 26. How to implement own HashSet in java? Or How HashSet works in java ?**

**Answer**. [****Set Custom implementation/ Or How HashSet works in java](http://javamadesoeasy.com/2015/02/set-custom-implementation.html)

**Collection interview Question 27. How to implement own LinkedHashSet in java? Or How LinkedHashSet works in java ?**

**Answer**. [LinkedHashSet Custom implementation/ How LinkedHashSet works in java](http://javamadesoeasy.com/2015/02/linkedhashset-custom-implementation.html)

**Collection interview Question 28. What do you mean by fail-fast and fast-safe? What is ConcurrentModificationException?**

**Answer**.

Iterator returned by few Collection framework Classesare **fail-fast,** means any structural modification made to these classes during iteration will throw ConcurrentModificationException.

Some important classes whose returned iterator is **fail-fast >**

* [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)
* [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)
* [**vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)
* [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)

Iterator returned by few Collection framework Classes are **fail-safe,** means any structural modification made to these classes during iteration won’t throw any Exception.

Some important classes whose returned iterator is **fail-safe >**

### [CopyOnWriteArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)

### [CopyOnWriteArraySet](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)

### [ConcurrentSkipListSet](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)

### For more detail read : [ConcurrentModificationException, Fail-fast and Fail-safe in detail in java](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)

**Collection interview Question 29. What are different ways of iterating over elements in List?**

**Answer**.

*Creating* [*ArrayList*](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) *and add element.*

|  |
| --- |
| List<String> arrayList=**new** ArrayList<String>();  arrayList.add("javaMadeSoEasy"); |

1. *Iterate over elements in ArrayList using* ***iterator****()*

iterator() method returns iterator to iterate over elements in ArrayList.

|  |
| --- |
| Iterator<String> iterator=arrayList.iterator();  **while**(iterator.hasNext()){            System.*out*.println(iterator.next());     } |

*iterator returned by ArrayList is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*.*

1. *Iterate over elements in ArrayList using* ***listIterator****()*

|  |
| --- |
| ListIterator<String> listIterator=arrayList.listIterator(); |

***ListIterator returned by ArrayList is also fail fast***.

1. *Iterate over elements in list using* ***enumeration***

|  |
| --- |
| Enumeration<String> listEnum=Collections.*enumeration*(arrayList);  **while**(listEnum.hasMoreElements()){        System.*out*.println(listEnum.nextElement());     } |

enumeration is also fail-fast.

1. *Iterate over elements in list using* ***enhanced for loop***

|  |
| --- |
| **for** (String string : arrayList) {               System.*out*.println(string);        } |

**enhanced for loop** is also fail-fast.

**Collection interview Question 30. What are different ways of iterating over elements in Set?**

**Answer**. *Creating* [*HashSet*](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *and add element.*

|  |
| --- |
| Set<String> hashSet=**new** HashSet<String>();  hashSet.add("javaMadeSoEasy"); |

1. *Iterate over elements in HashSet using iterator()*

iterator() method returns iterator to iterate over elements in HashSet.

|  |
| --- |
| Iterator<String> iterator=hashSet.iterator();  **while**(iterator.hasNext()){            System.*out*.println(iterator.next());     } |

*iterator returned by HashSet is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*.*

1. *Iterate over elements in Set using enumeration*

|  |
| --- |
| Enumeration<String> listEnum=Collections.*enumeration*(set);  **while**(listEnum.hasMoreElements()){        System.*out*.println(listEnum.nextElement());     } |

enumeration is also fail-fast.

1. *Iterate over elements in Set using* ***enhanced for loop***

|  |
| --- |
| **for** (String string : set) {               System.*out*.println(string);        } |

**enhanced for loop** is also fail-fast.

**Collection interview Question 31. What are different ways of iterating over keys, values and entry in Map?**

**Answer**. *Create and put key-value pairs in* [*HashMap*](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) *>*

|  |
| --- |
| Map<Integer,String> hashMap=**new** HashMap<Integer,String>();        hashMap.put(11, "javaMadeSoEasy");  hashMap.put(21, "bmw");  hashMap.put(31, "ferrari"); |

1. Iterate over keys -

***hashMap.keySet().iterator()*** method returns iterator to iterate over keys in HashMap.

|  |
| --- |
| Iterator<Integer> keyIterator=hashMap.keySet().iterator();  **while**(keyIterator.hasNext()){   System.*out*.println(keyIterator.next());  }  /\*OUTPUT  21  11  31  \*/ |

1. Iterate over values -

***hashMap.values().iterator()*** method returns iterator to iterate over keys in HashMap.

|  |
| --- |
| Iterator<String> valueIterator=hashMap.values().iterator();  **while**(valueIterator.hasNext()){   System.*out*.println(valueIterator.next());  }  /\*OUTPUT  javaMadeSoEasy  audi  ferrari  \*/ |

*iterator returned is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*..*

1. Iterate over Entry-

***hashMap.entrySet().iterator()*** method returns iterator to iterate over keys in HashMap.

|  |
| --- |
| Iterator<Entry<Integer, String>> entryIterator=hashMap.entrySet().iterator();  **while**(entryIterator.hasNext()){     System.*out*.println(entryIterator.next());  }  /\*OUTPUT  21=javaMadeSoEasy  11=audi  31=ferrari  \*/ |

*iterator returned is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*..*

**Collection interview Question 32. What is difference between** [**Comparable and Comparator**](http://www.javamadesoeasy.com/2015/04/comparable-vs-comparator-differences.html)**? How can you sort List?**

**Answer**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***Comparable*** | ***Comparator*** |
| 1 | Comparing instances of class | Comparable is used to compare instances of same class | Comparator can be used to compare instances of same or different classes. |
| **2** | **sorting order** | Comparable can be implemented by class which need to define a **natural ordering for its objects.**  **Example** - String, Integer, Long , [Date](http://www.javamadesoeasy.com/2015/07/creating-date-in-java-using-calendar.html) and all other wrapper classes implements Comparable. | Comparator is implemented when one wants a **different sorting order** and define custom way of comparing two instances. |
| 3 | Changes to class | For using Comparable, original Class must implement it.    **Example-**  **class** Employee **implements Comparable<Employee>**    For using Comparable, Employee Class must implement it, no other class can implement it.  As used in **Program 1** | Class itself can implement Comparator  or  any other class can implement Comparator. Hence avoiding modification to original class.  **Example-**  **class ComparatorName implements Comparator<Employee>**  **class ComparatorId implements Comparator<Employee>**  In above example modifications were made to **ComparatorName** and **ComparatorId.** Hence avoiding modification to Employee class.  As used in **Program 4** |
| 4 | Sorting on basis on one or many criteria | Provides sorting only on **one** criteria, **because** Comparable can be implemented by original class only. | We can use Comparator to sort class on **many** criterias **because** class itself or any other class can implement Comparator. |
| 5 | Method | compareTo method  **@Override**  **public int compareTo(Employee obj) {**  **//sort Employee on basis of name(ascending order)**  **return this.name.compareTo(obj.name);**  **}**  Method compares **this** with **obj** object and returns a integer.   * positive – **this** is **greater** than **obj** * zero – **this** is **equal** to **obj** * negative – **this** is **less** than **obj**     As used in **Program 1** | compare method  **@Override**  **public int compare(Employee obj1, Employee obj2) {**  **//sort Employee on basis of name(ascending order)**  **return obj1.name.compareTo(obj2.name);**  **}**    Method compares **obj1** with **obj2** object and returns a integer.   * positive – **obj1** is **greater** than **obj2** * zero – **obj1** is **equal** to **obj2** * negative – **obj1** is **less** than **obj2**     As used in **Program 3** |
| 6 | Package | **java.lang**  **java.lang** package is automatically imported by every program in java.  Hence, we need to write explicit statement for importing java.lang.Comparable. | **java.util**  We need to write explicit import statement -  **import** java.util.Comparator |
| 7 | Using **Collections.sort** | Let's say we wanna sort list of Employee,  **Collections.sort(**list**)** uses Comparable interface for sorting class.  As used in Program 1 | Let's say we wanna sort list of Employee,  **Collections.*sort*(list,new ComparatorName());**  uses Comparator interface for sorting class.  As used in Program 5 |

# Comparable vs Comparator in Java

[**2.8**](https://www.geeksforgeeks.org/easy/)

Java provides two interfaces to sort objects using data members of the class:

1. Comparable
2. Comparator

**Using Comparable Interface**

A comparable object is capable of comparing itself with another object. The class itself must implements the **java.lang.Comparable** interface to compare its instances.

Consider a Movie class that has members like, rating, name, year. Suppose we wish to sort a list of Movies based on year of release. We can implement the Comparable interface with the Movie class, and we override the method compareTo() of Comparable interface.

|  |
| --- |
| // A Java program to demonstrate use of Comparable  import java.io.\*;  import java.util.\*;    // A class 'Movie' that implements Comparable  class Movie implements Comparable<Movie>  {      private double rating;      private String name;      private int year;        // Used to sort movies by year      public int compareTo(Movie m)      {          return this.year - m.year;      }        // Constructor      public Movie(String nm, double rt, int yr)      {          this.name = nm;          this.rating = rt;          this.year = yr;      }        // Getter methods for accessing private data      public double getRating() { return rating; }      public String getName()   {  return name; }      public int getYear()      {  return year;  }  }    // Driver class  class Main  {      public static void main(String[] args)      {          ArrayList<Movie> list = new ArrayList<Movie>();          list.add(new Movie("Force Awakens", 8.3, 2015));          list.add(new Movie("Star Wars", 8.7, 1977));          list.add(new Movie("Empire Strikes Back", 8.8, 1980));          list.add(new Movie("Return of the Jedi", 8.4, 1983));            Collections.sort(list);            System.out.println("Movies after sorting : ");          for (Movie movie: list)          {              System.out.println(movie.getName() + " " +                                 movie.getRating() + " " +                                 movie.getYear());          }      }  } |

Run on IDE

Output:

Movies after sorting :

Star Wars 8.7 1977

Empire Strikes Back 8.8 1980

Return of the Jedi 8.4 1983

Force Awakens 8.3 2015

Now, suppose we want sort movies by their rating and names also. When we make a collection element comparable(by having it implement Comparable), we get only one chance to implement the compareTo() method. The solution is using [Comparator.](https://www.geeksforgeeks.org/comparator-interface-java/)

**Using Comparator**

Unlike Comparable, Comparator is external to the element type we are comparing. It’s a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.

Collections class has a second sort() method and it takes Comparator. The sort() method invokes the compare() to sort objects.

To compare movies by Rating, we need to do 3 things :

1. Create a class that implements Comparator (and thus the compare() method that does the work previously done by compareTo()).
2. Make an instance of the Comparator class.
3. Call the overloaded sort() method, giving it both the list and the instance of the class that implements Comparator.

|  |
| --- |
| //A Java program to demonstrate Comparator interface  import java.io.\*;  import java.util.\*;    // A class 'Movie' that implements Comparable  class Movie implements Comparable<Movie>  {      private double rating;      private String name;      private int year;        // Used to sort movies by year      public int compareTo(Movie m)      {          return this.year - m.year;      }        // Constructor      public Movie(String nm, double rt, int yr)      {          this.name = nm;          this.rating = rt;          this.year = yr;      }        // Getter methods for accessing private data      public double getRating() { return rating; }      public String getName()   {  return name; }      public int getYear()      {  return year;  }  }    // Class to compare Movies by ratings  class RatingCompare implements Comparator<Movie>  {      public int compare(Movie m1, Movie m2)      {          if (m1.getRating() < m2.getRating()) return -1;          if (m1.getRating() > m2.getRating()) return 1;          else return 0;      }  }    // Class to compare Movies by name  class NameCompare implements Comparator<Movie>  {      public int compare(Movie m1, Movie m2)      {          return m1.getName().compareTo(m2.getName());      }  }    // Driver class  class Main  {      public static void main(String[] args)      {          ArrayList<Movie> list = new ArrayList<Movie>();          list.add(new Movie("Force Awakens", 8.3, 2015));          list.add(new Movie("Star Wars", 8.7, 1977));          list.add(new Movie("Empire Strikes Back", 8.8, 1980));          list.add(new Movie("Return of the Jedi", 8.4, 1983));            // Sort by rating : (1) Create an object of ratingCompare          //                  (2) Call Collections.sort          //                  (3) Print Sorted list          System.out.println("Sorted by rating");          RatingCompare ratingCompare = new RatingCompare();          Collections.sort(list, ratingCompare);          for (Movie movie: list)              System.out.println(movie.getRating() + " " +                                 movie.getName() + " " +                                 movie.getYear());              // Call overloaded sort method with RatingCompare          // (Same three steps as above)          System.out.println("\nSorted by name");          NameCompare nameCompare = new NameCompare();          Collections.sort(list, nameCompare);          for (Movie movie: list)              System.out.println(movie.getName() + " " +                                 movie.getRating() + " " +                                 movie.getYear());            // Uses Comparable to sort by year          System.out.println("\nSorted by year");          Collections.sort(list);          for (Movie movie: list)              System.out.println(movie.getYear() + " " +                                 movie.getRating() + " " +                                 movie.getName()+" ");      }  } |

Run on IDE

Output :

Sorted by rating

8.3 Force Awakens 2015

8.4 Return of the Jedi 1983

8.7 Star Wars 1977

8.8 Empire Strikes Back 1980

Sorted by name

Empire Strikes Back 8.8 1980

Force Awakens 8.3 2015

Return of the Jedi 8.4 1983

Star Wars 8.7 1977

Sorted by year

1977 8.7 Star Wars

1980 8.8 Empire Strikes Back

1983 8.4 Return of the Jedi

2015 8.3 Force Awakens

* Comparable is meant for objects with natural ordering which means the object itself must know how it is to be ordered. For example Roll Numbers of students. Whereas, Comparator interface sorting is done through a separate class.
* Logically, Comparable interface compares “this” reference with the object specified and Comparator in Java compares two different class objects provided.
* If any class implements Comparable interface in Java then collection of that object either List or Array can be sorted automatically by using Collections.sort() or Arrays.sort() method and objects will be sorted based on there natural order defined by CompareTo method.

***To summarize, if sorting of objects needs to be based on natural order then use Comparable whereas if you sorting needs to be done on attributes of different objects, then use Comparator in Java.***

**Collection interview Question 33. How sort method of Collections class works internally?**

**Answer**. *Collections.sort internally calls Arrays.sort,*

*Arrays.Sort() internally uses* [*Merge Sort*](http://javamadesoeasy.blogspot.in/2015/01/merge-sort.html)*.*

If number of elements is less than 7 then [Insertion Sort](http://www.javamadesoeasy.com/2015/01/insertion-sort.html) is used rather than [*Merge Sort*](http://javamadesoeasy.blogspot.in/2015/01/merge-sort.html). (because in case elements are less than 7 it offers better time complexity)

**Collection interview Question 34. How can you sort given HashMap on basis of keys?**

**Answer**.

### Please Read : [Sort Map by key in Ascending and descending order by implementing Comparator interface and overriding its compare method and using TreeMap](http://www.javamadesoeasy.com/2015/04/sort-map-by-key-in-ascending-and.html)

**Collection interview Question 35. How can you sort given HashMap on basis of values?**

**Answer**.

### Please Read : [Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method](http://www.javamadesoeasy.com/2015/04/sort-map-by-value-in-ascending-and.html)

**Collection interview Question 36. In what all possible ways you can sort a given Set?**

**Answer**.

**Please Read :** [**Sort Set by using TreeSet and by implementing Comparator and Comparable interface**](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

**Collection interview Question 37. How you can sort arrays? And how Comparator of superclass can be used by subclasses?**

**Answer**.

**Please Read :** [**Arrays.sort to sort arrays by implementing Comparator and how Comparator of superclass can be used by subclasses**](http://www.javamadesoeasy.com/2015/04/arrayssort-to-sort-arrays-by.html)

**Collection interview Question 38. What are differences between** [**ArrayList vs CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)**?**

**Answer**.

***Differences*** *between java.util.*[***ArrayList***](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)*and java.util.concurrent.****CopyOnWriteArrayList****in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | **java.util.concurrent. CopyOnWriteArrayList** |
| 1 | synchronization | ArrayList is not **synchronized**  (because 2 threads on same ArrayList object can access it at same time).  I have created **program** to show see consequence of using ArrayList in multithreading environment.  In the program i will implement our own arrayList. | **CopyOnWriteArrayList**is **synchronized**  (because 2 threads on same CopyOnWriteArrayList object cannot access it at same time). |
| 2 | Iterator and listIterator | Iterator and listIterator returned by ArrayList are [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to ArrayList during iteration using Iterator or listIterator will throw ConcurrentModificationException in java.  As shown in Program 1 below. | Iterator and listIterator returned by CopyOnWriteArrayList are **Fail-safe** in java.  As shown in Program 2 below. |
| 3 | Enumeration is fail-fast | **Enumeration** returned by ArrayList is **fail-fast**, means any structural modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException.  As shown in Program 1 below. | **Enumeration** returned by CopyOnWriteArrayList is **fail-safe.**    As shown in Program 2 below. |
| 4 | Iterate using **enhanced for loop** | Iteration done on ArrayList using **enhanced for loop** is **Fail-fast,** means any structural modification made to ArrayList during iteration using **enhanced for loop** will throw ConcurrentModificationException.  As shown in Program 1 below. | Iteration done on CopyOnWriteArrayList using **enhanced for loop** is **Fail-safe.**  As shown in Program 2 below. |
| 5 | Performance | ArrayList is not synchronized, hence its operations are **faster** as compared to CopyOnWriteArrayList. | CopyOnWriteArrayList is synchronized, hence its operations are **slower** as compared to ArrayList. |
| 6 | AbstractList | ArrayList extends AbstractList (abstract class) which provides implementation to  List interface to minimize the effort required to implement this interface backed by RandomAccess interface. | CopyOnWriteArrayList does not extends AbstractList, though CopyOnWriteArrayList also implements RandomAccess interface. |
| 7 | Introduced in which java version | ArrayList was introduced in second version of java (1.2) i.e. **JDK 2.0** | CopyOnWriteArrayList was introduced in fifth version of java (1.5) i.e. **JDK 5.0** |
| 8 | Package | java.util | java.util.**concurrent** |

**For more detail read :**[ArrayList vs CopyOnWriteArrayList - Similarity and Differences with program](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)

**Collection interview Question 39. What are differences between** [**HashSet vs CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)**?**

**Answer**.

***Differences*** *between java.util.*[***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*and java.util.concurrent.****CopyOnWriteArraySet*** *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.HashSet*** | **java.util.concurrent. CopyOnWriteArraySet** |
| 1 | synchronization | HashSet is not **synchronized**  (because 2 threads on same HashSet object can access it at same time) in java. | **CopyOnWriteArraySet**is **synchronized**  (because 2 threads on same CopyOnWriteArraySet object cannot access it at same time) in java. |
| 2 | Iterator | Iterator returned by HashSet is [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to HashSet during iteration using Iterator will throw ConcurrentModificationException in java.  As shown in Program 1 below. | Iterator returned by **CopyOnWriteArraySet** is **Fail-safe** in java.  As shown in Program 2 below. |
| 3 | Enumeration is fail-fast | **Enumeration** returned by HashSet is **fail-fast**, means any structural modification made to HashSet during iteration using Enumeration will throw ConcurrentModificationException.  As shown in Program 1 below. | **Enumeration** returned by CopyOnWriteArraySet is **fail-safe.**  As shown in Program 2 below. |
| 4 | Iterate using **enhanced for loop** | Iteration done on HashSet using **enhanced for loop** is **Fail-fast,** means any structural modification made to HashSet during iteration using **enhanced for loop** will throw ConcurrentModificationException.  As shown in Program 1 below. | Iteration done on CopyOnWriteArraySet using **enhanced for loop** is **Fail-safe.**    As shown in Program 2 below. |
| 5 | Performance | HashSet is not synchronized, hence its operations are **faster** as compared to CopyOnWriteArraySet. | CopyOnWriteArraySet is synchronized, hence its operations are **slower** as compared to HashSet. |
| 6 | Introduced in which java version | HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | CopyOnWriteArraySet  was introduced in fifth version of java (1.5) i.e. **JDK 5.0** |
| 7 | Package | java.util | java.util.**concurrent** |

### *For more detail read :* [HashSet vs CopyOnWriteArraySet - Similarity and Differences with program](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)

**Collection interview Question 40. What are differences between** [**TreeSet vs ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)**?**

**Answer**.

***Differences*** *between java.util.*[***TreeSet***](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) *and java.util.concurrent.****ConcurrentSkipListSet*** *in java**>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.TreeSet*** | **java.util.concurrent. ConcurrentSkipListSet** |
| 1 | synchronization | TreeSet is not **synchronized**  (because 2 threads on same TreeSet object can access it at same time) in java. | **ConcurrentSkipListSet**is **synchronized**  (because 2 threads on same ConcurrentSkipListSet object cannot access it at same time) in java. |
| 2 | Iterator | Iterator returned by TreeSet is [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to TreeSet during iteration using Iterator will throw ConcurrentModificationException in java.  As shown in Program 1 below. | Iterator returned by **ConcurrentSkipListSet** is **Fail-safe** in java.    As shown in Program 2 below. |
| 3 | Enumeration is fail-fast | **Enumeration** returned by TreeSet is **fail-fast**, means any structural modification made to TreeSet during iteration using Enumeration will throw ConcurrentModificationException.  As shown in Program 1 below. | **Enumeration** returned by ConcurrentSkipListSet is **fail-safe.**    As shown in Program 2 below. |
| 4 | Iterate using **enhanced for loop** | Iteration done on TreeSet using **enhanced for loop** is **Fail-fast,** means any structural modification made to TreeSet during iteration using **enhanced for loop** will throw ConcurrentModificationException.  As shown in Program 1 below. | Iteration done on ConcurrentSkipListSet using **enhanced for loop** is **Fail-safe.**  As shown in Program 2 below. |
| 5 | Performance | TreeSet is not synchronized, hence its operations are **faster** as compared to ConcurrentSkipListSet. | ConcurrentSkipListSet is synchronized, hence its operations are **slower** as compared to TreeSet. |
| 6 | Introduced in which java version | TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | ConcurrentSkipListSet was introduced in sixth version of java (1.6) i.e. **JDK 6.0** |
| 7 | Package | java.util | java.util.**concurrent** |

### *For more detail read :* [TreeSet vs ConcurrentSkipListSet - Similarity and Differences with program](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)

**Collection interview Question 41. What are differences between** [**TreeMap vs ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)**?**

**Answer**.

***Differences*** *between java.util.*[***TreeMap***](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)*and java.util.concurrent.****ConcurrentSkipListMap*** *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.TreeMap*** | ***java.util.concurrent. ConcurrentSkipListMap*** |
| 1 | synchronization | TreeMap is **not synchronized**  (because 2 threads on same TreeMap object can access it at same time) in java. | ConcurrentSkipListMap is **synchronized** (because 2 threads on same ConcurrentSkipListMap object cannot access it at same time) in java. |
| 2 | Iterator | The iterators returned by the iterator() method of Map's “collection view methods" are [***fail-fast***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*>*   * *map.keySet().iterator()* * *map.values().iterator()* * *map.entrySet().iterator()*   all three iterators are ***fail-fast*,** means any structural modification made to TreeMap during iteration using any of 3 Iterator will throw ConcurrentModificationException.  As shown in Program 1 below. | The iterators returned by the iterator() method of Map's “collection view methods" are *fail-safe >*   * *map.keySet().iterator()* * *map.values().iterator()* * *map.entrySet().iterator()*   all three iterators are ***fail-safe.***    As shown in Program 2 below. |
| 3 | Performance | TreeMap is not synchronized, hence its operations are **faster** as compared to ConcurrentSkipListMap. | ConcurrentSkipListMap is synchronized, hence its operations are **slower** as compared to TreeMap. |
| 4 | Introduced inin which java version | TreeMap was introduced in second version of java i.e. **JDK 2.0** | ConcurrentSkipListMap was introduced in sixth version of java i.e. **JDK 6.0** |
| 5 | Package | java.util | java.util.**concurrent** |
| 6 | Implements which interface | Map  SortedMap  NavigableMap | Map  SortedMap  NavigableMap  ConcurrentNavigableMap |

### *For more detail read :* [TreeMap vs ConcurrentSkipListMap - Similarity and Differences with program](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)

**Collection interview Question 43. Can we use null element in TreeSet? Give reason?**

**Answer**. No, [TreeSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) does **not** allows to store **any null keys**.

Any attempt to add null throws runtimeException (NullPointerException).

TreeSet internally compares elements for sorting elements by natural order ([comparator may be used for sorting](http://www.javamadesoeasy.com/2015/04/program-to-sort-set-in-ascending-order_24.html), if defined at creation time) and null is not comparable, Any attempt to compare null with other object will throw NullPointerException.

**Collection interview Question 44. Can we use null key in TreeMap? Give reason?**

**Answer**. No, [TreeMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) **not allow to store null key.**

Any attempt to store null key throws runtimeException (NullPointerException).

TreeMap internally compares keys for sorting keys by natural order ([comparator may be used for sorting](http://www.javamadesoeasy.com/2015/04/program-to-sort-set-in-ascending-order_24.html), if defined at creation time)  and null is not comparable, Any attempt to compare null with other object will throw NullPointerException.

**Collection interview Question 45.  How ConcurrentHashMap works? Can 2 threads on same ConcurrentHashMap object access it concurrently?**

**Answer**. [*ConcurrentHashMap*](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently.

**Can threads read the segment locked by some other thread?**

Yes. When thread locks one segment for updation it does not block it for retrieval (done by get method) hence some other thread can read the segment (by get method), but it will be able to read the data before locking.

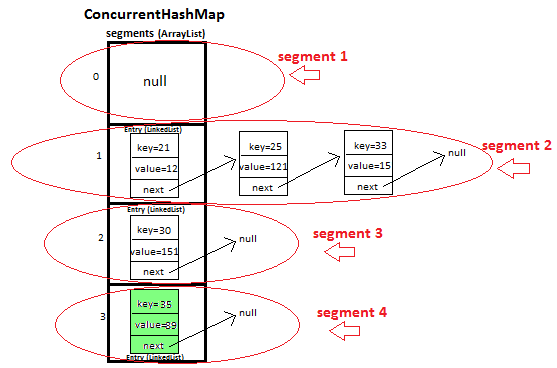
For operations such as putAll concurrent retrievals may reflect removal of only some entries.

For operations such as clear concurrent retrievals may reflect removal of only some entries.

***Segments*** *in ConcurrentHashMap with* ***diagram*** *>*

we have ConcurrentHashMap with **4 segments -**

(Diagram shows how **segments** are formed in ConcurrentHashMap)

**[](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)**

**Collection interview Question 46. Write a program to show consequence of using ArrayList in multithreading environment?**

**Answer.** Program to show [consequence of using ArrayList in multithreading environment in java](http://www.javamadesoeasy.com/2015/05/consequence-of-using-arraylist-in.html)

[dri](https://docs.google.com/document/d/10tUFPo2Xcw5JiwNR_bqIGEw52V6eUGP0KvE2nTHkb-U/edit) [blog](https://www.blogger.com/blogger.g?blogID=5056459490283781613#editor/target=post;postID=4105800325043770563;onPublishedMenu=allposts;onClosedMenu=allposts;postNum=0;src=link)

**Collection interview Question 47. Write a program to show advantage of using Vector in multithreading environment?**

**Answer.**  Program to show [advantage of using Vector in multithreading environment in java](http://www.javamadesoeasy.com/2015/05/advantage-of-using-vector-in.html)

**Collection interview Question 48. Mention properties of most frequently used Collection classes and Interfaces? Mention as many properties as much you can.**

**Answer**. This question is real test for experienced developers, this will test your in depth awareness of Collection classes and Interfaces. Answering this question in detail will really ensure your selection.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) | **Duplicate elements** | **insertion order** | **Sorted by natural order** | **synchronized** | **null elements** | **Iterator** |
| [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) | Yes | Yes |  |  | Yes | Iterator & listIterator  are  [Fail-fast](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) |
| [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) | Yes | Yes |  |  | Yes | Iterator & listIterator  are  Fail-fast |
| [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) | Yes | Yes |  | Yes | Yes | Iterator & listIterator  are  [**Fail-safe**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) |
|  |  |  |  |  |  |  |
| [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) | **Duplicate elements** | **insertion order** | **Sorted by natural order** | **synchronized** | **null elements** | **Iterator** |
| [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) |  |  |  |  | Yes | Fail-fast |
| [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) |  | Yes |  |  | Yes | Fail-fast |
| [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) |  |  | Yes |  | No | Fail-fast |
| [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) |  |  | Yes | Yes | No | **Fail-safe** |
|  |  |  |  |  |  |  |
| [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | **Duplicate Keys** | **insertion order of keys** | **Sorted by natural order of keys** | **synchronized** | **null keys or null values** | **Iterator**  **Map implementations returns 3 iterators >**  *map.keySet().iterator()*  *map.values().iterator()*  *map.entrySet().iterator()* |
| [**HashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) |  |  |  |  | one null key and many null values | All are Fail-fast |
| [**Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) |  |  |  | Yes | No | All are Fail-fast |
| [**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) |  |  |  | Yes | No | All are **Fail-safe** |
| [**TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) |  |  | Yes |  | Null key not allowed,  Allow many null values | All are Fail-fast |
| [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) |  |  | Yes | Yes | No | All are **Fail-safe** |

[**Collection - List, Set and Map all properties in tabular form**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

**Collection interview Question. 49 Which list class must be preferred in multithreading environment, considering performance constraint?**

**Answer**. [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)

**Collection interview Question 50. Which Set class must be preferred in multithreading environment, considering performance constraint?**

**Answer**. [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)(allows null and elements aren't sorted in natural order) **or** [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)(doesn’t allows null and elements are sorted in natural order)

**Select one depending on your requirement.**

**Collection interview Question 51. Which Map class must be preferred in multithreading environment, considering performance constraint?**

**Answer**. [**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)(keys aren't sorted in natural order) **or** [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)(keys are sorted in natural order)

**Select one depending on your requirement.**

**Collection interview Question 52. Let’s say you have to build dictionary and multiple users can add data in that dictionary? And you can use 2 Collection classes? Which Collection classes you will prefer and WHY?**

**Answer**. It’s very **important question** which test your **logical** reasoning and your ability to create robust applications in [multithreading](http://www.javamadesoeasy.com/2015/03/what-is-thread-in-java.html) environment.

We must use [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) and [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)  >

|  |
| --- |
| **ConcurrentSkipListMap<String, TreeSet<String>> myDictionary =**  **new ConcurrentSkipListMap<String, TreeSet<String>>();** |

Store words in [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) as key>

* keys are sorted in **natural order** (words will be sorted in natural order),
* **doesn’t allow null** keys (words can’t be null)
* **doesn’t allow duplicate** keys (words can’t be duplicate) and
* [synchronized](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html), so 2 threads won’t create synchronization problems (will take care of different uses adding words concurrently)

for storing meaning of word in dictionary we must use [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) as value in ConcurrentSkipListMap **because one word can have many meanings** >

* elements are sorted in **natural order** (meaning of word are sorted in natural order),
* **doesn’t allow null** elements (meaning of word can’t be null),
* **doesn’t allow duplicate** elements (meaning of word can’t be duplicate)

**Program for creating and using Java dictionary using Collection classes>**

|  |
| --- |
| **package** com.ankit.dictionary;  **import** java.util.TreeSet;  **import** java.util.concurrent.ConcurrentSkipListMap;  /\*\* \*/  **public** **class** MyDictionary {  **public** **static** **void** main(String[] args) {  **ConcurrentSkipListMap<String, TreeSet<String>> myDictionary =**  **new ConcurrentSkipListMap<String, TreeSet<String>>();**            TreeSet<String> innocentMeaning = **new** TreeSet<String>();            innocentMeaning.add("not responsible for an event yet suffering its consequences");            innocentMeaning.add("not guilty of a crime");            myDictionary.put("innocent", innocentMeaning);            TreeSet<String> appealingMeaning = **new** TreeSet<String>();            appealingMeaning.add("attractive");            appealingMeaning.add("expressing a desire for help");            myDictionary.put("appealing", appealingMeaning);            System.*out*.println(myDictionary);     }  }  /\* OUTPUT  {**appealing**=[attractive, expressing a desire for help], **innocent**=[not guilty of a crime, not responsible for an event yet suffering its consequences]}  \*/ |

**Collection interview Question 53. Why to use java.util.WeakHashMap map which is so inconsistent and unpredictable in behaviour?**

**Answer**. Let's say we have huge application which consists of lots n lots of object and may run short of memory at any time, we will like [garbage collector](http://www.javamadesoeasy.com/2015/09/how-garbage-collection-works-internally.html) to quickly discard less used key value pair to free up some memory. As, behavior of the WeakHashMap class depends upon garbage collector.

I believe discarding less used key-value is always going to a better option than running out of memory.

2. Sort by Value

Converts the Map into a List<Map>, sorts the List<Map> with a custom Comparator and put it into a new insertion order map – LinkedHashMap

Map ---> List<Map> ---> Collections.sort() --> List<Map> (Sorted) ---> LinkedHashMap

SortByValueExample1.java

package com.mkyong.test;

import java.util.\*;

public class SortByValueExample1 {

public static void main(String[] args) {

Map<String, Integer> unsortMap = new HashMap<String, Integer>();

unsortMap.put("z", 10);

unsortMap.put("b", 5);

unsortMap.put("a", 6);

unsortMap.put("c", 20);

unsortMap.put("d", 1);

unsortMap.put("e", 7);

unsortMap.put("y", 8);

unsortMap.put("n", 99);

unsortMap.put("j", 50);

unsortMap.put("m", 2);

unsortMap.put("f", 9);

System.out.println("Unsort Map......");

printMap(unsortMap);

System.out.println("\nSorted Map......By Value");

Map<String, Integer> sortedMap = sortByValue(unsortMap);

printMap(sortedMap);

}

private static Map<String, Integer> sortByValue(Map<String, Integer> unsortMap) {

// 1. Convert Map to List of Map

List<Map.Entry<String, Integer>> list =

new LinkedList<Map.Entry<String, Integer>>(unsortMap.entrySet());

// 2. Sort list with Collections.sort(), provide a custom Comparator

// Try switch the o1 o2 position for a different order

Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {

public int compare(Map.Entry<String, Integer> o1,

Map.Entry<String, Integer> o2) {

return (o1.getValue()).compareTo(o2.getValue());

}

});

// 3. Loop the sorted list and put it into a new insertion order Map LinkedHashMap

Map<String, Integer> sortedMap = new LinkedHashMap<String, Integer>();

for (Map.Entry<String, Integer> entry : list) {

sortedMap.put(entry.getKey(), entry.getValue());

}

/\*

//classic iterator example

for (Iterator<Map.Entry<String, Integer>> it = list.iterator(); it.hasNext(); ) {

Map.Entry<String, Integer> entry = it.next();

sortedMap.put(entry.getKey(), entry.getValue());

}\*/

return sortedMap;

}

public static <K, V> void printMap(Map<K, V> map) {

for (Map.Entry<K, V> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey()

+ " Value : " + entry.getValue());

}

}

}

Output

Unsort Map......

Key : a Value : 6

Key : b Value : 5

Key : c Value : 20

Key : d Value : 1

Key : e Value : 7

Key : f Value : 9

Key : y Value : 8

Key : z Value : 10

Key : j Value : 50

Key : m Value : 2

Key : n Value : 99

Sorted Map......By Value

Key : d Value : 1

Key : m Value : 2

Key : b Value : 5

Key : a Value : 6

Key : e Value : 7

Key : y Value : 8

Key : f Value : 9

Key : z Value : 10

Key : c Value : 20

Key : j Value : 50

Key : n Value : 99

2.2 Upgrade the above sortByValue() method to support generics.

public static <K, V extends Comparable<? super V>> Map<K, V> sortByValue(Map<K, V> unsortMap) {

List<Map.Entry<K, V>> list =

new LinkedList<Map.Entry<K, V>>(unsortMap.entrySet());

Collections.sort(list, new Comparator<Map.Entry<K, V>>() {

public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

return (o1.getValue()).compareTo(o2.getValue());

}

});

Map<K, V> result = new LinkedHashMap<K, V>();

for (Map.Entry<K, V> entry : list) {

result.put(entry.getKey(), entry.getValue());

}

return result;

}

*Update History*

How to count duplicated items in Java List

By [mkyong](https://www.mkyong.com/author/mkyong/) | November 7, 2012 | Viewed : 130,624 times +787 pv/w

A Java example to show you how to count the total number of duplicated entries in a List, using Collections.frequencyand Map.

CountDuplicatedList.java

package com.mkyong;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.TreeMap;

public class CountDuplicatedList {

public static void main(String[] args) {

List<String> list = new ArrayList<String>();

list.add("a");

list.add("b");

list.add("c");

list.add("d");

list.add("b");

list.add("c");

list.add("a");

list.add("a");

list.add("a");

System.out.println("\nExample 1 - Count 'a' with frequency");

System.out.println("a : " + Collections.frequency(list, "a"));

System.out.println("\nExample 2 - Count all with frequency");

Set<String> uniqueSet = new HashSet<String>(list);

for (String temp : uniqueSet) {

System.out.println(temp + ": " + Collections.frequency(list, temp));

}

System.out.println("\nExample 3 - Count all with Map");

Map<String, Integer> map = new HashMap<String, Integer>();

for (String temp : list) {

Integer count = map.get(temp);

map.put(temp, (count == null) ? 1 : count + 1);

}

printMap(map);

System.out.println("\nSorted Map");

Map<String, Integer> treeMap = new TreeMap<String, Integer>(map);

printMap(treeMap);

}

public static void printMap(Map<String, Integer> map){

for (Map.Entry<String, Integer> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey() + " Value : "

+ entry.getValue());

}

}

}

Output

Example 1 - Count 'a' with frequency

a : 4

Example 2 - Count all with frequency

d: 1

b: 2

c: 2

a: 4

Example 3 - Count all with Map

Key : d Value : 1

Key : b Value : 2

Key : c Value : 2

Key : a Value : 4

Sorted Map

Key : a Value : 4

Key : b Value : 2

Key : c Value : 2

Key : d Value : 1

Java – Check if key exists in HashMap

By [mkyong](https://www.mkyong.com/author/mkyong/) | September 5, 2015 | Updated : June 30, 2016 | Viewed : 38,200 times +487 pv/w

In Java, you can use Map.containsKey() to check if a key exists in a Map.

TestMap.java

package com.mkyong.examples;

import java.util.HashMap;

import java.util.Map;

public class TestMap {

public static void main(String[] args) {

Map<String, Integer> fruits = new HashMap<>();

fruits.put("apple", 1);

fruits.put("orange", 2);

fruits.put("banana", 3);

fruits.put("watermelon", null);

System.out.println("1. Is key 'apple' exists?");

if (fruits.containsKey("apple")) {

//key exists

System.out.println("yes! - " + fruits.get("apple"));

} else {

//key does not exists

System.out.println("no!");

}

System.out.println("\n2. Is key 'watermelon' exists?");

if (fruits.containsKey("watermelon")) {

System.out.println("yes! - " + fruits.get("watermelon"));

} else {

System.out.println("no!");

}

}

}

Output

1. Is key 'apple' exists?

yes! - 1

2. Is key 'watermelon' exists?

yes! - null

How to sort a Map in Java

By [mkyong](https://www.mkyong.com/author/mkyong/) | July 7, 2010 | Updated : August 12, 2016 | Viewed : 381,166 times +1,946 pv/w

Few Java examples to sort a Map by its keys or values.

**Note**  
If you are using Java 8, refer to this article – [How to use Stream APIs to sort a Map](http://www.mkyong.com/java8/java-8-how-to-sort-a-map/)

1. Sort by Key

1.1 Uses java.util.TreeMap, it will sort the Map by keys automatically.

SortByKeyExample1.java

package com.mkyong.test;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

public class SortByKeyExample1 {

public static void main(String[] args) {

Map<String, String> unsortMap = new HashMap<String, String>();

unsortMap.put("Z", "z");

unsortMap.put("B", "b");

unsortMap.put("A", "a");

unsortMap.put("C", "c");

unsortMap.put("D", "d");

unsortMap.put("E", "e");

unsortMap.put("Y", "y");

unsortMap.put("N", "n");

unsortMap.put("J", "j");

unsortMap.put("M", "m");

unsortMap.put("F", "f");

System.out.println("Unsort Map......");

printMap(unsortMap);

System.out.println("\nSorted Map......By Key");

Map<String, String> treeMap = new TreeMap<String, String>(unsortMap);

printMap(treeMap);

}

//pretty print a map

public static <K, V> void printMap(Map<K, V> map) {

for (Map.Entry<K, V> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey()

+ " Value : " + entry.getValue());

}

}

}

Output

Unsort Map......

Key : A Value : a

Key : B Value : b

Key : C Value : c

Key : D Value : d

Key : E Value : e

Key : F Value : f

Key : Y Value : y

Key : Z Value : z

Key : J Value : j

Key : M Value : m

Key : N Value : n

Sorted Map......By Key

Key : A Value : a

Key : B Value : b

Key : C Value : c

Key : D Value : d

Key : E Value : e

Key : F Value : f

Key : J Value : j

Key : M Value : m

Key : N Value : n

Key : Y Value : y

Key : Z Value : z

1.2 Yet another java.util.TreeMap example, provide a custom Comparator to sort the key in descending order.

SortByKeyExample2.java

package com.mkyong.test;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Map;

import java.util.TreeMap;

public class SortByKeyExample2 {

public static void main(String[] args) {

Map<Integer, String> unsortMap = new HashMap<Integer, String>();

unsortMap.put(10, "z");

unsortMap.put(5, "b");

unsortMap.put(6, "a");

unsortMap.put(20, "c");

unsortMap.put(1, "d");

unsortMap.put(7, "e");

unsortMap.put(8, "y");

unsortMap.put(99, "n");

unsortMap.put(50, "j");

unsortMap.put(2, "m");

unsortMap.put(9, "f");

System.out.println("Unsort Map......");

printMap(unsortMap);

System.out.println("\nSorted Map......By Key");

Map<Integer, String> treeMap = new TreeMap<Integer, String>(

new Comparator<Integer>() {

@Override

public int compare(Integer o1, Integer o2) {

return o2.compareTo(o1);

}

});

/\* For Java 8, try this lambda

Map<Integer, String> treeMap = new TreeMap<>(

(Comparator<Integer>) (o1, o2) -> o2.compareTo(o1)

);

\*/

treeMap.putAll(unsortMap);

printMap(treeMap);

}

public static <K, V> void printMap(Map<K, V> map) {

for (Map.Entry<K, V> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey()

+ " Value : " + entry.getValue());

}

}

}

Output

Unsort Map......

Key : 1 Value : d

Key : 50 Value : j

Key : 2 Value : m

Key : 99 Value : n

Key : 20 Value : c

Key : 5 Value : b

Key : 6 Value : a

Key : 7 Value : e

Key : 8 Value : y

Key : 9 Value : f

Key : 10 Value : z

Sorted Map......By Key

Key : 99 Value : n

Key : 50 Value : j

Key : 20 Value : c

Key : 10 Value : z

Key : 9 Value : f

Key : 8 Value : y

Key : 7 Value : e

Key : 6 Value : a

Key : 5 Value : b

Key : 2 Value : m

Key : 1 Value : d

2. Sort by Value

Converts the Map into a List<Map>, sorts the List<Map> with a custom Comparator and put it into a new insertion order map – LinkedHashMap

Map ---> List<Map> ---> Collections.sort() --> List<Map> (Sorted) ---> LinkedHashMap

SortByValueExample1.java

package com.mkyong.test;

import java.util.\*;

public class SortByValueExample1 {

public static void main(String[] args) {

Map<String, Integer> unsortMap = new HashMap<String, Integer>();

unsortMap.put("z", 10);

unsortMap.put("b", 5);

unsortMap.put("a", 6);

unsortMap.put("c", 20);

unsortMap.put("d", 1);

unsortMap.put("e", 7);

unsortMap.put("y", 8);

unsortMap.put("n", 99);

unsortMap.put("j", 50);

unsortMap.put("m", 2);

unsortMap.put("f", 9);

System.out.println("Unsort Map......");

printMap(unsortMap);

System.out.println("\nSorted Map......By Value");

Map<String, Integer> sortedMap = sortByValue(unsortMap);

printMap(sortedMap);

}

private static Map<String, Integer> sortByValue(Map<String, Integer> unsortMap) {

// 1. Convert Map to List of Map

List<Map.Entry<String, Integer>> list =

new LinkedList<Map.Entry<String, Integer>>(unsortMap.entrySet());

// 2. Sort list with Collections.sort(), provide a custom Comparator

// Try switch the o1 o2 position for a different order

Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {

public int compare(Map.Entry<String, Integer> o1,

Map.Entry<String, Integer> o2) {

return (o1.getValue()).compareTo(o2.getValue());

}

});

// 3. Loop the sorted list and put it into a new insertion order Map LinkedHashMap

Map<String, Integer> sortedMap = new LinkedHashMap<String, Integer>();

for (Map.Entry<String, Integer> entry : list) {

sortedMap.put(entry.getKey(), entry.getValue());

}

/\*

//classic iterator example

for (Iterator<Map.Entry<String, Integer>> it = list.iterator(); it.hasNext(); ) {

Map.Entry<String, Integer> entry = it.next();

sortedMap.put(entry.getKey(), entry.getValue());

}\*/

return sortedMap;

}

public static <K, V> void printMap(Map<K, V> map) {

for (Map.Entry<K, V> entry : map.entrySet()) {

System.out.println("Key : " + entry.getKey()

+ " Value : " + entry.getValue());

}

}

}

Output

Unsort Map......

Key : a Value : 6

Key : b Value : 5

Key : c Value : 20

Key : d Value : 1

Key : e Value : 7

Key : f Value : 9

Key : y Value : 8

Key : z Value : 10

Key : j Value : 50

Key : m Value : 2

Key : n Value : 99

Sorted Map......By Value

Key : d Value : 1

Key : m Value : 2

Key : b Value : 5

Key : a Value : 6

Key : e Value : 7

Key : y Value : 8

Key : f Value : 9

Key : z Value : 10

Key : c Value : 20

Key : j Value : 50

Key : n Value : 99

2.2 Upgrade the above sortByValue() method to support generics.

public static <K, V extends Comparable<? super V>> Map<K, V> sortByValue(Map<K, V> unsortMap) {

List<Map.Entry<K, V>> list =

new LinkedList<Map.Entry<K, V>>(unsortMap.entrySet());

Collections.sort(list, new Comparator<Map.Entry<K, V>>() {

public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

return (o1.getValue()).compareTo(o2.getValue());

}

});

Map<K, V> result = new LinkedHashMap<K, V>();

for (Map.Entry<K, V> entry : list) {

result.put(entry.getKey(), entry.getValue());

}

return result;

}

# How to work with Java 6’s NavigableSet and NavigableMap

By [Rajiv Srivastava](https://www.mkyong.com/author/rajivmca2004/) | July 26, 2012 | Updated : August 29, 2012 | Viewed : 22,363 times +55 pv/w

You can use latest Java 6’s Collection API to navigate a set and Map collections. These API gives a lot of flexibility to find out required result from the collection.

## 1. NavigableMap Example

package com.example.collection;

import java.util.NavigableMap;

import java.util.TreeMap;

public class NavigableMapDemo {

public static void main(String[] args) {

NavigableMap<String,Integer> navigableMap=new TreeMap<String, Integer>();

navigableMap.put("X", 500);

navigableMap.put("B", 600);

navigableMap.put("A", 700);

navigableMap.put("T", 800);

navigableMap.put("Y", 900);

navigableMap.put("Z", 200);

System.out.printf("Descending Set : %s%n",navigableMap.descendingKeySet());

System.out.printf("Floor Entry : %s%n",navigableMap.floorEntry("L"));

System.out.printf("First Entry : %s%n",navigableMap.firstEntry());

System.out.printf("Last Key : %s%n",navigableMap.lastKey());

System.out.printf("First Key : %s%n",navigableMap.firstKey());

System.out.printf("Original Map : %s%n",navigableMap);

System.out.printf("Reverse Map : %s%n",navigableMap.descendingMap());

}

}

Output

Descending Set : [Z, Y, X, T, B, A]

Floor Entry : B=600

First Entry : A=700

Last Key : Z

First Key : A

Original Map : {A=700, B=600, T=800, X=500, Y=900, Z=200}

Reverse Map : {Z=200, Y=900, X=500, T=800, B=600, A=700}

## 2. NavigableSet Example

package com.example.collection;

import java.util.Arrays;

import java.util.Iterator;

import java.util.NavigableSet;

import java.util.TreeSet;

public class NavigableSetDemo {

public static void main(String[] args) {

NavigableSet<String> navigableSet = new TreeSet<String>(Arrays.asList(

"X", "B", "A", "Z", "T"));

Iterator<String> iterator = navigableSet.descendingIterator();

System.out.println("Original Set :");

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

iterator = navigableSet.iterator();

System.out.println("Sorted Navigable Set :");

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

System.out.printf("Head Set : %s.%n", navigableSet.headSet("X"));

System.out.printf("Tail Set : %s.%n", navigableSet.tailSet("T", false));

System.out.printf("Sub Set : %s.%n",

navigableSet.subSet("B", true, "X", true));

System.out.printf("Last Element : %s%n", navigableSet.last());

System.out.printf("First Element : %s%n", navigableSet.first());

System.out.printf("Reverse Set : %s%n", navigableSet.descendingSet());

System.out.printf("Original Set : %s%n", navigableSet);

}

}

Output

Original Set :

Z

X

T

B

A

Sorted Navigable Set :

A

B

T

X

Z

Head Set : [A, B, T].

Tail Set : [X, Z].

Sub Set : [B, T, X].

Last Element : Z

First Element : A

Reverse Set : [Z, X, T, B, A]

Original Set : [A, B, T, X, Z]

# How to convert String to Byte in Java?

By [mkyong](https://www.mkyong.com/author/mkyong/) | February 27, 2009 | Updated : August 30, 2012 | Viewed : 46,975 times +262 pv/w

Here i demonstrate how do **convert a String to a Byte** variable with String’s **getBytes()** function.

public class TestByte

{

public static void main(String[] argv) {

String example = "This is an example";

byte[] bytes = example.getBytes();

System.out.println("Text : " + example);

System.out.println("Text [Byte Format] : " + bytes);

}

}

Java – How to convert Char to String

By [mkyong](https://www.mkyong.com/author/mkyong/) | July 11, 2009 | Updated : July 27, 2016 | Viewed : 126,716 times +113 pv/w

A Java example to show you how to convert a Char into a String and vise verse.

