Java Strings:

Q1 what is String?

String is a class in java which is present in **java.lang** package. According to **oracle** docs.
The String class represents **character** strings.
Strings are constant, their values cannot be changed after they are created.

Q2 Is String immutable in java?

Yes, String class is **immutable** in java. Immutable means once the object is created, its value **cannot** be changed.

Q3 Is String a keyword in java?

No, String is not a keyword in java.

Q4 How many objects are created using *new* keyword?

String str = new String("Java Hungry");
Two objects are created by the above statement. One object in the heap memory and one object in the String constant pool.

Q6 How to convert String to char Array?

You can convert String to char Array using **toCharArray()** method.

Q7 How many different ways you can create a String object?

You can create a String object using **two** ways. First is using **new** operator and second is using string **literal**. Objects created using new operator are stored in the **heap** memory while string literals are stored in the **string constant pool**.

String str = "javahungry"; // **String literal**
String str = new String("javahungry"); // **using new operator**

Q8 Are String thread-safe in java?

As we know String objects are **immutable**. It means they **are** thread-safe also.

Q9 which class is final among String, StringBuilder and StringBuffer?

All are final classes.

Q10 Is String primitive type or object (derived) type in java?

String is **object** (derived) type in java.

Q11 Can we use String in switch statement?

Yes, you can use String in switch statement in java **7**. Prior to java 7, you had to use if-else statements to achieve the task.

Q12 what is the difference between String and StringBuffer in java?

String

String is immutable (once created **cannot** be changed) object. The object created as a String is stored in the **Constant String Pool.**

Every immutable object in Java is **thread-safe**, that implies **String** is also thread-safe. String **cannot** be used by two threads simultaneously.

String once assigned cannot be changed.

```
String  demo = "hello" ;

// The above object is stored in constant string pool and its value cannot be modified.

demo="Bye" ;

//new "Bye" string is created in constant pool and referenced by the demo variable

// "hello" string still exists in string constant pool and its value is not overridden but we lost reference to the "hello" string
```

StringBuffer:

StringBuffer is **mutable** means one can change the value of the object. The object created through StringBuffer is stored in the **heap**. StringBuffer has the same methods as the StringBuilder, but each method in StringBuffer is **synchronized** that is StringBuffer is **thread-safe**.

Due to this it does not allow **two** threads to simultaneously access the same method. Each method can be accessed by one thread at a time.

But being thread-safe has **disadvantages** too as the performance of the StringBuffer hits due to thread safe property. Thus **StringBuilder** is **faster** than the StringBuffer when calling the same methods of each class.

StringBuffer value **can** be changed, it means it can be assigned to the new value. Nowadays it's a most common interview question, the differences between the above classes.

String Buffer can be **converted** to the string by using toString() method.

```
StringBuffer demo1 = new StringBuffer("Hello") ;

// The above object stored in heap and its value can be changed.

demo1 = new StringBuffer("Bye");

//Above statement is right as it modifies the value which is allowed in the StringBuffer.
```

StringBuilder:

StringBuilder is **same** as the StringBuffer, that is it stores the object in **heap** and it can also be modified. The main difference between the StringBuffer and StringBuilder is that StringBuilder is **not thread safe**.

StringBuilder is **fast** as it is not thread safe.

```
StringBuilder demo2= new StringBuilder("Hello");

// the above object too is stored in the heap and its value can be modified

demo2=new StringBuilder("Bye");

// above statement is right as it modifies the value which is allowed in the StringBuilder
```

1. StringBuilder is **not** thread-safe. StringBuffer **is** thread-safe.

2. StringBuilder is **not** synchronized and StringBuffer **is** synchronized.

3. StringBuilder is **faster** while StringBuffer is **slower** as it is thread-safe.

Q13 Explain the difference between below statements:

```
String str = new String("abc");

String  str =  "abc";
```

In the first statement, **String str = new String("abc");**
JVM will create **one** object in the **heap** memory. Another **object** in the **String constant pool**, if the object is **not** present. Otherwise, if present in the String constant pool, it will return the **reference** to it.
In the second statement, **String str = "abc";**
JVM checks for the string "abc" in the **String constant pool**. If the string is **not** present in the constant pool then it will **create** a new String object and stores it in pool.
If the string "abc" is found in string constant pool, then it simply **creates** a reference to it but does not create a new object.

