Middlebury

CSCI 201: Data Structures

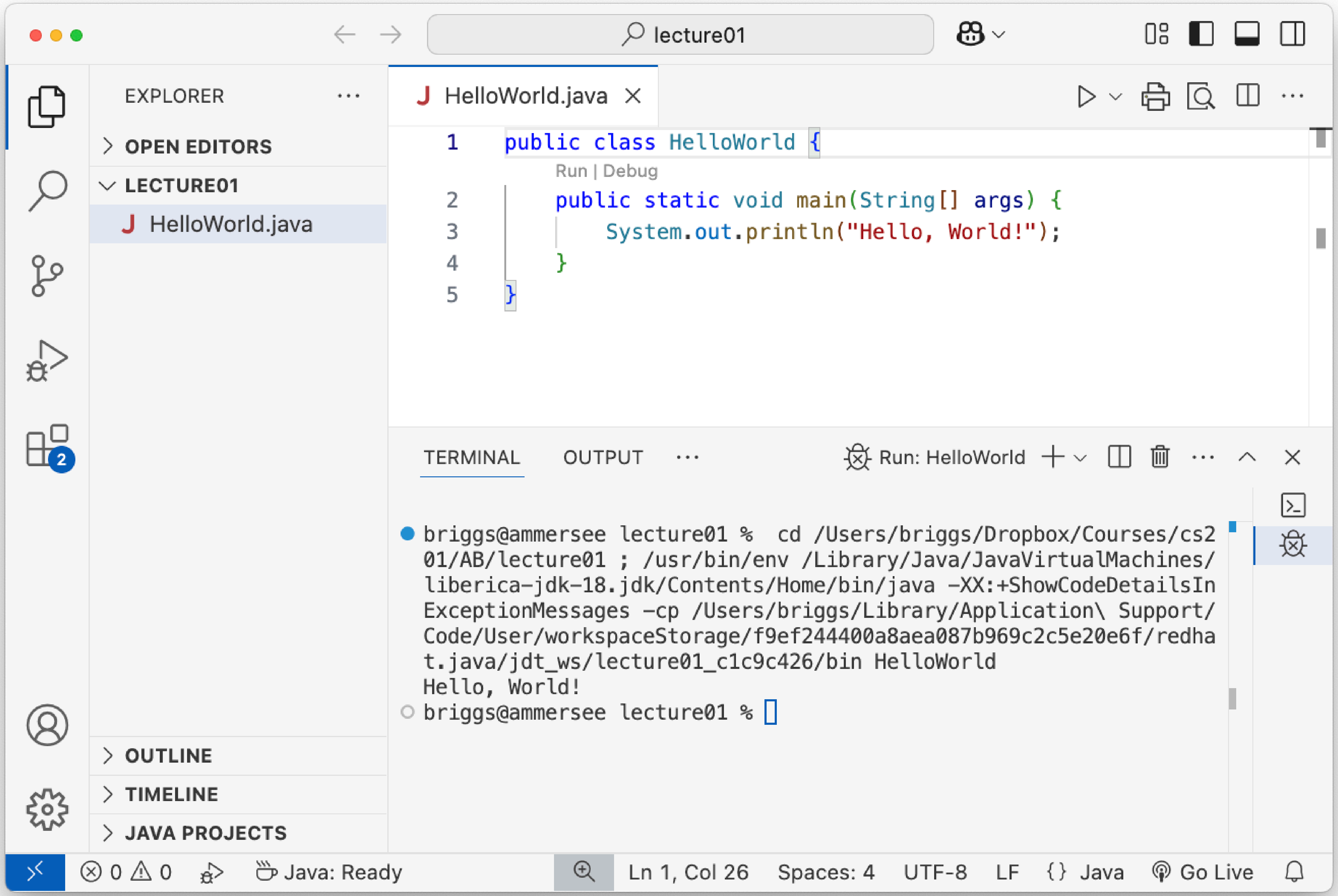
Spring 2025

Lecture 1W: Java Control Structures

Goals for today

- Why do we need a `public static void main`?
- Use a few `String` methods: `length`, `charAt`, `equals`, `replace`, `subtring`.
- Write **functions**.
- Declare and assign `boolean` variables.
- Use comparative operators: `<`, `>`, `<=`, `>=`, `==`, `!=`.
- Use logical operators: `&&` (and), `||` (or), `!` (not).
- Write `if`, `if/else`, `if/else if/else` statements.
- Write `for`-loops and `while`-loops.

