

Understanding Strings in Java

Today, I learned about **Strings in Java**. In Java, `String` is a **class** present in the `java.lang` package. We can create a `String` in two ways, and understanding how memory is managed is crucial for optimizing performance.

1 Using the **new** Keyword

When we create a `String` object using the `new` keyword, it is stored in the **heap memory**. Two objects may be created:

1. One in the **heap memory**.
2. Another in the **String Constant Pool (SCP)** if it doesn't already exist.

Syntax:

```
String str1 = new String("Java Programming");
```

Memory Representation:

Heap Memory : String Constant Pool

```
+-----+ +-----+
| str1 (obj) | => | "Java Programming" |
+-----+ +-----+
```

1. Every time `new` is used, a **new object** is created in **heap memory**, even if the string already exists in SCP.
2. **Heap memory objects are subject to Garbage Collection (GC)** when they are no longer referenced.

2 Using String Literals

When we create a `String` using **string literals**, the object is directly stored in the **String Constant Pool (SCP)**.

Syntax:

```
String str2 = "Java Learn";
```

Memory Representation:

String Constant Pool:

```
+-----+  
| "Java Learn" | <= str2  
+-----+
```

1. If a **String with the same value already exists**, JVM **does not create a new object**.
2. Instead, the new reference **points to the existing object**.
3. **Strings in SCP are not garbage collected** unless explicitly removed.

Memory Optimization & String Pool Behavior

One major advantage of the **String Constant Pool** is its ability to **reuse objects**, thereby reducing memory consumption.

Example:

```
String str2 = "Java Learn";  
String str3 = "Java Learn"; // Points to the same object as str2
```

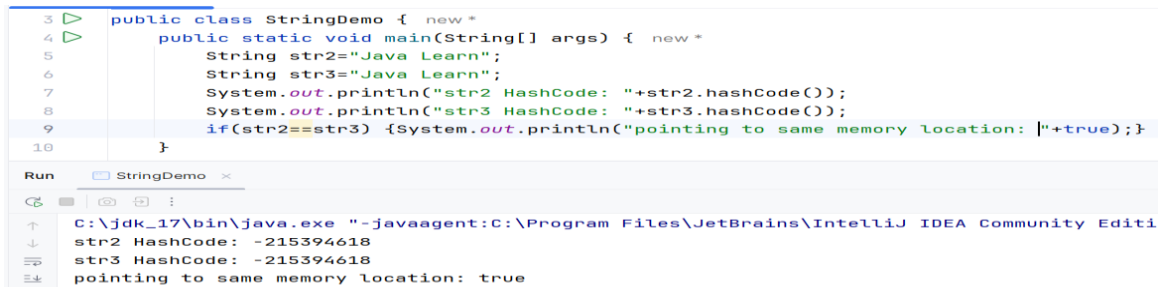
Memory Representation:

Heap Memory: (No new objects created)

String Constant Pool:

```
+-----+  
| "Java Learn" | <= str2  
|             | <= str3  
+-----+
```

1. `str2` and `str3` **point to the same memory location**, improving efficiency.
2. This eliminates duplicate string objects, optimizing memory usage.



```
3 public class StringDemo { new *  
4     public static void main(String[] args) { new *  
5         String str2="Java Learn";  
6         String str3="Java Learn";  
7         System.out.println("str2 hashCode: "+str2.hashCode());  
8         System.out.println("str3 hashCode: "+str3.hashCode());  
9         if(str2==str3) {System.out.println("pointing to same memory location: |"+true);}  
10    }
```

Run StringDemo

```
C:\jdk_17\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Editi  
str2 hashCode: -215394618  
str3 hashCode: -215394618  
pointing to same memory location: true
```

✂ How JVM Handles String Pooling

1. **When creating a string literal ("Java"):**
2. **JVM checks if it already exists in SCP.**
3. If yes, it **returns the reference** to the existing object.
4. If no, it **creates a new object in SCP.**
5. **When using `new String()`:**
6. **A new object is always created in Heap Memory**, even if the value exists in SCP.
7. The reference in Heap Memory may point to an SCP object using `.intern()`.

Example with `intern()` Method

```
String str4 = new String("Hello");  
String str5 = str4.intern(); // Forces SCP reference  
Heap Memory: String Constant Pool:
```

```
+-----+ +-----+  
| str4 (obj) | -> | "Hello" | <= str5  
+-----+ +-----+
```

`str4` is stored in **Heap Memory**, while `str5` **references the SCP object**.

Key Takeaways

- ✓ Using `new` keyword **always creates a new object** in Heap Memory.
- ✓ String literals **reuse objects** in the **String Constant Pool**.
- ✓ **Garbage Collector (GC)** does not clean SCP unless explicitly removed.
- ✓ Using `.intern()` helps reference **SCP objects explicitly**.
- ✓ String Pool **reduces memory usage and optimizes performance**.

Let me know if you have any thoughts or improvements! 