

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

```
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
```

- Anything between "" is a string literal

Strings

- **Strings** are “objects” of the String class
- String is an object type:
 - Strings hold sequences of characters (a, b, c, \$, etc)
 - Can perform operations on strings like concatenation and others
- Write `String variableName;` to declare a string object

String initialization

- There are two ways to initialize a string
- `String variableName = new String("stringLiteral");`
 - Example: `String name = new String("Lisa");`
- `String variableName = "stringLiteral";`
 - Example: `String name = "Lisa";`

String values: null

- A String, like all objects, can be initialized to a `null` reference
- **A null reference** means that the variable does not refer to a space in memory
 - **String variableName = null;** creates a null string object

More on null in future lectures about objects. Just keep this in the back of your mind :)

String operations

- **Concatenation**
- Use the “+” or “+=” operators to concatenate (combine) two Strings

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


String operations

- Using “+” or “+=” operators to append a primitive type value to a String will automatically convert that value to String

```
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.

- Example: If we have

```
String a = "Serena";  
String b = " Williams";
```

- We can do:

```
String c = a + b; // assigns "Serena Williams"
```

- Or equivalently:

```
String c = a.concat(b); // assigns "Serena Williams"
```

There is NO space around the '.'



String methods

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

```
String a = "Serena";  
a.length(); // returns 6
```

String methods

- `String 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 (if the **to** index is not specified it will contain the rest of the string)

```
String a = "Serena";  
          0 1 2 3 4 5
```

```
String b = a.substring(0, 3);           0 1 2  
System.out.println(b); // prints "Ser"  
String c = a.substring(3);           3 4 5  
System.out.println(c); // prints "ena"
```

String methods

- `int indexOf(String str)` method 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";  
          0 1 2 3 4 5
```

```
int x = a.indexOf("er"); // returns 1  
int y = a.indexOf("ena"); // returns 3  
int z = a.indexOf("sa"); // returns -1
```

Comparing Strings

- Strings (and objects) **cannot** be compared using operators like `==` and `<` or `>`
- The method `compareTo` 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

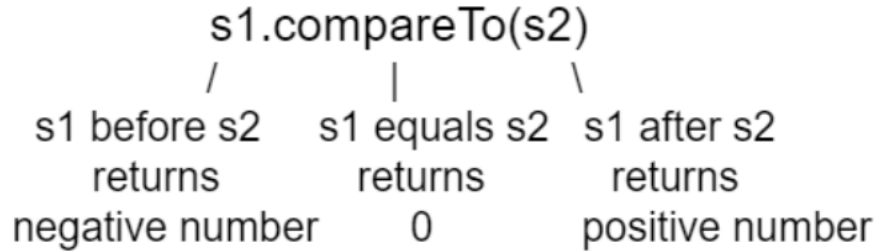


Figure 2: `compareTo` returns a negative or positive value or 0 based on alphabetical order

```
String a = "Serena";
```

```
String b = "Williams";
```

S comes before **W** in the alphabet

```
a.compareTo(b); // returns negative number -4
```

```
b.compareTo(a); // returns positive number 4
```

String equality

DO NOT USE ==

- The equals method compares the two strings character by character and returns true or false

```
String a = "Serena";
```

```
String b = "Williams";
```

```
a.equals(b); // returns false
```

```
a.equals(a); // returns 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

- Problem1:
 - Given a string, we will print a new string made of 3 copies of the last 2 characters of the original string.
- Problem2:
 - Given a string, the program will print a version without both the first and last characters
- Both assume the input strings have length ≥ 2