**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.

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

String is a derived type.

**What is String in Java? String is a data type?**

String is a Class in java and defined in java.lang package. It’s not a primitive data type like int and long. String class represents character Strings. String is used in almost all the Java applications and there are some interesting facts we should know about String. String is immutable and final in Java and JVM uses String Pool to store all the String objects.

Some other interesting things about String is the way we can instantiate a String object using double quotes and overloading of “+” operator for concatenation.

# What is the difference between String s1 = "Hello"; and String s1 = new String("Hello"); in Java?

String S1 = new String("Hello");  
This statement creates a String object in the Heap Memory.  
String S1 = "Hello";  
This statement creates a String literal with value "Hello" in the String Pool.  
For more understanding, let's take 3 cases.

**Case 1.** String Object and Literal

String s1 = "Hello";

String s2 = new String("Hello");

Since s1 and s2 are reference variables, they'd be pointing at their respective memory locations right?

s1 points to String Pool's location

and

s2 points to Heap Memory location.

Now testing if they're equal:

if(s1 == s2)

Would return false, as the reference variables are checked.

Where as

s2.equals(s1)

will return true as equals function checks the individual characters in both the reference variables.

An Interesting Fact:

"String Literal Pool" is a memory area that contains all the string literals used in the program.

Whenever a new literal is encountered, the JRE checks if it exists in the String pool or not. It does not contain duplicate literals.

If you create two reference variables with same value "Hello", they'd both point to the same location.

**Case 2: Two String Literals:**

String s1 = "Hello";

String s2 ="Hello";

if(s1==s2)

will be true.

**Case 3: One string object and anonymous literal**

String s1 = new String("Hello");

if(s1 == "Hello")

will be false.

In this case , "Hello" creates a literal in the pool which has a different memory location than S1's heap memory.

**Can we use String in switch case?**

This is a tricky question used to check your knowledge of current Java developments. Java 7 extended the capability of switch case to use Strings also, earlier java versions doesn't support this.

If you are implementing conditional flow for Strings, you can use if-else conditions and you can use switch case if you are using Java 7 or higher versions.

**Why String is immutable or final in Java**

There are several benefits of String because it's immutable and final.

String Pool is possible because String is immutable in java.

It increases security because any hacker can't change its value and it's used for storing sensitive information such as database username, password etc.

Since String is immutable, it's safe to use in multi-threading and we don't need any synchronization.

Strings are used in java classloader and immutability provides security that correct class is getting loaded by Classloader.

**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.

**What does String intern() method do?**

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

This method always return a String that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

**Does String is thread-safe in Java?**

Strings are immutable, so we can't change it's value in program. Hence it's thread-safe and can be safely used in multi-threaded environment.

**How does substring method work in Java?**

The substring(int beginIndex, int endIndex) method returns a string that starts with beginIndex and ends with endIndex-1.

String x = "abcdef";

x = x.substring(1,3);

System.out.println(x);

**Output:bc**

You may know that because x is immutable, when x is assigned with the result of x.substring(1,3), it points to a totally new string like the following:

String in java is a sequence of characters. String is more like a utility class which works on that character sequence. This character sequence is maintained as a array called value[], for example

private final char value[];

String internally defines two private variables called offset and count to manage the char array. The declarations can be as shown below:

/\*\* The offset is the first index of the storage that is used. \*/

private final int offset;

/\*\* The count is the number of characters in the String. \*/

private final int count;

Everytime we create a substring from any string object, substring() method assigns the new values of offset and count variables. The internal char array is unchanged. This is a possible source of memory leak if substring() method is used without care.

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

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

**Do you have any idea why strings have been made immutable in java?**

String is immutable for several reasons, here is a summary:

**String Pool:** If string is not immutable, changing the string with one reference will lead to the wrong value for the other references.

**Security**: parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. If it were mutable, these parameters could be easily changed.

**Synchronization and concurrency**: making String immutable automatically makes them thread safe thereby solving the synchronization issues.

**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).

**Class loading**: String is used as arguments for class loading. If mutable, it could result in wrong class being loaded (because mutable objects change their state).

**Efficiency** The hashcode of string is frequently used in Java. For example, in a HashMap. Being immutable guarantees that hashcode will always the same, so that it can be cached without worrying the changes. That means, there is no need to calculate hashcode every time it is used.

**Difference between String and StringBuffer**

**1).** String class is immutable.

StringBuffer class is mutable.

**2).** String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.

StringBuffer class doesn't override the equals() method of Object class.

**3).** Methods are non-synchronized in String while Methods are synchronized in StringBuffer.

**4).** We can create Object of String class using new operator or without new Operator.

**5).** Concatanation Operator(+) can be used with String while Concatanation Operator (+) can’t be used.

6). hasCode() is overridden using content of String while hasCode() is not be overridden in StringBuffer.

**Difference between StringBuffer and StringBuilder.**

**1).**StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

**2)** StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

**How to convert String to char and vice versa?**

This is a tricky question because String is a sequence of characters, so we can't convert it to a single character. We can use use charAt method to get the character at given index or we can use toCharArray() method to convert String to character array.

**How to convert String to byte array and vice versa?**

We can use String getBytes() method to convert String to byte array and we can use String constructor new String(byte[] arr) to convert byte array to String.

**How to Split String in java?**

We can use split(String regex) to split the String into String array based on the provided regular expression.

**Why Char array is preferred over String for storing password?**

String is immutable in java and stored in String pool. Once it's created it stays in the pool until unless garbage collected, so even though we are done with password it's available in memory for longer duration and there is no way to avoid it. It's a security risk because anyone having access to memory dump can find the password as clear text.

If we use char array to store password, we can set it to blank once we are done with it. So we can control for how long it's available in memory that avoids the security threat with String.

**Why String is popular HashMap key in Java?**

Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for key in a Map and its processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys.

**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.

**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.

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

StringBuffer

**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.

**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.