# ■ String Immutability in Java

## ■ Definition

A String in Java is **Immutable**, meaning once it is created, its value cannot be changed. Any modification creates a **new String object** instead of changing the existing one.

## ■ Memory Illustration (String Pool)

Example:

```
String s1 = "Hello"; String s2 = "Hello";
```

Both s1 and s2 point to the **same object** in the String Pool.

Diagram:

```
■■■■■■■■■■■■■■■■ ■ "Hello" ■ ← Both s1 and s2 point here
■■■■■■■■■■■■■■■■ ↑ ↑ s1 s2
```

If we do:

```
s1 = "Dello";
```

```
■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■ ■ "Hello" ■ ■ "Dello" ■
■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■ ↑ ↑ s2 s1
```

## ■ Example Code

```
public class Immutability { public static void main(String[] args) {
String s = "Hello"; String x = "Hello"; s = s.substring(0, 2) + 'y' +
s.substring(3); System.out.println(s); // Heylo System.out.println(x); //
Hello } }
```

## ■ Why is String Immutable?

1■■ Security → Strings are used in sensitive operations (file paths, networking).

2■■ Thread-Safety → Safe to share across threads without locks.

3■■ String Pooling → Saves memory by reusing literals.

4■■ Hashing & Collections → Strings are reliable keys in HashMap/HashSet.

## ■ Alternatives (Mutable Options)

- StringBuilder → Mutable, fast, not thread-safe.
- StringBuffer → Mutable, thread-safe, slower.

## ■ Quick Comparison

| Feature | String (Immutable) | StringBuilder | StringBuffer |
|---|---|---|---|
| Mutability | ■ No | ■ Yes | ■ Yes |
| Thread-safety | ■ Safe | ■ Not Safe | ■ Safe |
| Performance | ■ Moderate | ■■ Fast | ■ Slower |
| Memory efficiency | ■ (Pooling) | ■ | ■ |