Strings & Characters
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.

```java
String variableName = "stringLiteral";
String firstName = "Lisa";
```
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.

```java
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**
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()`

`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"
```

Serena
012345
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.

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

Serena
012345
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

```java
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

```
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.

```
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
```
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 $\geq 2$. 
The `char` Data Type
Strings

Recall that Strings are sequences of characters:

"Harry Smith" → string of 11 characters including space (' ')
"215-898-3500" → string of 12 characters including digits and '-'
"a" → string of one character, 'a'
""" → empty string (string of 0 characters)
String Iteration Toolkit

Given a String s...

- determine its length using `s.length()`
- get a character at a given position `i` using `s.charAt(i)`.
  - `i` must be between 0 and `s.length() - 1`

```java
for (int i = 0; i < s.length(); i++) {
    System.out.println(s.charAt(i));
}
```
char

char is a primitive data type used to store a single character.

- char literals are expressed using single quotes ('')
  - 'a', '8', ' ' are all char values
  - 'aa' is not a valid char because it expresses two characters!
- char values are represented using letters/digits/punctuation, but they are represented internally as small integers!
  - Computers only understand numbers (just 0 and 1, really), so we have to be clever about how we get them to think about symbols like letters.
ASCII (American Standard Code for Information Interchange) is a system of assigning numbers to characters in order to store them in computers.

- System allowed for representing 128 different symbols using numbers 0–127.
- Fine back in the day, but quite limited, especially outside of American English alphabet.
Unicode is also a system of assigning numbers to characters in order to store them in computers.

- System is very complicated—not exactly as simple as 'A' == 65
- Can express characters from multiple alphabets and also emoji (🌈 🌴 😎)

Technically, Java uses the 16 bit Unicode standard to map char values to integers, but we'll often say "ASCII" (as-kee) for shorthand.
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>ASCII</th>
<th>Unicode</th>
<th>char</th>
<th>STRINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>start of heading</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>start of text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>end of text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>end of transmission</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>enquiry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>acknowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>bell</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>backspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>horizontal tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>012</td>
<td>LF</td>
<td>NL line feed, new line</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>013</td>
<td>VT</td>
<td>vertical tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>014</td>
<td>FF</td>
<td>NP form feed, new page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>015</td>
<td>CR</td>
<td>carriage return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>016</td>
<td>SO</td>
<td>shift out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>017</td>
<td>SI</td>
<td>shift in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE</td>
<td>data link escape</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1</td>
<td>device control 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2</td>
<td>device control 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3</td>
<td>device control 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4</td>
<td>device control 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK</td>
<td>negative acknowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN</td>
<td>synchronous idle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB</td>
<td>end of trans. block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>030</td>
<td>CAN</td>
<td>cancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM</td>
<td>end of medium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>032</td>
<td>SUB</td>
<td>substitute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>033</td>
<td>ESC</td>
<td>escape</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>035</td>
<td>GS</td>
<td>group separator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASCII, Unicode, & char
**char Values as Numbers**

"Given a char c, how can I ask if it's a lowercase letter from 'a'–'z'?"

- Since char values have a number representation, it means that we can straightforwardly order them using > and <

```java
if ('a' <= c && c <= 'z') {
    System.out.println(c + " is a lowercase letter");
}
```
"Given a char c, how can I turn it from an uppercase letter to a lowercase letter?"

- Since char values have a number representation, it means that we can modify them using simple arithmetic
- 'a' – 'z' are 97–122 respectively
- 'A' – 'Z' are 65–90
- The difference between a lowercase and uppercase letter is 32

```
char lowercase = c + 32;
```
Converting a `char` to a `String`

Use concatenation to append a `char` on to the end of an empty string ("") in order to get a `String` that contains just the `char` value.

```java
char c = 'a';
String s = "" + c;
System.out.println(s); // prints "a"
```

⚠️⚠️⚠️ Use this for Letter Viewer in HW1! ⚠️⚠️⚠️