

# **Modern Computer Programming -- An Expressions First Approach with Java**

---

# **Modern Computer Programming -- An Expressions First Approach with Java**

---

---

## Table of Contents

1. Evaluating Simple Expressions .....	1
Read Evaluate Print Loops (REPLs) .....	1
Expressions .....	2
Arithmetic Expressions .....	3

---

## List of Figures

1.1. Using the JShell REPL .....	2
----------------------------------	---

---

# Chapter 1. Evaluating Simple Expressions

Developing computer programs in a programming language such as Java has its parallels in writing a book in a natural language like English. Both natural languages and programming languages have rules of *syntax* that govern how different elements of the language can be joined together. In the case of English, these are the rules for putting words together in a sentence that you may have studied in an English grammar class. For instance, the prior sentence in this paragraph is well-formed (it follows the rules of proper English grammar), but sentence this one not because rules syntax properly follow not.

That last sentence may have taken you a moment to understand, but I expect that you did. This is the principal difference between a natural language and a programming language: the latter doesn't make any sense at all to the computer unless it is well-formed (strictly follows all rules of syntax). Thus, it's important to have a good foundation of understanding for what these rules are in Java before diving in to writing complex programs. The good news is that some of the more simple syntax in Java, namely expressions, are also considerably powerful, especially when used with a REPL (such as the recently release Java REPL JShell). We'll dive in to some basic expressions, what a REPL is, and how to use a REPL in this first chapter.

## Read Evaluate Print Loops (REPLs)

There are many ways to categorize programming languages. For instance, some programming languages are called *object-oriented*, *functional*, or *statically typed*, to name a few, and many of these categories overlap. An important category, as it relates to teaching Java to a beginning programmer, is that of *scripting languages*. This is important because Java has never in the past been considered one<sup>1</sup>, and this has led to problems when introducing it to someone who has never programmed before.

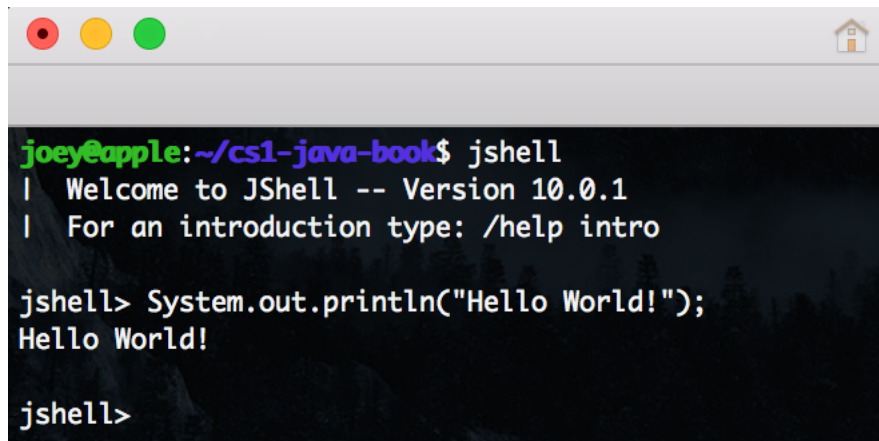
A scripting language is one that is intended to quickly write short, ad-hoc programs that perform a simple task, possibly only once. An important tool for working with a scripting language is a REPL (Read Evaluate Print Loop). A REPL allows you to run small snippets of a program, one at a time, and observe what the computer does in real time. This way the programmer can quickly react, correct errors, and experiment. This functionality is of obvious importance to a new programmer learning how to use the language.

Java, for a very long time, was infamous for its *wordiness*. That is, it took a lot of typing to say anything meaningful. For example, the first program most introductory programming books show the reader is *Hello World!* -- a program that simply prints these words to the screen. In Java, this required the programmer to write 5 lines of code. This is essentially the opposite of the goal of a scripting language.

This changed somewhat in September 2017 when an official REPL was added to Java, called JShell. In JShell, *Hello World!* is just one line:

---

<sup>1</sup>Do not confuse Java with Javascript, which is an entirely different language and is indeed a scripting language.

