**1) Is String a keyword in java?**

**No. String is not a keyword in java. String is a final class in java.lang package which is used to represent the set of characters in java.**

**2) Is String a primitive type or derived type?**

**String is a derived type.**

**3) In how many ways you can create string objects in java?**

**There are two ways to create string objects in java. One is using new operator and another one is using string literals. The objects created using new operator are stored in the heap memory and objects created using string literals are stored in string constant pool.**

**String s1 = new String("abc"); //Creating string object using new operator**

**String s2 = "abc"; //Creating string object using string literal**

**4) What is string constant pool?**

**String objects are most used data objects in Java. Hence, java has a special arrangement to store the string objects. String Constant Pool is one such arrangement. String Constant Pool is the memory space in heap memory specially allocated to store the string objects created using string literals. In String Constant Pool, there will be no two string objects having the same content.**

**Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exist an object in the string constant pool with the same content, then it returns the reference of that object. It doesn’t create a new object. If the content is different from the existing objects then only it creates new object.**

**5) What is special about string objects as compared to objects of other derived types?**

**One special thing about string objects is that you can create string objects without using new operator i.e using string literals. This is not possible with other derived types (except wrapper classes). One more special thing about strings is that you can concatenate two string objects using ‘+’. This is the relaxation java gives to string objects as they will be used most of the time while coding. And also java provides string constant pool to store the string objects.**

**6) What do you mean by mutable and immutable objects?**

**Immutable objects are like constants. You can’t modify them once they are created. They are final in nature. Where as mutable objects are concerned, you can perform modifications to them.**

**[quads id=5]**

**7) Which is the final class in these three classes – String, StringBuffer and StringBuilder?**

**All three are final. (Interviewer will ask this type of questions to confuse you)**

**8) What is the difference between String, StringBuffer and StringBuilder?**

## ****Immutability :****

**This is main reason why StringBuffer and StringBuilder are introduced. As objects of String class are immutable, objects of StringBuffer and StringBuilder class are mutable. You can change the contents of StringBuffer and StringBuider objects at any time of execution. When you change the content, new objects are not created. Instead of that the changes are applied to existing object. Thus solving memory issues may caused by String class.**

## ****Object Creation :****

**You have to use ‘new‘ operator to create objects to StringBuffer and StringBuilder classes. You can’t use string literals to create objects to these classes. For example, you can’t write StringBuffer sb = “JAVA” or StringBuilder sb = “JAVA”. It gives compile time error. But, you can use both string literals and new operator to create objects to String class.**

## ****Storage Area :****

**As objects of StringBuffer and StringBuilder are created using only new operator, they are stored in heap memory. Where as objects of String class are created using both string literals and new operator, they are stored in string constant pool as well as heap memory.**

## ****Thread Safety :****

**Any immutable object in java is thread safety. Because they are unchangeable once they are created. Any type of thread can’t change the content of immutable object. This applies to objects of String class also. Of the StringBuffer and StringBuilder objects, only StringBuffer objects are thread safety. All necessary methods in StringBuffer class are synchronized so that only one thread can enter into it’s object at any point of time. Where as StringBuilder objects are not thread safety.**

## ****Performance :****

**Because of thread safety property of String and StringBuffer classes, they reduces the performance of multithreaded applications. Because, multiple threads can’t enter into objects of these classes simultaneously. One thread has to wait until another thread is finished with them. But, you will not find performance problems if you use StringBuilder class. Becuase, multiple threads can enter into objects of this class. But, be aware that StringBuilder is not thread safety.**

## ****String Concatenation :****

**There will be serious performance issues when you are performing lots of string concatenation using String class. This is because, each time you perform string concatenation using string class, a new object will be created with the concatenated string. This slows down an application. But, if you use either StringBuffer or StringBuilder instead of String class, your application will perform better. Below program shows time taken by all three classes to perform string concatenation 10000 times.**

## ****equals() and hashCode() Methods :****

**In StringBuffer and StringBuilder classes, equals() and hashCode methods are not overrided. Where as in String class they are overrided.**

## ****toString() Method :****

**toString() method is overrided in all three classes. You can also convert StringBuffer and StringBuilder objects to String type by calling toString() method on them.**

**9) Why StringBuffer and StringBuilder classes are introduced in java when there already exist String class to represent the set of characters?**

**The objects of String class are immutable in nature. i.e you can’t modify them once they are created. If you try to modify them, a new object will be created with modified content. This may cause memory and performance issues if you are performing lots of string modifications in your code. To overcome these issues, StingBuffer and StringBuilder classes are introduced in java.**

