Get Back to Programming

Java Edition

by Juman Byun

Introduction

I aim to give your feet wet on programming as quickly as possible. An intended audience is someone who had a year of programming experience at school but have not gotten paid to create computer programs.

I really would like to keep this material as short as possible. I will try to reference the official learning materials from the official Java website.

More conceptual, usual definitions were used for easier understanding rather formal definitions. There are so many books that will give you comprehensive information about Java if you want that. This material is created to be companion material to the class at The George Washington University.

Java is a programming language that many professionals use. It is reason enough to learn programming using it. I will keep the following subjects to absolute minimum: historical significance of this language, comprehensiveness, design philosophy. They are fun to talk about. So, if you take my class, we can have few minutes of story time. But we are not going to cover it in this material.

Table of Contents

1 Your first program

Let's take a look at "Hello, World" program in Java ("Lesson: A Closer Look at the 'Hello World!' Application (The Java™ Tutorials > Getting Started)," n.d.). If you are learning a computer language, it is a good way to get an idea what the programming is like.

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

Code 1 HelloWorldApp.java

All Code 1 HelloWorldApp.java above does is print Hello World! on your screen. You place all your program (or you can call it code) in the yellow highlighted area. That's all. Everything else is really extra construct built-up over time. Other languages do away with those extra constructs or more accurately speaking it give you an option to write a program without those constructs. They are loosely called scripting languages.

When you run it, you will see a result like this:

```
Hello World!
```

We will learn how to actually run it on your computer later.

Side story: back in the old days, people did not have a monitor as in LCD or LED monitor. They had a printer as the output device. The command such as print stuck from that era.

1.1 Running your program

First, you would need to install JDK (Java Development Kit) on your computer. It allows you to program in Java.

http://www.oracle.com/technetwork/java/javase/downloads/

In order to run a Java program, you have to compile it. Compilation translates your human readable program into an executable. Another way to say running a program is executing a program.

Windows user: after JDK installation, you need to add it to your path. Press the start button. Type and execute *Edit environment variables for your account*. Add the following path to a variable called PATH:

C:\Program Files\Java\jdk1.8.0_121\bin

If there are more than one path already there, insert a semicolon (;) and then insert the path above.

Mac users: No action is required. JDK is ready to use as soon as you install the JDK.

The following command will produce HelloWorldApp.class, which is the Java executable.

```
javac HelloWorldApp.java
```

Once the executable is produce, you can run it by the following command. Note that you do not specify the extension.

```
java HelloWorldApp
```

You would repeat this when you develop software. It is commonly called compilation cycle. It is a misnomer but it is stuck. It is rather an edit-run cycle.

1.2 Java Language Grammar

Java language is made up of rules, keywords and symbols. They are extra facility to help you program easier. After all, all computer languages are meant to do one thing – manipulate memory with commands according to input. Some parts of memory are wired as output.

A set of words and symbols in Java language grammar are sometimes called lexical elements. The phrase - lexical elements - comes from lexical analysis in the context of computer language processing. A compiler reads characters and recognize a series of them into something meaningful called “token”. Say a number or a command.

1.2.1 Variables

A variable is a place where data is stored. Say we have a number somewhere in our computer’s memory – 14. We should have a name where that number is so we can refer to it later. Let’s call it myInteger. Now we want to change the number to 15. We denote this in the following grammar in the sense that we make the memory (myInteger) be “equated to” 15. The semicolon at the end indicates the end of the line.

```
myInteger = 15;
```

Let’s see another example. As weird as it sounds, let’s have a variable called myGreeting “equated to” Hello. In Java, texts are called strings in the sense that it is a string of characters. They have to be closed in straight double-quotes.

```
myGreeting = "Hello";
```

In reality, this “equating” action is called assignment. The left side is called variable (or variable name) and the right side “value”. Although the equal sign comes from the equating concept, you would sound very weird if you say that.

1.2.2 Data-types

Before you do the assignment above, you have to declare Java know what kind of values you will be assigning. myInteger above has an integer value of 15. The type of values you will assign is called data-type. In this case, it is integer. You would use the following statement to declare this variable:

```
int myInteger;
```

Another way to look at it is that data type is a way to communicate how to lump several bits as one unit.

There are two kinds of data types in Java. The eight primitive data types are byte, short, int, long, float, double, Boolean and char ("Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics)," n.d.).