Q14 How many objects will be created for the following code:

```
String str1 = "abc";

String str2 = new String("abc");
```

Two objects are created. Object created using ***new*** operator is stored in the **heap** memory (str2).
Object created using String **literal** str1 is stored in the **string constant pool**.

Q15 How many objects will be created for the following code:

```
String str1 = "abc";

String str2 = "abc";
```

Only **one** object is created. String str1 will create a new object in **String constant pool**, while String str2 will create a reference to the String str1.

Q16 How many objects will be created for the following code:

```
String str1 = new String("abc");

String str2 = new String("abc");
```

Three objects are created. For the first statement (str1) **two** objects are created one in **String constant pool** and one in **heap** memory.
But for the **second** statement (str2), compulsory 1 **new** object is created in **heap** memory but no new object is created in string constant pool as it is already present.
Hence, a total of 2+1 = **3** objects are created.

Q17 What is String intern() method?

According to oracle docs,
When the **intern()** method is invoked, if the String constant pool already **contains** a **string** equal to the **String** object as determined by the equals(Object) method, then the string from the pool is **returned**.
Otherwise the String object is **added** to the pool and a **reference** to the String object is returned.
The task of intern() method is to put String (which is passed to the intern method) into string constant pool.

Q18 what are mutable and immutable objects in java?

Mutable objects value can be changed. **StringBuilder** and **StringBuffer** are the examples of the mutable objects. **Immutable** objects value cannot be changed once created. **String** is an immutable class in java.

Q19 Why is String immutable in java?

There are various reasons to make String immutable in java.

1. **Security**: String is made immutable to help increase the Security. Sensitive data like **username**, **password** can be stored as the Strings **can't** be modified once created.
2. **Caching**: when compiler optimizes your String objects, it sees that if **two** objects have **same** value (a="test", and b="test") and thus you need only **one** string object (for both a and b, these two will point to the same object).
3. **Class loading**: String objects are used for Class loading. It is possible that **wrong** class has been loaded in the JVM, if the String is **mutable** i.e modifiable.
3. **Thread Safe:** Immutable Strings are **thread-safe**. Synchronization is not required when we use them in the multithreading environment.

Q20 How will you create an immutable class in java?

You can create immutable class in java by implementing below points:

1. Make the class **final** so it **cannot** be extended (inherited)
2. Make all fields **private** so one **cannot** access them from outside the class.
3. Do not provide **setter** methods for the variables.
4. Declare all mutable fields as **final** so that its value can be assigned only once.

Q21 How will you create mutable String objects in java?

As we have discussed, by using StringBuffer and StringBuilder objects.

Q22 what is the difference between Java String and C, C++ Strings?

In **C** and **Java** both programming language treat String object as **char** Array.
Java String is an object while C String is a NULL terminated character array. Java String object allows calling different methods like toUpperCase(), length(), substring().

Q23 Why String is mostly used as a key in HashMap class?

String is mostly used as a key in HashMap class because it **implements equals()** and **hashCode()** methods which is **required** for an Object to be used as key in HashMap.

Q24 is it possible to call String class methods using String literals?

Yes, it is **possible** to call String class methods using String literals. For example:

```
"javahungry".indexOf(u)
"javahungry".charAt(0)
"javahungry".compareTo("javahungry")
```

Q25 How to Split String in java?

You can use **split()** method of java.lang.String class or **StringTokenizer** to split a comma separated String. String **split()** method is easier to use and better because it expects a regular expression and returns an array of String which you can manipulate in the program code.

