CLASS: String

The `String` class is a **predefined** class in the `java.lang` package. It is one of the **most important** classes in Java due to its **immutability**.

- **Immutable**: Once a `String` object is created, its content **cannot be modified.**
- Any operation that modifies a string returns a **new object**.
- To use the modified string, you must assign it to a **reference variable**, or it becomes **unreachable**.

```java
class StringClass {
    public static void main(String[] args) {
        String s = new String("String");
        System.out.println(s); // String

        s.concat("Class"); // New object created, but not referenced
        System.out.println(s); // String

        s = s.concat("Class");
        System.out.println(s); // StringClass
    }
}
```

Initialization Methods

1. **Using `new` keyword:**

```java
String ob = new String("Literal");
```

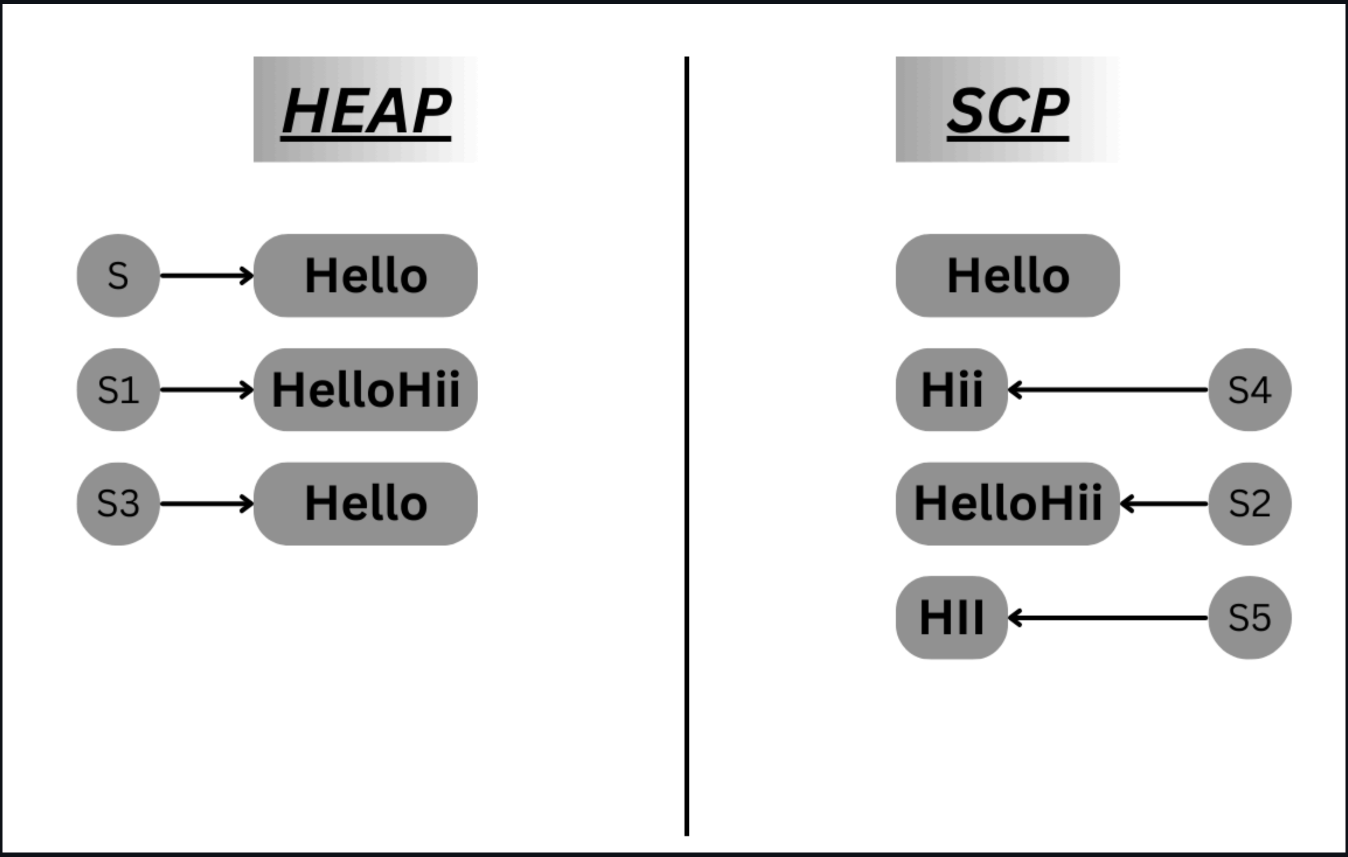
2. **Using string literals:**

```java
String ob = "Literal";
```