**Figure 1.1. Using the JShell REPL**

```
joey@apple:~/cs1-java-book$ jshell
| Welcome to JShell -- Version 10.0.1
| For an introduction type: /help intro

jshell> System.out.println("Hello World!");
Hello World!

jshell>
```

JShell is an example of a command-line program run from the system terminal. You can see in the figure that the first thing JShell did after I started it up was to welcome me and indicate what version is running. After that you see the

jshell>

*command prompt.* At that prompt I typed `System.out.println("Hello World!");` which is a snippet of Java that instructs the computer to print the same message back to me. JShell read this command in from me (the R in REPL), then did as it was instructed (the E and P steps). You can see the JShell REPL complying with that command below, and then returning me to the command prompt for another command (L for loop -- after each command, it waits for another, looping indefinitely).

There are many commands you can enter at the command prompt that are not Java code snippets but are unique to using JShell. A very important one is the `/help` command. Typing in this command causes JShell to report on all the other special commands available to you. Another important command is `/exit`, which terminates JShell. We will see other commands as they become relevant in later sections.

## Expressions

*Expressions* are the most basic syntactic building blocks of a computer program. Think of an expression as a simple question that you're asking of the computer. The answer you get back from the computer is called a *value*, and the process of determining the value for an expression is called *evaluation*.

You've worked with expressions before if you've ever used a calculator. For instance, an expression you might feed in to a calculator is `2 + 3`. The answer the calculator would give you back is 5. In this case, the calculator read the expression `2 + 3`, evaluated the expression by performing addition, and returned the resulting value 5 back to you. This is similar to how expressions are used in programming languages, with the difference being that there are many more types of expressions in Java than are likely supported by your calculator!

Literal expressions are the most simple expressions in Java. A *literal expression* is one that directly represents the value it evaluates to. For instance, the expression 5 evaluates to, wait for it..., 5. Other examples of literal expressions are 21, 90000 (note, no commas), or 0.

One catch is that there are more than one type of numeric value in Java! 5 is an example of what is called an integer (shortened to `int` in Java). There are some constraints on the size of an integer (the minimum value is  $-2^{31}$  or  $-2147483648$  and the maximum value is  $2^{31}-1$  or 2147483648). Considering this rule, the expression 2147483649 would be a syntax error, since it is too large to be an `int`.

There are also other integer types with different ranges of values. For instance, the type `long` has a range of  $-2^{63}$  to  $2^{63}$ . This type of value is often used when a number is too large to be represented as

an int. Note that you can specify you want to create a long value using a special syntax for the literal expression. For instance, `46L` is a legal expression that evaluates to the long integer value 46.

`9.2` is an example of a floating point value. These are values with a decimal component. Like integer values, there is more than one floating point type in Java (single and double precision). The difference between these types is the precision (essentially the number of significant digits) to to which a value will approximate some real number. By default, the literal expressions you type in that contain a decimal point will be double precision (shortened to double).

Internally, these values are stored in the same way you would use scientific notation. Each floating point value consists of two components: an exponent and a mantissa. The value is determined by multiplying the mantissa by 10 to the power of the exponent.

In addition to writing out floating point literals with a decimal point, there is a format for literal values that uses scientific notation by placing a letter E between the mantisa and the exponent. For instance, `5E3` is the same as `5000.0`.

## Arithmetic Expressions

Expressions can be comined together to form a larger, more complex, compound expression. To combine two expressions, a *binary operator* is placed between them. The operator performs some function over the values that the two expressions evaluate to (these values are called *operands*). For instance, `+` is a binary operator representing the addition operation. `5` is a numeric literal expression, and `5 + 5` is a more complex, compound expression using the addition operator. For example, `5 + 5` evaluates to the value 10.

There are several binary arithmetic operators which work over numeric expressions in Java. Each is described below:

- `+` (the addition operator)
- `-` (the subtraction operator)
- `*` (the multiplication operator)
- `/` (the division operator)
- `%` (the modulus operator)

Note that, when using the `/` operator with integers, you get back the integer quotient (rather than a floating point value). Modulus gives you the remainder when performing integer division.

There is also a unary operator in Java. This is an operator that works over only one operand rather than two. This is the negation operator, which, like subtraction, is represented with `-`. For instance, `-5` is a compound expression that is the negation of 5.

Also similar to algebra, note that these operators follow the same order of precedence. Multiplication and division are evaluated before addition an subtraction. For example, `2 * 3 + 1` is equal to 7, not 8.

You can also use parentheses to force a certain order of evaluation for your expressions, just as you did in algebra. For example, the expression `2 * (3 + 1)` does, in fact, evaluate to 8.