Java 1: Weekly Videos and Curriculum

Site:	CCP	Printed by:	Elliot Hill
Course:	Backend Software Development (2024)	Date:	Friday, November 1, 2024, 4:43 PM
Book:	Java 1: Weekly Videos and Curriculum		

Table of contents

1. Java and Programming

What is Programming

05:36

The definition of Computer Programming in the simplest form is the process of preparing a program for a computer. When presented with a task, or a request to have a computer execute a task, it is the job of a Software Developer or Programmer to write code in a language that is understood by a computer to accomplish that task and to complete the request -- this is programming.   As you progress through this course, you will be given a task or set of tasks each week, and asked to craft or "program" the solution. To do so, you will write a computer program to accomplish the requirements of that task.

Before we start to program a computer and learn how to write code, we need to imagine what it takes to teach a computer to do a task? What exactly is programming?

One way to describe programming is the following: *Programming is writing instructions to tell a computer how to move, manipulate and display data*.   Everything that we see on a website or in an application is a visual representation of some type of data. The data is what drives the program or the experience. Programming is all about data.

Let's take a look at this example. Imagine a bank account. A bank account may contain money. There will be a number of interactions that a person could have with their bank account. If you were teaching a person about a bank account, you might instruct them how to do the following:

- Retrieve the balance in the account.
- Deposit money into the account.
- Withdraw money from the account.

It seems simple enough, right? The idea is to break the problem down into smaller pieces, so that we can instruct a person how to do each task.

What if you are trying to get a computer to execute these requests? The same is true with writing a program to accomplish these tasks. The goal is to break down the problem into smaller pieces to instruct the computer how to do each task.

- The computer would need to have a place to store & manipulate its **data** (**variables**).
- The computer would need to know **what** we want it to do (**tasks, methods**),
- The computer would have to be taught **how** to accomplish those tasks (**operations**).

Stay tuned, we will learn more about all of those in the future.

Here is the example that is used in this video -- notice the following sentence:

> **"The software developer could solve complex challenges, because she had learned how to break problems down into the smallest possible details."**
>
> **Question**: How many words are there in this sentence?
>
> **Explanation**: For a person, it seems very simple. How do we know when one word stops and another begins?
>
> **Programming Task**: Write a program to get a computer to count the words in sentence above.
> Think about what you could use to teach the computer to do something that seems simple to us.

Breaking down problems into the most specific possible details is how a computer is going to read our commands, and that's how we need to start thinking as a developer.

Review: Programming is writing instructions for a computer that tell it how to move, manipulate, and display data in an automated fashion.

Digging a bit deeper -- Interesting Topics related to Programming:

1. ***Top-Down*** *vs.* ***Bottom-Up*** *Programming and Design -- look this up in an internet search to see different approaches to programming.*
2. ***Steps to Programming****:*

   - *Decide how to solve a task or request -- remember to break the task into smaller pieces to be solved*
   - *Write the code to solve the task by first writing code to solve each smaller piece of the solution*
   - *Compile and/or Run the program that has been written.*
   - *Test and Debug the program*
     - *Test: Make sure the code works as designed.*
     - *Debug: Track Down and Fix any flaws in the code that you wrote.*
   - *Document exactly what has been coded and how it works.*

Java

05:07

Overview of Java:

Java is the computer programming language that we are going to be using throughout this course.

Java was created at Sun Microsystems, Inc. with James Gosling as the lead architect. Java was first released in 1995, and brought something new to the programming world. The Java implementation minimizes dependencies, and is platform independent and portable. The Java compiler (*javac*) compiles Java source code, and converts that code into *bytecode*, which is executed by the **JVM** (Java Virtual Machine). The **JVM** transforms the *bytecode* into machine-readable code.

The two-step compilation process is how platform independence leading to portability is accomplished -- one of Java's greatest features.

- **platform independence**: A **Java** program that is compiled on one machine can be executed on any other machine, regardless of the Operating System, as long as the target machine has a **JVM** installed.
- **portability**: a program (set of code) will run identically on different platforms, with no changes such as recompilation or rewriting source code.