ConvertCharToString.java

package com.mkyong.utils;

public class ConvertCharToString {

public static void main(String[] args) {

String website = "http://www.mkyong.com";

//convert a String into char

char charH = website.charAt(0); //h

char charP = website.charAt(3);//p

char charM = website.charAt(11);//m

System.out.println(charH);

System.out.println(charP);

System.out.println(charM);

//convert char back to String

String temp = Character.toString(charM);

if ("m".equals(temp)) {

System.out.println("match");

}

}

}

Output

h

p

m

match

How to convert String to Date – Java

By [mkyong](https://www.mkyong.com/author/mkyong/) | July 7, 2013 | Updated : March 10, 2017 | Viewed : 962,759 times +5,278 pv/w

In this tutorial, we will show you how to convert a String to java.util.Date. Many Java beginners are stuck in the Date conversion, hope this summary guide will helps you in some ways.

// String -> Date

SimpleDateFormat.parse(String);

// Date -> String

SimpleDateFormat.format(date);

Refer to table below for some of the common date and time patterns used in java.text.SimpleDateFormat, refer to this [JavaDoc](http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html)

|  |  |  |
| --- | --- | --- |
| Letter | Description | Examples |
| y | Year | 2013 |
| M | Month in year | July, 07, 7 |
| d | Day in month | 1-31 |
| E | Day name in week | Friday, Sunday |
| a | Am/pm marker | AM, PM |
| H | Hour in day | 0-23 |
| h | Hour in am/pm | 1-12 |
| m | Minute in hour | 0-60 |
| s | Second in minute | 0-60 |

**Note**  
You may interest at this Java 8 example – [How to convert String to LocalDate](http://www.mkyong.com/java8/java-8-how-to-convert-string-to-localdate/)

1. String = 7-Jun-2013

If 3 ‘M’, then the month is interpreted as text (Mon-Dec), else number (01-12).

TestDateExample1.java

package com.mkyong.date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class TestDateExample1 {

public static void main(String[] argv) {

SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy");

String dateInString = "7-Jun-2013";

try {

Date date = formatter.parse(dateInString);

System.out.println(date);

System.out.println(formatter.format(date));

} catch (ParseException e) {

e.printStackTrace();

}

}

}

Output

Fri Jun 07 00:00:00 MYT 2013

07-Jun-2013

2. String = 07/06/2013

TestDateExample2.java

package com.mkyong.date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class TestDateExample2 {

public static void main(String[] argv) {

SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");

String dateInString = "07/06/2013";

try {

Date date = formatter.parse(dateInString);

System.out.println(date);

System.out.println(formatter.format(date));

} catch (ParseException e) {

e.printStackTrace();

}

}

}

Output

Fri Jun 07 00:00:00 MYT 2013

07/06/2013

3. String = Fri, June 7 2013

TestDateExample3.java

package com.mkyong.date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class TestDateExample3 {

public static void main(String[] argv) {

SimpleDateFormat formatter = new SimpleDateFormat("E, MMM dd yyyy");

String dateInString = "Fri, June 7 2013";

try {

Date date = formatter.parse(dateInString);

System.out.println(date);

System.out.println(formatter.format(date));

} catch (ParseException e) {

e.printStackTrace();

}

}

}

Output

Fri Jun 07 00:00:00 MYT 2013

Fri, Jun 07 2013

4. String = Friday, Jun 7, 2013 12:10:56 PM

TestDateExample4.java

package com.mkyong.date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class TestDateExample4 {

public static void main(String[] argv) {

SimpleDateFormat formatter = new SimpleDateFormat("EEEE, MMM dd, yyyy HH:mm:ss a");

String dateInString = "Friday, Jun 7, 2013 12:10:56 PM";

try {

Date date = formatter.parse(dateInString);

System.out.println(date);

System.out.println(formatter.format(date));

} catch (ParseException e) {

e.printStackTrace();

}

}

}

Output

Fri Jun 07 12:10:56 MYT 2013

Friday, Jun 07, 2013 12:10:56 PM

5. String = 2014-10-05T15:23:01Z

Z suffix means UTC, java.util.SimpleDateFormat doesn’t parse it correctly, you need to replace the suffix Z with ‘+0000’.

TestDateExample5.java

package com.mkyong.date;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class TestDateExample5 {

public static void main(String[] argv) {

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssZ");

String dateInString = "2014-10-05T15:23:01Z";

try {

Date date = formatter.parse(dateInString.replaceAll("Z$", "+0000"));

System.out.println(date);

System.out.println("time zone : " + TimeZone.getDefault().getID());

System.out.println(formatter.format(date));

} catch (ParseException e) {

e.printStackTrace();

}

}

}

Output

Sun Oct 05 23:23:01 MYT 2014

time zone : Asia/Kuala\_Lumpur

2014-10-05T23:23:01+0800

In Java 8, you can convert it into a java.time.Instant object, and display it with a specified time zone.

TestDateExample6.java

package com.mkyong.date;

import java.time.\*;

public class TestDateExample6 {

public static void main(String[] argv) {

String dateInString = "2014-10-05T15:23:01Z";

Instant instant = Instant.parse(dateInString);

System.out.println(instant);

//get date time only

LocalDateTime result = LocalDateTime.ofInstant(instant, ZoneId.of(ZoneOffset.UTC.getId()));

System.out.println(result);

//get date time + timezone

ZonedDateTime zonedDateTime = instant.atZone(ZoneId.of("Africa/Tripoli"));

System.out.println(zonedDateTime);

//get date time + timezone

ZonedDateTime zonedDateTime2 = instant.atZone(ZoneId.of("Europe/Athens"));

System.out.println(zonedDateTime2);

}

}

Output

2014-10-05T15:23:01Z

2014-10-05T15:23:01

2014-10-05T17:23:01+02:00[Africa/Tripoli]

2014-10-05T18:23:01+03:00[Europe/Athens]

Java Date and Calendar examples

By [mkyong](https://www.mkyong.com/author/mkyong/) | October 21, 2013 | Updated : January 22, 2015 | Viewed : 875,279 times +4,924 pv/w

This tutorial shows you how to work with java.util.Date and java.util.Calendar.

1. Java Date Examples

Few examples to work with Date APIs.

**Example 1.1** – Convert Date to String.

SimpleDateFormat sdf = new SimpleDateFormat("dd/M/yyyy");

String date = sdf.format(new Date());

System.out.println(date); //15/10/2013

**Example 1.2** – Convert String to Date.

SimpleDateFormat sdf = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");

String dateInString = "31-08-1982 10:20:56";

Date date = sdf.parse(dateInString);

System.out.println(date); //Tue Aug 31 10:20:56 SGT 1982

*P.S Refer to this –*[*SimpleDateFormat JavaDoc*](http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html)*for detail date and time patterns.*

**Example 1.3** – Get current date time

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

Date date = new Date();

System.out.println(dateFormat.format(date)); //2013/10/15 16:16:39

**Example 1.4** – Convert Calendar to Date

Calendar calendar = Calendar.getInstance();

Date date = calendar.getTime();

2. Java Calendar Examples

Few examples to work with Calendar APIs.

**Example 2.1** – Get current date time

SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd HH:mm:ss");

Calendar calendar = new GregorianCalendar(2013,0,31);

System.out.println(sdf.format(calendar.getTime()));

Output

2013 Jan 31 00:00:00

**Example 2.2** – Simple Calendar example

SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd HH:mm:ss");

Calendar calendar = new GregorianCalendar(2013,1,28,13,24,56);

int year = calendar.get(Calendar.YEAR);

int month = calendar.get(Calendar.MONTH); // Jan = 0, dec = 11

int dayOfMonth = calendar.get(Calendar.DAY\_OF\_MONTH);

int dayOfWeek = calendar.get(Calendar.DAY\_OF\_WEEK);

int weekOfYear = calendar.get(Calendar.WEEK\_OF\_YEAR);

int weekOfMonth= calendar.get(Calendar.WEEK\_OF\_MONTH);

int hour = calendar.get(Calendar.HOUR); // 12 hour clock

int hourOfDay = calendar.get(Calendar.HOUR\_OF\_DAY); // 24 hour clock

int minute = calendar.get(Calendar.MINUTE);

int second = calendar.get(Calendar.SECOND);

int millisecond= calendar.get(Calendar.MILLISECOND);

System.out.println(sdf.format(calendar.getTime()));

System.out.println("year \t\t: " + year);

System.out.println("month \t\t: " + month);

System.out.println("dayOfMonth \t: " + dayOfMonth);

System.out.println("dayOfWeek \t: " + dayOfWeek);

System.out.println("weekOfYear \t: " + weekOfYear);

System.out.println("weekOfMonth \t: " + weekOfMonth);

System.out.println("hour \t\t: " + hour);

System.out.println("hourOfDay \t: " + hourOfDay);

System.out.println("minute \t\t: " + minute);

System.out.println("second \t\t: " + second);

System.out.println("millisecond \t: " + millisecond);

Output

2013 Feb 28 13:24:56

year : 2013

month : 1

dayOfMonth : 28

dayOfWeek : 5

weekOfYear : 9

weekOfMonth : 5

hour : 1

hourOfDay : 13

minute : 24

second : 56

millisecond : 0

**Example 2.3** – Set a date manually.

SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd HH:mm:ss");

Calendar calendar = new GregorianCalendar(2013,1,28,13,24,56);

System.out.println("#1. " + sdf.format(calendar.getTime()));

//update a date

calendar.set(Calendar.YEAR, 2014);

calendar.set(Calendar.MONTH, 11);

calendar.set(Calendar.MINUTE, 33);

System.out.println("#2. " + sdf.format(calendar.getTime()));

Output

#1. 2013 Feb 28 13:24:56

#2. 2014 Dec 28 13:33:56

**Example 2.4**– Add or subtract from a date.

SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd");

Calendar calendar = new GregorianCalendar(2013,10,28);

System.out.println("Date : " + sdf.format(calendar.getTime()));

//add one month

calendar.add(Calendar.MONTH, 1);

System.out.println("Date : " + sdf.format(calendar.getTime()));

//subtract 10 days

calendar.add(Calendar.DAY\_OF\_MONTH, -10);

System.out.println("Date : " + sdf.format(calendar.getTime()));

Output

Date : 2013 Nov 28

Date : 2013 Dec 28

Date : 2013 Dec 18

**Example 2.5**– Convert Date to Calendar.

SimpleDateFormat sdf = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");

String dateInString = "22-01-2015 10:20:56";

Date date = sdf.parse(dateInString);

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

# Java 8 forEach examples

By [mkyong](https://www.mkyong.com/author/mkyong/) | August 20, 2015 | Updated : August 18, 2016 | Viewed : 477,452 times +10,234 pv/w

In this article, we will show you how to loop a List and a Map with the new Java 8 forEach statement.

## 1. forEach and Map

1.1 Normal way to loop a Map.

Map<String, Integer> items = new HashMap<>();

items.put("A", 10);

items.put("B", 20);

items.put("C", 30);

items.put("D", 40);

items.put("E", 50);

items.put("F", 60);

for (Map.Entry<String, Integer> entry : items.entrySet()) {

System.out.println("Item : " + entry.getKey() + " Count : " + entry.getValue());

}

1.2 In Java 8, you can loop a Map with forEach + lambda expression.

Map<String, Integer> items = new HashMap<>();

items.put("A", 10);

items.put("B", 20);

items.put("C", 30);

items.put("D", 40);

items.put("E", 50);

items.put("F", 60);

items.forEach((k,v)->System.out.println("Item : " + k + " Count : " + v));

items.forEach((k,v)->{

System.out.println("Item : " + k + " Count : " + v);

if("E".equals(k)){

System.out.println("Hello E");

}

});

## 2. forEach and List

2.1 Normal for-loop to loop a List.

List<String> items = new ArrayList<>();

items.add("A");

items.add("B");

items.add("C");

items.add("D");

items.add("E");

for(String item : items){

System.out.println(item);

}

2.2 In Java 8, you can loop a List with forEach + lambda expression or method reference.

List<String> items = new ArrayList<>();

items.add("A");

items.add("B");

items.add("C");

items.add("D");

items.add("E");

//lambda

//Output : A,B,C,D,E

items.forEach(item->System.out.println(item));

//Output : C

items.forEach(item->{

if("C".equals(item)){

System.out.println(item);

}

});

//method reference

//Output : A,B,C,D,E

items.forEach(System.out::println);

//Stream and filter

//Output : B

items.stream()

.filter(s->s.contains("B"))

.forEach(System.out::println);

Java enum example

By [mkyong](https://www.mkyong.com/author/mkyong/) | October 21, 2012 | Updated : March 24, 2016 | Viewed : 328,099 times +4,459 pv/w

Some of the Java enum examples, and how to use it, nothing special, just for self-reference.

**Note**  
Consider the Enum type if your program consists of a fixed set of constants, like seasons of the year, operations calculator, user status and etc.

1. Basic Enum

UserStatus.java

public enum UserStatus {

PENDING,

ACTIVE,

INACTIVE,

DELETED;

}

Test.java

public class Test {

public static void main(String[] args) {

//ACTIVE

System.out.println(UserStatus.ACTIVE);

}

}

2. Enum + Instance field

WhoisRIR.java

public enum WhoisRIR {

ARIN("whois.arin.net"),

RIPE("whois.ripe.net"),

APNIC("whois.apnic.net"),

AFRINIC("whois.afrinic.net"),

LACNIC("whois.lacnic.net"),

JPNIC("whois.nic.ad.jp"),

KRNIC("whois.nic.or.kr"),

CNNIC("ipwhois.cnnic.cn"),

UNKNOWN("");

private String url;

WhoisRIR(String url) {

this.url = url;

}

public String url() {

return url;

}

}

Test.java

public class Test {

public static void main(String[] args) {

//whois.arin.net

System.out.println(WhoisRIR.ARIN.url());

}

}

3. Enum + Method + Some logic

Operation.java

public enum Operation {

PLUS,

MINUS,

TIMES,

DIVIDE;

double calculate(double x, double y) {

switch (this) {

case PLUS:

return x + y;

case MINUS:

return x - y;

case TIMES:

return x \* y;

case DIVIDE:

return x / y;

default:

throw new AssertionError("Unknown operations " + this);

}

}

}

Test.java

public class Test {

public static void main(String[] args) {

double result = Operation.PLUS.calculate(1, 2);

System.out.println(result); //3.0

}

}

4. How to use Enum

4.1 To loop a Enum object.

public class Test {

public static void main(String[] args) {

for (UserStatus status : UserStatus.values()) {

System.out.println(status);

}

}

}

Output

PENDING

ACTIVE

INACTIVE

DELETED

4.2 To compare the Enum values, use == operator.

public class Test {

public static void main(String[] args) {

WhoisRIR rir = WhoisRIR.APNIC;

if(rir == WhoisRIR.APNIC) {

System.out.println("This is APNIC : " + rir.url());

}

}

}

Output

This is APNIC : whois.apnic.net

4.3 Switch case.

public class Test {

public static void main(String[] args) {

WhoisRIR rir = WhoisRIR.RIPE;

switch (rir) {

case ARIN:

System.out.println("This is ARIN");

break;

case APNIC:

System.out.println("This is APNIC");

break;

case RIPE:

System.out.println("This is RIPE");

break;

default:

throw new AssertionError("Unknown RIR " + rir);

}

}

}

Output

This is RIPE

4.2 Convert a String to Enum object.

public class Test {

public static void main(String[] args) {

//enum valueOf + uppercase

Operation op = Operation.valueOf("times".toUpperCase());

System.out.println(op.calculate(10, 3));

}

}

Output

30.0

Java Properties file examples

By [mkyong](https://www.mkyong.com/author/mkyong/) | January 19, 2010 | Updated : January 17, 2014 | Viewed : 1,581,415 times +7,916 pv/w

Normally, Java properties file is used to store project configuration data or settings. In this tutorial, we will show you how to read and write to/from a properties file.

1. Write to properties file

Set the property value, and write it into a properties file named config.properties.

App.java

package com.mkyong.properties;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.util.Properties;

public class App {

public static void main(String[] args) {

Properties prop = new Properties();

OutputStream output = null;

try {

output = new FileOutputStream("config.properties");

// set the properties value

prop.setProperty("database", "localhost");

prop.setProperty("dbuser", "mkyong");

prop.setProperty("dbpassword", "password");

// save properties to project root folder

prop.store(output, null);

} catch (IOException io) {

io.printStackTrace();

} finally {

if (output != null) {

try {

output.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

*Output*

config.properties

#Fri Jan 17 22:37:45 MYT 2014

dbpassword=password

database=localhost

dbuser=mkyong

*P.S If file path is not specified, then this properties file will be stored in your project root folder.*

2. Load a properties file

Load a properties file from the file system and retrieved the property value.

App.java

package com.mkyong.properties;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

public class App {

public static void main(String[] args) {

Properties prop = new Properties();

InputStream input = null;

try {

input = new FileInputStream("config.properties");

// load a properties file

prop.load(input);

// get the property value and print it out

System.out.println(prop.getProperty("database"));

System.out.println(prop.getProperty("dbuser"));

System.out.println(prop.getProperty("dbpassword"));

} catch (IOException ex) {

ex.printStackTrace();

} finally {

if (input != null) {

try {

input.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

*Output*

localhost

mkyong

password

3. Load a properties file from classpath

Load a properties file config.properties from project classpath, and retrieved the property value.

*P.S Assume properties file “config.properties” is in your project classpath root folder.*

App.java

package com.mkyong.properties;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

public class App {

public static void main( String[] args ){

Properties prop = new Properties();

InputStream input = null;

try {

String filename = "config.properties";

input = App3.class.getClassLoader().getResourceAsStream(filename);

if(input==null){

System.out.println("Sorry, unable to find " + filename);

return;

}

//load a properties file from class path, inside static method

prop.load(input);

//get the property value and print it out

System.out.println(prop.getProperty("database"));

System.out.println(prop.getProperty("dbuser"));

System.out.println(prop.getProperty("dbpassword"));

} catch (IOException ex) {

ex.printStackTrace();

} finally{

if(input!=null){

try {

input.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

*Output*

localhost

mkyong

password

For non-static method, use this :

prop.load(getClass().getClassLoader().getResourceAsStream("config.properties"));

4. Prints everything from a properties file

Load a properties file config.properties from project classpath, and get the keys and values.

App.java

package com.mkyong.properties;

import java.io.IOException;

import java.io.InputStream;

import java.util.Enumeration;

import java.util.Properties;

public class App {

public static void main(String[] args) {

App app = new App();

app.printThemAll();

}

private void printThemAll() {

Properties prop = new Properties();

InputStream input = null;

try {

String filename = "config.properties";

input = getClass().getClassLoader().getResourceAsStream(filename);

if (input == null) {

System.out.println("Sorry, unable to find " + filename);

return;

}

prop.load(input);

Enumeration<?> e = prop.propertyNames();

while (e.hasMoreElements()) {

String key = (String) e.nextElement();

String value = prop.getProperty(key);

System.out.println("Key : " + key + ", Value : " + value);

}

} catch (IOException ex) {

ex.printStackTrace();

} finally {

if (input != null) {

try {

input.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

*Output*

Key : dbpassword, Value : password

Key : database, Value : localhost

Key : dbuser, Value : mkyong

# How to read input from console – Java

By [mkyong](https://www.mkyong.com/author/mkyong/) | December 11, 2008 | Updated : July 27, 2016 | Viewed : 316,643 times +860 pv/w

In Java, there are 3 ways to read input from a console :

1. BufferedReader + InputStreamReader (Classic)
2. Scanner (JDK 1.5)
3. System.console (JDK 1.6)

## 1. BufferedReader + InputStreamReader

For experienced Java developers, you will miss this classic way to read a system input.

ReadConsole.java

package com.mkyong;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class ReadConsole {

public static void main(String[] args) {

BufferedReader br = null;

try {

br = new BufferedReader(new InputStreamReader(System.in));

while (true) {

System.out.print("Enter something : ");

String input = br.readLine();

if ("q".equals(input)) {

System.out.println("Exit!");

System.exit(0);

}

System.out.println("input : " + input);

System.out.println("-----------\n");

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (br != null) {

try {

br.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

Output

Enter something : old and classic

input : old and classic

-----------

Enter something : q

Exit!

## 2. Scanner

In JDK 1.5, the developer starts to use java.util.Scanner to read system input.

ReadConsole2.java

package com.mkyong;

public class ReadConsole2 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.print("Enter something : ");

String input = scanner.nextLine();

if ("q".equals(input)) {

System.out.println("Exit!");

break;

}

System.out.println("input : " + input);

System.out.println("-----------\n");

}

scanner.close();

}

}

Output

Enter something : hello jdk 1.5

input : hello jdk 1.5

-----------

Enter something : scanner example

input : scanner example

-----------

Enter something : q

Exit!

## 3. System.console

In JDK 1.6, the developer starts to switch to the more simple and powerful java.io.Console class.

ReadConsole3.java

package com.mkyong;

public class ReadConsole3 {

public static void main(String[] args) {

while (true) {

System.out.print("Enter something : ");

String input = System.console().readLine();

if ("q".equals(input)) {

System.out.println("Exit!");

System.exit(0);

}

System.out.println("input : " + input);

System.out.println("-----------\n");

}

}

}

Output

Enter something : hello jdk 1.6

input : hello jdk 1.6

-----------

Enter something : console example

input : cosole example

-----------

Enter something : q

Exit!

**Note**  
This System.console() is usable only outside IDE, a bit hard for testing.

Quartz 1.6 scheduler tutorial

By [mkyong](https://www.mkyong.com/author/mkyong/) | April 3, 2010 | Updated : August 30, 2012 | Viewed : 303,482 times +547 pv/w

[Quartz](http://www.quartz-scheduler.org/) is a powerful and advance scheduler framework, to help Java developer to scheduler a job to run at a specified date and time.

This tutorial show you how to develop a scheduler job using Quartz 1.6.3.

**Note**  
This example is a bit outdate, unless you are still using the old Quartz 1.6.3 library, otherwise, you may interest of this latest [Quartz 2.1.5 example](http://www.mkyong.com/java/quartz-2-scheduler-tutorial/).

1. Download Quartz

You can get the Quartz library from [official website](http://www.quartz-scheduler.org/) or Maven central repository

File : pom.xml

<dependencies>

<!-- Quartz API -->

<dependency>

<groupId>opensymphony</groupId>

<artifactId>quartz</artifactId>

<version>1.6.3</version>

</dependency>

<dependency>

<groupId>commons-collections</groupId>

<artifactId>commons-collections</artifactId>

<version>3.2.1</version>

</dependency>

<dependency>

<groupId>org.apache.directory.studio</groupId>

<artifactId>org.apache.commons.logging</artifactId>

<version>1.1.1</version>

</dependency>

</dependencies>

2. Quartz Job

Quartz job is defined what you want to run?

File : HelloJob

package com.mkyong.common;

import org.quartz.Job;

import org.quartz.JobExecutionContext;

import org.quartz.JobExecutionException;

public class HelloJob implements Job

{

public void execute(JobExecutionContext context)

throws JobExecutionException {

System.out.println("Hello Quartz!");

}

}

3. Quartz Trigger

Quartz trigger is defined when the Quartz will run your above Quartz’s job?

There are two types of Quartz triggers :

* SimpleTrigger – Allows to set start time, end time, repeat interval.
* CronTrigger – Allows Unix cron expression to specify the dates and times to run your job.

Unix cron expression  
The Unix cron expression is highly flexible and powerful, you can learn and see many cron expression examples in following websites.

1. <http://en.wikipedia.org/wiki/CRON_expression>
2. <http://www.quartz-scheduler.org/docs/examples/Example3.html>

SimpleTrigger – Run every 30 seconds.

SimpleTrigger trigger = new SimpleTrigger();

trigger.setName("dummyTriggerName");

trigger.setStartTime(new Date(System.currentTimeMillis() + 1000));

trigger.setRepeatCount(SimpleTrigger.REPEAT\_INDEFINITELY);

trigger.setRepeatInterval(30000);

CronTrigger – Run every 30 seconds.

CronTrigger trigger = new CronTrigger();

trigger.setName("dummyTriggerName");

trigger.setCronExpression("0/30 \* \* \* \* ?");

4. Scheduler

Scheduler class links both “**Job**” and “**Trigger**” together and execute it.

Scheduler scheduler = new StdSchedulerFactory().getScheduler();

scheduler.start();

scheduler.scheduleJob(job, trigger);

5. Full Example

Here are two full examples to use Quartz, via SimpleTrigger and CronTrigger.

**SimpleTrigger example**  
Run very 30 seconds with a 1 second delay for the first time of execution.

package com.mkyong.common;

import java.util.Date;

import org.quartz.JobDetail;

import org.quartz.Scheduler;

import org.quartz.SimpleTrigger;

import org.quartz.impl.StdSchedulerFactory;

public class SimpleTriggerExample

{

public static void main( String[] args ) throws Exception

{

JobDetail job = new JobDetail();

job.setName("dummyJobName");

job.setJobClass(HelloJob.class);

//configure the scheduler time

SimpleTrigger trigger = new SimpleTrigger();

trigger.setStartTime(new Date(System.currentTimeMillis() + 1000));

trigger.setRepeatCount(SimpleTrigger.REPEAT\_INDEFINITELY);

trigger.setRepeatInterval(30000);

//schedule it

Scheduler scheduler = new StdSchedulerFactory().getScheduler();

scheduler.start();

scheduler.scheduleJob(job, trigger);

}

}

**CronTrigger example**  
Same, run the job at every 30 seconds.

package com.mkyong.common;

import org.quartz.CronTrigger;

import org.quartz.JobDetail;

import org.quartz.Scheduler;

import org.quartz.impl.StdSchedulerFactory;

public class CronTriggerExample

{

public static void main( String[] args ) throws Exception

{

JobDetail job = new JobDetail();

job.setName("dummyJobName");

job.setJobClass(HelloJob.class);

CronTrigger trigger = new CronTrigger();

trigger.setName("dummyTriggerName");

trigger.setCronExpression("0/30 \* \* \* \* ?");

//schedule it

Scheduler scheduler = new StdSchedulerFactory().getScheduler();

scheduler.start();

scheduler.scheduleJob(job, trigger);

}

}

Java Regular Expression Tutorial

By [mkyong](http://www.mkyong.com/author/mkyong/) | April 7, 2010 | Updated : August 30, 2012 | Viewed : 158,559 times +184 pv/w

Java has comprehensive support for Regular Expression functionality through the **java.util.regex** package. The regular expression language is easy to learn but hard to master, the better way to learn it is through examples. In theoretical, regular expression can match almost any stuff you want, the only limitation is in your imagination.

Happy learning Java Regular Expression :)

* [Username regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-username-with-regular-expression/)  
  Username regular expression example in Java and unit tested with TestNG.

^[a-z0-9\_-]{3,15}$

* [Password regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-password-with-regular-expression/)  
  Password regular expression example in Java and unit tested with TestNG.

((?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%]).{6,20})

* [Hex color code regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-hex-color-code-with-regular-expression/)  
  Hex color code regular expression example in Java and unit tested with TestNG.

^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$

* [E-mail address regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-email-address-with-regular-expression/)  
  E-mail address regular expression example in Java and unit tested with TestNG.
* ^[\_A-Za-z0-9-]+(\\.[\_A-Za-z0-9-]+)\*@

[A-Za-z0-9]+(\\.[A-Za-z0-9]+)\*(\\.[A-Za-z]{2,})$

* [Image file extension regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-image-file-extension-with-regular-expression/)  
  Image file extension regular expression example in Java and unit tested with TestNG.

([^\s]+(\.(?i)(jpg|png|gif|bmp))$)

* [IP Address regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-ip-address-with-regular-expression/)  
  IP Address regular expression example in Java and unit tested with TestNG.
* ^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.

([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])$

* [Time in 12 Hours format regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-time-in-12-hours-format-with-regular-expression/)  
  Time in 12 Hours format regular expression example in Java and unit tested with TestNG.

(1[012]|[1-9]):[0-5][0-9](\\s)?(?i)(am|pm)

* [Time in 24 Hours format regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-time-in-24-hours-format-with-regular-expression/)  
  Time in 24 Hours format regular expression example in Java and unit tested with TestNG.

([01]?[0-9]|2[0-3]):[0-5][0-9]

* [Date regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-date-with-regular-expression/)  
  Date regular expression example in Java and unit tested with TestNG.

(0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[012])/((19|20)\\d\\d)

* [HTML tag regular expression](http://www.mkyong.com/regular-expressions/how-to-validate-html-tag-with-regular-expression/)  
  HTML tag regular expression example in Java and unit tested with TestNG.

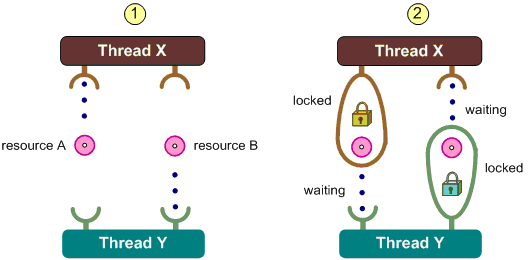
<("[^"]\*"|'[^']\*'|[^'">])\*>

* [HTML Links regular expression](http://www.mkyong.com/regular-expressions/how-to-extract-html-links-with-regular-expression/)  
  HTML links regular expression example in Java and unit tested with TestNG.

(?i)<a([^>]+)>(.+?)</a>

\s\*(?i)href\s\*=\s\*(\"([^"]\*\")|'[^']\*'|([^'">\s]+));

Deadlock in Java Multithreading

[**synchronized**](http://quiz.geeksforgeeks.org/synchronized-in-java/) keyword is used to make the class or method thread-safe which means only one thread can have lock of synchronized method and use it, other threads have to wait till the lock releases and anyone of them acquire that lock.  
It is important to use if our program is running in multi-threaded environment where two or more threads execute simultaneously. But sometimes it also causes a problem which is called [**Deadlock**](http://quiz.geeksforgeeks.org/operating-system-process-management-deadlock-introduction/). Below is a simple example of Deadlock condition.  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/threads_deadlock.gif)  
Image source: https://software.intel.com/en-us/articles/multi-threading-in-the-net-environment

|  |
| --- |
| // Java program to illustrate Deadlock  // in multithreading.  class Util  {      // Util class to sleep a thread      static void sleep(long millis)      {          try          {              Thread.sleep(millis);          }          catch (InterruptedException e)          {              e.printStackTrace();          }      }  }    // This class is shared by both threads  class Shared  {      // first synchronized method      synchronized void test1(Shared s2)      {          System.out.println("test1-begin");          Util.sleep(1000);            // taking object lock of s2 enters          // into test2 method          s2.test2(this);          System.out.println("test1-end");      }        // second synchronized method      synchronized void test2(Shared s1)      {          System.out.println("test2-begin");          Util.sleep(1000);            // taking object lock of s1 enters          // into test1 method          s1.test1(this);          System.out.println("test2-end");      }  }      class Thread1 extends Thread  {      private Shared s1;      private Shared s2;        // constructor to initialize fields      public Thread1(Shared s1, Shared s2)      {          this.s1 = s1;          this.s2 = s2;      }        // run method to start a thread      @Override      public void run()      {          // taking object lock of s1 enters          // into test1 method          s1.test1(s2);      }  }      class Thread2 extends Thread  {      private Shared s1;      private Shared s2;        // constructor to initialize fields      public Thread2(Shared s1, Shared s2)      {          this.s1 = s1;          this.s2 = s2;      }        // run method to start a thread      @Override      public void run()      {          // taking object lock of s2          // enters into test2 method          s2.test2(s1);      }  }      public class GFG  {      public static void main(String[] args)      {          // creating one object          Shared s1 = new Shared();            // creating second object          Shared s2 = new Shared();            // creating first thread and starting it          Thread1 t1 = new Thread1(s1, s2);          t1.start();            // creating second thread and starting it          Thread2 t2 = new Thread2(s1, s2);          t2.start();            // sleeping main thread          Util.sleep(2000);      }  } |

Run on IDE

Output : test1-begin

test2-begin

It is not recommended to run the above program with online IDE. We can copy the source code and run it on our local machine. We can see that it runs for indefinite time, because threads are in deadlock condition and doesn’t let code to execute. Now let’s see step by step what is happening there.

1. Thread t1 starts and calls test1 method by taking the object lock of s1.
2. Thread t2 starts and calls test2 method by taking the object lock of s2.
3. t1 prints test1-begin and t2 prints test-2 begin and both waits for 1 second, so that both threads can be started if any of them is not.
4. t1 tries to take object lock of s2 and call method test2 but as it is already acquired by t2 so it waits till it become free. It will not release lock of s1 until it gets lock of s2.
5. Same happens with t2. It tries to take object lock of s1 and call method test1 but it is already acquired by t1, so it has to wait till t1 release the lock. t2 will also not release lock of s2 until it gets lock of s1.
6. Now, both threads are in wait state, waiting for each other to release locks. Now there is a race around condition that who will release the lock first.
7. As none of them is ready to release lock, so this is the Dead Lock condition.
8. When you will run this program, it will be look like execution is paused.

**Detect Dead Lock condition**

We can also detect deadlock by running this program on cmd. We have to collect Thread Dump. Command to collect depends on OS type. If we are using Windows and Java 8, command is jcmd $PID Thread.print  
We can get PID by running jps command. Thread dump for above program is below:

5524:

2017-04-21 09:57:39

Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.25-b02 mixed mode):

"DestroyJavaVM" #12 prio=5 os\_prio=0 tid=0x0000000002690800 nid=0xba8 waiting on condition [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"Thread-1" #11 prio=5 os\_prio=0 tid=0x0000000018bbf800 nid=0x12bc waiting for monitor entry [0x000000001937f000]

java.lang.Thread.State: BLOCKED (on object monitor)

at Shared.test1(GFG.java:15)

- waiting to lock (a Shared)

at Shared.test2(GFG.java:29)

- locked (a Shared)

at Thread2.run(GFG.java:68)

"Thread-0" #10 prio=5 os\_prio=0 tid=0x0000000018bbc000 nid=0x1d8 waiting for monitor entry [0x000000001927f000]

java.lang.Thread.State: BLOCKED (on object monitor)

at Shared.test2(GFG.java:25)

- waiting to lock (a Shared)

at Shared.test1(GFG.java:19)

- locked (a Shared)

at Thread1.run(GFG.java:49)

"Service Thread" #9 daemon prio=9 os\_prio=0 tid=0x000000001737d800 nid=0x1680 runnable [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"C1 CompilerThread2" #8 daemon prio=9 os\_prio=2 tid=0x000000001732b800 nid=0x17b0 waiting on condition [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"C2 CompilerThread1" #7 daemon prio=9 os\_prio=2 tid=0x0000000017320800 nid=0x7b4 waiting on condition [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"C2 CompilerThread0" #6 daemon prio=9 os\_prio=2 tid=0x000000001731b000 nid=0x21b0 waiting on condition [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"Attach Listener" #5 daemon prio=5 os\_prio=2 tid=0x0000000017319800 nid=0x1294 waiting on condition [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"Signal Dispatcher" #4 daemon prio=9 os\_prio=2 tid=0x0000000017318000 nid=0x1efc runnable [0x0000000000000000]

java.lang.Thread.State: RUNNABLE

"Finalizer" #3 daemon prio=8 os\_prio=1 tid=0x0000000002781800 nid=0x5a0 in Object.wait() [0x000000001867f000]

java.lang.Thread.State: WAITING (on object monitor)

at java.lang.Object.wait(Native Method)

- waiting on (a java.lang.ref.ReferenceQueue$Lock)

at java.lang.ref.ReferenceQueue.remove(Unknown Source)

- locked (a java.lang.ref.ReferenceQueue$Lock)

at java.lang.ref.ReferenceQueue.remove(Unknown Source)

at java.lang.ref.Finalizer$FinalizerThread.run(Unknown Source)

"Reference Handler" #2 daemon prio=10 os\_prio=2 tid=0x000000000277a800 nid=0x15b4 in Object.wait() [0x000000001857f000]

java.lang.Thread.State: WAITING (on object monitor)

at java.lang.Object.wait(Native Method)

- waiting on (a java.lang.ref.Reference$Lock)

at java.lang.Object.wait(Unknown Source)

at java.lang.ref.Reference$ReferenceHandler.run(Unknown Source)

- locked (a java.lang.ref.Reference$Lock)

"VM Thread" os\_prio=2 tid=0x00000000172e6000 nid=0x1fec runnable

"GC task thread#0 (ParallelGC)" os\_prio=0 tid=0x00000000026a6000 nid=0x21fc runnable

"GC task thread#1 (ParallelGC)" os\_prio=0 tid=0x00000000026a7800 nid=0x2110 runnable

"GC task thread#2 (ParallelGC)" os\_prio=0 tid=0x00000000026a9000 nid=0xc54 runnable

"GC task thread#3 (ParallelGC)" os\_prio=0 tid=0x00000000026ab800 nid=0x704 runnable

"VM Periodic Task Thread" os\_prio=2 tid=0x0000000018ba0800 nid=0x610 waiting on condition

JNI global references: 6

Found one Java-level deadlock:

=============================

"Thread-1":

waiting to lock monitor 0x0000000018bc1e88 (object 0x00000000d5d645a0, a Shared),

which is held by "Thread-0"

"Thread-0":

waiting to lock monitor 0x0000000002780e88 (object 0x00000000d5d645b0, a Shared),

which is held by "Thread-1"

Java stack information for the threads listed above:

===================================================

"Thread-1":

at Shared.test1(GFG.java:15)

- waiting to lock (a Shared)

at Shared.test2(GFG.java:29)

- locked (a Shared)

at Thread2.run(GFG.java:68)

"Thread-0":

at Shared.test2(GFG.java:25)

- waiting to lock (a Shared)

at Shared.test1(GFG.java:19)

- locked (a Shared)

at Thread1.run(GFG.java:49)

Found 1 deadlock.

As we can see there is clearly mentioned that found 1 deadlock. It is possible that the same message appears when you try on your machine.

**Avoid Dead Lock condition**

We can avoid dead lock condition by knowing its possibilities. It’s a very complex process and not easy to catch. But still if we try, we can avoid this. There are some methods by which we can avoid this condition. We can’t completely remove its possibility but we can reduce.

* **Avoid Nested Locks :**This is the main reason for dead lock. Dead Lock mainly happens when we give locks to multiple threads. Avoid giving lock to multiple threads if we already have given to one.
* **Avoid Unnecessary Locks :**We should have lock only those members which are required. Having lock on unnecessarily can lead to dead lock.
* **Using thread join :**Dead lock condition appears when one thread is waiting other to finish. If this condition occurs we can use Thread.join with maximum time you think the execution will take.

**Important Points :**

* If threads are waiting for each other to finish, then the condition is known as Deadlock.
* Deadlock condition is a complex condition which occurs only in case of multiple threads.
* Deadlock condition can break our code at run time and can destroy business logic.
* We should avoid this condition as much as we can.

Inter-thread Communication in Java

Prerequisite : [Multithreading in Java](http://geeksquiz.com/multithreading-in-java/), [Synchronized in Java](http://quiz.geeksforgeeks.org/synchronized-in-java/)

**What is Polling and what are problems with it?**  
The process of testing a condition repeatedly till it becomes true is known as polling.

Polling is usually implemented with the help of loops to check whether a particular condition is true or not. If it is true, certain action is taken. This waste many CPU cycles and makes the implementation inefficient.  
For example, in a classic queuing problem where one thread is producing data and other is consuming it.

**How Java multi threading tackles this problem?**  
To avoid polling, Java uses three methods, namely, **wait(), notify() and notifyAll().**  
All these methods belong to object class as final so that all classes have them. They must be used within a synchronized block only.

* **wait()-**It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify().
* **notify()-**It wakes up one single thread that called wait() on the same object. It should be noted that calling notify() does not actually give up a lock on a resource.
* **notifyAll()-**It wakes up all the threads that called wait() on the same object.

**A simple Java program to demonstrate the three methods-**  
Please note that this program might only run in offline IDEs as it contains taking input at several points.

|  |
| --- |
| // Java program to demonstrate inter-thread communication  // (wait(), join() and notify()) in Java  import java.util.Scanner;  public class threadexample  {      public static void main(String[] args)                             throws InterruptedException      {          final PC pc = new PC();            // Create a thread object that calls pc.produce()          Thread t1 = new Thread(new Runnable()          {              @Override              public void run()              {                  try                  {                      pc.produce();                  }                  catch(InterruptedException e)                  {                      e.printStackTrace();                  }              }          });            // Create another thread object that calls          // pc.consume()          Thread t2 = new Thread(new Runnable()          {              @Override              public void run()              {                  try                  {                      pc.consume();                  }                  catch(InterruptedException e)                  {                      e.printStackTrace();                  }              }          });            // Start both threads          t1.start();          t2.start();            // t1 finishes before t2          t1.join();          t2.join();      }        // PC (Produce Consumer) class with produce() and      // consume() methods.      public static class PC      {          // Prints a string and waits for consume()          public void produce()throws InterruptedException          {              // synchronized block ensures only one thread              // running at a time.              synchronized(this)              {                  System.out.println("producer thread running");                    // releases the lock on shared resource                  wait();                    // and waits till some other method invokes notify().                  System.out.println("Resumed");              }          }            // Sleeps for some time and waits for a key press. After key          // is pressed, it notifies produce().          public void consume()throws InterruptedException          {              // this makes the produce thread to run first.              Thread.sleep(1000);              Scanner s = new Scanner(System.in);                // synchronized block ensures only one thread              // running at a time.              synchronized(this)              {                  System.out.println("Waiting for return key.");                  s.nextLine();                  System.out.println("Return key pressed");                    // notifies the produce thread that it                  // can wake up.                  notify();                    // Sleep                  Thread.sleep(2000);              }          }      }  } |

Run on IDE

Output:

producer thread running

Waiting for return key.

Return key pressed

Resumed

As monstrous as it seems, it really is a piece of cake if you go through it twice.

1. In the main class a new PC object is created.
2. It runs produce and consume methods of PC object using two different threads namely t1 and t2 and wait for these threads to finish.

Lets understand how our produce and consume method works.

* First of all, use of synchronized block ensures that only one thread at a time runs. Also since there is a sleep method just at the beginning of consume loop, the produce thread gets a kickstart.
* When the wait is called in produce method, it does two things. Firstly it releases the lock it holds on PC object. Secondly it makes the produce thread to go on a waiting state until all other threads have terminated, that is it can again acquire a lock on PC object and some other method wakes it up by invoking notify or notifyAll on the same object.
* Therefore we see that as soon as wait is called, the control transfers to consume thread and it prints -“Waiting for return key”.
* After we press the return key, consume method invokes notify(). It also does 2 things- Firstly, unlike wait(), it does not releases the lock on shared resource therefore for getting the desired result, it is advised to use notify only at the end of your method. Secondly, it notifies the waiting threads that now they can wake up but only after the current method terminates.
* As you might have observed that even after notifying, the control does not immediately passes over to the produce thread. The reason for it being that we have called Thread.sleep() after notify(). As we already know that the consume thread is holding a lock on PC object, another thread cannot access it until it has released the lock. Hence only after the consume thread finishes its sleep time and thereafter terminates by itself, the produce thread cannot take back the control.
* After a 2 second pause, the program terminates to its completion.

If you are still confused as to why we have used notify in consume thread, try removing it and running your program again. As you must have noticed now that the program never terminates.  
The reason for this is straightforward-When you called wait on produce thread, it went on waiting and never terminated. Since a program runs till all its threads have terminated, it runs on and on.  
There is a second way round this problem. You can use a second variant of wait().

void wait(long timeout)

This would make the calling thread sleep only for a time specified.

**References :**  
Java 2 Complete Reference

This article is contributed by **Rishabh Mahrsee**.If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

# Producer Consumer Problem with Wait and Notify Example

Producer Consumer Problem is a classical concurrency problem and in fact it is one of the concurrency design pattern. In last article we have seen solving [Producer Consumer problem in Java using blocking Queue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html) but one of my reader emailed me and requested code example and explanation of solving Producer Consumer problem in Java  with [wait and notify method](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html)as well, Since its often asked as one of the top [coding question in Java](http://java67.blogspot.sg/2012/08/10-java-coding-interview-questions-and.html). In this Java tutorial, I have put the code example of wait notify version of earlier producer consumer concurrency design pattern. You can see this is much longer code with explicit handling blocking conditions like when shared queue is full and when queue is empty. Since we have replaced[BlockingQueue](http://javarevisited.blogspot.sg/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html) with Vector we need to implement blocking using [wait and notify](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) and that's why we have introduced produce(int i) and consume() method. If you see I have kept consumer thread little slow by allowing it to sleep for 50 Milli second to give an opportunity to producer to fill the queue, which helps to understand that Producer thread is also waiting when Queue is full.

## Java program to solve Producer Consumer Problem in Java

[How to solve Producer Consumer Problem in Java with Example](http://1.bp.blogspot.com/-_GCqP1vu06Q/UBaTOZM869I/AAAAAAAAAaw/ykubu9U9kK4/s1600/java_logo_50_50.jpg)Here is complete Java program to solve producer consumer problem in Java programming language. In this program we have used wait and notify method from java.lang.Object class instead of using BlockingQueue for flow control.

**import** java.util.Vector;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
/\*\*  
 \* **Java program to solve Producer Consumer problem using wait and notify**  
 \* method in Java. Producer Consumer is also a popular concurrency design pattern.  
 \*  
 \* @author Javin Paul  
 \*/  
**public** **class** ProducerConsumerSolution {  
  
    **public** **static** **void** main(**String** args[]) {  
        **Vector** sharedQueue = **new** **Vector**();  
        **int** size = 4;  
        **Thread** prodThread = **new** **Thread**(**new** Producer(sharedQueue, size), "Producer");  
        **Thread** consThread = **new** **Thread**(**new** Consumer(sharedQueue, size), "Consumer");  
        prodThread.start();  
        consThread.start();  
    }  
}  
  
**class** Producer **implements** **Runnable** {  
  
    **private** **final** **Vector** sharedQueue;  
    **private** **final** **int** SIZE;  
  
    **public** Producer(**Vector** sharedQueue, **int** size) {  
        **this**.sharedQueue = sharedQueue;  
        **this**.SIZE = size;  
    }  
  
    @**Override**  
    **public** **void** run() {  
        for (**int** i = 0; i < 7; i++) {  
            **System**.out.println("Produced: " + i);  
            **try** {  
                produce(i);  
            } **catch** (**InterruptedException** ex) {  
                **Logger**.getLogger(Producer.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
            }  
  
        }  
    }  
  
    **private** **void** produce(**int** i) **throws** **InterruptedException** {  
  
        *//wait if queue is full*  
        while (sharedQueue.size() == SIZE) {  
            **synchronized** (sharedQueue) {  
                **System**.out.println("Queue is full " + **Thread**.currentThread().getName()

                                    + " is waiting , size: " + sharedQueue.size());  
  
                sharedQueue.wait();  
            }  
        }  
  
        *//producing element and notify consumers*  
        **synchronized** (sharedQueue) {  
            sharedQueue.add(i);  
            sharedQueue.notifyAll();  
        }  
    }  
}  
  
**class** Consumer **implements** **Runnable** {  
  
    **private** **final** **Vector** sharedQueue;  
    **private** **final** **int** SIZE;  
  
    **public** Consumer(**Vector** sharedQueue, **int** size) {  
        **this**.sharedQueue = sharedQueue;  
        **this**.SIZE = size;  
    }  
  
    @**Override**  
    **public** **void** run() {  
        while (**true**) {  
            **try** {  
                **System**.out.println("Consumed: " + consume());  
                **Thread**.sleep(50);  
            } **catch** (**InterruptedException** ex) {  
                **Logger**.getLogger(Consumer.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
            }  
  
        }  
    }  
  
    **private** **int** consume() **throws** **InterruptedException** {  
        *//wait if queue is empty*  
        while (sharedQueue.isEmpty()) {  
            **synchronized** (sharedQueue) {  
                **System**.out.println("Queue is empty " + **Thread**.currentThread().getName()

                                    + " is waiting , size: " + sharedQueue.size());  
  
                sharedQueue.wait();  
            }  
        }  
  
        *//Otherwise consume element and notify waiting producer*  
        **synchronized** (sharedQueue) {  
            sharedQueue.notifyAll();  
            **return** (**Integer**) sharedQueue.remove(0);  
        }  
    }  
}  
  
**Output:**  
Produced: 0  
**Queue** is empty Consumer is waiting , size: 0  
Produced: 1  
Consumed: 0  
Produced: 2  
Produced: 3  
Produced: 4  
Produced: 5  
**Queue** is full Producer is waiting , size: 4  
Consumed: 1  
Produced: 6  
**Queue** is full Producer is waiting , size: 4  
Consumed: 2  
Consumed: 3  
Consumed: 4  
Consumed: 5  
Consumed: 6  
**Queue** is empty Consumer is waiting , size: 0

# How to Work With wait(), notify() and notifyAll() in Java?

January 8, 2015 by Lokesh Gupta

[**Multithreading**](https://howtodoinjava.com/category/core-java/multi-threading/) in java is pretty complex topic and requires a lot of attention while writing application code dealing with multiple threads accessing one/more shared resources at any given time. Java 5, introduced some classes like [**BlockingQueue**](https://howtodoinjava.com/java-5/how-to-use-blockingqueue-and-threadpoolexecutor-in-java/) and **Executors** which take away some of the complexity by providing easy to use APIs. Programmers using these classes will feel a lot more confident than programmers directly handling synchronization stuff using **wait() and notify()**method calls. I will also recommend to use these newer APIs over synchronization yourself, BUT many times we are required to do so for various reasons e.g. maintaining legacy code. A good knowledge around these methods will help you in such situation when arrived. In this tutorial, I am discussing some **concepts around methods wait(), notify() and notifyAll()**.

## What are wait(), notify() and notifyAll() methods?

Before moving into concepts, lets note down few very basic definitions involved for these methods.

The Object class in Java has three final methods that allow threads to communicate about the locked status of a resource. These are :

1. **wait()** : It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify(). The wait() method releases the lock prior to waiting and reacquires the lock prior to returning from the wait() method. The wait() method is actually tightly integrated with the synchronization lock, using a feature not available directly from the synchronization mechanism. In other words, it is not possible for us to implement the wait() method purely in Java: **it is a native method**.

General syntax for calling wait() method is like this:

|  |
| --- |
| synchronized( lockObject )  {      while( ! condition )      {          lockObject.wait();      }        //take the action here;  } |

1. **notify()** : It wakes up one single thread that called wait() on the same object. It should be noted that calling notify() does not actually give up a lock on a resource. It tells a waiting thread that that thread can wake up. However, the lock is not actually given up until the notifier’s synchronized block has completed. So, if a notifier calls notify() on a resource but the notifier still needs to perform 10 seconds of actions on the resource within its synchronized block, the thread that had been waiting will need to wait at least another additional 10 seconds for the notifier to release the lock on the object, even though notify() had been called.

