# How to reverse a String

Write a Java program to find the first non-repeated character in a String is a common question on coding tests. Since String is a popular topic in various programming interviews, It's better to prepare well with some well-known questions like [reversing String using recursion](http://javarevisited.blogspot.sg/2012/01/how-to-reverse-string-in-java-using.html), or checking if a String is a palindrome or not. This question is also in the same league. Before jumping into solution, let's first understand this question. You need to write a function, which will accept a String and return first non-repeated character, for example in the world "hello", except 'l' all are non-repeated, but 'h' is the first non-repeated character. Similarly, in word "swiss" 'w' is the first non-repeated character. One way to solve this problem is creating a table to store count of each character, and then picking the first entry which is not repeated. The key thing to remember is order, *your code must return first non-repeated letter*.  
  
By the way, In this article, we will see 3 examples to find the first non-repeated character from a String. Our first solution uses [LinkedHashMap](http://javarevisited.blogspot.sg/2012/12/how-to-sort-hashmap-java-by-key-and-value.html) to store character count since LinkedHashMap maintains insertion order and we are inserting character in the order they appear in String, once we scanned String, we just need to iterate through LinkedHashMap and choose the entry with value 1. Yes, this solution require one LinkedHashMap and two for loops.

Our second solution is a trade-off between time and space, to find first non repeated character in one pass. This time, we have used one Set and one List to keep repeating and non-repeating character separately. Once we finish scanning through String, which is O(n), we can get the magic character by accessing List which is O(1) operator. Since List is an ordered collection get(0) returns first element.  
  
Our third solution is also similar, but this time we have used HashMap instead of LinkedHashMap and we loop through String again to find first non-repeated character. In next section, we will the code example and unit test for this programming question. You can also see my list of [String interview Questions](http://javarevisited.blogspot.sg/2012/10/10-java-string-interview-question-answers-top.html) for more of such problems and questions from Java programming language.

**How to find First Non-Repeated Character from String**

[How to find first non-repeated character in a String in Java](http://1.bp.blogspot.com/-IxSm_itJ4sg/Tg6ZejT0xII/AAAAAAAAAMY/9PHh0EZmiaY/s1600/java_logo_50_50.jpg)Here is the full code sample of finding first duplicate character in a given String. This program has three method to *find first non-repeated character*. Each uses their own algorithm to do this programming task. First algorithm is implemented in getFirstNonRepeatedChar(String str) method. It first gets character array from given [String](http://javarevisited.blogspot.sg/2013/07/java-string-tutorial-and-examples-beginners-programming.html) and loop through it to build a hash table with character as key and their count as value. In next step, It loop through LinkedHashMap to find an entry with value 1, that's your first non-repeated character, because LinkedHashMap maintains insertion order, and we iterate through character array from beginning to end. Bad part is it requires two iteration, first one is proportional to number of character in String, and second is proportional to number of duplicate characters in String. In worst case, where String contains non-repeated character at end, it will take 2\*N time to solve this problem.  
  
Second way to find first non-repeated or unique character is coded on firstNonRepeatingChar(String word) ,this solution finds first non repeated character in a String in just one pass. It applies classical space-time trade-off technique. It uses two storage to cut down one iteration, standard space vs time trade-off. Since we store repeated and non-repeated characters separately, at the end of iteration, first element from List is our first non repeated character from String. This one is slightly better than previous one, though it's your choice to return null or [empty string](http://javarevisited.blogspot.sg/2013/02/5-ways-to-check-if-string-is-empty-in-java-examples.html) if there is no non-repeated character in the String. Third way to solve this programming question is implemented in  firstNonRepeatedCharacter(String word) method. It's very similar to first one except the fact that instead of LinkedHashMap, we have used HashMap. Since later doesn't guarantee any order, we have to rely on original String for *finding first non repeated character*. Here is the algorithm of this third solution. First step : Scan String and store count of each character in [HashMap](http://javarevisited.blogspot.sg/2011/02/how-hashmap-works-in-java.html). Second Step : traverse String and get count for each character from Map. Since we are going through String from first to last character, when count for any character is 1, we break, it's the first non repeated character. Here order is achieved by going through String again.

**import** **java.io.IOException**;

**import** **java.util.ArrayList**;

**import** **java.util.HashMap**;

**import** **java.util.HashSet**;

**import** **java.util.LinkedHashMap**;

**import** **java.util.List**;

**import** **java.util.Map**;

**import** **java.util.Map.Entry**;

**import** **java.util.Set**;

/\*\*

\* Java Program to find first duplicate, non-repeated character in a String.

\* It demonstrate three simple example to do this programming problem.

\*

\* @author Javarevisited

\*/

**public** **class** **Programming** {

/\*

\* Using LinkedHashMap to find first non repeated character of String

\* Algorithm :

\* Step 1: get character array and loop through it to build a

\* hash table with char and their count.

\* Step 2: loop through LinkedHashMap to find an entry with

\* value 1, that's your first non-repeated character,

\* as LinkedHashMap maintains insertion order.

\*/

**public** **static** **char** **getFirstNonRepeatedChar**(String str) {

Map<Character,Integer> counts = **new** LinkedHashMap<>(str.length());

**for** (**char** c : str.toCharArray()) {

counts.put(c, counts.containsKey(c) ? counts.get(c) + **1** : **1**);

}

**for** (Entry<Character,Integer> entry : counts.entrySet()) {

**if** (entry.getValue() == **1**) {

**return** entry.getKey();

}

}

**throw** **new** **RuntimeException**("didn't find any non repeated Character");

}

/\*

\* Finds first non repeated character in a String in just one pass.

\* It uses two storage to cut down one iteration, standard space vs time

\* trade-off.Since we store repeated and non-repeated character separately,

\* at the end of iteration, first element from List is our first non

\* repeated character from String.

\*/

**public** **static** **char** **firstNonRepeatingChar**(String word) {

Set<Character> repeating = **new** HashSet<>();

List<Character> nonRepeating = **new** ArrayList<>();

**for** (**int** i = **0**; i < word.length(); i++) {

**char** letter = word.charAt(i);

**if** (repeating.contains(letter)) {

**continue**;

}

**if** (nonRepeating.contains(letter)) {

nonRepeating.remove((Character) letter);

repeating.add(letter);

} **else** {

nonRepeating.add(letter);

}

}

**return** nonRepeating.get(**0**);

}

/\*

\* Using HashMap to find first non-repeated character from String in Java.

\* Algorithm :

\* Step 1 : Scan String and store count of each character in HashMap

\* Step 2 : traverse String and get count for each character from Map.

\* Since we are going through String from first to last character,

\* when count for any character is 1, we break, it's the first

\* non repeated character. Here order is achieved by going

\* through String again.

\*/

**public** **static** **char** **firstNonRepeatedCharacter**(String word) {

HashMap<Character,Integer> scoreboard = **new** HashMap<>();

// build table [char -> count]

**for** (**int** i = **0**; i < word.length(); i++) {

**char** c = word.charAt(i);

**if** (scoreboard.containsKey(c)) {

scoreboard.put(c, scoreboard.get(c) + **1**);

} **else** {

scoreboard.put(c, **1**);

}

}

// since HashMap doesn't maintain order, going through string again

**for** (**int** i = **0**; i < word.length(); i++) {

**char** c = word.charAt(i);

**if** (scoreboard.get(c) == **1**) {

**return** c;

}

}

**throw** **new** **RuntimeException**("Undefined behaviour");

}

}

**JUnit Test to find First Unique Character**

Here are some [JUnit test cases](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) to test each of this method. We test different kind of inputs, one which contains duplicates, and other which doesn't contains duplicates. Since program has not defined what to do in case of empty String, null String and what to return if only contains duplicates, you are feel free to do in a way which make sense.

**import** **static** org.junit.Assert.\*;

**import** **org.junit.Test**;

**public** **class** **ProgrammingTest** {

@Test

**public** **void** **testFirstNonRepeatedCharacter**() {

assertEquals('b', Programming.firstNonRepeatedCharacter("abcdefghija"));

assertEquals('h', Programming.firstNonRepeatedCharacter("hello"));

assertEquals('J', Programming.firstNonRepeatedCharacter("Java"));

assertEquals('i', Programming.firstNonRepeatedCharacter("simplest"));

}

@Test

**public** **void** **testFirstNonRepeatingChar**() {

assertEquals('b', Programming.firstNonRepeatingChar("abcdefghija"));

assertEquals('h', Programming.firstNonRepeatingChar("hello"));

assertEquals('J', Programming.firstNonRepeatingChar("Java"));

assertEquals('i', Programming.firstNonRepeatingChar("simplest"));

}

@Test

**public** **void** **testGetFirstNonRepeatedChar**() {

assertEquals('b', Programming.getFirstNonRepeatedChar("abcdefghija"));

assertEquals('h', Programming.getFirstNonRepeatedChar("hello"));

assertEquals('J', Programming.getFirstNonRepeatedChar("Java"));

assertEquals('i', Programming.getFirstNonRepeatedChar("simplest"));

}

}

If you can enhance this test cases to check more scenario, just go for it. There is no better way to impress interviewer then writing detailed, creative test cases, which many programmer can't think of or just don't put effort to come up.  
  
That's all on **How to find first non-repeated character of a String in Java**. We have seen three ways to solve this problem, although they use pretty much similar logic, they are different from each other. This program is also very good for beginners to master Java Collection framework. It gives you an opportunity to explore different Map implementations and understand [difference between HashMap and LinkedHashMap](http://java67.blogspot.sg/2012/08/difference-between-hashmap-and-LinkedHashMap-Java.html) to decide when to use them. By the way, if you know any other way to solve this problem, feel free to share. You can also share your interview experience, If you have faced this question on Interviews.

Read more: <http://javarevisited.blogspot.com/2014/03/3-ways-to-find-first-non-repeated-character-String-programming-problem.html#ixzz4esTPsvHF>

# [How to Count Occurrences of a Character in String - Java Programming Exercise Example](http://javarevisited.blogspot.in/2012/12/how-to-count-occurrence-of-character-in-String.html)

Write a program to count the number of occurrences of a character in String is one of the common [programming interview questions](http://javarevisited.blogspot.sg/2011/06/top-programming-interview-questions.html) not just in Java but also in other programming languages like C or C++. As String in a very popular topic on programming interviews and there are lot of good programming exercise on String like "count number of vowels or consonants in String", "count number of characters in String" , [How to reverse String in Java](http://javarevisited.blogspot.sg/2012/01/how-to-reverse-string-in-java-using.html) using recursion or without using StringBuffer etc, it becomes extremely important to have solid knowledge of String in Java or any other programming language. Though, this question is mostly used to test candidate's coding ability i.e. whether he can convert logic to code or not.  
  
In Interview, most of the time Interviewer will ask you to write a program without using any API method, as Java is very rich and it always some kind of nice method to do the job, But it also important to know rich Java and open source libraries for writing [production quality code](http://javarevisited.blogspot.sg/2011/09/how-to-write-production-quality-code.html).  
  
Anyway, in this question, we will see both API based and non-API based(except few) ways to count a number of occurrences of a character in String on Java.

## **Java program to count occurrences of a character in String**

[How to find number of occurence of character or substring in String in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)In this Java program, we will see a couple of ways to count, how many times a particular character is present in String. First, we'll see Spring framework’s StringUtils class and its [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) countOccurrenceOf(String, character) which takes a String and character and returns occurrence of character into that String.  
  
After that, we will see Apache commons StringUtils class for counting occurrence of a character in String. Apache commons StringUtils provide countMatches() method which can be used to count the occurrence of one character or [substring](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html).  
  
Finally, we will see the most simple way of counting character using standard for loop and Java 5 enhanced for loop. This solution can be extended not just to finding the occurrence of character but also finding occurrences of a substring.  
  
Btw, if you are solving this question as part of your Java interview preparation, you can also check [Cracking the Coding Interview](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20), a collection of 189 programming questions and solutions from various programming job interviews. Your perfect companion for developing coding sense required solving these kinds of problems on interviews.

Now, let's see the Java program to count number of occurrence of any character on String:

**import** org.springframework.util.StringUtils;

/\*\*  
 \* Java program to count the number of occurrence of any character on String.  
 \* @author Javin Paul  
 \*/  
**public** **class** CountCharacters {  
  
    **public** **static** **void** main(**String** args[]) {  
            
        **String** input = "Today is Monday"; *//count number of "a" on this String.*  
        
        *//Using Spring framework StringUtils class for finding occurrence of another String*  
        **int** count = StringUtils.countOccurrencesOf(input, "a");  
        
        **System**.out.println("count of occurrence of character 'a' on String: " +

" Today is Monday' using Spring StringUtils " + count);  
  
        
        *//Using Apache commons lang StringUtils class*  
        **int** number = org.apache.commons.lang.StringUtils.countMatches(input, "a");  
        **System**.out.println("count of character 'a' on String: 'Today is Monday' using commons StringUtils " + number);  
        
        *//counting occurrence of character with loop*  
        **int** charCount = 0;  
        for(**int** i =0 ; i<input.length(); i++){  
            if(input.charAt(i) == 'a'){  
                charCount++;  
            }  
        }  
        **System**.out.println("count of character 'a' on String: 'Today is Monday' using for loop  " + charCount);  
        
        *//a more elegant way of counting occurrence of character in String using foreach loop*  
        
        charCount = 0; *//resetting character count*  
        for(**char** ch: input.toCharArray()){  
            if(ch == 'a'){  
                charCount++;  
            }  
        }       
        **System**.out.println("count of character 'a' on String: 'Today is Monday' using for each loop  " + charCount);  
    }  
    
          
}  
  
**Output**  
count of occurrence of character 'a' on **String**: 'Today is Monday' using **Spring** StringUtils 2  
count of character 'a' on **String**: 'Today is Monday' using commons StringUtils 2  
count of character 'a' on **String**: 'Today is Monday' using for loop  2  
count of character 'a' on **String**: 'Today is Monday' using for each loop  2

Well, the beauty of this questions is that Interviewer can twist it on many ways, they can ask you to write a recursive function to count occurrences of a particular character or they can even ask to count how many times each character has appeared.  
  
So if a String contains multiple characters and you need to store count of each character, consider using [HashMap](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html) for storing character as key and number of occurrence as value. Though there are other ways of doing it as well but I like the HashMap way of counting character for simplicity.

Read more: <http://javarevisited.blogspot.com/2012/12/how-to-count-occurrence-of-character-in-String.html#ixzz4esTbHRUa>

# [How to check if two String are Anagram in Java - Program Example](http://javarevisited.blogspot.in/2013/03/Anagram-how-to-check-if-two-string-are-anagrams-example-tutorial.html)

Write a Java program to check if two String are anagram of each other, is another good coding question asked at fresher level Java Interviews. This question is on similar level of [finding middle element of LinkedList in one pass](http://javarevisited.blogspot.com/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html) and [swapping two numbers without using temp variable](http://javarevisited.blogspot.sg/2013/02/swap-two-numbers-without-third-temp-variable-java-program-example-tutorial.html). By the way two String are called anagram, if they contains same characters but on different order e.g. army and mary, stop and pots etc. Anagrams are actually mix-up of characters in String. If you are familiar with String API, i.e. java.lang.String than you can easily solve this problem. In order to check if Strings are anagram, you need to get there character array and see if they are equal or not. Though you can also use indexOf(), substring() and [StringBuffer or StringBuilder](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) class to solve this question. In this Java program, we will see 3 ways to solve this interview questions, and check if two String are anagram or not. By the way, if you are preparing for Java interview, it's good to prepare some [data structures and algorithms questions](http://javarevisited.blogspot.sg/2013/03/top-15-data-structures-algorithm-interview-questions-answers-java-programming.html) as well. More often, there is one or more questions from programming, coding and logic in these interviews.

## **Java program to check if String is anagram**

[String Anagram Check - 3 ways to find if two Strings are anagrams or not](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)As I said, there are multiple ways to find if two string are anagram or not. Classical way is getting character array of each String, and then comparing them, if both char array is equal then Strings are anagram. But before comparing, make sure that both String are in same case e.g. lowercase or uppercase and character arrays are sorted, because [equals method of Arrays](http://javarevisited.blogspot.com/2012/12/how-to-compare-arrays-in-java-equals-deepequals-primitive-object.html), return true, only if array contains same length, and each index has same character.

