Strings Can Look Like Values

`String`, or `string`, is a class that represents text. Technically its value is stored as a collection of `char` objects.

Since it is a class, it is a reference type. In some cases its behavior looks like a value type:

> A `string` reference will always point to the original object, so “modifying” one reference to a `string` will not affect other references
>
> Comparing strings with the equality operator (`==`) performs a value, not referential, comparison

Here’s are two examples of the first behavior (modifying one reference doesn’t affect the others):

```
// Example 1
string dog = "chihuahua";
string tinyDog = dog;
dog = "dalmation";
Console.WriteLine(dog);
// Output: "dalmation"
Console.WriteLine(tinyDog);
// Output: "chihuahua"

// Example 2
string s1 = "Hello ";
string s2 = s1;
s1 += "World";
System.Console.WriteLine(s1);
// Output: "Hello World"
System.Console.WriteLine(s2);
// Output: "Hello"
```

The can be explained by the fact that strings are *immutable*: they cannot be changed after they are created. Anything that appears to modify a string actually returns a new `string` object.

Here’s an example of the second behavior (value-like comparisons):

```
string s = "hello";
string t = "hello";
// b is true
bool b = (s == t);
```

Typically we want to compare strings by value, so this makes it easier to write in code and it also gives the C# compiler flexibility in how it implements the program (it doesn't have to worry about where the actual string value is stored).

☑**Instructions**

1.
Create two `string` variables with the same value: `"immutable"`.

Hint

Declare the each variable on its own line.

Make sure you spelled "immutable" correctly!

2.
Compare the two variables using `==` and print the result.

Why does this return `true`?

Hint

With `==`, strings are tested for value equality, not referential equality.

3.
Now repeat the process with two `Object` variables:

- Construct two new `Object` instances and store them in two new variables
- Compare them with `==`

Make sure to call `new Object()` twice. Why are the results different?