General syntax for calling notify() method is like this:

|  |
| --- |
| synchronized(lockObject)  {      //establish\_the\_condition;        lockObject.notify();        //any additional code if needed  } |

1. **notifyAll()** : It wakes up all the threads that called wait() on the same object. The highest priority thread will run first in most of the situation, though not guaranteed. Other things are same as notify()method above.

General syntax for calling notify() method is like this:

|  |
| --- |
| synchronized(lockObject)  {      establish\_the\_condition;        lockObject.notifyAll();  } |

In general, a thread that uses the wait() method confirms that a condition does not exist (typically by checking a variable) and then calls the wait() method. When another thread establishes the condition (typically by setting the same variable), it calls the notify() method. The wait-and-notify mechanism does not specify what the specific condition/ variable value is. It is on developer’s hand to specify the condition to be checked before calling wait() or notify().

So far, we learned few basic things which you probably already knew. Let’s write a small program to understand how these methods should be used to get desired results.

## How to Use with wait(), notify() and notifyAll() Methods

In this exercise, we will solve **producer consumer problem** using wait() and notify() methods. To keep program simple and to keep focus on usage of wait() and notify() methods, we will involve only one producer and one consumer thread.

Other features of the program are :

1) Producer thread produce a new resource in every 1 second and put it in ‘taskQueue’.  
2) Consumer thread takes 1 seconds to process consumed resource from ‘taskQueue’.  
3) Max capacity of taskQueue is 5 i.e. maximum 5 resources can exist inside ‘taskQueue’ at any given time.  
4) Both threads run infinitely.

#### Designing Producer Thread

Below is the code for producer thread based on our requirements :

|  |
| --- |
| class Producer implements Runnable  {     private final List<Integer> taskQueue;     private final int           MAX\_CAPACITY;       public Producer(List<Integer> sharedQueue, int size)     {        this.taskQueue = sharedQueue;        this.MAX\_CAPACITY = size;     }       @Override     public void run()     {        int counter = 0;        while (true)        {           try           {              produce(counter++);           }           catch (InterruptedException ex)           {              ex.printStackTrace();           }        }     }       private void produce(int i) throws InterruptedException     {        synchronized (taskQueue)        {           while (taskQueue.size() == MAX\_CAPACITY)           {              System.out.println("Queue is full " + Thread.currentThread().getName() + " is waiting , size: " + taskQueue.size());              taskQueue.wait();           }             Thread.sleep(1000);           taskQueue.add(i);           System.out.println("Produced: " + i);           taskQueue.notifyAll();        }     }  } |

1) Here “produce(counter++)” code has been written inside infinite loop so that producer keeps producing elements at regular interval.  
2) We have written the produce() method code following the general guideline to write wait() method as mentioned in first section.  
3) Once the wait() is over, producer add an element in taskQueue and called notifyAll() method. Because the last-time wait() method was called by consumer thread (that’s why producer is out of waiting state), consumer gets the notification.  
4) Consumer thread after getting notification, if ready to consume the element as per written logic.  
5) Note that both threads use sleep() methods as well for simulating time delays in creating and consuming elements.

#### Designing Consumer Thread

Below is the code for consumer thread based on our requirements :

|  |
| --- |
| class Consumer implements Runnable  {     private final List<Integer> taskQueue;       public Consumer(List<Integer> sharedQueue)     {        this.taskQueue = sharedQueue;     }       @Override     public void run()     {        while (true)        {           try           {              consume();           } catch (InterruptedException ex)           {              ex.printStackTrace();           }        }     }       private void consume() throws InterruptedException     {        synchronized (taskQueue)        {           while (taskQueue.isEmpty())           {              System.out.println("Queue is empty " + Thread.currentThread().getName() + " is waiting , size: " + taskQueue.size());              taskQueue.wait();           }           Thread.sleep(1000);           int i = (Integer) taskQueue.remove(0);           System.out.println("Consumed: " + i);           taskQueue.notifyAll();        }     }  } |

1) Here “consume()” code has been written inside infinite loop so that consumer keeps consuming elements whenever it finds something in taskQueue..  
2) Once the wait() is over, consumer removes an element in taskQueue and called notifyAll() method. Because the last-time wait() method was called by producer thread (that’s why producer is in waiting state), producer gets the notification.  
3) Producer thread after getting notification, if ready to produce the element as per written logic.

#### Test the Application

Now lets test producer and consumer threads.

|  |
| --- |
| public class ProducerConsumerExampleWithWaitAndNotify  {     public static void main(String[] args)     {        List<Integer> taskQueue = new ArrayList<Integer>();        int MAX\_CAPACITY = 5;        Thread tProducer = new Thread(new Producer(taskQueue, MAX\_CAPACITY), "Producer");        Thread tConsumer = new Thread(new Consumer(taskQueue), "Consumer");        tProducer.start();        tConsumer.start();     }  }    Output:    Produced: 0  Consumed: 0  Queue is empty Consumer is waiting , size: 0  Produced: 1  Produced: 2  Consumed: 1  Consumed: 2  Queue is empty Consumer is waiting , size: 0  Produced: 3  Produced: 4  Consumed: 3  Produced: 5  Consumed: 4  Produced: 6  Consumed: 5  Consumed: 6  Queue is empty Consumer is waiting , size: 0  Produced: 7  Consumed: 7  Queue is empty Consumer is waiting , size: 0 |

I will suggest you to change the time taken by producer and consumer threads to different times, and check the different outputs in different scenario.

## Interview Questions on wait(), notify() and notifyAll() Methods

#### What happens when notify() is called and no thread is waiting?

In general practice, this will not be the case in most scenarios if these methods are used correctly. Though if the notify() method is called when no other thread is waiting, notify() simply returns and the notification is lost.

Since the wait-and-notify mechanism does not know the condition about which it is sending notification, it assumes that a notification goes unheard if no thread is waiting. A thread that later executes the wait() method has to wait for another notification to occur.

#### Can there be a race condition during the period that the wait() method releases OR reacquires the lock?

The wait() method is tightly integrated with the lock mechanism. The object lock is not actually freed until the waiting thread is already in a state in which it can receive notifications. It means only when thread state is changed such that it is able to receive notifications, lock is held. The system prevents any race conditions from occurring in this mechanism.

Similarly, system ensures that lock should be held by object completely before moving the thread out of waiting state.

#### If a thread receives a notification, is it guaranteed that the condition is set correctly?

Simply, no. Prior to calling the wait() method, a thread should always test the condition while holding the synchronization lock. Upon returning from the wait() method, the thread should always retest the condition to determine if it should wait again. This is because another thread can also test the condition and determine that a wait is not necessary — processing the valid data that was set by the notification thread.

This is a common case when multiple threads are involved in the notifications. More particularly, the threads that are processing the data can be thought of as consumers; they consume the data produced by other threads. There is no guarantee that when a consumer receives a notification that it has not been processed by another consumer. As such, when a consumer wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but the state may have been changed after the notify() method was called and before the consumer thread woke up. Waiting threads must provide the option to check the state and to return back to a waiting state in case the notification has already been handled. This is why we always put calls to the wait() method in a loop.

#### What happens when more than one thread is waiting for notification? Which threads actually get the notification when the notify() method is called?

It depends on many factors.Java specification doesn’t define which thread gets notified. In runtime, which thread actually receives the notification varies based on several factors, including the implementation of the Java virtual machine and scheduling and timing issues during the execution of the program. There is no way to determine, even on a single processor platform, which of multiple threads receives the notification.

Just like the notify() method, the notifyAll() method does not allow us to decide which thread gets the notification: they all get notified. When all the threads receive the notification, it is possible to work out a mechanism for the threads to choose among themselves which thread should continue and which thread(s) should call the wait() method again.

#### Does the notifyAll() method really wake up all the threads?

Yes and no. All of the waiting threads wake up, but they still have to reacquire the object lock. So the threads do not run in parallel: they must each wait for the object lock to be freed. Thus, only one thread can run at a time, and only after the thread that called the notifyAll() method releases its lock.

#### Why would you want to wake up all of the threads if only one is going to execute at all?

There are a few reasons. For example, there might be more than one condition to wait for. Since we cannot control which thread gets the notification, it is entirely possible that a notification wakes up a thread that is waiting for an entirely different condition. By waking up all the threads, we can design the program so that the threads decide among themselves which thread should execute next. Another option could be when producers generate data that can satisfy more than one consumer. Since it may be difficult to determine how many consumers can be satisfied with the notification, an option is to notify them all, allowing the consumers to sort it out among themselves.

Read more: <http://www.java67.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html#ixzz4i1U1kOKi>

# Top 12 Java Thread, Concurrency and Multithreading Interview Questions For experienced

**Java concurrency and thread interview questions answers**  
Multithreading is an important feature of Java programming language, which means threads are also an important part of any Java interview. It's true and in fact at beginners and freshers level [Thread interview questions in Java](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html) are one of most difficult to answer. One reason for interview question related to multithreading and concurrency being difficult is confusion around how multiple threads works together and second is threads are genuinely a complicated topic to understand and use correctly. Mostly *thread interview questions* checks Java programmers knowledge on Java Thread API, Java concurrency API, issues related to multi-threading like a [race condition](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html), [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html), and [deadlock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html). Some time multithreading and concurrency interview question also focus on parallel design patterns like solving the producer-consumer problem, implementing work steal pattern or solving dining philosopher problem in Java. This is especially true while interviewing experienced Java developers with 4 to 6 years of experience.  
  
In this article, we will take a look at different kinds of **multithreading and concurrency questions asked in various interviews**e.g. on telephonic or face to face interview, on written test, to both experienced and senior Java developers, and some tips to answer them correctly.  
  
Questions asked on [telephonic](http://java67.blogspot.com/2015/03/top-40-core-java-interview-questions-answers-telephonic-round.html) or the [first round of interviews](http://java67.blogspot.com/2014/07/21-frequently-asked-java-interview-questions-answers.html) are tend to be easier and you must answer them to the point with a keyword, which interviewer is expecting. On the face to face interview, be prepare for different kinds of follow-up questions. Books like [Java Programming Interview exposed](http://www.amazon.com/Java-Programming-Interviews-Exposed-Markham/dp/1118722868?tag=javamysqlanta-20) also helps you to prepare better and answer to the point.

## Java Interview questions on Concurrency and multithreading

As I said, in this Java article, not only,  I will share some of the most commonly asked thread interview questions at freshers and beginners level, like up to 2 to 4 years of experience, and some tips and tricks to answer them correctly.  
  
By the way, these thread interview questions are equally useful for senior Java developers or guys with some Java experience in hand. I have tried to share answers to these interview questions on the thread as well but I suggest you do some research and learn the topic well to answer any follow-up questions, which comes due to your response to these *thread questions in Java*.  
  
Anyway here is my collection of **Java thread interview questions** and how to answers them in Java :

**1) What is the difference between start and run method in Java Thread?**([answer](http://javarevisited.blogspot.com/2014/09/common-java-multi-threading-mistakes-1-run-vs-start.html))  
This thread interview question is also ask as if start() method eventually call run() method then why do you need to call start() method, why not call run() method directly. well, reason is that because start method creates a new thread and call the code written inside the run method on a new thread while calling run method executes that code on the same thread. You can also see the article [start vs run method in Java](http://javarevisited.blogspot.sg/2012/03/difference-between-start-and-run-method.html) for more details.  
  
  
**2) Write code to avoid deadlock in Java where N threads are accessing N shared resources?**([answer](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html))  
This is a classic Java multithreading interview questions, which appears on almost every list of Java thread questions. This question is based on risk and issues faced by parallel programs without proper synchronization or incorrect synchronization. This question explores the concept of looking and best practices on acquiring and releasing the lock on shared resource. By the way, it's been covered in many places as well and I suggest reading  [How to prevent deadlock in Java](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html), not only for detail answer of this Java multithreading question but also to learn how to prevent deadlock in Java.  
  
  
  
**3) Which one is better to implement thread in Java ? extending Thread class or implementing Runnable?**([answer](http://javarevisited.blogspot.com/2012/01/difference-thread-vs-runnable-interface.html))  
Well, this is another frequently asked questions on any Java thread interview. Essentially these are two way to implement Thread in Java, by extending java.lang.Thread class or by implementing java.lang.Runnable interface.  
  
By extending the class you are using your chance to extend one any only one class as Java does not support multiple inheritances, by implementing a Runnable interface you can still extend another class. So extending Runnable or even Callable is a better choice. You can also see [Runnable vs Thread class in Java](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) for more answers on this questions.  
  
Given its simplicity and fact-based nature, this question mostly appears on either telephonic round or initial screening rounds. Key points to mention, while answering this question includes multiple inheritance at the class level and separation of defining a task and execution of a task. Runnable only represent a task, while Thread represent both tasks and it's execution.  
  
  
  
**4) What is Busy Spinning? Why will you use Busy Spinning as wait strategy?**([answer](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html))  
This is one of the advanced concurrency interview questions in Java and only asked to experienced and senior Java developers, with lots of concurrent coding experience under the belt. By the way, the concept of *busy spinning* is not new, but its usage with multi-core processor has risen recently (see [The Art of Multiprocessor Programming](https://www.amazon.com/Art-Multiprocessor-Programming-Revised-Reprint/dp/0123973376?tag=javamysqlanta-20)).

# The busy waiting is a wait strategy, where one thread wait for a condition to become true, but instead of calling wait or sleep method and releasing CPU, it just spins. This is particularly useful if the condition is going to be true quite quickly i.e. in a millisecond or microsecond. Theadvantageof not releasing CPU is that all cached data and instruction remain unaffected, which may be lost, had this thread is suspended on one core and brought back to another thread. If you can answer this question, that rest assure of a good impression. 5) What is the difference betweenCountDownLatchandCyclicBarrierin Java?([answer](http://java67.blogspot.com/2012/08/difference-between-countdownlatch-and-cyclicbarrier-java.html)) The [CountDownLatch](http://javarevisited.blogspot.sg/2012/07/countdownlatch-example-in-java.html)and[CyclicBarrier in Java](http://javarevisited.blogspot.sg/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html)are two important concurrency utility which is added on Java 5 Concurrency API. Both are used to implement scenario, where one thread has to wait for other thread before starting processing but there is a difference between them. The key point to mention, while answering this question is that CountDownLatch is not reusable once the count reaches to zero, while CyclicBarrier can be reused even after the barrier is broken. You can also see my previous article[difference between CyclicBarrier and CountDownLatch in Java](http://java67.blogspot.com/2012/08/difference-between-countdownlatch-and-cyclicbarrier-java.html)for a more detailed answer of this concurrency interview question and a real life example of where to use these concurrency utilities. CountDownLatch in Java

[**2**](https://www.geeksforgeeks.org/easy/)

CountDownLatch is used to make sure that a task waits for other threads before it starts. To understand its application, let us consider a server where the main task can only start when all the required services have started.

**Working of CountDownLatch:**  
When we create an object of CountDownLatch, we specify the number if threads it should wait for, all such thread are required to do count down by calling CountDownLatch.countDown() once they are completed or ready to the job. As soon as count reaches zero, the waiting task starts running.

**Example of CountDownLatch in JAVA:**

|  |
| --- |
| /\* Java Program to demonstrate how to use CountDownLatch,     Its used when a thread needs to wait for other threads     before starting its work. \*/  import java.util.concurrent.CountDownLatch;    public class CountDownLatchDemo  {      public static void main(String args[]) throws InterruptedException      {          // Let us create task that is going to wait for four          // threads before it starts          CountDownLatch latch = new CountDownLatch(4);            // Let us create four worker threads and start them.          Worker first = new Worker(1000, latch, "WORKER-1");          Worker second = new Worker(2000, latch, "WORKER-2");          Worker third = new Worker(3000, latch, "WORKER-3");          Worker fourth = new Worker(4000, latch, "WORKER-4");          first.start();          second.start();          third.start();          fourth.start();            // The main task waits for four threads          latch.await();            // Main thread has started          System.out.println(Thread.currentThread().getName() +                             " has finished");      }  }    // A class to represent threads for which the main thread  // waits.  class Worker extends Thread  {      private int delay;      private CountDownLatch latch;        public Worker(int delay, CountDownLatch latch,                                       String name)      {          super(name);          this.delay = delay;          this.latch = latch;      }        @Override      public void run()      {          try          {              Thread.sleep(delay);              latch.countDown();              System.out.println(Thread.currentThread().getName()                                 + " finished");          }          catch (InterruptedException e)          {              e.printStackTrace();          }      }  } |

Run on IDE

**Output:**

WORKER-1 finished

WORKER-2 finished

WORKER-3 finished

WORKER-4 finished

main has finished

**Facts about CountDownLatch:**

1. Creating an object of CountDownLatch by passing an int to its constructor (the count), is actually number of invited parties (threads) for an event.
2. The thread, which is dependent on other threads to start processing, waits on until every other thread has called count down. All threads, which are waiting on await() proceed together once count down reaches to zero.
3. countDown() method decrements the count and await() method blocks until count == 0

Java.util.concurrent.CyclicBarrier in Java

[**4.5**](https://www.geeksforgeeks.org/hard/)

CyclicBarrier is used to make threads wait for each other. It is used when different threads process a part of computation and when all threads have completed the execution, the result needs to be combined in the parent thread. In other words, a CyclicBarrier is used when multiple thread carry out different sub tasks and the output of these sub tasks need to be combined to form the final output. After completing its execution, threads call await() method and wait for other threads to reach the barrier. Once all the threads have reached, the barriers then give the way for threads to proceed.

**Working of CyclicBarrier**

CyclicBarriers are defined in java.util.concurrent package. First a new instance of a CyclicBarriers is created specifying the number of threads that the barriers should wait upon.

**CyclicBarrier newBarrier = new CyclicBarrier(numberOfThreads);**

Each and every thread does some computation and after completing it’s execution, calls await() methods as shown:

public void run()

{

// thread does the computation

newBarrier.await();

}

**Working of CyclicBarrier:**

https://contribute.geeksforgeeks.org/wp-content/uploads/cyclicbarrier.png  
Once the number of threads that called await() equals **numberOfThreads**, the barrier then gives a way for the waiting threads. The CyclicBarrier can also be initialized with some action that is performed once all the threads have reached the barrier. This action can combine/utilize the result of computation of individual thread waiting in the barrier.

Runnable action = ...

//action to be performed when all threads reach the barrier;

CyclicBarrier newBarrier = new CyclicBarrier(numberOfThreads, action);

**Important Methods of CyclicBarrier:**

1. **getParties:** Returns the number of parties required to trip this barrier.  
   **Syntax:**

public int getParties()

**Returns:**  
the number of parties required to trip this barrier

1. **reset:** Resets the barrier to its initial state.  
   **Syntax:**

public void reset()

**Returns:**  
void but resets the barrier to its initial state. If any parties are currently waiting at the barrier, they will return with a BrokenBarrierException.

1. **isBroken:** Queries if this barrier is in a broken state.  
   **Syntax:**

public boolean isBroken()

**Returns:**  
true if one or more parties broke out of this barrier due to interruption or timeout since construction or the last reset, or a barrier action failed due to an exception; false otherwise.

1. **getNumberWaiting:** Returns the number of parties currently waiting at the barrier.  
   **Syntax:**

public int getNumberWaiting()

**Returns:**  
the number of parties currently blocked in await()

1. **await:** Waits until all parties have invoked await on this barrier.  
   **Syntax:**

public int await() throws InterruptedException, BrokenBarrierException

**Returns:**  
the arrival index of the current thread, where index getParties() – 1 indicates the first to arrive and zero indicates the last to arrive.

1. **await:** Waits until all parties have invoked await on this barrier, or the specified waiting time elapses.  
   **Syntax:**
2. public int await(long timeout, TimeUnit unit)
3. throws InterruptedException,

BrokenBarrierException, TimeoutException

**Returns:**  
the arrival index of the current thread, where index getParties() – 1 indicates the first to arrive and zero indicates the last to arrive

|  |
| --- |
| //JAVA program to demonstrate execution on Cyclic Barrier    import java.util.concurrent.TimeUnit;  import java.util.concurrent.TimeoutException;  import java.util.concurrent.BrokenBarrierException;  import java.util.concurrent.CyclicBarrier;    class Computation1 implements Runnable  {      public static int product = 0;      public void run()      {          product = 2 \* 3;          try          {              Tester.newBarrier.await();          }          catch (InterruptedException | BrokenBarrierException e)          {              e.printStackTrace();          }      }  }    class Computation2 implements Runnable  {      public static int sum = 0;      public void run()      {          // check if newBarrier is broken or not          System.out.println("Is the barrier broken? - " + Tester.newBarrier.isBroken());          sum = 10 + 20;          try          {              Tester.newBarrier.await(3000, TimeUnit.MILLISECONDS);                // number of parties waiting at the barrier              System.out.println("Number of parties waiting at the barrier "+              "at this point = " + Tester.newBarrier.getNumberWaiting());          }          catch (InterruptedException | BrokenBarrierException e)          {              e.printStackTrace();          }          catch (TimeoutException e)          {              e.printStackTrace();          }      }  }      public class Tester implements Runnable  {      public static CyclicBarrier newBarrier = new CyclicBarrier(3);        public static void main(String[] args)      {          // parent thread          Tester test = new Tester();            Thread t1 = new Thread(test);          t1.start();      }      public void run()      {          System.out.println("Number of parties required to trip the barrier = "+          newBarrier.getParties());          System.out.println("Sum of product and sum = " + (Computation1.product +          Computation2.sum));            // objects on which the child thread has to run          Computation1 comp1 = new Computation1();          Computation2 comp2 = new Computation2();            // creation of child thread          Thread t1 = new Thread(comp1);          Thread t2 = new Thread(comp2);            // moving child thread to runnable state          t1.start();          t2.start();            try          {              Tester.newBarrier.await();          }          catch (InterruptedException | BrokenBarrierException e)          {              e.printStackTrace();          }            // barrier breaks as the number of thread waiting for the barrier          // at this point = 3          System.out.println("Sum of product and sum = " + (Computation1.product +          Computation2.sum));            // Resetting the newBarrier          newBarrier.reset();          System.out.println("Barrier reset successful");      }  } |

Run on IDE

**Output:**

<Number of parties required to trip the barrier = 3

Sum of product and sum = 0

Is the barrier broken? - false

Number of parties waiting at the barrier at this point = 0

Sum of product and sum = 36

Barrier reset successful

**Explanation:** The value of (sum + product) = 0 is printed on the console because the child thread has’t yet ran to set the values of sum and product variable. Following this, (sum + product) = 36 is printed on the console because the child threads ran setting the values of sum and product. Furthermore, the number of waiting thread on the barrier reached 3, due to which the barrier then allowed all thread to pass and finally 36 was printed. The value of “Number of parties waiting at the barrier at this point” = 0 because all the three threads had already called await() method and hence, the barrier is no longer active. In the end, newBarrier is reset and can be used again.

**BrokenBarrierException**

A barrier breaks when any of the waiting thread leaves the barrier. This happens when one or more waiting thread is interrupted or when the waiting time is completed because the thread called the await() methods with a timeout as follows:

newBarrier.await(1000, TimeUnit.MILLISECONDS);

// thread calling this await()

// methods waits for only 1000 milliseconds.

When the barrier breaks due to one of more participating threads, the await() methods of all the other threads throws a BrokenThreadException. Whereas, the threads that are already waiting in the barriers have their await() call terminated.

**Difference between a CyclicBarrier and a CountDownLatch**

* A CountDownLatch can be used only once in a program(until it’s count reaches 0).
* A CyclicBarrier can be used again and again once all the threads in a barriers is released.

[CountDownLatch](http://codepumpkin.com/?p=895) Vs [CyclicBarrier](http://codepumpkin.com/?p=935): Though both are used as a synchronization aid that allows one or more threads to wait but there are certain differences between them that you should know in order to know when one of these utilities will serve you better.

As per the [java.util.concurrent API](http://download.oracle.com/javase/6/docs/api/java/util/concurrent/package-summary.html),

* CountDownLatch: A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads complete.
* CyclicBarrier: A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point.

Here are few basic differences.

**1.**In CountDownLatch**,**onlymain thread waits for other threads to complete their execution, where as in CyclicBarrier, Each worker threads wait for each other to complete their execution. Let's understand this by following Example :

* **CountDownLatch:**

Consider a IT world scenario where manager divided modules between development teams (A and B) and he wants to assign it to QA team for testing only when both the teams completes their task.

* Here manager thread works as main thread and development team works as worker thread. Manager thread waits for development teams thread to complete their task. Once developer teams complete their tasks, they will inform manager thread and then manager thread assign modules to QA team.
* **CyclicBarrier:**

Consider the same scenario where manager divided modules between development teams (A and B). He goes on leave. He asked both teams to wait for each other to complete their respective taskand  once both teams are done, assign it to QA team for testing.

* Here manager thread works as main thread and development team works as worker thread. Development team threads wait for other development team threads after completing their task.

In Other words, a CountDownLatch initialized to N can be used to make one thread wait until N threads have completed some action. Where as if CyclicBarrier has been initialized to 3 then you should have at least 3 threads to call await().

**2.** **Reusability :**We can not **reuse**same CountDownLatch instance once count reaches to zero and latch is open. CyclicBarrier can be reused after all the waiting threads are released.

**3. barrierAction**:  A  CyclicBarrier  supports an optional Runnable command that is run once per barrier point, after the last thread in the party arrives, but before any threads are released.

So with CyclicBarrier you have an option to have an **Action class specified** in the CyclicBarrier constructor that will be run after the last thread has called await(). This barrier action is useful for updating shared-state before any of the parties continue.

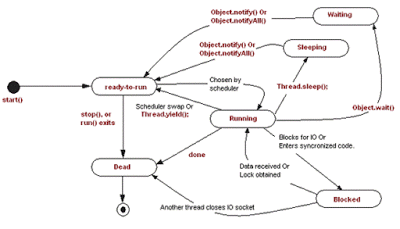
|  |  |
| --- | --- |
| 1 | public CyclicBarrier(int parties, Runnable barrierAction) |

CountDownLatch doesn't provide any such constructor to specify a runnable action.

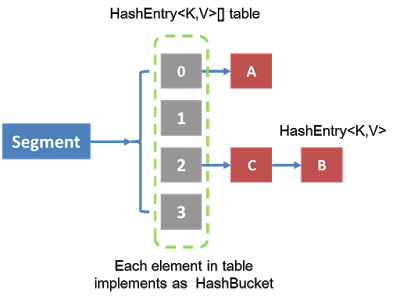
**Summary**

|  | **COUNT DOWN LATCH** | **CYCLIC BARRIER** |
| --- | --- | --- |
| 1 | Main threads waits for other threads to complete their execution. | Each worker threads wait for each other to complete their execution |
| 2 | We can not reuse same CountDownLatch instance once count reaches to zero and latch is open. | CyclicBarrier can be reused by resetting Barrier, Once barrier is broken. |
| 3 | No Such Action can be provided with CountDownLatch | We can provide barrierAction that is run once per barrier point, after the last thread in the party arrives, but before any threads are released. |

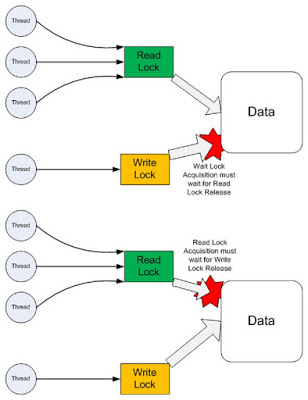
**6) What is the difference between wait and sleep in Java?**([method](http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html))  
One more *classic Java multithreading question* from the telephonic round of interviews. The key point to mention while answering this question is to mention that wait will release the lock and must be called from the synchronized context, while sleep will only pause the thread for some time and keep the lock.  
  
By the way, both methods throw IntrupptedException and can be interrupted, which can lead to some follow-up questions like, can we awake a sleeping or waiting for a thread in Java? You can also read a detailed answer on my post of same title [here](http://java67.blogspot.com/2012/08/what-are-difference-between-wait-and.html).

[](https://2.bp.blogspot.com/-kJOX8eUPO80/V10_YpHWQYI/AAAAAAAAGNg/6x_PZfACG4wwiacTyn25tkE3B7TO85P0QCLcB/s1600/Sleep+vs+wait+vs+yield+method+java.gif)

**7) How do you solve producer consumer problem in Java?**([solution](http://java67.blogspot.com/2015/12/producer-consumer-solution-using-blocking-queue-java.html))  
One of my favorite questions during any Java multithreading interview, Almost half of the concurrency problems can be categorized in the producer-consumer pattern. There are basically two ways to solve this problem in Java, One by [using wait and notify method](http://java67.blogspot.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html) and other by using [BlockingQueue](http://javarevisited.blogspot.com/2014/06/synchronousqueue-example-in-java.html) in Java.  later is easy to implement and a good choice if you are coding in Java 5.  
  
The key points to mention, while answering this question is thread-safety and blocking nature of BlockingQueue and how that helps, while writing concurrent code. You can also expect lots of follow-up questions including, what happen if you have multiple producer or multiple consumers, what will happen if a producer is faster than consumer thread or vice-versa. You can also see this link for example of [how to code producer-consumer design in Java using blocking queue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html)  
  
  
  
**8) Why is ConcurrentHashMap faster than Hashtable in Java?**([answer](http://javarevisited.blogspot.com/2016/05/what-is-difference-between-synchronized.html))  
ConcurrentHashMap is introduced as an alternative of Hashtable in Java 5, it is faster because of its design. ConcurrentHashMap divides the whole map into different segments and only lock a particular segment during the update operation, instead of [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html), which locks whole Map.  
  
The ConcurrentHashMap also provides lock-free read, which is not possible in Hashtable, because of this and lock striping, ConcurrentHashMap is faster than Hashtable, especially when a number of the reader is more than the number of writers.  
  
In order to better answer this popular Java concurrency interview questions, I suggest reading my post about the [internal working of ConcurrentHashMap in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html).

[](https://1.bp.blogspot.com/-s28yqJthVWo/V10_G31sSCI/AAAAAAAAGNY/8oJE5cDD5BUa5VBiCI8Bx-UdpD8_MoeSwCLcB/s1600/Internal+implementation+of+ConcurrentHashMap+in+Java.png)

**9) What is the difference between submit() and execute() method of Executor and ExecutorService in Java? (**[answer](http://javarevisited.blogspot.com/2016/04/difference-between-ExecutorServie-submit-vs-Executor-execute-method-in-Java.html)**)**  
The main difference between submit and execute method from ExecutorService interface is that former return a result in the form of a Future object, while later doesn't return a result. By the way, both are used to submit a task to thread pool in Java but one is defined in Executor interface,while other is added into ExecutorService interface. This multithreading interview question is also asked in the first round of Java interviews.  
  
  
**10) How do you share data between two threads in Java? (**[answer](http://javarevisited.blogspot.com/2013/12/inter-thread-communication-in-java-wait-notify-example.html)**)**  
One more Java multithreading question from the telephonic round of interview. You can share data between thread by using shared object or shared data structures like Queue. Depending upon, what you are using, you need to provide the thread-safety guarantee, and one way of providing thread-safety is using synchronized keyword.  
  
If you use concurrent collection classes from Java 5 e.g. [BlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html), you can easily share data without being bothered about thread safety and inter-thread communication. I like this thread question, because of it's simplicity and effectiveness. This also leads further follow-up questions on issues which arise due to sharing data between threads e.g. race conditions.  
  
  
**11) What is ReentrantLock in Java? Have you used it before? (**[answer](http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html)**)**  
ReentrantLock is an alternative of synchronized keyword in Java, it is introduced to handle some of the limitations of synchronized keywords. Many concurrency utility classes and concurrent collection classes from Java 5, including ConcurrentHashMap uses ReentrantLock, to leverage optimization.  
  
The ReentrantLock mostly uses an atomic variable and faster CAS operation to provides better performance. Key points to mention are the [difference between ReentrantLock and synchronized keyword in Java](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html), which includes the ability to acquire lock interruptibly, timeout feature while waiting for lock etc. ReentrantLock also gives the option to create fair lock in Java.Once again a very good Java concurrency interview question for experienced Java programmers.  
  
  
**12) What is ReadWriteLock in Java? What is the benefit of using ReadWriteLock in Java?(**[answer](http://javarevisited.blogspot.com/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html)**)**  
This is usually a follow-up question of previous Java concurrency questions. The  ReadWriteLock is again based upon the concept of lock striping, one of the advance thread-safety mechanism which advocates separating locks for reading and writing operations (see [Concurrent Programming in Java: Patterns and principles](https://www.amazon.com/Concurrent-Programming-Java%C2%99-Principles-Pattern/dp/0201310090?tag=javamysqlanta-20) by Doug Lea for more details).  
  
If you have noticed before, reading operation can be done without locking if there is no writer and that can hugely improve the performance of any application. The ReadWriteLock leverage this idea and provide policies to allow maximum concurrency level. Java Concurrency API also provides an implementation of this concept as ReentrantReadWriteLock.  
  
Depending upon Interviewer and experience of the candidate, you can even expect to provide your own implementation of ReadWriteLock, so be prepare for that as well.  
  
Here is a nice diagram which clearly explains working of read-write lock in Java:

[](https://4.bp.blogspot.com/-yYqp8m0BGdk/V11A-s0qwVI/AAAAAAAAGNw/LL_qGX5_qD4dHTFMe3kklI4jYWyDHzIyQCLcB/s1600/ReadWriteLock+in+Java.jpg)

These were some of my favorite*interview questions based on multithreading and concurrent in Java*. Threading and Concurrency is a big topic in Java and has lots of interesting, [tricky and tough question](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html) but for starters and freshers, these questions certainly help to clear any thread interview in Java.   
  
As I said, mentioning the key points are very important while answering questions on multithreading and concurrency.  I also suggest further reading [Java Concurrency in Practice](http://www.amazon.com/dp/0321349601/?tag=javamysqlanta-20) to learn more about locking, synchronization, concurrent collections and concurrency utility classes to do well in core Java and multithreading interviews.

Read more: <http://www.java67.com/2012/08/5-thread-interview-questions-answers-in.html#ixzz4i1Uzb1Nr>

# [5 Difference between StringBuffer, StringBuilder and String in Java](http://www.java67.com/2016/10/5-difference-between-stringbuffer.html)

Though all three classes StringBuffer, StringBuilder and String are used for representing text data in Java there are some significant differences between them. One of the most notable differences between StringBuilder, StringBuffer, and String in Java is that both StringBuffer and StrinBuilder are **Mutable**class but [String is Immutable in Java](http://www.java67.com/2014/01/why-string-class-has-made-immutable-or-final-java.html). What this means is, you can add, remove or replace characters from StringBuffer and StringBuilder object but any change on String object e.g. converting uppercase to lowercase or appending a new character using [String concatenation](http://www.java67.com/2015/05/4-ways-to-concatenate-strings-in-java.html) will always result in a new String object. Another key difference between them is that both StringBuffer and String are **thread-safe** but StringBuilder is not thread-safe in Java. String achieves its thread-safety from Immutability but StringBuffer achieves it via synchronization, which is also the main difference between the StringBuffer and StringBuilder in Java.

[Read more »](http://www.java67.com/2016/10/5-difference-between-stringbuffer.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/10/5-difference-between-stringbuffer.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4996537153844234988)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4996537153844234988)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4996537153844234988&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4996537153844234988&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4996537153844234988&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4996537153844234988&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4996537153844234988&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java String tutorial](http://www.java67.com/search/label/Java%20String%20tutorial?&max-results=3)

# [Difference between Heap and Stack Memory in Java JVM](http://www.java67.com/2016/10/difference-between-heap-and-stack-memory-in-java-JVM.html)

One of the many traits of a good programmer is how well he understands the fundamental and if you want to check the fundamentals of Java programmer than asking the difference between heap and stack memory is a good choice. Even though both are part of JVM and both consumers memory allocated to the Java process, there are many differences between them e.g. [Heap memory](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) is shared by all threads of Java application but Stack memory is local to each thread. Objects are created in heap memory but method frames are stored in [Stack memory](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html), and size of heap space is much bigger than the small size of Stack in Java. Even if you know this much information about heap and stack in Java, you are one of the better candidates, but let's see some more details to impress the interviewer.

[Read more »](http://www.java67.com/2016/10/difference-between-heap-and-stack-memory-in-java-JVM.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/10/difference-between-heap-and-stack-memory-in-java-JVM.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8211796334594716831)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8211796334594716831)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8211796334594716831&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8211796334594716831&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8211796334594716831&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8211796334594716831&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8211796334594716831&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3)

# [5 Difference between Iterator and ListIterator in Java?](http://www.java67.com/2016/09/5-difference-between-iterator-and-ListIterator-in-java.html)

The Iterator is the standard way to traverse a collection in Java. You can use Iterator to traverse a List, Set, Map, Stack, Queue or any Collection, but you might not know that there is another way to traverse over List in Java? Yes, it's called the ListIterator. There are many differences between Iterator and ListIterator in Java, but the most significant of them is that [Iterator](http://www.java67.com/2013/02/java-iterator-example.html) only allows you to traverse in **one direction** i.e. forward, you have just got a next() method to get the next element, there is no previous() method to get the previous element. On the other hand, ListIterator allows you to traverse the list in **both directions** i.e. forward and backward. It has got both next() and previous() method to access the next and previous element from List.

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?max-results=3#ixzz4i1VaRkQz>

# [Difference between HashMap vs IdentityHashMap in Java?](http://www.java67.com/2016/08/difference-between-hashmap-and-IdentityHashMap-in-java.html)

The IdentityHashMap is one of the lesser known Map implementation from JDK. Unlike general purposes Map implementations like [HashMap](http://www.java67.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) and [LinkedHashMap](http://www.java67.com/2012/08/difference-between-hashmap-and-LinkedHashMap-Java.html), it is very special and it's internal working is quite different than HashMap. The main difference between IdentityHashMap and HashMap in Java is that former uses equality operator (==) instead of equals() method to compare keys. Which means you need the same key object to retrieve the value from IdentityHashMap, you cannot retrieve values by using another key which is logically equal to previous key. Another important difference between HashMap and IdentityHashMap is that IdentityHashMap doesn't use [hashCode()](http://www.java67.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) method instead it uses **System.identityHashCode()** method. This is a significant difference because now you can use *mutable objects* as key in Map whose hash code are likely to change when the mapping is stored inside IdentityHashMap.

[Read more »](http://www.java67.com/2016/08/difference-between-hashmap-and-IdentityHashMap-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/08/difference-between-hashmap-and-IdentityHashMap-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5483452647136970228)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5483452647136970228)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5483452647136970228&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5483452647136970228&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5483452647136970228&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5483452647136970228&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5483452647136970228&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java collection tutorial](http://www.java67.com/search/label/Java%20collection%20tutorial?&max-results=3)

# [How to sort array in descending order - Java Example](http://www.java67.com/2016/07/how-to-sort-array-in-descending-order-in-java.html)

It's easy to sort an object array in decreasing or reverse order, just provide the Comparator with opposite order. You can even use Collections.reverseOrder() if you want to sort array in the decreasing order, which returns a reverse Comparator to sort objects in the order opposite of their natural ordering defined by the [compareTo()](http://www.java67.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) method. Unfortunately, for a primitive array, there is no direct way to sort in descending order. The Arrays.sort() method which is used to sort a primitive array in Java doesn't accept a boolean to sort the primitive array in reverse order. You might have seen the error **"no suitable method found for sort(int[],comparator<object>)"** which occurs when programmers try to call the **Arrays.sort(**) method by passing reverse Comparator defined by Collection.reverseOrder(). That will work fine with [Integer array](http://www.java67.com/2014/03/how-to-find-top-two-maximum-number-from-integer-array-java.html) but will not work with an int array. The only way to sort a primitive array in descending order is, first sort the array in ascending order and then reverse the array in place as shown [here](http://javarevisited.blogspot.sg/2015/03/how-to-reverse-array-in-place-in-java.html). This is also true for two-dimensional primitive arrays.

[Read more »](http://www.java67.com/2016/07/how-to-sort-array-in-descending-order-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/07/how-to-sort-array-in-descending-order-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8049387425088596475)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8049387425088596475)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8049387425088596475&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8049387425088596475&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8049387425088596475&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8049387425088596475&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8049387425088596475&target=pinterest)

Labels: [array](http://www.java67.com/search/label/array?&max-results=3), [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3)

# [Difference between final vs finally and finalize in Java? Answer](http://www.java67.com/2016/06/difference-between-final-vs-finally-vs-finalize-in-java.html)

The final, finally, and finalize are one of the three confusing keywords, modifiers and methods in Java. They sounds similar but they are for different purpose. For example, [final keyword](http://java67.blogspot.com/2015/07/how-to-use-final-keyword-in-java-example.html) is a modifier in Java. When you use final keyword with a class it becomes a final class and no one can extend this e.g. [String is final in Java](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html). When you use the final modifier with method than it cannot be overridden in subclass, and when you use the final keyword with variable, it become a constant i.e. its value cannot be changed once assigned. On the other hand, finally is a keyword related to exception handling in Java. It's often used with try block and it's part of try, catch and finally trio. A [finally bloc](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html)k can be used with or without catch block in Java and its guaranteed to be always executed, irrespective of what happens inside try block. This is the reason finally block is used to do cleanup and free resources.

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2016-08-11T08:12:00-07:00&max-results=6#ixzz4i1VkpVp3>

# [5 Difference between BufferedReader and Scanner class in Java - Example](http://www.java67.com/2016/06/5-difference-between-bufferedreader-and-scanner-in-java.html)

Even though both BufferedReader and Scanner can read a file or user input from command prompt in Java, there some significant differences between them. One of the main difference between BufferedReader and Scanner class is that former is meant to just read String while later is meant to both read and parse text data into Java primitive type e.g. int, short, float, double, and long. In other words, *BufferedRedaer can only read* *String* but Scanner can read both String and other data types like int, float, long, double, float etc. This functional difference drives several other differences on their usage. Another difference is Scanner is newer than BufferedReader, only introduced in Java 5, while BufferedReader is present in Java from JDK 1.1 version. This means, you have access to BufferedReader in almost all JDK version mainly Java 1.4 but Scanner is only available after Java 5.  This is also a popular [core Java questions from interviews](http://java67.blogspot.com/2015/03/top-40-core-java-interview-questions-answers-telephonic-round.html). Since many developer lack Java IO skill, questions like this test their knowledge about API and how to do some practical task.

[Read more »](http://www.java67.com/2016/06/5-difference-between-bufferedreader-and-scanner-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/06/5-difference-between-bufferedreader-and-scanner-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4295864714768692260)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4295864714768692260)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4295864714768692260&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4295864714768692260&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4295864714768692260&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4295864714768692260&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4295864714768692260&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [java io tutorial](http://www.java67.com/search/label/java%20io%20tutorial?&max-results=3)

# [Difference between static and nonstatic member variables in Java](http://www.java67.com/2016/05/difference-between-static-and-nonstatic-member-variable-in-java.html)

In the [last article](http://java67.blogspot.com/2016/04/difference-between-static-vs-non-static-method-in-java.html), I had explained about some key difference between static and nonstatic methods in Java, and in this part, I'll explain the difference between static and nonstatic member variables in Java. The concept of static remains same, that doesn't change with method or member variables but there are still some subtle details, which every Java programmer should know and understand. As with [static methods](http://java67.blogspot.com/2014/10/difference-between-static-and-non-static-method-java-programming.html), a static member variable belongs to a class and a non-static member variable belongs to an instance. This means, the **value of a static variable will be same for all instances**, but the value of a non-static variable will be different for different objects. That is also referred as the state of objects. The value of nonstatic member variable actually defines the state of objects.

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2016-08-11T08:12:00-07:00&max-results=6#ixzz4i1Vy0lrV>

# [Difference between Static vs non Static method in Java](http://www.java67.com/2016/04/difference-between-static-vs-non-static-method-in-java.html)

One of the key difference between a static and a non-static method is that static method belongs to a class while non-static method belongs to the instance. This means you can call a static method without creating any instance of the class by just using the name of the class e.g. Math.random() for [creating random numbers](http://java67.blogspot.com/2015/01/how-to-get-random-number-between-0-and-1-java.html) in Java. Often utility methods which don't use the member variables of the class are declared static. On the other hand, you need an instance of the class to call a non-static method in Java. You cannot call it without creating an object because they are dependent upon the member variables which has different values for different instances.  One more important *difference between the static and non-static method* is that you cannot use a [non-static member variable](http://javarevisited.blogspot.com/2012/02/why-non-static-variable-cannot-be.html) inside a static method, you cannot even call a non-static method from the static method, but the opposite is true e.g. you can call a static function from a non-static method in Java.

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2016-08-11T08:12:00-07:00&max-results=6#ixzz4i1W52MFi>

# [Double Checked Locking in Java and Why it was broken before JDK 5?](http://www.java67.com/2016/04/why-double-checked-locking-was-broken-before-java5.html)

**Double checked locking pattern** is one of the interesting topics on Java Interviews. Earlier, it was asked to see if Java developer can write code using [synchronized block](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html) or not and now it ask to gauge the candidate's understanding of concurrency, volatile and synchronization in Java. One of the simplest ways to write thread-safe Singleton was to make the getInstance() method synchronized but prior to JDK 1.6, a simple uncontented synchronization block was expensive and that lead many developers to write the getInstance() method of Singleton class using [double-checked locking idiom](http://javarevisited.blogspot.com/2014/05/double-checked-locking-on-singleton-in-java.html). This was one of the clever idiom of that time which only uses synchronization when the Singleton object is created as seen in the following code and thus improves the performance of getInstance() method, which is used to retrieve the Singleton object.

[Read more »](http://www.java67.com/2016/04/why-double-checked-locking-was-broken-before-java5.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[4 comments:](http://www.java67.com/2016/04/why-double-checked-locking-was-broken-before-java5.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3962334384924231161)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3962334384924231161)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3962334384924231161&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3962334384924231161&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3962334384924231161&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3962334384924231161&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3962334384924231161&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [design pattern](http://www.java67.com/search/label/design%20pattern?&max-results=3)

# [Difference between FileReader vs FileInputStream in Java?](http://www.java67.com/2016/03/difference-between-filereader-vs.html)

Even though both FileReader and FileInputStream are used to read data from a file in Java, they are quite a different. The main difference between the FileReader and FileInputStream is that one read data from [character stream](http://java67.blogspot.com/2014/05/3-examples-to-read-inputstream-as-String-Java-Guava-Commons.html) while other read data from a [byte stream](http://java67.blogspot.com/2015/05/how-to-convert-byte-array-to-string-in-java-example.html). The **FileReader**automatically converts the raw bytes into character by using platform's default character encoding. This means you should use this class if you are reading from a text file which has same character encoding as the default one. If you happen to read a text file encoded in different character encoding then you should use **InputStreamReader**with specified character encoding. An InputStreamReader is a bridge between byte stream and character stream and can take a FileInputStream as a source. Though, it's worth remembering that it caches the [character encoding](http://javarevisited.blogspot.com/2015/02/difference-between-utf-8-utf-16-and-utf.html) which means you cannot change the encoding scheme programmatically.

[Read more »](http://www.java67.com/2016/03/difference-between-filereader-vs.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[2 comments:](http://www.java67.com/2016/03/difference-between-filereader-vs.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=1726140705727331230)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=1726140705727331230)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=1726140705727331230&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=1726140705727331230&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=1726140705727331230&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=1726140705727331230&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=1726140705727331230&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java IO interview question](http://www.java67.com/search/label/Java%20IO%20interview%20question?&max-results=3), [java io tutorial](http://www.java67.com/search/label/java%20io%20tutorial?&max-results=3)

# [15 Technical Core Java Interview Questions Answers for Experienced Developers](http://www.java67.com/2013/07/15-advanced-core-java-interview-questions-answers-senior-experienced-5-6-years-programmers-developers.html)

When the experience of a Java Programmer grows in the years e.g. when it goes from beginner years ( 2 to 4) to more experience or sort of senior level ( 5 to 7 years), [*Core Java Interview Questions*](http://java67.blogspot.com/2012/09/top-10-tough-core-java-interview-questions-answers.html) also changes a bit. Of course, basics like **data structure**, **algorithms,**and **object-oriented programming** remains same, but types of questions will become more advanced and their answers will definitely need to be more detailed and accurate. I often receive queries about core Java questions asked to a senior developer of 5 to 6-year experience, or, sometimes, I am going for an interview of senior Java developer, what kind of questions I should expect. This sometimes puzzles me, that once you become senior, you automatically starts taking part in the interview, and you should have an idea of what to expect on Interviews, but at the same time, I can understand that having an idea of questions before going on Interview, helps preparation. Of course, you are not going to get a question like the one you have faced on [2 to 3 years level Java Interviews](http://java67.blogspot.sg/2012/10/java-interview-questions-for-2-to-3-4-years-experienced.html), but It also depends on different rounds of Interviews.

[Read more »](http://www.java67.com/2013/07/15-advanced-core-java-interview-questions-answers-senior-experienced-5-6-years-programmers-developers.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[5 comments:](http://www.java67.com/2013/07/15-advanced-core-java-interview-questions-answers-senior-experienced-5-6-years-programmers-developers.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4897165616056825411)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4897165616056825411)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4897165616056825411&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4897165616056825411&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4897165616056825411&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4897165616056825411&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4897165616056825411&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java interview questions](http://www.java67.com/search/label/Java%20interview%20questions?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3)

# [5 Examples of substring() in Java](http://www.java67.com/2016/01/5-examples-of-substring-in-java.html)

SubString in Java is a useful method from java.lang.String class, which is used to create smaller String from bigger ones. The way Substring works prior to Java 1.7 can create a subtle[memory leak](http://java67.blogspot.com/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html) because both String and their substring shares same character array. Which means, if you have a big String of 200MB and created a substring of 2MB from that, that could prevent 200MB String from being garbage collected. I agree this doesn't look normal and indeed was a bug, but it was like that till Java 1.6 and it's various update. One reason, which I could think, why Java designer initially thought like that, maybe to save memory by sharing char array and to make, creating substring faster by just copying pointers, instead of data. Nevertheless, this was reported as bug and Oracle have fixed it, so no more [substring memory leak issue](http://javarevisited.blogspot.com/2011/10/how-substring-in-java-works.html) in Java 7.