For simplicity, I have left checking if String is null or empty and converting them into uppercase or lowercase, you can do that if you want. If Interviewer ask you to write [production quality code](http://javarevisited.blogspot.com/2011/09/how-to-write-production-quality-code.html), then I would suggest definitely put those checks and throw IllegalArgumentException for null String or you can simply return false. I would personally prefer to return false rather than throwing Exception, similar to [equals() method](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html). Anyway, here are *three ways to check if two String are Anagram* or not. I have also included a JUnit Test to verify various String which contains both anagram and not.

import java.util.Arrays;

*/\*\**

*\* Java program - String Anagram Example.*

*\* This program checks if two Strings are anagrams or not*

*\**

*\* @author Javin Paul*

*\*/*

public class AnagramCheck {

*/\**

*\* One way to find if two Strings are anagram in Java. This method*

*\* assumes both arguments are not null and in lowercase.*

*\**

*\* @return true, if both String are anagram*

*\*/*

public static boolean isAnagram(String word, String anagram){

if(word.length() != anagram.length()){

return false;

}

char[] chars = word.toCharArray();

for(char c : chars){

int index = anagram.indexOf(c);

if(index != -1){

anagram = anagram.substring(0,index) + anagram.substring(index +1, anagram.length());

}else{

return false;

}

}

return anagram.isEmpty();

}

*/\**

*\* Another way to check if two Strings are anagram or not in Java*

*\* This method assumes that both word and anagram are not null and lowercase*

*\* @return true, if both Strings are anagram.*

*\*/*

public static boolean iAnagram(String word, String anagram){

char[] charFromWord = word.toCharArray();

char[] charFromAnagram = anagram.toCharArray();

Arrays.sort(charFromWord);

Arrays.sort(charFromAnagram);

return Arrays.equals(charFromWord, charFromAnagram);

}

public static boolean checkAnagram(String first, String second){

char[] characters = first.toCharArray();

StringBuilder sbSecond = new StringBuilder(second);

for(char ch : characters){

int index = sbSecond.indexOf("" + ch);

if(index != -1){

sbSecond.deleteCharAt(index);

}else{

return false;

}

}

return sbSecond.length()==0 ? true : false;

}

}

## **JUnit Test Case for String Anagram Exmaple**

here is our [JUnit tests](http://javarevisited.blogspot.com/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) for all three 3 methods of AnagramCheck class, we have actually tested all method with similar set of input.

import org.junit.Test;

import static org.junit.Assert.\*;

*/\*\**

*\* JUnit test class to test various anagram program for various String input.*

*\*/*

public class StringAnagramTest {

@Test

public void testIsAnagram() {

assertTrue(AnagramCheck.isAnagram("word", "wrdo"));

assertTrue(AnagramCheck.isAnagram("mary", "army"));

assertTrue(AnagramCheck.isAnagram("stop", "tops"));

assertTrue(AnagramCheck.isAnagram("boat", "btoa"));

assertFalse(AnagramCheck.isAnagram("pure", "in"));

assertFalse(AnagramCheck.isAnagram("fill", "fil"));

assertFalse(AnagramCheck.isAnagram("b", "bbb"));

assertFalse(AnagramCheck.isAnagram("ccc", "ccccccc"));

assertTrue(AnagramCheck.isAnagram("a", "a"));

assertFalse(AnagramCheck.isAnagram("sleep", "slep"));

}

@Test

public void testIAnagram() {

assertTrue(AnagramCheck.iAnagram("word", "wrdo"));

assertTrue(AnagramCheck.iAnagram("boat", "btoa"));

assertFalse(AnagramCheck.iAnagram("pure", "in"));

assertFalse(AnagramCheck.iAnagram("fill", "fil"));

assertTrue(AnagramCheck.iAnagram("a", "a"));

assertFalse(AnagramCheck.iAnagram("b", "bbb"));

assertFalse(AnagramCheck.iAnagram("ccc", "ccccccc"));

assertFalse(AnagramCheck.iAnagram("sleep", "slep"));

}

@Test

public void testcheckAnagram() {

assertTrue(AnagramCheck.checkAnagram("word", "wrdo"));

assertFalse(AnagramCheck.checkAnagram("b", "bbb"));

assertFalse(AnagramCheck.checkAnagram("ccc", "ccccccc"));

assertTrue(AnagramCheck.checkAnagram("a", "a"));

assertFalse(AnagramCheck.checkAnagram("sleep", "slep"));

assertTrue(AnagramCheck.checkAnagram("boat", "btoa"));

assertFalse(AnagramCheck.checkAnagram("pure", "in"));

assertFalse(AnagramCheck.checkAnagram("fill", "fil"));

}

}

Output

Testsuite: StringAnagramTest

Tests run: 3, Failures: 0, Errors: 0, Time elapsed: 0.094 sec

Our AnagramCheck class contains 3 [static methods](http://javarevisited.blogspot.com/2013/03/can-we-overload-and-override-static-method-java.html) to verify if Strings are anagram or not. First one, takes character array of first String and loop through it, then finds that character in second String, and deletes it by using [substring method](http://javarevisited.blogspot.com/2011/10/how-substring-in-java-works.html). If second String doesn't contains character than method return false immediately. At the end of test if second String is empty than both Strings are anagram because they contains same set of characters. To improve performance, we have checked length at very start of this method, as two String with different length can not be anagram of each other. Third method is exactly same of first one, except, it uses deleteCharAt(int index) method of [StringBuilder](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) for deleting characters.

Read more: <http://javarevisited.blogspot.com/2013/03/Anagram-how-to-check-if-two-string-are-anagrams-example-tutorial.html#ixzz4esTrGiWB>

# [How to Convert String to Integer to String in Java with Example](http://javarevisited.blogspot.in/2011/08/convert-string-to-integer-to-string.html)

Converting **String to integer** and **Integer to String** is one of the basic tasks of Java and most people learned about it when they learn Java programming. Even though *String to integer* and *Integer to String* conversion is basic stuff but same time its most useful also because of its frequent need given that String and Integer are two most widely used type in all sort of program and you often gets data between any of these formats. One of the common tasks of programming is converting one data type another e.g. [Converting Enum to String](http://javarevisited.blogspot.sg/2011/12/convert-enum-string-java-example.html) or [Converting Double to String](http://javarevisited.blogspot.sg/2011/10/convert-double-to-string-example.html), Which is similar to converting String to Integer. Some programmers asked a question that why not Autoboxing can be used to Convert String to int primitive or Integer Object?   
  
  
Remember autoboxing only converts primitive to Object it doesn't convert one data type to other. Few days back I had to convert a binary String into integer number and then I thought about this post to document all the way I know to convert Integer to String Object and String to Integer object.   
  
Here is my way of converting **String to Integer in Java with example** :

## String to Integer Conversion in Java

here are four different ways of converting String object into int or Integer in Java :

**1) By using Intger.parseInt(String string-to-be-converted) method**  
This is my preferred way of converting an String to int in Java, Extremely easy and most flexible way of converting String to Integer. Let see an example of String to int conversion:

 //using Integer.parseInt

**int** i = Integer.parseInt("123");

 System.out.println("i: " + i);

Integer.parseInt() method will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) NumberFormatException if String provided is not a proper number. Same technique can be used to convert other data type like float and Double to String in Java. Java API provides static methods like Float.parseFloat() and Double.parseDouble() to perform data type conversion.  
  
  
  
**2) Integer.valueOf() method**  
There is another way of converting String into Integer which was hidden to me for long time mostly because I was satisfied with Integer.parseInt() method. This is an example of [Factory method design pattern in Java](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html) and known as Integer.valueOf(), this is also a static method like main and can be used as utility for string to int conversion. Let’s see an example of using Integer.valueOf() to convert String into int in java.

//How to convert numeric string = "000000081" into Integer value = 81

**int** i = Integer.parseInt("000000081");

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

It will *ignore the leading zeros and convert the string into int*. This method also throws **NumberFormatException** if string provided does not represent actual number. Another interesting point about static valueOf() method is that it is used to create instance of wrapper class during [Autoboxing in Java](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html) and can cause subtle [issues while comparing primitive to Objec](http://javarevisited.blogspot.sg/2010/10/what-is-problem-while-using-in.html)t e.g. int to Integer using equality operator (==),  because caches Integer instance in the range -128 to 127.

### How to convert Integer to String in Java

[String to integer conversion, Int to string example](http://javarevisited.blogspot.com/2011/06/noclassdeffounderror-exception-in.html)In the previous example of this String to int  conversion  we have seen changing String value into int primitive type and this part of Java tutorial we will see opposite i.e. **conversion of Integer Object to String**. In my opinion this is simpler than previous one. You can simply concatenate any number with empty String and it will create a new String. Under the carpet + operator uses either StringBuffer or StringBuilder to concatenate String in Java. anyway there are couple of more ways to convert int into String and we will see those here with examples.  
  
**Int to String in Java using "+" operator**  
Anything could not be more easy and simple than this. You don't have to do anything special just use"+" concatenation operator with String to convert int variable into String object as shown in the following example:

String price = "" + **123**;

Simplicity aside, Using String concatenation for converting int to String is also one of the most poor way of doing it. I know it's temptation because, it's also the most easiest way to do and that's the main reason of it polluting code. When you write code like "" + 10 to convert numeric 10 to String, your code is translated into following :

**new** **StringBuilder**().append( "" ).append( **10** ).toString();

StringBuilder(String) constructor allocates a buffer containing 16 characters. So , appending upto to 16 characters to that StringBuilder will not require buffer reallocation, but appending more than 16 character will expand StringBuider buffer. Though it's not going to happen because Integer.MAX\_VALUE is 2147483647, which is less than 16 character. At the end, StringBuilder.toString() will create a new String object with a copy of StringBuider buffer. This means for converting a single integer value to String you will need to allocate: one StringBuilder, one char array char[16], one String and one char[] of appropriate size to fit your input value. If you use String.vauleOf() will not only benefit from cached set of values but also you will at least avoid creating a StringBuilder. To learn more about these two, see my post [StringBuffer vs StringBuilder vs String](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html)   
  
  
  
**One more example of converting int to String**  
There are many ways to convert an int variable into String,  In case if you don't like above example of string conversion than here is one more example of doing same thing. In this example of converting Integer to String we have used String.valueOf() method which is another static utility method to convert any integer value to String. In-fact String.valueOf() method is overloaded to accept almost all primitive type so you can use it convert char, double, float or any other data type into String. [Static binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) is used to call corresponding method in Java. Here is an example of int to String using **String.valueOf()**

String price = String.valueOf(**123**);

After execution of above line Integer 123 will be converted into String “123”.   
  
**Int to string using String. format()**  
This is a new way of converting an int primitive to String object and introduced in JDK 1.5 along-with several other important features like [Enum](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html), Generics and Variable argument methods. String.format() is even more powerful and can be used in variety of [way to format String in Java](http://javarevisited.blogspot.sg/2012/08/how-to-format-string-in-java-printf.html). This is just another use case of String.format() method for displaying int as string. Here is an example of converting int to String using String.format method:

String price = String.format ("%d", **123**);

Indeed conversion of String to Integer object or int primitive to String is pretty basic stuff but I thought let's document for quick reference of anyone who might forget it or just wanted to refresh it. By the way *if you know any other way of string-int-string conversion* than please let us know and I will include it here.

Read more: <http://javarevisited.blogspot.com/2011/08/convert-string-to-integer-to-string.html#ixzz4esU9c3OO>

# [String vs StringBuffer vs StringBuilder in Java](http://javarevisited.blogspot.in/2011/07/string-vs-stringbuffer-vs-stringbuilder.html)

**Difference between String, StringBuffer, and StringBuilder**

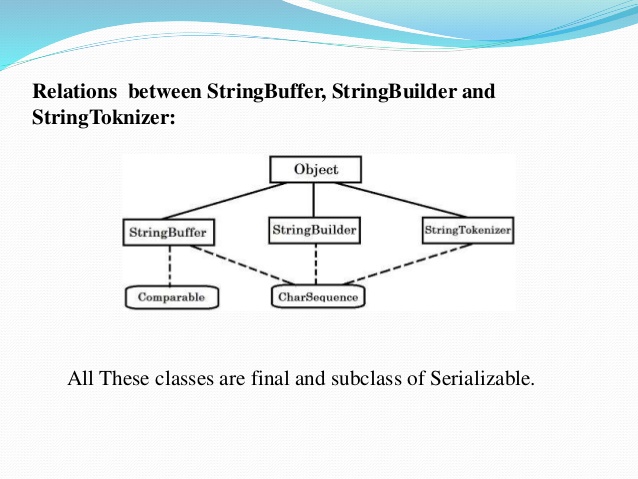
The String is one of the most important classes in Java and anyone who starts with Java programming uses String to print something on the console by using famous System.out.println() statements. Many Java beginners not aware that [String is immutable and final in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html) and every modification in String creates a new String object. For example, when you get the substring, you get a new String, when you convert uppercase String to lowercase, a new String is created. Even when you remove space by calling the trim() method, a new String is returned. So, now the big question is how do you manipulate String in Java without creating String garbage? StringBuilder and StringBuffer are the answer to this question. StringBuffer is old class but StringBuilder is newly added in Java 5 along with major improvements in [Enum](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html), [Generics](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html), [varargs methods](http://javarevisited.blogspot.sg/2011/09/variable-argument-in-java5-varargs.html) and [Autoboxing in Java](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html).  
  
No matter which kind of application you are working you will find the heavy usage of Java String class but if you do profiling of your application you will find that String is the one class which creates lots of garbage because of much temporary String created in the program.  
  
In this Java tutorial we will see What is String in Java, some important properties of String in Java, What is StringBuffer in Java , When to use StringBuffer in Java , StringBuilder in Java and how it can be used in place of StringBuffer, What are differences between String and StringBuffer and StringBuilder in Java  which is also a [frequently asked core Java question](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html) and mostly String vs StringBuilder vs StringBuffer.  
  
Now let's start with String.

## **Differences between String, StringBuffer, and StringBuilder in Java**

### String in Java

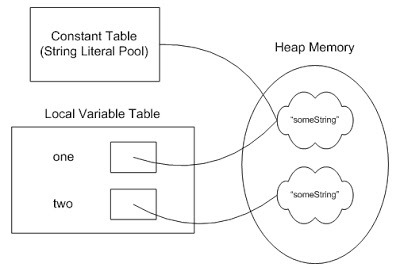
[string and stringbuffer, string vs stringbuffer vs stringbuilder](http://javarevisited.blogspot.com/2010/10/what-is-difference-between-enumeration.html)Before looking *difference between String and StringBuffer or StringBuilder* let’s see some fundamental properties of String Class in Java  
  
1) The string is immutable in Java:  String is by design immutable in Java you can check [this post](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) for a reason. Immutability offers a lot of benefit to the String class e.g. his hashcode value can be cached which makes it a faster hashmap key and one of the reasons [why String is a popular key in HashMap](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html). Because String is final it can be safely shared between multiple threads  without any extra synchronization.  
  
  
2) When we represent a string in double quotes like "abcd" they are referred as String literal and String literals are created in String pools. When you [compare two String literals](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html) using equality operator "==" it returns true because they are the actually same instance of String. Anyway comparing an object with equality operator is bad practice in Java and you should always use the [equals method](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) to check equality.  
  
  
3) The **"+"** operator is overloaded for String and used to concatenated two string. Internally "+" operation is implemented using either StringBuffer or StringBuilder. See [Core Java for Impatient](http://www.amazon.com/Core-Java-Impatient-Cay-Horstmann/dp/0321996321?tag=javamysqlanta-20) for more details on String concatenation in Java.

4) Strings are backed up by character Array and represented in UTF-16 format. By the way, this behavior can cause a memory leak in String because same character array is shared between source String and SubString which can prevent source String from being garbage collected. See [How SubString works in Java](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html) for more details.  
  
  
5) String class overrides equals() and hashcode() method and two Strings are considered to be equal if they contain exactly same character in same order and in same case. If you want to ignore case comparison of two strings consider using equalsIgnoreCase() method. See  [how to correctly override equals method in Java](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html)  to learn more about best practices on equals method. Another worth noting point is that equals method must be consistent with compareTo() method for String because SortedSet and SortedMap e.g. TreeMap use the [compareTo method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) to compare String in Java.  
  
