Week 1

- Terminology and a few commands
- Hello, World
- Introduction to variables and data types
- Operators
- User input
- Dealing with problematic code

Follow along http://tiny.cc/javaslides

Terminology and commands

Before we start writing code, let's cover a few terms, commands, and other useful information that you will need to know to effectively continue with the next sections.

Code editor: this is a program used to write anything, including code. Examples include Nodepad, Notepad++, Sublime Text, Atom, IntelliJ. If you're unsure about white editor to use, start with Atom.

What ever you use, make sure you disable syntax highlighting.

Terminal/Command Prompt: this is a program used to *run* other programs. Your computer already has this program installed.

- To open the command prompt in Windows, click on the Start menu and search for "*command*".
- To open the terminal in Macos, open Spotlight `Cmd + Space` and search for "*terminal*".

Command: a computer program similar to your browser, text editor, and music player, but with a few differences. First, you run commands in your terminal or command prompt. And second, commands generally don't have a GUI, meaning that you won't see a window or be able to click on buttons when you run them, but just text.

"Running a command": when the phrase "*running a command*" or "*run this command*" is used, the intention is that you open your command prompt, carefully type in the example command, and press the Enter key to run it.

Compile: or "*compiling*" is the act of taking code that you wrote and preparing it so that your computer can run it. Not all programming languages need to be compiled, but Java does.

Commands

- `cd` : lets you move from one folder to another folder. Remember the phrase "*change directory*" (change = c, directory = d)
- `dir` (Windows) or `ls` (Macos/Linux): this command is used to list all of the files and folders in the folder you are currently in.
- `pwd` (Macos/Linux): this command will tell you what folder you are on right now. Useful when you want to `cd` into another folder.

Commands

- `javac` : the Java compiler. For example, if you had a file named "HelloWorld.java" and you wanted to compiled it, you would do so by running `javac HelloWorld.java` in your terminal/command prompt.
- `java` : another Java related command. This one is used to actually run your compiled program. If you wanted to run the HelloWorld program from the example above, you would run `java HelloWorld` .

Reading these slides: code

Below is an example of how code is represented in these slides. Notice the gray background color. When you see code in slides, it will usually mean that you should write it in your text editor.

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

Reading these slides: commands

Commands will a lot like code with the exeption that they will have a ">" at the beginnig of the line. That's how you will know if it is a command instead of Java code. When you see this, it means that you should open your terminal/command prompt and run the command or commands.

```
> javac HelloWorld.java
> java HelloWorld
```

The example above is telling you to run `javac HelloWorld.java`, press enter, then run `java HelloWorld` and press enter again.

Hello, World and saving your code

Now we can get started writing code. In this section we're going to use a text editor to write a Java program, and then go into the terminal/command prompt to compile and run your program.

Make sure you create a folder that you will have access to in the future, and remember where it is as well. This will be your "Java Class Code" folder from now on. You should probably name it "*JavaClassCode*".

In your text editor, create a new file and write the program below. In your code folder, create a new folder and name it "*week01*". In that new folder save the new file and name it "*HelloWorld.java*". Write this code and save it to your file:

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

Go to your terminal and run the following commands (make sure you're in the right folder)

```
> javac HelloWorld.java
> java HelloWorld
```

If everything worked you should see `Hello, World!` as the output of running the last command.


You're a Java Programmer!!!

Change something about your code and see what happens. What happens if you change *"Hello, World"*? Do you see your updates? What happens if you remove the word void? Did you get an error? Make any change you'd like. Below is a reference to the original code if you want to undo any changes you make.

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

Variables

> A variable is a symbolic name for (or reference to) information. The variable's name represents what information the variable contains. They are called variables because the represented information can change but the operations on the variable remain the same. [1]

Three things are required to create a new variable in Java: the data type, a name, and an optional value.