How did setting up VS Code and Java go?



```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

```
briggs@ammersee lecture01 %  cd /Users/briggs/Dropbox/Courses/cs2
01/AB/lecture01 ; /usr/bin/env /Library/Java/JavaVirtualMachines/
liberica-jdk-18.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsIn
ExceptionMessages -cp /Users/briggs/Library/Application\ Support/
Code/User/workspaceStorage/f9ef244400a8aea087b969c2c5e20e6f/redha
t.java/jdt_ws/lecture01_c1c9c426/bin HelloWorld
Hello, World!
briggs@ammersee lecture01 % 
```

What's with the `public static void main`?

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hi CS 201!");
    }
}
```

- Each `.java` file has a single `public class`.
- Name of the file matches this class name, e.g. `HelloWorld.java` for the `HelloWorld` class.
- Must have a `public static void main` (**PSVM**) method defined in this class if you want to run it.
- Larger projects can have multiple files: only one **PSVM** is needed in one of these files.

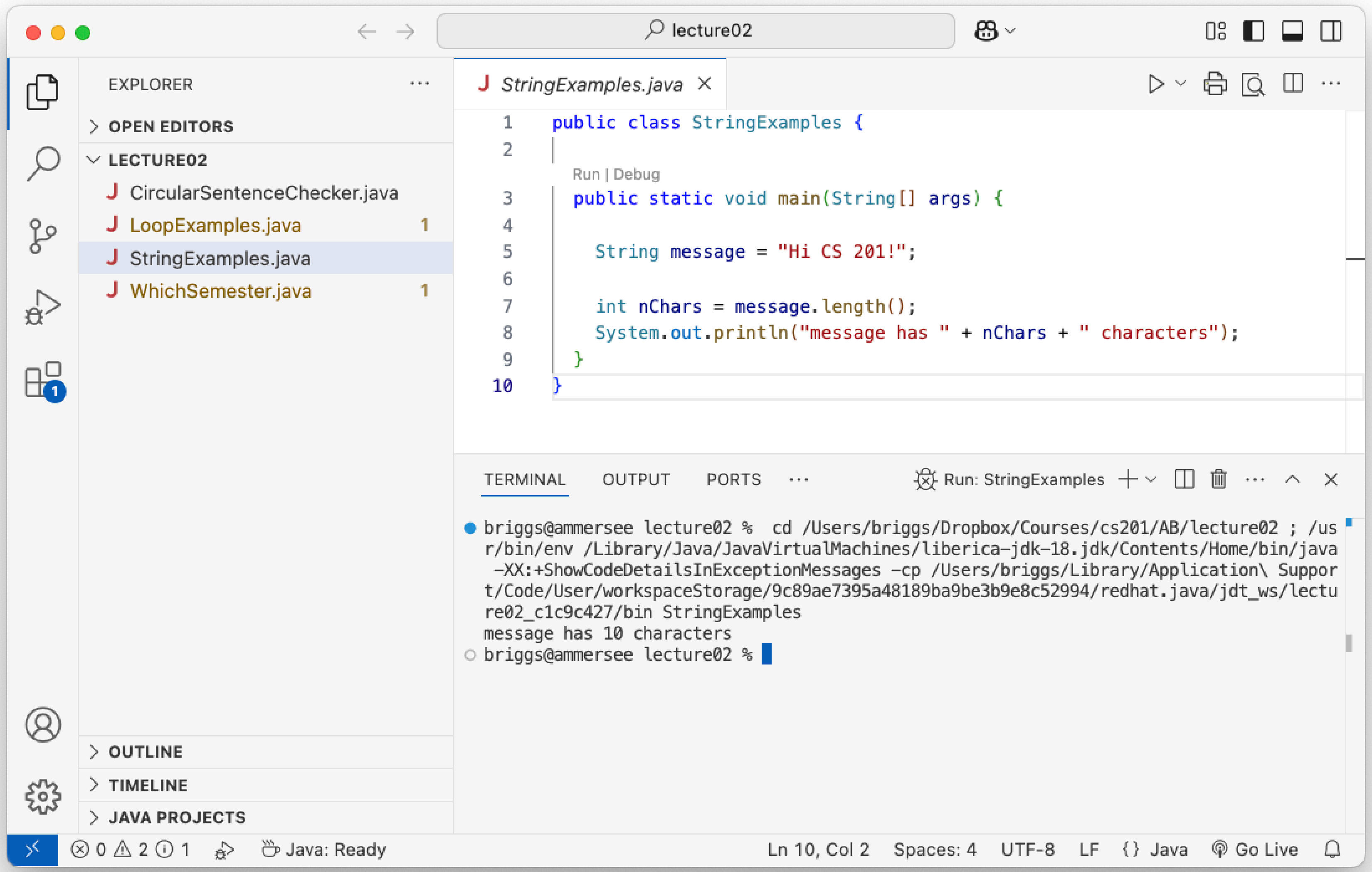
Breaking down `public static void main` (PSVM)