7) toString() method provides String representation of any object and it's declared in Object class and it's recommended for other class to implement this and provide String representation.  
  
8) The string is represented using the UTF-16 format in Java.  
  
9) In Java, you can create [String from char array](http://javarevisited.blogspot.sg/2012/02/how-to-convert-char-to-string-in-java.html), byte array, another string, from StringBuffer or from StringBuilder. Java String class provides a constructor for all of these.  
  
10) Even though all StringBuffer, StringBuilder, and String are from same type hierarchy i.e. they extend from CharSequence interface, you cannot cast StringBuilder to StirngBuffer or StringBuilder to String in Java. It will throw java.lang.ClasscastException, if you tried to cast even StringBuffer to String in Java.  
  
Here is a nice diagram which shows the relationship between StringBuffer, StringBuilder, and String in Java:

[](https://3.bp.blogspot.com/--S7jbJe0JF0/VvT4DrNJPHI/AAAAAAAAFWg/AlpmiNiljXQ20Vpnj0ZjvwoTVYinC_OgA/s1600/String%2Bvs%2BStringBuffer%2Bvs%2BStringBuilder%2Bin%2BJava.jpg)

### Problem with String in Java

One of its biggest strength Immutability is also biggest problem of Java String if not used correctly. Many times we create a String and then perform a lot of operation on them e.g. converting a string into uppercase, lowercase , getting [substring](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html) out of it , concatenating with other string etc. Since String is an immutable class every time a new String is created and the older one is discarded which creates lots of temporary garbage in the heap.  
  
If String is created using [String literal](http://java67.blogspot.com/2014/08/difference-between-string-literal-and-new-String-object-Java.html) they remain in the String pool. To resolve this problem Java provides us, two Classes, **StringBuffer and StringBuilder**. String Buffer is an older class but StringBuilder is relatively new and added in JDK 5.

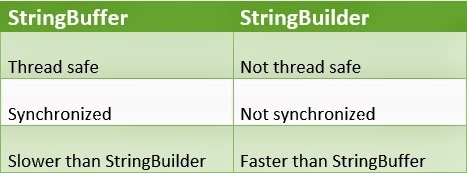
[](http://javarevisited.blogspot.com/2015/12/when-to-use-intern-method-of-string-in-java.html)

### Differences between String and StringBuffer in Java

The main *difference between String and StringBuffer* is String is immutable while StringBuffer is mutable means you can modify a StringBuffer object once you created it without creating any new object. This mutable property makes StringBuffer an ideal choice for dealing with Strings in Java.  
  
You can convert a StringBuffer into String by its toString() method. String vs StringBuffer or what is the difference between StringBuffer and String is one of the [popular Java interview questions](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html) for either phone interview or first round. Nowadays they also include StringBuilder and ask String vs StringBuffer vs StringBuilder.  
  
So be preparing for that. In the next section, we will see the difference between StringBuffer and StringBuilder in Java. If you are preparing for Java interviews, then you can also check [Java Programming Interview exposed](http://www.amazon.com/Java-Programming-Interviews-Exposed-Markham/dp/1118722868?tag=javamysqlanta-20) for more such questions, one of the best book, which covers all important topics for Java interviews.

### Difference between StringBuilder and StringBuffer in Java

**StringBuffer** is very good with mutable String but it has one disadvantage all its public methods are [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) which makes it [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) but same time slow. In JDK 5 they provided a similar class called StringBuilder in Java which is a copy of StringBuffer but without synchronization. Try to use ***StringBuilder*** whenever possible it performs better in most of the cases than StringBuffer class.  
  
You can also use "+" for concatenating two string because "+" operation is internally implemented using either **StringBuffer or StringBuilder in Java**. If you see StringBuilder vs StringBuffer you will find that they are exactly similar and all API methods applicable to StringBuffer are also applicable to StringBuilder in Java.  
  
On the other hand, [String vs StringBuffer](http://java67.blogspot.com/2012/08/difference-between-string-and-stringbuffer-in-java.html) is completely different and there API is also completely different, same is true for StringBuilder vs String. Here is a nice summary of the difference between StringBuffer and StringBuilder in Java:

[](http://java67.blogspot.com/2014/05/difference-between-stringbuilder-and-StringBuffer-java.html)

### Summary

In summary here is list of difference between StringBuffer, String, and StringBuilder in Java :

1) The String object is immutable in Java but StringBuffer and StringBuilder are mutable objects.

2) StringBuffer is synchronizedwhile StringBuilder is not which makes StringBuilder faster than StringBuffer.

3) Concatenation operator "+" is internally implemented using either StringBuffer or StringBuilder.

4) Use String if you require [immutability](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html), use StringBuffer in java if you need mutable + [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) and use StringBuilder in Java if you require mutable + without thread-safety.

That's all on famous String vs StringBuffer or StringBuffer vs StringBuilder discussion. All these differences help to avoid the common coding mistake of using String in place of StringBuffer in many places. from Java 5 onwards either use + operator of StringBuilder for concatenating String in Java.

Read more: <http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html#ixzz4esUMxRTf>

# [Why String is Immutable or Final in Java](http://javarevisited.blogspot.in/2010/10/why-string-is-immutable-or-final-in-java.html)

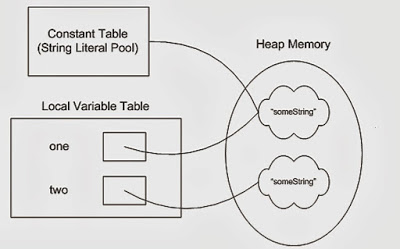
The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, [*String was made final*](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc by extending and overriding behaviors. Another reason of *why String class is immutable* could die due to HashMap.

Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since [HashMap works in the principle of hashing](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.  
  
If you are an Indian cricket fan, you may be able to correlate with my next sentence. The string is VVS Laxman of Java, i.e. very very special class. I have not seen a single Java program which is written without using String. That's why a solid understanding of String is very important for a Java developer.

Important and popularity of String as data type, transfer object and mediator has also made it popular in Java interviews. *Why String is immutable in Java* is one of the most frequently asked [String Interview questions in Java](http://javarevisited.blogspot.com/2012/10/10-java-string-interview-question-answers-top.html), which starts with discussion of,  what is String, how String in Java is different than String in C and C++, and then shifted towards what is immutable object in Java , what are the benefits of immutable object, why do you use them and which scenarios should you use them. This question sometimes also asked, *"Why String is final in Java"*.  
  
  
On a similar note, if you are preparing for Java interviews, I would suggest you take a loot at the [Java Programming interview exposed](http://www.amazon.com/Java-Programming-Interviews-Exposed-Markham/dp/1118722868?tag=javamysqlanta-20) book, an excellent resource for senior and mid-level Java programmer. It contains questions from all important Java topic including multi-threading, collection, GC, JVM internals and framework like Spring and Hibernate, as shown below:

## Why String is Final in Java

As I said, there could be many possible answers to this question, and the only designer of String class can answer it with confidence. I was expecting some clue in Joshua Bloch's [Effective Java](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) book, but he also didn't mention it. I think following two reasons make a lot of sense on why String class is made Immutable or final in Java: 1) Imagine String pool facility without making string immutable , its not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many reference variables, so if any one of them change the value others will be automatically gets affected i.e. lets say  
  
String A = "Test"  
String B = "Test"  
  
Now String B called, "Test".toUpperCase() which change the same object into "TEST", so A will also be "TEST" which is not desirable. Here is a nice diagram which shows how String literals are created in heap memory and String literal pool.

[](http://1.bp.blogspot.com/-4AB58TuuluE/VmLVMNKK4zI/AAAAAAAAENo/Bbkx5wwXlRA/s1600/String%2Bin%2BJava%2Bis%2BImmutable.jpg)

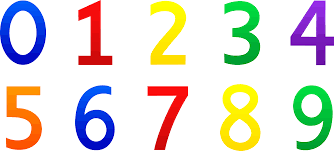
2) String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string, you can pass database URL as a string for opening database connection, you can [open any file in Java](http://javarevisited.blogspot.sg/2012/07/read-file-line-by-line-java-example-scanner.html) by passing the name of the file as argument to File I/O classes.  
  
In case, if String is not immutable, this would lead serious security threat, I mean someone can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access to that file. Because of immutability, you don't need to worry about that kind of threats. This reason also gels with, **Why String is final in Java**, by making java.lang.String final, Java designer ensured that no one overrides any behavior of String class.  
3)Since String is immutable it can safely share between many threads which is very important for multithreaded programming and to avoid any synchronization issues in Java, Immutability also makes String instance thread-safe in Java, means you don't need to synchronize String operation externally. Another important point to note about String is the [memory leak caused by SubString](http://javarevisited.blogspot.sg/2011/10/how-substring-in-java-works.html), which is not a thread related issues but something to be aware of.  
  
  
4) Another reason of **Why String is immutable in Java** is to allow String to cache its hashcode, being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in hashmap in Java.  This one is also suggested by  Jaroslav Sedlacek in comments below. In short because String is immutable, no one can change its contents once created which guarantees hashCode of String to be same on multiple invocations.  
  
  
5) Another good reason of Why String is immutable in Java suggested by Dan Bergh Johnsson on comments is: The absolutely most important reason that String is immutable is that it is used by the [class loading mechanism](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter"  
  
  
Security and String pool being primary reason of making String immutable, I believe there could be some more very convincing reasons as well, Please post those reasons as comments and I will include those on this post. By the way, above reason holds good to answer, another [Java interview questions](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html) **"Why String is final in Java".** Also to be immutable you have to be final so that your subclass doesn't break immutability.  what do you guys think?

Read more: <http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-or-final-in-java.html#ixzz4esUa9wc1>

# [How to Find Missing Number on Integer Array of 1 to 100 - BitSet Example](http://javarevisited.blogspot.in/2014/11/how-to-find-missing-number-on-integer-array-java.html)

One of the most frequently asked question on programming interviews is, write a program to find the missing number in an array in Java, C# or any other language; depending upon which language you choose. This kind of [coding interview questions](http://javarevisited.blogspot.com/2011/06/top-programming-interview-questions.html) are not only asked in small start-ups but also on some of the biggest technical companies like Google, Amazon, Facebook, Microsoft, mostly when they visit the campus of reputed universities to hire graduates. Simplest version of this question is to *find missing elements in an area of 100 integers*, which contains numbers between 1 and 100. This can easily be solved by calculating the sum of the series using n(n+1)/2, and this is also one of the quickest and efficient ways, but it cannot be used if the array contains more than one missing numbers or if the [array contains duplicates](http://javarevisited.blogspot.com/2015/06/3-ways-to-find-duplicate-elements-in-array-java.html).  
  
This gives interviewer some nice follow-up questions to check whether a candidate can apply his knowledge of the slightly different condition or not. So if you get through this, they will ask you to find the missing number in an array of duplicates. This might be tricky but you will soon find out that another way to find missing and duplicate number in the array is to sort it.  
  
In a sorted array, you can compare whether a number is equal to expected next number or not. Alternatively, you can also use BitSet in Java to solve this problem.

## Java Program to find missing numbers

[](http://1.bp.blogspot.com/-9YkRrRhx-Zk/VHiQ8rxLLRI/AAAAAAAACIQ/Y9b9NGRfd0I/s1600/missing%2Bnumber%2Bin%2BJava%2Barray.png)

Let's understand the problem statement, we have numbers from 1 to 100 that are put into an integer array, what's the best way to find out which number is missing? If Interviewer especially mentions 1 to 100 then you can apply the above trick about the sum of the series as shown below as well. If it has more than one missing element that you can use BitSet class, of course only if your interviewer allows it.  
  
1) Sum of the series: Formula: n (n+1)/2( but only work for one missing number)  
2) Use BitSet, if an array has more than one missing elements.  
  
I have provided a BitSet solution with another purpose, to introduce with this nice utility class. In many interviews, I have asked about this class to Java developers, but many of them not even aware of this. I think this problem is a nice way to learn how to use BitSet in Java as well.  
  
By the way, if you are going for interview, then apart from this question, its also good to know [how to find duplicate number in array](http://javarevisited.blogspot.com/2012/02/how-to-check-or-detect-duplicate.html) and [program to find second highest number in an integer array](http://java67.blogspot.com/2014/02/how-to-find-largest-and-smallest-number-array-in-java.html). More often than not, those are asked as follow-up question after this.

**import** **java.util.Arrays**;

**import** **java.util.BitSet**;

/\*\*

\* Java program to find missing elements in a Integer array containing

\* numbers from 1 to 100.

\*

\* @author Javin Paul

\*/

**public** **class** **MissingNumberInArray** {

**public** **static** **void** **main**(String args[]) {

// one missing number

printMissingNumber(**new** **int**[]{**1**, **2**, **3**, **4**, **6**}, **6**);

// two missing number

printMissingNumber(**new** **int**[]{**1**, **2**, **3**, **4**, **6**, **7**, **9**, **8**, **10**}, **10**);

// three missing number

printMissingNumber(**new** **int**[]{**1**, **2**, **3**, **4**, **6**, **9**, **8**}, **10**);

// four missing number

printMissingNumber(**new** **int**[]{**1**, **2**, **3**, **4**, **9**, **8**}, **10**);

// Only one missing number in array

**int**[] iArray = **new** **int**[]{**1**, **2**, **3**, **5**};

**int** missing = getMissingNumber(iArray, **5**);

System.out.printf("Missing number in array %s is %d %n",

Arrays.toString(iArray), missing);

}

/\*\*

\* A general method to find missing values from an integer array in Java.

\* This method will work even if array has more than one missing element.

\*/

**private** **static** **void** **printMissingNumber**(**int**[] numbers, **int** count) {

**int** missingCount = count - numbers.length;

BitSet bitSet = **new** BitSet(count);

**for** (**int** number : numbers) {

bitSet.set(number - **1**);

}

System.out.printf("Missing numbers in integer array %s, with total number %d is %n",

Arrays.toString(numbers), count);

**int** lastMissingIndex = **0**;

**for** (**int** i = **0**; i < missingCount; i++) {

lastMissingIndex = bitSet.nextClearBit(lastMissingIndex);

System.out.println(++lastMissingIndex);

}

}

/\*\*

\* Java method to find missing number in array of size n containing

\* numbers from 1 to n only.

\* can be used to find missing elements on integer array of

\* numbers from 1 to 100 or 1 - 1000

\*/

**private** **static** **int** **getMissingNumber**(**int**[] numbers, **int** totalCount) {

**int** expectedSum = totalCount \* ((totalCount + **1**) / **2**);

**int** actualSum = **0**;

**for** (**int** i : numbers) {

actualSum += i;

}

**return** expectedSum - actualSum;

}

}

Output

Missing numbers in integer array [**1**, **2**, **3**, **4**, **6**], with total number **6** is

**5**

Missing numbers in integer array [**1**, **2**, **3**, **4**, **6**, **7**, **9**, **8**, **10**], with total number **10** is

**5**

Missing numbers in integer array [**1**, **2**, **3**, **4**, **6**, **9**, **8**], with total number **10** is

**5**

**7**

**10**

Missing numbers in integer array [**1**, **2**, **3**, **4**, **9**, **8**], with total number **10** is

**5**

**6**

**7**

**10**

Missing number in array [**1**, **2**, **3**, **5**] is **4**

You can see that how using a right data structure can solve the problem easily. This is the key takeaway of this program, for the more coding question, you can check the [Cracking the Coding Interviews](http://www.amazon.com/Cracking-Coding-Interview-6th-Edition/dp/0984782850/?tag=javamysqlanta-20), a collection of 189 coding questions from programming interviews of tech companies like Google, Amazon, Microsoft and others.