A valid variable name can be made up of letters, numbers, dollar signs `$` and underscores `_`. They should always begin with letters, however (`$` and `_` are valid characters at the beginning of variable names, but this should be avoided.)

```
String activityName = "Java programming";
int age = 47;
double pi = 3.14159;
```

Additional terminology

Declaration: as previously mentioned, a value is optional when creating a variable. When a variable is created and it does not have a value, this is a way of *declaring* the variable, meaning you are telling your program that this variable exists and it is information that should be tracked, but that you do not yet know what the value is and will set it after the program starts running. Keep in mind that the variable's data type is only required when it is being *declared*.

```
int x;
```

Definition: a variable is *defined* when it is given a value. As shown in earlier examples, a variable can be declared and defined in one step, but that is not required. The first example below declares and then defines `x` and the one below that does the same but in one step:

```
int x;
x = 7;
```

```
int x = 7;
```

Data types

What's a type?

Remember the code you wrote in *HelloWorldWithArguments.java*?

```
public class HelloWorldWithArguments {
  public static void main(String[] args) {
    String name = "World";

    if (args.length > 0) {
      name = args[0];
    }

    System.out.println("Hello, " + name + "!");
  }
}
```

```java
public class DataTypesSample {
  public static void main(String[] args) {
    // "int" stands for "Integer"
    int age = 97;

    String name = "Marcos";
    String message = "Hello, " + name +
                     ", you are " + age +
                     " years old";

    System.out.println(message);
  }
}
```

- `byte`, `short`, `int`, and `long` are for integers, such as `1`, `99`, and `32423325`
- `float` and `double` are for floating-point numbers, such as `3.14`, `3.1`, and `89.12e32`
- `char` is for a single character, such as `a`, `B`, and `\n`
- `boolean` is for storing either `true` or `false` (think of this as on/off or yes/no)
- `String` is for a sequence of characters, or a string. For example, `Hello, World!`

- `byte` = 1 Byte, signed, -128 to 127
- `short` = 2 Bytes, signed, -32,768 to 32,767
- `int` = 4 Bytes, signed, -2^{31} to $2^{31}-1$
- `long` = 8 Bytes, signed, -2^{63} to $2^{63}-1$
- `float` = 4 Bytes, signed
- `double` = 8 Bytes, signed
- `char` = 2 Bytes, unsigned
- `boolean` = 1 Byte, unsigned

What happens if I use the wrong type?

```
public class BigByte {
  public static void main(String[] args) {
    byte oneTwentySeven = 127;
    byte oneTwentyEight = 128;

    System.out.println(oneTwentySeven);
    System.out.println(oneTwentyEight);
  }
}
```

Why?

You should use the smallest amount of data possible to accomplish a task, so that your computer doesn't run out of memory.

Operators

You're already using operators

```
String variableName = "string variable value";
```

Assigned Operator (1 of 2)

- `=`

```
int age = 92;

age = 93;
age = age + 1;
```

Arithmetic Operators

- `*`
- `/`
- `%`
- `+`
- `-`

Arithmetic Operators

```
int a = 37;
int b = 21;

int multiplication = a * b;
int division = a / b;
int modulos = a % b;
int addition = a + b;
int subtraction = a - b;
```

Assigned Operators (2 of 2)

- `+=`
- `-=`
- `*=`
- `/=`
- `%=`

```
int age = 92;

age += 1;
```

Unary Operators

- `++`
- `--`

```
int age = 92;

age++;
```

Pre-unary vs. post-unary

```
public class PostUnaryOperators {
  public static void main(String[] args) {
    int age = 92;
    int nextAge = age++;

    System.out.println("Age = " + age);
    System.out.println("Next Age = " + nextAge);
  }
}
```

```
Age = 93
Next Age = 92
```

Pre-unary vs. post-unary

```
public class PreUnaryOperators {
  public static void main(String[] args) {
    int age = 92;
    int nextAge = ++age;

    System.out.println("Age = " + age);
    System.out.println("Next Age = " + nextAge);
  }
}
```

```
Age = 93
Next Age = 93
```

Relational Operators

- `<`
- `>`
- `<=`
- `>=`

Equality Operators

- `==`
- `!=`

User Input

Now that we have written a program which has output and know much more about variables, let's update your code to work with user input. Commands can take arguments, so let's use an argument to make our Hello World program have the ability to say "Hello" to more than just "World" depending on if we run our program with an argument or not.

New Terminology!