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hi CS 201!");
    }
}
```

- `public`: accessible by other classes
- `static`: don't need to create an instance of the `class` (i.e. an object), can use the `class` name directly via the dot operator `.` (e.g. we can call `HelloWorld.main`)
- Including `static` is similar to excluding `self` in a `Python` class method
- `void`: this method returns nothing
- `main`: starting-point of the program

Following along with the examples for today

- Download the file from the `code` link at go/cs201 for today's class.
- Extract the contents (`lecture02`) and then move this folder to your `cs201` folder.
- Open **VS Code**, then `File -> Open Folder`, navigate to the `lecture02` folder and open it.



```java
public class StringExamples {

    public static void main(String[] args) {

        String message = "Hi CS 201!";

        int nChars = message.length();
        System.out.println("message has " + nChars + " characters");
    }
}
```

```
briggs@ammersee lecture02 %  cd /Users/briggs/Dropbox/Courses/cs201/AB/lecture02 ; /us
r/bin/env /Library/Java/JavaVirtualMachines/liberica-jdk-18.jdk/Contents/Home/bin/java
 -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/briggs/Library/Application\ Suppor
t/Code/User/workspaceStorage/9c89ae7395a48189ba9be3b9e8c52994/redhat.java/jdt_ws/lectu
re02_c1c9c427/bin StringExamples
message has 10 characters
briggs@ammersee lecture02 %
```

A note about `Java` primitive types

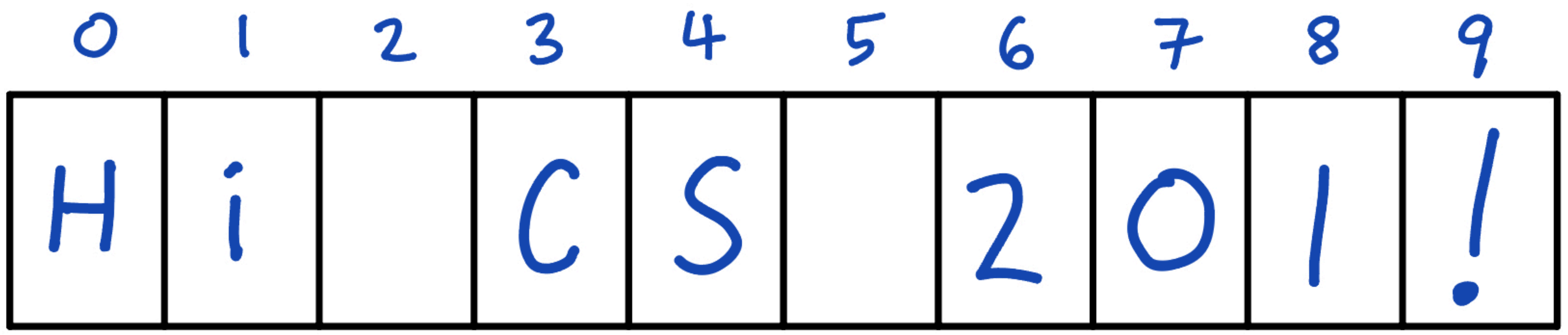
Name	Type	Range
`boolean`	Boolean	`true` or `false`
`char`	16-bit Unicode character	`\u0000` (0) to `\uffff` (65,536)
`byte`	8-bit integer	$[-128, 127]$
`short`	16-bit integer	$[-2^{15}, 2^{15} - 1]$
`int`	32-bit integer	$[-2^{31}, 2^{31} - 1]$
`long`	64-bit integer	$[-2^{63}, 2^{63} - 1]$
`float`	32-bit real	$[-3.4028\text{e}38, 3.4028\text{e}38]$
`double`	64-bit real	$[-1.7977\text{e}308, 1.7977\text{e}308]$

Note: `String` is not a primitive type!

Practice with `String`s! Please open `StringExamples.java`

The `String` class has some useful **methods**. We can call them using `.` on an instance of a `String`:

- `length()`: returns the number of characters.
- `charAt(i)`: returns the `char` at (integer) index `i`.
- `toUpperCase()`: returns `String` in which each character is converted to upper-case.
- `toLowerCase()`: returns `String` in which each character is converted to lower-case.
- `replace(oldChar, newChar)`: returns `String` in which each character equal to `oldChar` is replaced with `newChar`.
- `equals(otherStr)`: returns whether all `char`s match those in `otherStr` (in order),
- `substring(start, end)`: returns `String` with `char`s from `start` to `end`.



What were some of the types returned by the `String` methods we used?

A little more about the difference between `static` and dynamic methods

Here, we have an instance of a `String` (object), `length()` is not `static`.