Common Java Acronyms:

- **JDK** -- the development platform -- **Java Development Kit** -- what you will install on your system.
- **JVM** -- for runtime execution -- **Java Virtual Machine** -- provides a place for Java byte code to be executed.
- **JRE** -- a software environment, which includes a set of tools used for Java application development, also referred to as the implementation of the JVM -- **Java Runtime Environment**

In 2009, when Oracle Corporation bought Sun Microsystems, Inc., Java became an Oracle Product. You will find all things Java on the Oracle website. The Java Tutorials, JDK 17 downloads, and all of Java language documentation (e.g. search for "javadocs java 17 String" to find the documentation on String object in Java 17)

Java is *older*, but it is still in high demand based on its wide usage and its innovative features. In 2023, Java was listed as the third most popular programming language after (1) *Python* and (2) *C*. **Reference**: *TIOBE Index, Retrieved March 16, 2023*. As a result of Java being widely used, Java only adds features that are tested and that fit well into the Java architecture. Java carefully adds features based on success. Java is still widely used in the industry, which is fantastic for job security and job opportunity.

Java Environment Set-up:

Installation of Java:

We will be installing the current **LTS** of Java. Java has a number of **Long Term Support** versions of the language. The videos here speak of installing Java 8 (v1.8). Since Java 8 was released, there have only been two additional LTS releases of Java, including Java 11, and Java 17. At this point, you are welcome to install any of these LTS releases.

In this course, we will be using the **JDK**, which is the Java Development Kit. It is important to download the **JDK** of the Java Version that you choose.

Download and install the Java JDK version of your choice, keeping in mind the LTS versions.

Installation of Eclipse:

We will also be installing an **Integrated Development Environment** or **IDE**. The IDE that we use in this course is **Eclipse (for Java Developers)**. An **IDE** is a sophisticated text editor that has built-in knowledge of one or more programming languages. This is the location where we will be writing our code, in particular our Java code.

After you download and install **Eclipse (for Java Developers)** following the Promineo Tech installation instructions, open the application. The first time that you open Eclipse, you will see a "Welcome" page. Feel free to read the information that is located there. For our purposes, we are going to exit out of that Welcome page, and you should see the standard Eclipse layout.

Java Project Set-up:

Create a New Java Project:

We will start by creating a new **Java Project**. Got to the top list of options, click on "File", "New", and "Java Project". A "New Java Project" window will pop up, and you will give this new **Java Project** a name: *MyFirstProject* and it should show up in the **Package Explorer** box on the left side of your IDE Window. If you have more than one version of Java JDK installed, you can choose the version that you want when creating a new **Java Project**.

Once *MyFirstProject* appears in the **Package Explorer**, click on the greater-than (>) symbol to the left of the name of the project. That will open up *MyFirstProject* directory structure.

You will see that there is a JRE System Library listed that corresponds to the Java Version that you are using, AND an *src* folder. The *src* folder is the source folder, and that is where our code will be placed. All of our code is going to go into files called **classes**, which we will discuss more fully later.

Create a New Java Package:

Another industry standard way to organize any projects that you write is to create **packages**. A **package** is simply a folder in which we create our **classes**. To create a **package**, right click on the *src* folder, and select "New" and "Package". When the "New Java Package" window pops up, give the new **Java Package** a name. A typical way to create a package naming convention is to use your company website name in reverse. For example, If your company website is *promineotech.com*, then the default package might be "com.promineotech". Once you click "Finish", you will notice a empty package name *com.promineotech* will appear in the *src* folder.

Create a New Java Class:

For now, right click on the *com.promineotech* package, and select "New" and "Class". When the "New Java Class" window pops up, give the new **Java Class** a name. For this example, use *Application*, and also click the option to create a method stub called ***main***. Put a check mark next to the **public static void main(String[] args)**, then click "Finish".

This will create the *Application.java* class, in your *com.promineotech* package, which in turn is in your *src* folder. *Application.java* will have a *main* method, which is the entry point into a Java program. All of the Java programs and applications that we are going to write will have a main method, and our Java code will be put inside the curly braces of the **main** method.

