**Day11**

**Strings Questions**

1. What is String in Java? Is it a datatype?

The string is a final class in Java defined in java.lang package. You can assign a sequence of characters to a string variable. For example String name = "Gaurav";

No, String is not a datatype like int, char, or long.

When you assign a sequence of character to String variable, you are creating a string object.

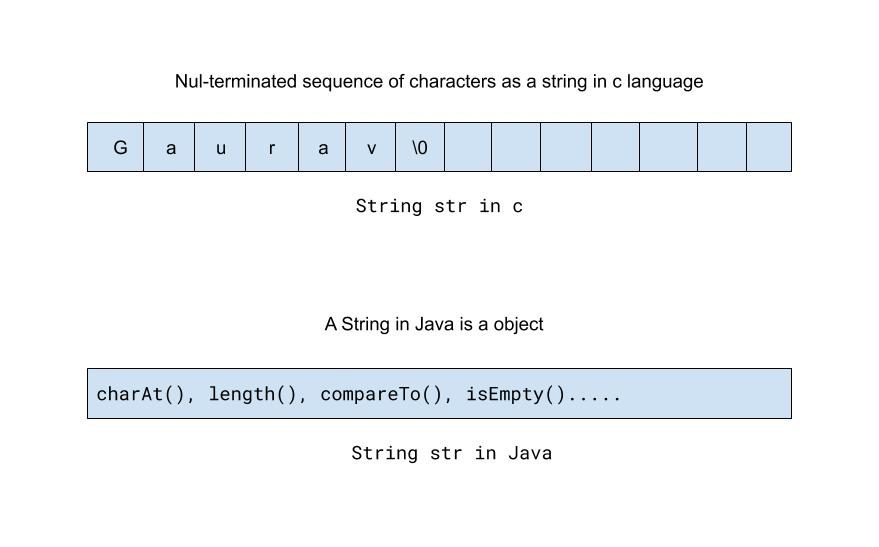
Every String literal is an instance of the String class, and its value can not be changed.

2. What is the difference between String in C language and String in Java?

If your resume contains something related to the ‘C’ language, they can ask you this question.

String in Java and C is completely different. In ‘C’ language String is a null-terminated character array.

In the image given below, I have shown the structure of the string in C and Java.



Showing String in C and Java

The string is more abstract in Java.

The string class comes with java.lang package and has lots of predefined methods that a programmer can use to operate on a string or get information about a String.

So String is more feature-rich in Java than C.

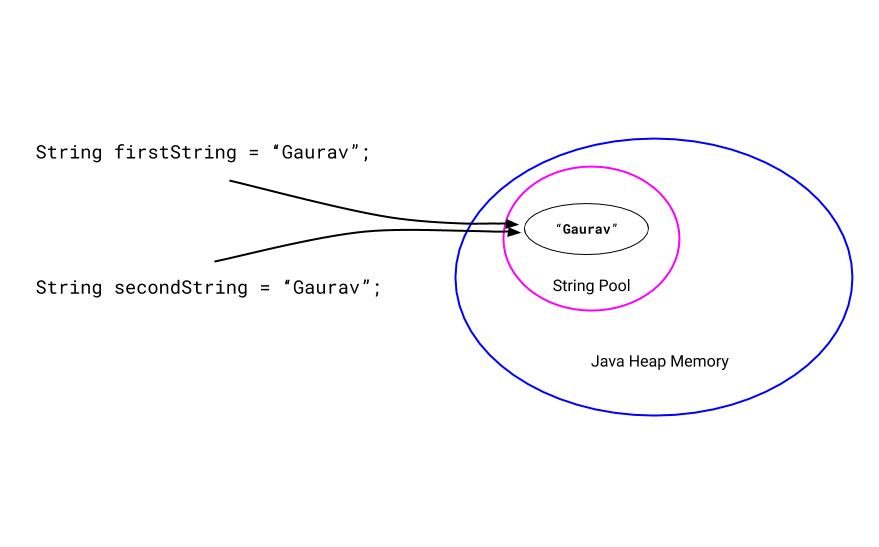
3. What is the String pool in Java?

The String pool is a special type of memory maintained by the JVM.

String pool is used to store unique string objects.

When you assign the same string literal to different string variables, JVM saves only one copy of the String object in the String pool, and String variables will start referring to that string object.

I have shown the pictorial explanation of the above sentence in the following diagram.