That's all on this **program to find missing element in an array of 100 elements**. As I said, it's good to know the trick, which just require you to calculate sum of numbers and then subtract that from actual sum, but you can not use that if array has more than one missing numbers. On the other hand, BitSet solution is more general, as you can use it to find more than one missing values on integer array. For more programming questions, you can also check [here](http://java67.blogspot.com/2012/08/10-java-coding-interview-questions-and.html)

Read more: <http://javarevisited.blogspot.com/2014/11/how-to-find-missing-number-on-integer-array-java.html#ixzz4esUsSF7o>

# [How to check or detect duplicate elements in Array in Java](http://javarevisited.blogspot.in/2012/02/how-to-check-or-detect-duplicate.html)

**Detecting duplicate elements in Java** array is another [programming interview question](http://javarevisited.blogspot.com/2011/06/top-programming-interview-questions.html) I like. There could be a lot of ways you can check if your **array contains duplicate elements** or not and sometimes you discover a unique way of checking duplicates by asking this question on Java interview. Beauty of this question is that it has endless number of follow-up question so if interviewee gets through this question you can ask to him about time complexity and space or to improve his algorithm to make it fast .you can even ask to find those duplicate elements in Array which even can go from one duplicate to many repeating elements in Array. As I said you can really test programming skill around an array of a Java programmer.

## **Checking Array for duplicate elements Java**

In this Java tutorial, we will see a couple of ways to find if an array contains duplicates or not in Java. We will use the unique property of Java collection class Set which doesn’t allow duplicates to check java array for duplicate elements.  Here are five ways we can check if an array has duplicates or not:

1) **brute force method** which compares each element of Array to all other elements and returns true if it founds duplicates. Though this is not an efficient choice it is the one which first comes to mind.

2) Another quick way of checking if a Java array contains duplicates or not is to **convert that array into Set**. Since Set doesn’t allow duplicates size of  the corresponding Set will be smaller than original Array if Array contains duplicates otherwise the size of both Array and Set will be same.

3) One more way to detect duplication in java array is adding every element of the array into HashSet which is a Set implementation. Since the add(Object obj) method of Set returns false if Set already contains an element to be added, it can be used to find out if the array contains duplicates in Java or not.

[How to find or detect duplicate elements in Array in Java](http://javarevisited.blogspot.com/2011/09/servlet-interview-questions-answers.html)In next section, we will complete code example of all three ways of **duplicate detection on Array in java**. Remember this discussion is just confirming whether an array contains duplicate or not , it's not finding out actual duplicate elements from Array though you can easily extend example Java program to accomplish that task based on your requirement.   
  
  
This is also one of the popular programming interviews questions, asked in several interviews. I also suggest you to solves problems from [Cracking the Coding Interview: 189 Programming Questions and Solutions](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20). One of the best book to prepare for software developer interviews.

## **Code Example of checking duplicate on Array in Java**

Here is complete code sample of all above methods to check if your array contains duplicates or not.

**import** java.util.Arrays;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**public** **class** **CheckDuplicatesInJavaArray** {

**public** **static** **void** main(String args[])  {

       String[] withDuplicates = **new** String[] {"one","two","three","one"};

        String[] withoutDuplicates = **new** String[] {"one","two","three"};

        System.*out*.println("Checking array with duplicate using brute force: " + *bruteforce*(withDuplicates));

        System.*out*.println("Checking array without any duplicate using brute force: " + *bruteforce*(withoutDuplicates));

        System.*out*.println("Checking array with duplicate using Set and List: " + *checkDuplicateUsingSet*(withDuplicates));

        System.*out*.println("Checking array without any duplicate using Set and List: " + *checkDuplicateUsingSet*(withoutDuplicates));

        System.*out*.println("Checking array with duplicate using Set and List: " + *checkDuplicateUsingAdd*(withDuplicates));

        System.*out*.println("Checking array without any duplicate using Set and List: " + *checkDuplicateUsingAdd*(withoutDuplicates));

    }

    /\*

     \* brute force way of checking if array contains duplicates in Java

     \* comparing each element to all other elements of array

     \* complexity on order of O(n^2) not advised in production

     \*/

**public** **static** **boolean** bruteforce(String[] input) {

**for** (**int** i = 0; i < input.length; i++) {

**for** (**int** j = 0; j < input.length; j++) {

**if** (input[i].equals(input[j]) && i != j) {

**return** **true**;

                }

            }

        }

**return** **false**;

    }

    /\*

     \* detect duplicate in array by comparing size of List and Set

     \* since Set doesn't contain duplicate, size must be less for an array which contains duplicates

     \*/

**public** **static** **boolean** checkDuplicateUsingSet(String[] input){

        List inputList = Arrays.*asList*(input);

        Set inputSet = **new** HashSet(inputList);

**if**(inputSet.size()< inputList.size())

**return** **true**;

        }

**return** **false**;

    }

    /\*

     \* Since Set doesn't allow duplicates add() return false

     \* if we try to add duplicates into Set and this property

     \* can be used to check if array contains duplicates in Java

     \*/

**public** **static** **boolean** checkDuplicateUsingAdd(String[] input) {

        Set tempSet = **new** HashSet();

**for** (String str : input) {

**if** (!tempSet.add(str)) {

**return** **true**;

            }

        }

**return** **false**;

    }

}

**Output:**

Checking array with duplicate using brute force: true

Checking array without any duplicate using brute force: false

Checking array with duplicate using Set and List: true

Checking array without any duplicate using Set and List: false

Checking array with duplicate using Set and List: true

Checking array without any duplicate using Set and List: false

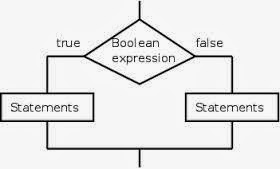
That’s all on **how to check if an Array contains duplicate or not in Java**. You see we have used Java Collection API in two of our example, there can be other pure programming solution as well. You may be asked to detect duplicates without using Java API in the real interview. Let us know if you come across some other good way *of checking duplicates in an array without using Java API.*

Read more: <http://javarevisited.blogspot.com/2012/02/how-to-check-or-detect-duplicate.html#ixzz4esV53Moz>

# How to Find Top Two Maximum Number from Integer array in Java

In this post, I have come with another simple programming problems for Java beginners. I love to share short programming problems because they help in developing programming sense. Many people will argue against simple problems like [prime numbers](http://java67.blogspot.sg/2014/01/how-to-check-if-given-number-is-prime.html), [palindrome](http://java67.blogspot.sg/2012/09/palindrome-java-program-to-check-number.html), and [factorial](http://javarevisited.blogspot.sg/2012/04/java-program-to-find-factorial-of.html), but I really find them useful, especially for beginners. A beginner is far away to solve a complex data structure problem or even more complex problems like those appear in TopCoder or other programming sites. Programmers learn gradually and they need the joy of doing something and seeing result much quickly than any other. Small success motivates them. Anyway, here is our problem statement, you need to write a Java program to *find top two maximum numbers in the given array*. You can not use any sorting functions and you should iterate the array only once. Use of any kind of collection class e.g. [TreeSet or LinkedHashSet](http://java67.blogspot.sg/2014/01/when-to-use-linkedhashset-vs-treeset-vs-hashset-java.html) is also not allowed.  
  
For example, if given integer array is [20, 34, 21, 87, 92, 2147483647] then first maximum is 2147483647 and second maximum is 92. Bonus points are for those, who can also write *JUnit test cases*, more test scenarios, more points.  
  
Few more bonus points for those who can write code to deal with really large array, something which may not entirely fit on memory.

## Java Program To Find Top Two Maximum Numbers from Integer Array

[](http://3.bp.blogspot.com/-S-QEd_gY4mY/UzWIK8goJgI/AAAAAAAABWI/1hH6rAPTT8Q/s1600/if-else+java.jpg)Here is our sample Java program, It solves the problem following given problem statement and under constraints states. For example it doesn't use any sorting algorithm e.g. [bubble sort](http://java67.blogspot.sg/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html) or quicksort, or Collections.sort() method. As I said before,  this kind of program is good exercise for mastering basic building blocks of any programming language e.g. loops, if-else block and learning relational operator like less than (<) and greater than (>). It take advantage of if-else control statement to solve this problem. All we do is we start with two variables as max1 and max2 and initialized them with Integer.MIN\_VALUE, which is the limit of minimum value. Now we [iterate through array](http://java67.blogspot.sg/2013/08/how-to-iterate-over-array-in-java-15.html) and compare each number against these two number, if current number is greater than max1 then max1 = number and max2 = max1. Otherwise if it only greater than max2 then we only update max2 with current number. At the end of iteration, max1 and max2 points to top two numbers from given array. You see problem solved without any utility class or third-party library.

**import** **java.util.Arrays**;

/\*\*

\* Java program to find top two maximum numbers from an integer array.

\*

\* @author http://java67.blogspot.com

\*/

**public** **class** **TopTwoMaximum**{

**public** **static** **void** **main**(String args[]) {

topTwo(**new** **int**[]{**20**, **34**, **21**, **87**, **92**, Integer.MAX\_VALUE});

topTwo(**new** **int**[]{**0**, Integer.MIN\_VALUE, -**2**});

topTwo(**new** **int**[]{Integer.MAX\_VALUE, **0**, Integer.MAX\_VALUE});

topTwo(**new** **int**[]{**1**, **1**, **0**});

}

**public** **static** **void** **topTwo**(**int**[] numbers) {

**int** max1 = Integer.MIN\_VALUE;

**int** max2 = Integer.MIN\_VALUE;

**for** (**int** number : numbers) {

**if** (number > max1) {

max2 = max1;

max1 = number;

} **else** **if** (number > max2) {

max2 = number;

}

}

System.out.println("Given integer array : " + Arrays.toString(numbers));

System.out.println("First maximum number is : " + max1);

System.out.println("Second maximum number is : " + max2);

}

}

*Output:*

Given integer array : [**20**, **34**, **21**, **87**, **92**, **2147483647**]

First maximum number is : **2147483647**

Second maximum number is : **92**

Given integer array : [**0**, -**2147483648**, -**2**]

First maximum number is : **0**

Second maximum number is : -**2**

Given integer array : [**2147483647**, **0**, **2147483647**]

First maximum number is : **2147483647**

Second maximum number is : **2147483647**

Given integer array : [**1**, **1**, **0**]

First maximum number is : **1**

Second maximum number is : **1**

From output, you can see that our method topTwo(int[] numbers) are working properly for different set of inputs. I have choose main method over JUnit test for testing my code, which is quick and dirty. JUnit testing provides you more option and better framework to write testcases. If you don't know [how to write JUnit test case](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html), see that link. It explains that in details.  
  
That's all on **how to find two maximum from integer array in Java**. How about extending it further and finding top three numbers from integer array? Can you do that without any help? Once you done that, You can probably try finding top three minimum numbers from array. For example if given array is [11, 2, 5, 4] then top three minimum numbers are 2, 4 and 5. You can even extend this logic even to [find largest and smallest number in array](http://java67.blogspot.sg/2014/02/how-to-find-largest-and-smallest-number-array-in-java.html), as shown in that example

Read more: <http://www.java67.com/2014/03/how-to-find-top-two-maximum-number-from-integer-array-java.html#ixzz4esVGEjE5>

# [How to Remove Duplicates from Array without using Java Collection API](http://javarevisited.blogspot.in/2014/01/how-to-remove-duplicates-from-array-java-without-collection-API.html)

This is a coding question recently asked to one of my readers in a Java Technical interview. Question was to remove duplicates from an integer array without using any collection API classes like Set or LinkedHashSet, which can make this task trivial. In general, if you need to do this for any project work, I suggest better using Set interface, particularly [LinkedHashSet](http://javarevisited.blogspot.sg/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html), because that also keep the order on which elements are inserted into Set. Only for technical interview perspective, you need to do this using either loops or recursion,  depending upon what is your strongest area. In this article, I am sharing a naive solution, which has lots of limitation to be considered as [production quality code](http://javarevisited.blogspot.sg/2011/09/how-to-write-production-quality-code.html), It's not the best solution but still a solution.  
  
The main problem, while dealing with an *array is not finding duplicates*, it's about *removing* them. Since an array is a static, fixed length data structure, you can not change its length. This means, deleting an element from an array requires creating a new array and copying content into that array.  
  
If your input array contains lots of duplicates then this may result in lots of temporary arrays. It also increases cost of copying contents, which can be very bad. Given this restriction, you need to come out with a strategy to minimize both [memory and CPU](http://javarevisited.blogspot.sg/2013/06/find-cpu-and-memory-used-by-java-solaris-prstat-command-example.html) requirements.

### Java Program to remove duplicates from integer array without Collection

[Java Program to remove duplicates from Integer array without Collection](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)In this program, we have not used any collection class to remove duplicates, earlier, I had shown you a way to [remove duplicates from ArrayList](http://javarevisited.blogspot.sg/2012/12/how-to-remove-duplicates-elements-from-ArrayList-Java.html), which was using LinkedHashSet. You can still use that solution if the interviewer doesn't mention without Collection specifically.  
  
All you need to do is to convert your array into ArrayList first then subsequently create a LinkedHashSet from that ArrayList. In this example, we are removing duplicates from the array by not copying them into result array, this solution is not actually deleting duplicates instead it replacing it with default value i.e. zero.  
  
You can also see [Cracking the Coding Interview](http://www.amazon.com/Cracking-Coding-Interview-6th-Edition/dp/0984782850/?tag=javamysqlanta-20), a collection of 189 coding questions from various programming interviews from tech companies like Amazon, Google, Facebook, and Microsoft. That will help you to develop the coding sense you need to solve problems like this.

Now, let's see our Java solution for removing duplicates from integer array:

**import** **java.util.Arrays**;

**import** **org.slf4j.Logger**;

**import** **org.slf4j.LoggerFactory**;

/\*\*

 \* Java program to remove duplicates from this array. You don't

 \* need to physically delete duplicate elements, replacing with null, or

 \* empty or default value is ok.

 \*

 \* @author http://javarevisited.blogspot.com

 \*/

**public** **class** **TechnicalInterviewTest** {

**private** **static** **final** Logger logger = LoggerFactory.getLogger(TechnicalInterviewTest.class);

**public** **static** **void** **main**(String args[]) {

**int**[][] test = **new** **int**[][]{

            {**1**, **1**, **2**, **2**, **3**, **4**, **5**},

            {**1**, **1**, **1**, **1**, **1**, **1**, **1**},

            {**1**, **2**, **3**, **4**, **5**, **6**, **7**},

            {**1**, **2**, **1**, **1**, **1**, **1**, **1**},};

**for** (**int**[] input : test) {

            System.out.println("Array with Duplicates       : " + Arrays.toString(input));

            System.out.println("After removing duplicates   : " + Arrays.toString(removeDuplicates(input)));

        }

    }

    /\*

     \* Method to remove duplicates from array in Java, without using

     \* Collection classes e.g. Set or ArrayList. Algorithm for this

     \* method is simple, it first sort the array and then compare adjacent

     \* objects, leaving out duplicates, which is already in the result.

     \*/

**public** **static** **int**[] **removeDuplicates**(**int**[] numbersWithDuplicates) {

        // Sorting array to bring duplicates together

        Arrays.sort(numbersWithDuplicates);

**int**[] result = **new** **int**[numbersWithDuplicates.length];

**int** previous = numbersWithDuplicates[**0**];

        result[**0**] = previous;

**for** (**int** i = **1**; i < numbersWithDuplicates.length; i++) {

**int** ch = numbersWithDuplicates[i];

**if** (previous != ch) {

                result[i] = ch;

            }

            previous = ch;

        }

**return** result;

    }

}

**Output :**

Array with Duplicates       : [**1**, **1**, **2**, **2**, **3**, **4**, **5**]

After removing duplicates   : [**1**, **0**, **2**, **0**, **3**, **4**, **5**]

Array with Duplicates       : [**1**, **1**, **1**, **1**, **1**, **1**, **1**]

After removing duplicates   : [**1**, **0**, **0**, **0**, **0**, **0**, **0**]

Array with Duplicates       : [**1**, **2**, **3**, **4**, **5**, **6**, **7**]

After removing duplicates   : [**1**, **2**, **3**, **4**, **5**, **6**, **7**]

Array with Duplicates       : [**1**, **2**, **1**, **1**, **1**, **1**, **1**]

After removing duplicates   : [**1**, **0**, **0**, **0**, **0**, **0**, **2**]

That's it about **how to remove duplicates from an array in Java without using Collection class**. As I said before, this solution is not perfect and has some serious limitation, which is an exercise for you to find out. One hint I can give is that array itself can contain default value as duplicates e.g. 0 for int, even if you use any Magic number e.g. Integer.MAX\_VALUE, you can  not be certain that they will not be part of the input.  
  