```java
public class HelloWorld {
    public static void main(String[] args) {
        String message = "Hello, World!";
        int nChars = message.length();
    }
}
```

Here, we want to call the `speak` method of the `HelloWorld` class, but since it's called from `HelloWorld.main` (which is `static`), that method needs to be declared `static` too. Otherwise we'll get a compiler error: `non-static method speak() cannot be referenced from a static context`.

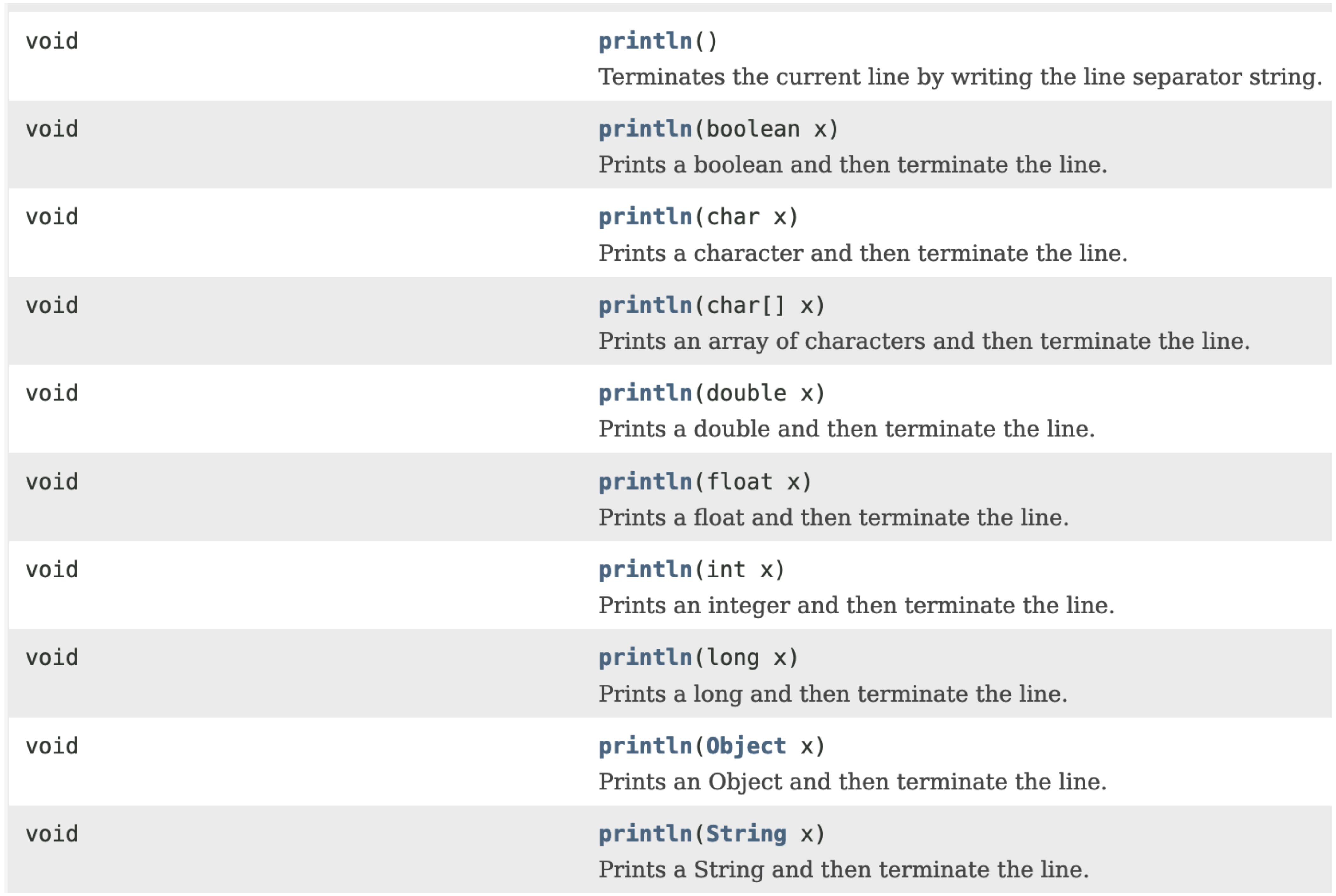
```java
public class HelloWorld {
    public static void speak() {
      System.out.println("Hello, World!");
    }

    public static void main(String[] args) {
        speak(); // can also use HelloWorld.speak()
    }
}
```

But! There's another way: we can create an **instance** of the `HelloWorld` class and then make the `speak` function dynamic (more on this next week).

Use a search engine (e.g. Google) to search for "java PrintStream"

Then look for a function we've used a lot so far... (hint: to print things).
What does this function return?



`void`	`println()` Terminates the current line by writing the line separator string.
`void`	`println(boolean x)` Prints a boolean and then terminate the line.
`void`	`println(char x)` Prints a character and then terminate the line.
`void`	`println(char[] x)` Prints an array of characters and then terminate the line.
`void`	`println(double x)` Prints a double and then terminate the line.
`void`	`println(float x)` Prints a float and then terminate the line.
`void`	`println(int x)` Prints an integer and then terminate the line.
`void`	`println(long x)` Prints a long and then terminate the line.
`void`	`println(Object x)` Prints an Object and then terminate the line.
`void`	`println(String x)` Prints a String and then terminate the line.

Writing our own function to see if the message is friendly

```java
public static boolean isFriendlyMessage(String s) {
  return s.contains(":)");
}
```

General format of an `if`-statement