Q26 How do you compare two Strings in Java?

Use equals**()** method to compare two Strings. **Avoid** using "==" operator.
The main reason to use **equals()** method is that it always compare String values i.e **content**.
(==) operator compares the **reference** in the memory.

```
String str1 = "abc";
String str2 = new String("abc");
System.out.println(str1 == str2); //false
System.out.println(str1.equals(str2)); //true
```

Q27 Explain the difference between str1.equals("abc") and "abc".equals(str1), where str1 is any String object?

If str1 value is "abc" then both statements will give the result true. Main difference between the two statements arises when we pass str1 value as NULL. If the str1 is null then first statement will throw null pointer exception while second statement will return false.

Q28 what is Heap memory?

Heaps are **memory** areas allocated to **each** program. Memory allocated to heaps can be **dynamically** allocated, unlike memory allocated to stacks.
As a result, the heap segment can be **requested** and **released** whenever the **program** needs it. This memory is also global, which means that it can be **accessed** and **modified** from wherever in the program it is allocated instead of being localized by the function in which it is allocated. Dynamically allocated memory is referenced using 'pointers', which in turn leads to slight performance degradation over the use of local variables (on the stack).
Heap memory is also known as "**dynamic**" memory.
Heap memory is different from local stack memory. It not only differs in the way it allocates and deallocates variables when the function is called but also in the way it deallocates the variable when the function exit. This memory "block" is usually determined automatically according to the size of the object they are creating.

- It is stored in computer RAM memory just like the stack.
- It has a slower allocation of variables in comparison to variables on the stack.
- It works on the basis of using on-demand to allocate a block of data for use by the program.
- It can have fragmentation when there are a lot of allocations and deallocations.
- In C++, variables on the heap must be destroyed manually and never fall out of scope. The data is freed with delete, delete[], or free.
- In C++ or C, data created on the heap will be pointed to by pointers and allocated with new or malloc respectively.
- We can use heap memory if you don't exactly know the actual size of data needed at run time or if you need to allocate a lot of data.
- It is responsible for memory leaks

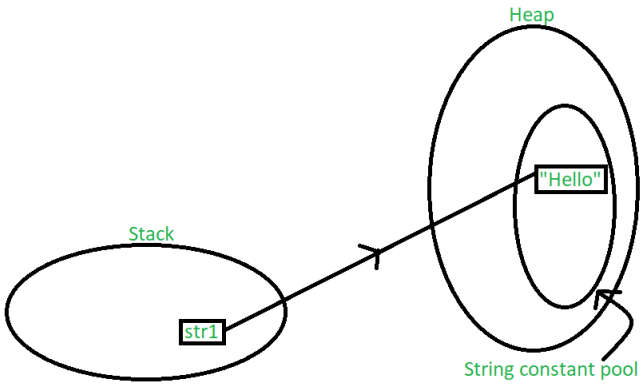
Advantages of heap memory:

Heap **doesn't** have any **limit** on memory size.
It allows you to **access** variables **globally**.
Garbage collection runs on the heap memory to free the memory used by the object.
The **heap** method is also used in the **Priority Queue**.

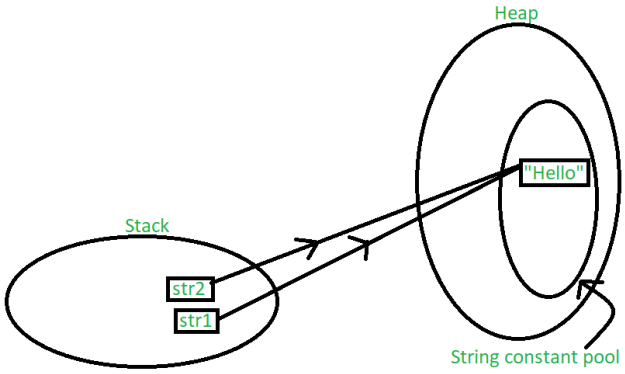
Disadvantages of heap memory:

It takes too much **time** to execute compared to the **stack**.
It takes **more time** to compute.
It can provide the **maximum** memory an OS can provide
Memory management is more **complicated** in heap memory as it is used globally.