[Read more »](http://www.java67.com/2016/01/5-examples-of-substring-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/01/5-examples-of-substring-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=197621884277523543)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=197621884277523543)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=197621884277523543&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=197621884277523543&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=197621884277523543&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=197621884277523543&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=197621884277523543&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3), [String](http://www.java67.com/search/label/String?&max-results=3) Location: [United States](https://maps.google.com/maps?q=United+States@37.09024,-95.71289100000001&z=10)

# [Difference between NoClassDefFoundError vs ClassNotFoundExcepiton in Java](http://www.java67.com/2012/12/noclassdeffounderror-vs-classnotfoundexception-java.html)

Both NoClassDefFoundError and ClassNotFoundException are dangerous errors which come when JVM or [ClassLoader](http://javarevisited.blogspot.com.au/2012/12/how-classloader-works-in-java.html) not able to locate class during the [class loading process](http://javarevisited.blogspot.com.au/2012/07/when-class-loading-initialization-java-example.html). Since different ClassLoader loads classes from a different location, sometimes this issue may be caused because of incorrect CLASSPATH as well i.e. some [JAR files](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html) from lib are missing or from the old version. Though look quite similar there is a subtle *difference between NoClassDefFoundError and ClassNotFoundException*, NoClassDefFoundError indicates that class was present during the time of compilation but not available when you run Java program, sometimes error on static initializer block can also result in NoClassDefFoundError.

ClassNotFoundException Vs NoClassDefFoundError in Java

[**2**](https://www.geeksforgeeks.org/easy/)

**ClassNotFoundException** and **NoClassDefFoundError** both occur when class is not found at runtime. They are related to Java classpath.

**ClassNotFoundException**

**ClassNotFoundException** occurs when you try to load a class at runtime using **Class.forName()** or **loadClass()** methods and requested classes are not found in classpath. Most of the time this exception will occur when you try to run application without updating classpath with JAR files. This exception is a **checked Exception** derived from **java.lang.Exception** class and you need to provide **explicit handling** for it. This exception also occurs when you have two class loaders and if a ClassLoader tries to access a class which is loaded by another classloader in Java. You must be wondering that what actually is classloader in Java. **Java ClassLoader** is a part of Java Runtime Environment that dynamically loads Java classes in JVM(Java Virtual Machine). The Java Runtime System does not need to know about files and files system because of classloaders.  
ClassNotFoundException is raised in below program as class “GeeksForGeeks” is not found in classpath.

|  |
| --- |
| // Java program to illustrate  // ClassNotFoundException  public class Example {        public static void main(String args[]) {          try          {              Class.forName("GeeksForGeeks");          }          catch (ClassNotFoundException ex)          {              ex.printStackTrace();          }      }  } |

Run on IDE

**Output**

java.lang.ClassNotFoundException: GeeksForGeeks

**NoClassDefFoundError**

NoClassDefFoundError occurs when class was present during compile time and program was compiled and linked successfully but class was **not** present during runtime. It is error which is derived from **LinkageError**. Linkage error occurs when a class has some dependencies on another class and latter class changes after compilation of former class. NoClassFoundError is the result of **implicit loading** of class because of calling a method or accessing a variable from that class. This error is more difficult to debug and find the reason why this error occurred. So in this case you should always check the classes which are dependent on this class.  
Note: This program will not run on IDE. Try to run it on your own systems.  
First make any two classes for a java program and link them.

|  |
| --- |
| // Java program to illustrate  // NoClassDefFoundError  class GeeksForGeeks  {      void greeting()      {          System.out.println("hello!");      }  }    class G4G {      public static void main(String args[])      {          GeeksForGeeks geeks = new geeksForGeeks();          geeks.greeting();      }  } |

Run on IDE

Above program will be successfully compiled and generate two classes GeeksForGeeks.class and G4G.class .  
Now remove GeeksForGeeks.class file and run G4G.class.  
At Java runtime **NoClassDefFoundError** will be thrown.

**ClassNotFoundException Vs NoClassDefFoundError**

* As the name suggests, ClassNotFoundException is an exception while NoClassDefFoundError is an error.
* ClassNotFoundException occurs when classpath is does not get updated with required JAR files while error occurs when required class definition is not present at runtime.

# [What is Polymorphism in Java? Overriding or Overloading?](http://www.java67.com/2012/10/difference-between-polymorphism-overloading-overriding-java.html)

**Polymorphism vs Overloading vs Overriding**

Someone asked me What are the difference between Polymorphism and Overriding in Java and the similar *difference between Polymorphism and Overloading*. Well, they are not two different things, **Polymorphism** is an object oriented or OOPS concept like [Abstraction](http://javarevisited.blogspot.in/2010/10/abstraction-in-java.html), Encapsulation or Inheritance which facilitate the use of the interface and allows Java program to take advantage of dynamic binding in Java. Polymorphism is also a way through which a Type can behave differently than expected based upon which kind of Object it is pointing. Overloading and overriding are two forms of Polymorphism available in Java.

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2016-04-10T03:21:00-07:00&max-results=6#ixzz4i1WFbshQ>

# [Top 5 Java Main method Interview Questions with Answers](http://www.java67.com/2016/01/main-method-interview-questions-in-java-answers.html)

The main() method in Java is starting point of any standalone core Java application. JVM starts executing Java program from main method and the thread which executes main is called main thread in Java. [The main method](http://java67.blogspot.sg/2012/08/what-is-main-method-in-java-why-main-is.html) is also an important topic in Java interviews for 2 to 3 years experienced developer. In this Java article, we will a couple of questions related to the main method in Java. Apart from [Why main is static in Java](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html), I see following questions keep coming related to the main method:

1. Can we overload the main method in Java? Which main method JVM will call?
2. Can we override the main method in Java?
3. Can we make main final in Java?
4. Can we make main synchronized in Java?
5. How to call a nonstatic method from main in Java?

## 5 Main Method Interview Questions with Answers

**Can we overload main in Java?**

Yes you can [overload](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) the main method in Java, nothing wrong with this but Java will only call your specific main method, i.e. main method with the following signature:

public static void main(String[] args) or  public static void main(String args...) which is the main method as [variable argument method](http://javarevisited.blogspot.sg/2011/09/variable-argument-in-java5-varargs.html) and only supported post-Java 5 world.

**Can we override main in Java?**

No, you can not override the main method in Java, Why? because main is a [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) and in Java static method is bonded during compile time and you can not [override](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) static method in Java. If you declare a method with same name and signature its called method hiding.

**Can we make main final in Java?**

Of course, you can make the main method final in Java. JVM has no issue with that. Unlike any [final method](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), you can not override main in Java.

**Can we make main synchronized in Java?**

Yes, main can be synchronized in Java,  [synchronized modifier](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) is allowed in the main signature and you can make your main method synchronized in Java.

**How to call a nonstatic method from main in Java?**

This question applies not only to main but all static methods in Java. Since [nonstatic methods can not be called from static context](http://javarevisited.blogspot.sg/2012/02/why-non-static-variable-cannot-be.html) directly, you need to first create an Object as local variable and then you can call nonstatic method using that object, as shown in the following example:

**import** java.util.Date;  
  
  
/\*\*  
 \* Java program to show how to call non static method from static method in Java  
 \*  
 \* @author http://java67.blogspot.com  
 \*/  
**public** **class** StaticTest {  
  
    **public** **static** **void** main(**String** args[]) {  
        
        *// calling non static method from main in Java*  
        *//printCurrentTime(); //compile time error - can not call non static method from main*  
        
        StaticTest test = **new** StaticTest();  
        test.printCurrentTime();  
        
    }   
    
    
    **public** **void** printCurrentTime(){  
       **System**.out.println(**new** **Date**());  
    }  
}  
  
**Output:**  
Tue Nov 06 19:07:54 IST 2012

### Summary

1. The main() method is a static method
2. You can overload main() method in Java.
3. You cannot override main() method in Java
4. You can make the main method final in Java
5. You can make the main method synchronized in Java.
6. You cannot call a non-static method from main in Java.

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2016-01-17T06:52:00-08:00&max-results=6#ixzz4i1WVgjDn>

[Java Program to reverse an array in place? Fastest Example](http://www.java67.com/2016/01/java-program-to-reverse-array-in-place.html)

It's easy to [reverse an array](http://javarevisited.blogspot.com/2013/03/how-to-reverse-array-in-java-int-String-array-example.html) if you have the luxury to use another array, but how would you reverse an array if a temporary buffer is not allowed? This is one of the testing array interview questions, which often proved tricky for Java programmers. Well, you can also *reverse an array in place*without using an additional buffer. If you know how to access array elements and [how to loop over an array in Java](http://java67.blogspot.com/2013/08/how-to-iterate-over-array-in-java-15.html) using traditional for loop, you can easily solve this problem without using additional space. All you need to do is loop over the array from start to the middle element and swap first element to the last, second element to the second last etc. Once you reach the middle element, your array is already sorted and that too without using any additional space. You can even use this algorithm to [reverse a String in Java](http://java67.blogspot.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.html) as well. After all, a String is backed by character array.

[Read more »](http://www.java67.com/2016/01/java-program-to-reverse-array-in-place.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2016/01/java-program-to-reverse-array-in-place.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=9091617095156465926)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=9091617095156465926)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=9091617095156465926&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=9091617095156465926&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=9091617095156465926&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=9091617095156465926&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=9091617095156465926&target=pinterest)

Labels: [array](http://www.java67.com/search/label/array?&max-results=3), [Coding Problems](http://www.java67.com/search/label/Coding%20Problems?&max-results=3), [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [data structure and algorithm](http://www.java67.com/search/label/data%20structure%20and%20algorithm?&max-results=3)

[Java Interview Questions Answers for Freshers (1 to 4 years experienced)](http://www.java67.com/2015/12/java-interview-questions-answers-for-freshers-1-to-4-years-experience.html)

I have shared many posts about [Java Interview Questions](http://java67.blogspot.com/2015/03/top-40-core-java-interview-questions-answers-telephonic-round.html) but in this article, I will share Java Interview questions and answers especially for *junior Java developers*. This includes college graduates, who are looking for the job, Java programmers who have 1 to 2 years of experience in Java and junior developers who have 2 to 4 years of experience in Java, C++ etc. At this level, Java questions are usually not very tough. If you have good knowledge of key programming areas e.g. data structure and algorithms, and a good understanding of Java programming language and JDK API, then you are in good shape. Interviewer often looks for candidates who are willing to learn, good at learning in quick time and knows basic stuff e.g. sorting list, searching through the array, starting stopping a thread and can read existing code. In this article, you will find questions from the areas targeted by Interview for freshers and Java developers with 1 to 4 years of experience e.g. [coding](http://java67.blogspot.com/2012/08/10-java-coding-interview-questions-and.html), [array](http://java67.blogspot.com/2015/07/array-concepts-interview-questions-answers-java.html), [OOP](http://java67.blogspot.com/2015/08/top-10-method-overloading-overriding-interview-questions-answers-java.html), [main method](http://java67.blogspot.com/2012/12/main-method-interview-questions-in-java-answers.html), [Collections](http://java67.blogspot.com/2012/09/java-collection-interview-questions.html), [ArrayList](http://java67.blogspot.com/2015/06/20-java-arraylist-interview-questions.html), [Enums](http://java67.blogspot.com/2013/07/15-java-enum-interview-questions-amswers-for-experienced-programmers.html), [Threads](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html), [design patterns](http://java67.blogspot.com/2012/09/top-10-java-design-pattern-interview-question-answer.html), [JDBC](http://java67.blogspot.com/2012/12/jdbc-interview-questions-answers-in-Java-2-4-years-experienced.html), [SQL](http://java67.blogspot.com/2013/04/10-frequently-asked-sql-query-interview-questions-answers-database.html), [Linux](http://java67.blogspot.com/2012/09/10-linux-and-unix-interview-questions-answers-wipro-tcs-capegemini.html) and other basic concepts.

[Read more »](http://www.java67.com/2015/12/java-interview-questions-answers-for-freshers-1-to-4-years-experience.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[2 comments:](http://www.java67.com/2015/12/java-interview-questions-answers-for-freshers-1-to-4-years-experience.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=2791453413136140493)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=2791453413136140493)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2791453413136140493&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2791453413136140493&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2791453413136140493&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2791453413136140493&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2791453413136140493&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3)

[2 Ways to find duplicate elements in an Array - Java](http://www.java67.com/2015/10/2-ways-to-find-duplicate-elements-in-java-array.html)

Problem: You have given an array of objects, which could be an array of integers and or array of Strings or any object which implements the Comparable interface. How would you find duplicate elements from an array? Can you solve this problem in O(n) complexity? This is actually one of the frequently asked [coding problems from Java interviews](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html). There are multiple ways to solve this problem and you will learn two popular ways here, first the brute force way, which involves comparing each element with every other element and other which uses a hash table like data structure to reduce the time complexity of problem from quadratic to linear, of course by trading off some space complexity. This also shows that how by using a suitable data structure you can come up with a better algorithm to solve a problem. If you are preparing for programming job interviews, then I also suggest you take a look at [Cracking the Coding Interview](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20) book, which contains 150 programming questions and solutions, good enough to do well on any programming job interviews e.g. Java, C++, Python or Ruby.

# Synchronization of ArrayList in Java

[**4.3**](https://www.geeksforgeeks.org/hard/)

Implementation of arrayList is not synchronized is by default. It means if a thread modifies it structurally and multiple threads access it concurrently, it must be synchronized externally. Structural modification means addition or deletion of element(s) from the list or explicitly resizes the backing array. Changing the value of existing element is not structural modification.

There are two way to create **Synchronized** Arraylist.  
1. Collections.synchronizedList() method.  
2. Using CopyOnWriteArrayList.

**Method 1: Using Collections.synchronizedList() method**

To do serial access, it is critical that all access to the backing list is accomplished through returned list. It is imperative that user **manually** synchronize on the returned list when iterating over it.

**public static List<T> synchronizedList(List<T> list)**

Accepts a List which could be implementation of **List**

interface. e.g. ArrayList, LinkedList.

Returns a Synchronized(thread-safe) list backed by the

specified list.

Parameter list is the list to be wrapped in a synchronize list.

T represents [generic](https://www.geeksforgeeks.org/generics-in-java/)

|  |
| --- |
| // Java program to demonstrate working of  // Collections.synchronizedList  import java.util.\*;    class GFG  {      public static void main (String[] args)      {          List<String> list =             Collections.synchronizedList(new ArrayList<String>());            list.add("practice");          list.add("code");          list.add("quiz");            synchronized(list)          {              // must be in synchronized block              Iterator it = list.iterator();                while (it.hasNext())                  System.out.println(it.next());          }      }  } |

Run on IDE

Output:

practice

code

quiz

**Method 2: Using CopyOnWriteArrayList**

CopyOnWriteArrayList<T> threadSafeList = new CopyOnWriteArrayList<T>();

Create an empty List.

It implements **List** interface.

It is a thread-safe variant of ArrayList.

T represents [generic](https://www.geeksforgeeks.org/generics-in-java/)

A thread-safe variant of ArrayList in which all mutative operations (e.g. add, set, remove..) are implemented by creating a separate copy of underlying **array**. It achieves **thread-safety** by creating a separate copy of List which is a is different way than vector or other collections use to provide **thread-safety**.

* It is useful when you can’t or don’t want to **synchronize** the traversal, yet need to prevent interference among concurrent threads.
* It is **costly** as involves separate **Array copy** with every write operation(e.g. add, set, remove..)
* It is very efficient when you have **List** and need to traverse over its elements and don’t modify often it.

Iterator does not throw [**ConcurrentModificationException**](https://docs.oracle.com/javase/7/docs/api/java/util/ConcurrentModificationException.html) even if copyOnWriteArrayList is modified once iterator is created because iterator is iterating over the separate copy of ArrayList while write operation is happening on another copy of ArrayList.

|  |
| --- |
| // Java program to illustrate the thread-safe ArrayList.  import java.io.\*;  import java.util.Iterator;  import java.util.concurrent.CopyOnWriteArrayList;    class GFG  {      public static void main (String[] args)      {          // creating a thread-safe Arraylist.          CopyOnWriteArrayList<String> threadSafeList              = new CopyOnWriteArrayList<String>();            // Adding elements to synchronized ArrayList          threadSafeList.add("geek");          threadSafeList.add("code");          threadSafeList.add("practice");            System.out.println("Elements of synchronized ArrayList :");            // Iterating on the synchronized ArrayList using iterator.          Iterator<String> it = threadSafeList.iterator();            while (it.hasNext())              System.out.println(it.next());      }  } |

Run on IDE

Output:

Elements of synchronized ArrayList :

geek

code

practice

**What happens if we try to modify CopyOnWriteArrayList through iterator’s own method?**  
It throws UnsupportedOperationException if you try to modify **CopyOnWriteArrayList** through iterator’s own method(e.g. add(), set(), remove()).

|  |
| --- |
| // Java program to illustrate the thread-safe ArrayList  import java.io.\*;  import java.util.Iterator;  import java.util.concurrent.CopyOnWriteArrayList;    class GFG  {      public static void main (String[] args)      {          // creating a thread-safe Arraylist.          CopyOnWriteArrayList<String> threadSafeList =              new CopyOnWriteArrayList<String>();            // Adding elements to synchronized ArrayList          threadSafeList.add("geek");          threadSafeList.add("code");          threadSafeList.add("practice");            System.out.println("Elements of synchronized ArrayList :");            // Iterating on the synchronized ArrayList using iterator.          Iterator<String> it = threadSafeList.iterator();            while (it.hasNext())          {              String str = it.next();              it.remove();          }      }  } |

Run on IDE

Runtime Error:

Exception in thread "main" java.lang.UnsupportedOperationException

at java.util.concurrent.CopyOnWriteArrayList$COWIterator.remove

(CopyOnWriteArrayList.java:1176)

at GFG.main(File.java:28)

Other constructors of CopyOnWriteArrayList :  
1. **CopyOnWriteArrayList(Collection<? extends E> c)** : Creates a list containing the elements of the specified collection, in the order they are returned by the collection’s iterator.  
2. **CopyOnWriteArrayList(E[] toCopyIn)** : Creates a list holding a copy of the given array.

**Why to use arrayList when vector is synchronized?**

1. **Performance:** Vector is **synchronized** and thread-safe and because of this, it is slightly slower than ArrayList.
2. **Functionality:** Vector synchronizes at the level of each individual operation. Generally a programmer like to synchronize a whole sequence of operations. Synchronizing individual operations is both less safe and slower.
3. **Vectors obsolete:**Vectors are considered obsolete an d unofficially deprecated in java. Also,vector synchronizes on each individual operation which is almost never done. Mostly java programmers prefer using ArrayList since they will probably synchronize the arrayList explicitly anyway if they need to do synchronization.

[Top 10 Tricky Java interview questions and Answers](http://www.java67.com/2012/09/top-10-tricky-java-interview-questions-answers.html)

What is a tricky question? Well, tricky Java interview questions are those questions which have some surprise element on it. If you try to answer a tricky question with common sense, you will most likely fail because they require some specific knowledge. Most of the tricky Java questions comes from confusing concepts like function overloading and overriding, Multi-threading which is really tricky to master, character encoding, [checked vs unchecked exceptions](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html) and subtle Java programming details like Integer overflow. Most important thing to answer a tricky Java question is attitude and analytical thinking, which helps even if you don't know the answer. Anyway in this Java article we will see 10 Java questions which are real tricky and requires more than average knowledge of Java programming language to answer them correctly. As per my experience, there is always one or two tricky or [tough Java interview question](http://java67.blogspot.sg/2012/09/top-10-tough-core-java-interview-questions-answers.html) on any core Java or J2EE interviews, so it's good to prepare tricky questions from Java in advance.  
  
If I take an interview, I purposefully put this kind of question to gauge the depth of candidate's understanding in Java. Another advantage of asking such question is the surprising element, which is a key factor to put the candidate on some pressure during interviews.

Since these questions are less common, there is good chance that many Java developer doesn't know about it.  You won't find these questions even on popular Java interview books like [Java Programming Interview exposed](http://javarevisited.blogspot.com/2015/12/5-good-books-for-java-jee-programming.html), which is nevertheless an excellent guide for Java interviews.

Btw, if you don't find these question tricky enough, then you should check Joshua Bloch's another classic book, [Java Puzzlers](http://www.java67.com/2016/06/12-must-read-advance-java-books-for-intermediate-programmers.html) for super tricky questions. I am sure you will find them challenging enough.

**10 Tricky Java interview question - Answered**

Here is my list of 10 tricky Java interview questions, Though I have prepared and shared lot of difficult core Java interview question and answers, But I have chosen them as Top 10 tricky questions because you can not guess answers of this tricky Java questions easily, you need some subtle details of Java programming language to answer these questions.

**Question: What does the following Java program print?**

public class Test {

public static void main(String[] args) {

System.out.println(Math.min(Double.MIN\_VALUE, 0.0d));

}

}

Answer: This question is tricky because unlike the [Integer](http://java67.blogspot.sg/2013/03/how-to-convert-java-string-to-int-or.html), where MIN\_VALUE is negative, both the MAX\_VALUE and MIN\_VALUE of the Double class are positive numbers. The Double.MIN\_VALUE is 2^(-1074), a double constant whose magnitude is the least among all double values. So unlike the obvious answer, this program will print 0.0 because Double.MIN\_VALUE is greater than 0. I have asked this question to Java developer having experience up to 3 to 5 years and surprisingly almost 70% candidate got it wrong.

**What will happen if you put return statement or System.exit () on try or catch block? Will finally block execute?**  
This is a very popular tricky Java question and it's tricky because many programmers think that no matter what, but the [finally block](http://java67.blogspot.com/2016/06/difference-between-final-vs-finally-vs-finalize-in-java.html) will always execute. This question challenge that concept by putting a return statement in the try or catch block or calling System.exit() from try or catch block. Answer of this tricky question in Java is that finally block will execute even if you put a return statement in the try block or catch block but finally block won't run if you call System.exit() from try or catch block.

**Question: Can you override a private or static method in Java?**  
Another popular Java tricky question, As I said method overriding is a good topic to ask trick questions in Java. Anyway, [you can not override a private or static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html), if you create a similar method with same return type and same method arguments in child class then it will hide the superclass method, this is known as method hiding.

Similarly, you cannot override a private method in sub class because it's not accessible there, what you do is create another private method with the same name in the child class. See [Can you override a private method in Java](http://java67.blogspot.sg/2012/08/can-we-override-private-method-in-java.html) or more details.

**Question: What do the expression 1.0 / 0.0 will return? will it throw Exception? any compile time error?**  
Answer: This is another tricky question from Double class. Though Java developer knows about the double primitive type and Double class, while doing floating point arithmetic they don't pay enough attention to Double.INFINITY, NaN, and -0.0 and other rules that govern the arithmetic calculations involving them. The simple answer to this question is that it will not throw ArithmeticExcpetion and return Double.INFINITY.  
  
Also, note that the comparison x == Double.NaN always evaluates to false, even if x itself is a NaN. To test if x is a NaN, one should use the method call Double.isNaN(x) to check if given number is NaN or not. If you know SQL, this is very close to NULL there.   
  
Btw, If you are running out of time for your interview preparation, you can also check out [Java Programming Interviews exposed](http://www.amazon.com/Java-Programming-Interviews-Exposed-Markham/dp/1118722868?tag=javamysqlanta-20) for more of such popular questions,

**Does Java support multiple inheritances?**  
This is the trickiest question in Java if C++ can support direct multiple inheritances than why not Java is the argument Interviewer often give. Answer of this question is much more subtle then it looks like, because Java does support multiple inheritances of Type by allowing an interface to extend other interfaces, what Java doesn't support is multiple inheritances of implementation. This distinction also gets blur because of default method of Java 8, which now provides Java, multiple inheritances of behavior as well. See [why multiple inheritances are not supported in Java](http://javarevisited.blogspot.sg/2011/07/why-multiple-inheritances-are-not.html) to answer this tricky Java question.

**What will happen if we put a key object in a HashMap which is already there?**  
This tricky Java question is part of another frequently asked question, How HashMap works in Java. HashMap is also a popular topic to create confusing and tricky question in Java. Answer of this question is if you put the same key again then it will replace the old mapping because HashMap doesn't allow duplicate keys. The Same key will result in the same hashcode and will end up at the same position in the bucket.

 Each bucket contains a linked list of Map.Entry object, which contains both Key and Value. Now Java will take the Key object from each entry and compare with this new key using equals() method, if that return true then value object in that entry will be replaced by new value. See [How HashMap works in Java](http://java67.blogspot.sg/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) for more tricky Java questions from HashMap.

**Question: What does the following Java program print?**

public class Test {

public static void main(String[] args) throws Exception {

char[] chars = new char[] {'\u0097'};

String str = new String(chars);

byte[] bytes = str.getBytes();

System.out.println(Arrays.toString(bytes));

}

}

Answer: The trickiness of this question lies on character encoding and how String to byte array conversion works. In this program, we are first creating a String from a character array, which just has one character '\u0097', after that we are getting the byte array from that String and printing that byte. Since \u0097 is within the 8-bit range of byte primitive type, it is reasonable to guess that the str.getBytes() call will return a byte array that contains one element with a value of -105 ((byte) 0x97).  
  
However, that's not what the program prints and that's why this question is tricky. As a matter of fact, the output of the program is operating system and locale dependent. On a Windows XP with the US locale, the above program prints [63], if you run this program on Linux or Solaris, you will get different values.  
  
To answer this question correctly, you need to know about how Unicode characters are represented in Java char values and in Java strings, and what role character encoding plays in String.getBytes().  
  
In simple word, t[o convert a string to a byte array](http://javarevisited.blogspot.sg/2014/08/2-examples-to-convert-byte-array-to-String-in-Java.html), Java iterate through all the characters that the string represents and turn each one into a number of bytes and finally put the bytes together. The rule that maps each Unicode character into a byte array is called a character encoding. So It's possible that if same character encoding is not used during both encoding and decoding then retrieved value may not be correct. When we call str.getBytes() without specifying a character encoding scheme, the JVM uses the default character encoding of the platform to do the job.  
  
The default encoding scheme is operating system and locale dependent. On Linux, it is UTF-8 and on Windows with a US locale, the default encoding is Cp1252. This explains the output we get from running this program on Windows machines with a US locale. No matter which character encoding scheme is used, Java will always translate Unicode characters not recognized by the encoding to 63, which represents the character U+003F (the question mark, ?) in all encodings.

**If a method throws NullPointerException in the superclass, can we override it with a method which throws RuntimeException?**  
One more tricky Java questions from the overloading and overriding concept. The answer is you can very well throw superclass of RuntimeException in overridden method, but you can not do same if its checked Exception. See [Rules of method overriding in Java](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) for more details.

**What is the issue with following implementation of compareTo() method in Java**

public int compareTo(Object o){

Employee emp = (Employee) o;

return this.id - e.id;

}

**where an id is an integer number.**

Well, three is nothing wrong in this Java question until you guarantee that id is always positive. This Java question becomes tricky when you can't guarantee that id is positive or negative. the tricky part is, If id becomes negative than **subtraction may overflow** and produce an incorrect result. See [How to override compareTo method in Java](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) for the complete answer of this Java tricky question for an experienced programmer.

**How do you ensure that N thread can access N resources without deadlock?**  
If you are not well versed in writing multi-threading code then this is a real tricky question for you. This Java question can be tricky even for the experienced and senior programmer, who are not really exposed to deadlock and race conditions. The key point here is ordering, if you acquire resources in a particular order and release resources in the reverse order you can prevent deadlock. See [how to avoid deadlock in Java](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html) for a sample code example.

**Question: Consider the following Java code snippet, which is initializing two variables and both are not volatile, and two threads T1 and T2 are modifying these values as following, both are not synchronized**

int x = 0;

boolean bExit = false;

Thread 1 (not synchronized)

x = 1;

bExit = true;

Thread 2 (not synchronized)

if (bExit == true)

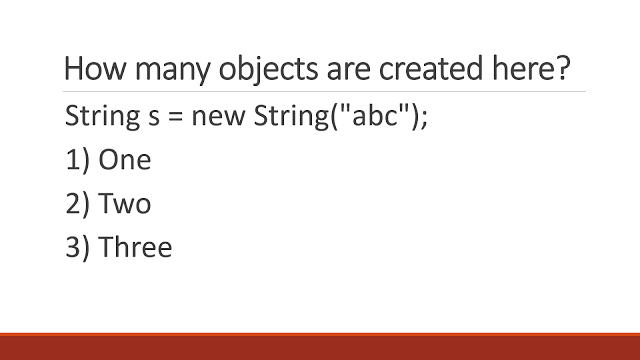
System.out.println("x=" + x);

**Now tell us, is it possible for Thread 2 to print “x=0”?**  
  
Answer: It's impossible for a list of tricky Java questions to not contain anything from multi-threading. This is the simplest one I can get. Answer of this question is Yes, It's possible that thread T2 may print x=0.Why? because without any instruction to compiler e.g. synchronized or volatile, bExit=true might come before x=1 in compiler reordering. Also, x=1 might not become visible in Thread 2, so Thread 2 will load x=0. Now, how do you fix it?  
  
 When I asked this question to a couple of programmers they answer differently, one suggests to make both threads synchronized on a common mutex, another one said make both variable volatile. Both are correct, as it will prevent reordering and guarantee visibility.  
  
But the best answer is you just need to make bExit as volatile, then Thread 2 can only print “x=1”. x does not need to be volatile because x cannot be reordered to come after bExit=true when bExit is volatile.

**What is difference between CyclicBarrier and CountDownLatch in Java**  
Relatively newer Java tricky question, only been introduced from Java 5. The main difference between both of them is that you can reuse CyclicBarrier even if Barrier is broken, but you can not reuse CountDownLatch in Java. See [CyclicBarrier vs CountDownLatch in Java](http://java67.blogspot.sg/2012/08/difference-between-countdownlatch-and-cyclicbarrier-java.html) for more differences.

**What is the difference between StringBuffer and StringBuilder in Java?**  
Classic Java questions which some people think tricky and some consider very easy. StringBuilder in Java was introduced in JDK 1.5 and the only difference between both of them is that StringBuffer methods e.g. length(), capacity() or append() are [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) while corresponding methods in StringBuilder are not synchronized.  
  
Because of this fundamental difference, concatenation of String using StringBuilder is faster than StringBuffer. Actually, it's considered the bad practice to use StringBuffer anymore, because, in almost 99% scenario, you perform string concatenation on the same thread. See [StringBuilder vs StringBuffer](http://javarevisited.blogspot.sg/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) for more differences.

**Can you access a non-static variable in the static context?**  
Another tricky Java question from Java fundamentals. No, you can not access a non-static variable from the static context in Java. If you try, it will give compile time error. This is actually a common problem beginner in Java face when they try to access instance variable inside the main method. Because main is static in Java, and instance variables are non-static, you can not access instance variable inside main. See, [why you can not access a non-static variable from static method](http://javarevisited.blogspot.sg/2012/02/why-non-static-variable-cannot-be.html) to learn more about this tricky Java questions.  
  
  
**How many String objects are created by the following code?**

[](https://4.bp.blogspot.com/-kitlSuknjow/V3xrNGYnOLI/AAAAAAAAGi0/8mkbr22IlngwqfdKzZgB_SRWkqNe0Ar_wCLcB/s1600/How+many+object+is+created+here+String+tricky+question+java.png)

Now, it's practice time, here are some questions for you guys to answer, these are contributed by readers of this blog, big thanks to them.

1. When doesn't Singleton remain Singleton in Java?
2. is it possible to load a class by two ClassLoader?
3. is it possible for equals() to return false, even if contents of two Objects are same?
4. Why compareTo() should be consistent to equals() method in Java?
5. When do Double and BigDecimal give different answers for equals() and compareTo() == 0.
6. How does "has before" apply to volatile work?
7. Why is 0.1 \* 3 != 0.3,
8. Why is (Integer) 1 == (Integer) 1 but (Integer) 222 != (Integer) 222 and which command arguments change this.
9. What happens when an exception is thrown by a Thread?
10. Difference between notify() and notifyAll() call?
11. Difference between System.exit() and System.halt() method?
12. Does following code legal in Java? is it an example of method overloading or overriding?

public String getDescription(Object obj){

return obj.toString;

}

public String getDescription(String obj){

return obj;

}

and

public void getDescription(String obj){

return obj;

}

This was my list of Some of the most common tricky questions in Java. It's not a bad idea to prepare tricky Java question before appearing for any core Java or J2EE interview. One or two open-ended or tricky question is quite common in Java interviews.

# Java Singleton Design Pattern Practices with Examples

In [previous](https://www.geeksforgeeks.org/singleton-design-pattern-introduction/) articles, we discussed about singleton design pattern and singleton class [implementation](https://www.geeksforgeeks.org/singleton-design-pattern/)in detail.  
In this article, we will see how we can create singleton classes. After reading this article you will be able to create your singleton class according to your use, simplicity and removed bottlenecks.  
There are many ways this can be done in Java. All these ways differs in their implementation of the pattern, but in the end, they all achieve the same end result of a single instance.

1. **Eager initialization:** This is the simplest method of creating a singleton class. In this, object of class is created when it is loaded to the memory by JVM. It is done by assigning the reference an instance directly.  
   It can be used when program will always use instance of this class, or the cost of creating the instance is not too large in terms of resources and time.

|  |
| --- |
| // Java code to create singleton class by  // Eager Initialization  public class GFG  {    // public instance initialized when loading the class    public static GFG instance = new GFG();      private GFG()    {      // private constructor    }  } |

1. Run on IDE
2. **Pros:**
   1. Very simple to implement.
   2. No need to implement getInstance() method. Instance can be accessed directly.

**Cons:**

* 1. May lead to resource wastage. Because instance of class is created always, whether it is required or not.
  2. CPU time is also wasted in creation of instance if it is not required.
  3. Exception handling is not possible.

1. **Using static block:** This is also a sub part of Eager initialization. The only difference is object is created in a static block so that we can have access on its creation, like exception handling. In this way also, object is created at the time of class loading.  
   It can be used when there is a chance of exceptions in creating object with eager initialization.

|  |
| --- |
| // Java code to create singleton class  // Using Static block  public class GFG  {    // public instance    public static GFG instance;      private GFG()    {      // private constructor    }      {      // static block to initialize instance      instance = new GFG();    }  } |

Run on IDE

**Pros:**

* 1. Very simple to implement.
  2. No need to implement getInstance() method. Instance can be accessed directly.
  3. Exceptions can be handled in static block.

**Cons:**

* 1. May lead to resource wastage. Because instance of class is created always, whether it is required or not.
  2. CPU time is also wasted in creation of instance if it is not required.

1. **Lazy initialization:** In this method, object is created only if it is needed. This may prevent resource wastage. An implementation of getInstance() method is required which return the instance. There is a null check that if object is not created then create, otherwise return previously created. To make sure that class cannot be instantiated in any other way, constructor is made final. As object is created with in a method, it ensures that object will not be created until and unless it is required. Instance is kept private so that no one can access it directly.  
   It can be used in a single threaded environment because multiple threads can break singleton property because they can access get instance method simultaneously and create multiple objects.

|  |
| --- |
| //Java Code to create singleton class  // With Lazy initialization  public class GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method    private static GFG instance;      private GFG()    {      // private constructor    }      //method to return instance of class    public static GFG getInstance()    {      if (instance == null)      {        // if instance is null, initialize        instance = new GFG();      }      return instance;    }  } |

1. Run on IDE
2. **Pros:**
   1. Object is created only if it is needed. It may overcome resource overcome and wastage of CPU time.
   2. Exception handling is also possible in method.

**Cons:**

* 1. Every time a condition of null has to be checked.
  2. instance can’t be accessed directly.
  3. In multithreaded environment, it may break singleton property.

1. **Thread Safe Singleton:** A thread safe singleton in created so that singleton property is maintained even in multithreaded environment. To make a singleton class thread-safe, getInstance() method is made synchronized so that multiple threads can’t access it simultaneously.

|  |
| --- |
| // Java program to create Thread Safe  // Singleton class  public class GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method    private static GFG instance;      private GFG()    {      // private constructor    }     //synchronized method to control simultaneous access    synchronized public static GFG getInstance()    {      if (instance == null)      {        // if instance is null, initialize        instance = new GFG();      }      return instance;    }  } |

1. Run on IDE
2. **Pros:**
   1. Lazy initialization is possible.
   2. It is also thread safe.

**Cons:**

* 1. getInstance() method is synchronized so it causes slow performance as multiple threads can’t access it simultaneously.

1. **Lazy initialization with Double check locking:** In this mechanism, we overcome the overhead problem of synchronized code. In this method, getInstance is not synchronized but the block which creates instance is synchronized so that minimum number of threads have to wait and that’s only for first time.

|  |
| --- |
| // Java code to explain double check locking  public class GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method    private static GFG instance;      private GFG()    {      // private constructor    }      public static GFG getInstance()    {      if (instance == null)      {        //synchronized block to remove overhead        synchronized (GFG.class)        {          if(instance==null)          {            // if instance is null, initialize            instance = new GFG();          }          }      }      return instance;    }  } |

1. Run on IDE
2. **Pros:**
   1. Lazy initialization is possible.
   2. It is also thread safe.
   3. Performance reduced because of synchronized keyword is overcome.

**Cons:**

* 1. First time, it can affect performance.

As cons. of double check locking method is bearable so it can be used for high performance multi-threaded applications.

1. **Bill Pugh Singleton Implementation:** Prior to Java5, memory model had a lot of issues and above methods caused failure in certain scenarios in multithreaded environment. So, Bill Pugh suggested a concept of inner static classes to use for singleton.

|  |
| --- |
| // Java code for Bill Pugh Singleton Implementaion  public class GFG  {      private GFG()    {      // private constructor    }      // Inner class to provide instance of class    private static class BillPughSingleton    {      private static final GFG INSTANCE = new GFG();    }      public static GFG getInstance()    {      return BillPughSingleton.INSTANCE;    }  } |

1. Run on IDE
2. When the singleton class is loaded, inner class is not loaded and hence doesn’t create object when loading the class. Inner class is created only when getInstance() method is called. So it may seem like eager initialization but it is lazy initialization.  
   This is the most widely used approach as it doesn’t use synchronization.

**When to use What**

1. Eager initialization is easy to implement but it may cause resource and CPU time wastage. Use it only if cost of initializing a class is less in terms of resources or your program will always need the instance of class.
2. By using Static block in Eager initialization we can provide exception handling and also can control over instance.
3. Using synchronized we can create singleton class in multi-threading environment also but it can cause slow performance, so we can use Double check locking mechanism.
4. Bill Pugh implementation is most widely used approach for singleton classes. Most developers prefer it because of its simplicity and advantages.

This article is contributed by **Vishal Garg**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

# [Difference between Public, Private and Protected modifier in Java?](http://www.java67.com/2015/08/difference-between-public-private-and-protected-in-java.html)

In Java, you have got something called access modifier, which specifies accessibility of class, methods and variables. There is four access modifier in Java namely public, private, protected and the default access modifier, also known as [package level modifier](http://java67.blogspot.sg/2012/12/what-is-public-private-protected-package-default-private-access-modifier-java.html). The difference between these access modifier comes in their ability to restrict access to a class, method or variables, public is the least restrictive access modifier while private is the most restrictive access modifier, package and protected lies in between. Another key *difference between public, protected, package and private modifier* come from the point where you can apply them, for example, you cannot use [private](http://java67.blogspot.com/2012/12/what-is-public-private-protected-package-default-private-access-modifier-java.html) or [protected modifier](http://javarevisited.blogspot.com/2012/10/difference-between-private-protected-public-package-access-java.html) with a top level class but you can use public modifier there.

[Read more »](http://www.java67.com/2015/08/difference-between-public-private-and-protected-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2015/08/difference-between-public-private-and-protected-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5887320483475491389)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5887320483475491389)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5887320483475491389&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5887320483475491389&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5887320483475491389&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5887320483475491389&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5887320483475491389&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3)

# [HashSet vs TreeSet in Java? Similarities and Differences](http://www.java67.com/2015/07/difference-between-hashset-and-treeset-in-java.html)

HashSet and TreeSet both implement same interface i.e  java.util.Set interface and they possess the quality of Set interface means duplicate elements are not allowed. Both HashSet and TreeSet are used for to store unique elements, but HashSet doesn't care about any order and TreeSet keeps a thing in order. Ordering or sorting on TreeSet can be customized by using Comparator interface, by default TreeSet uses elements natural order for sorting, which is defined by compareTo() method of java.lang.Comparable interface. What is the difference between HashSet and TreeSet is is also one the [frequently asked Java interview question](http://java67.blogspot.sg/2014/07/21-frequently-asked-java-interview-questions-answers.html), So you should know about similarities and difference between them? It also helps you to understand when to use both TreeSet and HashSet and what are the scenario when we should use this sets. Let's go through the similarities and *difference between HashSet and TreeSet in Java*.

[Read more »](http://www.java67.com/2015/07/difference-between-hashset-and-treeset-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[1 comment:](http://www.java67.com/2015/07/difference-between-hashset-and-treeset-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=2368699350492134968)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=2368699350492134968)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2368699350492134968&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2368699350492134968&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2368699350492134968&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2368699350492134968&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2368699350492134968&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java Interview Question](http://www.java67.com/search/label/Java%20Interview%20Question?&max-results=3)

# [20 Java ArrayList Interview Questions Answers](http://www.java67.com/2015/06/20-java-arraylist-interview-questions.html)

In this article, I am going to share some of the good Java Interview questions based upon ArrayList class. I have hardly seen a Java interview without any question from ArrayList, and why not its one of the most popular collection class and every Java developer use it on their day to day work. Another reason for asking a question related to ArrayList is that you can ask a wide variety of question to really check breadth and depth of candidate's knowledge. To give you some idea about ArrayList, it's a collection class which implements List interface. It's an alternative to array data structure whose size you cannot change once created. ArrayList is a dynamic array, which can grow and resize itself. By implementing List interface it also got some properties e.g. ordering, ArrayList keeps the element in the order they are inserted and it also provides constant time search operation if you know the index of element e.g. **get(index)** is **O(1)** operation. This makes an ArrayList ideal choice when you are looking to retrieve values based upon the index.

[Read more »](http://www.java67.com/2015/06/20-java-arraylist-interview-questions.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[3 comments:](http://www.java67.com/2015/06/20-java-arraylist-interview-questions.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=7472831782791488499)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=7472831782791488499)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7472831782791488499&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7472831782791488499&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7472831782791488499&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7472831782791488499&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7472831782791488499&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java Interview Question](http://www.java67.com/search/label/Java%20Interview%20Question?&max-results=3)

# Start Fail safe/fast

# [What is fail safe and fail fast Iterator in Java?](http://www.java67.com/2015/06/what-is-fail-safe-and-fail-fast-iterator-in-java.html)

Java Collections supports two types of Iterator, fail safe and fail fast. The main distinction between a fail-fast and fail-safe Iterator is whether or not the underlying collection can be modified while its begin iterated. If you have used Collection like ArrayList then you know that when you iterate over them, no other thread should modify the collection. If Iterator detects any structural change after iteration has begun e.g adding or removing a new element then it throws ConcurrentModificationException,  this is known as fail-fast behavior and these iterators are called *fail-fast iterator* because they fail as soon as they detect any modification . Though it's not necessary that iterator will throw this exception when multiple threads modified it simultaneously. it can happen even with the single thread when you try to remove elements  by using ArrayList's remove() method instead of Iterator's remove method, as discussed in my earlier post, [2 ways to remove objects from ArrayList](http://java67.blogspot.sg/2014/03/2-ways-to-remove-elementsobjects-from-ArrayList-java.html).

# Fail Fast and Fail Safe Iterators in Java

[**3**](https://www.geeksforgeeks.org/medium/)

In this article, I am going to explain how those collections behave which doesn’t iterate as fail-fast. First of all, there is no term as fail-safe given in many places as Java SE specifications does not use this term. I am using fail safe to segregate between Fail fast and Non fail-fast iterators.

**Concurrent Modification:** Concurrent Modification in programming means to modify an object concurrently when another task is already running over it. For example, in Java to modify a collection when another thread is iterating over it. Some Iterator implementations (including those of all the general purpose collection implementations provided by the JRE) may choose to throw ConcurrentModificationException if this behavior is detected.

**Fail Fast And Fail Safe Iterators in Java**

[Iterators](https://contribute.geeksforgeeks.org/iterators-in-java/) in java are used to iterate over the Collection objects.Fail-Fast iterators immediately throw ConcurrentModificationException if there is **structural modification** of the collection. Structural modification means adding, removing or updating any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes are some examples of fail-fast Iterator.

Fail-Safe iterators don’t throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that’s why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes are examples of fail-safe Iterator.

**How Fail Fast Iterator works ?**

To know whether the collection is structurally modified or not, fail-fast iterators use an internal flag called modCount which is updated each time a collection is modified.Fail-fast iterators checks the modCount flag whenever it gets the next value (i.e. using next() method), and if it finds that the modCount has been modified after this iterator has been created, it throws ConcurrentModificationException.

|  |
| --- |
| // Java code to illustrate  // Fail Fast Iterator in Java  import java.util.HashMap;  import java.util.Iterator;  import java.util.Map;    public class FailFastExample {      public static void main(String[] args)      {          Map<String, String> cityCode = new HashMap<String, String>();          cityCode.put("Delhi", "India");          cityCode.put("Moscow", "Russia");          cityCode.put("New York", "USA");            Iterator iterator = cityCode.keySet().iterator();            while (iterator.hasNext()) {              System.out.println(cityCode.get(iterator.next()));                // adding an element to Map              // exception will be thrown on next call              // of next() method.              cityCode.put("Istanbul", "Turkey");          }      }  } |

Run on IDE

**Output :**

India

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.HashMap$HashIterator.nextNode(HashMap.java:1442)

at java.util.HashMap$KeyIterator.next(HashMap.java:1466)

at FailFastExample.main(FailFastExample.java:18)

**Important points of fail-fast iterators :**

* These iterators throw ConcurrentModificationException if a collection is modified while iterating over it.
* They use original collection to traverse over the elements of the collection.
* These iterators don’t require extra memory.
* Ex : Iterators returned by ArrayList, Vector, HashMap.

**Note 1(from java-docs):**The fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.

**Note 2 :**If you remove an element via Iterator remove() method, exception will not be thrown. However, in case of removing via a particular collection remove() method, ConcurrentModificationException will be thrown. Below code snippet will demonstrate this:

|  |
| --- |
| // Java code to demonstrate remove  // case in Fail-fast iterators    import java.util.ArrayList;  import java.util.Iterator;    public class FailFastExample {      public static void main(String[] args)      {          ArrayList<Integer> al = new ArrayList<>();          al.add(1);          al.add(2);          al.add(3);          al.add(4);          al.add(5);            Iterator<Integer> itr = al.iterator();          while (itr.hasNext()) {              if (itr.next() == 2) {                  // will not throw Exception                  itr.remove();              }          }            System.out.println(al);            itr = al.iterator();          while (itr.hasNext()) {              if (itr.next() == 3) {                  // will throw Exception on                  // next call of next() method                  al.remove(3);              }          }      }  } |

Run on IDE

**Output :**

[1, 3, 4, 5]

Exception in thread "main" java.util.ConcurrentModificationException

at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:901)

at java.util.ArrayList$Itr.next(ArrayList.java:851)

at FailFastExample.main(FailFastExample.java:28)

**Fail Safe Iterator**

First of all, there is no term as fail-safe given in many places as Java SE specifications does not use this term. I am using this term to demonstrate the difference between Fail Fast and Non-Fail Fast Iterator. These iterators make a copy of the internal collection (object array) and iterates over the copied collection. Any structural modification done to the iterator affects the copied collection, **not original collection**. So, original collection remains structurally **unchanged**.

* Fail-safe iterators allow modifications of a collection while iterating over it.
* These iterators don’t throw any Exception if a collection is modified while iterating over it.
* They use copy of original collection to traverse over the elements of the collection.
* These iterators require extra memory for cloning of collection. Ex : ConcurrentHashMap, CopyOnWriteArrayList

**Example of Fail Safe Iterator in Java:**

|  |
| --- |
| // Java code to illustrate  // Fail Safe Iterator in Java  import java.util.concurrent.CopyOnWriteArrayList;  import java.util.Iterator;    class FailSafe {      public static void main(String args[])      {          CopyOnWriteArrayList<Integer> list              = new CopyOnWriteArrayList<Integer>(new Integer[] { 1, 3, 5, 8 });          Iterator itr = list.iterator();          while (itr.hasNext()) {              Integer no = (Integer)itr.next();              System.out.println(no);              if (no == 8)                    // This will not print,                  // hence it has created separate copy                  list.add(14);          }      }  } |

Run on IDE

**Output:**

1

3

5

8

Also, those collections which don’t use fail-fast concept may not necessarily create clone/snapshot of it in memory to avoid ConcurrentModificationException. For example, in case of ConcurrentHashMap, it does not operate on a separate copy although it is not fail-fast. Instead, it has semantics that is described by the official specification as weakly consistent(memory consistency properties in Java). Below code snippet will demonstrate this:

**Example of Fail-Safe Iterator which does not create separate copy**

|  |
| --- |
| // Java program to illustrate  // Fail-Safe Iterator which  // does not create separate copy  import java.util.concurrent.ConcurrentHashMap;  import java.util.Iterator;    public class FailSafeItr {      public static void main(String[] args)      {            // Creating a ConcurrentHashMap          ConcurrentHashMap<String, Integer> map              = new ConcurrentHashMap<String, Integer>();            map.put("ONE", 1);          map.put("TWO", 2);          map.put("THREE", 3);          map.put("FOUR", 4);            // Getting an Iterator from map          Iterator it = map.keySet().iterator();            while (it.hasNext()) {              String key = (String)it.next();              System.out.println(key + " : " + map.get(key));                // This will reflect in iterator.              // Hence, it has not created separate copy              map.put("SEVEN", 7);          }      }  } |

Run on IDE

**Output**

ONE : 1

FOUR : 4

TWO : 2

THREE : 3

SEVEN : 7

**Note(from java-docs)** : The iterators returned by ConcurrentHashMap is weakly consistent. This means that this iterator can tolerate concurrent modification, traverses elements as they existed when iterator was constructed and may (but not guaranteed to) reflect modifications to the collection after the construction of the iterator.

**Difference between Fail Fast Iterator and Fail Safe Iterator**

The major difference is fail-safe iterator doesn’t throw any Exception, contrary to fail-fast Iterator.This is because they work on a clone of Collection instead of the original collection and that’s why they are called as the fail-safe iterator.

# End Fail safe/fast

# [Difference between synchronized ArrayList and CopyOnWriteArrayList in Java?](http://www.java67.com/2015/06/difference-between-synchronized-arraylist-and-copyOnWriteArrayList-java.html)

Though both synchronized ArrayList and CopyOnWriteArrayList provides you thread-safety and you can use both of them when your list is shared between multiple threads, there is a subtle difference between them, Synchronized ArrayList is a synchronized collection while CopyOnWriteArrayList is a concurrent collection. What does this mean? It means is that CopyOnWriteArrayList is designed keeping concurrency in mind and it is more scalable than [synchronized ArrayList](http://java67.blogspot.sg/2014/12/how-to-synchronize-arraylist-in-java.html) if the list is primarily used for reading. You know that ArrayList is not synchronized, so you cannot directly use it in a multi-threaded environment where you list is accessed and modified by multiple threads. In order to use ArrayList in such environment, you need to first get a synchronized list by calling Collections.synchronizedList().