1.2.3 Literals

Let's re-visit an example above before we go into the definition of literals. I am assigning a value "Hello" into a variable myGreeting.

```
myGreeting = "Hello";
```

We call a String value "Hello" as a String literal.
Side Story: Shouldn't we call "Hello" a text rather than String? It means a string of characters. Imagine a neckless made of beads where the beads are made of letters.

Def. Literals: A *literal* is the source code representation of a fixed value. ("Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics)," n.d.).

Side Story: There is a H word in programming. It is usually negative. It is "hard-coding". Hard-coded programs are not easily configurable. It often requires expensive modification. One of the ways to hard-code things is using lots of literals. On the other hand, completely configurable software is often hard to use. A good example is SGML and Ada. They tried to incorporate all the available features in their respective domain. Try to find a good balance between hard-coding vs. configurable design.

1.2.4 Operators

An operator is a short symbol to execute a function. Let's look at the following code.

```
a = b + c;
```

The plus sign is the operator.

Side Story: We could have written with the syntax of a function usage like this: a = add(b, c). I imagine the creators of Java figured that addition would be a very frequently used function that deserves an extremely short shortcut.

If you want to read and understand Java code, you would have to understand all the operators in ("Operators (The Java™ Tutorials > Learning the Java Language > Language Basics)," n.d.). For these, there is no way around. You could skip some other deep knowledge about Java. But not this one. I assume that you understand most of them after one year of programming as required by this course as a prerequisite.

1.2.5 Statements

A statement is like a line of code except Java's line ends with a semi-colon.

Side Story: Newer languages such Groovy, Swift and Python are shedding this weird line-ending tradition of C family languages. It is there to explicit mark the end. In other word, an infrequently used character (;) makes it easy to program the compiler to recognize the end of a statement that spans multiple lines.

An expression is part of a statement and it gets evaluated into a value. It is made up of variables, operators, and method invocations. ("Expressions, Statements, and Blocks (The Java™ Tutorials > Learning the Java Language > Language Basics)," n.d.)

Side Story: method invocation is very much like function call in a practical sense. Some may argue differently on a language design philosophy basis. In object-oriented languages, "functions" are groups within the object.

A block is a group of statements. It starts with { ("left curly brace") and ends with } ("right curly brace"). It is a way to abstract multiple lines of code into one unit.

1.2.6 Control-Flow

You want to execute different parts of programming depending on the situation. Control-flow statements do exactly that ("Control Flow Statements (The Java™ Tutorials > Learning the Java Language > Language Basics)," n.d.)

1.3 I/O (Input and Output)

I/O makes your program do different things depending on the **input** from somewhere. Once you execute your program. You would want to get the result. That is **output**. For example, your key strokes are input. How your program respond is the output.

Side Story: Under the hood, all of these operations are memory read and write. Your OS (Operating System) abstracts these into consistent function calls regardless of different hardware. Then, languages like Java further wrap these into hierarchical, easier-to-understand API (Application Programming Interface) ("Lesson: Basic I/O (The Java™ Tutorials > Essential Classes)," n.d.).

1.4 Assignments

1.4.1 Problem 1

Read the content of a text file and write the same text except any vowels to another file. Please implement the program using Buffered Stream I/O ("Reading, Writing, and Creating Files (The Java™ Tutorials > Essential Classes > Basic I/O)," n.d.)

1.4.2 Problem 2

Read the content of a text file and display frequency of each word. The output should be to the standard output in the following CSV-like format.

```
are, 14
inflation, 4
is, 23

policy, 13
targeting, 9
the, 53
```

2 References

Control Flow Statements (The Java™ Tutorials > Learning the Java Language > Language Basics). (n.d.). Retrieved January 23, 2017, from https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

Expressions, Statements, and Blocks (The Java™ Tutorials > Learning the Java Language > Language Basics). (n.d.). Retrieved January 23, 2017, from https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html

Lesson: A Closer Look at the "Hello World!" Application (The Java™ Tutorials > Getting Started). (n.d.). Retrieved January 6, 2017, from https://docs.oracle.com/javase/tutorial/getStarted/application/

Lesson: Basic I/O (The Java™ Tutorials > Essential Classes). (n.d.). Retrieved January 23, 2017, from http://docs.oracle.com/javase/tutorial/essential/io/

Operators (The Java™ Tutorials > Learning the Java Language > Language Basics). (n.d.). Retrieved January 23, 2017, from https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics). (n.d.). Retrieved January 6, 2017, from https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Reading, Writing, and Creating Files (The Java™ Tutorials > Essential Classes > Basic I/O). (n.d.). Retrieved January 23, 2017, from http://docs.oracle.com/javase/tutorial/essential/io/file.html