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

### Example 5: String Case Conversion

In this example, we'll demonstrate how to convert the case of strings from uppercase to lowercase and vice versa.

```java
public class StringExample5 {
    public static void main(String[] args) {
        String original = "Hello, World!";

        // Convert to lowercase
        String lowercase = original.toLowerCase();

        // Convert to uppercase
        String uppercase = original.toUpperCase();

        // Displaying the results
        System.out.println("Original: " + original);
        System.out.println("Lowercase: " + lowercase);
        System.out.println("Uppercase: " + uppercase);
    }
}
```

Output:
```
Original: Hello, World!
Lowercase: hello, world!
Uppercase: HELLO, WORLD!
```

In this example, we use the `toLowerCase()` and `toUpperCase()` methods to convert the case of the `original` string.

### Example 6: String Trimming

In this example, we'll demonstrate how to remove leading and trailing whitespace from a string using the `trim()` method.

```java
public class StringExample6 {
```

```
    public static void main(String[] args) {
        String text = "    Hello, World!    ";

        // Remove leading and trailing whitespace
        String trimmed = text.trim();

        // Displaying the results
        System.out.println("Original: \"" + text + "\"");
        System.out.println("Trimmed: \"" + trimmed + "\"");
    }
}
```

Output:
```
Original: "    Hello, World!    "
Trimmed: "Hello, World!"
```

In this example, we use the `trim()` method to remove any leading and trailing whitespace from the `text` string.

### Example 7: String Concatenation with StringBuilder

When you need to concatenate multiple strings efficiently, you can use the `StringBuilder` class to avoid unnecessary string object creations, which can be expensive.

```java
public class StringExample7 {
    public static void main(String[] args) {
        StringBuilder stringBuilder = new StringBuilder();

        // Append strings to the StringBuilder
        stringBuilder.append("Hello, ");
        stringBuilder.append("World!");

        // Convert StringBuilder to a String
        String result = stringBuilder.toString();

        // Displaying the result
        System.out.println("Result: " + result);
    }
}
```

Output:
```
Result: Hello, World!
```

In this example, we use a `StringBuilder` to efficiently concatenate multiple strings. The `append()` method is used to add strings to the `StringBuilder`, and then we convert it to a regular `String` using `toString()`.

Certainly! Here are a few more examples of common operations and concepts related to Java strings:

### Example 8: String Formatting

You can format strings in Java using the `String.format()` method. It allows you to create formatted strings with placeholders.

```java
public class StringExample8 {
    public static void main(String[] args) {
        String name = "Alice";
        int age = 28;

        // Using String.format() to create a formatted string
        String formattedString = String.format("Name: %s, Age: %d", name, age);

        // Displaying the formatted string
        System.out.println(formattedString);
    }
}
```

Output:
```
Name: Alice, Age: 28
```

In this example, we use the `%s` and `%d` placeholders in the format string, and then we provide the values for those placeholders using the `String.format()` method.

### Example 9: String Comparison (Ignoring Case)

Sometimes you need to compare strings while ignoring their case. You can achieve this using the `equalsIgnoreCase()` method.

```java
public class StringExample9 {
    public static void main(String[] args) {
        String str1 = "Java";
        String str2 = "java";

        // Comparing strings ignoring case
        boolean isEqualIgnoreCase = str1.equalsIgnoreCase(str2); // true

        // Displaying the result
        System.out.println("str1 equals (ignore case) str2: " + isEqualIgnoreCase);
    }
}
```

Output:
```
str1 equals (ignore case) str2: true
```

The `equalsIgnoreCase()` method compares two strings while ignoring their case, so "Java" and "java" are considered equal in this case.

### Example 10: String Immutability

In Java, strings are immutable, meaning their values cannot be changed after creation. Any operation that appears to modify a string actually creates a new string. Here's an example to illustrate this concept:

```java
public class StringExample10 {
    public static void main(String[] args) {
        String original = "Hello";
        String modified = original + ", World!";

        // Displaying the results
        System.out.println("Original: " + original);
        System.out.println("Modified: " + modified);
    }
}
```

Output:
```
Original: Hello
Modified: Hello, World!
```

In this example, when we concatenate "original" and ", World!" to create "modified," we are not modifying the "original" string. Instead, a new string is created, and "original" remains unchanged.