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2015-10-20T08:51:00-07:00&max-results=6#ixzz4i1XBDNcI>

# [Difference between String literal and New String object in Java](http://www.java67.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html)

String is a special class in Java API and has so many special behaviours which is not obvious to many programmers. In order to master Java, first step is to master String class, and one way to explore is checking what kind of String related questions are asked on Java interviews. Apart from usual questions like [why String is final](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html), or  [equals vs == operator](http://java67.blogspot.sg/2012/11/difference-between-operator-and-equals-method-in.html), one of the most frequently asked question is *what is difference between String literal and String object in Java*. For example, what is the difference between String object created in following two expression :

String strObject = new String("Java");

and

String strLiteral = "Java";

Both expression gives you String object, but there is subtle difference between them. When you create String object using new() operator, it always create a new object in [heap memory](http://java67.blogspot.sg/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html). On the other hand, if you create object using String literal syntax e.g. "Java", it may return an existing object from String pool (a cache of String object in Perm gen space, which is now moved to heap space in recent Java release), if it's already exists. Otherwise it will create a new string object and put in string pool for future re-use. In rest of this article, why it is one of the most important thing you should remember about String in Java.

[Read more »](http://www.java67.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[32 comments:](http://www.java67.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5086152910667818979)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5086152910667818979)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5086152910667818979&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5086152910667818979&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5086152910667818979&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5086152910667818979&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5086152910667818979&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java Interview Question](http://www.java67.com/search/label/Java%20Interview%20Question?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3), [String](http://www.java67.com/search/label/String?&max-results=3)

# [Top 21 Frequently Asked Java Interview Questions Answers](http://www.java67.com/2014/07/21-frequently-asked-java-interview-questions-answers.html)

If you have been to couple of Java interviews then you know that there are some questions which keep repeating e.g. [difference between == and equals() method](http://java67.blogspot.sg/2012/11/difference-between-operator-and-equals-method-in.html) and may of it's popular cousins like HashMap vs Hashtable, ArrayList vs LinkedList, difference between equals() and hashCode(), or difference between Comparator and Comparable in Java. I call them frequently asked Java interview questions, and I suggest every Java developer to make a list of them for their own reference and revision. I am sure many Java programmer already has such list of questions handy, if not this is a good time to find and make your own list. These are the questions which you simply can't afford to miss, especially at freshers level. They appear at various stage of Java interviews. Most likely you will see them on telephonic round, where Interviewer just want to filter candidates between who knows Java and who doesn't.

[Read more »](http://www.java67.com/2014/07/21-frequently-asked-java-interview-questions-answers.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[13 comments:](http://www.java67.com/2014/07/21-frequently-asked-java-interview-questions-answers.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=2373873848802632442)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=2373873848802632442)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2373873848802632442&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2373873848802632442&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2373873848802632442&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2373873848802632442&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=2373873848802632442&target=pinterest)

Labels: [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3), [Sorting Algorithm](http://www.java67.com/search/label/Sorting%20Algorithm?&max-results=3)

# [Difference between StringBuilder and StringBuffer in Java](http://www.java67.com/2014/05/difference-between-stringbuilder-and-StringBuffer-java.html)

If you are in a hurry and heading straight to interview then I won't take much of your time, In a couple of words, the main difference between StringBuffer and StringBuilder is between four parameters, synchronization, speed, thread-safety, and availability. StringBuffer is synchronized and that's why thread-safe, but StringBuilder is not synchronized, not thread-safe and that's why fast. Regarding availability, StringBuffer is available from Java 1.0 while StringBuilder was added in Java 5. Now we can take a breath, and can continue with rest of this article. In Java, there are three classes to deal with String data type, String, StringBuffer and StringBuilder. All of three belongs to java.lang package, which is automatically imported into every Java program thus you don't need to do any import for using StringBuilder and StringBuffer.

# equals() on String and StringBuffer objects in Java

[**1**](https://www.geeksforgeeks.org/basic/)

Consider the following codes in java:

|  |
| --- |
| // This program prints false  class GFG {      public static void main(String[] args) {      StringBuffer sb1 = new StringBuffer("GFG");      StringBuffer sb2 = new StringBuffer("GFG");      System.out.println(sb1.equals(sb2));    }  } |

Run on IDE

**Output:**

false

|  |
| --- |
| // This program prints true  class GFG {      public static void main(String[] args) {      String s1 = "GFG";      String s2 = "GFG";      System.out.println(s1.equals(s2));    }  } |

Run on IDE

**Output:**

true

The output is false for the first example and true for the second example. In second example, parameter to equals() belongs [String class](https://www.geeksforgeeks.org/string-class-in-java/), while in second example it to [StringBuffer class](https://www.geeksforgeeks.org/stringbuffer-class-in-java/). When an object of String is passed, the strings are compared. But when object of StringBuffer is passed references are compared because StringBuffer does not [override equals method of Object class](https://www.geeksforgeeks.org/overriding-equals-method-in-java/).

For example, following first program prints false and second prints true.

|  |
| --- |
| // This program prints false  class GFG {      public static void main(String[] args) {      String s1 = "GFG";      StringBuffer sb1 = new StringBuffer("GFG");      System.out.println(s1.equals(sb1));    }  } |

Run on IDE

**Output:**

false

|  |
| --- |
| // This program prints true  class GFG {      public static void main(String[] args) {      String s1 = "GFG";      StringBuffer sb1 = new StringBuffer("GFG");      String s2 = sb1.toString();      System.out.println(s1.equals(s2));    }  } |

Run on IDE

**Output:**

true

# [2 Ways to sort HashMap in Java? Example](http://www.java67.com/2014/04/2-ways-to-sort-hashmap-in-java-example.html)

So you have a Map in your Java program and you want to process its data in sorted order. Since Map doesn't guarantee any order for its keys and values, you always end up with unsorted keys and Map. If you really need a sorted Map then think about using TreeMap, which keeps all keys in a sorted order. This could be either natural order of keys (defined by [Comparable](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html)) or a custom order (defined by [Comparator](http://java67.blogspot.sg/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html)), which you can provide while creating an instance of TreeMap. If you don't have your data in a sorted Map then only option remains is to get the keys, sort them and then process data in that order. Since keys are unique in Map, it returns a Set of keys, which means you cannot sort them by using Collections.sort() method, which accept a List. So what to do? Well, we can convert our Set into List as shown in [this](http://java67.blogspot.sg/2012/07/copy-elements-from-list-to-set-in-java-collection-example.html)example, then sort them using sort() method in any order and process them accordingly.

[Read more »](http://www.java67.com/2014/04/2-ways-to-sort-hashmap-in-java-example.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[No comments:](http://www.java67.com/2014/04/2-ways-to-sort-hashmap-in-java-example.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=6595992729929928502)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=6595992729929928502)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6595992729929928502&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6595992729929928502&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6595992729929928502&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6595992729929928502&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6595992729929928502&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java collection tutorial](http://www.java67.com/search/label/Java%20collection%20tutorial?&max-results=3), [Java Interview Question](http://www.java67.com/search/label/Java%20Interview%20Question?&max-results=3)

# [Difference between Static and Dynamic Binding in Java](http://www.java67.com/2014/02/difference-between-static-and-dynamic.html)

When you call a method in Java, it is resolved either at compile time or at runtime, depending upon whether it's a virtual method or a static method. When a method call is resolved at compile time, it is known as **static binding**, while if method invocation is resolved at runtime, it is known as **Dynamic binding or Late binding**. Since Java is an object-oriented programming language and by virtue of that it supports [Polymorphism](http://java67.blogspot.sg/2012/10/difference-between-polymorphism-overloading-overriding-java.html). Because of polymorphism, a reference variable of type Parent can hold an object of type Child, which extends Parent. Now if you call a virtual method (not private, final or static) on this object, then Compiler can not find actual method, because it could be the one, which is defined in the Parent class, or the one which Child has overridden. This call can only be resolved at runtime when the actual object is available. That's why this is known as *runtime or dynamic binding*.

[Read more »](http://www.java67.com/2014/02/difference-between-static-and-dynamic.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[1 comment:](http://www.java67.com/2014/02/difference-between-static-and-dynamic.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=7886517836893804033)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=7886517836893804033)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7886517836893804033&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7886517836893804033&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7886517836893804033&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7886517836893804033&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7886517836893804033&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java Interview Question](http://www.java67.com/search/label/Java%20Interview%20Question?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3)

# [How HashSet Internally Works in Java](http://www.java67.com/2014/01/how-hashset-is-implemented-or-works-internally-java.html)

Not many programmer know that HashSet is internally implemented using HashMap in Java, so if you know [How HashMap works internally in Java](http://java67.blogspot.sg/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html), more likely you can figure out *how HashSet works in Java*. But, now a curious Java developer can question that, how come HashSet uses HashMap, because you need a key value pair to use with Map, while in HashSet we only store one object. Good question, isn't it? If you remember some functionality of earlier class, then you know that [HashMap allows duplicate values](http://java67.blogspot.sg/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html) and this property is exploited while implementing HashSet in Java. Since HashSet implements Set interface it needs to guarantee uniqueness and this is achieved by storing elements as keys with same value always. Things gets clear by checking HashSet.java from JDK source code. All you need to look at is, how elements are stored in HashSet and how they are retrieved from HashSet. Since HashSet doesn't provide any direct method for retrieving object e.g. get(Key key) from HashMap or get(int index) from List, only way to get object from HashSet is via Iterator. See [here](http://java67.blogspot.sg/2012/10/how-to-iterate-over-hashset-in-java.html)for code example of iterating over HashSet in Java. When you create an object of HashSet in Java, it internally create instance of backup Map with default initial capacity 16 and default load factor 0.75 as shown below :

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2014-09-10T07:09:00-07:00&max-results=6#ixzz4i1XdqDT7>

# Why String is immutable or final in Java

Why String is [immutable](https://www.journaldev.com/129/how-to-create-immutable-class-in-java) in java is one of the popular interview question. String is one of the most used classes in any programming language. We know that String is immutable and final in java and java runtime maintains a [String pool](https://www.journaldev.com/797/what-is-java-string-pool) that makes it a special class.

## Why String is immutable in Java?

Let’s look at some benefits of String immutability, that will help in understanding why String is immutable in java.

1. [String pool](https://www.journaldev.com/797/what-is-java-string-pool) is possible only because String is immutable in java, this way Java Runtime saves a lot of java heap space because different String variables can refer to same String variable in the pool. If String would not have been immutable, then String interning would not have been possible because if any variable would have changed the value, it would have been reflected to other variables also.
2. If String is not immutable then it would cause severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](https://www.journaldev.com/741/java-socket-programming-server-client) host and port details passed as String. Since String is immutable it’s value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.
3. Since String is immutable, it is safe for [multithreading](https://www.journaldev.com/1079/multithreading-in-java) and a single String instance can be shared across different threads. This avoid the usage of synchronization for thread safety, Strings are implicitly thread safe.
4. Strings are used in [java classloader](https://www.journaldev.com/349/java-classloader) and immutability provides security that correct class is getting loaded by Classloader. For example, think of an instance where you are trying to load java.sql.Connection class but the referenced value is changed to myhacked.Connection class that can do unwanted things to your database.
5. Since String is immutable, its **hashcode** is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

Above are some of the reasons I could think of that shows benefits of String immutability. It’s a great feature of [Java String](https://www.journaldev.com/16928/java-string) class and makes it special.

# 

# [Can we Override Private Method in Java? Inner Class?](http://www.java67.com/2013/08/can-we-override-private-method-in-java-inner-class.html)

No, you cannot override private methods in Java, private methods are non-virtual in Java and access differently than non-private one. Since [method overriding](http://java67.blogspot.com/2012/08/what-is-method-overriding-in-java-example-tutorial.html) can only be done on derived class and private methods are not accessible in a subclass, you just can not override them. By the way, one more possibility of overriding private methods in an inner class, since private methods are accessible in an inner class, and that’s why it is one of the [tricky java interview questions](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html). Anyway, this will also not work because private methods are bonded during compile time and only Type (or Class) is used to locate a private method. For Example in below code where it looks like that [nested class](http://java67.blogspot.com/2012/10/nested-class-java-static-vs-non-static-inner.html) is an overriding private method, but if you call privateMethod() with a type of super class but the object of the subclass, it will only execute privateMethod() declared in the parent class, which is not exactly method overriding.

[Read more »](http://www.java67.com/2013/08/can-we-override-private-method-in-java-inner-class.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[7 comments:](http://www.java67.com/2013/08/can-we-override-private-method-in-java-inner-class.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4366972295390608191)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4366972295390608191)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4366972295390608191&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4366972295390608191&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4366972295390608191&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4366972295390608191&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4366972295390608191&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3) Location: [United States](https://maps.google.com/maps?q=United+States@37.09024,-95.71289100000001&z=10)

# [Best Way to Iterate Over Each Entry of HashMap in Java](http://www.java67.com/2013/08/best-way-to-iterate-over-each-entry-in.html)

What is the best way to Iterate over HashMap in Java? and not just HashMap, but any Map implementation including old Hashtable, TreeMap, LinkedHashMap and relatively newer ConcurrentHashMap, is a frequently asked query from Java Programmers, with some experience under his belt. Well, when it comes to choosing between [different ways to iterate over Map in Java](http://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html), it's you need, which plays an important role. For example, if you just want to iterate over each entry of HashMap, without modifying Map, then iterating over entry set using Java 1.5 foreach loop seems the most elegant solution to me. The reason, it just two lines of code using a foreach loop and Generics, and by getting the set of entries, we get key and value together, without further searching in HashMap. This makes it also the fastest way to loop over HashMap in Java.

[Read more »](http://www.java67.com/2013/08/best-way-to-iterate-over-each-entry-in.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[7 comments:](http://www.java67.com/2013/08/best-way-to-iterate-over-each-entry-in.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8776434022551483645)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8776434022551483645)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8776434022551483645&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8776434022551483645&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8776434022551483645&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8776434022551483645&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8776434022551483645&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java collection tutorial](http://www.java67.com/search/label/Java%20collection%20tutorial?&max-results=3) Location: [United States](https://maps.google.com/maps?q=United+States@37.09024,-95.71289100000001&z=10)

# [Difference between Comparator and Comparable in Java - Interview Question](http://www.java67.com/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html)

Comparator and Comparable are two interfaces in Java API, which is used to compare two objects in Java. Though both are used for comparison there are some differences between them, a major difference between Comparable and Comparator is that former is used to define the natural ordering of object e.g. lexicographic order for java.lang.String, while later is used to define any alternative ordering for an object.  The main usage of java.lang.Comparable and java.util.Comparator interface is for [sorting a list of objects in Java](http://java67.blogspot.com/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html). For example to sort a list of Employee by their Id, we can use Comparable interface and provide additional sorting capability, we can define multiple comparators e.g. AgeComparator to compare the age of the employee, SalaryComparator to compare the salary of employees etc.  This brings another i*mportant difference between Comparator and Comparable interface in Java*, you can have only one ordering via Comparable e.g. natural ordering while you can define multiple Comparator for alternative ordering as discussed above.

# [15 Java Enum Interview Questions for Developers with Answers](http://www.java67.com/2013/07/15-java-enum-interview-questions-amswers-for-experienced-programmers.html)

Enum was introduced in Java 5 and since then it's been very popular among Java developers and widely used in different Java applications. Since Enum in Java is much more versatile than Enum in C or C++, it also presents lots of interesting use cases, couple of them, we have seen in my article [10 ways to use Enum in Java](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html). But, despite being so popular, many Java programmer are still not aware of functionality provided by Enum and subtle details of using Enum in Java code. I realized this fact, when couple of my readers asked me some of the questions e.g. **Can Enum implement an interface in Java** or **Why we can not create Enum instances outside of Enum**, stating that these has been asked to them in there Java Interviews. This motivates me to put together a list of frequently asked question in Java Enum, which not only helps to do well in Interviews, but also open new path for learning. As I had said before, lot of time a question in Interviews, makes you to take a topic more seriously than otherwise, which is not a bad thing, and given the power and facilities offered by Java Enum, I think it's high time to get master of it.

[Read more »](http://www.java67.com/2013/07/15-java-enum-interview-questions-amswers-for-experienced-programmers.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[5 comments:](http://www.java67.com/2013/07/15-java-enum-interview-questions-amswers-for-experienced-programmers.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4543208265068388382)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4543208265068388382)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4543208265068388382&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4543208265068388382&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4543208265068388382&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4543208265068388382&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4543208265068388382&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [enum](http://www.java67.com/search/label/enum?&max-results=3)

# [Difference between this and super keywords in Java](http://www.java67.com/2013/06/difference-between-this-and-super-keyword-java.html)

this and super are two special keywords in Java, which is used to represent current instance of a class and it's super class. Java Programmers often confused between them and not very familiar with there special properties, which is asked at various [core Java interviews](http://java67.blogspot.com/2012/09/top-10-tough-core-java-interview-questions-answers.html). A couple of questions, which I remember about this and super keyword is  that, **Can we reassign this in Java?**  and the ***difference between this and super keyword in Java***. Do you want to try that? Ok, I am not giving the answer now, rather I will let you know the answer at the end of this post. As I said in the first line, the main difference between this and super in Java is that this represents current instance of a class, while super represent current instance of the parent class. Now where does this and super variables used, well you might have seen examples of calling one constructor from other i.e. [constructor chaining](http://java67.blogspot.com/2012/12/how-constructor-chaining-works-in-java.html), that's achieved by using this and super keyword?

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2014-01-22T06:49:00-08:00&max-results=6#ixzz4i1XxK3Qz>

[How get method of HashMap or Hashtable works internally in Java](http://www.java67.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html)

In this article, I am revisiting a couple of interesting question related to the internal working of HashMap in Java, mostly asked senior Java developers, ranging from 4 to 6 and up to 8 years of experience. I did cover lot of these questions from HashMap, ranging from thread-safety to race conditions, in my post about [internal working of Java HashMap](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), but I thought to revisit two of those questions, *How get method of HashMap or Hashtable works internally in Java* and *What happens if two different keys return the same hashCode*, how do you return value from HashMap in that case. These are the question, which is highly popular in investment banking domain, and preferred choice of interviewer, while interviewing experienced Java professional.

[Read more »](http://www.java67.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[8 comments:](http://www.java67.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4545729862003998075)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4545729862003998075)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4545729862003998075&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4545729862003998075&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4545729862003998075&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4545729862003998075&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4545729862003998075&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java collection tutorial](http://www.java67.com/search/label/Java%20collection%20tutorial?&max-results=3), [Sorting Algorithm](http://www.java67.com/search/label/Sorting%20Algorithm?&max-results=3)

# Difference between Deep and Shallow Copy in Java Object Cloning

Shallow copy and deep copy is related with cloning process so before go into the deep of shallow and deep copy we need to understand what is clone in java. Clone is nothing but the process of copying one object to produce the exact object, which is not guaranteed. We all know in Java object is referred by reference we can not copy one object directly to another object. So we have cloning process to achieve this objective. Now one question arises in mind why we need this process so the answer is whenever we need a local copy of the object to modify the object in some method but not in method caller.  So we can define Cloning as “**create a copy of object** “  .I think now we are somehow clear about the cloning but there  is more to it depending on how we are doing this copy, we can divide cloning into two types.

* Shallow Copy
* Deep Copy

Before going into the deep of shallow and deep copy we need to understand how we achieve cloning in java.

### How to Clone in java?

[Difference between deep cloning vs shallow cloning in Java](http://3.bp.blogspot.com/-1lzFJzIgaHk/UF2Ci6kY5pI/AAAAAAAAAes/OYiM7r-DHzc/s1600/17.jpg)In Java, everything is achieved through class, object and interface .By default no Java class support cloning but Java provide one interface called Cloneable, which is a [marker interface](http://javarevisited.blogspot.com/2012/01/what-is-marker-interfaces-in-java-and.html) andbyimplementingthis interface we can make the duplicate copy of our object by calling clone() method of  java.lang.Object class.   
  
  
  
This Method is protected inside the object class and Cloneable interface is  a marker interface and this method also throw  **CloneNotSupportedException**if we have not implemented thisinterface and try to call clone() method of Object class.  By default any clone() method gives the **shallow copy** of the object i.e. if we invoke **super.clone()** then it’s a shallow copy but if we want to **deep copy** we have to override the clone() method and make it public and give own definition of making copy of object. Now we let’s see  what is shallow and deep copy of object in Java programming language.

### Shallow Copy

Whenever we use default implementation of clone method we get shallow copy of object means it create new instance and copy all the field of object to that new instance and return it as **object type** we need to explicitly cast it back to our original object. This is shallow copy of the object. clone() method of the object class support shallow copy of the object. If the object contains primitive as well as non primitive or reference type variable In  shallow copy, the cloned object also refers to the same object to which the original object refers as only the object references gets copied and not the referred objects themselves. That's why the name shallow copy or shallow cloning in Java. If only primitive type fields or [Immutable objects](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html) are there then there is no difference between shallow and deep copy in Java.

### Deep Copy

Whenever we need own meaning of copy not to use default implementation we call it as deep copy, whenever we need deep copy of the object we need to implement according to our need. So for deep copy we need to ensure all the member class also implement the Cloneable interface and override the clone() method of the object class. After that we override the clone() method in all those classes even in the classes where we have only primitive type members otherwise we would not be able to call the protected clone() method of Object class on the instances of those classes inside some other class. It’s typical restriction of the protected access.

### Difference between Shallow and Deep Copy in Java

I think now we know what is deep and shallow copy of object in Java, let see some difference between them so that we can get some more clarity on them.

* When we call Object.clone(), this method performs a shallow copy of object, by copying data field by field, and if we override this method and by convention first call super.clone(), and then modify some fields to "deep" copy, then we get deep copy of object. This modification is done to ensure that original and cloned object are independent to each other.
* In shallow copy main or parent object is copied, but they share same fields or children if fields are modified in one parent object other parent fields have automatic same changes occur,but in deep copy this is not the case.
* If our parent object contains only primitive value then shallow copy is good for making clone of any object because in new object value is copied but if parent object contains any other object then only reference value is copied in new parent object and both will point to same object so in that case according to our need we can go for deep copy.
* Deep copy is expensive as compare to shallow copy in terms of object creation, because it involves recursive copying of data from other mutable objects, which is part of original object.

This is all about deep copy and shallow copy of objects in Java. Now the question comes when we use shallow copy and when go for deep copy , so answer would be  simple that if the object has only primitive fields or Immutable objects, then obviously we will go for shallow copy, but if the object has references to other mutable objects, then based on the requirement, shallow copy or deep copy can be chosen. Means  if the references are not modified anytime, then there is no point in going for deep copy, We can go for shallow copy. But if the references are modified often, then you need to go for deep copy. Again there is no hard and fast rule, it all depends on the requirement.

Hope this article will help to make clear about deep and shallow copy of cloning process.

Read more: <http://www.java67.com/2013/05/difference-between-deep-copy-vs-shallow-cloning-java.html#ixzz5UpdMvnE8>

# Java transient keyword example

By Lokesh Gupta | Filed Under: [Java Basics](https://howtodoinjava.com/java/basics/)

The **Java transient keyword** is used on class attributes/variables to indicate that serialization process of such class should ignore such variables while creating a persistent byte stream for any instance of that class.

**A transient variable is a variable that can not be serialized. According to Java Language Specification [**[**jls-8.3.1.3**](https://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.3.1.3)**] – “Variables may be marked transient to indicate that they are not part of the persistent state of an object.”**

In this post, I will discussing various concepts involved around usage of **transient** keyword in context of [**serialization**](https://howtodoinjava.com/java/serialization/a-mini-guide-for-implementing-serializable-interface-in-java/).

Table of Contents

1. [What is transient keyword in Java?](https://howtodoinjava.com/java/basics/transient-keyword-in-java-with-real-time-example/#what_is_transient)

2. [When should we use transient keyword in java?](https://howtodoinjava.com/java/basics/transient-keyword-in-java-with-real-time-example/#when_to_use_transient)

3. [Usage of transient with final keyword](https://howtodoinjava.com/java/basics/transient-keyword-in-java-with-real-time-example/#transient_with_final)

4. [Case study: How does a HashMap use transient?](https://howtodoinjava.com/java/basics/transient-keyword-in-java-with-real-time-example/#transient_case_study)

5. [Summary Notes](https://howtodoinjava.com/java/basics/transient-keyword-in-java-with-real-time-example/#summary)

## 1. What is Java transient keyword

The modifier **transient** in java can be applied to field members of a class to turn off serialization on these field members. Every field marked as **transient** will not be serialized. You use the **transient**keyword to indicate to the java virtual machine that the **transient** variable is not part of the persistent state of an object.

Let’s write an very basic example to understand what exactly above analogy means. I will create an Employee class and will define 3 attributes i.e. firstName, lastName and confidentialInfo. We do not want to store/save “**confidentialInfo**” for some purpose so we will mark the field as “**transient**“.

class Employee implements Serializable

{

private String firstName;

private String lastName;

private transient String confidentialInfo;

//Setters and Getters

}

Now let’s **serialize an instance of Employee** class.

try

{

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("empInfo.ser"));

Employee emp = new Employee();

emp.setFirstName("Lokesh");

emp.setLastName("Gupta");

emp.setConfidentialInfo("password");

//Serialize the object

oos.writeObject(emp);

oos.close();

} catch (Exception e)

{

System.out.println(e);

}

Now let’s **de-serialize back into java object**, and verify if “confidentialInfo” was saved or not?

try

{

ObjectInputStream ooi = new ObjectInputStream(new FileInputStream("empInfo.ser"));

//Read the object back

Employee readEmpInfo = (Employee) ooi.readObject();

System.out.println(readEmpInfo.getFirstName());

System.out.println(readEmpInfo.getLastName());

System.out.println(readEmpInfo.getConfidentialInfo());

ooi.close();

} catch (Exception e)

{

System.out.println(e);

}

Program Output.

Lokesh

Gupta

null

Clearly, “confidentialInfo” was not saved to persistent state while serialization and that’s exactly why we use “**transient**” keyword in java.

## 2. When should we use transient keyword in java?

Now we have a very knowledge of “**transient**” keyword. Let’s expand out understanding by identifying the situations where you will need the **use of transient keyword**.

1. First and very logical case would be where you may have **fields that are derived/calculated from other fields** within instance of class. They should be calculated programmatically everytime rather than having the state be persisted via serialization. An example could be time-stamp based value; such as age of a person OR duration between a timestamp and current timestamp. In both cases, you would be calculating value of variable based on current system time rather than when the instance was serialized.
2. Second logical example can be**any secure information** which should not leak outside the JVM in any form (either in database OR byte stream).
3. Another example could be fields which are **not marked as “Serializable”** inside JDK or application code. Classes which do not implement Serializable interface and are referenced within any serializable class, cannot be serialized; and will throw “java.io.NotSerializableException” exception. These non-serializable references should be marked “transient” before serializing the main class.
4. And lastly, there are times when it **simply doesn’t make sense to serialize some fields**. Period. For example, In any class if you have added a logger reference, then what’s use of serializing that logger instance. Absolutely no use. You serialize the information which represent the state of instance, logically. Loggers never share the state of an instance. They are just utilities for programming/debugging purpose. Similar example can be reference of a Thread class. Threads represent a state of a process at any given point of time, and there is no use to store thread state with your instance; simply because they do not constitute the state of your class’s instance.

Above four usecases are when you should use the keyword “**transient**” with reference variables. If you have some more logical cases where “**transient**” can be used; please share with me and I will update that here in list so that everybody can benefit from your knowledge.

Read More : [A min guide for implementing serializable interface](https://howtodoinjava.com/java/serialization/a-mini-guide-for-implementing-serializable-interface-in-java/)

## 3. Transient with final

I am talking about use of **transient** with **final** keyword specifically because it behaves differently in different situations which is not generally the case with other keywords in java.

For making this concept practical, I have modified the Employee class as below:

private String firstName;

private String lastName;

//final field 1

public final transient String confidentialInfo = "password";

//final field 2

public final transient Logger logger = Logger.getLogger("demo");

Now when I again run the serialization (write/read) again, below is the output:

Program output.

Lokesh

Gupta

password

null

This is strange. We have marked the “confidentialInfo” to transient; and still the field was serialized. For similar declaration, logger was not serialized. Why?

Reason is that whenever any final field/reference is evaluated as “[**constant expression**](https://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html#jls-15.28)“, it is serialized by JVM ignoring the presence of transient keyword.

In above example, value “password” is a constant expression and instance of logger “demo” is reference. So by rule, confidentialInfo was persisted where as logger was not.

Are you thinking, what if I remove “transient” from both fields? Well, then fields implementing Serializable references will persist otherwise not. So, if you remove transient in above code, String (which implements Serializable) will be persisted; where as Logger (which does NOT implements Serializable) will not be persisted AND “java.io.NotSerializableException” will be thrown.

If you want to persist the state of non-serializable fields then use readObject() and writeObject() methods. writeObject()/readObject() are usually chained into serialization/deserialization mechanisms internally and thus called automatically.

Read More : [SerialVersionUID in java and related fast facts](https://howtodoinjava.com/java/serialization/serialversionuid/)

## 4. Case study : How does a HashMap use transient keyword?

So far, we have been talking about **concepts related to “transient” keyword** which are mostly theoretical in nature. Let’s understand the proper use of “**transient**” which is **used inside HashMap class**very logically. It will give you better idea about **real life usage of transient keyword in java**.

Before understanding the solution which has been created using transient, let’s **first identify the problem** itself.

HashMap is used to store key-value pairs, we all know that. And we also know that location of keys inside HashMap is calculated based on hash code obtained for instance of key. Now when we serialize a HashMap that means all keys inside HashMap and all values respective to key’s will also be serialized. After serialization, when we de-serialize the HashMap instance then all key instances will also be deserialized. We know that during this serialization/deserialization process, there may be loss of information (used to calculate hashcode) as well as most important is it is a NEW INSTANCE itself.

In java, **any two instances (even of same class) can not have same hashcode**. This is a big problem because the location where keys should be placed according to new hashcodes, are not in there correct positions. When retrieving the value of a key, you will be referring to the wrong index in this new HashMap.

Read More : [Working with hashCode and equals methods in java](https://howtodoinjava.com/java/basics/java-hashcode-equals-methods/)

So, when a hash map is serialized, it means that the hash index, and hence the ordering of the table is no longer valid and should not be preserved. This is the problem statement.

Now look at how it is solved inside HashMap class. If go through the sourcecode of [**HashMap.java**](http://www.docjar.com/html/api/java/util/HashMap.java.html), you will find below declarations:

transient Entry table[];

transient int size;

transient int modCount;

transient int hashSeed;

private transient Set entrySet;

All important fields have been marked at “**transient**” (all of them are actually calculated/change at runtime), so they are not part of serialized HashMap instance. To populate again this important information back, HashMap class uses **writeObject()** and **readObject()** methods as below:

private void writeObject(ObjectOutputStream objectoutputstream) throws IOException

{

objectoutputstream.defaultWriteObject();

if (table == EMPTY\_TABLE)

objectoutputstream.writeInt(roundUpToPowerOf2(threshold));

else

objectoutputstream.writeInt(table.length);

objectoutputstream.writeInt(size);

if (size > 0)

{

Map.Entry entry;

for (Iterator iterator = entrySet0().iterator(); iterator.hasNext(); objectoutputstream.writeObject(entry.getValue()))

{

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

objectoutputstream.writeObject(entry.getKey());

}

}

}

private void readObject(ObjectInputStream objectinputstream) throws IOException, ClassNotFoundException

{

objectinputstream.defaultReadObject();

if (loadFactor <= 0.0F || Float.isNaN(loadFactor))

throw new InvalidObjectException((new StringBuilder())

.append("Illegal load factor: ").append(loadFactor).toString());

table = (Entry[]) EMPTY\_TABLE;

objectinputstream.readInt();

int i = objectinputstream.readInt();

if (i < 0)

throw new InvalidObjectException((new StringBuilder()).append("Illegal mappings count: ").append(i).toString());

int j = (int) Math.min((float) i \* Math.min(1.0F / loadFactor, 4F), 1.073742E+009F);

if (i > 0)

inflateTable(j);

else

threshold = j;

init();

for (int k = 0; k < i; k++)

{

Object obj = objectinputstream.readObject();

Object obj1 = objectinputstream.readObject();

putForCreate(obj, obj1);

}

}

With above code, HashMap still let the non-transient fields to be treated as they would normally do, but they wrote the stored key-value pairs at the end of the byte array one after the other. While de-serializing, it let the non-transient variables to be handled by default de-serialization process and then read the key-value pairs one by one. **For each key the hash and the index is calculated again and is inserted to the correct position in the table** so that it can be retrieved again without any error.

Above use of transient keyword was a very good example of proper usecase. You should keep remember it and mention it whenever it is asked in your next [java interview question](https://howtodoinjava.com/java-interview-questions/).

Related Post: [How HashMap works in Java?](https://howtodoinjava.com/java/collections/how-hashmap-works-in-java/)

## 5. Summary Notes

1. The modifier transient can be applied to field members of a class to turn off serialization on these field members.
2. You can use transient keyword in classes with fields which needs to be secured or calculated on existing state fields. And use it when it simply doesn’t make sense to serialize those fields such as loggers and threads.
3. Serialization does not care about access modifiers such as private; all non-transient fields are considered part of an object’s persistent state and are eligible for persistence.
4. Whenever any final field/reference is evaluated as “constant expression”, it is serialized by JVM ignoring the presence of transient keyword.
5. A good usecase of transient keyword in java is HashMap class.

[How to Override Equals, HashCode and CompareTo method in Java](http://www.java67.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html)

Though modern IDE like Eclipse, IntelliJ or Netbeans allows you to generate equals, hashCode and compareTo methods for your value classes, it's equally important, you know how to do that by hand. By overriding equals and hashcode method by yourself, you know how they work, what kind of errors you can get, and most importantly, it's expected form you, as a Java programmer in any [core Java interview](http://java67.blogspot.com/2012/09/top-10-tough-core-java-interview-questions-answers.html). More often, you would see a coding question in Java, which ask you to override equals(), hashcode(), compare() and compareTo() methods for a value class. Since I have already shared some tips on How to override compareTo method in Java, and couple of example of writing your own Comparator in Java, here I am sharing another simple example of overriding equals and hashCode methods. If you know rules of overriding equals and hashCode, you might know that, whenever you override equals, you must have to override hashCode, otherwise your object will not behave properly on various collection classes e.g. [Map or Set](http://java67.blogspot.sg/2013/01/difference-between-set-list-and-map-in-java.html), which uses equals, compareTo, hashCode to implement there invariants e.g. Set implementations should not allow any duplicates.

[Read more »](http://www.java67.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[5 comments:](http://www.java67.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=6623235597521393353)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=6623235597521393353)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6623235597521393353&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6623235597521393353&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6623235597521393353&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6623235597521393353&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6623235597521393353&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java programming Tutorial](http://www.java67.com/search/label/Java%20programming%20Tutorial?&max-results=3)

[Difference between notify and notifyAll in Java](http://www.java67.com/2013/03/difference-between-wait-vs-notify-vs-notifyAll-java-thread.html)

wait, notify, and notifyAll methods are used for inter-thread communication in Java. wait() allows a thread to check for a condition, and wait if the condition doesn't meet, while notifying() and notifyAll() method informs waiting for a thread for rechecking condition, after changing the state of a shared variable. One good example of how to wait and notify method works is [Producer consumer problem](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html), where one thread produces and wait if the bucket is full; and another thread consumes and waits if the bucket is empty. Both Producer and Consumer thread, notify each other as well. Producer thread notifies consumer thread after inserting an item in the shared queue, while consumer thread notifies producer, after consuming item from the queue. Though Both notify() and notifyAll()  are used to notify waiting for threads, waiting on shared queue object, but there are some subtle differences between notify and notifyAll in Java.

[Read more »](http://www.java67.com/2013/03/difference-between-wait-vs-notify-vs-notifyAll-java-thread.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[1 comment:](http://www.java67.com/2013/03/difference-between-wait-vs-notify-vs-notifyAll-java-thread.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8518867980212190279)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8518867980212190279)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8518867980212190279&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8518867980212190279&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8518867980212190279&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8518867980212190279&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8518867980212190279&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [thread](http://www.java67.com/search/label/thread?&max-results=3)

[Difference between JDK and JRE in Java Platform](http://www.java67.com/2013/02/difference-between-jdk-and-jre-in-java.html)

Java Platform offers JRE and JDK to run Java programs. JRE stands for Java runtime environment and JDK stands for Java development kit. JRE is meant for normal users, who wants to run Java program in their computer. JRE is normally used to run Java programs downloadedover internet e.g. Java Applets and Java Desktop application built using [AWT and Swing](http://java67.blogspot.com/2013/01/10-awt-swing-interview-questions-answers-java.html). The main difference between JRE and JDK, comes from the fact that they are different tools. JDK is created for Java programmers and contains tools required for Java programming, e.g. javacfor compiling Java source files to [.class files](http://javarevisited.blogspot.com/2012/05/10-points-about-class-file-in-java.html). Without JDK, you can not create Java applications and programs. By the way, JDK comes with its own JRE, but when you run Java program using java command, the JRE which comes first in System PATH is used for execution.

[Read more »](http://www.java67.com/2013/02/difference-between-jdk-and-jre-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[2 comments:](http://www.java67.com/2013/02/difference-between-jdk-and-jre-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8569148139596326967)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=8569148139596326967)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8569148139596326967&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8569148139596326967&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8569148139596326967&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8569148139596326967&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=8569148139596326967&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3)

[Difference between JIT and JVM in Java - Interview Question](http://www.java67.com/2013/02/difference-between-jit-and-jvm-in-java.html)

The main difference between JIT and JVM is that JIT is part of JVM itself and its main function is to improve the performance of JVM by directly compiling some hot code (code which executes above a certain threshold) into native instruction. JIT stands for Just In time compilation and JVM stands for Java Virtual Machine. JVM is a virtual machine used in Java programming platform to execute or run Java programs. The main advantage of JVM is that [JVM  makes Java platform-independent](http://java67.blogspot.com/2012/08/how-java-achieves-platform-independence.html) by executing bytecodes. Java source code is compiled into class files, which contains bytecode. These byte codes are then executed by JVM. Now here comes JIT. Since the execution of bytecode is slower than the execution of machine language code, because JVM first needs to translate bytecode into machine language code.

# [Can abstract class have Constructor in Java - Interview Question](http://www.java67.com/2013/02/can-abstract-class-have-constructor-in-java.html)

Yes, an abstract class can have a constructor in Java. You can either explicitly provide a constructor to abstract class or if you don't, the compiler will add [default constructor](http://java67.blogspot.com/2012/12/how-constructor-chaining-works-in-java.html) of no argument in abstract class. This is true for all classes and it also applies to an abstract class. For those who want to recall what is an abstract class in Java, it's a class which can not be instantiated with new() operator or any other ways. In order to use an abstract class in Java,  You need to extend it and provide a concrete class. Abstract class is commonly used to define a base class for a type hierarchy with default implementation, which is applicable to all child classes. By the way, [difference between interface and abstract class in Java](http://java67.blogspot.sg/2012/09/what-is-difference-between-interface-abstract-class-java.html) is also one of the popular and tricky Java questions and should be prepared well for Java interviews.

[Read more »](http://www.java67.com/2013/02/can-abstract-class-have-constructor-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[3 comments:](http://www.java67.com/2013/02/can-abstract-class-have-constructor-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3495565766918344761)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3495565766918344761)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3495565766918344761&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3495565766918344761&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3495565766918344761&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3495565766918344761&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3495565766918344761&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java Interview Question](http://www.java67.com/search/label/Java%20Interview%20Question?&max-results=3)

# [Difference between Set, List and Map in Java - Interview question](http://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html)

Set, List and Map are three important interface of Java collection framework and Difference between Set, List and Map in Java is one of the most frequently asked [Java Collection interview question](http://java67.blogspot.com/2012/09/java-collection-interview-questions.html). Some time this question is asked as When to use List, Set and Map in Java. Clearly, interviewer is looking to know that whether you are familiar with fundamentals of Java collection framework or not. In order to decide when to use List, Set or Map , you need to know what are these interfaces and what functionality they provide. [List in Java](http://java67.blogspot.com/2012/07/sort-list-ascending-descending-order-set-arraylist.html) provides ordered and indexed collection which may contain duplicates. Set provides an un-ordered collection of unique objects, i.e. Set doesn't allow duplicates, while Map provides a data structure based on key value pair and hashing. All three List, Set and Map are interfaces in Java and there are many concrete implementation of them are available in Collection API. ArrayList and LinkedList are two most popular used List implementation while [LinkedHashSet, TreeSet and HashSet](http://javarevisited.blogspot.com/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html) are frequently used Set implementation. In this Java article we will see *difference between Map, Set and List in Java* and learn when to use List, Set or Map.

[Read more »](http://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[26 comments:](http://www.java67.com/2013/01/difference-between-set-list-and-map-in-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3814741245718182578)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3814741245718182578)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3814741245718182578&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3814741245718182578&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3814741245718182578&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3814741245718182578&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3814741245718182578&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java collection tutorial](http://www.java67.com/search/label/Java%20collection%20tutorial?&max-results=3)

# [Difference between synchronized block and method in Java](http://www.java67.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html)

Synchronized block and synchronized methods are two ways to use [synchronized keyword in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) and implement mutual exclusion on critical section of code. Since Java is mainly used to write multi-threading programs,  which present various kinds of thread related issues like [thread-safety](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html), [deadlock](http://www.blogger.com/javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html) and [race conditions](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html), which plagues into code mainly because of poor understanding of synchronization mechanism provided by Java programming language. Java provides inbuilt synchronized and [volatile keyword](http://java67.blogspot.sg/2012/08/what-is-volatile-variable-in-java-when.html) to achieve synchronization in Java. Main *difference between synchronized method and synchronized block* is selection of lock on which critical section is locked. Synchronized method depending upon whether its a [static method](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) or non static locks on either class level lock or object lock. Class level lock is one for each class and represented by class literal e.g. Stirng.class. Object level lock is provided by current object e.g. this instance, You should [never mix static and non static synchronized method in Java](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html).. On the other hand synchronized block locks on monitor evaluated by expression provided as parameter to synchronized block. In next section we will see an example of both synchronized method and synchronized block to understand this difference better.

**How to use Volatile keyword in Java**  
What is volatile variable in Java and when to use  the volatile variable in Java is a famous [multi-threading interview question](http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html) in Java interviews. Though many programmers know what is a volatile variable but they fail on second part i.e. where to use volatile variable in Java as it's not common to have a clear understanding and hands-on on volatile in Java. In this tutorial, we will address this gap by providing a simple example of the volatile variable in Java and discussing some when to use the volatile variable in Java. Anyway,  the volatile keyword in Java is used as an indicator to Java compiler and Thread that do not cache value of this variable and always read it from [main memory](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html). So if you want to share any variable in which read and write operation is atomic by implementation e.g. read and write in an int or a boolean variable then  you can declare them as volatile variable.  
  
From Java 5 along with major changes like Autoboxing, Enum, Generics and Variable arguments , Java introduces some change in Java Memory Model (JMM), Which guarantees visibility of changes made from one thread to another also as "happens-before" which solves the problem of memory writes that happen in one thread can "leak through" and be seen by another thread.  
  
The Java volatile keyword cannot be used with method or class and it can only be used with a variable. Java volatile keyword also guarantees visibility and ordering, after Java 5 write to any volatile variable happens before any read into the volatile variable. By the way use of volatile keyword also prevents compiler or JVM from the reordering of code or moving away them from synchronization barrier.

## The Volatile variable Example in Java

To Understand example of volatile keyword in java let’s go back to [Singleton pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) and see [double checked locking in Singleton](http://javarevisited.blogspot.com/2014/05/double-checked-locking-on-singleton-in-java.html) with Volatile and without the volatile keyword in java.

/\*\*

\* Java program to demonstrate where to use Volatile keyword in Java.

\* In this example Singleton Instance is declared as volatile variable to ensure

\* every thread see updated value for \_instance.

\*

\* @author Javin Paul

\*/

**public** **class** **Singleton**{

**private** **static** **volatile** Singleton \_instance; //volatile variable

**public** **static** Singleton **getInstance**(){

**if**(\_instance == **null**){

**synchronized**(Singleton.class){

**if**(\_instance == **null**)

\_instance = **new** Singleton();

}

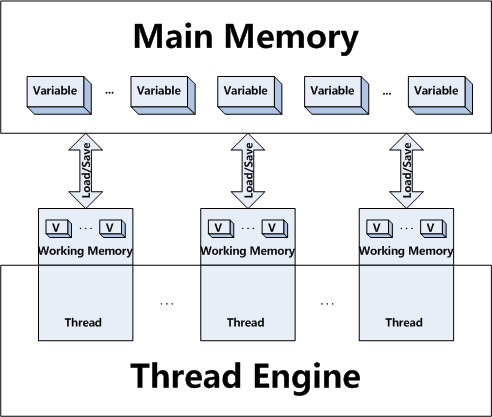
}

**return** \_instance;

}

If you look at the code carefully you will be able to figure out:  
1) We are only creating instance one time  
2) We are creating instance lazily at the time of the first request comes.

If we do not make the \_instance variable volatile than the Thread which is creating instance of Singleton is not able to communicate other thread, that instance has been created until it comes out of the Singleton block, so if Thread A is creating Singleton instance and just after creation lost the CPU, all other thread will not be able to see value of \_instance as not null and they will believe its still [null](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html).

[](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https://www.udemy.com/multithreading-and-parallel-computing-in-java/)

Why? because reader threads are not doing any locking and until writer thread comes out of synchronized block, memory will not be synchronized and value of \_instance will not be updated in main memory. With Volatile keyword in Java, this is handled by Java himself and such updates will be visible by all reader threads.  
  
So in Summary apart from [synchronized keyword in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), volatile keyword is also used to communicate the content of memory between threads.

### Let’s see another example of volatile keyword in Java

most of the time while writing game we use a variable bExit to check whether user has pressed exit button or not, value of this variable is updated in [event thread](http://javarevisited.blogspot.sg/2011/09/invokeandwait-invokelater-swing-example.html) and checked in game thread, So if we don't use volatile keyword with this variable, Game Thread might miss update from event handler thread if it's not synchronized in Java already. volatile keyword in java guarantees that value of the volatile variable will always be read from main memory and "*happens-before"* relationship in Java Memory model will ensure that content of memory will be communicated to different threads.

**private** **boolean** bExit**;**

**while(!**bExit**)** **{**

checkUserPosition**();**

updateUserPosition**();**

**}**

In this code example, One Thread (Game Thread) can cache the value of "bExit" instead of getting it from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) every time and if in between any other thread (Event handler Thread) changes the value; it would not be visible to this thread. Making boolean variable "bExit" as volatile in java ensures this will not happen.  
  
Also, If you have not read already then I also suggest you read the topic about volatile variable from [Java Concurrency in Practice](http://www.amazon.com/dp/0321349601/?tag=javamysqlanta-20) book by Brian Goetz, one of the must read to truly understand this complex concept.

## When to use Volatile variable in Java

One of the most important thing in learning of volatile keyword is understanding when to use volatile variable in Java. Many [programmer](http://javarevisited.blogspot.sg/2011/06/top-programming-interview-questions.html) knows what is volatile variable and how does it work but they never really used volatile for any practical purpose. Here are couple of example to demonstrate when to use Volatile keyword in Java:

1) You can use Volatile variable if you want to read and write long and [double](http://javarevisited.blogspot.sg/2011/10/convert-double-to-string-example.html) variable atomically. long and double both are [64 bit](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html) data type and by default writing of long and double is not atomic and platform dependence. Many platform perform write in long and double variable 2 step, writing 32 bit in each step, due to this its possible for a Thread to see 32 bit from two different write. You can avoid this issue by making long and double variable volatile in Java.  
  
  
2) A volatile variable can be used as an alternative way of achieving [synchronization in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) in some cases, like Visibility. with volatile variable, it's guaranteed that all reader thread will see updated value of the volatile variable once write operation completed, without volatile keyword different reader thread may see different values.  
  
  
3) volatile variable can be used to inform the compiler that a particular field is subject to be accessed by multiple threads, which will prevent the compiler from doing any reordering or any kind of optimization which is not desirable in a multi-threaded environment. Without volatile variable compiler can re-order the code, free to cache value of volatile variable instead of always reading from main memory. like following example without volatile variable may result in an [infinite loop](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)

**private** **boolean** isActive **=** thread**;**

**public** **void** printMessage**(){**

**while(**isActive**){**

System**.**out**.**println**(**"Thread is Active"**);**

**}**

**}**

without the *volatile modifier*, it's not guaranteed that one [Thread](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) sees the updated value of isActive from other thread. The compiler is also free to cache value of isActive instead of reading it from main memory in every iteration. By making isActive a volatile variable you avoid these issue.  
  
  
4) Another place where a volatile variable can be used is to fixing double checked locking in Singleton pattern. As we discussed in [Why should you use Enum as Singleton](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) that double checked locking was broken in Java 1.4 environment.

### Important points on Volatile keyword in Java

1. The volatile keyword in Java is only application to a variable and using volatile keyword with class and method is illegal.  
  
2. volatile keyword in Java guarantees that value of the volatile variable will always be read from main memory and not from Thread's local cache.  
  
3. In Java reads and writes are [atomic](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) for all variables declared using Java volatile keyword (including long and double variables).  
  
4. Using the volatile keyword in Java on variables reduces the risk of memory consistency errors because any write to a volatile variable in Java establishes a happens-before relationship with subsequent reads of that same variable.  
  
5. From Java 5 changes to a volatile variable are always visible to other threads. What's more, it also means that when a thread reads a volatile variable in Java, it sees not just the [latest change to the volatile variable](http://java67.blogspot.sg/2012/08/what-is-volatile-variable-in-java-when.html) but also the side effects of the code that led up the change.  
  
6. Reads and writes are atomic for reference variables are for most primitive variables (all types except long and double) even without the use of volatile keyword in Java.  
  
7. An access to a volatile variable in Java never has a chance to block, since we are only doing a simple read or write, so unlike a synchronized block we will never hold on to any lock or wait for any [lock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html).  
  
8. Java volatile variable that is an object reference may be null.  
  
9. Java volatile keyword doesn't mean atomic, its common misconception that after declaring volatile ++ will be atomic, to make the operation atomic you still need to ensure exclusive access using synchronized method or block in Java.  
  