Problems that can be solved with heap memory:
The following are some important points about Garbage Collection.
The Java Virtual Machine invokes **garbage collection** to get rid of **unused** heap memory objects. It **removes** every object that is not being used **anymore** by the running Java program. In this process, unused memory **is freed up** for other new objects to use.
Garbage collection calls the **finalize()** method of an object **before** removing it from memory and giving it a chance to be cleaned up. If the programmer does not override this method, the default finalize method will be **invoked** (the method defined in the Object class).
Garbage collection is invoked based on the size of dynamically allocated memory from the heap. It is slow, and hard to predict. Programs with real-time performance constraints may find this difficult to handle.
Let us understand why we get different output with below explanation.
String is a sequence of characters. One of the most important characteristics of a string in Java is that they are **immutable**. In other words, once created, the internal state of a string remains the same throughout the execution of the program. This immutability is achieved through the use of a special **string constant pool** in the **heap**. In this article, we will understand about the storage of the strings.
A **string constant pool** is a **separate** place in the **heap** memory where the values of all the strings which are defined in the program are **stored**. When we declare a string, an object of type String is created in the **stack**, while an instance with the value of the string is created in the **heap**. On standard assignment of a value to a string variable, the variable is allocated stack, while the value is stored in the **heap** in the **string constant pool**. For example, let's assign some value to a string str1. In java, a string is defined and the value is assigned as:
String str1 = "Hello";
The following illustration explains the memory allocation for the above declaration:



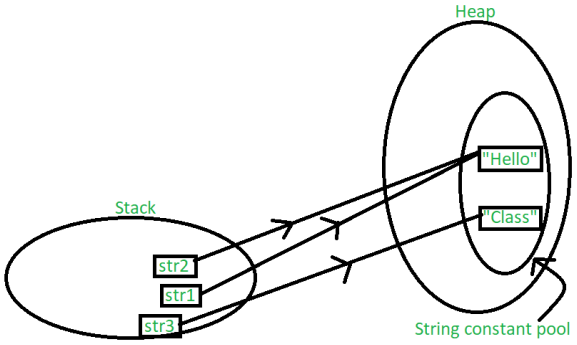
In the above scenario, a **string** object is created in the **stack**, and the value "Hello" is created and **stored** in the **heap**. Since we have normally assigned the value, it is stored in the **constant pool** area of the heap. A **pointer** points towards the value stored in the **heap** from the object in the **stack**. Now, let's take the same example with multiple string variables having the same value as follows:
String str1 = "Hello";
String str2 = "Hello";
The following illustration explains the memory allocation for the above declaration:



In this case, both the string objects get **created** in the **stack**, but another instance of the value “Hello” is not created in the **heap**. Instead, the previous instance of “Hello” is re-used. The **string constant pool** is a small cache that resides within the heap. Java stores all the values inside the string constant pool on direct allocation. This way, if a similar value needs to be accessed again, a new string object created in the stack can reference it directly with the help of a pointer. In other words, the string constant pool exists mainly to reduce memory usage and improve the re-use of existing instances in memory. When a string object is assigned a different value, the new value will be registered in the string constant pool as a separate instance. Let’s understand this with the following example:

```
String str1 = "Hello";
String str2 = "Hello";
String str3 = "Class";
```