Regarding removing duplicate permanently from result array, one approach could be to count a number of duplicates and then create an array of right size i.e. length - duplicates, and then copying content from intermediate result array to final array, leaving out elements which are marked duplicate.

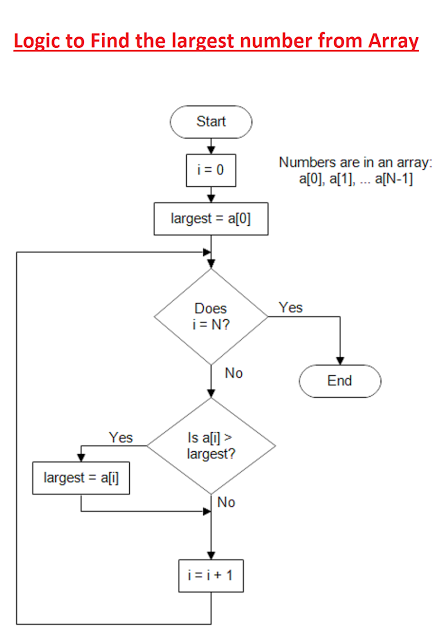
Read more: <http://javarevisited.blogspot.com/2014/01/how-to-remove-duplicates-from-array-java-without-collection-API.html#ixzz4esVQhDje>

# How to find largest and smallest number from integer array - Java Solution

Good Understanding of array data structure is very important for any software developer, and to develop this understanding there are lots of programming exercise beginners can do. One of them is writing *a program to find smallest and largest number in an integer array*. Java programmers are no different than others, so they can do this program in Java, not just to understand array but also relational operators available in Java.  In this program, you need to write a method, yes we call the function a method in Java, which will accept an [integer array](http://java67.blogspot.sg/2012/12/difference-between-array-vs-arraylist-java.html) and then print largest and smallest number from that array. Use of any third-party library or API method is not allowed, which means you need to do this exercise by using basic tools of Java programming language, which includes operators, control statements,  keyword and some classes from java.lang package.  
  
This problem is also known as **finding maximum and minimum numbers in an array**, and technique mentioned here can be used in any other programming language as well. As a bonus point, you can also write [JUnit test cases](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) to test your method, I have not done so and relied on simple main method to test my code to show the output and keep it short, essential for any example or demo.  
  
Btw, if you preparing for programming job interview, then don't forget to check the [Cracking the Coding Interview](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20) book. It contains 150 Programming Questions and Solutions, which is more than enough for many coding interviews.

## Java Program to find smallest and largest number in an integer array

Here is full code example of Java program to find smallest and largest number from an integer array. You can create a Java source file with name MaximumMinimumArrayDemo.java and copy code there to compile and execute in your favorite IDE. If you don't have IDE setup, you can also compile and run this program by following steps I have shown on [HelloWorld in Java](http://java67.blogspot.sg/2013/03/helloworld-in-java-how-to-write-compile-example-tutorial.html).  
  
If you look at the code here, we have created a method called **largestAndSmallest**(**int**[] numbers)  to print largest and smallest number from int array passed to the program.  We use two variables largest and smallest to store the maximum and minimum values from the array. Initially largest is initialized with Integer.MIN\_VALUE and smallest is initialized with Integer.MAX\_VALUE.  
  
In each iteration of the loop, we compare current number with largest and smallest and update them accordingly. Since if a number is larger than largest, it can't be smaller than smallest, which means you don't need to check if the first condition is true, that's why we have used if-else code block, where else part will only execute if the first condition is not true.  
  
Here is another logic to find the largest element from an array in Java, here instead of assigning the variable with Integer.MAX\_VALUE, we have assigned the first element from the array.

[](http://1.bp.blogspot.com/-t91k8GIn7a8/VhPeSHpZZhI/AAAAAAAAD54/zl2F8Yf7p7E/s1600/find-largest-number%2Bin%2Barray%2Bjava.png)

Since array doesn't override [the toString method in Java](http://javarevisited.blogspot.sg/2012/09/override-tostring-method-java-tips-example-code.html), we have used Arrays.toString() to print contents of an array. Remember this function is outside of core logic, so it's Ok to use it. Since this is a static method we can directly call this from the main method in Java, and so does our test code. We pass the random array to this method and see if largest and smallest number returned by the method is correct or not. For automated testing, a Unit test is better but for demonstration, you can use the main method.  
  
  
**Java Program to find the largest and smallest element in array:**

import java.util.Arrays;

/\*\*

\* Java program to find largest and smallest number from an array in Java.

\* You cannot use any library method both from Java and third-party library.

\*

\* @author http://java67.blogspot.com

\*/

public class MaximumMinimumArrayDemo{

public static void main(String args[]) {

largestAndSmallest(new int[]{-20, 34, 21, -87, 92,

Integer.MAX\_VALUE});

largestAndSmallest(new int[]{10, Integer.MIN\_VALUE, -2});

largestAndSmallest(new int[]{Integer.MAX\_VALUE, 40,

Integer.MAX\_VALUE});

largestAndSmallest(new int[]{1, -1, 0});

}

public static void largestAndSmallest(int[] numbers) {

int largest = Integer.MIN\_VALUE;

int smallest = Integer.MAX\_VALUE;

for (int number : numbers) {

if (number > largest) {

largest = number;

} else if (number < smallest) {

smallest = number;

}

}

System.out.println("Given integer array : " + Arrays.toString(numbers));

System.out.println("Largest number in array is : " + largest);

System.out.println("Smallest number in array is : " + smallest);

}

}

Output:

Given integer array : [-20, 34, 21, -87, 92, 2147483647]

Largest number in array is : 2147483647

Smallest number in array is : -87

Given integer array : [10, -2147483648, -2]

Largest number in array is : 10

Smallest number in array is : -2147483648

Given integer array : [2147483647, 40, 2147483647]

Largest number in array is : 2147483647

Smallest number in array is : 40

Given integer array : [1, -1, 0]

Largest number in array is : 1

Smallest number in array is : -1

That's all about **How to find largest and smallest number from integer array in Java**. As I said this question can also be asked as to find the maximum and minimum numbers in an [Array in Java](http://javarevisited.blogspot.sg/2013/11/java-array-101-for-programmers-and.html), so don't get confused there. By the way, there are more ways to do the same task and you can practice it to code solution differently. Can you write a solution which is different than this? go ahead and give it a try.

Read more: <http://www.java67.com/2014/02/how-to-find-largest-and-smallest-number-array-in-java.html#ixzz4esVbqXpC>

# [How to find middle element of LinkedList in Java in one pass](http://javarevisited.blogspot.in/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html)

How do you find middle element of LinkedList in one pass is a programming question often asked Java and non-Java programmers in telephonic Interview. This question is similar to [checking palindrome](http://javarevisited.blogspot.sg/2012/12/how-to-check-if-number-is-palindrome-or-not-example.html) or [calculating the factorial](http://javarevisited.blogspot.sg/2012/04/java-program-to-find-factorial-of.html), where Interviewer sometimes also ask to write code. In order to answer this question candidate must be familiar with LinkedList data structure i.e. In the case of singly LinkedList, each node of Linked List contains data and pointer, which is the address of next Linked List and the last element of Singly Linked List points towards the null. Since in order to find middle element of Linked List you need to find the length of LinkedList, which is counting elements till end i.e. until you find the last element of Linked List. What makes this data structure Interview question interesting is that you need to *find middle element of LinkedList in one pass* and you don’t know the length of LinkedList. This is where candidates logical ability puts into the test, whether he is familiar with space and time trade off or not etc.  
   
As if you think carefully you can solve this problem by using two pointers as mentioned in my last post on [How to find length of Singly Linked List in Java](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html). By using two pointers, incrementing one at each iteration and other at every second iteration. When first pointer will point at end of Linked List, second pointer will be pointing at middle node of Linked List.   
  
In fact this two pointer approach can solve multiple similar problems e.g. How to find 3rd element from last in a Linked List in one Iteration or How to find nth element from last in a Linked List. In this Java programming tutorial we will see a Java program which finds middle element of Linked List in one Iteration.

## **Java program to find middle element of LinkedList in one pass**

[How to find middle element of Linked List in Java with Example](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)Here is complete Java program to find middle node of Linked List in Java. Remember LinkedList class here is our custom class and don’t confuse this class with [java.util.LinkedList](http://javarevisited.blogspot.sg/2012/02/difference-between-linkedlist-vs.html)  which is a popular Collection class in Java. In this Java program, our class LinkedList represent a linked list data structure which contains collection of node and has head and tail. Each Node contains data and address part. Main method of LinkedListTest class is used to simulate the problem, where we created Linked List and added few elements on it and then iterate over them to find middle element of Linked List in one pass in Java.

**import** test.LinkedList.Node;  
  
/\*\*  
 \* **Java program to find middle element of linked list in one pass**.  
 \* In order to find middle element of linked list we need to find length first  
 \* but since we can only traverse linked list one time, we will use two pointers  
 \* one which we will increment on each iteration while other which will be  
 \* incremented every second iteration. so when first pointer will point to the  
 \* end of linked list, second will be pointing to the middle element of linked list  
 \* @author  
 \*/  
**public** **class** LinkedListTest {  
    
    
    **public** **static** **void** main(**String** args[]) {  
        *//creating LinkedList with 5 elements including head*  
      **LinkedList** linkedList = **new** **LinkedList**();  
      **LinkedList**.**Node** head = linkedList.head();  
      linkedList.add( **new** **LinkedList**.**Node**("1"));  
      linkedList.add( **new** **LinkedList**.**Node**("2"));  
      linkedList.add( **new** **LinkedList**.**Node**("3"));  
      linkedList.add( **new** **LinkedList**.**Node**("4"));  
      
      *//finding middle element of LinkedList in single pass*  
      **LinkedList**.**Node** current = head;  
      **int** length = 0;  
      **LinkedList**.**Node** middle = head;  
      
      while(current.next() != **null**){  
          length++;  
          if(length%2 ==0){  
              middle = middle.next();  
          }  
          current = current.next();  
      }  
      
      if(length%2 == 1){  
          middle = middle.next();  
      }  
  
      **System**.out.println("length of LinkedList: " + length);  
      **System**.out.println("middle element of LinkedList : " + middle);  
        
    }   
    
}  
  
**class** **LinkedList**{  
    **private** **Node** head;  
    **private** **Node** tail;  
    
    **public** **LinkedList**(){  
        **this**.head = **new** **Node**("head");  
        tail = head;  
    }  
    
    **public** **Node** head(){  
        **return** head;  
    }  
    
    **public** **void** add(**Node** node){  
        tail.next = node;  
        tail = node;  
    }  
    
    **public** **static** **class** **Node**{  
        **private** **Node** next;  
        **private** **String** data;  
  
        **public** **Node**(**String** data){  
            **this**.data = data;  
        }  
        
        **public** **String** data() {  
            **return** data;  
        }  
  
        **public** **void** setData(**String** data) {  
            **this**.data = data;  
        }  
  
        **public** **Node** next() {  
            **return** next;  
        }  
  
        **public** **void** setNext(**Node** next) {  
            **this**.next = next;  
        }  
        
        **public** **String** toString(){  
            **return** **this**.data;  
        }  
    }  
}  
**Output:**  
length of LinkedList: 4  
middle element of LinkedList : 2

That’s all on **How to find middle element of LinkedList in one pass.** As I said this is a good interview question to separate programmers from non programmers. Also technique mentioned here to find middle node of LinkedList can be used to find 3rd element from Last or nth element from last in a LinkedList as well.

Read more: <http://javarevisited.blogspot.com/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html#ixzz4esVp0gNJ>

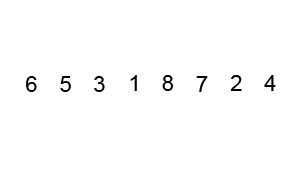
# [Quicksort Sorting Algorithm in Java](http://javarevisited.blogspot.in/2014/08/quicksort-sorting-algorithm-in-java-in-place-example.html)

Quicksort algorithm is one of the most used sorting algorithm, especially to sort large list and most of the programming languages, library have implemented it in one or another way. In Java, Arrays.sort() method sorts primitive data types using double pivot Quicksort algorithm, authored by Joshua Bloach and others. This implementation provides better performance for lot of data sets, where traditional quicksort algorithm reduced into quadratic performance. This method also uses MergeSort, another good sorting algorithm, to sort objects. QuickSort implementations are also available in C++ STL library. Have you ever thought *why quicksort is so popular?* because on average it is one of the fastest sorting algorithm we have. On average quicksort is a O(n log n) algorithm, while it's worst case is O(n^2), which is much better comparing with Bubble Sort or Insertion Sort. It's also one of the [popular algorithm interview question](http://javarevisited.blogspot.sg/2013/03/top-15-data-structures-algorithm-interview-questions-answers-java-programming.html), so as a programmer you must know *how QuickSort works as well* as *how to implement Quicksort in Java* or any other programming language. One of the most important thing interviewer look in your quicksort implementation is choice of pivot and whether you are sorting in place or not. In "*in-place"* sorting, actual sorting takes place in same array and no additional space is needed. Due to this reason, quicksort is very efficient in sorting large list of numbers, as no additional memory is required, a very space efficient sorting algorithm. Quicksort is also one of the naturally recursive algorithm and serves a good exercise for Java programmers to master [art of recursion](http://javarevisited.blogspot.sg/2012/12/recursion-in-java-with-example-programming.html).

### How QuickSort Algorithm works

Quicksort is a divide and conquer algorithm, which means original list is divided into multiple list, each of them is sorted individually and then sorted output is merged to produce the sorted list. Here is step by step explanation of how quicksort algorithm works.

Steps to implement Quick sort algorithm in place:  
  
1) Choose an element, called pivot, from the list or array. Generally pivot is the middle element of array.  
  
2) Reorder the list so that all elements with values less than the pivot come before the pivot, and all elements with values greater than the pivot come after it (equal values can go either way). This is also known as *partitioning*. After partitioning the pivot is in its final position.  
  
3) Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values. If the array contains only one element or zero elements then the array is sorted.  
  
Following GIF image will help you to understand [working of Quick sort algorithm](http://java67.blogspot.sg/2014/07/quicksort-algorithm-in-java-in-place-example.html) little better. In this image we have an array of integers which is not sorted and we need to sort them in ascending order. Our array is {6, 5, 3, 1, 8, 7, 2, 4} and we first choose 3 as pivot. Now partitioning starts and we pick 6 on left side of side, because its greater than 3. Now on right side, we leave 4 because its greater than 3, and pick 2 for swapping with 6. After swapping our list look like {2, 5, 3, 1, 8, 7, 6, 4}. Now we pick 5 on left side, and 1 on right side because it's greater than 3 and swap them again. Now, our array looks like {2, 1, 3, 5, 8, 7, 6, 4}. Since we are done with all elements with respect to 3 as pivot, we can now take the sub-array at left side of 3 and apply same procedure. This will sort the left array. Now on right side, we choose 4 as pivot, and repeat same procedure, which result in 4 swapped against 5. Now we take right side again with 6 as pivot and apply same procedure.

[](http://3.bp.blogspot.com/-2XKswkxXtE0/U8EkLJIt2BI/AAAAAAAABr0/0S34Aan5jMI/s1600/Quicksort-in-Java-example-fast.gif)

**Sorting an array of integer using QuickSort sorting algorithm**

### Java Program to implement QuickSort Algorithm

Here is a Java program to [sort an array of integers using QuickSort algorithm](http://java67.blogspot.sg/2014/08/4-examples-to-sort-array-in-java.html). It is an in-place, recursive implementation of QuickSort. Logic is encapsulated in QuickSort class, and method quickSort(int low, int high). This method is called recursively to sort the array. This algorithm work exactly as explained in above GIF image, so if you understand the logic there, its very easy to write by your own.

import java.util.Arrays;

/\*\*

\* Test class to sort array of integers using Quicksort algorithm in Java.