```
if (condition1) {
  // statements executed when condition1 is true
} else if (condition2) {
  // statements executed when condition1 is false but condition2 is true
} else if (condition3) {
  // statements executed when condition1 and condition2 are false but condition3 is true
} else {
  // statements executed when condition1, condition2 and condition3 are false
}
```

Technically, we don't need braces for single-line blocks, but we still suggest using them.

How to put together conditional statements?

Operators in Java are mostly the same as Python, except the logical operators.

Operator	Type	Notes	(Python)
`+`, `-`	arithmetic	add, subtract	`+`, `-`
`*`, `/`	arithmetic	multiply, divide (note `5/4` gives `1`)	`*`, `/`
`%`	arithmetic	modulus (e.g. `5 % 2` gives `1`)	`%`
`<`, `<=`	comparative	less than, less than or equal to	`<`, `<=`
`>`, `>=`	comparative	greater than, greater than or equal to	`>`, `>=`
`==`	comparative	equality (ONLY for primitive types: `boolean`, `byte`, `short`, `int`, `long`, `float`, `double`, `char`)	`==`
`!`	logical	logical NOT	`not`
`&&`	logical	logical AND	`and`
`||`	logical	logical OR	`or`

Example: please open `WhichSemester.java`

```java
public class WhichSemester {

  // x is the month, should be between [1, 12]
  public static String whichSemester(int x) {
    if (x == 1) { // January
      return "Winter";
    } else if (x > 1 && x <= 5) { // February - May
      return "Spring";
    } else if (x > 8 && x <= 12) { // September - December
      return "Fall";
    }
    return ""; // no semester
  }

  public static void main(String[] args) {
    String semester = whichSemester(4);
    if (semester.length() == 0) { // can also use semester.isEmpty()
      System.out.println("We're on break!");
    } else {
      System.out.println("It's the " + semester + " semester");
    }
  }
}
```

General format of a `for`-loop

```java
for (initialization; continue_condition; step_statement) {
  // statements executed while continue_condition is true
}
```

Example:

```java
for (int i = 0; i < 10; i++) {
  System.out.println(i);
}
```

Please open `LoopExamples.java`

Be careful with the scope of variables

Variables declared within `{}` can only be used within the `{}` or within nested blocks (conditionals, loops).

```java
for (int k = 0; k < 10; k++) {
  int kSquared = k * k;
  System.out.println(kSquared);
}
System.out.println(kSquared); // compiler error: cannot find symbol
```

We can also write the previous `for`-loops using a `while`-loop.

```java
int i = 0;
while (i < message.length()) {
  c = message.charAt(i);
  System.out.println(c);
  i++;
}
```

or:

```java
int i = 0;
while (true) {
  if (i >= message.length()) break; // one-line if-statement!
  c = message.charAt(i);
  System.out.println(c);
  i++;
}
```

Exercise: write a circular sentence checker in `CircularSentenceChecker.java`

A circular sentence is a sentence in which:

- The first letter of the sentence is the same as the last letter of the sentence.
- The last letter of one word is the first letter of the next word.
- Assume there is one space between words.

Examples :-)

- you use extra avocado on noodles strangely
- Middlebury yields some extra adventurous students studying gladly yonder researching good data at the ecosystem

Possible solution to circular sentence checker

```java
public class CircularSentenceChecker {
    public boolean isCircularSentence(String sentence) {
      sentence = sentence.toLowerCase();
      int n = sentence.length();
      char first = sentence.charAt(0);
      for (int i = 0; i < n; i++) {
          if (sentence.charAt(i) == ' ') {
              if (sentence.charAt(i - 1) != sentence.charAt(i + 1)) return false;
          }
      }
      return (sentence.charAt(n - 1) == first);
    }

    public static void main(String[] args) {
      boolean result = isCircularSentence("you use extra avocado on noodles strangely");
      System.out.println(result); // prints true
    }
}
```

One last note: comments (either single line `//` or multiline `/* */`)

```
// this is a single line comment

/*
 this is a multiline comment
 everything here will be ignored by the compiler
 until it encounters the following symbols (on the next line)
*/
```

Next week, we'll also look at ways to document our code.

See you Friday!

- Please complete / update the Introduction Form if not done yet.
- Read through the Lab 1 write-up (see assignments column at go/cs201).