10. If a variable is not shared between multiple threads, you don't need to use volatile keyword with that variable.

## Difference between synchronized and volatile keyword in Java

What is the difference between volatile and synchronized is another popular [core Java question](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html) asked on multi-threading and concurrency interviews. Remember volatile is not a replacement of synchronized keyword but can be used as an alternative in certain cases. Here are few differences between volatile and synchronized keyword in Java.  
  
1. The volatile keyword in Java is a field modifier while synchronized modifies code blocks and methods.  
  
2. Synchronized obtains and releases the lock on monitor’s Java volatile keyword doesn't require that.  
  
3. Threads in Java can be blocked for waiting for any monitor in case of synchronized, that is not the case with the [volatile keyword](http://java67.blogspot.com/2012/11/difference-between-transient-vs-volatile-modifier-variable-java.html) in Java.  
  
4. Synchronized method affects performance more than a volatile keyword in Java.  
  
5. Since volatile keyword in Java only synchronizes the value of one variable between Thread memory and "main" memory while synchronized synchronizes the value of all variable between thread memory and "main" memory and locks and releases a monitor to boot. Due to this reason synchronized keyword in Java is likely to have more overhead than volatile.  
  
6. You can not synchronize on the null object but your volatile variable in Java could be null.  
  
7. From Java 5 writing into a volatile field has the same memory effect as a monitor release, and reading from a volatile field has the same memory effect as a monitor acquire

# Types of References in Java

In Java there are four types of references differentiated on the way by which they are garbage collected.

1. Strong References
2. Weak References
3. Soft References
4. Phantom References

Prerequisite: [Garbage Collection](https://www.geeksforgeeks.org/garbage-collection-java/)

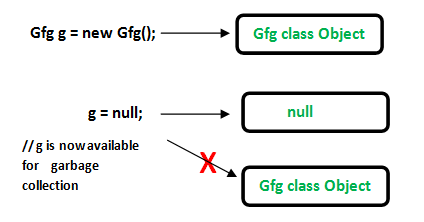
* **Strong References:**This is the default type/class of Reference Object. Any object which has an active strong reference are not eligible for garbage collection. The object is garbage collected only when the variable which was strongly referenced points to null.
* MyClass obj = new MyClass ();

Here ‘obj’ object is strong reference to newly created instance of MyClass, currently obj is active object so can’t be garbage collected.

obj = null;

//'obj' object is no longer referencing to the instance.

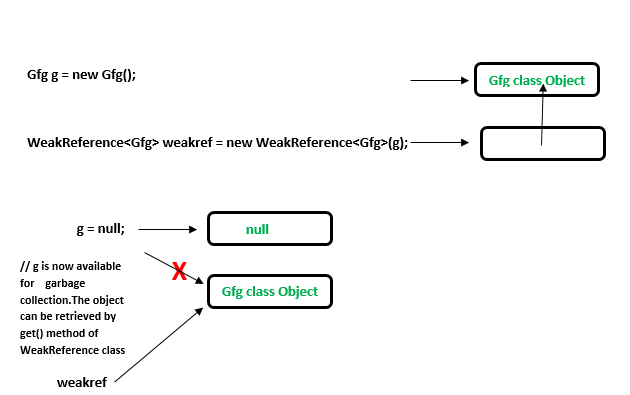
So the 'MyClass type object is now available for garbage collection.



|  |
| --- |
| // Java program to illustrate Strong reference  class Gfg  {      //Code..  }  public class Example  {      public static void main(String[] args)      {           //Strong Reference - by default          Gfg g = new Gfg();            //Now, object to which 'g' was pointing earlier is          //eligible for garbage collection.          g = null;      }  } |

Copy CodeRun on IDE

* **Weak References:**Weak Reference Objects are not the default type/class of Reference Object and they should be explicitly specified while using them.
  + This type of reference is used in WeakHashMap to reference the entry objects .
  + If JVM detects an object with only weak references (i.e. no strong or soft references linked to any object object), this object will be marked for garbage collection.
  + To create such references [java.lang.ref.WeakReference](https://docs.oracle.com/javase/7/docs/api/java/lang/ref/WeakReference.html) class is used.
  + These references are used in real time applications while establishing a DBConnection which might be cleaned up by Garbage Collector when the application using the database gets closed.



|  |
| --- |
| //Java Code to illustrate Weak reference  import java.lang.ref.WeakReference;  class Gfg  {      //code      public void x()      {          System.out.println("GeeksforGeeks");      }  }    public class Example  {      public static void main(String[] args)      {          // Strong Reference          Gfg g = new Gfg();          g.x();            // Creating Weak Reference to Gfg-type object to which 'g'          // is also pointing.          WeakReference<Gfg> weakref = new WeakReference<Gfg>(g);            //Now, Gfg-type object to which 'g' was pointing earlier          //is available for garbage collection.          //But, it will be garbage collected only when JVM needs memory.          g = null;            // You can retrieve back the object which          // has been weakly referenced.          // It succesfully calls the method.          g = weakref.get();            g.x();      }  } |

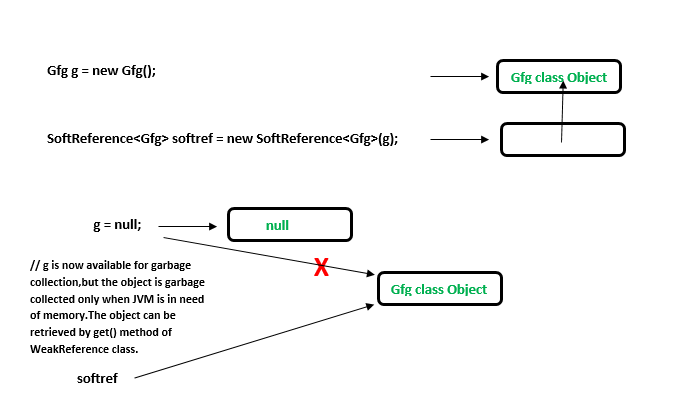
Copy CodeRun on IDE

Output:

GeeksforGeeks

GeeksforGeeks

Two different levels of weakness can be enlisted: Soft and Phantom

* **Soft References:** In Soft reference, even if the object is free for garbage collection then also its not garbage collected, until JVM is in need of memory badly.The objects gets cleared from the memory when JVM runs out of memory.To create such references [java.lang.ref.SoftReference](https://docs.oracle.com/javase/7/docs/api/java/lang/ref/SoftReference.html)class is used.  
  

|  |
| --- |
| //Code to illustrate Soft reference  import java.lang.ref.SoftReference;  class Gfg  {      //code..      public void x()      {          System.out.println("GeeksforGeeks");      }  }    public class Example  {      public static void main(String[] args)      {          // Strong Reference          Gfg g = new Gfg();          g.x();            // Creating Soft Reference to Gfg-type object to which 'g'          // is also pointing.          SoftReference<Gfg> softref = new SoftReference<Gfg>(g);            // Now, Gfg-type object to which 'g' was pointing          // earlier is available for garbage collection.          g = null;            // You can retrieve back the object which          // has been weakly referenced.          // It succesfully calls the method.          g = softref.get();            g.x();      }  } |

* Copy CodeRun on IDE
* Output:
* GeeksforGeeks
* GeeksforGeeks
* **Phantom References:**The objects which are being referenced by phantom references are eligible for garbage collection. But, before removing them from the memory, JVM puts them in a queue called ‘reference queue’ . They are put in a reference queue after calling finalize() method on them.To create such references [java.lang.ref.PhantomReference](https://docs.oracle.com/javase/7/docs/api/java/lang/ref/PhantomReference.html) class is used.

|  |
| --- |
| //Code to illustrate Phantom reference  import java.lang.ref.\*;  class Gfg  {      //code      public void x()      {          System.out.println("GeeksforGeeks");      }  }    public class Example  {      public static void main(String[] args)      {          //Strong Reference          Gfg g = new Gfg();          g.x();            //Creating reference queue          ReferenceQueue<Gfg> refQueue = new ReferenceQueue<Gfg>();            //Creating Phantom Reference to Gfg-type object to which 'g'          //is also pointing.          PhantomReference<Gfg> phantomRef = null;            phantomRef = new PhantomReference<Gfg>(g,refQueue);            //Now, Gfg-type object to which 'g' was pointing          //earlier is available for garbage collection.          //But, this object is kept in 'refQueue' before          //removing it from the memory.          g = null;            //It always returns null.          g = phantomRef.get();            //It shows NullPointerException.          g.x();      }  } |

* Copy CodeRun on IDE
* Runtime Error:
* Exception in thread "main" java.lang.NullPointerException
* at Example.main(Example.java:31)
* Output:
* GeeksforGeeks

## 1) Strong References

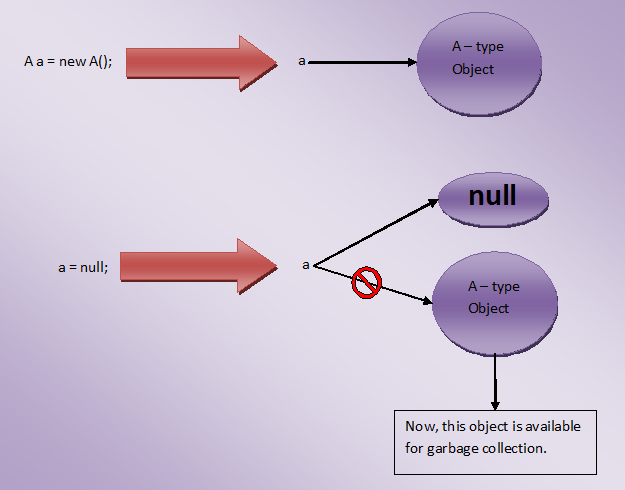
These type of references we use daily while writing the code. Any object in the memory which has active **strong reference** is not eligible for garbage collection. For example, in the below program, reference variable **‘a’** is a strong reference which is pointing to class A-type object. At this point of time, this object can’t be garbage collected as it has strong reference.

1. class A
2. {
3. //Class A
4. }
5. public class MainClass
6. {
7. public static void **main**(**String**[] args)
8. {
9. A a = **new** **A**(); //Strong Reference
10. a = null; //Now, object to which 'a' is pointing earlier is eligible for garbage collection.
11. }
12. }

If you make reference **‘a’** to point to null like in Line 12, then, object to which ‘a’ is pointing earlier will become eligible for garbage collection. Because, it will have no active references pointing to it. This object is most likely to be garbage collected when garbage collector decides to run.

**[READ :  Memory Management In Java - Stack And Heap](https://javaconceptoftheday.com/memory-management-java/" \t "_blank)**

Look at the below picture for more precise understanding.



## 2) Soft References

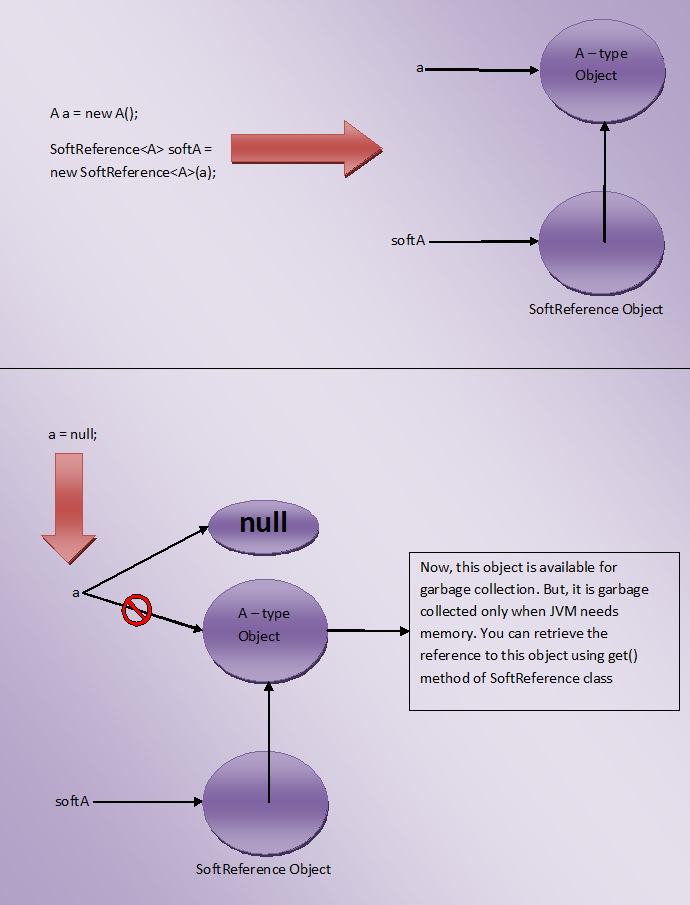
The objects which are softly referenced will not be garbage collected (even though they are available for garbage collection) until JVM badly needs memory. These objects will be cleared from the memory only if JVM runs out of memory. You can create a soft reference to an existing object by using  **java.lang.ref.SoftReference** class. Below is the code example on how to create a soft reference.

1. class A
2. {
3. //A Class
4. }
5. public class MainClass
6. {
7. public static void **main**(**String**[] args)
8. {
9. A a = **new** **A**(); //Strong Reference
10. //Creating Soft Reference to A-type object to which 'a' is also pointing
11. SoftReference<A> softA = **new** SoftReference<A>(a);
12. a = null; //Now, A-type object to which 'a' is pointing earlier is eligible for garbage collection. But, it will be garbage collected only when JVM needs memory.
13. a = softA.**get**(); //You can retrieve back the object which has been softly referenced
14. }
15. }

In the above example, you create two strong references – ‘**a**‘ and ‘**softA**‘. ‘a’ is pointing to A-type object and ‘softA’ is pointing to SoftReference type object. This SoftReference type object is internally referring to A-type object to which ‘a’ is also pointing. When ‘a’ is made to point to null, object to which ‘a’ is pointing earlier becomes eligible for garbage collection. But, it will be garbage collected only when JVM needs memory. Because, it is softly referenced by ‘softA’ object.

**[READ :  Garbage Collection And finalize() method In Java](https://javaconceptoftheday.com/garbage-collection-finalize-method-java/" \t "_blank)**

Look at the below picture for more clarity.



One more use of SoftReference class is that you can retrieve back the object which has been softly referenced. It will be done by using **get()** method. This method returns reference to the object if object is not cleared from the memory. If object is cleared from the memory, it will return null.

## 3) Weak References

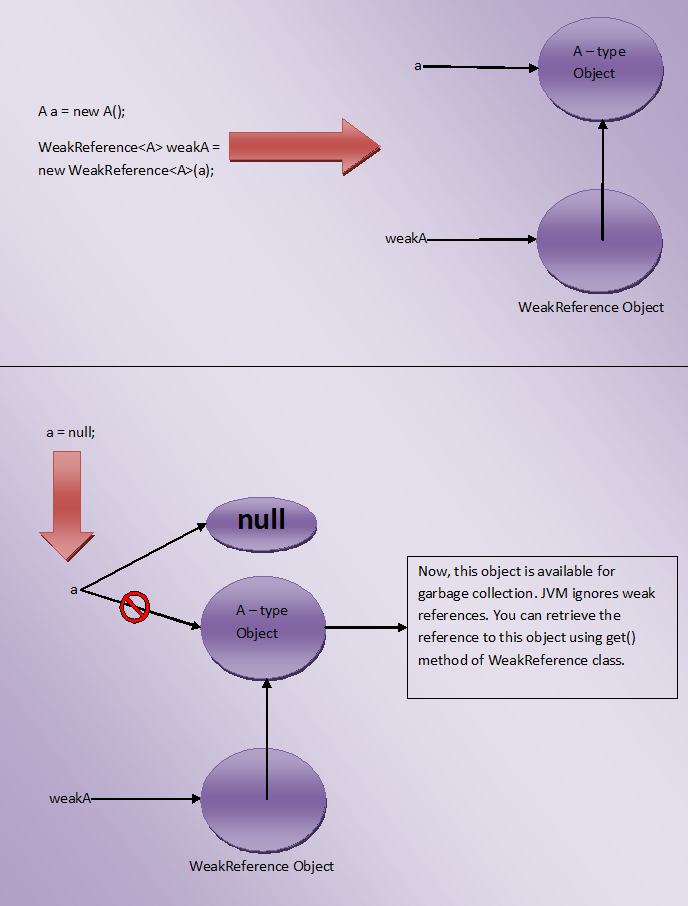
JVM ignores the **weak references**. That means objects which has only week references are eligible for garbage collection. They are likely to be garbage collected when JVM runs garbage collector thread. JVM doesn’t show any regard for weak references.

Below is the code which shows how to create weak references.

1. class A
2. {
3. //A Class
4. }
5. public class MainClass
6. {
7. public static void **main**(**String**[] args)
8. {
9. A a = **new** **A**(); //Strong Reference
10. //Creating Weak Reference to A-type object to which 'a' is also pointing.
11. WeakReference<A> weakA = **new** WeakReference<A>(a);
12. a = null; //Now, A-type object to which 'a' is pointing earlier is available for garbage collection.
13. a = weakA.**get**(); //You can retrieve back the object which has been weakly referenced.
14. }
15. }

Look at the below picture for more clear understanding.

**[READ :  SIB - Static Initialization Block, Static Variables And Static Methods](https://javaconceptoftheday.com/static-members-java/" \t "_blank)**



You may think that what is the use of creating weak references if they are ignored by the JVM, Use of weak reference is that you can retrieve back the weakly referenced object if it is not yet removed from the memory. This is done using get() method of WeakReference class. It will return reference to the object if object is not yet removed from the memory.

## 4) Phantom References

The objects which are being referenced by **phantom references** are eligible for garbage collection. But, before removing them from the memory, JVM puts them in a queue called **‘reference queue’**. They are put in a reference queue after calling finalize() method on them. You can’t retrieve back the objects which are being phantom referenced. That means calling get() method on phantom reference always returns null.

Below example shows how to create Phantom References.

1. class A
2. {
3. //A Class
4. }
5. public class MainClass
6. {
7. public static void **main**(**String**[] args)
8. {
9. A a = **new** **A**(); //Strong Reference
10. //Creating ReferenceQueue
11. ReferenceQueue<A> refQueue = **new** ReferenceQueue<A>();
12. //Creating Phantom Reference to A-type object to which 'a' is also pointing
13. PhantomReference<A> phantomA = **new** PhantomReference<A>(a, refQueue);
14. a = null; //Now, A-type object to which 'a' is pointing earlier is available for garbage collection. But, this object is kept in 'refQueue' before removing it from the memory.
15. a = phantomA.**get**(); //it always returns null
16. }
17. }

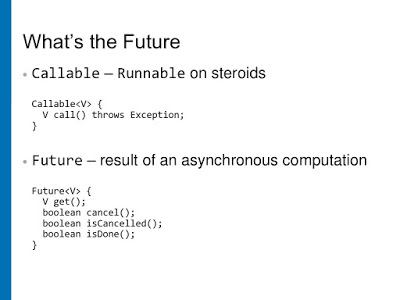
# [Difference between Callable and Runnable in Java - Interview question](http://www.java67.com/2013/01/difference-between-callable-and-runnable-java.html)

Difference between Callable and Runnable interface in Java is one of the interesting questions from my list of [Top 15 Java multi-threading questions](http://javarevisited.blogspot.it/2011/07/java-multi-threading-interview.html), and it’s also very popular in various Java Interviews. The Callable interface is newer than Runnable interface and added on Java 5 release along with other major changes e.g. [Generics](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html), [Enum](http://java67.blogspot.sg/2012/11/java-enum-example-with-constructor.html), [Static imports](http://javarevisited.blogspot.sg/2012/10/what-is-static-import-in-java-5-example-tutorial.html) and [variable argument method](http://javarevisited.blogspot.sg/2011/09/variable-argument-in-java5-varargs.html). Though both Callable and Runnable interface are designed to represent a task, which can be executed by any thread, there is some significant difference between them. In my opinion, the major difference between Callable and Runnable interface is that Callable can return the result of an operation performed inside call() method, which was one of the limitations with Runnable interface.

Callable interface was added in Java 5 to complement existing Runnable interface, which is used to wrap a task and pass it to a Thread or thread pool for asynchronous execution. Callable actually represent an asynchronous computation, whose value is available via Future object. All the code which needs to be executed asynchronously goes into call() method. Callable is also a single abstract method type (SAM type), so it can be used along with [lambda expression on Java 8](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html). Both Callable and Future are parametric type and can be used to wrap classes like Integer, String or anything else. When you pass a Callable to thread pool, it choose one thread and execute the Callable. It immediately return a Future object which promises to hold result of computation once done. You can then call get() method of Future, which will return result of computation or block if Computation is not complete. If you don't like indefinite blocking then you can also use overloaded get() method with timeout. Future also allows you to cancel the task if its not started or interrupt if its started. We will see, how we can calculate factorial of large number using Callable and Future in Java. BTW, if you are serious about mastering concurrency API of Java, I suggest you to also take a look at one of the best book on the subject, [Java Concurrency in Practice](http://www.amazon.com/dp/0321349601/?tag=javamysqlanta-20) by Brian Goetz. It is one of the book I keep refer whenever I have a doubt or want to refresh my knowledge.

### Callable vs Runnable

Many of you would be familiar with Runnable interface, one of the most popular way to use thread in Java, but you would be happy to know that Runnable is not the only way to create a task which can be executed by parallel threads. You can also use Callable interface to do the same. Main *difference between Runnable and Callable* is that Runnable cannot return any value back to caller but Callable can return value. Another difference is that call() method from Callable can also throw checked exception which was not possible by run() method of Runnable interface. See [here](http://java67.blogspot.sg/2013/01/difference-between-callable-and-runnable-java.html) to learn more about difference between Runnable and Callable in Java.

[](https://4.bp.blogspot.com/-EGpKPBfDJug/VYGQJR5jmiI/AAAAAAAADFs/oq5q2BgVtms/s1600/Callable+and+Future+Java.jpg)

## Callable and Future Example in Java

Here is our complete Java program to demonstrate *how you can use Callable and Future together* to implement asynchronous processing  in Java. Once you started using thread pool, Callable and Future, you don't need to wait for task to be completed, you can move on with other task and comeback to check whether task is completed or not. If task is finished then just get the result by calling get() method, but remember its a [blocking call](http://javarevisited.blogspot.sg/2012/02/what-is-blocking-methods-in-java-and.html), so it will block if task is not finished.

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

/\*\*

\* Simple Java program to demonstrate how to use Callable and Future class in

\* Java. You can use FutureTask for asynchronous processing.

\*

\* @author WINDOWS 8

\*

\*/

public class HelloWorldApp {

public static void main(String... args) throws InterruptedException, ExecutionException {

// creating thread pool to execute task which implements Callable

ExecutorService es = Executors.newSingleThreadExecutor();

System.out.println("submitted callable task to calculate factorial of 10");

Future result10 = es.submit(new FactorialCalculator(10));

System.out.println("submitted callable task to calculate factorial of 15");

Future result15 = es.submit(new FactorialCalculator(15));

System.out.println("submitted callable task to calculate factorial of 20");

Future result20 = es.submit(new FactorialCalculator(20));

System.out.println("Calling get method of Future to fetch result of factorial 10");

long factorialof10 = result10.get();

System.out.println("factorial of 10 is : " + factorialof10);

System.out.println("Calling get method of Future to get result of factorial 15");

long factorialof15 = result15.get();

System.out.println("factorial of 15 is : " + factorialof15);

System.out.println("Calling get method of Future to get result of factorial 20");

long factorialof20 = result20.get();

System.out.println("factorial of 20 is : " + factorialof20);

}

}

class FactorialCalculator implements *Callable<Long>* {

private int number;

public FactorialCalculator(int number){

this.number = number;

}

@Override

public Long call() throws Exception {

return factorial(number);

}

private long factorial(int n) throws InterruptedException {

long result = 1;

while (n != 0) {

result = n \* result;

n = n - 1;

Thread.sleep(100);

}

return result;

}

}

Output

submitted callable task to calculate factorial of 10

submitted callable task to calculate factorial of 15

submitted callable task to calculate factorial of 20

Calling get method of Future to fetch result of factorial 10

factorial of 10 is : 3628800

Calling get method of Future to get result of factorial 15

factorial of 15 is : 1307674368000

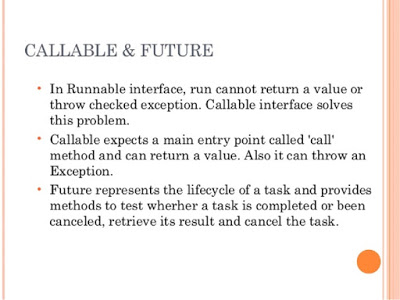
Calling get method of Future to get result of factorial 20

factorial of 20 is : 2432902008176640000

When you run this program, you will see that first 4 lines will be printed immediately because submit() is a non blocking method, it just takes a task and [returns a Future object](http://javarevisited.blogspot.in/2015/01/how-to-use-future-and-futuretask-in-Java.html), it doesn't wait until task is completed. That's why you see that all three tasks to calculate factorials are submitted immediately, but they are not done yet. When our code calls get() on first Future object its blocked until the calculation is done, that's why you will see the fifth line printing after sometime. For next two lines also same story because when you call get() method it will block until result is available. BTW, you don't need to block, you can even use isDone() method to check if calculation is completed or not before calling get method.

### Important points about Callable and Future

1) Callable is a SAM type interface, so it can be used in lambda expression.  
  
2) Callable has just one method call() which holds all the code needs to executed asynchronously.  
  
3) In Runnable interface, there was no way to return the result of computation or throw checked exception but with Callable you can both return a value and can throw [checked exception](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html).  
  
4) You can use get() method of Future to retrieve result once computation is done. You can check if computation is finished or not by using isDone() method.  
  
5) You can cancel the computation by using Future.cancel() method.  
  
6) get() is a blocking call and it blocks until computation is completed.

[](https://4.bp.blogspot.com/-3iDp1Cf_8vA/VYGQKU_h5JI/AAAAAAAADF0/CjHRI0uNPAY/s1600/Callable+and+Future+Example+in+Java.jpg)

That's all about **how to use Callable and Future object in Java**. You can wrap asynchronous computation inside call() method and pass it to a single thread or thread pool for execution. you don't need to wait until execution complete, your thread can carry on with future object returned by call method. Once computation is done you can query the future object and get the result back.

# [10 AWT Swing Interview Questions Answers in Java](http://www.java67.com/2013/01/10-awt-swing-interview-questions-answers-java.html)

AWT and Swing Interview Questions are part of any Java interviews which involves GUI development work. Since AWT(Abstract Windows Toolkit) and Swing offers most popular Java GUI solutions, Yes, JavaFX  has still lot of path to cover in terms of popularity and usability of Swing. In Investment banking, where Java rules on server side applicationdevelopment, Swing was heavily used in GUI development couple of years back. Now trends is shifting towards C# for GUI development due to couple of reasons e.g. good Swing developers are not easy to be found as compared to C# developers. By the way there are still many development work is going on Swing. One of the reason Swing developers are high in demand is because Swing is quickly becoming a niche technology. This is surprising because  Swing is part of Java, but given steep learning curve of Swing to master different layouts and components e.g. JTable or [JList](http://javarevisited.blogspot.sg/2011/08/java-swing-tutorial-jlist-example.html) , I tend to believe it. In this article we will see some good Swing Interview questions for practice. Questions like [InvokeLater vs InvokeAndWait are](http://javarevisited.blogspot.sg/2011/09/invokeandwait-invokelater-swing-example.html)classic, which is always worth preparing.

[Read more »](http://www.java67.com/2013/01/10-awt-swing-interview-questions-answers-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[1 comment:](http://www.java67.com/2013/01/10-awt-swing-interview-questions-answers-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=6921595902857291933)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=6921595902857291933)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6921595902857291933&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6921595902857291933&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6921595902857291933&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6921595902857291933&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=6921595902857291933&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [swing](http://www.java67.com/search/label/swing?&max-results=3)

# [How Constructor Chaining works in Java - Example](http://www.java67.com/2012/12/how-constructor-chaining-works-in-java.html)

How to call one constructor from another constructor in Java or What is Constructor Chaining in Java is one of the [tricky questions in Java interviews](http://java67.blogspot.sg/2012/09/top-10-tricky-java-interview-questions-answers.html). Well, you can use [this keyword](http://javarevisited.blogspot.sg/2012/01/this-keyword-java-example-tutorial.html) to call one constructor from another constructor of the same class if you want to call a constructor from based class or super class then you can use super keyword. Calling one constructor from other is called **Constructor chaining in Java**. Constructors can call each other automatically or explicitly using this() and super() keywords. this() denotes a [no-argument constructor](http://javarevisited.blogspot.sg/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) of the same class and super() denotes a no argument or default constructor of parent class. Also having multiple constructors in the same class is known as [constructor overloading in Java](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html).

[Read more »](http://www.java67.com/2012/12/how-constructor-chaining-works-in-java.html#more)

Read more: <http://www.java67.com/search/label/core%20java%20interview%20question%20answer?updated-max=2013-02-20T04:57:00-08:00&max-results=6#ixzz4i1YNsgdj>

# [Producer Consumer Problem with Wait and Notify Example](http://www.java67.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html)

Producer Consumer Problem is a classical concurrency problem and in fact it is one of the concurrency design pattern. In last article we have seen solving [Producer Consumer problem in Java using blocking Queue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html) but one of my reader emailed me and requested code example and explanation of solving Producer Consumer problem in Java  with [wait and notify method](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html)as well, Since its often asked as one of the top [coding question in Java](http://java67.blogspot.sg/2012/08/10-java-coding-interview-questions-and.html). In this Java tutorial, I have put the code example of wait notify version of earlier producer consumer concurrency design pattern. You can see this is much longer code with explicit handling blocking conditions like when shared queue is full and when queue is empty. Since we have replaced[BlockingQueue](http://javarevisited.blogspot.sg/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html) with Vector we need to implement blocking using [wait and notify](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html) and that's why we have introduced produce(int i) and consume() method. If you see I have kept consumer thread little slow by allowing it to sleep for 50 Milli second to give an opportunity to producer to fill the queue, which helps to understand that Producer thread is also waiting when Queue is full.

[Read more »](http://www.java67.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[21 comments:](http://www.java67.com/2012/12/producer-consumer-problem-with-wait-and-notify-example.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=7312900353389350784)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=7312900353389350784)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7312900353389350784&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7312900353389350784&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7312900353389350784&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7312900353389350784&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=7312900353389350784&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [java concurrency tutorial](http://www.java67.com/search/label/java%20concurrency%20tutorial?&max-results=3), [thread](http://www.java67.com/search/label/thread?&max-results=3)

# [When to use ArrayList vs LinkedList in Java](http://www.java67.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html)

ArrayList and LinkedList are two popular concrete implementations of List interface from Java's popular Collection framework. Being List implementation both ArrayList and LinkedList are ordered, the index based and allows duplicate. Despite being from same type hierarchy there are a lot of differences between these two classes which makes them popular among Java interviewers. The main difference between ArrayList vs LinkedList is that former is backed by an array while later is based upon linked list data structure, which makes the performance of add(), remove(), contains() and iterator() different for both ArrayList and LinkedList. The difference between ArrayList and LinkedList is also an important  Java collection interview questions, as much popular as [Vector vs ArrayList](http://java67.blogspot.sg/2012/09/arraylist-vs-vector-in-java-interview.html) or [HashMap vs HashSet in Java](http://java67.blogspot.sg/2012/08/difference-between-hashset-and-hashmap.html). Sometimes this is also asked as **When to use LinkedList** and **When to use ArrayList in Java**. In this Java collection tutorial, we will compareLinkedList vs ArrayList on the various parameter which will help us to decide [when to use ArrayList over LinkedList in Java](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html).

[Read more »](http://www.java67.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[12 comments:](http://www.java67.com/2012/12/difference-between-arraylist-vs-LinkedList-java.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4322207026608371273)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4322207026608371273)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4322207026608371273&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4322207026608371273&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4322207026608371273&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4322207026608371273&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4322207026608371273&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [Java collection tutorial](http://www.java67.com/search/label/Java%20collection%20tutorial?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3)

# [JDBC Interview questions answers in Java - 2 to 4 years experienced programmer](http://www.java67.com/2012/12/jdbc-interview-questions-answers-in-Java-2-4-years-experienced.html)

JDBC Interview question forms one of the important section in Java Interviews. Similar to [multithreading](http://java67.blogspot.sg/2012/08/5-thread-interview-questions-answers-in.html), [Collection framework](http://javarevisited.blogspot.sg/2011/11/collection-interview-questions-answers.html) and [Garbage collection interview question](http://javarevisited.blogspot.sg/2012/10/10-garbage-collection-interview-question-answer.html), JDBC question must be prepared by any Java programmer. Most of questions from JDBC or Java database connectivity comes from API and basic architecture of JDBC which also involves JDBC drivers. A good understanding of JDBC API along with database basics like [transactions](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html) also help to do well in JDBC interviews. I have collected some of *frequently asked JDBC Interview question* for quick reference. This will help to revise some important JDBC concepts and also give a chance to explore JDBC API to newcomers. If you have any other JDBC interview question, which has been asked to you or friends, and you think it’s good to share among Java community then please share with us. Let's see my 11 questions from JDBC, not so tough but worth preparing.

[Read more »](http://www.java67.com/2012/12/jdbc-interview-questions-answers-in-Java-2-4-years-experienced.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[3 comments:](http://www.java67.com/2012/12/jdbc-interview-questions-answers-in-Java-2-4-years-experienced.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4776056681638330066)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=4776056681638330066)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4776056681638330066&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4776056681638330066&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4776056681638330066&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4776056681638330066&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=4776056681638330066&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [JDBC](http://www.java67.com/search/label/JDBC?&max-results=3), [sql](http://www.java67.com/search/label/sql?&max-results=3), [SQL interview Question](http://www.java67.com/search/label/SQL%20interview%20Question?&max-results=3)

# [Difference between RuntimeException and checked Exception in Java](http://www.java67.com/2012/12/difference-between-runtimeexception-and-checked-exception.html)

**RuntimeException vs Checked Exception in Java**

Java Exceptions are divided in two categories RuntimeException also known as [unchecked Exception](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html) and checked Exception. *Main difference between RuntimeException and checked Exception* is that, It is mandatory to provide try catch or [try finally block](http://javarevisited.blogspot.sg/2012/11/difference-between-final-finally-and-finalize-java.html) to handle checked Exception and failure to do so will result in compile time error, while in case ofRuntimeException this is not mandatory. Difference between checked and unchecked exception is one of the most popular question on Java [interview for 2 to years experienced](http://java67.blogspot.sg/2012/10/java-interview-questions-for-2-to-3-4-years-experienced.html) developer especially related to Exception concepts. Answer to this question is rather similar as mentioned in previous lines and they are mostly asked along with other Java Exception interview questions like [difference between throw and throws](http://java67.blogspot.sg/2012/10/difference-between-throw-vs-throws-in.html) and [Error vs Exception](http://java67.blogspot.sg/2012/12/difference-between-error-vs-exception.html). Any Exception which is subclass of RuntimeException are called unchecked and mandatory exception handling is not requirement for them.

[Read more »](http://www.java67.com/2012/12/difference-between-runtimeexception-and-checked-exception.html#more)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[1 comment:](http://www.java67.com/2012/12/difference-between-runtimeexception-and-checked-exception.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3118299439057674677)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=3118299439057674677)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3118299439057674677&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3118299439057674677&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3118299439057674677&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3118299439057674677&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=3118299439057674677&target=pinterest)

Labels: [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [exception](http://www.java67.com/search/label/exception?&max-results=3)

# [How to reverse String in Java with or without StringBuffer Example](http://www.java67.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.html)

**Reverse String in Java**

There are many ways to reverse String in Java. You can use rich Java API to quickly reverse contents of any String object. Java library provides [StringBuffer and StringBuilder](http://javarevisited.blogspot.ca/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) class with reverse() method which can be used to reverse String in Java. Since converting between String and StringBuffer or StringBuilder is very easy it's the most easy wayavailable to reverse String in Java. At the same time Writing Java program to reverse String in Java without StringBuffer is one of the popular [Java String interview question](http://javarevisited.blogspot.sg/2012/10/10-java-string-interview-question-answers-top.html), which requires you to reverse String by applying logic and by not using API methods. Since reverse is a recursive job, you can use recursion as well as loop to reverse String in Java. In this Java tutorial I have shown **How to reverse String using StringBuffer, StringBuilder** and using **pure loop with logic**. You can also check [How to reverse String with recursion in Java](http://javarevisited.blogspot.ca/2012/01/how-to-reverse-string-in-java-using.html), if you want to see recursive code. let's see complete Java program for this beautiful Java programming exercise.

## Java program to reverse String in Java

[Java program to reverse String in Java without StringBuffer or StringBuilder](http://3.bp.blogspot.com/-1lzFJzIgaHk/UF2Ci6kY5pI/AAAAAAAAAes/OYiM7r-DHzc/s1600/17.jpg)Here is my complete code program to reverse any String in Java. In [main method](http://java67.blogspot.sg/2012/08/what-is-main-method-in-java-why-main-is.html) we have first used StringBuffer and StringBuilder to *reverse contents of String* and then we wrote our own logic to reverse String. This uses toCharArray() method of String class which return [character array of String](http://javarevisited.blogspot.ca/2012/03/why-character-array-is-better-than.html). By looping through character array and appending it into empty String we can get reversed String in Java, as shown in following example.

/\*\*  
 \*  
 \* **Java program to reverse String in Java**. There are multiple ways to reverse  
 \* String in Java, you can either take help of standard Java API StringBuffer  
 \* to reverse String in Java. StringBuffer has a reverse() method which return StringBuffer  
 \* with reversed contents. On the other hand you can also reverse it by applying your  
 \* own logic, if asked to reverse String without using StringBuffer in Java. By the way  
 \* you can also use StringBuilder to reverse String in Java. StringBuilder is non thread-safe  
 \* version of StringBuffer and provides similar API. You can use StringBuilder's reverse()  
 \* method to reverse content and then convert it back to String  
 \*  
 \* @author http://java67.blogspot.com  
 \*/  
**public** **class** StringReverseExample {  
    
    
    **public** **static** **void** main(String args[]) {  
        
        *//quick wasy to reverse String in Java - Use StringBuffer*  
        String word = "HelloWorld";  
        String reverse = **new** StringBuffer(word).reverse().toString();  
        System.out.printf(" original String : %s , reversed String %s  %n", word, reverse);  
        
        *//another quick to reverse String in Java - use StringBuilder*  
        word = "WakeUp";  
        reverse = **new** StringBuilder(word).reverse().toString();  
        System.out.printf(" original String : %s , reversed String %s %n", word, reverse);  
        
        *//one way to reverse String without using StringBuffer or StringBuilder is writing*

*//own utility method*  
        word = "Band";  
        reverse = reverse(word);  
        System.out.printf(" original String : %s , reversed String %s %n", word, reverse);  
    }     
    
    
    **public** **static** String reverse(String source){  
        **if**(source == **null** || source.isEmpty()){  
            **return** source;  
        }         
        String reverse = "";  
        **for**(**int** i = source.length() -1; i>=0; i--){  
            reverse = reverse + source.charAt(i);  
        }  
        
        **return** reverse;  
    }  
      
}  
  
**Output:**  
original String : HelloWorld , reversed String dlroWolleH   
original String : WakeUp , reversed String pUekaW  
original String : Band , reversed String dnaB

That's all on **How to reverse String in Java with and without StringBuffer and StringBuilder**. Though being a Java programmer I prefer to use library and suggest any one to use StringBuffer or StringBuilder to reverse String for any production use. Though its also a [good programming exercise](http://javarevisited.blogspot.ca/2011/06/top-programming-interview-questions.html) and you should practice it before going for any Java programming interview.

Other **Java tutorials** you may like

[String matches examples in Java](http://java67.blogspot.sg/2012/09/java-string-matches-example-regular-expression.html)

[Difference between String and StringBuffer in Java](http://java67.blogspot.sg/2012/08/difference-between-string-and-stringbuffer-in-java.html)

[Best way to convert numbers to String in Java](http://java67.blogspot.sg/2012/10/best-way-to-convert-numbers-to-string-in-java-example.html)

[Difference between == and equals method in Java](http://java67.blogspot.sg/2012/11/difference-between-operator-and-equals-method-in.html)

[How to create Enum from String in Java](http://java67.blogspot.sg/2012/08/how-to-create-enum-from-string-in-java.html)

[How to use contains and indexOf method in String Java](http://java67.blogspot.sg/2012/10/string-contains-and-indexof-example.html)

Posted by [Javin Paul](https://plus.google.com/114528699166048052030)[43 comments:](http://www.java67.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.html#comment-form)[[http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5120298873160712092)](https://www.blogger.com/email-post.g?blogID=694855878384792308&postID=5120298873160712092)

[Email This](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5120298873160712092&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5120298873160712092&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5120298873160712092&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5120298873160712092&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=694855878384792308&postID=5120298873160712092&target=pinterest)

Labels: [coding](http://www.java67.com/search/label/coding?&max-results=3), [Coding Problems](http://www.java67.com/search/label/Coding%20Problems?&max-results=3), [core java](http://www.java67.com/search/label/core%20java?&max-results=3), [core java interview question answer](http://www.java67.com/search/label/core%20java%20interview%20question%20answer?&max-results=3), [programming](http://www.java67.com/search/label/programming?&max-results=3)

# [Difference between Error vs Exception in Java - Interview question](http://www.java67.com/2012/12/difference-between-error-vs-exception.html)

Both Error and Exception are derived from java.lang.Throwable in Java but main difference between Error and Exception is kind of error they represent. java.lang.Error represent errors which are generally can not be handled and usually refer catastrophic failure e.g. running out of System resources, some examples of Error in Java are[java.lang.OutOfMemoryError](http://javarevisited.blogspot.sg/2011/09/javalangoutofmemoryerror-permgen-space.html) or [Java.lang.NoClassDefFoundError](http://java67.blogspot.sg/2012/08/what-is-noclassdeffounderror-in-java.html) and [java.lang.UnSupportedClassVersionError](http://java67.blogspot.sg/2012/10/how-to-fix-javalangunsupportedclassversionerror-major-minor-version-49-50-51.html). On the other hand java.lang.Exception represent errors which can be catch and dealt e.g. IOException which comes while performing I/O operations i.e. [reading files and directories](http://javarevisited.blogspot.ca/2011/12/read-and-write-text-file-java.html). Clear understanding of Error and Exception is must for any serious Java programmer and good programming and debugging skills are required to overcome issues which caused Error and Exception in Java. Apart from its must have knowledge in Java application development, **difference between Error and Exception** is also a popular questions on Java interviews related to Exception handling, similar to [difference between throw and throws in Java](http://java67.blogspot.sg/2012/10/difference-between-throw-vs-throws-in.html). In this Java article we will briefly see major difference between Error and Exception in Java which include both syntactical and logical difference.

Important Questions and Anseres

**Question 1: What’s wrong using HashMap in the multi-threaded environment? When does the get() method go to an infinite loop?** ([answer](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html))

Well, nothing is wrong, depending on how you use it. For example, if you initialize the HashMap just by one thread and then all threads are only reading from it, then it’s perfectly fine. One example of this is a Map which contains configuration properties.

The real problem starts when at-least one of that thread is updating HashMap i.e. adding, changing or removing any key value pair. Since put() operation can cause re-sizing and which can further lead to infinite loop, that’s why either you should use [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) or [ConcurrentHashMap](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html), later is better.

**Question 2. Does overriding the hashCode() method have any performance implication?** ([answer](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html))

This is a good question and open to all, as per my knowledge a poor hash code function will result in the frequent collision in HashMap which eventually increases the time for adding an object into Hash Map.

From Java 8 onwards though, collision will not impact performance as much as it does in earlier versions, because after a threshold the linked list will be replaced by the binary tree, which will give you O(logN) performance in the worst case, as compared to O(n) of linked list.

**Question 3: Do all properties of an Immutable Object need to be final?** ([answer](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))

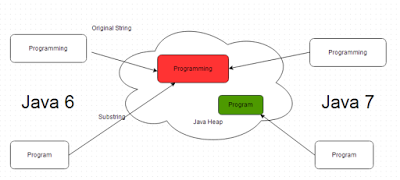
Not necessarily, as stated above you can achieve same functionality by making the member non-final but private and not modifying them except in a constructor. Don’t provide setter methods for them and if it is a mutable object, then don’t ever leak any reference for that member.

Remember making a reference variable final, only ensures that it will not be reassigned to a different value, but you can still change individual properties of object, pointed by that reference variable. This is one of the key point, Interviewer like to hear from candidates.

**Question 4: How does the substring() method inside String works?** ([answer](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html))

Another good Java interview question, I think the answer is not sufficient, but here it is “Substring creates a new object out of source string by taking a portion of original string”.

This question was mainly asked to see if the developer is familiar with the risk of memory leaks, which a sub-string can create. Until Java 1.7, substring holds the reference of the original character array, which means even a sub-string of 5 character long, can prevent 1GB character array from garbage collection, by holding a strong reference.

[](http://www.javacodegeeks.com/wp-content/uploads/2015/11/How-SubString-works-in-Java.png)

This issue is fixed in Java 1.7, where the original character array is not referenced anymore, but that change also made the creation of substring a bit more costly in terms of time. Earlier it was on the range of O(1), which could be O(n) in worst case on Java 7.

**Question**\*\*  5: Can you write a critical section code for the singleton?\*\* ([answer](http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html))

This core Java question is a followup of the previous question and expecting the candidate to write Java singleton using double checked locking. Remember to use the volatile variable to make Singleton thread-safe.

**Question 6: How do you handle error condition while writing stored procedure or accessing stored procedure from java?** ([answer](http://javarevisited.blogspot.com/2013/04/spring-framework-tutorial-call-stored-procedures-from-java.html))

This is one of the tough Java interview questions and its open for all, my friend didn’t know the answer so he didn’t mind telling me. My take is that stored procedure should return an error code if some operation fails but if stored procedure itself fails than catching SQLException is the only choice.

**Question 7 : What is difference between Executor.submit() and Executer.execute() methods ?** ([answer](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html))

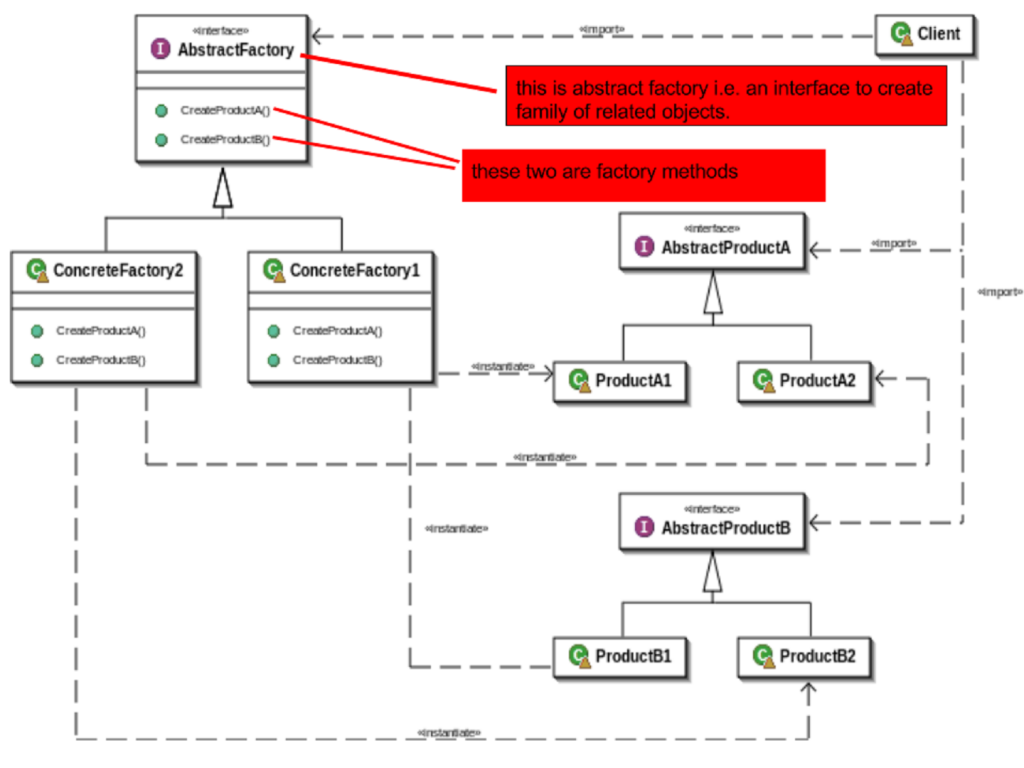
This question is from my list of Top 15 Java multi-threading question answers. It’s getting popular day by day because of huge demand of Java developers with good concurrency skills. The answer is that former returns an object of Future which can be used to find result from worker thread.

There is a difference when looking at exception handling. If your tasks throw an exception and if it was submitted with executing this exception will go to the uncaught exception handler (when you don’t have provided one explicitly, the default one will just print the stack trace to System.err).

If you submitted the task with submit any thrown exception, [checked exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) or not, is then part of the task’s return status. For a task that was submitted with submitting and that terminates with an exception, the Future.get() will re-throw this exception, wrapped in an ExecutionException.

**Question 8:  What is the difference between factory and abstract factory pattern?**([answer](http://javarevisited.blogspot.sg/2013/01/difference-between-factory-and-abstract-factory-design-pattern-java.html))

Abstract Factory provides one more level of abstraction. Consider different factories each extended from an Abstract Factory and responsible for the creation of different hierarchies of objects based on the type of factory. E.g. AbstractFactory extended by AutomobileFactory, UserFactory, RoleFactory etc. Each individual factory would be responsible for the creation of objects in that genre. Here is UML diagram of factory and abstract factory pattern:

[](http://www.javacodegeeks.com/wp-content/uploads/2015/11/factory-vs-abstract-factory-pattern.png)

**Question 9: What is a Singleton? Is it better to make the whole method synchronized or only critical section synchronized?** ([answer](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))

Singleton in Java is a class with just one instance in the whole Java application, for example, java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced Enum its very easy.

**Question 10:  Can you write code for iterating over HashMap in Java 4 and Java 5?**([answer](http://java67.blogspot.com/2014/05/3-examples-to-loop-map-in-java-foreach.html))

Tricky one but he managed to write using while and a for loop. Actually there are four ways to iterate over any Map in Java, one involves using keySet() and iterating over key and then using get() method to retrieve values, which is bit expensive.

Second method involves using entrySet() and iterating over them either by using for each loop or while with Iterator.hasNext() method. This one is a better approach because both key and value objects are available to you during Iteration and you don’t need to call the get() method for retrieving value, which could give O(n) performance in case of huge linked list at one bucket. See my post [4 ways to iterate over Map in Java](http://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html) for detailed explanation and code examples.

**Question 11 : When do you override hashCode() and equals()?** ([answer](http://javarevisited.blogspot.com/2013/08/10-equals-and-hashcode-interview.html))

Whenever necessary, especially if you want to do equality check based upon business logic rather than object equality, e.g. two employee objects are equal if they have the same emp\_id, despite the fact that they are two different objects, created by different part of the code.

