1. Immutable Strings in Java

A string is a sequence of characters. In Java, similar to other programming languages, strings are part of predefined types. Java has `java.lang.String` class whose instances represent the strings.

The `String` class is an immutable class. Immutable means a `String` cannot be changed once its instance has been created.

Commonly, a lot of sensitive information (usernames, passwords, URLs, ports, databases, socket connections) are represented and passed around as strings. By having this information immutable, the code becomes secure to a wide range of security threats.

The string immutability also permits the caching of string literals, which allows applications to use a large number of string literals with a minimum impact on the heap memory and garbage collector.

In a mutable context, a modification of a string literal may lead to corrupted variables.

2. What is String Constant Pool in Java?

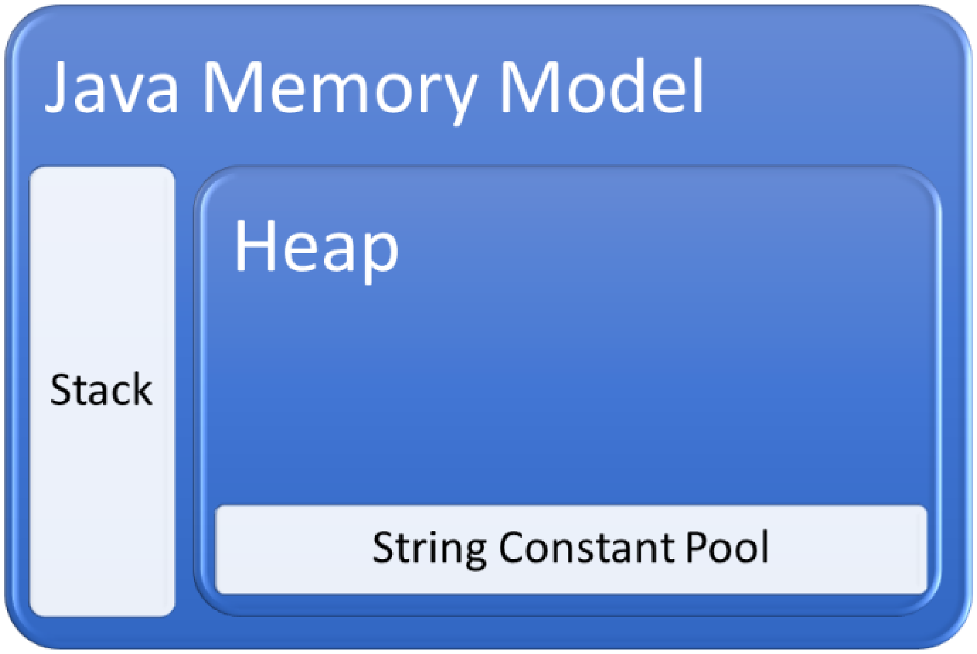
Memory in Java is divided into three parts, i.e., Heap, Stack, and String Pool. The String Constant Pool is a special area used for the storage of string literals.

- Please note that, before Java 7, the string pool was part of the *Permanent Generation* memory area.
- Starting Java 7, Strings are allocated in the Java heap area along with the other objects created by the application.
- Later, in Java 8, Permanent Generation has been completely removed.

So in the latest JVMs, String pool is a special area in heap memory allocated for storing the String literals.

Note

Though the String pool has been moved from Permgen space to heap memory area, all the concepts around the String creation, literals, objects and interning have not changed.



3. Difference between String Literals and String Objects

In Java, String Literal is a String created using double quotes while String Object is a String created using the `new()` operator.
Note that string literals are created in the String pool area, and String objects are created in the Heap memory area.

```
String strLiteral = "Hello World";

String strObj = new String("Hello World");
```

Suppose we want to create two strings with same content “`howtodoinjava`“. If a String with content “`howtodoinjava`” already exists, the new literals will be pointing to the already existing literal. In the case of String objects, a new String object will be created in the heap every time.

Let’s see an example.

```
String a = "howtodoinjava";
String b = "howtodoinjava";

System.out.println(a == b);     //true
```

In the above program, we created two string literals with the same content. After ‘a’ is created in the string pool, the next string literal ‘`b`‘ points to the same object in the memory area so `'a == b'` is `true`.

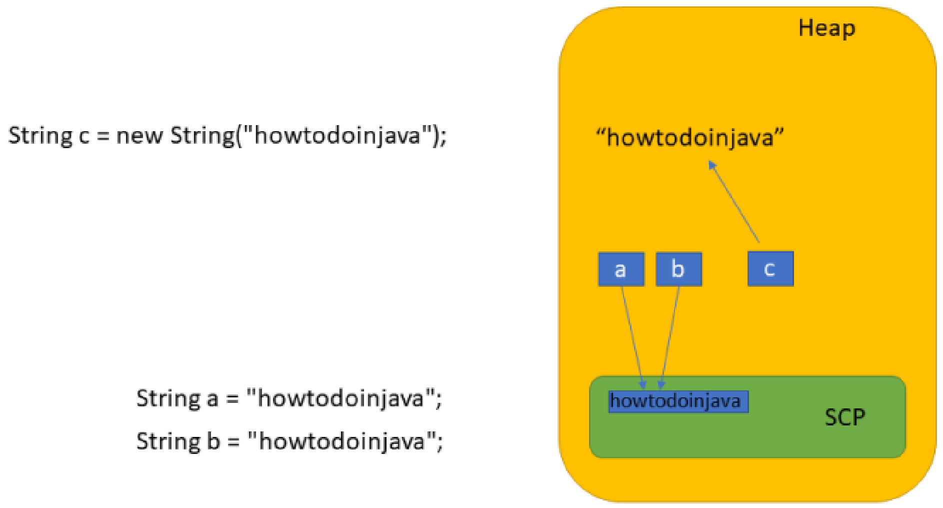
```
String a = "howtodoinjava";
String b = "howtodoinjava";

System.out.println(a == b);

String c = new String("howtodoinjava");

System.out.println(a == b);     //true
System.out.println(b == c);     //false
```

In above program, we created a new String object but with similar content. When we check the object reference equality, we see that `b` and `c` point to separate objects. It means that when we created the String object `c`, a new object was created in the memory.


String c = new String("howtodoinjava");
String a = "howtodoinjava";
String b = "howtodoinjava";
Heap
"howtodoinjava"
a
b
c
howtodoinjava
SCP

4. String.intern() Method

We know that string literals are created in the String pool, and string objects are created in the heap memory area.

We can use the method String.intern() to create string literals for the string objects. When invoked on a string object, method `intern()` creates an exact copy of a String object in the heap memory and stores it in the String constant pool.

```
String a = "howtodoinjava";
String b = "howtodoinjava";

String c = new String("howtodoinjava");
String d = c.intern();
```

In the above example, strings `a`, `b` and `d` will refer to the same string literal in the SCP. The string `c` will continue to point to the object in the heap.

5. Advantages

5.1. Enhanced Security

As stated earlier, the string pool allows being string immutable. Immutable objects help in making the application more secure because they may store sensitive information.

As we cannot change immutable objects, so it helps in making the security even more effective.

5.2. Thread Safety

Strings are perhaps the most used objects in a Java application. Imagine if strings were mutable. In that case, it would have been a nightmare to manage the thread safety in an application.

Any immutable object is thread-safe by its nature. This means that multiple threads can share and manipulate strings with no risk of corruption and inconsistency.