Two string variables pointing to the single string object from the string pool

The purpose of maintaining this special type of memory is memory optimization.

4. Why String is immutable?

In most of the Java Interviews, you will face this question. Why do you think Java language designers kept string immutable?

You can give the following reasons.

Java String pool is possible because the String is immutable.

If you assign the same string literal to many string variables, JVM will save only one copy of the string object in the Java string pool, and these variables will start referring to that string object.

If you were not asked about the String pool before this question, please give a little background about the string pool concept in Java. Please refer to the previous question.

Also, another reason can be Security. We know that almost every Java program contains a string, and it is used to save important data like usernames and passwords. So it should not be changed in-between. Otherwise, there will be a security problem.

5. How many objects will be created from the following code?



|  |  |
| --- | --- |
| 1  2  3 | String firstString = "Gaurav";  String secondString = "Gaurav";  String thirdString =  new String("Gaurav"); |

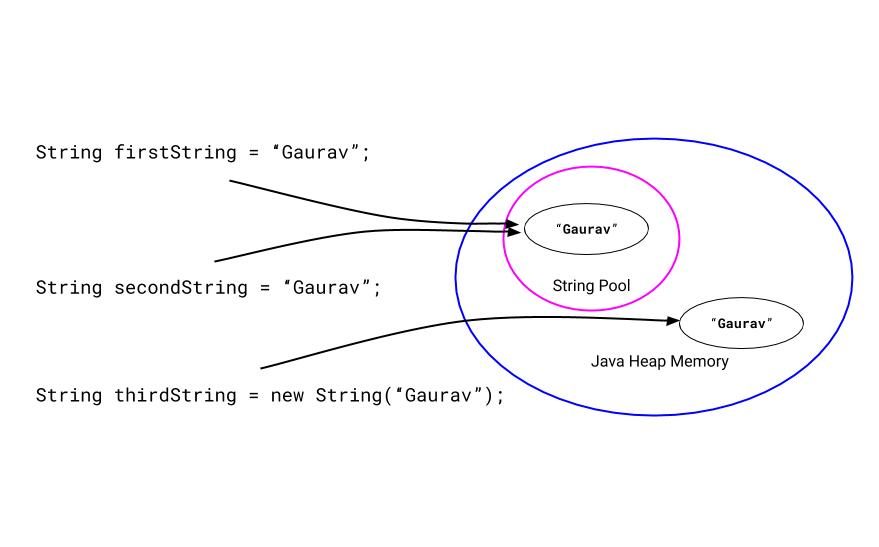
By seeing the above code, only two string objects will be created. The first two variables will refer to the same string object with the value "Gaurav". JVM uses the string pool concept to store only one copy of duplicate string objects to string constant pool.

But when we use a new keyword to create a new string, a new string object will be created and stored in the Java heap memory.

So for the third variable thirdString, a new string object will be created and stored in a Java heap space.

So there will be a total of two objects, one from the Java string pool and one from the Java heap memory.

Below, I have shown these two objects in the following diagram.



Showing the two string objects from the Java string pool and Java heap memory

6. What is the intern() method?

intern() method is used to add the unique copy of the string object to the Java string pool manually.

We know, when we create a string using a new keyword, it will be stored in the heap memory.

We can store the unique copy of that string object in the Java string pool using the intern() method.

When you do such a thing, JVM will check if the string object with the same value is present in the string pool or not.

If a string object with the same value is present, JVM will simply provide the reference of that object to the respective string variable.

If a string object with the same value is not present in the string pool, JVM creates a string object with the same value in the String pool and returns its reference to the string variable.

7. What is the difference between the String and StringBuffer?

The String is a final class in Java. The String is immutable. That means we can not change the value of the String object afterword.

Since the string is widely used in applications, we have to perform several operations on the String object. Which generates a new String object each time, and all previous objects will be garbage object putting the pressure on the Garbage collector.

Hence, the Java team introduced the StringBuffer class. It is a mutable String object, which means you can change its value.

The string is immutable, but the StringBuffer is mutable.

8. What is the difference between the StringBuffer and StringBuilder?

We know String is immutable in Java. But using StringBuffer and StringBuilder, you can create editable string objects.

When Java Team realizes the need for the editable string object, they have introduced the StringBuffer class. But all the methods of the StringBuffer class are synchronized. That means at a time, only one thread can access a method of the StringBuffer.