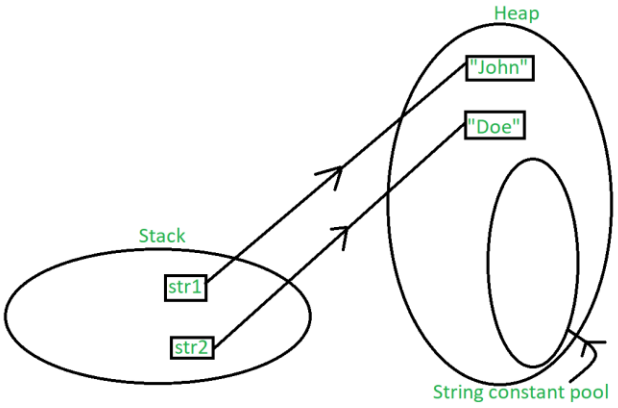
The following illustration explains the memory allocation for the above declaration:



One way to skip this memory allocation is to use the new keyword while creating a new string object. The ‘new’ keyword forces a new instance to always be created regardless of whether the same value was used previously or not. Using ‘new’ forces the instance to be created in the heap outside the string constant pool which is clear, since caching and re-using of instances isn’t allowed here. Let’s understand this with an example:

```
String str1 = new String("John");
String str2 = new String("Doe");
```

The following illustration explains the memory allocation for the above declaration:



Q29 Write a program to reverse a String in java?

1:

```
import java.io.*;
import java.util.*;

public class reverseString {
 public static void main(String[] args) {
  String input="AliveisAwesome";
  StringBuilder input1 = new StringBuilder();
  input1.append(input);
  input1=input1.reverse();
  for (int i=0;i<input1.length();i++)
  System.out.print(input1.charAt(i));
 }}
```

2:

```
import java.io.*;
import java.util.*;

public class reverseString {
 public static void main(String[] args) {
    String input = "Be in present";
    char[] hello=input.toCharArray();
    List<Character> trial1= new ArrayList< Character >();
    for(char c: hello)
    trial1.add(c);
    Collections.reverse(trial1);
    ListIterator li = trial1.listIterator();
    while(li.hasNext())
   {System.out.print(li.next());}
```

```
}}
```

3:

```
import java.io.*;
import java.util.*;

public class reverseString {
 public static void main(String[] args) {
  String input = "Be in present";
  byte [] strAsByteArray = input.getBytes();
  byte [] result = new byte [strAsByteArray.length];

  for(int i = 0; i<strAsByteArray.length; i++){
  result[i] = strAsByteArray[strAsByteArray.length-i-1];
  }
  System.out.println( new String(result));
  }}
```

Q30 Write a java program to find the first non-repeated character in the String?

```
public class FirstNonRepeatedCharFirst {
   public static void main(String args[]) {
      String inputStr ="teeter";
      for(char i :inputStr.toCharArray()){
      if ( inputStr.indexOf(i) == inputStr.lastIndexOf(i)) {
         System.out.println("First non-repeating character is: "+i);
         break;
      }
      }
   }}
```

The lastIndexOf() method returns the position of the last occurrence of specified character(s) in a string.

Q31 Find out if String has all Unique Characters.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

public class UniqueChar2 {

   public static void main (String args[])
   {
      boolean result=false;
      String inputstring="Alive is awesome";
      System.out.println("String method 2 answer "+ method2(inputstring));
   }
   public static boolean method2(String input){
      for(int i=0; i < input.length();i++)
      {
         char charcterofinputstring=input.charAt(i);
         int count=0;
         for(int j=i; j < input.length();j++){
            if (charcterofinputstring == input.charAt(j))
            count++;
         }
         if(count > 1)
         return false; }
      return true;
   }}
```

Q32 String concatenation in java

```
import java.lang.*;

public class concatenateString {
   public static void main(String[] args) {
    // Two String objects
    String str1 = "Hello";
    String str2 = "World";
      /* Using concat() method, concat first string with second string and
       storing into result string */

    String result = str1.concat(str2);
      // Print the result
    System.out.println(result);
}}
```

Q33 How to Count Number of words in the String

