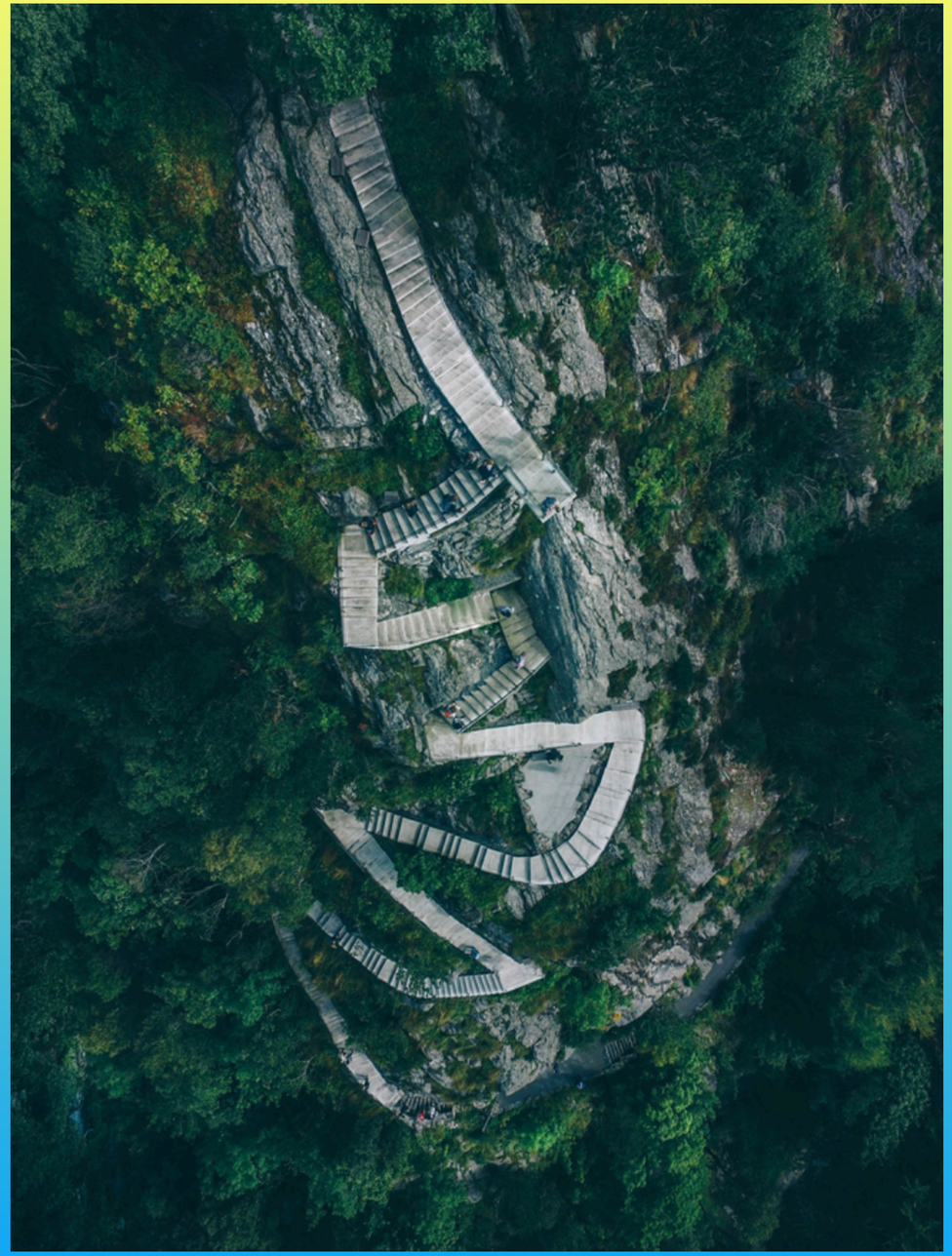
String Class and Methods in Java

Based on *Java: The Complete Reference* by Herbert Schildt



Lecture Overview

- Introduction to the String Class
- String Creation and Initialization
- Immutable Nature of Strings
- Common String Methods
- StringBuffer and StringBuilder
- String Comparison
- Practical Examples

Introduction to the String Class

Definition:

- A `String` is a sequence of characters.
- In Java, the `String` class is immutable, meaning once created, its value cannot be changed.

