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#ixzz4KpMoypTX>