Command arguments/parameters: sometimes you want to pass custom information to a command. This information is placed after the command name and is refered to as an argument or a parameter. For example, the command to compile a Java file is `javac` and this command takes a file name as one of its arguments. So if I wanted to compile a file named "HelloWorld.java" I would run `javac HelloWorld.java` where `javac` is the command and `HelloWorld.java` is the argument.

Input and output: there are many forms of input and output, but for the next sections, we mean input to be custom arguments you pass to your Java programs and output to be the text that you see after running a command in your terminal/commandline prompt.

In your text editor, create a new file and write the program below. Name this file "*HelloWorldWithArguments.java*" and save it in your "*week01*" folder.

```
public class HelloWorldWithArguments {
  public static void main(String[] args) {
    String name = "World";

    if (args.length > 0) {
      name = args[0];
    }

    System.out.println("Hello, " + name + "!");
  }
}
```

Notice some new things?

```
public class HelloWorldWithArguments {
  public static void main(String[] args) {
    String name = "World";

    if (args.length > 0) {
      name = args[0];
    }

    System.out.println("Hello, " + name + "!");
  }
}
```

Go to your terminal and run the following commands (make sure you're in the right folder)

```
> javac HelloWorldWithArguments.java
> java HelloWorldWithArguments Programmer
```

If everything worked you should see `Hello, Programmer!` as the output of running the last command.

Dealing with problematic code

You're going to make mistakes. This is true for *all* programmer, but specially in the beginning your code will fail to run more times than not. But this is ok. You'll be better and programming and better at using the tools at your disposal.

Reading compiler output

The compiler should become your friend. Learning how to understand what it is telling you will help accomplish this. When your code fails to compile, most of the time you will be given a very explicit reason why.

For example, try compiling `CompileError.java`. You should see something like this:

```
CompileError.java:3: error: cannot find symbol

    System.out.prntln("This is a string");
               ^
  symbol:   method prntln(String)
  location: variable out of type PrintStream

1 error
```

What can we get out of this?!?!?!

1. The file that the compiler had problems: `CompileError.java`
2. The line number in that file: `3`
3. The actual "thing" that is causing problems:

```
    System.out.prntln("This is a string");
               ^
  symbol:   method prntln(String)
  location: variable out of type PrintStream
```

4. Finally, the number of errors it found: `1 error`. Keep in mind that as you fix errors other may arise.

Reading runtime errors

Sometimes the compiler doesn't catch potential problems. This includes instances when there is nothing technically wrong with your code but just fail to check it will correctly run.

Compile and execute `RuntimeError.java` and let's talk about what's happening.

Running `javac RuntimeError.java` results in no errors, but when you run `java RuntimeError` you should see something like this:

```
Exception in thread "main" java.lang.NumberFormatExceptio
        at java.lang.NumberFormatException.forInputString
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at RuntimeError.main(RuntimeError.java:3)
```

What can we get out of this?!?!?!

1. The file that the compiler had problems: `RuntimeError.java`
2. The line number in that file: `3`
3. The exception that was thrown `NumberFormatException`
4. The stack trace:

```
at java.lang.Integer.parseInt(Integer.java:580)
at java.lang.Integer.parseInt(Integer.java:615)
at RuntimeError.main(RuntimeError.java:3)
```

New Terminology!

Exception: we'll cover this topic in much more detail in a later class, but for the time being think of exceptions as your code telling you that it ran into an issue. Different exceptions mean different issues, so this will help you get to the bottom of the problem.

Stack trace: stack traces are related to Exceptions and they include information about the files/lines that your code ran through before it ran into an error. Stack traces will help you find the exact location (file and line number) of your error.

Additional resources

- *Introduction and Setting up Your Environment* by Jim Wilson https://app.pluralsight.com/player?course=java-fundamentals-language&author=jim-wilson&name=java-fundamentals-language-m1&clip=0
- *Variables, Data Types, and Math Operators* by Jim Wilson https://app.pluralsight.com/player?course=java-fundamentals-language&author=jim-wilson&name=java-fundamentals-language-m3&clip=0

Reference list

1. Germain, https://www.cs.utah.edu/~germain/PPS/Topics/variables.html