\* @author Javin Paul

\*/

public class QuickSortDemo{

public static void main(String args[]) {

// unsorted integer array

int[] unsorted = {6, 5, 3, 1, 8, 7, 2, 4};

System.out.println("Unsorted array :" + Arrays.toString(unsorted));

QuickSort algorithm = new QuickSort();

// sorting integer array using quicksort algorithm

algorithm.sort(unsorted);

// printing sorted array

System.out.println("Sorted array :" + Arrays.toString(unsorted));

}

}

/\*\*

\* Java Program sort numbers using QuickSort Algorithm. QuickSort is a divide

\* and conquer algorithm, which divides the original list, sort it and then

\* merge it to create sorted output.

\*

\* @author Javin Paul

\*/

class QuickSort {

private int input[];

private int length;

public void sort(int[] numbers) {

if (numbers == null || numbers.length == 0) {

return;

}

this.input = numbers;

length = numbers.length;

quickSort(0, length - 1);

}

/\*

\* This method implements in-place quicksort algorithm recursively.

\*/

private void quickSort(int low, int high) {

int i = low;

int j = high;

// pivot is middle index

int pivot = input[low + (high - low) / 2];

// Divide into two arrays

while (i <= j) {

/\*\*

\* As shown in above image, In each iteration, we will identify a

\* number from left side which is greater then the pivot value, and

\* a number from right side which is less then the pivot value. Once

\* search is complete, we can swap both numbers.

\*/

while (input[i] < pivot) {

i++;

}

while (input[j] > pivot) {

j--;

}

if (i <= j) {

swap(i, j);

// move index to next position on both sides

i++;

j--;

}

}

// calls quickSort() method recursively

if (low < j) {

quickSort(low, j);

}

if (i < high) {

quickSort(i, high);

}

}

private void swap(int i, int j) {

int temp = input[i];

input[i] = input[j];

input[j] = temp;

}

}

Output :

Unsorted array :[6, 5, 3, 1, 8, 7, 2, 4]

Sorted array :[1, 2, 3, 4, 5, 6, 7, 8]

### Import points about Quicksort algorithm

Now we know how quick sort works and how to implement quicksort in Java, its time to revise some of the important points about this popular sorting algorithm.  
  
1) QuickSort is a divide and conquer algorithm. Large list is divided into two and sorted separately (conquered), sorted list is merge later.  
  
2) On "in-place" implementation of quick sort, list is sorted using same array, no additional array is required. Numbers are re-arranged pivot, also known as partitioning.  
  
3) Partitioning happen around pivot, which is usually middle element of array.  
  
4) Average case time complexity of Quicksort is O(n log n) and worst case time complexity is O(n ^2), which makes it one of the fasted sorting algorithm. Interesting thing is it's worst case performance is equal to [Bubble Sort](http://javarevisited.blogspot.sg/2014/08/bubble-sort-algorithm-in-java-with.html) :)  
  
5) Quicksort can be implemented with an in-place partitioning algorithm, so the entire sort can be done with only O(log n) additional space used by the stack during the recursion.  
  
6) Quicksort is also a good example of algorithm which makes best use of CPU caches, because of it's divide and conquer nature.  
  
7) In Java, Arrays.sort() method uses quick sort algorithm to sort array of primitives. It's different than our algorithm, and uses two pivots. Good thing is that it perform much better than most of the quicksort algorithm available on internet for different data sets, where traditional quick sort perform poorly. One more reason, not to reinvent the wheel but to use the library method, when it comes to write production code.  
  
  
That's all about **Quicksort sorting algorithm in Java**. It is one of the must know algorithm for all level of Java programmers, not that you need it often to implement it but to do well on interviews and use the lesson learned while implementing quick sort in Java. In our example, we have implemented quicksort "in-place", which is what you should do if asked to write quicksort in Java. Remember as Java programmer, you don't need to write your own implementation as library implementation are much better implemented and tested. You should use  Arrays.sort()  method to sort your array instead of writing your own sort method. One more reason of using library method is that they are usually improved over different version, and can take advantage of new machine instructions or native improvement.

Read more: <http://javarevisited.blogspot.com/2014/08/quicksort-sorting-algorithm-in-java-in-place-example.html#ixzz4esW4F29E>

# [How to use Comparator and Comparable in Java? With example](http://javarevisited.blogspot.in/2011/06/comparator-and-comparable-in-java.html)

**Comparator and Comparable in Java Examples**

Difference between Comparator and Comparable in Java is very [popular Java interview question](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html) mostly asked in telephonic round and writing code to sort object using Comparable or Comparator is popular on  written test round of interview.The question was this “How you will sort Employee object based on his EmployeeID and his name” and this involves the use of both Comparable as well as Comparator interface in Java. This post is my revision on Java fundamentals similar to I did about [equals method in Java](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) and  some tips to [override hashCode in Java](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html). All of these methods are fundamentals in Java programming language and correct understanding is must for any Java developer. **Comparators and comparable** in Java are two interfaces which is used to implement sorting in Java. It’s often required to sort objects stored in any collection classes like ArrayList, HashSet or in Array and that time we need to use either  compare() or  compareTo() method defined in java.util.Comparator and java.lang.Comparable. In this Java tutorial we will see example of  Comparator and Comparable to sort object in Java and discuss some best practices around when to use Comparator interface etc. Any way before moving ahead Let’s see some important differences between Comparable and Comparator in Java.

## **Comparator vs Comparable in Java**

[Difference between Comparator vs Comparable in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)Here are some of the common differences, which is worth remembering to answer this question if asked during a telephonic or face to face interview:

1) Comparator in Java is defined in java.util package while Comparable interface in Java is defined in java.lang package, which very much says that Comparator should be used as an utility to sort objects which Comparable should be provided by default.

2) Comparator interface in Java has method public int compare (Object o1, Object o2) which returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. While Comparable interface has method public int compareTo(Object o) which returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

3) If you see then logical difference between these two is *Comparator in Java* compare two objects provided to him, while Comparable interface compares "this" reference with the object specified. I have shared lot of tips on [how to override compareTo() method](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) and avoid some common mistakes programmer makes while implementing Comparable interface.

4) Comparable in Java is used to implement **natural ordering of object**. In Java API String, Date and wrapper classes implements Comparable interface.Its always good practice to override compareTo() for value objects.

5) If any class implement Comparable interface in Java then collection of that object either [List](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) or Array can be sorted automatically by using  Collections.sort() or Arrays.sort() method and object will be sorted based on there natural order defined by CompareTo method.

6)Objects which implement *Comparable in Java*  can be used as keys in a SortedMap like [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html) or elements in a SortedSet  for example TreeSet, without specifying any Comparator.

These were combination of some theoretical and practical differences between Comparator and Comparator interface in Java. It does help you to decide when to use Comparator vs Comparable but things will be more clear when we some best practices around using both of these interfaces. Now let’s see an example of Comparator in Java:

### Example of using Comparator and Comparable in Java

So in Summary if you want to **sort objects based on natural order** then use Comparable in Java and if you want to sort on some other attribute of object then use Comparator in Java. Now to understand these concepts lets see an example or real life coding:

1) There is class called Person, sort the Person based on person\_id, which is primary key in database

2) Sort the Person based on there name.

For a Person class, sorting based on person\_id can be treated as *natural order sorting* and sorting based on name field can be implemented using Comparator interface. To sort based on person\_id we need to implement compareTo() method.

**public** **class** Person **implements** **Comparable** {  
    **private** **int** person\_id;  
    **private** **String** name;  
      
    /\*\*  
     \* Compare current person with specified person  
     \* return zero if person\_id for both person is same   
     \* return negative if current person\_id is less than specified one  
     \* return positive if specified person\_id is greater than specified one  
     \*/  
  @Override   
    **public** **int** compareTo(Object o) {  
        Person p = (Person) o;   
        **return** **this**.person\_id - o.person\_id ;  
    }  
    ….  
}

Generally you should not use difference of integers to decide output of compareTo method as result of **integer subtraction can overflow** but if you are sure that both operands are positive then its one of the quickest way to compare two objects. See my post [things to remember while overriding compareTo in Java](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html) for more tips on compareTo.

And for sorting based on person name we can implement compare(Object o1, Object o2) method of Java Comparator class.

/\*\*  
 \* Comparator implementation which sorts Person objects on person\_id field  
 \*/  
**public** **class** SortByPerson\_ID **implements** **Comparator**{  
  
    **public** **int** compare(Object o1, Object o2) {  
        Person p1 = (Person) o;  
        Person p2 = (Person) o;   
        **return** p1.getPersonId() - p2.getPersonId();  
    }  
}

Similar guidelines applies while implementing compare() method as well and instead of using subtraction operator, its better to use logical operator to compare whether two integers are equal to, less than or greater than. You can write several types of Java Comparator based upon your need for example  reverseComparator , ANDComparator , ORComparator etc which will return negative or positive number based upon logical results. [String in Java](http://javarevisited.blogspot.sg/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) even provides an special comparator called CASE\_INSENSITIVE\_ORDER, to perform case insensitive comparison of String objects.

**How to Compare String in Java**

[String is immutable in Java](http://javarevisited.blogspot.sg/2010/10/why-string-is-immutable-in-java.html) and one of the most used value class. For comparing String in Java we should not be worrying because String implements Comparable interface and provides a lexicographic implementation for CompareTo method which compare two strings based on contents of characters or you can say in lexical order. You just need to call String.compareTo(AnotherString) and Java will determine whether specified String is greater than , equal to or less than current object. See my post [4 example to compare String in Java](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html) for alternatives ways of comparing String.

**How to Compare Dates in Java**

Dates are represented by java.util.Date class in Java and like String,  Date also implements Comparable in Java so they will be automatically sorted based on there natural ordering if they got stored in any sorted collection like TreeSet or TreeMap. If you explicitly wants to compare two dates in Java you can call Date.compareTo(AnotherDate) method in Java and it will tell whether specified date is greater than , equal to or less than current String. See my post [3 ways to compare Dates in Java](http://javarevisited.blogspot.sg/2012/02/3-example-to-compare-two-dates-in-java.html) for more alternatives of comparing two dates.

**When to use Comparator and Comparable in Java**

At last let’s see some best practices and recommendation on when to use Comparator or Comparable in Java:

1) If there is a natural or default way of sorting Object already exist during development of Class than use Comparable. This is intuitive and you given the class name people should be able to guess it correctly like Strings are sorted chronically, Employee can be sorted by there Id etc. On the other hand if an Object can be sorted on multiple ways and client is specifying on which parameter sorting should take place than use Comparator interface. for example Employee can again be sorted on name, salary or department and clients needs an API to do that. Comparator implementation can sort out this problem.

2) Some time you write code to sort object of a class for which you are not the original author, or you don't have access to code. In these cases you can not implement Comparable and Comparator is only way to sort those objects.

3) Beware with the fact that How those object will behave if stored in SorteSet or SortedMap like TreeSet and [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html). If an object doesn't implement Comparable than while putting them into SortedMap, always provided corresponding Comparator which can provide sorting logic.

4) Order of comparison is very important while implementing Comparable or Comparator interface. for example if you are sorting object based upon name than you can compare first name or last name on any order, so decide it judiciously. I have shared more detailed tips on compareTo on my post how to implement CompareTo in Java.

5) Comparator has a distinct advantage of being self descriptive  for example if you are writing Comparator to compare two Employees based upon there salary than name that comparator as SalaryComparator, on the other hand compareTo()

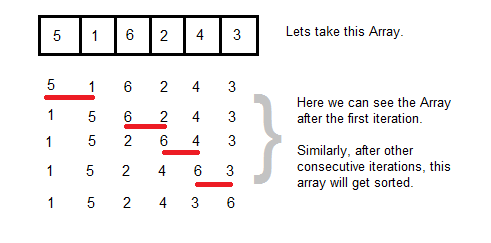
Read more: <http://javarevisited.blogspot.com/2011/06/comparator-and-comparable-in-java.html#ixzz4esWBz0y1>

# [Bubble Sort Algorithm in Java with Example](http://javarevisited.blogspot.in/2014/08/bubble-sort-algorithm-in-java-with.html)

Bubble Sort is the first sorting algorithm I learned during my college day, and after so many years it's the one I remember by heart. It's kind of weird that one of the most popular sorting algorithm is also one of the worst performing sorting algorithm. Bubble sort's average case performance is in O(n^2), which means as the size array grows, the time it take to sort that array increases quadratic. Due to this reason, bubble sort is not used in production code, instead quick sort and merge sort are preferred over it. In fact, Java's own Arrays.sort() method, which is the [easiest way to sort an array in Java](http://javarevisited.blogspot.com/2012/01/sort-array-in-java-ascending-and.html) also uses two pivot quicksort to sort primitive array and stable mergesort algorithm to sort object arrays.  
  
The reason of slow performance of this algorithm is excessive comparison and swapping, since it compare each element of array to another and swaps if it is on right side.  
  
Due to quadratic performance, bubble sort is best suited for small, almost sorted list e.g. {1, 2, 4, 3, 5} , where it just need to do one swapping. Ironically, best case performance of bubble sort, which is O(n) beats quicksort's best case performance of O(NlogN).  
  
Someone may argue that why teaching an algorithm which that poor performance, why not teach insertion or selection sort which is as easy as bubble sort, and performs better. IMHO, easiness of algorithm depends upon programmer as much as on algorithm itself.  
  
Many programmer will find *insertion sort* easier than *bubble sor*t but again there will be a lot many who will find bubble sort easier to remember, including myself. This is true, despite many of them have used insertion sort unknowingly in real life, e.g. sorting playing cards in hand.  
  
Another reason for learning this sorting algorithm is for comparative analysis, how you improve algorithms, how you come up with different algorithms for same problems. In short, despite of all its shortcomings, [bubble sort](http://java67.blogspot.sg/2012/12/bubble-sort-in-java-program-to-sort-integer-array-example.html) is still the most popular algorithm.  
  
In this tutorial, we will learn *how bubble sort works*, complexity and performance of bubble sort algorithm,  implementation and source code in Java and a step by step example of bubble sort.

### How Bubble Sort Algorithm works

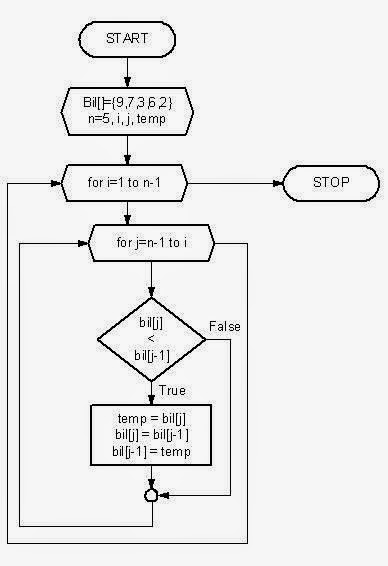
If you are the one who focus on names, then you might have got an idea how bubble sort works. Just like a bubble comes up from water, in bubble sort smallest or largest number, depending upon whether you are sorting array on ascending or descending order, bubbles up towards start or end of the array. We need at least N pass to sort the array completely and at the end of each pass one elements are sorted in its proper position. You can take first element from array, and start comparing it with other element, [swapping](http://javarevisited.blogspot.sg/2013/02/swap-two-numbers-without-third-temp-variable-java-program-example-tutorial.html) where it's lesser than the number you are comparing. You can start this comparison from start or from end, as we have compared elements from end in our bubble sort example. It is said that a picture is worth more than a thousand word and it's particularly true in case of understanding sorting algorithm. Let' see an *step by step example to sort array using bubble sort*, as I said after each pass largest number is sorted.

[](http://3.bp.blogspot.com/-rzUTGx1fUfY/U-zCFncNoRI/AAAAAAAABx4/YBwz3fXou9A/s1600/Java%2BBubble%2BSort%2BExample.png)

In this [array](http://javarevisited.blogspot.sg/2013/11/java-array-101-for-programmers-and.html), we start from index 0, which is 5 and starts comparing elements from start to end. So first element we compare 5 is 1, and since 5 is greater than 1 we swap them ( because ascending order sorted array will have larger number towards end). Next we compare 5 to 6, here  no swapping because 6 is greater than 5 and it's on higher index than 5. Now we compare 6 to 2, again we need swapping to move 6 towards end. At the end of this pass 6 reaches (bubbles up) at the top of the array. In next iteration 5 will be sorted on its position and after n iteration all elements will be sorted. Since we compare each element with another, we need two for loops and that result in complexity of O(n^2).