As a result, it was taking more time.

Latter, Java Team realizes that making all methods of the StringBuffer class synchronized was not a good idea, and they introduced a StringBuilder class. None of the methods of the StringBuilder class are synchronized.

Since all the methods of the StringBuffer class are synchronized, StringBuffer is thread-safe, slower, and less efficient as compared to StringBuilder.

Since none of the methods of the StringBuilder class is synchronized, StringBuilder is not thread-safe, faster, and efficient as compared to StringBuffer.

Also, you can check my detailed article on [difference between theStringBuffer and StringBuilder](https://coderolls.com/difference-between-stringbuffer-and-stringbuilder/).

9. Can we compare String using the == operator? What is the risk?

Yes, of course, we can compare String using the == operator. But when we are comparing string using the == operator, we are comparing their object reference, whether these string variables are pointing towards the same string object or not.

Most of the time, developers want to compare the content of the strings, but mistakenly they compare strings with == operator, instead of equals() method, which leads to an error.

Below, I have given a program, which shows the string comparison using the == operator and equals() method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | /\*  \* A java program showing the string comparison  \* using equals to operator ( == ).  \*/  public class StringCompareUsingEqualsOperator {    public static void main(String[] args) {    String firstString = "Gaurav";  String secondString = "Gaurav";    String thirdString =  new String("Gaurav");    System.out.print("Case 1 : ");  System.out.println(firstString == secondString); // true    System.out.print("Case 2 : ");  System.out.println(firstString == thirdString); // false    // Comparing strings using equals() method  System.out.print("Case 3 : ");  System.out.println(firstString.equals(thirdString)); // true  }    } |

The output of the above program will be:



|  |  |
| --- | --- |
| 1  2  3 | Case 1 : true  Case 2 : false  Case 3 : true |

In ‘case 1,’ we are comparing firstString and secondString using the equals to operator (==), since both the variables are pointing towards the same string object, it will print true.

In ‘case 2,’ we are comparing firstString and thirdString using the equals to operator i.e. == since both the variables are not pointing towards the same string object, it will print false.

You can see, for thirdString, we are using the new keyword, which creates a new object in Java heap memory.

In ‘case 3,’ we are comparing firstString and thirdString using equals() method. Even if both are the different string object, it has the same content hence it is printing true.

10. What are the ways to compare string?

We can compare strings using the equals() method, == operator and compareTo() method.

When we compare strings using the equals() method, we are comparing the content of the strings, whether these strings have the same content or not.

When we compare strings using the == operator, we are comparing the reference of the string, whether these variables are pointing to the same string object or not.

Also, we can compare string lexicographically (comparing strings by alphabetical order). We can use the compareTo() method to compare the strings lexicographically.

compareTo() method returns a negative integer, 0, or a positive integer.



|  |  |
| --- | --- |
| 1 | firstString.compareTo(secondString) |

If firstString is less than the secondString, it will return a negative integer. i.e firstString < secondString → returns a negative integer

If firstString is equal to the secondString it will return zero. i.e firstString == secondString → returns zero

If firstString is greater than the secondString, it will return a positive integer. i.e firstString > secondString → returns a positive integer

If you want to check string comparison in detail, please consider visiting [this article](https://coderolls.com/compare-strings-in-java/).

11. What is the use of the substring() method?

The substring() method in Java returns a ‘substring’ of the specified string.

The creation of the substring depends on the parameter passed to the substring() method.

There are two variants of the Substring method.

1. substring(int beginIndex)
2. substring(int beginIndex, int endIndex)

In the first method, we are just giving beinIndex parameter, while in the second variant we are giving both beginIndex and endIndex.

For the first variant, substring will be created from the beginIndex (**inclusive**) to the last character of the string.