You can now write code inside your **main** method. And right click on *Application.java,* and "Run As" a Java Application. The output from your main method should appear in a **Console** window within your IDE.

Try adding a line of code to your **main** method, maybe something like this:

```
System.out.println("My first Java Program!");
```

If you have followed the instructions above, and used the names suggested here, your code should look something like this:

```
package com.promineotech;

public class Application() {
    public static void main(String[] args) {
        System.out.println("My first Java Program!");
    }

}
```

Go ahead and run the program as a Java Application, and check out the Console!

2. Command Line Interface

CLI

Comparison: Command Line Interface (CLI) and Graphical User Interface (GUI):

If we have had experience working with a computer, odds are that we have used a **Graphical User Interface**, or **GUI**. A **GUI** is the visual representation of data that users interact with via mouse and keyboard. If we open a window or an application and there are buttons and text boxes, that is a **GUI**. **Graphical User Interfaces** or **GUI**s are great for displaying data in an extremely user-friendly manner, however, they are not always the quickest, most effective way to work with computers.

Command Line Interfaces or **CLI**s are text-based tools that allow us to interact with a computer and data. Rather than visual elements, everything on a **CLI** is represented via plain text. Instead of clicking on different buttons to perform some sort of action, users type in text commands. This removes the need for a mouse and reduces the time spent to perform certain operations.

Why do we care? Many programs used in software development only come as a **Command Line Interface** or **CLI** tool and have no **Graphical User Interface** or **GUI** version. There are many different **CLI**s, but the main two are **Command Prompt** (Windows) and **Terminal** (Mac and Linux).

Note: We will discuss some basic commands to navigate the file system via **CLI**, but before we do, it is important to realize that what we see in the **CLI** is the **same thing** we see in the **GUI**. If we are in a **folder** (also known as a **directory**) in our **GUI File Explorer** and in our **Command Prompt** (Windows), or our **GUI Finder** and in our **Terminal** (Mac), we will see the same files in each, the **GUI** will display the files graphically, and the **CLI** will display the files textually.

How to open a Command Prompt (Windows) **or a Terminal** (Mac):

Windows: To open our **CLI** on Windows, press the `windows` key and type `cmd`. Select the **Command Prompt**.

Mac: To open our **CLI** on Mac, press `command` + `space` and then type `terminal` into the search bar. Select the **Terminal**.

Common Commands To Navigate the CLI:
Note: there may be different commands depending on whether you are using **Command Prompt** (Windows) or **Terminal** (Mac).

- `dir` (Windows) or `ls` (Mac) -- show what is in the current directory. To use this command, type `dir` or `ls` and hit enter. We will see a list of everything in the current directory.
- `cd` - change directory (Windows & Mac). There are a few ways we can use this command.
  - (Mac) If we type `cd` and hit enter, we will navigate to our home directory.
  - (Windows) if we type `cd` and hit enter, your current drive and directory path will be displayed
  - If we would like to go back one directory, type `cd ..` and hit enter. For example, if we are in Documents/Projects and want to go back to just Documents, `cd ..` would do that.
  - If we want to go back multiple directories by adding another set of two dots separated by a slash. For example, if we are in Documents/Projects/MyProject and we want to go back to Documents, we can type `cd ../..` to go back 2 directories.
  - If we are in Documents and there is a directory inside Documents named Projects that we want to navigate to, we can type `cd /Projects`.
  - We can also navigate directly to any directory by typing in `cd` followed by the complete directory and hit enter. This enables us to jump to any directory at any time given that we know the complete path.
- `mkdir` - create a new directory (Windows & Mac). To use this command, we type in `mkdir` followed by the name of the directory we want to create. For example, `mkdir projects` would create a directory named `projects` in whichever directory we run the command.
- `copy` (Windows) or `cp` (Mac) - creates a copy of a file or directory, or copies a file or directory from one directory to another.
  - If we are in a directory that contains a file named test.txt and we want to make a copy of this file and call it testcopy.txt, we can run `copy test.txt testcopy.txt` or `cp test.txt testcopy.txt`.
  - If we want to copy the file to another directory named Documents/Projects, we could run `copy test.txt Documents/Projects` or `cp test.txt Documents/Projects`.
- `move` (Windows) or `mv` (Mac) - moves a file or directory to another location. This works similarly to `copy` or `cp` except that it will physically move a file instead of just copying a file. If we want to move a file named test.txt to a directory called Documents/Projects, we would run `move test.txt Documents/Projects` or `mv test.txt Documents/Projects`
- `cd` (Windows) or `pwd` (Mac) -- displays the path of your current directory