### FlowChart of Bubble Sort Algorithm

Another cool way to understand an algorithm is to draw it's flowchart. It will walk through each iteration in loop and how decisions are made during algorithm execution. Here is flowchart of our bubble sort algorithm, which complements our implementation of this sorting algorithm.

[](http://2.bp.blogspot.com/-uB94SH5WlZ0/U-zFpfRb6EI/AAAAAAAAByE/SyemBWPWRBU/s1600/Bubble%2BSort%2BFlowchart%2Bin%2BJava.jpg)

Here we have integer array {9, 7, 3, 6, 2} and start with four variable i, j, temp and array length which is stored in variable n. We have two for loop, outer loop runs from 1 to n-1. Our inner loop runs from n-1 to i. Many programmer make mistake here, if you start outer loop with second element than make sure to use j>=i condition on inner loop, or if you start with first element e.g. i=0, make sure you use j>i to avoid [ArrayIndexOutOfBound exception](http://javarevisited.blogspot.sg/2014/05/exception-in-thread-main-arrayindexoutofboundsexception-java.html). Now we compare each element and swap them to move smaller element towards front of array. As I said depending upon your navigation direction either largest element will be sorted at highest index in first pass or smallest element will be placed in lowest index. In this case, after first pass, smallest number will be sorted. This loop runs until j>=i after than it finishes and i becomes i + 1. This whole process repeats until outer loop is finished and that time your array is sorted. In flowchart, a diamond box is used for decision making, which is equivalent of if-else statement in code. You can see here decision box is inside inner loop, which means we do N comparison in each iteration, totals to NxN comparisons.

### Complexity and Performance of Bubble Sort Algorithm

As I said before compared to other sorting algorithm like quicksort, merge sort or shell sort, bubble sort performs poorly. These algorithm has average case complexity of O(NLogN), while average case complexity of bubble sort O(n^2). Ironically in best case bubble sort do better than [quicksort](http://java67.blogspot.com/2014/07/quicksort-algorithm-in-java-in-place-example.html) with O(n) performance.  Bubble sort is three times slower than quicksort or mergesort even for n = 100 but it's easier to implement and remember. here is the summary of bubble sort performance and complexity :  
  
Bubble sort Worst case performance       O(n^2)  
Bubble sort Best case performance         O(n)  
Bubble sort Average case performance    O(n^2)  
  
You can further explore insertion sort and selection sort, which also does sorting in similar time complexity. By the you can not only sort the array using bubble sort but ArrayList or any other collection class as well. Though you should really use Arrays.sort() or Collections.sort() for those purpose.

### Bubble Sort Implementation in Java

here is the Java program to implement bubble sort algorithm using Java programming language. Don't surprise with import of java.util.Array, we have not used it's sort method here, instead it is used to [print arrays in readable format](http://javarevisited.blogspot.sg/2012/12/3-example-to-print-array-values-in-java.html). I have created a swap function to swap numbers and improve readability of code, if you don't like you can in-line the code in the swap method inside if statement of inner loop. Though I have used main method for testing, as it demonstrate better, I would suggest you to write some unit test case for your bubble sort implementation. If you don't know how to do that, you can see this [JUnit tutorial](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html).

import java.util.Arrays;

/\*\*

\* Java program to implement bubble sort algorithm and sort integer array using

\* that method.

\*

\* @author Javin Paul

\*/

public class BubbleSort{

public static void main(String args[]) {

bubbleSort(new int[] { 20, 12, 45, 19, 91, 55 });

bubbleSort(new int[] { -1, 0, 1 });

bubbleSort(new int[] { -3, -9, -2, -1 });

}

/\*

\* This method sort the integer array using bubble sort algorithm

\*/

public static void bubbleSort(int[] numbers) {

System.out.printf("Unsorted array in Java :%s %n", Arrays.toString(numbers));

for (int i = 0; i < numbers.length; i++) {

for (int j = numbers.length -1; j > i; j--) {

if (numbers[j] < numbers[j - 1]) {

swap(numbers, j, j-1);

}

}

}

System.out.printf("Sorted Array using Bubble sort algorithm :%s %n",

Arrays.toString(numbers));

}

/\*

\* Utility method to swap two numbers in array

\*/

public static void swap(int[] array, int from, int to){

int temp = array[from];

array[from] = array[to];

array[to] = temp;

}

}

Output

Unsorted array in Java : [20, 12, 45, 19, 91, 55]

Sorted Array using Bubble sort algorithm : [12, 19, 20, 45, 55, 91]

Unsorted array in Java : [-1, 0, 1]

Sorted Array using Bubble sort algorithm : [-1, 0, 1]

Unsorted array in Java : [-3, -9, -2, -1]

Sorted Array using Bubble sort algorithm : [-9, -3, -2, -1]

### How to improve Bubble Sort Algorithm

In interview one of the popular follow-up question is how do you improve a particular algorithm, and Bubble Sort is no different than that. If you wrote bubble sort like the one we have shown here, interviewer will definitely going to ask about how do you improve your bubble sort method. In order to improve any algorithm, you must understand how each step of that algorithm works, then only you will be able to spot any deficiency in code. If you follow the tutorial, you will find that [array is sorted by moving elements to their correct position](http://java67.blogspot.sg/2014/08/4-examples-to-sort-array-in-java.html). In worst case situation if array is reverse sorted then we need to move every element, this will require n-1 passes, n-1 comparison in each pass and n-1 exchanges, but how about best case if array is already sorted, our existing bubble sort method is still going to take n-1 pass, same number of comparison but no exchange. If you observe carefully, you will find that after one pass through the array, the largest element will moved to the end of the array, but many other elements also moved toward their correct positions, as bubbles move toward the water’s surface. By leveraging this property, you can deduce that during a pass, if no pair of consecutive entries is out of order, then the array is sorted. Our current algorithm is not taking advantage of this property. If we track exchanges then we can decide whether additional iteration over array is needed or not. Here is an i*mproved version of Bubble Sort algorithm*, which will only take 1 iteration and n-1 comparison in best case, when array is already sorted. This will also improve Bubble sort's average case performance, as compared to our existing method which will always take N - 1 passes.

/\*

\* An improved version of Bubble Sort algorithm, which will only do

\* 1 pass and n-1 comparison if array is already sorted.

\*/

public static void bubbleSortImproved(int[] number) {

boolean swapped = true;

int last = number.length - 2;

// only continue if swapping of number has occurred

while (swapped) {

swapped = false;

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

if (number[i] > number[i + 1]) {

// pair is out of order, swap them

swap(number, i, i + 1);

swapped = true; // swapping occurred

}

}

// after each pass largest element moved to end of array

last--;

}

}

Now let's test this method for two input, one in which array is sorted (best case) and other on which only one pair is out of order. If we pass int array {10, 20, 30, 40, 50, 60} to this method,  initially will go inside while loop and make swapped=false. Then it will go inside for loop. when i =0 it will compare number[i] to number[i+1] i.e. 10 to 20 and check if 10 > 20, since it's not it will not go inside if block and no swapping will be occurred. When i=1, it will compare 20 > 30 again no swapping, next when i =2 30> 40 which is false so no exchange again, next i =3 so 40> 50, which is again false, so no swapping. Now last pair comparison i=4, it will compare 50 > 60 again this is false, so control will not go inside if block and no exchange will be made. Because of that swapped will remain false and control will not go inside while loop again. So you know that your [array is sorted just after one pass](http://javarevisited.blogspot.sg/2012/01/sort-array-in-java-ascending-and.html).  
  
Now consider another example, where just one pair is out of order, let's say String array names = {"Ada", "C++", "Lisp", "Java", "Scala"}, here only one pair is out of order e.g. "Lisp" should come after "Java". Let's see how our improved bubble sort algorithm work here. In first pass, comparison will continue without exchange until we compare "Lisp" to "Java", here "Lisp".compareTo("Java") > 0 will become true and swapping will occur, which means Java will go to the Lisp place, and Lisp will take Java's place. this will make boolean variable swapped=true, Now in last comparison on this pass, we compare "Lisp" to "Scala" and again no exchange. Now we will reduce last index by 1 because Scala is sorted at last position and will not participate  further. But now swapped variable is true, so control will again go inside while loop, and for loop but this time no exchanged will be made so it will not take another pass. Our array is now sorted in just two pass compared to N-1 pass of earlier implementation. This bubble sort implementation is much better and even perform better than Selection sort algorithm in average case because, now sorting is not proportional to total number of elements but only with number of pairs which are out-of-order.  
  
By the way to sort String array using Bubble Sort, you need to overload BubbleSortImproved() method to accept String[] and also need to use compareTo() method to [compare two String object lexicographically](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html). Here is Java program to sort String array using Bubble Sort :

import java.util.Arrays;

class BubbleSortImproved {

public static void main(String args[]) {

String[] test = {"Ada", "C++", "Lisp", "Java", "Scala"};

System.out.println("Before Sorting : " + Arrays.toString(test));

bubbleSortImproved(test);

System.out.println("After Sorting : " + Arrays.toString(test));

}

/\*

\* An improved implementation of Bubble Sort algorithm, which will only do

\* 1 pass and n-1 comparison if array is already sorted.

\*/

public static void bubbleSortImproved(String[] names) {

boolean swapped = true;

int last = names.length - 2;

// only continue if swapping of number has occurred

while (swapped) {

swapped = false;

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

if (names[i].compareTo(names[i + 1]) > 0) {

// pair is out of order, swap them

swap(names, i, i + 1);

swapped = true; // swapping occurred

}

}

// after each pass largest element moved to end of array

last--;

}

}

public static void swap(String[] names, int fromIdx, int toIdx) {

String temp = names[fromIdx]; // exchange

names[fromIdx] = names[toIdx];

names[toIdx] = temp;

}

}

Output:

Before Sorting : [Ada, C++, Lisp, Java, Scala]

After Sorting : [Ada, C++, Java, Lisp, Scala]

### Which one is better Selection Sort vs Bubble Sort?

Though both Selection Sort and Bubble sort has complexity of O(n^2) in worst case. On average, we expect the bubble sort to perform better than Selection sort, because bubble sort will finish sorting sooner than the selection sort due to more data movements for the same number of comparisons, because *we compare elements in pair on Bubble Sort*. If we use our improved implementation Bubble Sort then a boolean test to not enter on while loop when array gets sorted will also help. As I said, The worst case of the bubble sort happens when the original array is in descending order, while in best case, if the original array is already sorted, the bubble sort will perform only one pass whereas the selection sort will perform N - 1 passes. Given this, I think on average Bubble sort is better than Selection sort.  
  
  
That's all about **Bubble Sort in Java**. We have learned *how bubble sort algorithm works* and how do you implement it in Java. As I said, it is one of the simplest sorting algorithm and very easy to remember, but also it doesn't have any practical use apart from academics and in data structure and algorithm training classes. It's worst case performance is quadratic which means it not suitable for large array or list. If you have to use bubble sort, it's best suited for small, already sorted array in which case it has to very few swapping and it's performance is in O(n). If you love algorithms, you can see this problem of [finding cycle on linked list](http://javarevisited.blogspot.sg/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html).

Read more: <http://javarevisited.blogspot.com/2014/08/bubble-sort-algorithm-in-java-with.html#ixzz4esWRRTu4>

# How to reverse Integer in Java - LeetCode Solution

LeetCode has a problem to reverse digits of an integer number without using any library method like [reverse() method of StringBuffer](http://java67.blogspot.sg/2012/12/how-to-reverse-string-in-java-stringbuffer-stringbuilder.html). In LeetCode, you can solve this problem with many different languages e.g. Java, C, C++, C#, Python, Ruby and even JavaScript. BTW, in the article, we will learn how to solve this problem in Java. Before approaching solution let's first read the problem statement :  
  
Reverse digits of an integer.  
  
Example 1: x = 123, return 321  
Example 2: x = -123, return -321  
  
The problem looks simple but it's not simple there are many things you need to consider in order to produce a solution which is accepted by LeetCode, which has thousands of test cases to test your solution.  
  
  
For example, you need to consider not just about positive integer but also about negative integer. Remember, a positive integer can also be written using + sign, so you need to handle that as well. If the integer's last digit is 0, what should the output be? i.e. cases such as 10, 100.  
  
If the return type of method is an integer then you can simply return 1, it's perfectly Ok, but if the return type of method is String then you may need to return 001 or 0001. For the purpose of this solution, we expect our method to return an integer, so 1 is fine.

## Java Solution : Reverse Integer without library method

Here is my Java program to solve this problem of reversing integer number without using any direct library method. The crux of this problem is how to use division and modulo operator in Java to get the last digit of a number and get rid of the last digit as well.   
  
If you remember, I have shared this trick before when I was explaining [how to use modulo operator in Java](http://java67.blogspot.sg/2014/11/modulo-or-remainder-operator-in-java.html). This is a really neat trick and will help you to solve many programming problems where you need to divide numbers into individual digits. When you divide a number by 10, you get rid of the last digit, for example, 211/10 will give you 21, and 21/10 will give you 2, so you got rid of last 2 digits by dividing your number by 10 twice.  
  
Similarly, you can use number modulo 10 to get the last digit of the number, for example, 221%10 will return 1, which is the last digit and 22%10 will return 2, which is the last digit of 22. You can apply this logic until you processed the last digit. Now the question comes, how do you arrange those digits in reverse order? Well, you can use just opposite i.e. multiplication and addition to creating a new number with digits of the old number in reverse order.  I have used following logic to assemble digits into reverse order :

reverse = reverse \* 10 + lastDigit;

You can see by multiplying number by 10 you increase number of digits by 1 and then add last digit. For negative numbers, we multiply it by -1 to first make it positive and then apply same logic, while returning number we just multiply it by -1 again to convert the reversed number into negative.

import java.util.Scanner;

/\*\*

\* Java Program to reverse Integer in Java, number can be negative.

\* Example 1: x = 123, return 321

\* Example 2: x = -123, return -321

\*

\* @author Javin Paul

\*/

public class ReverseInteger{

public static void main(String args[]) {

int input = 5678;

int output = reverseInteger(5678);

System.out.println("Input : " + input + " Output : " + output);

}

/\*

\* Java method to reverse an integer value. there are couple of corner cases

\* which this method doesn't handle e.g. integer overflow.

\*/

public static int reverseInteger(int number) {

boolean isNegative = number < 0 ? true : false;

if(isNegative){

number = number \* -1;

}

int reverse = 0;

int lastDigit = 0;

while (number >= 1) {

lastDigit = number % 10; // gives you last digit

reverse = reverse \* 10 + lastDigit;

number = number / 10; // get rid of last digit

}

return isNegative == true? reverse\*-1 : reverse;

}

}

Result :

Input : 5678 Output : 8765

You can see that output is the just reverse of input. The first digit has exchanged position with last digit, second with second last and so on.  
  
By the way, if you are solving LeetCode problems as part of your interview preparation then you can also see [Programming Interviews Exposed](http://www.amazon.com/dp/1118261364/?tag=javamysqlanta-20) and [Cracking the Coding Interview](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20), two of the most useful books for preparing programming job interviews. You will learn more in a short time.

