INTRODUCTION TO JAVA

CS302 – Introduction to Programming
University of Wisconsin – Madison
Lecture 2

By Matthew Bernstein – matthewb@cs.wisc.edu

Our First Java Program

```
public class Main
{
        public static void main(String[] args)
        {
                // Say hello to the world.
                System.out.println(“Hello World!”);
        }
}
```

Case Sensitivity

- Java is case sensitive
- “Main” does not equal “main”

Statements

- A **statement** represents a single Java instruction
- Statements usually occupy a single line of a Java program
- All statements must be terminated by a semi-colon
    - Example:

```
int x = 4;
```

- The first statement that is executed is the first statement that appears in your “main” method

White Space Agnosticism

- White space characters include:
  - Single space, line-break, tab, etc…
- Java is agnostic to white space
- 1 white space character *between elements* is treated the same as 100 white space characters between elements
  - Example:

```
int x = 4;
int    x    =    4;
```

Comments

- Java allows you to insert text into your program that will not affect the program at all (treated as white space)
- To write a single line comment, use “//”
  - Anything following “//” on the same line will be ignored
  - Example:

```
// This text will be treated as white space
```

- To write a multi-line comment use “/*” and “*/”
  - Example:

```
/*
This text will be treated as white space
*/
```

Curly Braces

- Sections of code are enclosed by curly brackets
- An essential part of Java structure
- Common styles of writing curly braces (pick one and use consistently):

```
public static void main(String[] args) {
        // CODE HERE
}

public static void main(String[] args)
{
        // CODE HERE
}
```

Comments and Coding Style

- Comments should describe the purpose of a section of code
- White space should be used to make the code readable and organized
- Your use of comments and white space is extremely important!
- Convoluted and messy code will:
  - Result in bugs
  - Be difficult to integrate with other code
  - Will be difficult to extend in order to build new functionality

The Boilerplate

- The stuff not to worry about (for now):

```
public class Main
{
        public static void main(String[] args)
        {

                // YOUR CODE GOES HERE
        }
}
```

The Important Stuff (for now…)

```
public class Main
{
        public static void main(String[] args)
        {
                // Say hello to the world.
                System.out.println(“Hello World!”);
        }
}
```

Calling a Method

- Method name followed by parenthesis
- Inside the parenthesis, we place the argument
- Basic template: `methodName(argument);`
- Example:

```
System.out.println( “Hello World!” );
```

The code being executed by println simply prints its argument to the console (we don’t see that code here). println “lives in” System.out

Methods (a.k.a. Functions)

- A **method** is a section of code that carries out a particular task (example: add two numbers, sort a list, etc.)
- A method has a name
- A method can accept parameters, called **arguments**, that it uses to complete its task
- The code within a method can be executed by **calling** that method

Where is “System.out.println()”?

- Calling a method executes code located somewhere else. Where is “System.out.println()”?
- It comes from the **Java Class Library**, which is a large body of reusable Java code that has already been written to help you solve common problems. It comes prepackaged with the Java compiler.
- Things like writing to the console would be challenging to do from scratch and would involve interacting with the underlying system.

Classes (and other things not to worry about right now)

- A **class** is a fundamental structure used in Java programs. We will cover classes extensively later in the course. For now, your "main" method should be contained in a class that has the same name as your java file (see HelloWorld example).

- An **Access Modifier** ("public", "private", etc.) are used to describe what elements of your program have access to other elements of your program. Don't worry about them for now…

Errors

- Two Types of Errors:
  - **Compile-time Errors (“Syntax Errors”)** – There is something wrong with the rules of the language and the compiler is unable to translate your code to Java bytecode.
  - **Run-time Error** – The program is syntactically correct and can be compiled, but doesn’t do what it is supposed to do. Some run-time errors can cause the program to crash.
    - For example, trying to divide a number by 0 will cause the program to crash.

Exception Trace

- When a run-time error occurs that causes the program to crash, Eclipse will output an **Exception Trace**
- Example of an exception trace you might see in the Eclipse console when your program crashes:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
	at MeanMedianMode_Methods.bubbleSort(MeanMedianMode_Methods.java:168)
	at MeanMedianMode_Methods.calculateMedian(MeanMedianMode_Methods.java:118)
	at MeanMedianMode_Methods.main(MeanMedianMode_Methods.java:31)
```

Cool CS Link of the Day

- Visualizing Facebook's global network:
- https://www.facebook.com/note.php?note_id=469716398919

facebook
December 2010