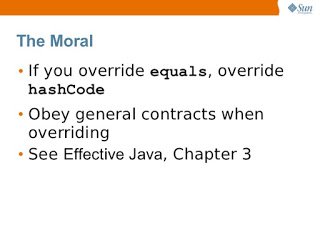
Also overriding both these methods are must if you want to use them as key in HashMap. Now as part of the equals-hashcode contract in Java, when you override equals, you must override hashcode as well, otherwise your object will not break invariant of classes e.g. Set, Map which relies on equals() method for functioning properly.

You can also check my post [5 tips on equals in Java](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html) to understand subtle issue which can arise while dealing with these two methods.

**Question 12 :. What will be the problem if you don’t override hashCode() method ?**([answer](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html))

If you don’t override the equals method, then the contract between equals and hashcode will not work, according to which, two objects which are equal by equals() must have the same hashcode. In this case, an other object may return different hashCode and will be stored on that location, which breaks invariant of HashMap class, because they are not supposed to allow duplicate keys.

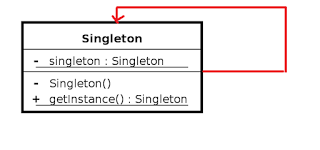
When you add object using put() method, it iterate through all Map.Entry object present in that bucket location, and update value of previous mapping, if Map already contains that key. This will not work if hashcode is not overridden.

[](http://www.javacodegeeks.com/wp-content/uploads/2015/11/java-puzzle-equal-hashcode.jpg)

**Question 13 : Is it better to synchronize critical sections of getInstance() method or the whole getInstance() method?** ([answer](http://javarevisited.blogspot.com/2014/05/double-checked-locking-on-singleton-in-java.html))

The answer is only the critical section, because if we lock the whole method, then every time some some one call this method, it will have to wait even though we are not creating any object. In other words, synchronization is only needed, when you create object, which happens only once.

Once object has created, there is no need for any synchronization. In fact, that’s very poor coding in terms of performance, as synchronized method reduce performance up to 10 to 20 times. Here is UML diagram of Singleton pattern:

[](http://www.javacodegeeks.com/wp-content/uploads/2015/11/Singleton-design-Pattern-Java.png)

By the way, there are several ways to create a thread-safe singleton in Java, which you can also mention as part of this question or any follow-up.

**Question 14: Where does equals() and hashCode() method comes in the picture during the get() operation?** ([answer](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html))

This core Java interview question is a follow-up of previous Java question and the candidate should know that once you mention hashCode, people are most likely ask, how they are used in HashMap. When you provide a key object, first it’s hashcode method is called to calculate bucket location. Since a bucket may contain more than one entry as linked list, each of those Map.Entry object is evaluated by using equals() method to see if they contain the actual key object or not.

**Questions 15: How do you avoid a deadlock in Java?** ([answer](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html))

You can avoid deadlock by breaking the circular wait condition. In order to do that, you can make an arrangement in the code to impose the ordering on acquisition and release of locks.

If lock will be acquired in a consistent order and released in just opposite order, there would not be a situation where one thread is holding a lock which is acquired by other and vice-versa. See the detailed answer for the code example and more detailed explanation.

**Question 16:  What is the difference between creating String as new() and literal?**([answer](http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html))

When we create the string with new() Operator, it’s created in heap and not added into string pool while String created using literal are created in String pool itself which exists in PermGen area of heap.

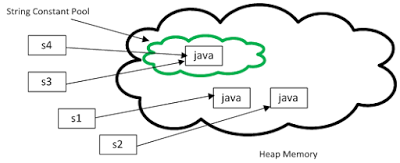
**String** str = **new** **String**("Test")

does not put the object str in String pool, we need to call String.intern() method which is used to put them into String pool explicitly. It’s only when you create String object as a String literal e.g. String s = "Test" that Java automatically puts that into the String pool.

By the way there is a catch here Since we are passing arguments as “Test”, which is a String literal, it will also create another object as “Test” on string pool. This is the one point, which has gone unnoticed until knowledgeable readers of Javarevisited blog suggested it.

To learn more about the difference between String literal and String object, see [this](http://java67.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html)article.

Here is a nice image which shows this difference quite well:

[](http://www.javacodegeeks.com/wp-content/uploads/2015/11/String-literal-vs-String-Object-in-Java.png)

**Question 17: What is an Immutable Object? Can you write an Immutable Class?** ( [answer](http://javarevisited.blogspot.in/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))

Immutable classes are Java classes whose objects can not be modified once created. Any modification in Immutable object results in the new object. For example, [String is immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html).

Mostly Immutable classes are also final in Java, in order to prevent sub classes from overriding methods, which can compromise Immutability. You can achieve the same functionality by making member as non-final but private and not modifying them except in constructor.

Apart form obvious, you also need to make sure that, you should not expose the internals of Immutable object, especially if it contains a mutable member. Similarly, when you accept the value for the mutable member from client e.g. java.util.Date, use [clone() method](http://javarevisited.blogspot.sg/2013/09/how-clone-method-works-in-java.html) keep a separate copy for yourself, to prevent the risk of malicious client modifying mutable reference after setting it.

The Same precaution needs to be taken while returning value for a mutable member, return another separate copy to the client, never return original reference held by Immutable class. You can see my post How to create an Immutable class in Java for step by step guide and code examples.

**Question 18: Give the simplest way to find out the time a method takes for execution without using any profiling tool?** ([answer](http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html))

Read the system time just before the method is invoked and immediately after thr method returns. Take the time difference, which will give you the time taken by a method for execution.

To put it in code…

long start = System.currentTimeMillis ();

**method** ();

long **end** = System.currentTimeMillis (); System.out.println (“Time taken **for** execution **is** ” + (**end** – start));

Remember that if the time taken for execution is too small, it might show that it is taking zero milliseconds for execution. Try it on a method which is big enough, in the sense the one which is doing considerable amount of processing

**Question 19: Which two methods you need to implement to use an Object as key in HashMap?** ([answer](http://javarevisited.blogspot.com/2013/01/difference-between-identityhashmap-and-hashmap-java.html))

In order to use any object as Key in HashMap or Hashtable, it must implement equals and hash-code methods in Java. Read How HashMap works in Java for a detailed explanation on how equals and hash code method is used to put and get an object from HashMap.

**Question 20: How would you prevent a client from directly instantiating your concrete classes? For example, you have a Cache interface and two implementation classes MemoryCache and DiskCache, How do you ensure there is no object of this two classes is created by client using new() keyword.**

I leave this question for you to practice and think about before I give the answer. I am sure you can figure out the right way to do this, as this is one of the important decision to keep control of classes in your hand, great from a maintenance perspective.

I am also very grateful to my readers who have generously contributed several good questions from Java Interviews for both beginners and experienced developers alike. I have already answered many of these questions in this blog and you can easily find a relevant post by using the search box at the top right corner of this page.

### Multithreading, Concurrency and Thread basics Questions

**1) Can we make array volatile in Java?**  
This is one of the tricky Java multi-threading questions you will see in senior Java developer Interview. Yes, you can make an array volatile in Java but only the reference which is pointing to an array, not the whole array. What I mean, if one thread changes the reference variable to points to another array, that will provide a volatile guarantee, but if multiple threads are changing individual array elements they won't be having happens before guarantee provided by the volatile modifier.  
  
  
**2) Can volatile make a non-atomic operation to atomic?**  
This another good question I love to ask on volatile, mostly as a follow-up of the previous question. This question is also not easy to answer because volatile is not about atomicity, but there are cases where you can use a volatile variable to make the operation atomic.  
  
One example I have seen is having a long field in your class. If you know that a long field is accessed by more than one thread e.g. a counter, a price field or anything, you better make it volatile. Why? because reading to a long variable is not atomic in Java and done in two steps, If one thread is writing or updating long value, it's possible for another thread to see half value (fist 32-bit). While reading/writing a volatile long or double (64 bit) is atomic.  
  
  
  
**3) What are practical uses of volatile modifier?**  
One of the practical use of the volatile variable is to make reading double and long atomic. Both double and long are 64-bit wide and they are read in two parts, first 32-bit first time and next 32-bit second time, which is non-atomic but volatile double and long read is atomic in Java. Another use of the volatile variable is to provide a memory barrier, just like it is used in Disrupter framework. Basically, Java Memory model inserts a write barrier after you write to a volatile variable and a read barrier before you read it. Which means, if you write to volatile field then it's guaranteed that any thread accessing that variable will see the value you wrote and anything you did before doing that right into the thread is guaranteed to have happened and any updated data values will also be visible to all threads, because the memory barrier flushed all other writes to the cache.  
  
  
**4) What guarantee volatile variable provides?**([answer](http://java67.blogspot.sg/2012/08/what-is-volatile-variable-in-java-when.html))  
volatile variables provide the guarantee about ordering and visibility e.g. volatile assignment cannot be re-ordered with other statements but in the absence of any synchronization instruction compiler, JVM or JIT are free to reorder statements for better performance. volatile also provides the happens-before guarantee which ensures changes made in one thread is visible to others. In some cases volatile also provide atomicity e.g. reading 64-bit data types like long and double are not atomic but read of volatile double or long is atomic.  
  
  
  
**5) Which one would be easy to write? synchronization code for 10 threads or 2 threads?**  
In terms of writing code, both will be of same complexity because synchronization code is independent of a number of threads. Choice of synchronization though depends upon a number of threads because the number of thread present more contention, so you go for advanced synchronization technique e.g. lock stripping, which requires more complex code and expertise.  
  
  
**6) How do you call wait() method? using if block or loop? Why?**([answer](http://javarevisited.blogspot.sg/2015/07/how-to-use-wait-notify-and-notifyall-in.html))  
wait() method should always be called in loop because it's possible that until thread gets CPU to start running again the condition might not hold, so it's always better to check condition in loop before proceeding. Here is the standard idiom of using wait and notify method in Java:

// The standard idiom for using the wait method

synchronized (obj) {

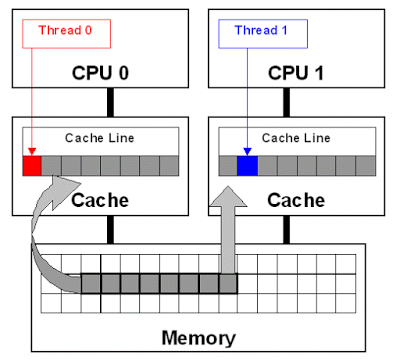
while (condition does not hold)

obj.wait(); // (Releases lock, and reacquires on wakeup)

... // Perform action appropriate to condition

}

See [Effective Java Item 69](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) to learn more about why wait method should call in the loop.  
  
  
**7)  What is false sharing in the context of multi-threading?**  
false sharing is one of the well-known performance issues on multi-core systems, where each process has its local cache. false sharing occurs when threads on different processor modify variables that reside on same cache line as shown in the following image:

[](https://2.bp.blogspot.com/-Tze9foqpb74/VepwCzXHGCI/AAAAAAAADtM/i4KQDaefqk4/s1600/False+Sharing+in+Multi-threaded+application.gif)

False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory. Like many concurrency issues, the primary way to avoid false sharing is careful code review and aligning your data structure with the size of a cache line.  
  
  
**8) What is busy spin? Why should you use it?**  
Busy spin is one of the technique to wait for events without releasing CPU. It's often done to avoid losing data in CPU cached which is lost if the thread is paused and resumed in some other core. So, if you are working on low latency system where your order processing thread currently doesn't have any order, instead of sleeping or calling wait(), you can just loop and then again check the queue for new messages. It's only beneficial if you need to wait for a very small amount of time e.g. in micro seconds or nano seconds. [LMAX Disrupter](http://lmax-exchange.github.io/disruptor/) framework, a high-performance inter-thread messaging library has a BusySpinWaitStrategy which is based on this concept and uses a busy spin loop for EventProcessors waiting on the barrier.  
  
  
**9) How do you take thread dump in Java?**  
You can take a thread dump of Java application in Linux by using **kill -3 PID**, where PID is the process id of Java process. In Windows, you can press **Ctrl + Break**. This will instruct JVM to print thread dump in standard out or err and it could go to console or log file depending upon your application configuration. If you have used Tomcat then when  
  
  
  
**10) is Swing thread-safe?**([answer](http://javarevisited.blogspot.sg/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html))  
No, Swing is not thread-safe. You cannot update Swing components e.g. JTable, JList or JPanel from any thread, in fact, they must be updated from GUI or AWT thread. That's why swings provide invokeAndWait() and invokeLater() method to request GUI update from any other threads. This methods put update request in AWT threads queue and can wait till update or return immediately for an asynchronous update. You can also check the detailed answer to learn more.  
  
  
**11) What is a thread local variable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html))  
Thread-local variables are variables confined to a thread, its like thread's own copy which is not shared between multiple threads. Java provides a ThreadLocal class to support thread-local variables. It's one of the many ways to achieve thread-safety. Though be careful while using thread local variable in manged environment e.g. with web servers where worker thread out lives any application variable. Any thread local variable which is not removed once its work is done can potentially cause a memory leak in Java application.  
  
  
**12) Write wait-notify code for producer-consumer problem?** ([answer](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html))  
Please see the answer for a code example. Just remember to call wait() and notify() method from synchronized block and test waiting for condition on the loop instead of if block.  
  
  
**13) Write code for thread-safe Singleton in Java?** ([answer](http://javarevisited.blogspot.in/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))  
Please see the answer for a code example and step by step guide to creating thread-safe singleton class in Java. When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads. Using Java enum as Singleton class is one of the easiest ways to create a thread-safe singleton in Java.  
  
  
**14) The difference between sleep and wait in Java?**([answer](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html))  
Though both are used to pause currently running thread, sleep() is actually meant for short pause because it doesn't release lock, while wait() is meant for conditional wait and that's why it release lock which can then be acquired by another thread to change the condition on which it is waiting.  
  
  
**15) What is an immutable object? How do you create an Immutable object in Java?** ([answer](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))  
Immutable objects are those whose state cannot be changed once created. Any modification will result in a new object e.g. String, Integer, and other wrapper class. Please see the answer for step by step guide to creating Immutable class in Java.  
  
  
**16) Can we create an Immutable object, which contains a mutable object?**  
Yes, its possible to create an Immutable object which may contain a mutable object, you just need to be a little bit careful not to share the reference of the mutable component, instead, you should return a copy of it if you have to. Most common example is an Object which contain the reference of java.util.Date object.

## Date types and Basic Java Interview Questions

**17) What is the right data type to represent a price in Java?**([answer](http://javarevisited.blogspot.sg/2012/02/java-mistake-1-using-float-and-double.html))  
BigDecimal if memory is not a concern and Performance is not critical, otherwise double with predefined precision.  
  
  
**18) How do you convert bytes to String?** ([answer](http://javarevisited.blogspot.sg/2014/08/2-examples-to-convert-byte-array-to-String-in-Java.html))  
you can convert bytes to the string using string constructor which accepts byte[], just make sure that right character encoding otherwise platform's default character encoding will be used which may or may not be same.  
  
  
**19) How do you convert bytes to long in Java?** (answer)  
This questions if for you to answer :-)  
  
  
**20) Can we cast an int value into byte variable? what will happen if the value of int is larger than byte?**  
Yes, we can cast but int is 32 bit long in java while byte is 8 bit long in java so when you cast an int to byte higher 24 bits are lost and a byte can only hold a value from -128 to 128.  
  
  
**21) There are two classes B extends A and C extends B, Can we cast B into C e.g. C = (C) B;**([answer](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html))  
  
  
**22) Which class contains clone method? Cloneable or Object?** ([answer](http://javarevisited.blogspot.sg/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html))  
java.lang.Cloneable is marker interface and doesn't contain any method clone method is defined in the object class. It is also knowing that clone() is a native method means it's implemented in C or C++ or any other native language.  
  
  
**23) Is ++ operator is thread-safe in Java?** (answer)  
 No it's not a thread safe operator because its involve multiple instructions like reading a value, incriminating it and storing it back into memory which can be overlapped between multiple threads.  
  
  
**24) Difference between a = a + b and a += b ?** (answer)  
The += operator implicitly cast the result of addition into the type of variable used to hold the result. When you add two integral variable e.g. variable of type byte, short, or int then they are first promoted to int and them addition happens. If result of addition is more than maximum value of a then a + b will give compile time error but a += b will be ok as shown below

byte a = 127;

byte b = 127;

b = a + b; *// error : cannot convert from int to byte*

b += a; *// ok*

**25) Can I store a double value in a long variable without casting?**([answer](http://java67.blogspot.com/2014/11/how-to-convert-double-to-long-in-java-example.html))  
No, you cannot store a double value into a long variable without casting because the range of double is more  that long and you we need to type cast. It's not dificult to answer this question but many develoepr get it wrong due to confusion on which one is bigger between double and long in Java.  
  
  
**26) What will this return 3\*0.1 == 0.3? true or false?**(answer)  
This is one of the really tricky questions. Out of 100, only 5 developers answered this question and only of them have explained the concept correctly. The short answer is false because some floating point numbers can not be represented exactly.  
  
  
**27) Which one will take more memory, an int or Integer?**(answer)  
An Integer object will take more memory an Integer is the an object and it  store meta data overhead about the object and int is primitive type so its takes less space.  
  
  
**28) Why is String Immutable in Java?** ([answer](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html))  
One of my favorite Java interview question. The String is Immutable in java because java designer thought that string will be heavily used and making it immutable allow some optimization easy sharing same String object between multiple clients. See the link for the more detailed answer. This is a great question for Java programmers with less experience as it gives them food for thought, to think about how things works in Java, what Jave designers might have thought when they created String class etc.  
  
**29) Can we use String in the switch case?** ([answer](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html))  
Yes from Java 7 onward we can use String in switch case but it is just syntactic sugar. Internally string hash code is used for the switch. See the detaiedl answer for more explanation and discussion.  
  
**30) What is constructor chaining in Java?** ([answer](http://java67.blogspot.sg/2012/12/how-constructor-chaining-works-in-java.html))  
When you call one constructor from other than it's known as constructor chaining in Java. This happens when you have multiple, overloaded constructor in the class.

### JVM Internals and Garbage Collection Interview Questions

In the year 2018 I have seen increased focus on JVM internal and Garbage collection tuning, monitoring Java appliation, dealing with Java performance issues on various Java interviews. This is actually become the prime topic for interviewing any experienced Java developer for senior position e.g. technical lead, VP or team lead. If you feel you are short of experience and knowledge in this area then you should read atleast one book mentioned in my list of [Java Performance books](http://javarevisited.blogspot.com/2014/07/top-5-java-performance-tuning-books.html). I vote goes to Java Performance, The Definitive guide by Scott.

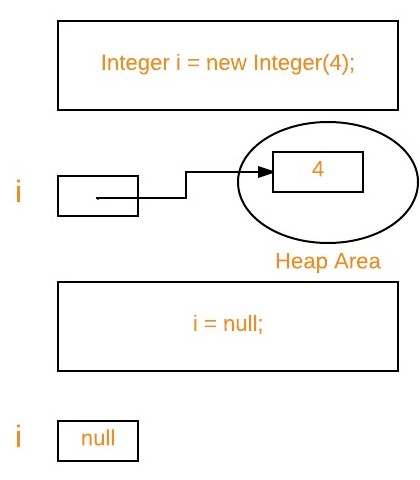
# Garbage Collection in Java

**Introduction**

* In C/C++, programmer is responsible for both creation and destruction of objects. Usually programmer neglects destruction of useless objects. Due to this negligence, at certain point, for creation of new objects, sufficient memory may not be available and entire program will terminate abnormally causing **OutOfMemoryErrors**.
* But in Java, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.
* Garbage collector is best example of [Daemon thread](https://www.geeksforgeeks.org/daemon-thread-java/) as it is always running in background.
* Main objective of Garbage Collector is to free heap memory by destroying **unreachable objects**.

**Important terms :**

1. **Unreachable objects :**An object is said to be unreachable iff it doesn’t contain any reference to it. Also note that objects which are part of [island of isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/) are also unreachable.
2. Integer i = new Integer(4);
3. // the new Integer object is reachable via the reference in 'i'
4. i = null;
5. // the Integer object is no longer reachable.



1. **Eligibility for garbage collection :**An object is said to be eligible for GC(garbage collection) iff it is unreachable. In above image, after i = null; integer object 4 in heap area is eligible for garbage collection.

**Ways to make an object eligible for GC**

* Even though programmer is not responsible to destroy useless objects but it is highly recommended to make an object unreachable(thus eligible for GC) if it is no longer required.
* There are generally four different ways to make an object eligible for garbage collection.
  + 1. Nullifying the reference variable
    2. Re-assigning the reference variable
    3. Object created inside method
    4. [Island of Isolation](https://www.geeksforgeeks.org/island-of-isolation-in-java/)

All above ways with examples are discussed in separate article : [How to make object eligible for garbage collection](https://www.geeksforgeeks.org/how-to-make-object-eligible-for-garbage-collection/)

**Ways for requesting**[**JVM**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)**to run Garbage Collector**

* Once we made object eligible for garbage collection, it may not destroy immediately by garbage collector. Whenever JVM runs Garbage Collector program, then only object will be destroyed. But when JVM runs Garbage Collector, we can not expect.
* We can also request JVM to run Garbage Collector. There are two ways to do it :
  + 1. **Using System.gc() method** : System class contain static method gc() for requesting JVM to run Garbage Collector.
    2. **Using Runtime.getRuntime().gc() method** : [Runtime class](https://www.geeksforgeeks.org/java-lang-runtime-class-in-java/) allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate requesting  // JVM to run Garbage Collector  public class Test  {      public static void main(String[] args) throws InterruptedException      {          Test t1 = new Test();          Test t2 = new Test();            // Nullifying the reference variable          t1 = null;            // requesting JVM for running Garbage Collector          System.gc();            // Nullifying the reference variable          t2 = null;            // requesting JVM for running Garbage Collector          Runtime.getRuntime().gc();        }        @Override      // finalize method is called on object once      // before garbage collecting it      protected void finalize() throws Throwable      {          System.out.println("Garbage collector called");          System.out.println("Object garbage collected : " + this);      }  } |

Output:

Garbage collector called

Object garbage collected : Test@46d08f12

Garbage collector called

Object garbage collected : Test@481779b8

**Note :**

* + 1. There is no guarantee that any one of above two methods will definitely run Garbage Collector.
    2. The call System.gc() is effectively equivalent to the call : Runtime.getRuntime().gc()

**Finalization**

* Just before destroying an object, Garbage Collector calls finalize() method on the object to perform cleanup activities. Once finalize() method completes, Garbage Collector destroys that object.
* finalize() method is present in [Object class](https://www.geeksforgeeks.org/object-class-in-java/) with following prototype.
* protected void finalize() throws Throwable

Based on our requirement, we can override finalize() method for perform our cleanup activities like closing connection from database.

**Note :**

* 1. The finalize() method called by Garbage Collector not [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). Although Garbage Collector is one of the module of JVM.
  2. [Object class](https://www.geeksforgeeks.org/object-class-in-java/) finalize() method has empty implementation, thus it is recommended to override finalize() method to dispose of system resources or to perform other cleanup.
  3. The finalize() method is never invoked more than once for any given object.
  4. If an uncaught exception is thrown by the finalize() method, the exception is ignored and finalization of that object terminates.

For examples on finalize() method, please see [Output of Java programs | Set 10 (Garbage Collection)](https://www.geeksforgeeks.org/output-of-java-programs-set-10-garbage-collection/)

**Related Articles :**

 [How to make object eligible for garbage collection in Java?](https://www.geeksforgeeks.org/how-to-make-object-eligible-for-garbage-collection/)

 [Island of Isolation in Java](https://www.geeksforgeeks.org/island-of-isolation-in-java/)

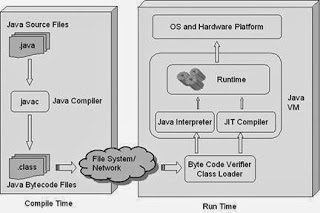
 [Output of Java programs | Set 10 (Garbage Collection)](https://www.geeksforgeeks.org/output-of-java-programs-set-10-garbage-collection/)

 [How to find max memory, free memory and total memory in Java?](https://www.geeksforgeeks.org/find-max-memory-free-memory-total-memory-java/)

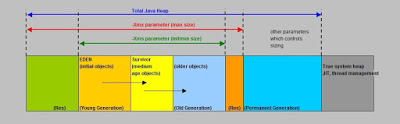
 [How JVM Works – JVM Architecture?](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)

This article is contributed by **Chirag Agarwal and Gaurav Miglani** .Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

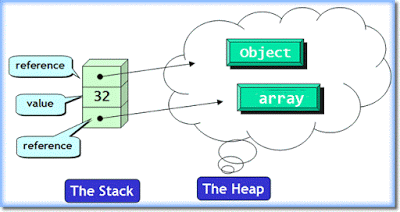
**31) What is the size of int in 64-bit JVM?**  
The size of an int variable is constant in Java, it's always 32-bit irrespective of platform. Which means the size of primitive int is same in both 32-bit and 64-bit Java virtual machine.  
  
**32) The difference between Serial and Parallel Garbage Collector?** ([answer](http://javarevisited.blogspot.sg/2011/04/garbage-collection-in-java.html))  
Even though both the serial and parallel collectors cause a stop-the-world pause during Garbage collection. The main difference between them is that a serial collector is a default copying collector which uses only one GC thread for garbage collection while a parallel collector uses multiple GC threads for garbage collection.  
  
**33) What is the size of an int variable in 32-bit and 64-bit JVM?**(answer)  
The size of int is same in both 32-bit and 64-bit JVM, it's always 32 bits or 4 bytes.  
  
**34) A difference between WeakReference and SoftReference in Java?**([answer](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html))  
Though both WeakReference and SoftReference helps garbage collector and memory efficient, WeakReference becomes eligible for garbage collection as soon as last strong reference is lost but SoftReference even thought it can not prevent GC, it can delay it until JVM absolutely need memory.  
  
**35) How do WeakHashMap works?**(answer)  
WeakHashMap works like a normal HashMap but uses WeakReference for keys, which means if the key object doesn't have any reference then both key/value mapping will become eligible for garbage collection.  
  
**36) What is -XX:+UseCompressedOops JVM option? Why use it?**([answer](http://javarevisited.blogspot.com/2012/06/what-is-xxusecompressedoops-in-64-bit.html))  
When you go migrate your Java application from 32-bit to 64-bit JVM, the heap requirement suddenly increases, almost double, due to increasing size of ordinary object pointer from 32 bit to 64 bit. This also adversely affect how much data you can keep in CPU cache, which is much smaller than memory. Since main motivation for moving to 64-bit JVM is to specify large heap size, you can save some memory by using compressed OOP. By using -XX:+UseCompressedOops, JVM uses 32-bit OOP instead of 64-bit OOP.  
  
  
**37) How do you find if JVM is 32-bit or 64-bit from Java Program?**([answer](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html))  
You can find that by checking some system properties like sun.arch.data.model or os.arch  
  
  
**38) What is the maximum heap size of 32-bit and 64-bit JVM?**([answer](http://javarevisited.blogspot.sg/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html))  
Theoretically, the maximum heap memory you can assign to a 32-bit JVM is 2^32 which is 4GB but practically the limit is much smaller. It also varies between operating systems e.g. form 1.5GB in Windows to almost 3GB in Solaris. 64-bit JVM allows you to specify larger heap size, theoretically 2^64 which is quite large but practically you can specify heap space up to 100GBs. There are even JVM e.g. Azul where heap space of 1000 gigs is also possible.  
  
  
**39) What is the difference between JRE, JDK, JVM and JIT?** ([answer](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html))  
JRE stands for Java run-time and it's required to run Java application. JDK stands for Java development kit and provides tools to develop Java program e.g. Java compiler. It also contains JRE. The JVM stands for Java virtual machine and it's the process responsible for running Java application. The JIT stands for Just In Time compilation and helps to boost the performance of Java application by converting Java byte code into native code when the crossed certain threshold i.e. mainly hot code is converted into native code.

[](https://2.bp.blogspot.com/-ls3yC0U7ouo/VhDqX-3OUbI/AAAAAAAAD40/Zcsc5uCaGq0/s1600/JVM+JRE+JDK.jpg)

**40) Explain Java Heap space and Garbage collection?** ([answer](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html))  
When a Java process is started using java command, memory is allocated to it. Part of this memory is used to create heap space, which is used to allocate memory to objects whenever they are created in the program. Garbage collection is the process inside JVM which reclaims memory from dead objects for future allocation.

[](https://3.bp.blogspot.com/-DqV12_uIeZ4/VhDqtPCVIVI/AAAAAAAAD48/uqWZB0BgZUI/s1600/java_heaps_memory.jpg)

**41) Can you guarantee the garbage collection process?**(answer)  
No, you cannot guarantee the garbage collection, though you can make a request using System.gc() or Runtime.gc() method.  
  
  
**42) How do you find memory usage from Java program? How much percent of the heap is used?**  
You can use memory related methods from java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java.  By using these methods, you can find out how many percents of the heap is used and how much heap space is remaining. Runtime.freeMemory() return amount of free memory in bytes, Runtime.totalMemory() returns total memory in bytes and Runtime.maxMemory() returns maximum memory in bytes.  
  
  
**43) What is the difference between stack and heap in Java?**([answer](http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html))  
Stack and heap are different memory areas in the JVM and they are used for different purposes. The stack is used to hold method frames and local variables while objects are always allocated memory from the heap. The stack is usually much smaller than heap memory and also didn't shared between multiple threads, but heap is shared among all threads in JVM.

[](https://1.bp.blogspot.com/-NZeVo83YJAA/VhDrDO0oWtI/AAAAAAAAD5E/mEek8Ll7NfU/s1600/Difference+between+stack+and+heap+memory+in+Java.gif)

## Basic Java concepts Interview Questions

**44) What's the difference between "a == b" and "a.equals(b)"?** ([answer](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html))  
The a = b does object reference matching if both a and b are an object and only return true if both are pointing to the same object in the heap space, on the other hand, a.equals(b) is used for logical mapping and its expected from an object to override this method to provide logical equality. For example, String class overrides this equals() method so that you can compare two Strings, which are the different object but contains same letters.  
  
  
**45) What is a.hashCode() used for? How is it related to a.equals(b)?**([answer](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
hashCode() method returns an int hash value corresponding to an object. It's used in hash based collection classes e.g Hashtable, HashMap, LinkedHashMap and so on. It's very tightly related to equals() method. According to Java specification, two objects which are equal to each other using equals() method must have same hash code.  
  
  
**46) Difference between final, finalize and finally?**([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-final-finally-and-finalize-java.html))  
The final is a modifier which you can apply to variable, methods and classes. If you make a variable final it means its value cannot be changed once initialized. finalize is a method, which is called just before an object is a garbage collected, giving it last chance to resurrect itself, but the call to finalize is not guaranteed. finally is a keyword which is used in exception handling along with try and catch. the finally block is always executed irrespective of whether an exception is thrown from try block or not.  
  
  
**47) What is a compile time constant in Java? What is the risk of using it?**  
public static final variables are also known as a compile time constant, the public is optional there. They are replaced with actual values at compile time because compiler know their value up-front and also knows that it cannot be changed during run-time. One of the problem with this is that if you happened to use a public static final variable from some in-house or third party library and their value changed later than your client will still be using old value even after you deploy a new version of JARs. To avoid that, make sure you compile your program when you upgrade dependency JAR files.

## Java Collections Framework Interview Questions

It also contains Data structure and algorithm Interview question in Java, questions on array, linked list, HashMap, ArrayList, Hashtable, Stack, Queue, PriorityQueue, LinkedHashMap and ConcurrentHashMap.  
  
**48) The difference between List, Set, Map, and Queue in Java?** ([answer](http://java67.blogspot.sg/2013/01/difference-between-set-list-and-map-in-java.html))  
The list is an ordered collection which allows duplicate. It also has an implementation which provides constant time index based access, but that is not guaranteed by List interface. Set is unordered collection which  
  
  
**49) Difference between poll() and remove() method?**  
Both poll() and remove() take out the object from the Queue but if poll() fails then it returns null but if remove fails it throws Exception.  
  
  
**50) The difference between LinkedHashMap and PriorityQueue in Java?**([answer](http://javarevisited.blogspot.sg/2013/10/what-is-priorityqueue-data-structure-java-example-tutorial.html))  
PriorityQueue guarantees that lowest or highest priority element always remain at the head of the queue, but LinkedHashMap maintains the order on which elements are inserted. When you iterate over a PriorityQueue, iterator doesn't guarantee any order but iterator of LinkedHashMap does guarantee the order on which elements are inserted.  
  
  
**51) Difference between ArrayList and LinkedList in Java?** ([answer](http://java67.blogspot.sg/2012/12/difference-between-arraylist-vs-LinkedList-java.html))  
The obvious difference between them is that ArrrayList is backed by array data structure, supprots random access and LinkedList is backed by linked list data structure and doesn't supprot random access. Accessing an element with the index is O(1) in ArrayList but its O(n) in LinkedList. See the answer for more detailed discussion.  
  
  
**52) What is a couple of ways that you could sort a collection?** ([answer](http://java67.blogspot.sg/2012/07/sort-list-ascending-descending-order-set-arraylist.html))  
You can either use the Sorted collection like TreeSet or TreeMap or you can sort using the ordered collection like a list and using Collections.sort() method.  
  
  
**53) How do you print Array in Java?** ([answer](http://java67.blogspot.sg/2014/03/how-to-print-array-in-java-example-tutorial.html))  
You can print an array by using the Arrays.toString() and Arrays.deepToString() method. Since array doesn't implement toString() by itself, just passing an array to System.out.println() will not print its contents but Arrays.toString() will print each element.

**54) LinkedList in Java is doubly or singly linked list?** (answer)  
It's a doubly linked list, you can check the code in JDK. In Eclipse, you can use the [shortcut](http://javarevisited.blogspot.com/2010/10/eclipse-tutorial-most-useful-eclipse.html), Ctrl + T to directly open this class in Editor.  
  
**55) Which kind of tree is used to implement TreeMap in Java?** (answer)  
A Red Black tree is used to implement TreeMap in Java.

**56) What is the difference between Hashtable and HashMap?**([answer](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html))  
There are many differences between these two classes, some of them are following:  
a) Hashtable is a legacy class and present from JDK 1, HashMap was added later.  
b) Hashtable is synchronized and slower but HashMap is not synchronized and faster.  
c) Hashtable doesn't allow null keys but HashMap allows one null key.  
See the answer for more differences between HashMap and Hashtable in Java.  
  
  
**57) How HashSet works internally in Java?** ([answer](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html))  
HashSet is internally implemented using an HashMap. Since a Map needs key and value, a default value is used for all keys. Similar to HashMap, HashSet doesn't allow duplicate keys and only one null key, I mean you can only store one null object in HashSet.  
  
  
**58) Write code to remove elements from ArrayList while iterating?** ([answer](http://javarevisited.blogspot.sg/2014/01/ow-to-remove-objects-from-collection-arraylist-java-iterator-traversing.html))  
 Key here is to check whether candidate uses ArrayList's remove() or Iterator's remove(). Here is the [sample code](http://java67.blogspot.com/2015/10/how-to-solve-concurrentmodificationexception-in-java-arraylist.html) which uses right way o remove elements from ArrayList while looping over and avoids ConcurrentModificationException.  
  
  
**59) Can I write my own container class and use it in the for-each loop?**  
Yes, you can write your own container class. You need to implement the Iterable interface if you want to loop over advanced for loop in Java, though. If you implement Collection then you by default get that property.  
  
  
**60) What is default size of ArrayList and HashMap in Java?**([answer](http://javarevisited.blogspot.sg/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html))  
As of Java 7 now, default size of ArrayList is 10 and default capacity of HashMap is 16, it must be power of 2. Here is code snippet from ArrayList  and HashMap class :

// from ArrayList.java JDK 1.7

private static final int DEFAULT\_CAPACITY = 10;

//from HashMap.java JDK 7

static final int DEFAULT\_INITIAL\_CAPACITY = 1 **<<** 4; // aka 16

**61) Is it possible for two unequal objects to have the same hashcode?**  
Yes, two unequal objects can have same hashcode that's why collision happen in a hashmap.  
the equal hashcode contract only says that two equal objects must have the same hashcode it doesn't say anything about the unequal object.  
  
**62) Can two equal object have the different hash code?**  
No, thats not possible according to hash code contract.  
  
  
**63) Can we use random numbers in the hashcode() method?** ([answer](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html))  
No, because hashcode of an object should be always same. See the answer to learning more about things to remember while overriding hashCode() method in Java.  
  
  
**64) What is the difference between Comparator and Comparable in Java?**([answer](http://java67.blogspot.sg/2013/08/difference-between-comparator-and-comparable-in-java-interface-sorting.html))  
The Comparable interface is used to define the  natural order of object while Comparator is used to define custom order. Comparable can be always one, but we can have multiple comparators to define customized order for objects.  
  
**65) Why you need to override hashcode, when you override equals in Java?** ([answer](http://javarevisited.blogspot.sg/2015/01/why-override-equals-hashcode-or-tostring-java.html))  
 Because equals have code contract mandates to override equals and hashcode together .since many container class like HashMap or HashSet depends on hashcode and equals contract.

### Java IO and NIO Interview questions

IO is very important from Java interview point of view. You should have a good knowledge of old Java IO, NIO, and NIO2 alsong with some operating system and disk IO fundamentals. Here are some frequently asked questions form Java IO.  
  
66) In my Java program, I have three sockets? How many threads I will need to handle that?  
  
67) How do you create ByteBuffer in Java?  
  
68) How do you write and read from ByteBuffer in Java?  
  
69) Is Java BIG endian or LITTLE endian?  
  
70) What is the byte order of ByteBuffer?  
  
71) The difference between direct buffer and non-direct buffer in Java? ([answer](http://javarevisited.blogspot.sg/2015/08/difference-between-direct-non-direct-mapped-bytebuffer-nio-java.html))  
  
72) What is the memory mapped buffer in Java? ([answer](http://javarevisited.blogspot.sg/2012/01/memorymapped-file-and-io-in-java.html))  
  
73) What is TCP NO DELAY socket option?  
  
74) What is the difference between TCP and UDP protocol? ([answer](http://javarevisited.blogspot.com/2014/07/9-difference-between-tcp-and-udp-protocol.html))  
  
75) The difference between ByteBuffer and StringBuffer in Java? (answer)

### Java Best Practices Interview question

Contains best practices from different parts of Java programming e.g. Collections, String, IO, Multi-threading, Error and Exception handling, design patterns etc. This section is mostly for experience Java developer, technical lead,  AVP, team lead and coders who are responsible for products. If you want to create quality products you must know and follow the best practices.  
  
**76) What best practices you follow while writing multi-threaded code in Java?** ([answer](http://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html))  
Here are couple of best practices which I follow while writing concurrent code in Java:  
a) Always name your thread, this will help in debugging.  
b) minimize the scope of synchronization, instead of making whole method synchronized, only critical section should be synchronized.  
c) prefer volatile over synchronized if you can can.  
e) use higher level concurrency utilities instead of waitn() and notify for inter thread communication e.g. BlockingQueue, CountDownLatch and Semeaphore.  
e) Prefer concurrent collection over synchronized collection in Java. They provide better scalability.  
  
  
**77) Tell me few best practices you apply while using Collections in Java?**(answer)  
Here are couple of best practices I follow while using Collectionc classes from Java:  
a) Always use the right collection e.g. if you need non-synchronized list then use ArrayList and not Vector.  
b) Prefer concurrent collection over synchronized collection because they are more scalable.  
c) Always use interface to a represent and access a collection e.g. use List to store ArrayList, Map to store HashMap and so on.  
d) Use iterator to loop over collection.  
e) Always use generics with collection.  
  
  
**78) Can you tell us at least 5 best practice you use while using threads in Java?** ([answer](http://java67.blogspot.com/2014/01/10-points-about-thread-and-javalangthread-in-java.html))  
This is similar to the previous question and you can use the answer given there. Particularly with thread, you should:  
a) name your thread  
b) keep your task and thread separate, use Runnable or Callable with thread pool executor.  
c) use thread pool  
d) use volatile to indicate compiler about ordering, visibility, and atomicity.  
e) avoid thread local variable because incorrect use of ThreadLocal class in Java can create a memory leak.  
Look there are many best practices and I give extra points to the developer which bring something new, something even I don't know. I make sure to ask this question to Java developers of 8 to 10 years of experience just to gauge his hands on experience and knowledge.  
  
  
**79) Name 5 IO best practices?** (answer)  
IO is very important for performance of your Java application. Ideally you should avoid IO in critical path of your application. Here are couple of Java IO best practices you can follow:

a) Use buffered IO classes instead of reading individual bytes and char.

b) Use classes from NIO and NIO2

c) Always close streams in finally block or use try-with-resource statements.

d) use memory mapped file for faster IO.

If a Java candidate doesn't know about IO and NIO, especially if he has at least 2 to 4 years of experience, he needs some reading.  
  
  
**80) Name 5 JDBC best practices your follow?** ([answer](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html))  
Another good Java best practices for experienced Java developer of 7 to 8 years experience. Why it's important? because they are the ones which set the trend in the code and educate junior developers. There are many best practices and you can name as per your confort and conviniece. Here are some of the more common ones:  
a) use batch statement for inserting and updating data.  
b) use PreparedStatement to avoid SQL exception and better performance.  
c) use database connection pool  
d) access resultset using column name instead of column indexes.  
e) Don't generate dynamic SQL by concatenating String with user input.  
  
  
**81) Name couple of method overloading best practices in Java?** ([answer](http://javarevisited.blogspot.sg/2013/01/java-best-practices-method-overloading-constructor.html))  
Here are some best practices you can follow while overloading a method in Java to avoid confusion with auto-boxing:  
a) Don't overload method where one accepts int and other accepts Integer.  
b) Don't overload method where number of argument is same and only order of argument is different.  
c) Use varargs after overloaded methods has more than 5 arguments.

### Date, Time and Calendar Interview questions in Java

**82) Does SimpleDateFormat is safe to use in the multi-threaded program?** ([answer](http://javarevisited.blogspot.sg/2012/03/simpledateformat-in-java-is-not-thread.html))  
No, unfortunately, DateFormat and all its implementations including SimpleDateFormat is not thread-safe, hence should not be used in the multi-threaded program until external thread-safety measures are applied e.g. confining SimpleDateFormat object into a ThreadLocal variable. If you don't do that, you will get an incorrect result while parsing or formatting dates in Java. Though, for all practical date time purpose, I highly recommend **joda-time** library.  
  
  
**83) How do you format a date in Java? e.g. in the ddMMyyyy format?** ([answer](http://javarevisited.blogspot.com/2011/09/convert-date-to-string-simpledateformat.html))  
You can either use SimpleDateFormat class or joda-time library to format date in Java. DateFormat class allows you to format date on many popular formats. Please see the answer for code samples to format date into different formats e.g. dd-MM-yyyy or ddMMyyyy.  
  
  
84) How do you show timezone in formatted date in Java? ([answer](http://java67.blogspot.sg/2013/01/how-to-format-date-in-java-simpledateformat-example.html))  
  
85) The difference between java.util.Date and java.sql.Date in Java? ([answer](http://java67.blogspot.sg/2014/02/how-to-convert-javautildate-to-javasqldate-example.html))  
  
86) How to you calculate the difference between two dates in Java? ([program](http://javarevisited.blogspot.sg/2015/07/how-to-find-number-of-days-between-two-dates-in-java.html))  
  
87) How do you convert a String(YYYYMMDD) to date in Java? ([answer](http://java67.blogspot.sg/2014/12/string-to-date-example-in-java-multithreading.html))

### Unit testing JUnit Interview questions

89) How do you test static method? (answer)  
You can use PowerMock library to test static methods in Java.  
  
90) How to do you test a method for an exception using JUnit? ([answer](http://javarevisited.blogspot.sg/2013/04/JUnit-tutorial-example-test-exception-thrown-by-java-method.html))  
  
91) Which unit testing libraries you have used for testing Java programs? (answer)  
  
92) What is the difference between @Before and @BeforeClass annotation? ([answer](http://javarevisited.blogspot.sg/2013/04/JUnit-tutorial-example-test-exception-thrown-by-java-method.html))

### Programming and Coding Questions

93) How to check if a String contains only numeric digits? ([solution](http://java67.blogspot.com/2014/01/java-regular-expression-to-check-numbers-in-String.html))  
  
94) How to write LRU cache in Java using Generics? (answer)  
  
95) Write a Java program to convert bytes to long? (answer)  
  
96) How to reverse a String in Java without using StringBuffer? ([solution](http://java67.blogspot.com/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.htm))  
  
97) How to find the word with the highest frequency from a file in Java? ([solution](http://java67.blogspot.com/2015/10/java-program-to-find-repeated-words-and-count.html))  
  
98) How do you check if two given String are anagrams? ([solution](http://javarevisited.blogspot.sg/2013/03/Anagram-how-to-check-if-two-string-are-anagrams-example-tutorial.html))  
  
99) How to print all permutation of a String in Java? ([solution](http://javarevisited.blogspot.com/2015/08/how-to-find-all-permutations-of-string-java-example.html))  
  
100) How do you print duplicate elements from an array in Java? ([solution](http://javarevisited.blogspot.com/2015/06/3-ways-to-find-duplicate-elements-in-array-java.html))  
  
101) How to convert String to int in Java? ([solution](http://java67.blogspot.com/2015/08/2-ways-to-parse-string-to-int-in-java.html))  
  
102) How to swap two integers without using temp variable? ([solution](http://java67.blogspot.com/2015/08/how-to-swap-two-integers-without-using.html))

### Java Interview questions from OOP and Design Patterns

It contains Java Interview questions from SOLID design principles, OOP fundamentals e.g. class, object, interface, Inheritance, Polymorphism, Encapsulation, and Abstraction as well as more advanced concepts like Composition, Aggregation, and Association. It also contains questions from GOF design patterns.  
  
**103) What is the interface? Why you use it if you cannot write anything concrete on it?**  
The interface is used to define API. It tells about the contract your classes will follow. It also supports abstraction because a client can use interface method to leverage multiple implementations e.g. by using List interface you can take advantage of [random access of ArrayList](http://javarevisited.blogspot.com/2015/07/java-arraylist-tutorial.html) as well as flexible insertion and deletion of LinkedList. The interface doesn't allow you to write code to keep things abstract but from Java 8 you can declare static and default methods inside interface which are concrete.  
  
  
**104) The difference between abstract class and interface in Java?**([answer](http://javarevisited.blogspot.sg/2013/05/difference-between-abstract-class-vs-interface-java-when-prefer-over-design-oops.html))  
There are multiple differences between abstract class and interface in Java, but the most important one is Java's restriction on allowing a class to extend just one class but allows it to implement multiple interfaces. An abstract class is good to define default behavior for a family of class, but the interface is good to define Type which is later used to leverage Polymorphism. Please check the answer for a more thorough discussion of this question.  
  
  
**105) Which design pattern have you used in your production code? apart from Singleton?**  
This is something you can answer from your experience. You can generally say about dependency injection, factory pattern, decorator pattern or observer pattern, whichever you have used. Though be prepared to answer follow-up question based upon the pattern you choose.  
  
  
**106) Can you explain Liskov Substitution principle?** ([answer](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html))  
This is one of the toughest questions I have asked in Java interviews. Out of 50 candidates, I have almost asked only 5 have managed to answer it. I am not posting an answer to this question as I like you to do some research, practice and spend some time to understand this confusing principle well.  
  
  
**107) What is Law of Demeter violation? Why it matters?** ([answer](http://javarevisited.blogspot.com/2014/05/law-of-demeter-example-in-java.html))  
Believe it or not, Java is all about application programming and structuring code. If  you have good knowledge of common coding best practices, patterns and what not to do than only you can write quality code.  Law of Demeter suggests you "talk to friends and not stranger", hence used to reduce coupling between classes.  
  
  
**108) What is Adapter pattern? When to use it?**  
Another frequently asked Java design pattern questions. It provides interface conversion. If your client is using some interface but you have something else, you can write an Adapter to bridge them together. This is good for Java software engineer having 2 to 3 years experience because the question is neither difficult nor tricky but requires knowledge of OOP design patterns.  
  
  
**109) What is "dependency injection" and "inversion of control"? Why would someone use it?**([answer](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html))  
  
**110) What is an abstract class? How is it different from an interface? Why would you use it?**([answer](http://java67.blogspot.sg/2014/06/why-abstract-class-is-important-in-java.html))  
One more classic question from Programming Job interviews, it is as old as chuck Norris. An abstract class is a class which can have state, code and implementation, but an interface is a contract which is totally abstract. Since I have answered it many times, I am only giving you the gist here but you should read the article linked to answer to learn this useful concept in much more detail.  
  
  
**111) Which one is better constructor injection or setter dependency injection?**([answer](http://javarevisited.blogspot.sg/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html))  
Each has their own advantage and disadvantage. Constructor injection guaranteed that class will be initialized with all its dependency, but setter injection provides flexibility to set an optional dependency. Setter injection is also more readable if you are using an XML file to describe dependency. Rule of thumb is to use constructor injection for mandatory dependency and use setter injection for optional dependency.  
  
  
**112) What is difference between dependency injection and factory design pattern?** ([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-dependency-injection.html))  
Though both patterns help to take out object creation part from application logic, use of dependency injection results in cleaner code than factory pattern. By using dependency injection, your classes are nothing but POJO which only knows about dependency but doesn't care how they are acquired. In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern. Please see the answer for a more detailed discussion on this topic.  
  
  
**113) Difference between Adapter and Decorator pattern?**([answer](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
Though the structure of Adapter and Decorator pattern is similar, the difference comes on the intent of each pattern. The adapter pattern is used to bridge the gap between two interfaces, but Decorator pattern is used to add new functionality into the class without the modifying existing code.  
  
  
**114) Difference between Adapter and Proxy Pattern?**([answer](http://javarevisited.blogspot.sg/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
Similar to the previous question, the difference between Adapter and Proxy patterns is in their intent. Since both Adapter and Proxy pattern encapsulate the class which actually does the job, hence result in the same structure, but Adapter pattern is used for interface conversion while the Proxy pattern is used to add an extra level of indirection to support distribute, controlled or intelligent access.  
  
  
**115) What is Template method pattern?** (answer)  
Template pattern provides an outline of an algorithm and lets you configure or customize its steps. For examples, you can view a sorting algorithm as a template to sort object. It defines steps for sorting but let you configure how to compare them using Comparable or something similar in another language. The method which outlines the algorithms is also known as template method.  
  