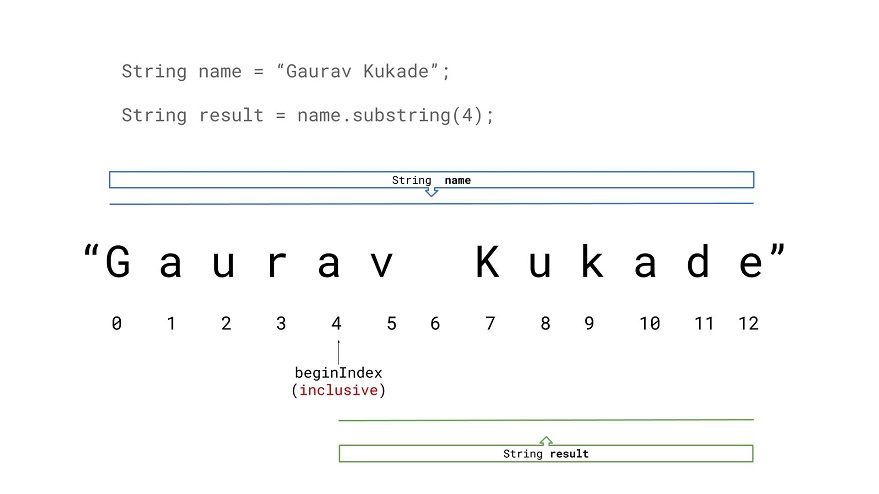
For the second variant, substring will be created from the beginIndex (**inclusive**) to the endIndex (**exclusive**).

See the following diagram to understand substring() method.

In the first diagram, I have the first variant of the substring() method.



|  |  |
| --- | --- |
| 1  2  3  4 | String name = "Gaurav Kukade";  String result = name.substring(4);    System.out.println(result); // it will print "av kukade" |

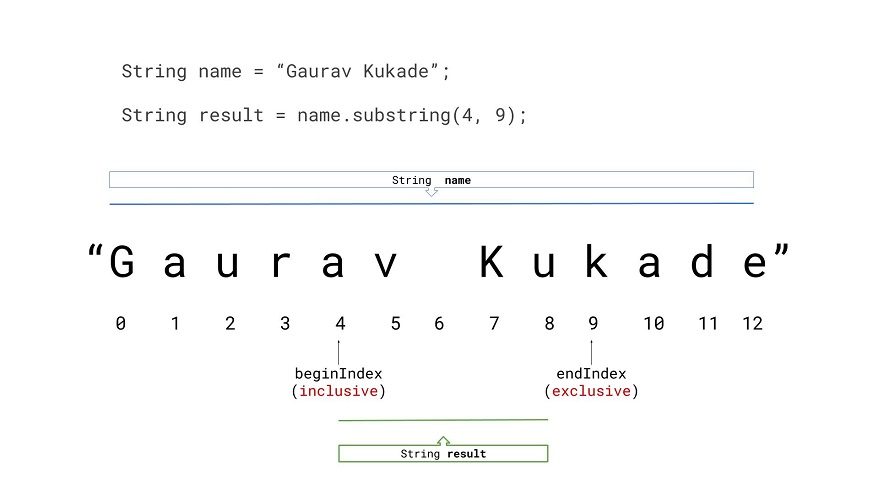


substring(int beginIndex) pictorial explanation

In the second diagram, I have the first variant of the substring() method.



|  |  |
| --- | --- |
| 1  2  3  4 | String name = "Gaurav Kukade";  String result = name.substring(4, 9);    System.out.println(result); // it will print "av ku" |



substring(int beginIndex, int endIndex) pictorial explanation

12. How to check if the String is empty?

Java String class has a special method to check if the string is empty or not.

isEmpty() method internally checks if the length of the string is zero. If it is zero, that means the string is empty, and the isEmpty() method will return true.

If the length of the string is not zero, then the isEmpty() method will return false.

13. What is the format() method in Java String? What is the difference between the format() method and the printf() method?

format() method and printf() method both format the string. The only difference is that the format() method returns the formatted string, and the printf() method prints the formatted string.

So when you want the formatted string to use in the program. you can use the format method. And when you want to just print the formatted string, you can use the printf() method.

14. Can you say String is ‘thread-safe’ in Java?

Yes, we can say that the string is thread-safe.

As we know, String is immutable in Java. That means once we created a string, we can not modify it afterword. Hence there is no issue of multiple threads accessing a string object.

15. Why most of the time string is used as HashMap key?

The string is immutable. So one thing is fixed that it will not be changed once created.

Hence the calculated hashcode can be cached and used in the program. This will save our effort for calculating the hashcode again and again. So, a string can be used efficiently than other HashMap key objects.

16. Can you convert String to Int and vice versa?

Yes, you can convert string to int and vice versa.

You can convert string to an integer using the valueOf() method and the parseInt() method of the Integer class.

Also, you can convert an integer to string using the valueOf() method of the String class.