**10) How many objects will be created in the following code and where they will be stored in the memory?**

**String s1 = "abc";**

**String s2 = "abc";**

**Only one object will be created and this object will be stored in the string constant pool.**

**11) How do you create mutable string objects?**

**Using StringBuffer and StringBuilder classes. These classes provide mutable string objects.**

**12) Which one will you prefer among “==” and equals() method to compare two string objects?**

**I prefer equals() method because it compares two string objects based on their content. That provides more logical comparison of two string objects. If you use “==” operator, it checks only references of two objects are equal or not. It may not be suitable in all situations. So, rather stick to equals() method to compare two string objects. [more]**

**13) Which class will you recommend among String, StringBuffer and StringBuilder classes if I want mutable and thread safe objects?**

**StringBuffer**

**14) How do you convert given string to char array?**

**Using toCharArray() method.**

**15) How many objects will be created in the following code and where they will be stored?**

**String s1 = new String("abc");**

**String s2 = "abc";**

**Here, two string objects will be created. Object created using new operator(s1) will be stored in the heap memory. The object created using string literal(s2) is stored in the string constant pool.**

**16) Where exactly string constant pool is located in the memory?**

**Inside the heap memory. JVM reserves some part of the heap memory to store string objects created using string literals. [more]**

**17) I am performing lots of string concatenation and string modification in my code. which class among string, StringBuffer and StringBuilder improves the performance of my code. Remember I also want thread safe code?**

**StringBuffer class gives better performance in this scenario. As String class is immutable, if you use this class, a new object will be created after every string concatenation or string modification. This will lower the performance of the code. You can use StringBuilder also, but it is not thread safe. So, StringBuffer will be optimal choice here.**

**18) What is string intern?**

**String object in the string constant pool is called as String Intern. You can create an exact copy of heap memory string object in string constant pool. This process of creating an exact copy of heap memory string object in the string constant pool is called interning. intern() method is used for interning**

**19) What is the main difference between Java strings and C, C++ strings?**

In C and C++, strings are terminated with null character. But in java, strings are not terminated with null character. Strings are treated as objects in java.

**20) How many objects will be created in the following code and where they will be stored?**

String s1 = new String("abc");

String s2 = new String("abc");

Two objects will be created and they will be stored in the heap memory.

**21) Can we call String class methods using string literals?**

Yes, we can call String class methods using string literals. Here are some examples,

"abc".charAt(0)

"abc".compareTo("abc")

"abc".indexOf('c')

**22) do you have any idea why strings have been made immutable in java?**

a) Immutable strings increase security. As they can’t be modified once they are created, so we can use them to store sensitive data like username, password etc.

b) Immutable strings are thread safe. So, we can use them in a multi threaded code without synchronization.

c) String objects are used in class loading. If strings are mutable, it is possible that wrong class is being loaded as mutable objects are modifiable.

[[More](http://javarevisited.blogspot.in/2010/10/why-string-is-immutable-in-java.html)]

**23) What do you think about string constant pool? Why they have provided this pool as we can store string objects in the heap memory itself?**

String constant pool increases the reusability of existing string objects. When you are creating a string object using string literal, JVM first checks string constant pool. If that object is available, it returns reference of that object rather creating a new object. This will also speed up your application as only reference is returned and also saves the memory as no two objects with same content are created.

**24) What is the similarity and difference between String and StringBuffer class?**

The main similarity between String and StringBuffer class is that both are thread safe. The main difference between them is that String objects are immutable where as StringBuffer objects are mutable.

**25) What is the similarity and difference between StringBuffer and StringBuilder class?**

The main similarity between StringBuffer and StringBuilder class is that both produces mutable string objects. The main difference between them is that StringBuffer class is thread safe where as StringBuilder class is not thread safe.

#### Q) Why are Strings immutable in java?

## 5 Reasons of Why String is final or Immutable in Java

**1) String Pool**  
Java designer knows that String is going to be most used data type in all kind of Java applications and that's why they wanted to optimize from start. One of key step on that direction was idea of storing String literals in String pool. Goal was to reduce temporary String object by sharing them and in order to share, they must have to be from Immutable class. You cannot share a mutable object with two parties which are unknown to each other. Let's take an hypothetical example, where two reference variable is pointing to same String object:  
  
String s1 = "Java";  
String s2 = "Java";  
  
Now if s1 changes the object from "Java" to "C++", reference variable also got value s2="C++", which it doesn't even know about it. By making String immutable, this sharing of String literal was possible. In short, key idea of String pool cannot be implemented without making String final or Immutable in Java.  
  
