Learning Objectives: String Comparison

- **Compare strings with `==` and `!=`**
- **Compare strings with `equals()`**
- **Compare strings with `compareTo()`**

== & !=

Comparing with ==

The == operator can be used with strings just like it is with numbers or boolean values.

```
String string1 = "It's Friday!";
String string2 = "It's Friday!";

System.out.println(string1 == string2);
```

challenge

What happens if you:

- Change the value of `string1` to `"it's friday!"`?
- Change the value of `string2` to `"it\'s friday!"`?

Comparing with !=

You can also test for string inequality with the `!=` operator.

```
String string1 = "It's Friday!";
String string2 = "It's Monday.";

System.out.println(string1 != string2);
```

challenge

What happens if you:

- Change the value of `string2` to `"It's Friday"`?
- Change the value of `string2` to `"It's Friday!"`?

Equals

Creating a “New” String

In Java, there are actually two different common ways to create a string. One way is to create a string variable and assign string values to it such as `String string1 = "It's Friday!";`. Another way is to use the keyword `new` such as `String string2 = new String("It's Friday!");`. These two ways will result in the same output when the strings are printed. Both `string1` and `string2` contain the characters `It's Friday!`.

However, since the strings were created using two **different** methods, Java actually treats them as two different items. Because of this, you *cannot* use the `==` operator to compare `string1` and `string2`. Doing so will result in a boolean of `false`.

```java
String string1 = "It's Friday!";
String string2 = new String("It's Friday!");
String string3 = new String("It's Friday!");
String string4 = "It's Friday!";

System.out.println(string1 == string2);
```

challenge

What happens if you:

- Change the print statement to `System.out.println(string1 == string4);`?
- Change the print statement to `System.out.println(string2 == string3);`?

important

IMPORTANT

NOTE that the `new` string method in Java always creates a new **unique** string that is not comparable to any strings created before it. Thus, even if two strings contain the same characters but are created using the `new` method, they will still be incomparable.

The equals() Method

The `equals()` method in Java enables you to compare strings regardless of how they were created.

```java
String string1 = "It's Friday!";
String string2 = new String("It's Friday!");
String string3 = new String("It's Friday!");
String string4 = "It's Friday!";
String string5 = "It's Friday.";

System.out.println(string1.equals(string2));
```

challenge

What happens if you:

- Change the print statement to `System.out.println(string1.equals(string4));`?
- Change the print statement to `System.out.println(string2.equals(string3));`?
- Change the print statement to `System.out.println(string2.equals(string4));`?
- Change the print statement to `System.out.println(string4.equals(string5));`?

Compare To

Lexicographical Order

In Java, strings can be compared lexicographically, meaning they can be compared according to how they will appear in the dictionary. You can use the `compareTo()` method to determine which of two strings comes first. A return value of a **negative** integer means the first string comes first, a return value of a **positive** integer means the second string comes first, and a return value of `0` means the strings are equal and neither comes first.

```
String string1 = "apple";
String string2 = "cat";

if (string1.compareTo(string2) < 0) {
  System.out.println("string1 comes first");
}
else if (string1.compareTo(string2) > 0) {
  System.out.println("string2 comes first");
}
else {
  System.out.println("the strings are equal");
}
```

challenge

What happens if you:

- Change `string2` to `"apple"`?
- Change `string2` to `"10"`?
- Change `string1` to `"2"` in your current code?

Why Does "10" Come Before "2"?

When Java compares strings lexicographically, it compares each character of the strings one by one from left to right. Since the first character in `10` is `1`, and `1` comes before `2`, `10` is considered to come before `2` even though numerically `2` is supposed to come first.

```java
String string1 = "123";
String string2 = "9";

if (string1.compareTo(string2) < 0) {
  System.out.println("string1 comes first");
}
else if (string1.compareTo(string2) > 0) {
  System.out.println("string2 comes first");
}
else {
  System.out.println("the strings are equal");
}
```

challenge

What happens if you:

- Change `string1` to `"apple"`?
- Change `string2` to `"Apple"` in your current code?
- Change `string1` to an empty string `""` in your current code?

Letters vs. Numbers vs. Empty Strings

Lexicographically speaking, empty strings always come first, followed by numbers, then uppercase letters, and finally lowercase letters.