Read more: <http://www.java67.com/2015/08/how-to-reverse-integer-in-java-leetcode-solution.html#ixzz4esWjdJno>

**List of Java programming interview questions : programming concepts**  
**Java programs for practice**

1. [Print prime numbers?](http://instanceofjavaforus.blogspot.in/2014/12/program-to-print-prime-numbers-in-java.html)
2. [Format text using printf() method in java](http://www.instanceofjava.com/2017/03/java-printf-formatting-table-double.html)
3. [Java program to Convert arraylist to array](http://www.instanceofjava.com/2017/03/convert-arraylist-to-array-in-java-with.html)
4. [Finalize() method in java with example program](http://www.instanceofjava.com/2017/03/finalize-keyword-method-in-java-example.html)
5. [5 Different ways to print arrays in java](http://www.instanceofjava.com/2017/03/how-to-print-array-in-java-for-loop.html)
6. [Hibernate Criteria Query Language (HCQL)](http://www.instanceofjava.com/2016/11/hcql-hibernate-criteria-query-language.html)
7. [How to format text using printf() method in java](http://www.instanceofjava.com/2017/03/java-printf-formatting-table-double.html)
8. [Final method in java with example program](http://www.instanceofjava.com/2017/03/java-final-method-with-example-program.html)
9. [Java mutliple choice questions with answers on constructors](http://www.instanceofjava.com/2016/12/java-mcq-questions-with-answers.html)
10. [What are the advantages and disadvantages of Hibernate over JDBC](http://www.instanceofjava.com/2016/10/advantages-and-disadvantages-hibernate.html)

1. [How to generate unique random numbers in java](http://www.instanceofjava.com/2016/08/random-number-generator-java-range.html)
2. [Java mcq with answers on this keyword](http://www.instanceofjava.com/2016/12/java-mcq-with-answers-on-this-keyword.html)
3. [Quicksort algorithm in java with example program](http://www.instanceofjava.com/2016/08/quicksort-example-in-java-recursion.html)
4. [Implementation of selection sort algorithm in java with Example program](http://www.instanceofjava.com/2016/08/selection-sort-in-java-example-program.html)
5. [Tower of hanoi recursive solution using Java](http://www.instanceofjava.com/2016/08/towers-of-hanoi-java-program-code.html)
6. [Creating array of objects in java example program](http://www.instanceofjava.com/2016/08/creating-array-of-objects-in-java.html)
7. [Finding Factorial of a Number in Java](http://www.instanceofjava.com/2016/08/factorial-program-in-java-example.html)
8. [Print Pascals triangle in java program](http://www.instanceofjava.com/2016/08/pascals-triangle-in-java-program.html)
9. [Deep copy](http://www.instanceofjava.com/2016/08/deep-copy-in-java-example-program.html)   vs  [Shallow copy](http://www.instanceofjava.com/2016/08/shallow-copy-java-example-program.html)
10. [8 different ways to convert int to String in java](http://www.instanceofjava.com/2016/08/how-to-convert-integer-to-string-in-java.html)
11. [Find top two maximum numbers in a array java](http://www.instanceofjava.com/2016/08/find-top-two-maximum-numbers-in-array.html)
12. [Exception handling in method overriding](http://www.instanceofjava.com/2016/07/exception-handling-in-method-overriding.html)
13. [How to run jsp program in eclipse using tomcat](http://www.instanceofjava.com/2016/08/how-to-run-jsp-program-in-eclipse-tomcat.html)
14. [Java interface programming questions](http://www.instanceofjava.com/2016/07/java-interface-programming-questions.html)
15. [Java program to remove vowels from string java](http://www.instanceofjava.com/2016/07/remove-vowels-from-string-java.html)
16. [Java programming interview questions and answers for experienced](http://www.instanceofjava.com/2016/07/java-programming-interview-questions.html)
17. [Java Program to reverse vowels in a string](http://www.instanceofjava.com/2016/07/reverse-vowels-of-string-java.html)
18. [Super keyword java programs for interview for freshers and experienced](http://www.instanceofjava.com/2016/07/super-java-programs-for-beginners.html)
19. [Java programming interview questions on this keyword part 1](http://www.instanceofjava.com/2016/07/java-programming-questions-this-keyword.html)
20. [Super keyword in  java inheritance](http://www.instanceofjava.com/2016/07/super-keyword-in-java-inheritance.html)
21. [Java programming interview questions on this keyword part 2](http://www.instanceofjava.com/2016/07/java-interview-programming-this-keyword.html)
22. [Java programs on this keyword part 3](http://www.instanceofjava.com/2016/07/this-keyword-interview-programs-practice.html)
23. [Servlets interview questions](http://www.instanceofjava.com/2015/05/java-servlet-interview-questions.html)
24. [Find shortest palindrome string](http://www.instanceofjava.com/2016/07/find-shortest-palindrome-in-string-java.html)
25. [Validate email using javascript](http://www.instanceofjava.com/2016/06/validate-email-address-javascript.html)
26. [Java XML parsing using  DOM Parser](http://www.instanceofjava.com/2016/06/java-xml-parsing-using-dom-parser.html)
27. [Dynamic polymorphism in Java](http://www.instanceofjava.com/2016/06/dynamic-polymorphism-in-java-example.html)
28. [Find missing numbers in an array](http://www.instanceofjava.com/2016/06/program-find-missing-numbers-in-array.html)
29. [Collection vs Collections](http://www.instanceofjava.com/2016/06/collection-vs-collections-in-java-with.html)
30. [Custom iterator in java](http://www.instanceofjava.com/2016/04/custom-iterator-in-java-example.html)
31. [Unreachable Blocks in java](http://www.instanceofjava.com/2016/04/unreachable-blocks-in-java-example.html)
32. [Get table cell data using JavaScript](http://www.instanceofjava.com/2016/05/javascript-get-table-row-td-cell-value.html)
33. [Enum in java](http://www.instanceofjava.com/2016/04/enum-in-java-example.html)
34. [3 different ways to print exception message in java](http://www.instanceofjava.com/2016/04/ways-to-find-details-of-exception-java.html)
35. [Can we have try without catch block in java](http://www.instanceofjava.com/2016/04/can-we-have-try-without-catch-in-java.html)
36. [toString() method in java with example program](http://www.instanceofjava.com/2016/06/tostring-method-in-java-example-program.html)
37. [Remove duplicates from arraylist without using collections](http://www.instanceofjava.com/2016/04/remove-duplicates-from-arraylist.html)
38. [Explain Hibernate Configuration file](http://www.instanceofjava.com/2016/03/hibernate-configuration-file-java.html)
39. [What happens if we place return statement in try catch blocks](http://www.instanceofjava.com/2016/03/return-statement-in-try-catch-block-java.html)
40. [Write a java program to convert binary to decimal](http://www.instanceofjava.com/2016/03/convert-binary-to-decimal-java-program.html)
41. [Java interview questions on exception handling](http://www.instanceofjava.com/2016/04/exception-handling-interview-questions.html)
42. [Java Program to convert Decimal to Binary](http://www.instanceofjava.com/2016/03/convert-binary-to-decimal-java-program_21.html)
43. [Is it possible to print message without using System.out.println()](http://www.instanceofjava.com/2016/03/print-without-using-system-out-prinltn.html)
44. [Java program to restrict a class from creating not more than three objects](http://www.instanceofjava.com/2016/03/restrict-class-creating-multple-objects.html)
45. [Java basic interview programs on this keyword](http://www.instanceofjava.com/2016/03/this-keyword-in-java-with-example.html)
46. [Java Program to Sort elements of Java ArrayList Example](http://www.instanceofjava.com/2016/03/how-to-sort-elements-of-arraylist-in-java.html)
47. [Interfaces allows constructors?](http://www.instanceofjava.com/2016/02/java-interface-constructor-example.html)
48. [Can we create static constructor in java](http://www.instanceofjava.com/2016/02/java-static-constructor-method.html)
49. [Super keyword interview questions java](http://www.instanceofjava.com/2016/03/super-keyword-in-java-example.html)
50. [Java interview questions on final keyword](http://www.instanceofjava.com/2016/03/java-interview-questions-final-keyword.html)
51. [Can we create private constructor in java](http://www.instanceofjava.com/2016/02/private-constructor-java-example.html)
52. [Java Program Find Second highest number in an integer array](http://www.instanceofjava.com/2016/02/find-second-highest-largest-number-java.html)
53. [Java interview programming questions on interfaces](http://www.instanceofjava.com/2016/03/java-interface-interview-questions.html)
54. [Top 15 abstract class interview questions](http://www.instanceofjava.com/2016/02/abstract-method-class-example-programs.html)
55. [Java interview Questions on main() method](http://www.instanceofjava.com/2015/08/java-interview-questions-on-main-method.html)
56. [Sort employee object by id in descending order using comparable and TreesSet](http://www.instanceofjava.com/2016/03/sort-descending-order-using-treeset-java.html)
57. [Top 20 collection framework interview Questions](http://www.instanceofjava.com/2015/07/collections-interview-questions-java.html)
58. [Java Interview Program to find smallest and second smallest number in an array](http://www.instanceofjava.com/2016/02/java-program-to-find-second-smallest.html)
59. [Java Coding Interview programming Questions : Java Test on HashMap](http://www.instanceofjava.com/2016/02/java-coding-interview-questions-hashmap.html)
60. [Explain java data types with example programs](http://www.instanceofjava.com/2016/03/data-types-in-java-examples.html)
61. [How to check internet connection using java](http://www.instanceofjava.com/2016/02/how-to-check-internet-conection-in-java.html)
62. [Constructor chaining in java with example programs](http://www.instanceofjava.com/2016/02/constructor-chaining-in-java-using-this-super.html)
63. [Top 10 Interview Programs and questions on method overriding in java](http://www.instanceofjava.com/2016/02/method-overriding-in-java-definition.html)
64. [Swap two numbers without using third variable in java](http://www.instanceofjava.com/2015/12/swap-two-numbers-without-using-third-variable-java.html)
65. [Find sum of digits in java](http://www.instanceofjava.com/2015/12/find-sum-of-digits-in-java-recursion.html)
66. [How to create immutable class in java](http://www.instanceofjava.com/2015/12/how-to-make-class-custom-immutable-in-java.html)
67. [AtomicInteger in java](http://www.instanceofjava.com/2015/12/atomic-integer-example-java-interview.html)
68. [Check Even or Odd without using modulus and division](http://www.instanceofjava.com/2015/03/even-or-odd-without-using-mod-division.html)
69. [String Reverse Without using String API](http://instanceofjavaforus.blogspot.in/2014/12/reverse-string-without-using-string.html)
70. [Find Biggest substring in between specified character](http://www.instanceofjava.com/2016/02/find-longest-substring-in-between-specified-string.html)
71. [Check string is palindrome or not?](http://instanceofjavaforus.blogspot.in/2014/12/check-whether-string-is-palindrome-or.html)
72. [Reverse a number in java?](http://instanceofjavaforus.blogspot.in/2014/12/reverse-number-program-in-java.html)
73. [Fibonacci series with Recursive?](http://instanceofjavaforus.blogspot.in/2014/12/fibonacci-series-using-recursion-in-java.html)
74. [Fibonacci series without using Recursive?](http://instanceofjavaforus.blogspot.in/2014/12/fibonacci-series-without-using.html)
75. [Sort the String using string API?](http://instanceofjavaforus.blogspot.in/2014/12/sort-string-using-string-api.html)
76. [Sort the String without using String API?](http://instanceofjavaforus.blogspot.in/2014/12/sorting-string-without-using-string-api.html)
77. [what is the difference between method overloading and method overriding?](http://instanceofjavaforus.blogspot.in/2014/12/what-is-difference-between-method.html)
78. [How to find largest element in an array with index and value ?](http://instanceofjavaforus.blogspot.in/2014/12/how-to-find-largest-element-in-array.html)
79. [Sort integer array using bubble sort in java?](http://www.instanceofjava.com/2015/12/bubble-sort-algorithm-technique-in-java.html)
80. [Object Cloning in java example?](http://instanceofjavaforus.blogspot.in/2014/12/object-cloning-in-java-example.html)
81. [Method Overriding in java?](http://instanceofjavaforus.blogspot.in/2014/12/method-overriding-example-program.html)
82. [Program for create Singleton class?](http://instanceofjavaforus.blogspot.in/2014/12/write-program-to-create-singleton-class.html)
83. [Print numbers in pyramid shape?](http://instanceofjavaforus.blogspot.in/2014/12/print-numbers-in-pyramid-shape.html)
84. [Check armstrong number or not?](http://instanceofjavaforus.blogspot.in/2014/12/check-armstrong-number-or-not.html)
85. [Producer Consumer Problem?](http://instanceofjavaforus.blogspot.com/2014/12/producer-consumer-problem.html)
86. [Remove duplicate elements from an array](http://instanceofjavaforus.blogspot.in/2014/12/remove-duplicates-from-array-java.html)
87. [Convert Byte Array to String](http://instanceofjavaforus.blogspot.in/2014/12/convert-byte-array-to-string.html)
88. [Print 1 to 10 without using loops](http://instanceofjavaforus.blogspot.in/2014/12/print-1-to-10-without-using-loop-in-java.html)
89. [Add 2 Matrices](http://instanceofjavaforus.blogspot.in/2015/01/add-two-matrices.html)
90. [Multiply 2 Matrices](http://instanceofjavaforus.blogspot.in/2015/01/multiply-two-matrices.html)
91. [How to Add elements to hash map and Display](http://www.instanceofjava.com/2014/12/how-to-add-elements-to-hash-map-and.html)
92. [Sort ArrayList in descending order](http://www.instanceofjava.com/2015/01/sort-arraylist-in-descending-order.html)
93. [Sort Object Using Comparator](http://www.instanceofjava.com/2014/12/sort-object-using-comparator.html)
94. [Count Number of Occurrences of character in a String](http://www.instanceofjava.com/2014/12/count-number-of-occurrences-of.html)
95. [Can we Overload static methods in java](http://www.instanceofjava.com/2015/06/can-we-overload-static-methods-in-java.html)
96. [Can we Override static methods in java](http://www.instanceofjava.com/2015/06/can-we-override-static-methods-in-java.html)
97. [Can we call super class static methods from sub class](http://www.instanceofjava.com/2016/02/can-we-call-superclass-static-method.html)
98. [Explain return type in java](http://www.instanceofjava.com/2015/06/return-type-statemet-in-java-example.html)
99. [Can we call Sub class methods using super class object?](http://www.instanceofjava.com/2015/07/calling-subclass-method-from-superclass.html)
100. [Can we Override private methods ?](http://www.instanceofjava.com/2015/07/can-we-override-private-method-in-java.html)
101. [Basic Programming Questions to Practice : Test your Skill](http://www.instanceofjava.com/2016/02/java-basic-programming-for-freshers-interview.html)
102. [Java programming interview questions on collections](http://www.instanceofjava.com/2016/03/java-collections-interview-questions.html)

Read Also:  
  
Java programming exercises for beginners   
  
[1. Top 15 Garbage Collection Interview Questions](http://www.instanceofjava.com/2015/03/garbage-collections-interview-questions.html)  
  
[2. Top 10 Oops Concepts Interview Questions](http://www.instanceofjava.com/2015/03/oops-concepts-interview-questions.html)  
  
[3. Top 15 Java Interview Questions on Constructors](http://www.instanceofjava.com/2015/04/java-interview-questions-on-constructors.html)  
  
[4. Top 10 Inheritance Interview Questions](http://www.instanceofjava.com/2015/04/top-10-inheritance-interview-questions.html)  
  
[5. Interview Programs on Strings](http://www.instanceofjava.com/2015/04/java-interview-programs-on-strings.html)  
  
[6. 10 Interesting Core Java Interview Coding Questions and Answers](http://www.instanceofjava.com/2015/05/10-tricky-core-java-interview-coding.html)  
  
[7. Top 20 Basic Java Interview Questions for Frehsers](http://www.instanceofjava.com/2015/06/java-basic-interview-questions-freshers.html)  
  
[8. Top 10 interview Question on Static keyword.](http://www.instanceofjava.com/2015/07/core-java-interview-questions-on-static.html)  
  
[9.Top 20 Java interview Questions on Increment and Decrement operators](http://www.instanceofjava.com/2015/07/increment-decrement-operators-interview.html)   
  
[10.Top 10 Interview Questions on main() method](http://www.instanceofjava.com/2015/08/java-interview-questions-on-main-method.html)  
  
[11. Top 12Java Experienced interview Programming Questions on Strings](http://www.instanceofjava.com/2015/09/java-2-3-years-experience-interview-programming-questions-on-strings.html)  
  
[12.Pattern Programs in java Part-1](http://www.instanceofjava.com/2015/09/pattern-program-in-java-displays-pyramid-rhombus-stars.html)  
 [13.Pattern Programs in java Part-2](http://www.instanceofjava.com/2015/09/pattern-programs-in-java-using-for-loop.html)  
  
[14.Pattern Programs in java Part-3](http://www.instanceofjava.com/2015/09/java-program-print-stars-in-diamond.html)