```
public class StringDemo{
   static int i,c=0,res;
   static int wordcount(String s){
      char ch[]= new char[s.length()];      //in string especially we have to mention the () after length
      for(i=0;i<s.length();i++){
         ch[i]= s.charAt(i);
         if( ((i>0)&&(ch[i]!=' ')&&(ch[i-1]==' ')) || ((ch[0]!=' ')&&(i==0)) )
         c++; }
      return c;}
   public static void main (String args[]){
      res=StringDemo.wordcount("   manchester united is also known as red devil ");
      //string is always passed in double quotes
      System.out.println("The number of words in the String are :  "+res);
   }}
```

Q34 How to remove all the white-spaces in the String

```
public class SqueezeString{
   static int i;
      static void squeeze(String s){
      for(i=0;i<s.length();i++){
         char ch=s.charAt(i);
         if(ch != ' ')
         System.out.print(ch);
      }
   }
   public static void main (String args[])
   {
      System.out.println("Original String is : ");
      System.out.println("   manchester united is also known as red devil   ");
      SqueezeString.squeeze("   manchester united is also known as red devil    ");
   }
}
```

Q35 How to calculate total number of characters in the String

```
public class CountCharacters{
   static int i,c=0,res;
   static int charcount(String s){
      for(i=0,c=0;i<s.length();i++){
         char ch=s.charAt(i);
         if(ch!=' ')
         c++;
      }
      return c;
   }
   public static void main (String args[]){
      res=CountCharacters.charcount(" manchester united is also known as red devil ");
      //string is always passed in double quotes
      System.out.println("The number of characters in the String are :  "+res);
   }}
```

Q36 How to calculate total number of vowels in String?

```
public class VowelsCount{
      static int i,c,res;
      static int vowelcount(String s) {
      for(i=0,c=0;i<s.length();i++){
         char ch=s.charAt(i);
         if((ch=='a')||(ch=='e')||(ch=='i')||(ch=='o')||(ch=='u'))
         c++; }
      return c ; }
   public static void main (String args[])
   {
      System.out.println("Original String is : ");
      System.out.println("   manchester united is also known as red devil   ");
      res=VowelsCount.vowelcount("   manchester united is also known as red devil ");
      System.out.println("The number of vowels in the string is  :" + res); }}
```

Q37Consider the following two Java programs, the first program produces output as “Yes”, but the second program produces output “No”. Can you guess the reason?

```
// Program 1: Comparing two references to objects
// created using literals.
import java.util.*;
class GFG {
    public static void main(String[] args) {
        String s1 = "abc";
        String s2 = "abc";
        // Note that this == compares whether
        // s1 and s2 refer to same object or not
      if (s1 == s2)
        System.out.println("Yes");
      else
        System.out.println("No");
    }}
```

Output:

Yes

```
// Program 2: Comparing two references to objects
// created using new operator.
import java.util.*;
class GFG {
    public static void main(String[] args) {
        String s1 = new String("abc");
        String s2 = new String("abc");
          // Note that this == compares whether
        // s1 and s2 refer to same object or not
      if (s1 == s2)
        System.out.println("Yes");
      else
        System.out.println("No");
    }}
```

Output:

No

Answer:

The reason for the difference in output between the two Java programs lies in how strings are handled and **stored** in Java.

In Program 1, both s1 and s2 are created using **string literals**. In Java, string literals are interned, meaning that the compiler ensures that only one copy of each distinct string is created and shared. Therefore, when you use == to compare the references, s1 and s2 point to the same interned string object, and the comparison returns true.

In Program 2, you are explicitly using the **new operator** to create two different string objects. Even though the content of the strings is the same, using new creates separate objects in memory. When you use == to compare the references, it checks whether s1 and s2 point to the exact same object in memory, and in this case, they do not. Hence, the comparison returns false.

To correctly compare the content of strings in Java, you should use the **equals**

```
if (s1.equals(s2))
    System.out.println("Yes");
else
    System.out.println("No");
```

References:

Java how to program tenth edition: paul deitel, Harvey deitel
javaTpoint
tutorialsPoint
geeksforgeeks