Below, I have given a program which shows the string to integer and integer to string conversion.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | /\*  \* A Java program to convert String to Integer and vice versa.  \*  \* We are using the valueOf() method and parseInt() method of  \* the wrapper classes.  \*  \*/  public class Conversion{  public static void main(String [] args){    String str = "1254";    int number = 7895;    // convert string to int using Integer.parseInt() method  int parseIntResult1 = Integer.parseInt(str);    // convert string to int using Integer.valueOf() method  int valueOfResult1 = Integer.valueOf(str);    System.out.println("Converting String to Integer:");  System.out.println("Using the Integer.parseInt() method : "+parseIntResult1);  System.out.println("Using the Integer.valueOf() method : "+valueOfResult1);    System.out.println("\n");  // convert integer to string using String.valueOf() method  String valueOfResult2 = String.valueOf(number);    System.out.println("Converting Integer to String :");  System.out.println("Using the String.valueOf() method : "+valueOfResult2);    }  } |

The output of the above program will be,



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Converting String to Integer:  Using the Integer.parseInt() method : 1254  Using the Integer.valueOf() method : 1254      Converting Integer to String :  Using the String.valueOf() method : 7895 |

17. What is the split() method?

The split method is used to split the string based on the provided regex expression.

The Signature of the split method is



|  |  |
| --- | --- |
| 1 | public String[] split(String regex) |

This method will return an array of the split substrings.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\*  \* A Java program showing the uses of split method.  \*  \*/  public class SplitExample {    public static void main(String[] args) {  String name = "My, name, is ,Gaurav!";    String [] substringArray = name.split(",");    for(String substring : substringArray) {  System.out.print(substring);  }  }    } |

The output of the above program will be



|  |  |
| --- | --- |
| 1 | My name is Gaurav! |

18. What is the difference between "Gaurav Kukade".equals(str) and str.equals("Gaurav Kukade")?

Both look the same, and it will check if the content of the string variable str is equal to the string "Gaurav Kukade" or not.

But their behavior will change suddenly when a string variable str = null. The first code snippet will return false, but the second code snippet will through a NullPointerExpection.

Below I have given a program which uses both ways to compare string using equals() method.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\*  \* A Java program which checks  \* both ways of string comparison using equals() method.  \*/    public class StringExample {    public static void main(String[] args) {    String str = "Gaurav Kukade";    System.out.println("Gaurav Kukade".equals(str)); // true    System.out.println(str.equals("Gaurav Kukade")); // true  }    } |

The output of the above program will be



|  |  |
| --- | --- |
| 1  2 | true  true |

It is print true both times because the content of both strings is equal to each other.

Now, we will check a program where the str=null



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /\*  \* A Java program which checks  \* both ways of string comparison using equals() method.  \*  \* This program throws a NullPointerException at second print statement.  \*/    public class StringNullExample {    public static void main(String[] args) {    String str = null;    System.out.println("Gaurav Kukade".equals(str)); // false    System.out.println(str.equals("Gaurav Kukade")); // NullPointerException  }    } |

The output of the above program will be



|  |  |
| --- | --- |
| 1  2  3 | false  Exception in thread "main" java.lang.NullPointerException  at StringNullExample.main(StringNullExample.java:14) |

We can see the above output, for first code snippet it is print false but for the second code snippet, it is throwing NullPointerException.

It is one of the most important tricks to avoid the null pointer exception in java string. So this question is important.

19. Can we use a string in switch case in java?

Yes, from Java 7 we can use String in switch case.

Below, I have given a program that shows the use of the string in switch case.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | /\*  \* A Java program showing the use of  \* String in switch case.  \*/  public class StringInSwitchExample  {      public static void main(String[] args)      {          String str = "two";          switch(str)          {              case "one":                  System.out.println("January");                  break;              case "two":                  System.out.println("February");                  break;              case "three":                  System.out.println("March");                  break;              default:                  System.out.println("Invalid month number");          }      }  } |

The output of the above program will be



|  |  |
| --- | --- |
| 1 | February |

20. How string concatenation using the + operator works in Java?

+ operator is the only overloaded operator. You can use it for both adding two numbers as well as for string concatenation purposes.

If you are using the Java version 1.5 or above, string concatenation internally uses the append() method of the StringBuilder. And for versions lower than 1.5, it uses the append() method of the StringBuffer class.

If you think any other question should be in the list above, please write them down in the comment section below.