Creation:

- `String` is part of the `java.lang` package, so no import is necessary.

```
String str = "Hello";
```

```
String str1 = "Hello World";  // String literal
String str2 = new String("Hello World");  // Using 'new' keyword
```

String Creation and Initialization

Ways to Create Strings:

- **String Literal**: Stored in the String pool.
- **String Object**: Created using the `new` keyword.

String Pool:

- Java optimizes memory usage by storing literals in a special memory area called the string pool.

```
String str1 = "Java";  // String literal in the pool
String str2 = new String("Java");  // Object in the heap
```

Immutability of String

What is Immutability?

- Once a `String` object is created, it cannot be changed.

Why are Strings Immutable?

- For security, synchronization, and efficient memory management (string pool reuse).

```java
String s = "Java";
s.concat(" Programming");  // Does not change original string
System.out.println(s);  // Outputs: "Java"
```

Common Methods of the String Class

Length: `length()`

- Returns the length of the string.

```
String str = "Hello";
int len = str.length();  // Outputs: 5
```

Character at Index: `charAt()`

```
String str = "Hello";
char ch = str.charAt(1);  // Outputs: 'e'
```

Substring and Concatenation

Substring: `substring(int startIndex)` or `substring(int startIndex, int endIndex)`

```java
String str = "Hello";
String sub = str.substring(0, 2);  // Outputs: "He"
```

Concatenation: `concat()` or `+` operator.

```java
String str = "Hello";
String result = str.concat(" World");  // Outputs: "Hello World"
```

String Comparison Methods

equals(): Checks content equality.

```
String str = "Hello";
boolean isEqual = str.equals("Hello");
```

== operator: Compares reference, not content.

compareTo(): Lexicographical comparison.

```
int result = str.compareTo("Hello");
```

Other Important String Methods

indexOf(): Finds the index of a character or substring.

```
int index = str.indexOf('l');  // Outputs: 2
```

toUpperCase() / toLowerCase(): Converts the case.

```
String upper = str.toUpperCase();  // Outputs: "HELLO"
```

trim(): Removes whitespace from both ends of a string.

```
String trimmed = str.trim();
```

StringBuffer and StringBuilder

Need for Mutable Strings:

- Since `String` is immutable, `StringBuffer` and `StringBuilder` allow for mutable strings.

StringBuffer:

- Synchronized, thread-safe, but slower.

StringBuilder:

- Not synchronized, faster for single-threaded operations.

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
```

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
```

String Methods vs. StringBuffer/StringBuilder Methods

String Methods: `concat()`, `substring()`, etc., create new objects since strings are immutable.
StringBuffer/StringBuilder Methods: `append()`, `insert()`, `delete()` modify the existing object.
Performance:

- For intensive string manipulations, `StringBuilder` or `StringBuffer` is preferred over `String`.

Practical Examples

Example 1: Reversing a String using `StringBuilder`.

```java
StringBuilder sb = new StringBuilder("Hello");
sb.reverse();
System.out.println(sb);  // Outputs: "olleH"
```

Example 2: Checking for Palindrome.

```java
String original = "madam";
String reverse = new StringBuilder(original).reverse().toString();
boolean isPalindrome = original.equals(reverse);
```

Best Practices

- Use `String` for constants and minimal string manipulations.
- Use `StringBuilder` for heavy string manipulations in single-threaded environments.
- Use `StringBuffer` for thread-safe operations in multi-threaded environments.

Summary

- Strings are immutable in Java.
- Various methods like `length()`, `substring()`, `concat()`, and `equals()` help manipulate strings.
- For mutable strings, use `StringBuffer` (synchronized) or `StringBuilder` (non-synchronized).
- Efficient use of strings is important for performance and memory management.

Thank You