**2) Security**  
Java has clear goal in terms of providing a secure environment at every level of service and String is critical in those whole security stuff. String has been widely used as parameter for many Java classes, e.g. for opening network connection, you can pass host and port as String, for reading files in Java you can pass path of files and directory as String and for opening database connection, you can pass database URL as String. If String was not immutable, a user might have granted to access a particular file in system, but after authentication he can change the PATH to something else, this could cause serious security issues. Similarly, while connecting to database or any other machine in network, mutating String value can pose security threats. Mutable strings could also cause security problem in Reflection as well, as the parameters are strings.  
  
**3) Use of String in Class Loading Mechanism**  
Another reason for making String final or Immutable was driven by the fact that it was heavily used in class loading mechanism. As String been not immutable, an attacker can take advantage of this fact and a request to load standard Java classes e.g. java.io.Reader can be changed to malicious class com.unknown.DataStolenReader. By keeping String final and immutable, we can at least be sure that JVM is loading correct classes.  
  
**4) Multithreading Benefits**  
Since Concurrency and Multi-threading was Java's key offering, it made lot of sense to think about thread-safety of String objects. Since it was expected that String will be used widely, making it Immutable means no external synchronization, means much cleaner code involving sharing of String between multiple threads. This single feature, makes already complicate, confusing and error prone concurrency coding much easier. Because String is immutable and we just share it between threads, it results in more readable code.  
  
**5) Optimization and Performance**  
Now when you make a class Immutable, you know in advance that, this class is not going to change once created. This guarantee open path for many performance optimizations e.g. caching. String itself knows that, I am not going to change, so String caches its hashcode. It even calculates hashcode lazily and once created, just caches it. In simple world, when you first call hashCode() method of any String object, it calculate hash code and all subsequent call to hashCode() returns already calculated, cached value. This results in good performance gain, given String is heavily used in hash based Maps e.g. [Hashtable](http://java67.blogspot.sg/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html)and [HashMap](http://java67.blogspot.sg/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html). Caching of hashcode was not possible without making it immutable and final, as it depends upon content of String itself.  
  
**Pros and Cons of String being Immutable or Final in Java**

Apart from above benefits, there is one more advantage that you can count due to String being final in Java. It's one of the most popular objects to be used as key in hash based collections e.g. HashMap and Hashtable. Though immutability is not an absolute requirement for HashMap keys, it’s much more safe to use Immutable object as key than mutable ones, because if state of mutable object is changed during its stay inside HashMap, it would be impossible to retrieve it back, given its equals() and hashCode() method depends upon the changed attribute. If a class is Immutable, there is no risk of changing its state, when it is stored inside hash based collections.  Another significant benefits, which I have already highlighted is its thread-safety. Since String is immutable, you can safely share it between threads without worrying about external synchronization. It makes concurrent code more readable and less error prone.  
  
Despite all these advantages, Immutability also has some disadvantages, e.g. it doesn't come without cost. Since String is immutable, it generates lots of temporary use and throw object, which creates pressure for Garbage collector. Java designer has already thought about it and storing String literals in pool is their solution to reduce String garbage. It does help, but you have to be careful to create String without using constructor e.g. new String() will not pick object from String pool. Also on average Java application generates too much garbage. Also storing Strings in pool has a hidden risk associated with it. String pool is located in PermGen Space of Java Heap, which is very limited as compared to Java Heap. Having too many String literals will quickly fill this space, resulting in java.lang.OutOfMemoryError: PermGen Space. Thankfully, Java language programmers has realized this problem and from Java 7 onwards, they have moved String pool to normal heap space, which is much much larger than PermGen space. There is another disadvantage of making String final, as it limits its extensibility. Now, you just cannot extend String to provide more functionality, though more general cases its hardly needed, still its limitation for those who wants to extend java.lang.String class.  
  
These 5 reasons definitely gives an hint that Why String class has been made Final and Immutable in Java. Of-course it's decision of Java designers but looks like above points contributes to take them this decision. Due to similar reasons wrapper classes like Integer, Long, Double and Float are also immutable and Final   
  
Read more: <http://www.java67.com/2014/01/why-string-class-has-made-immutable-or-final-java.html#ixzz5l3MVcalX>