Strings
Learning Objectives

- To be able to create and manipulate String values
- To be able to compare String values
Aside: Literal Values

- Literal values are "Hard-coded" values that are written in the code exactly as how they should be evaluated.
- Used most often for initializing a variable or as part of an expression

```java
int a = 3; // 3 is an int literal value
double b = a * 3.14; // 3.14 is a double literal
String s = "3.14"; // "3.14" is a string literal
```
Strings

- Strings hold sequences of characters (a, b, c, $, etc)
- Can perform operations on strings like concatenation and others
- Anything between "" is a string literal

**Strings** are “objects” of the String class, although they behave in many ways like a primitive type, so we study it now.
String Variable Declaration & Initialization

String variables work just like int, double, and boolean variables: declare them by writing the type & name of the variable and then give them an initial value.

```
String variableName = "stringLiteral";
String firstName = "Lisa";
```
String values

A String holds a sequence of characters

- Characters include things like 'a', 'b', '1', '$', '%', '.', etc.

These characters are stored in a sequence, and are numbered from the front of the sequence starting with 0. The last element is at index length – 1.

```
String example = "Hello!";
    012345
String birthday = "August 29";
    012345678
String empty = "";
```

We usually start counting at 0 in programming. Will see this more with arrays :)

String values: null

A String, since it is technically an object type, can be initialized to a null reference.

A null reference means that the variable does not refer to a space in memory.

```java
String variableName; // default String variable value is null
String nulledVar = null; // this sets a variable to null explicitly
```

⚠️ More on null in future lectures about objects. Just keep this in the back of your mind for now. ⚠️
String operations: Concatenation

Use the + or += operators to concatenate (combine) two Strings

```java
String a = "Serena";
String b = " Williams";
String c = a + b;
System.out.println(c);  // prints Serena Williams
```
String operations: Concatenation

Use the `+` or `+=` operators to append a primitive type value to a String

- will automatically convert that value to String

```java
String a = "Serena";
String b = " Williams";
String c = a + b + 100;
System.out.println(c); // prints Serena Williams100
```
Aside: Object methods and .

The + and += operator on strings is somewhat unique. Normally performing an operation on an object requires different syntax: the . operator.

```java
String a = "Serena";
String b = " Williams";
String c = a.concat(b); // same as a + b
System.out.println(c); // prints Serena Williams
```

There is NO space around the .
**String methods: length()**

The `length()` method returns the number (an `int`) of characters in the string, including spaces and special characters like punctuation.

```java
String a = "Serena";
int len = a.length(); // S, e, r, e, n, a is 6 characters
System.out.println(len); // prints 6
```
String methods: **substring()**

`substring(int from, int to)`

- returns a new string with the characters in the current string starting with the character at the `from` index and ending at the character before the `to` index.

```java
String a = "Serena";
String b = a.substring(0, 3);
System.out.println(b); // prints "Ser"
String c = a.substring(2, 4);
System.out.println(c); // prints "re"
```
String methods: `substring()`

`substring(int from)` returns a new string with all the characters in the current string starting after the character at the `from` index.

```
String a = "Serena";
String c = a.substring(3);
System.out.println(c); // prints "ena"
```
String methods: `indexOf()`

`indexOf(String str)` searches for the string `str` in the current string and returns:

- the index of the beginning of `str` in the current string,
- or `-1` if it isn’t found

```
String a = "Serena";
int x = a.indexOf("er"); // x has value 1
int y = a.indexOf("ena"); // y has value 3
int z = a.indexOf("sa"); // z has value -1
```
**String methods: `charAt()`**

`str.charAt(int index)` returns the `char` at position `index` in the input `str`:

- `index` must be between `0` and `str.length() - 1`
- return type is `char`, which is a data type used for storing individual characters

```java
String a = "Serena";
char x = a.charAt(0); // x has value 'S'
char y = a.charAt(2); // y has value 'r'
char z = a.charAt(5); // z has value 'a'
```
Comparing Strings

Strings (and objects) **cannot** be compared using operators like `==`, `<`, `>`. The method `first.compareTo(String second)` compares two strings character by character.

- If they are **equal**, it returns 0
- If the first string is alphabetically ordered **before** the second string, it returns a **negative number**
- If the first string is alphabetically ordered **after** the second string, it returns a **positive number**
Comparing Strings

```java
/// S comes before W in the alphabet
String a = "Serena";
String b = "Williams";
System.out.println(a.compareTo(b)); // prints -4;
System.out.println(b.compareTo(a)); // prints 4;
```

Figure 2: compareTo returns a negative or positive value or 0 based on alphabetical order
String equality

DO NOT USE `==` TO CHECK IF TWO STRINGS ARE EQUAL!
**String equality**

The `equals(String other)` method compares the two strings character by character and returns a `boolean`.

```java
String a = "Serena";
String b = "Williams";
System.out.println(a.equals(b)); // prints false
System.out.println(a.equals(a)); // prints true
```

`compareTo`, `equals` and most string methods are case-sensitive!

"HI".equals("hi"); // returns false
Live Demo: StringManips.java

Write a program StringManips.java that does two things

Problem 1:

• Given a string, we will print a new string made of 3 copies of the last 2 characters of the original string.

Problem 2:

• Given a string, the program will print a version without both the first and last characters

Both assume the input strings have length \( \geq 2 \).