String Operations

String trim(), replace(), split()

Learn essential Java string methods: trim() & strip() for whitespace removal, replace() & replaceAll() for substring replacement, split() to create arrays, & String.join() to recombine. Master Java string manipulation with practical examples.

When working with strings in Java, many useful methods allow you to clean up, replace parts of, or break down strings into smaller pieces.

This lesson will cover how to **trim** unnecessary spaces, **replace** characters or patterns, and **split** strings into parts. We'll also see how to recombine strings using `String.join()`.

Trimming Strings: `trim()` and `strip()`

Sometimes, strings come with extra spaces (also called "whitespace") at the beginning or end. These spaces can cause problems in your program, but you can remove them using `trim()` or `strip()`.

`trim()` Method

The `trim()` method removes all the leading (spaces at the start) and trailing spaces (spaces at the end) from a string but *does not remove spaces in the middle* of the string.

```
String text = "   Hello, World!   ";
String trimmedText = text.trim();  // "Hello, World!"
System.out.println(trimmedText);  // Outputs: "Hello, World!"
```

Here, the spaces before and after `"Hello, World!"` are removed, but the content in the middle stays the same.

`strip()` Method

The `strip()` method works similarly to `trim()`, but it's a newer and more powerful method. It removes not only regular spaces but also other types of invisible characters that are sometimes used to represent whitespace.

For most cases, `trim()` and `strip()` behave the same, but `strip()` can handle more edge cases, especially with different character sets.

```
String text = "   Hello   ";
String strippedText = text.strip();  // "Hello"
System.out.println(strippedText);  // Outputs: "Hello"
```

Replacing Parts of a String: `replace()` and `replaceAll()`

Sometimes, you need to change certain characters or parts of a string. Java provides two methods: `replace()` and `replaceAll()`.

`replace()` Method

The `replace()` method is used to replace **all occurrences** of a specific character or substring with another one. It works for both single characters and strings.

```
String text = "Hello, World!";
String newText = text.replace('l', 'x');  // "Hexxo, Worxd!"
```

```
System.out.println(newText);  // Outputs: "Hexxo, Worxd!"
```

In this example, every `'l'` is replaced with `'x'`.

You can also replace entire substrings:

```
String text = "I love Java!";
String newText = text.replace("Java", "Python");  // "I love Python!"
System.out.println(newText);  // Outputs: "I love Python!"
```

`replaceAll()` Method

The `replaceAll()` method is used when you want to replace **patterns** in a string using **regular expressions (regex)**. A regular expression allows you to search for more complex patterns.

```
String text = "123abc456";
String newText = text.replaceAll("\\d", "X");  // Replaces all digits with 'X'
System.out.println(newText);  // Outputs: "XXXabcXXX"
```

Here, `\\d` is a regex that matches any digit, so all digits in `"123abc456"` are replaced with `'X'`.

Splitting Strings: `split()`

The `split()` method is used to break a string into smaller pieces (substrings) based on a **delimiter**. A delimiter is a character or substring that tells Java where to divide the string. For example, commas `,`, spaces , or hyphens `-` can be used as delimiters.

Splitting by Spaces

```
String text = "Hello World Welcome";
String[] words = text.split(" ");  // Splits the string by spaces

for (String word : words) {
    System.out.println(word);
}
```

Here, the string is split into three parts: `"Hello"`, `"World"`, and `"Welcome"`, based on the spaces between the words.

Splitting by Comma

```
String csv = "apple,banana,cherry";
String[] fruits = csv.split(",");  // Splits by commas

for (String fruit : fruits) {
    System.out.println(fruit);
}
```

This will split the string `"apple,banana,cherry"` into an array: `["apple", "banana", "cherry"]`.

Recombining Strings: `String.join()`

After splitting a string, you may want to recombine the parts into a single string with a specific delimiter. You can use the `String.join()` method to do this.

Joining with a Comma

```
String[] words = {"apple", "banana", "cherry"};
String combined = String.join(", ", words);  // Combines with a comma and space

System.out.println(combined);  // Outputs: "apple, banana, cherry"
```

In this case, the array of words is joined back into a single string, with each word separated by `", "`.

Joining with No Delimiter

You can also join strings without any delimiter by using an empty string `""` as the separator.

```java
String[] parts = {"H", "e", "l", "l", "o"};
String combined = String.join("", parts);  // Combines with no spaces
System.out.println(combined);  // Outputs: "Hello"
```

Summary

- **Trimming**: Use `trim()` or `strip()` to remove spaces from the beginning and end of a string.
- **Replacing**: Use `replace()` to change specific characters or substrings.
  Use `replaceAll()` to replace patterns using regular expressions.
- **Splitting**: Use `split()` to break a string into parts based on a delimiter (like spaces or commas).
- **Joining**: Use `String.join()` to recombine an array of strings into a single string, with a delimiter of your choice.

These methods allow you to efficiently clean up and modify strings in Java, making string manipulation straightforward.

Next - String Formatting And Parsing
String Formatting in Java (%s, %d, %f) →