  
**116) When do you use Visitor design pattern?**(answer)  
The visitor pattern is a solution of problem where you need to add operation on a class hierarchy but without touching them. This pattern uses double dispatch to add another level of indirection.  
  
  
**117) When do you use Composite design pattern?**(answer)  
Composite design pattern arranges objects into tree structures to represent part-whole hierarchies. It allows clients treat individual objects and container of objects uniformly. Use Composite pattern when you want to represent part-whole hierarchies of objects.

**118) The difference between Inheritance and Composition?** ([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-inheritance-and-Composition-in-Java-OOP.html))  
Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies.  
  
  
**119) Describe overloading and overriding in Java?** ([answer](http://java67.blogspot.sg/2012/09/difference-between-overloading-vs-overriding-in-java.html))  
Both overloading and overriding allow you to write two methods of different functionality but with the same name, but overloading is compile time activity while overriding is run-time activity. Though you can overload a method in the same class, but you can only override a method in child classes. Inheritance is necessary for overriding.  
  
  
**120) The difference between nested public static class and a top level class in Java?** ([answer](http://javarevisited.blogspot.sg/2012/12/inner-class-and-nested-static-class-in-java-difference.html))  
You can have more than one nested public static class inside one class, but you can only have one top-level public class in a Java source file and its name must be same as the name of Java source file.  
  
  
**121) Difference between Composition, Aggregation and Association in OOP?** ([answer](http://javarevisited.blogspot.sg/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))  
If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation. In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.  
  
  
**122) Give me an example of design pattern which is based upon open closed principle?** ([answer](http://javarevisited.blogspot.sg/2011/11/great-example-of-open-closed-design.html))  
This is one of the practical questions I ask experienced Java programmer. I expect them to know about OOP design principles as well as patterns. Open closed design principle asserts that your code should be open for extension but closed for modification. Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code.  There are several design patterns which are based upon open closed design principle e.g. [Strategy pattern](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) if you need a new strategy, just implement the interface and configure, no need to modify core logic. One working example is Collections.sort() method which is based on Strategy pattern and follows the open-closed principle, you don't modify sort() method to sort a new object, what you do is just implement Comparator in your own way.  
  
  
**123) Difference between Abstract factory and Prototype design pattern?** (answer)  
This is the practice question for you, If you are feeling bored just reading and itching to write something, why not write the answer to this question. I would love to see an example the, which should answer where you should use the Abstract factory pattern and where is the Prototype pattern is more suitable.  
  
  
**124) When do you use Flyweight pattern?** (answer)  
This is another popular question from the design pattern. Many Java developers with 4 to 6 years of experience know the definition but failed to give any concrete example. Since many of you might not have used this pattern, it's better to look examples from JDK. You are more likely have used them before and they are easy to remember as well. Now let's see the answer.  
Flyweight pattern allows you to share object to support large numbers without actually creating too many objects. In order to use Flyweight pattern, you need to make your object Immutable so that they can be safely shared. String pool and pool of Integer and Long object in JDK are good examples of Flyweight pattern.

### Miscellaneous Java Interview Questions

It contains XML Processing in Java Interview question, JDBC Interview question, Regular expressions Interview questions, Java Error and Exception Interview Questions, Serialization,  
  
**125) The difference between nested static class and top level class?**([answer](http://java67.blogspot.sg/2012/10/nested-class-java-static-vs-non-static-inner.html))  
One of the fundamental questions from Java basics. I ask this question only to junior Java developers of 1 to 2 years of experience as it's too easy for an experience Java programmers. The answer is simple, a public top level class must have the same name as the name of the source file, there is no such requirement for nested static class. A nested class is always inside a top level class and you need to use the name of the top-level class to refer nested static class e.g. HashMap.Entry is a nested static class, where HashMap is a top level class and Entry is nested static class.  
  
  
**126) Can you write a regular expression to check if String is a number?**([solution](http://javarevisited.blogspot.sg/2012/10/regular-expression-example-in-java-to-check-String-number.html))  
If you are taking Java interviews then you should ask at least one question on the regular expression. This clearly [differentiates an average programmer with a good programmer](http://javarevisited.blogspot.com/2015/05/how-to-differentiate-between-average.html). Since one of the traits of a good developer is to know tools, regex is the best tool for searching something in the log file, preparing reports etc. Anyway, answer to this question is, a numeric String can only contain digits i.e. 0 to 9 and + and - sign that too at start of the String, by using this information you can write following regular expression to check if given String is number or not  
  
  
**127) The difference between checked and unchecked Exception in Java?**([answer](http://java67.blogspot.sg/2012/12/difference-between-runtimeexception-and-checked-exception.html))  
checked exception is checked by the compiler at compile time. It's mandatory for a method to either handle the checked exception or declare them in their throws clause. These are the ones which are a sub class of Exception but doesn't descend from RuntimeException. The unchecked exception is the descendant of RuntimeException and not checked by the compiler at compile time. This question is now becoming less popular and you would only find this with interviews with small companies, both investment banks and startups are moved on from this question.  
  
  
**128) The difference between throw and throws in Java?** ([answer](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html))  
the throw is used to actually throw an instance of java.lang.Throwable class, which means you can throw both Error and Exception using throw keyword e.g.

throw new IllegalArgumentException("size must be multiple of 2")

On the other hand, throws is used as part of method declaration and signals which kind of exceptions are thrown by this method so that its caller can handle them. It's mandatory to declare any unhandled checked exception in **throws** clause in Java. Like the previous question, this is another frequently asked Java interview question from errors and exception topic but too easy to answer.  
  
  
**129) The difference between Serializable and Externalizable in Java?** ([answer](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html))  
This is one of the frequently asked questions from Java Serialization. The interviewer has been asking this question since the day Serialization was introduced in Java, but yet only a few good candidate can answer this question with some confidence and practical knowledge. Serializable interface is used to make Java classes serializable so that they can be transferred over network or their state can be saved on disk, but it leverages default serialization built-in JVM, which is expensive, fragile and not secure. Externalizable allows you to fully control the Serialization process, specify a custom binary format and add more security measure.  
  
  
**130) The difference between DOM and SAX parser in Java?**([answer](http://javarevisited.blogspot.sg/2011/12/difference-between-dom-and-sax-parsers.html))  
Another common Java question but from XML parsing topic. It's rather simple to answer and that's why many interviewers prefers to ask this question on the telephonic round. DOM parser loads the whole XML into memory to create a tree based DOM model which helps it quickly locate nodes and make a change in the structure of XML while SAX parser is an event based parser and doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and not suitable to parse large XML files.  
  
  
**131) Tell me 3 features introduced on JDK 1.7?**([answer](http://javarevisited.blogspot.sg/2014/04/10-jdk-7-features-to-revisit-before-you.html))  
This is one of the good questions I ask to check whether the candidate is aware of recent development in Java technology space or not. Even though JDK 7 was not a big bang release like JDK 5 or JDK 8, it still has a lot of good feature to count on e.g. try-with-resource statements, which free you from closing streams and resources when you are done with that, Java automatically closes that. Fork-Join pool to implement something like the Map-reduce pattern in Java. Allowing String variable and literal into switch statements. Diamond operator for improved type inference, no need to declare generic type on the right-hand side of variable declaration anymore, results in more readable and succinct code. Another worth noting feature introduced was improved exception handling e.g. allowing you to catch multiple exceptions in the same catch block.  
  
  
**132) Tell me 5 features introduced in JDK 1.8?**([answer](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html))  
This is the follow-up question of the previous one. Java 8 is path breaking release in Java's history, here are the top 5 features from JDK 8 release

* **Lambda expression**, which allows you pass an anonymous function as object.
* **Stream API**, take advantage of multiple cores of modern CPU and allows you to write succinct code.
* **Date and Time API**, finally you have a solid and easy to use date and time library right into JDK
* **Extension methods**, now you can have static and default method into your interface
* **Repeated annotation**, allows you apply the same annotation multiple times on a type

**133) What is the difference between Maven and ANT in Java?** ([answer](http://javarevisited.blogspot.sg/2015/01/difference-between-maven-ant-jenkins-and-hudson.html))  
Another great question to check the all round knowledge of Java developers. It's easy to answer questions from core Java but when you ask about setting things up, building Java artifacts, many Java software engineer struggles. Coming back to the answer of this question, Though both are build tool and used to create Java application build, Maven is much more than that. It provides standard structure for Java project based upon "convention over configuration" concept and automatically manage dependencies (JAR files on which your application is dependent) for Java application. Please see the answer for more differences between Maven and ANT tool.  
  
  
That's all guys, **lots of Java Interview questions?** isn't it? I am sure if you can answer this list of Java questions you can easily crack any core Java or advanced Java interview. Though I have not included questions from Java EE or J2EE topics e.g. Servlet, JSP, JSF, JPA, JMS, EJB or any other Java EE technology or from major web frameworks like Spring MVC, Struts 2.0, Hibernate or both SOAP and RESTful web services, it's still useful for Java developers preparing for Java web developer position, because every Java interview starts with questions from fundamentals and JDK API. If you think, I have missed any popular Java question here and you think it should be in this list then feel free to suggest me. My goal is to create the best list of Java Interview Questions with latest and greatest question from recent interviews.  
  
  
Read more: <https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#ixzz5WEprlU5W>

**Question 1: What’s wrong with using HashMaps in a multi-threaded environment? When does a get() method go into an infinite loop?**([answer](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html))

Well, nothing is wrong — it depends on how you use it. For example, if you initialize the HashMap with just one thread and all threads are only reading from it, then it's perfectly fine.

One example of this is a Map that contains configuration properties. The real problem starts when at least one of those threads is updating the HashMap, i.e. adding, changing, or removing any key-value pair.

Since a put() operation can cause re-sizing, which can further lead to infinite loop, that's why either you should use a [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html) or [ConcurrentHashMap](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) (the latter is better).

**Question 2. Does not overriding a hashCode() method have any performance implication?**([answer](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html))

This is a good question and open to all, as per my knowledge, a poor hashCode function will result in frequent [collisions in HashMap](http://javarevisited.blogspot.sg/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html), which eventually increases the time for adding an object into said HashMap.

From [Java 8](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fjava-8-core-training-%2F) onwards, though, collisions will not impact performance as much as in earlier versions because, after a threshold, the linked list will be replaced by the binary tree, which will give you **O(log N)**performance in the worst case, as compared to **O(N)** of a linked list.

This is one of the several [tricky Java questions](http://java67.blogspot.com/2012/09/top-10-tricky-java-interview-questions-answers.html) you will face, as many developers only know about equals hashcode contracts and don't think about their performance implications.

**Question 3: Do all properties of immutable objects need to be final?**([answer](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))

Not necessarily. As stated above, you can achieve the same functionality by making members non-final but [private](http://javarevisited.blogspot.sg/2012/10/difference-between-private-protected-public-package-access-java.html#axzz59Lhz7uVu) — and not modifying them except in constructors.

Don't provide a [setter method](http://javarevisited.blogspot.sg/2012/12/getter-and-setter-method-vs-public-modifier-field-java.html#axzz55oDxm8vv) for them, and if it is a mutable object, then don't ever leak any reference to that member.

Remember, making a reference variable [final](http://javarevisited.blogspot.sg/2016/09/21-java-final-modifier-keyword-interview-questions-answers.html#axzz59Lhz7uVu) only ensures that it will not be reassigned a different value, but you can still change individual properties of an object pointed by that reference variable.

This is one of the key points an interviewer always likes to hear from candidates. By mentioning that, you can score some brownie points.

**Question 4: How does a substring () inside a String work?**([answer](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html))

Many developers know the answer: “A substring creates a new object out of the source string by taking a portion of original string.”

But I think that answer is insufficient.

This question was mainly asked to see if the developer is familiar with the risk of **memory leaks** that substrings can create.

Until Java 1.7, a substring holds the reference of the original character array, which means even a substring 5 characters long can prevent a 1GB character array from garbage collection by holding a [strong reference](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html#axzz54AAeS1IM).

This issue is fixed in Java 1.7, where the original character array is not referenced anymore, but that change also made the creation of substrings a bit more costly in terms of time. Earlier, it was in the range of O(1), which could be O(N) in the worst case in Java 7.

**Question 5: Can you write critical section code for a singleton?**([answer](http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html))

This core Java question is a follow-up of the previous question and expects the candidate to write a Java singleton using the [Double-Checked Locking](http://www.java67.com/2016/04/why-double-checked-locking-was-broken-before-java5.html) Pattern.

Remember to use a [volatile variable](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html) to make Singleton thread-safe.

Here is the code for a critical section of a [thread-safe singleton](http://javarevisited.blogspot.sg/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) class using the Double-Checked Locking idiom:

public class Singleton {

private static volatile Singleton \_instance;

/\*\*

\* Double checked locking code on Singleton

\* @return Singelton instance

\*/

public static Singleton getInstance() {

if (\_instance == null) {

synchronized(Singleton.class) {

if (\_instance == null) {

\_instance = new Singleton();

}

}

}

return \_instance;

}

}

**Question 6: How do you handle error conditions while writing a stored procedure or accessing stored procedures from Java?**([answer](http://javarevisited.blogspot.com/2013/04/spring-framework-tutorial-call-stored-procedures-from-java.html))

This is one of the [tough Java interview questions](http://www.java67.com/2012/09/top-10-tough-core-java-interview-questions-answers.html) and it's open for all. I have a friend who didn't know the answer, and he didn't mind telling me.

My take is that stored procedure should return an error code if some operation fails, but if the stored procedure itself fails, then catching a [SQLException](http://javarevisited.blogspot.sg/2016/09/javasqlsqlexception-no-suitable-driver-mysql-jdbc-localhost.html#axzz59Lhz7uVu) is the only choice.

**Question 7 : What is the difference between the Executor.submit() and Executer.execute() methods?**([answer](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html))

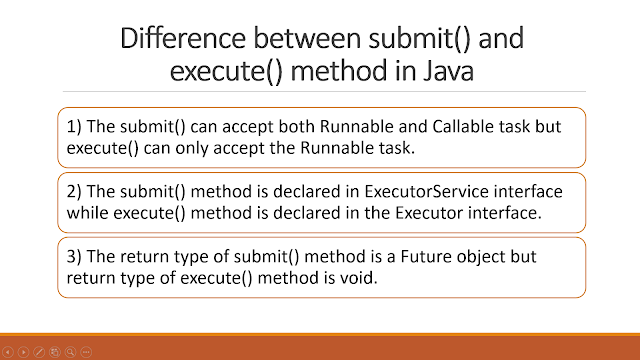
## Executor.execute() vs ExecutorService.submit() method

As I told in the first paragraph that key difference between the execute() and submit() method is that former cannot return the result but later can result of computation. Before seeing more difference, let's see some similarities between the execute() and submit() as well:

1) Both submit() and execute() methods are used to submit a task to [Executor framework](http://javarevisited.blogspot.com/2013/07/how-to-create-thread-pools-in-java-executors-framework-example-tutorial.html) for asynchronous execution.  
  
2) Both submit() and execute() can accept a Runnable task.  
  
3) You can access submit() and execute() from the ExecutorService interface because it also extends the Executor interface which declares the execute() method.  
  
Apart from the fact that submit() method can return output and execute() cannot, following are other notable differences between these two key methods of Executor framework of Java 5.  
  
1) The submit() can accept both [Runnable](http://java67.blogspot.com/2016/01/7-differences-between-extends-thread-vs-implements-Runnable-java.html) and [Callable](http://javarevisited.blogspot.com/2015/06/how-to-use-callable-and-future-in-java.html) task but execute() can only accept the Runnable task.  
  
2) The submit() method is declared in ExecutorService interface while execute() method is declared in the Executor interface.  
  
3) The return type of submit() method is a Future object but return type of execute() method is void.  
  
Btw, Cay S. Horstmann has also covered this essential topic on good detail in his classic book, [Core Java Volume 1 - Fundamentals](http://www.amazon.com/Volume-II-Advanced-Features-Edition-Series/dp/013708160X?tag=javamysqlanta-20), 10th Edition. You can refer that book for further reading on this topic.

## When to use submit() and execute() method in Java

Once you understand the difference between Executor.execute() and ExecutorService.submit() method you have the knowledge to decide w*hen to use submit() and when to use the execute() method*.  
  
In general, if you are doing computational task e.g. calculating some risk stats, [calculating factorial of large numbers](http://java67.blogspot.com/2015/09/how-to-use-biginteger-class-in-java.html) or doing some time-consuming computation e which results in some value then use the submit() method. It immediately returns a Future object, which can be later queried to get the value of computation by calling get() method.  
  
Remember, get() is a [blocking call](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html) so always call the version which accepts a timeout. While you can use the execute() method if you just want your code to be run in parallel by worker threads of the thread pool.  
  
Here is a nice summary of key differences between submit() vs execute() methods in Java:

[](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html)

That's all about **difference between ExecutorService.submit() and Executor.execute() method in Java**. Remember, the key difference between them is that submit() return a Future but execute() return nothing.  
  
The thread pools created by Executors class always return a reference of ExecutorService, which provides access to both submit() and execute() method as it also extend the Executor interface, which is source of execute() method.  
  
Use submit() if your doing computation e.g. calculating value of pie, and use execute() if you just want the code to be run in parallel by worker threads of thread pool.  
  
Read more: <https://javarevisited.blogspot.com/2016/04/difference-between-ExecutorServie-submit-vs-Executor-execute-method-in-Java.html#ixzz5axcGxEz2>

**Question 8: What is the difference between the Factory and Abstract Factory patterns?**([answer](http://javarevisited.blogspot.sg/2013/01/difference-between-factory-and-abstract-factory-design-pattern-java.html))

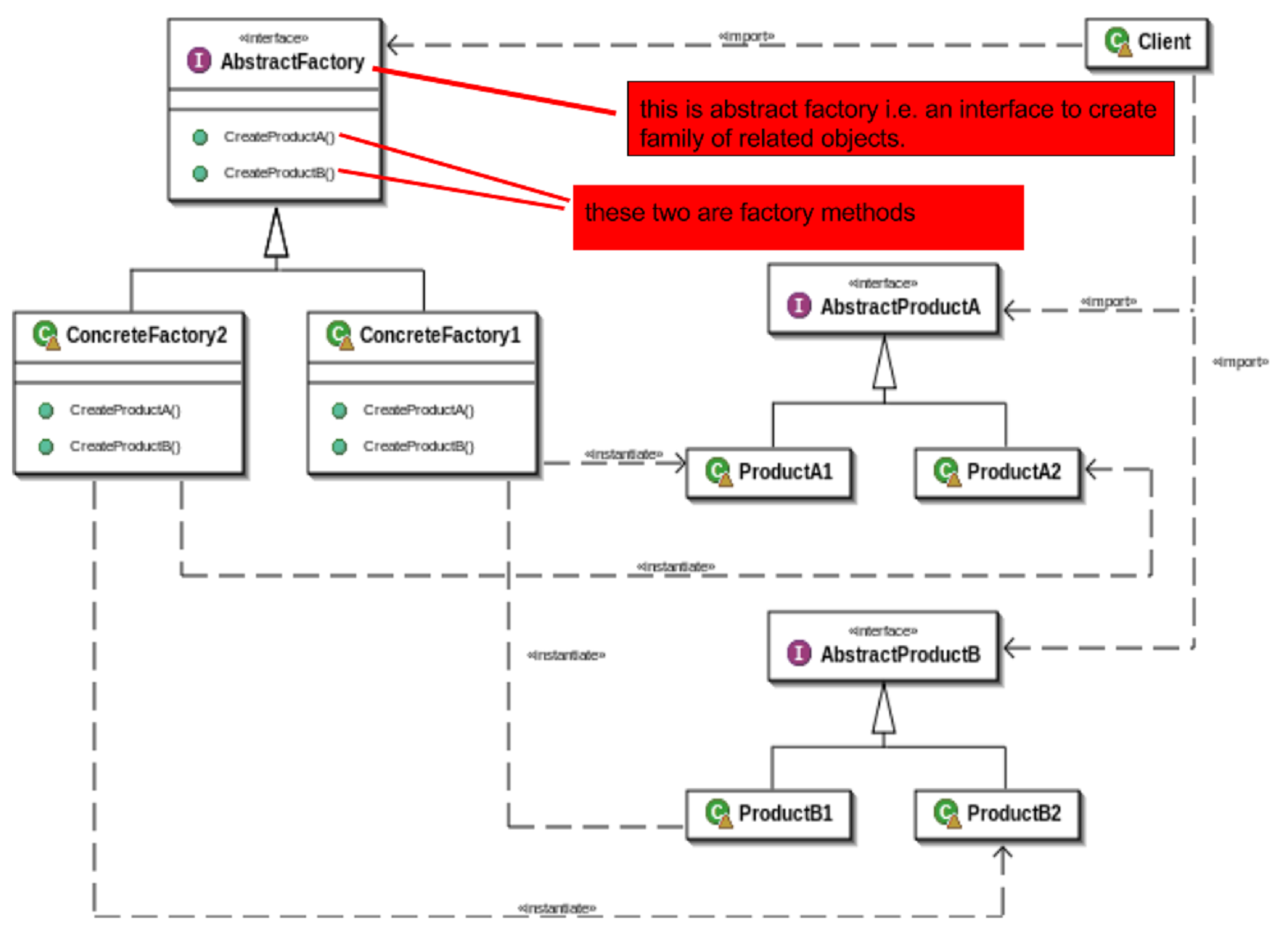
Compared to the Factory pattern, Abstract Factory provides one more level of [Abstraction](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html#axzz59Lhz7uVu).

Consider different factories each extended from an Abstract Factory and that are responsible for the creation of different hierarchies of objects based on the type of factory.

For example, AbstractFactory is extended by AutomobileFactory, UserFactory, RoleFactory, etc.

Each individual factory would be responsible for the creation of objects in that genre.

Here is a UML diagram of the [Factory](http://javarevisited.blogspot.sg/2015/06/difference-between-dependency-injection.html) and [Abstract Factory](http://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html) patterns:



**Question 9: What is Singleton? Is it better to make the whole method synchronized, or only the critical section?**([answer](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))

Singleton in Java is a class with just one instance in an entire Java application. For example, java.lang.Runtime is a Singleton class.

Creating a Singleton was tricky prior to Java 5, but once Java 5 introduced [Enum](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html), it becames very easy.

Please see my article [How to create thread-safe Singletons in Java](http://javarevisited.blogspot.sg/2012/07/why-enum-singleton-are-better-in-java.html#axzz4tzMEHSJw) for more details on writing Singletons using Enums and double-checked locking, which is the purpose of this Java interview question.

**Question 10: Can you write code for iterating over HashMaps in Java 4 and Java 5?**([answer](http://java67.blogspot.com/2014/05/3-examples-to-loop-map-in-java-foreach.html))

This is a tricky one, but as the answerer shows, he managed to write using while and a for loop. Actually, there are[four ways to iterate over any Map in Java](http://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html).

The method involves using [keySet()](http://www.java67.com/2016/05/keyset-vs-entryset-vs-values-in-java-map-example.html) and iterating over the key, then using a [get()](http://www.java67.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) method to retrieve values, which is bit expensive.

The second method involves using entrySet() and iterating over them either by using a for each loop or while with the Iterator.hasNext() method.

This one is a better approach because both key and value objects are available to you during iteration and you don't need to call the [get()](http://javarevisited.blogspot.sg/2011/02/how-hashmap-works-in-java.html) method for retrieving the value, which could give O(N) performance in the case of a huge linked list in one bucket.

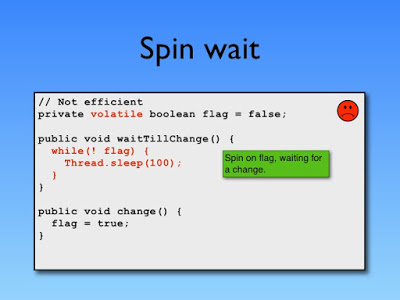
This could be slightly better if you are using Java 8 and there is a tree instead of the linked list.

## Conclusion

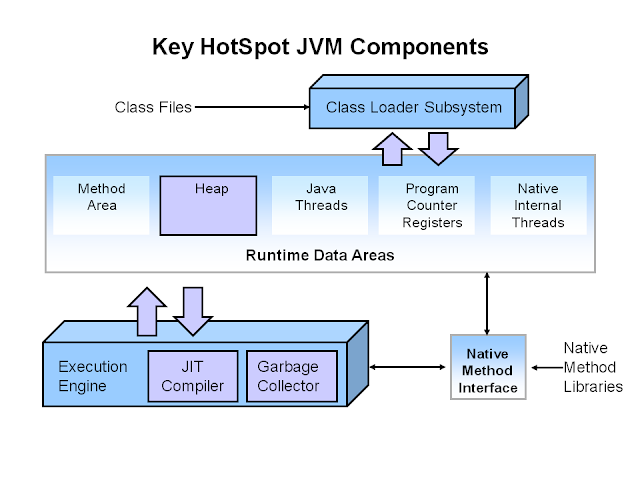
That's all about some of the common Java interview questions from investment banks. If you are going for a Java developer role, then expect a lot of focus on Java concurrency, multithreading, collections, JVM internals, garbage collection, and how to increase the performance of Java applications.

I have not touched all the topics you need to be ready for during Java interviews in this article, but if you are serious about preparation, here are a couple of useful resources to find more structured and topical questions:

**1. What is Busy Spinning? Why Should You Use It in Java?**  
One of the interesting multithreading question to senior Java programmers, busy spinning is a *waiting strategy*, in which a thread just wait in a loop, without releasing the CPU for going to [sleep](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html). This is a very advanced and specialized waiting strategy used in the high-frequency trading application when wait time between two messages is very minimal.  
  
By not releasing the CPU or [suspending the thread](http://java67.blogspot.com/2015/06/how-to-pause-thread-in-java-using-sleep.html), your thread retains all the cached data and instruction, which may be lost if the thread was suspended and resumed back in a different core of CPU.   
  
This question is quite popular in high-frequency low latency programming domain, where programmers are trying for extremely low latency in the range of micro to milliseconds. See here more [50+ advanced thread interview questions](http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html) for experienced programmers. 

[](https://3.bp.blogspot.com/-SN8fjcDcDS0/V6NDiqjeF-I/AAAAAAAAGxk/41lOd51QXCcjB1k4PQUYXNDfJpffg04bQCLcB/s1600/busy+spin+Java+multithreading.jpg)

**2. What is Read-Write Lock? Does ConcurrentHashMap in Java Uses The ReadWrite Lock?**  
ReadWrite Lock is an implementation of *lock stripping* technique, where two separate locks are used for read and write operation. Since read operation doesn't modify the state of the object, it's safe to allow multiple thread access to a shared object for reading without locking, and by splitting one lock into [read and write lock](http://javarevisited.blogspot.com/2014/10/how-to-use-locks-in-multi-threaded-java-program-example.html), you can easily do that.   
  
Java provides an implementation of read-write lock in the form of ReentrantReadWritLock class in the java.util.concurrent.lock package. This is worth looking before you decide to write your own read-write locking implementation.   
  
Also, the current implementation of java.util.ConcurrentHashMap doesn't use the ReadWriteLock, instead, it divides the Map into several segments and locks them separately using different locks. This means any given time, *only a portion of the ConcurrentHashMap is locked*, instead of the whole Map. See [how ConcurrentHashMap internally works in Java](http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html) for more detail.   
  
  
This core Java question is also very popular on senior and more experienced level Java interviews e.g. 4 to 6 years, where you expect Interviewer to go into more detail, e.g. by asking you to provided an implementation of the read-write lock with different policies. If you are an experienced Java programmer, consider reading [Java Concurrency in Practice](http://www.amazon.com/dp/0321349601/?tag=javamysqlanta-20) to gain more confidence about multithreading and concurrency in Java.   
  
  
  
**3. How to Make an Object Immutable in Java? Why Should You Make an Object Immutable?**  
Well, Immutability offers several advantages including thread-safety, ability to cache and result in more readable multithreading code. See [here](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html)to learn how to make object Immutable. Once again, this question can also go into more detail and depending on your answer, can bring several other questions e.g. when you mention Spring is Immutable, be ready with some reasons on [Why String is Immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html).  
  
  
**4. Which Design Patterns have You Used in Your Java Project?**  
Always expect some design patterns related question for Core Java Interview of senior developer position. It's a better strategy to mention any GOF design pattern rather than Singleton or MVC, which almost every other Java developer use it.   
  
Your best bet can be [Decorator pattern](http://java67.blogspot.sg/2013/07/decorator-design-pattern-in-java-real-life-example-tutorial.html) or may be [Dependency Injection Pattern](http://javarevisited.blogspot.sg/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html), which is quite popular in Spring Framework. It's also good to mention only the design patterns which you have *actually* used in your project and knows it's tradeoffs.   
  
It's common that once you mention a particular design pattern say Factory or Abstract Factory, Interviewer's next question would be, *have you used this pattern in your project?* So be ready with proper example and why you choose a particular pattern. You can also see [this](http://java67.blogspot.com/2012/09/top-10-java-design-pattern-interview-question-answer.html) article for more advanced design pattern questions from Java interviews.   
  
  
  
**5.  Do you know about Open Closed Design Principle or Liskov Substitution Principle?**  
Design patterns are based on [object-oriented design principles](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html), which I strongly felt every object-oriented developer and the programmer should know, or, at least, have a basic idea of what are these principles and how they help you to write better object oriented code. I  
  
f you don't know the answer to this question, you can politely say No, as it's not expected from you to know the answer to every question, but by answering this question, you can make your claim stronger as many experienced developers fail to answer basic questions like this. See [Clean Code](http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882?tag=javamysqlanta-20) to learn more about object-oriented and SOLID design principles.  
 **6. Which Design Pattern Will You Use to Shield Your Code From a Third Party library Which Will Likely to be Replaced by Another in Couple of Months?**  
This is just one example of  the scenario-based design pattern interview question. In order to test the practical experience of Java developers with more than 5 years experience, companies ask this kind of questions.  You can expect more real-world design problems in different formats, some with more detail explanation with context, or some with only intent around.   
  
One way to shield your code from third party library is to [code against an interface rather than implementation](http://javarevisited.blogspot.com/2014/11/why-use-interface-in-java-or-object-oriented-programming.html) and then use dependency injection to provide a particular implementation. This kind of questions is also asked quite frequently to experienced and senior *Java developers with 5 to 7 years of experience*.  
 **Question 7)  How  do you prevent SQL Injection in Java Code?**  
This question is more asked to J2EE and Java EE developers than core Java developers, but, it is still a good question to check the JDBC and Security skill of experienced Java programmers.  
  
You can [use PreparedStatement to avoid SQL injection](http://javarevisited.blogspot.com/2012/03/why-use-preparedstatement-in-java-jdbc.html) in Java code. Use of the PreparedStatement for executing SQL queries not only provides better performance but also shield your Java and J2EE application from SQL Injection attack.   
  
On a similar note, If you are working more on Java EE or J2EE side, then you should also be familiar with other security issues including *Session Fixation attack* or *Cross Site Scripting* attack and how to resolve them. These are some fields and questions where a good answer can make a lot of difference on your selection.   
 **Question 8) Tell me about different Reference types available in Java, e.g. WeakReference, SoftReference or PhantomReference? and Why should you use them?**  
Well, they are different reference types coming from java.lang.ref package and provided to assist Java Garbage Collector in a case of low memory issues. If you wrap an object with [WeakReference](http://javarevisited.blogspot.com/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html) than it will be eligible for garbage collected if there are o strong reference. They can later be reclaimed by Garbage collector if JVM is running low on memory.  
  
The java.util.WeakHashMap is a special Map implementation, whose keys are the object of WeakReference, so if only Map contains the reference of any object and no other, those object can be garbage collected if GC needs memory. See [Java Performance The Definitive Guide](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20) learn more about how to deal with performance issues in Java. 

[](http://java67.blogspot.com/2013/02/difference-between-jit-and-jvm-in-java.html)

**Question 9) How does get method of HashMap works in Java?**  
Yes, this is still one of the most popular core Java questions for senior developer interviews. You can also expect this question on telephonic round, followed by lot's of follow-up questions as discussed on my post [how does HashMap work in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html).   
  
The short answer to this question is that HashMap is based upon hash table data structure and uses hashCode() method to calculate hash code to find the bucket location on underlying array and equals() method to search the object in the same bucket in case of a collision. See [here](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) to learn a more about how does get() method of HashMap works in Java.   
  
  
  
**Question 10) Which Two Methods HashMap key Object Should Implement?**  
This is one of the follow-up questions I was saying about in previous questions. Since working of HashMap is based upon hash table data structure, any object which you want to use as key for HashMap or any other hash based collection e.g. Hashtable, or ConcurrentHashMap must implement [equals()](http://java67.blogspot.com/2012/11/difference-between-operator-and-equals-method-in.html) and [hashCode()](http://java67.blogspot.com/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html) method.   
  
The hashCode() is used to find the bucket location i.e. index of underlying array and equals() method is used to find the right object in linked list stored in the bucket in case of a collision. By the way, from Java 8, HashMap also started using tree data structure to store the object in case of collision to reduce worst case performance of HashMap from O(n) to O(logN). See the article for learning more about [how does HashMap handless collision in Java](http://javarevisited.blogspot.com/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html).   
  
  
  
**Question 11) Why Should an Object Used As the Key should be Immutable?**   
This is another follow-up of previous core Java interview questions. It's good to test the depth of technical knowledge of candidate by asking more and more question on the same topic. If you know about Immutability, you can answer this question by yourself. The short answer to this question is key should be immutable so that hashCode() method  always return the same value.

Since hash code returned by hashCode() method depends on upon the content of object i.e. values of member variables. If an object is [mutable](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) than those values can change and so is the hash code. If the same object returns different hash code once you inserted the value in HashMap, you will end up searching in different bucket location and will not able to retrieve the object. That's why a key object should be immutable. It's not a rule enforced by the compiler but you should take care of it as an experienced programmer. See the article for more [advanced Java Collection interview questions](http://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html).   
  
  
  
**Question 12) How does ConcurrentHashMap achieves its Scalability?**   
Sometimes this multithreading + collection interview question is also asked as, the difference between ConcurrentHashMap and Hashtable in Java. The problem with [synchronized HashMap](http://java67.blogspot.com/2015/02/how-to-synchronize-hashmap-in-java-with.html) or [Hashtable](http://java67.blogspot.com/2012/08/difference-between-hashmap-and-concurrentHashMap-java-collection.html) was that whole Map is locked when a thread performs any operation with Map.   
  
The java.util.ConcurrentHashMap class solves this problem by using *lock stripping* technique, where the whole map is locked at different segments and only a particular segment is locked during the write operation, not the whole map. The ConcurrentHashMap also achieves it's scalability by allowing lock-free reads as read is a thread-safe operation.  See [here](http://java67.blogspot.com/2012/08/5-thread-interview-questions-answers-in.html) for more advanced multi-threading and concurrency questions in Java.   
  
  
  
**Question 13) How do you share an object between threads? or How to pass an object from one thread to another?**  
There are multiple ways to do that e.g. Queues, Exchanger etc, but BlockingQueue using [Producer Consumer pattern](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html) is the easiest way to pass an object from thread to another.  
  
  
  
**Question 14) How do find if your program has a deadlock?**  
By taking thread dump using kill -3, using JConsole or VisualVM), I suggest to prepare this core java interview question in more detail, as Interviewer definitely likes to go with more detail e.g. they will press with questions like, have you really done that in your project or not?  
  
  
**Question 15) How do you avoid deadlock while coding?**  
By ensuring locks are acquire and released in an ordered manner, see [here](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html)for detail answer of this question.  
  
Read more: <http://www.java67.com/2013/07/15-advanced-core-java-interview-questions-answers-senior-experienced-5-6-years-programmers-developers.html#ixzz5WEqZjgW0>

## Top Java design pattern questions and answers

Here is my list of t*op 10 design pattern interview question in Java*. I have also provided an answer to those Java design pattern question as a link. no matter which level of Java interview is you going e.g. programmer, software engineer, senior software engineer in Java, you can expect few question from Java design pattern.  
  
**1. When to use Strategy Design Pattern in Java?**  
[Java design pattern interview question and answers for senior and experience programmer](http://3.bp.blogspot.com/-1lzFJzIgaHk/UF2Ci6kY5pI/AAAAAAAAAes/OYiM7r-DHzc/s1600/17.jpg)Strategy pattern in quite useful for implementing set of related algorithms e.g. compression algorithms, filtering strategies etc. Strategy design pattern allows you to create Context classes, which uses Strategy implementation classes for applying business rules. This pattern follows open closed design principle and quite useful in Java.  
  
One of a good example of Strategy pattern from JDK itself is a Collections.sort() method and [Comparator interface](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html), which is a strategy interface and defines a strategy for comparing objects. Because of this pattern, we don't need to modify sort() method (closed for modification) to compare any object, at the same time we can implement Comparator interface to define new comparing strategy (open for extension).  
  
  
**2. What is Observer design pattern in Java? When do you use Observer pattern in Java?**  
This is one of the most common Java design pattern interview questions. Observer pattern is based upon notification, there are two kinds of object Subject and Observer. Whenever there is change on subject's state observer will receive notification. See [What is Observer design pattern in Java with real life example](http://javarevisited.blogspot.sg/2011/12/observer-design-pattern-java-example.html) for more details.  
  
  
  
**3. Difference between Strategy and State design Pattern in Java?**  
This is an interesting Java design pattern interview questions as both Strategy and State pattern has the same structure. If you look at UML class diagram for both patterns they look exactly same, but their intent is totally different.  
  
State design pattern is used to define and manage the state of an object, while Strategy pattern is used to define a set of an interchangeable algorithm and let's client choose one of them. So [Strategy pattern](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Ffrom-0-to-1-design-patterns%2F) is a client driven pattern while Object can manage their state itself.  
  
  
  
**4. What is decorator pattern in Java? Can you give an example of Decorator pattern?**  
Decorator pattern is another popular Java design pattern question which is common because of its heavy usage in java.io package. BufferedReader and BufferedWriter are a good example of decorator pattern in Java. See [How to use Decorator pattern in Java](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html) for more details.  
  
  
**5. When to use Composite design Pattern in Java? Have you used previously in your project?**  
This design pattern question is asked on Java interview not just to check familiarity with the Composite pattern but also, whether a candidate has the real life experience or not.  
The*Composite pattern* is also a core Java design pattern, which allows you to treat both whole and part object to treat in a similar way. Client code, which deals with a Composite or individual object doesn't differentiate between them, it is possible because Composite class also implement the same interface as their individual part.  
  
One of the good examples of the Composite pattern from JDK is JPanel class, which is both Component and Container.  When the paint() method is called on JPanel, it internally called paint() method of individual components and let them draw themselves.  
  
On the second part of this design pattern interview question, be truthful, if you have used then say yes, otherwise say that you are familiar with the concept and used it by your own. By the way, always remember, giving an example from your project creates a better impression.  
  
  
  
**6. What is Singleton pattern in Java?**  
Singleton pattern in Java is a pattern which allows only one instance of Singleton class available in the whole application. java.lang.Runtime is a good example of Singleton pattern in Java. There are lot's of follow up questions on Singleton pattern see [10 Java singleton interview question answers](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) for those followups  
  
  
**7. Can you write thread-safe Singleton in Java?**  
There are multiple ways to write thread-safe singleton in Java e.g by writing singleton using double checked locking, by using static Singleton instance initialized during [class loading.](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html) By the way using Java enum to create thread-safe singleton is the most simple way. See [Why Enum singleton is better in Java](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) for more details.  
  
  
**8. When to use Template method design Pattern in Java?**  
The Template pattern is another popular core Java design pattern interview question. I have seen it appear many times in real life project itself. Template pattern outlines an algorithm in form of template method and lets subclass implement individual steps.  
  
The key point to mention, while answering this question is that template method should be final, so that subclass can not override and change steps of the algorithm, but same time individual step should be abstract, so that child classes can implement them.  
  
  
**9. What is Factory pattern in Java? What is the advantage of using a static factory method to create an object?**  
Factory pattern in Java is a creation Java design pattern and favorite on many Java interviews.Factory pattern used to create an object by providing static factory methods. There are many advantages of providing factory methods e.g. caching immutable objects, easy to introduce new objects etc. See [What is Factory pattern in Java and benefits](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html) for more details.  
  
  
**10. What is the difference between Decorator and Proxy pattern in Java?**  
Another tricky Java design pattern question and trick here is that both Decorator and Proxy implements the interface of the object they decorate or encapsulate. As I said, many Java design pattern can have similar or exactly same structure but they differ in their intent.  
  
Decorator pattern is used to implement functionality on an already created object, while a Proxy pattern is used for controlling access to an object.  
  
One more difference between Decorator and the Proxy design pattern is that Decorator doesn't create an object, instead, it get the object in its constructor, while Proxy actually creates objects. You can also read [Head First Analysis and Design](http://www.amazon.com/dp/0596008678/?tag=javamysqlanta-20) to understand the difference between them.  
  
  
**11. When to use Setter and Constructor Injection in Dependency Injection pattern?**  
Use Setter injection to provide optional dependencies of an object, while use Constructor iInjection to provide a mandatory dependency of an object, without which it can not work. This question is related to [Dependency Injection design pattern](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html) and mostly asked in the context of Spring framework, which is now become a standard for developing Java application.  
  
Since Spring provides IOC container, it also gives you a way to specify dependencies either by using setter methods or constructors. You can also take a look my [previous post](http://javarevisited.blogspot.com/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html) on the same topic.  
  
  
**12. What is difference between Factory and Abstract Factory in Java**  
I have already answered this question in detail with my article with the same title. The main difference is that Abstract Factory creates factory while Factory pattern creates objects. So both abstract the creation logic but one abstract is for factory and other for items. You can see [here](http://javarevisited.blogspot.sg/2013/01/difference-between-factory-and-abstract-factory-design-pattern-java.html) to answer this Java design pattern interview question.  
  
  
**13. When to use Adapter pattern in Java? Have you used it before in your project?**  
Use Adapter pattern when you need to make two class work with incompatible interfaces. Adapter pattern can also be used to encapsulate third party code so that your application only depends upon Adapter, which can adapt itself when third party code changes or you moved to a different third party library.  
  
By the way, this Java design pattern question can also be asked by providing an actual scenario. You can further read [Head First Design Pattern](http://www.amazon.com/dp/0596007124/?tag=javamysqlanta-20) to learn more about Adapter pattern and its real world usage. The book is updated for Java 8 as well so you will learn new, Java 8 way to implement these old design patterns.  
  
Read more: <http://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz5WEtONUU9>

**persistent**

Callable interface

Which executor framework you have used

Why need to overide the hashcode method

Can we call different class method using lamda expression

What is volatile and its used

Internal implentation of HashMap

Fail fast and fail safe example

Use of synchronized

What is detachedState

When will use the load method

Integration of spring and Hibernate

Why dialect is use

How spring container will create object internally.

How to create war file and deploy the application

Which build tool you have used

Use of @RestController in spring

How we call websrvices from spring

What is use of produces and consumes in spring

1. Java 7 and 8 features

2.Fibanacci series

3. How arrayList work internay and which DS using?

4.What is List internal implementation

5.JVM archituture

6.JDBC Connection(CRUD operation)

executeStatement/PraparedStatement/Callable Statement

7.Get all columns used in table using java code

9.How much memory a string can take in the memory size

String length\*2

10.HashMap vs Has

Shallow copy

Deep copy

cloning

Shallow copy

Shallow copy is the process in which the state of the object is copied to another object, but both the objects point to the same reference in heap area.

Deep Copy

In Deep Copy, two separate objects are created and in deep copy. In this each field of one object is copied to another object.

Now third category to overcome this difficulty in java is the concept of cloning.

Cloning in java is done by implements Cloneable interface. Cloneable is marker interface.

For more deep knowledge on cloning Refer : Cloning in java

What is transient

What is Volatile and synchronized keyword

Weak and Soft Reference?

1. Java 7 and 8 features

2.Fibanacci series

3. How arrayList work internay and which DS using?

4.What is List internal implementation

5.JVM archituture

6.JDBC Connection(CRUD operation)

executeStatement/PraparedStatement/Callable Statement

7.Get all columns used in table using java code

9.How much memory a string can take in the memory size

String length\*2

10.HashMap vs Has

DXC technology

* 1. WAP for two thread which will check the even and odd number
  2. WAP for an Arraylist.
  3. WAP to reverse string.
  4. DS

Core java

* 1. 1. Tell me some bilt in interface in java.
  2. 2. Diference between interface and abstract class
  3. 3. Internal implementation of hashmap.
  4. 4. What is final class and some exaples of final and imutable class.
  5. 6. Use of functional interface in java.
  6. 7. Some string operation

Hibernate

* 1. 1. Why we are using hibernate instead of jdbc.
  2. 2.lazy and eager loading in java
  3. 3. Diffrence between load and get.
  4. 5. When object will be detached.
  5. 6. Which types of creteria u have used in ur project.
  6. 7. Which types of annotation u have used in ur project.

Jdbc

* 1. 1.Prepared statement and callable uses

Spring

* 1. 1.which context u have used and why?
  2. 2. When will use singleton and prototype scope.
  3. 3. Flow of mvc once will click on login button of login page.
  4. 4. Which taglib u have used for validation of form.
  5. 5. How to call webservices from spring.

Restful web services

* 1. 1. Which api u have used for jax.rs

Servlet and jsp

* 1. 1. Which method we have to overide if want to implement servlet interface.
  2. 2. Default objects in jsp
  3. 3. Tag lib for form validation
  4. 4.when jsp page called

Sql

* 1. 1. How to write stored procedure.

Hibernate

* 1. 1 . In hibernate how we call stored procedure.
  2. 2. How to get employee department wise if we have two table emploee and department and depart column is exist in both the table
  3. 5. Diffrencfe between sendredired and forward method
  4. 2. What is normalization
  5. 2. WHAT IS NORMALIZATION AND HOW YOU HAVE USED IN UR PROJECT.

Capgemini

1.WAP for singleton and doubleton.

2.lisiterator list of methods.

3.WAP to insert employee data into HashMap when age>45 using iterator.

4. what is deep and shallow cloning

Suppose two class

Class A

{

Int a ,b;

}

Class Animal

{

Dog g;

Cat c;

}

* 1. Tell how many object will create when

Student sd=new Student(‘c’);

Heap----------------------SP

* 1. WA java code to make immutable class

**Cognizant**

Q. In how many way we can create the object of an class.

Q. What is thread and heap dump and what is the difference.

Q. How to find thread dump?

Q. How to minimize OutOfMemory error.

Q. Suppose that two table is there Employee and Department

Employee

Depid

Empid

Ename

Deppartment

Deptid

Dname

Dtails

Then find the departmanet details which having dname=’ABC’

Q. Suppose two class

Class A------Singleton

Class B------Prototype

Then how many object will create

Q. Suppose two jar is there and both are configured in pom.xml file and if we want to ignore one dependency in production then how u can do it.

Q. How to build the project using maven and deployed in production.

Q. What is the use of springboot and how it works.

Q. What is the use of stream api and how it works with collection

Q. Functional interface and it used in ur project.

Q. lambda expression use and use for addition of two numbers.

Q. How to define rest and post

Q. Can we access a class private variable in another class.

Q. How you did maven packaging(Means build the applicationand deployed into server)

Information technology

Q. WAP to count each charator of string in a word

For example

String s1=”shamim”

s---1

h—1

a---1

m—2

i—1

Q. WAP to get common word from two string

String s1=”I am shamim here”;

String s2=”here shamim is that”;

Common word is ‘shamim’,’here’

Q. How to take unique records from Map.

If phone number and name is matched then consider as duplicates record.

Q. How to call one resful service to another?

For example: first service is ‘withdraw()’ and second service is ‘deposit()’

Q. What is use of qualifier in spring? And is it use the autowired bytype or byname?

Q. Can we initialize abstract class inside constructor of different class?

Q. How to make class immutable?

Company Name :Flamenco Tech

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

1. Describe about your project.

2. How the struts know where navigation and about module.

3. How HashMap is working and how we can access value.

4. What is MVC Structure and what is the benefit for customer.

5. What is thread and WAP for consumer producer program.

6. Make java class as singleton.

7. Where we can use the final class .

8. Access specifier in java.

9. JSP code for modifying value in sub div.

10. How Action servlet work in Struts.

11. How java is secure.

12. Example of mutable and immutable.

13. Example on OutPutStream.

14. Without creating object how we can access variable and method.

Genpact Interview Questions

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

1.How the tomcat knows about servlet.

2.

Tanavat Technolgies,Kormangala

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

1.Explain about your project.

2.How the unlimited range data we can store in single variable.

3.We have two table emp and dept

EMP

--

Empno Empname Salary Deptno

101 Sham 20000 null

102 Sunny 30000 null

103 mohan 40000 D123

104 Nayan 23000 D122

Dept

-----

Deptno Name

D123 Mohan

D122 Nayan

Find Dept who has maximum employee

Find employee which is not belongs to any department.

4.How many types of query hibernate can understand.

5.Write simple query in hibernate.

6.List of method of Object class.

7.Autoboxing and Unboxing

8.Wrapper classes and its use.

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

Design pattern and how used with thread.

Web services :Which framework u have used.

How can we remove duplicate from list.

Methods of Object class and what is the difference between equal and hashcode.

How u can make the loose coupling.

How will you make immutable class/methods.

Have you got any error while using collection.

Which jdk are u using and explain some of java 8 feauters

Difference between iterator and listIterator

What is difference between fail-fast and fail-safe?

How to avoid ConcurrentModificationException while iterating a collection?

How can we get all key from Map

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

comany name: mphasis on 08 sept 2018

1. in a class two overloaded method is there then what is the o/ph

class A{

public a(string a)

{

}

public a(integer a)

{

}

psvm()

{

A a=new A();

a.a(null);

}

}

}

}

2. What is executer framework?

3. difference between creating an object using string literal and using new operator

Spring

-------

1. If two class is there and if we define scope of class A is 'singleton' and another one is prototype then how many object will

be creating.

hibernate

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

1. types of cascading

SGS Global telephonic interview

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

1. Explain about ur carrier.

2. What is Enumarator and in which situation will use?

3. Internal architecture of SET and explain why its not accepting duplicate.

4. Weak,Strong and ... access specifier

5. Super class of exception and what is the Error and give some example of checked exception.

6. LinkedList real scenario use.

7. Can java support multilabel inheritance?

8. Explain about thread schedular and how its work?

9. way of implementing thread.

10. iterator and list iterator and how we can iterate the record on hashmap.

11. difference between system.gc() and calling finalize method.

12. why wait and notify method is static?

13. is there any limition to create the thread?

Spring

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

1. what is IOC and give example to inject static method.

2. what is minimum configuration is required for spring configuration

3. types of annotation for configuring controller servlet?

web services

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

1. which APi you have used for Restful.

2. Difference between put and Delete?

3. explain about restful annotation

4.

hibernate

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