3. Variables, Data Types, and Operations

Variables and Data Types

Since programming is defined as moving, manipulating, and displaying data, we need a way to know what data we are working with. We need a way to **assign names to data**. To do this, we use something called **variables**.

Imagine that we have a piece of data: `25.74` How do we know what that data is, or what it represents? It could represent the *price for an item*, the *balance of a bank account*, *the distance from one place to another,* or it could represent *any number of things*. **Variables** allow us to identify what specific data is so that we can refer to the data by its variable name and write instructions telling the computer what to do with it.

Variables can refer to different types of data. Different types of data are typically used in different ways. For example, alphanumeric/textual data is typically used to label something (think anything with text), while numeric data deals more with values and math. **Variables** are placeholders to hold our data.

Java is a statically typed programming language, which means that any variable must first be declared before it can be used. To **declare a variable**, you must determine the variable's **data type** and **name**.

Example of **Java** variable declaration in the format of:

```
datatype variableName = initialValue;

int age = 29;

double bookCost = 14.99;
```

There are eight primitive data types in Java:

1. **int** - an integer
2. **short** - like an **int**, smaller number range
3. **long** - like an **int**, numbers can be higher
4. **boolean** - *true* or *false*
5. **char** - a single character - 'a', 'B', 'A', '&'
6. **byte** - 8 bits of data
7. **double** - decimal places 29.34
8. **float** - precision decimal data type

Additionally, there are predefined Object data types. Here is an example of one of those:

- **String** - textual data - "This is a string"

Resource:

- The Java Tutorials -- Primitive Data Types

Operations

07:16

Knowing that there are different types of data is great, but what is data good for if we don't use it in some way? For example, point of sales systems (the software used when we purchase something from a store, restaurant, etc) have to add up line items and then apply a tax to them. That means we have to perform actions on data (addition and multiplication in this case). In programming, these actions are called **operations**.

An **operation** consists of one or more pieces of data, known as **operands**, and an **operator**, and performs a calculation or action on the operands thus resulting in a new value. One operator we are already familiar with is the **assignment operator** (the equals sign or =), which assigns the data on the right-hand side to the **variable** name/identifier on it's left. **The data on the right and the variable on the left are**

the operands in this operation.

Remember our example from the first video on Programming?

Here are some variables that might be useful in a Savings Account Program**:**

```
double savingsAccountBalance = 29.00;
double depositAmount = 100.00;
double withdrawalAmount = 50.00;
double newSavingsAccountBalance = 0.00;
```

While **operations** take one or more **operands**, most **operations** take exactly **two operands**. For example, our **arithmetic operations** (addition, subtraction, multiplication, and division) take two numeric operands and perform their related math operation on them.

To see how some of these work, we can use the `System.out.println();` method, which prints values to the console when executed.

Again from our Savings Bank Account Example:

Balance Inquiry:

```
System.out.println("The balance in your account is: $" + savingsAccountBalance);
```

Now Deposit A Check:

```
newSavingsAccountBalance = savingsAccountBalance + depositAmount;
System.out.println("You have deposited $" + depositAmount);
System.out.println("Your new balance is $" + newSavingsAccountBalance);
```

Now Withdraw Cash:

```
newSavingsAccountBalance = newSavingsAccountBalance - withdrawalAmount;
System.out.println("Here is your withdrawal of $" + withdrawalAmount);
System.out.println("Your new balance is $" + newSavingsAccountBalance);
```

Coding Challenge: Research: Using *String.format* within the above *System.out.println* statements to be able to print out a variable of type double with two decimal places.

For example: The balance in your account is: $29.00