- With `new`, two objects are created: one in **Heap**, one in **SCP**.
- The reference points to the heap object.

```
String ob = new String("Literal");
                      |              |
                   (Heap)         (SCP)
                  "Literal"      "Literal"
```

```java
class HeapSCP {
    public static void main(String[] args) {
        String S = new String("Hello");
        String S1 = S.concat("Hii");
        String S2 = "HelloHii";
        String S3 = new String("Hello");
        String S4 = "Hii";
        String S5 = "HII";
    }
}
```



- If a string literal already exists in SCP, JVM reuses it.
- **Garbage Collector does not affect** objects in SCP.

Final vs Immutable

`final`

- Applicable for **class, methods, and variables**.
- Once initialized, cannot be changed, reassigned, or overridden.
- It is a **keyword** in Java.

`immutable`

- Applicable for **String objects**.
- Once a `String` object is created, its content **cannot be changed**.
- If a change is attempted, a **new object** will be created.
- Immutability is a **behavior of an object**.

Demo: Custom Immutable Class

```java
class Demo {
    final private int x;

    Demo(int x) {
        this.x = x;
    }

    public Demo modify(int x) {
        if (this.x == x) {
            return this;
        } else {
            return new Demo(x);
        }
    }

    public static void main(String[] args) {
        Demo d1 = new Demo(10);
        Demo d2 = d1.modify(10);
        Demo d3 = d2.modify(20);
    }
}
```

- In the above example, the `Demo` class is made **immutable** by declaring its field as `final` and only allowing state changes by returning a new object.
- `d1` and `d2` refer to the **same object**, while `d3` refers to a **new object** with updated state.

Constructors

- `String()`
- `String(String)`
- `String(StringBuffer)`

Commonly Used Methods

Return Type	Method Signature
`String`	`.concat(String)`
`char`	`.charAt(int)`
`int`	`.length()`
`String`	`.toLowerCase()`
`String`	`.toUpperCase()`
`boolean`	`.equals(Object)`
`boolean`	`.equalsIgnoreCase(String)`
`int`	`.indexOf(char)`
`int`	`.lastIndexOf(char)`
`String`	`.replace(char, char)`
`String`	`.substring(int)`
`String`	`.substring(int, int)`

Return Type	Method Signature
`String`	`.trim()`

`.concat(String)`, `.charAt(int)`, `.length()`

```java
class StringMethods1 {
    public static void main(String[] args) {
        String s = "Hello";
        System.out.println(s.concat("Sambit")); // HelloSambit
        System.out.println(s.charAt(1));        // e
        System.out.println(s.length());         // 5
    }
}
```

`.toUpperCase()`, `.toLowerCase()`

```java
class StringMethods2 {
    public static void main(String[] args) {
        String s = "HeLlO sAmBiT !!";
        System.out.println(s.toUpperCase()); // HELLO SAMBIT !!
        System.out.println(s.toLowerCase()); // hello sambit !!
    }
}
```

`.equals(Object)`, `.equalsIgnoreCase(String)`

```java
class StringMethods3 {
    public static void main(String[] args) {
        String s = "Hello";
        String s1 = "hello";
        System.out.println(s.equals(s1));           // false
        System.out.println(s.equalsIgnoreCase(s1)); // true
    }
}
```

`.indexOf(char)`, `.lastIndexOf(char)`

```java
class StringMethods4 {
    public static void main(String[] args) {
        String s = "Hello";
        System.out.println(s.indexOf('l'));      // 2
        System.out.println(s.lastIndexOf('l'));  // 3
    }
}
```

`.replace(char, char)`

```java
class StringMethods5 {
    public static void main(String[] args) {
        String s = "Hello";
        System.out.println(s.replace('l', 'p')); // Heppo
    }
}
```

`.substring(int)`, `.substring(int, int)`

```java
class StringMethods6 {
    public static void main(String[] args) {
        String s = "HelloSagar";
                   //0123456789
        System.out.println(s.substring(3));    // loSagar
        System.out.println(s.substring(3, 8)); // loSag
    }
}
```

`.trim()`

```java
class StringMethods7 {
    public static void main(String[] args) {
        String s = "  Hello   ";
        System.out.println(s.length()); // 10
        s = s.trim();
        System.out.println(s.length()); // 5
    }
}
```