Java Strings :

In Java, a `String` is a sequence of characters that represents text. It is a widely used data type for handling text and is part of the Java standard library. Here, I'll provide you with two code examples to help you understand how to work with Java strings.

Example 1: Basic String Operations

In this example, we'll cover some basic string operations such as creating strings, concatenating them, and finding the length of a string.

```java
public class StringExample1 {
    public static void main(String[] args) {
        // Creating strings
        String firstName = "John";
        String lastName = "Doe";

        // Concatenating strings
        String fullName = firstName + " " + lastName;

        // Finding the length of a string
        int length = fullName.length();

        // Displaying the results
        System.out.println("First Name: " + firstName);
        System.out.println("Last Name: " + lastName);
        System.out.println("Full Name: " + fullName);
        System.out.println("Length of Full Name: " + length);
    }
}
```

Output:
```
First Name: John
Last Name: Doe
Full Name: John Doe
Length of Full Name: 8
```

In this example, we create two string variables, `firstName` and `lastName`, and then concatenate them to create a `fullName`. We also find the length of the `fullName` string using the `length()` method.

Example 2: String Comparison

In this example, we demonstrate how to compare strings for equality using both the `equals()` method and the `==` operator.

```java
public class StringExample2 {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";
        String str3 = "Hello";
        String str4 = new String("Hello");

        // Using equals() method for string comparison
        boolean isEqual1 = str1.equals(str2); // false
        boolean isEqual2 = str1.equals(str3); // true

        // Using == operator for string comparison
        boolean isSameReference = str1 == str3; // true
        boolean isDifferentReference = str1 == str4; // false

        // Displaying the results
        System.out.println("str1 equals str2: " + isEqual1);
        System.out.println("str1 equals str3: " + isEqual2);
        System.out.println("str1 == str3: " + isSameReference);
        System.out.println("str1 == str4: " + isDifferentReference);
    }
}
```

Output:
```
str1 equals str2: false
str1 equals str3: true
str1 == str3: true
str1 == str4: false
```

In this example, we compare strings using both the `equals()` method (which compares the content) and the `==` operator (which compares references). Note that `str4` is created using the `new` keyword, so it has a different reference even though its content is the same as `str1`.

Certainly! Here are a few more examples of common operations you can perform with Java strings:

Example 3: String Substring and IndexOf

In this example, we'll demonstrate how to extract a substring from a string and find the index of a specific character or substring within a string.

```java
public class StringExample3 {
    public static void main(String[] args) {
        String sentence = "The quick brown fox jumps over the lazy dog";

        // Extracting a substring
        String substring1 = sentence.substring(10, 15); // "brown"

        // Finding the index of a character or substring
        int indexOfFox = sentence.indexOf("fox"); // 16
        int indexOfCat = sentence.indexOf("cat"); // -1 (not found)

        // Displaying the results
        System.out.println("Substring: " + substring1);
        System.out.println("Index of 'fox': " + indexOfFox);
        System.out.println("Index of 'cat': " + indexOfCat);
    }
}
```

Output:
```
Substring: brown
Index of 'fox': 16
Index of 'cat': -1
```

In this example, we use the `substring()` method to extract a portion of the `sentence` string. We also use the `indexOf()` method to find the index of the substring "fox" within the `sentence` string. If the substring or character is not found, `indexOf()` returns -1.

Example 4: String Splitting

In this example, we'll split a string into an array of substrings based on a delimiter.

```java
```

```
public class StringExample4 {
    public static void main(String[] args) {
        String csvData = "John,Doe,30,New York";

        // Splitting the string into an array
        String[] parts = csvData.split(",");

        // Accessing individual elements
        String firstName = parts[0]; // "John"
        String lastName = parts[1]; // "Doe"
        int age = Integer.parseInt(parts[2]); // 30
        String city = parts[3]; // "New York"

        // Displaying the results
        System.out.println("First Name: " + firstName);
        System.out.println("Last Name: " + lastName);
        System.out.println("Age: " + age);
        System.out.println("City: " + city);
    }
}
```

Output:
```
First Name: John
Last Name: Doe
Age: 30
City: New York
```

In this example, we split the `csvData` string into an array of substrings using the `split()` method, specifying a comma (`,`) as the delimiter. This allows us to easily access and work with individual elements of the CSV data.

These examples demonstrate additional string operations such as substring extraction, searching, splitting, and converting substrings to other data types. Java provides a rich set of methods for working with strings, making it versatile for various text processing tasks.