Java String Creation & String Pool

In Java, String is a special class that represents a sequence of characters. Unlike other objects, Strings are **immutable**, meaning their values cannot be changed once assigned.

1. String Creation in Java

There are three main ways to create a String in Java:

(i) Using String Literals

String s1 = "Hello";

String s2 = "Hello";

- When a string is created using **literals**, it is stored in the **String Pool** inside the **Heap Memory**.
- If the same string already exists in the pool, Java **does not create a new object** but refers to the existing one.

(ii) Using new Keyword

String s3 = new String("Hello");

- This explicitly creates a new String object in **Heap memory**, even if an identical string exists in the String Pool.
- It does **not** use the String Pool unless we call the intern() method.

(iii) Using intern() Method

String s4 = new String("Hello").intern();

- The intern() method forces the JVM to check the String Pool.
- If the string is already in the pool, it returns a reference to that string.
- Otherwise, it adds this string to the pool and returns a reference.

2. String Pool (String Constant Pool)

The **String Pool** is a special memory area inside the heap, optimized for storing string literals. When a string is created using a **literal**, it is **stored in the pool** to avoid duplication.

Example: String Pool Behavior

String a = "Java";

String b = "Java";

System.out.println(a == b); // true (Both refer to the same object in the pool)

However, when using new String("Java"), a **new object** is created outside the pool:

String c = new String("Java");

System.out.println(a == c); // false (Different memory locations)

If we use intern(), we get a reference from the **pool**:

String d = c.intern();

System.out.println(a == d); // true (Both refer to the same object in the pool)

3. Comparing Strings: == vs. equals()

(i) == Operator

- Compares **memory addresses** (references), not actual content.

- Returns true only if both references point to the **same object**.

Example:

String x = "Hello";

String y = new String("Hello");

System.out.println(x == y); // false (Different memory locations)

(ii) equals() Method

- Compares the **actual content** of the string.
- Returns true if the values are the same, regardless of memory location.

Example:

System.out.println(x.equals(y)); // true (Same content: "Hello")

Key Difference

Comparison Method	Compares	Returns true if
==	Memory address	Both references point to the same object
equals()	Actual content	The contents of the strings are the same

4. Practical Example

```
public class StringDemo {
   public static void main(String[] args) {
      String s1 = "Java";   // Stored in String Pool
      String s2 = "Java";   // Refers to the same object in Pool
      String s3 = new String("Java"); // Creates a new object in Heap
      String s4 = s3.intern(); // Refers to the String Pool object

      // Comparisons
      System.out.println(s1 == s2); // true (Same object in pool)
      System.out.println(s1 == s3); // false (Different memory locations)
      System.out.println(s1.equals(s3)); // true (Same content)
      System.out.println(s1 == s4); // true (s4 refers to the pool object)
   }
}
```

Expected Output

true

false

true

true

5. Summary

String Creation Method	Stored in	Uses String Pool?
Using Literals ("Hello")	String Pool	Yes
Using new (new String("Hello"))	Heap	No
Using intern() (new String("Hello").intern())	String Pool	Yes

Key Takeaways

1. Strings in Java are **immutable**.
2. The **String Pool** helps save memory by avoiding duplicate strings.
3. Use **==** to compare memory locations, and **equals()** to compare actual content.
4. intern() ensures that a string exists in the **String Pool**.