THE EXPERT'S VOICE® IN JAVA

Beginning Java 8 Fundamentals

Language Syntax, Arrays, Data Types, Objects, and Regular Expressions

COVERS JAVA 8 FEATURES SUCH AS COMPACT PROFILES, AND THE NEW DATE AND TIME APIS

Kishori Sharan
Foreword by John Zukowski, Co-Founding Author of Apress & Java expert

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

How This Book Came About

My first encounter with the Java programming language was during a one-week Java training session in 1997. I did not get a chance to use Java in a project until 1999. I read two Java books and took a Java 2 Programmer certification examination. I did very well on the test, scoring 95 percent. The three questions that I missed on the test made me realize that the books that I had read did not adequately cover details of all the topics necessary about Java. I made up my mind to write a book on the Java programming language. So, I formulated a plan to cover most of the topics that a Java developer needs to use the Java programming language effectively in a project, as well as to get a certification. I initially planned to cover all essential topics in Java in 700 to 800 pages.

As I progressed, I realized that a book covering most of the Java topics in detail could not be written in 700 to 800 hundred pages. One chapter alone that covered data types, operators, and statements spanned 90 pages. I was then faced with the question, "Should I shorten the content of the book or include all the details that I think a Java developer needs?" I opted for including all the details in the book, rather than shortening its content to keep the number of pages low. It has never been my intent to make lots of money from this book. I was never in a hurry to finish this book because that rush could have compromised the quality and the coverage of its contents. In short, I wrote this book to help the Java community understand and use the Java programming language effectively, without having to read many books on the same subject. I wrote this book with the plan that it would be a comprehensive one-stop reference for everyone who wants to learn and grasp the intricacies of the Java programming language.

One of my high school teachers used to tell us that if one wanted to understand a building, one must first understand the bricks, steel, and mortar that make up the building. The same logic applies to most of the things that we want to understand in our lives. It certainly applies to an understanding of the Java programming language. If you want to master the Java programming language, you must start by understanding its basic building blocks. I have used this approach throughout this book, endeavoring to build each topic by describing the basics first. In the book, you will rarely find a topic described without first learning its background. Wherever possible, I have tried to correlate the programming practices with activities in our daily life. Most of the books about the Java programming language available in the market either do not include any pictures at all or have only a few. I believe in the adage, "A picture is worth a thousand words." To a reader, a picture makes a topic easier to understand and remember. I have included plenty of illustrations in the book to aid readers in understanding and visualizing the contents. Developers who have little or no programming experience have difficulty in putting things together to make it a complete program. Keeping them in mind, the book contains over 240 complete Java programs that are ready to be compiled and run.

I spent countless hours doing research for writing this book. My main source of research was the Java Language Specification, white papers and articles on Java topics, and Java Specification Requests (JSRs). I also spent quite a bit of time reading the Java source code to learn more about some of the Java topics. Sometimes, it took a few months researching a topic before I could write the first sentence on the topic. Finally, it was always fun to play with Java programs, sometimes for hours, to add them to the book.

Structure of the Book

This book contains 18 chapters and three appendixes. The chapters contain fundamental topics of Java such as syntax, data types, operators, classes, objects, etc. The chapters are arranged in an order that aids learning the Java programming language faster. The first chapter, "Programming Concepts," explains basic concepts related to programming in general, without going into too much technical details; it introduces Java and its features. The second chapter, "Writing Java Programs," introduces the first program using Java; this chapter is especially written for those learning Java for the first time. Subsequent chapters introduce Java topics in an increasing order of complexity. The new features of Java 8 are included wherever they fit in the chapter. The new Date-Time API, which is one of the biggest addition in Java 8, has been discussed in great detail in over 80 pages in Chapter 12.

After finishing this book, to take your Java knowledge to the next level, two companion books are available by the author: *Beginning Java 8 Language Features* (ISBN 978-1-4302-6658-7) and *Beginning Java 8 APIs, Extensions, and Libraries* (ISBN 978-1-4302-6661-7).

Audience

This book is designed to be useful for anyone who wants to learn the Java programming language. If you are a beginner, with little or no programming background, you need to read from the first chapter to the last, in order. The book contains topics of various degrees of complexity. As a beginner, if you find yourself overwhelmed while reading a section in a chapter, you can skip to the next section or the next chapter, and revisit it later when you gain more experience.

If you are a Java developer with an intermediate or advanced level of experience, you can jump to a chapter or to a section in a chapter directly. If a section uses an unfamiliar topic, you need to visit that topic before continuing the current one.

If you are reading this book to get a certification in the Java programming language, you need to read almost all of the chapters, paying attention to all the detailed descriptions and rules. Most of the certification programs test your fundamental knowledge of the language, not the advanced knowledge. You need to read only those topics that are part of your certification test. Compiling and running over 240 complete Java programs will help you prepare for your certification.

If you are a student who is attending a class in the Java programming language, you need to read the first six chapters of this book thoroughly. These chapters cover the basics of the Java programming languages in detail. You cannot do well in a Java class unless you first master the basics. After covering the basics, you need to read only those chapters that are covered in your class syllabus. I am sure, you, as a Java student, do not need to read the entire book page-by-page.

How to Use This Book

This book is the beginning, not the end, for you to gain the knowledge of the Java programming language. If you are reading this book, it means you are heading in the right direction to learn the Java programming language that will enable you to excel in your academic and professional career. However, there is always a higher goal for you to achieve and you must constantly work harder to achieve it. The following quotations from some great thinkers may help you understand the importance of working hard and constantly looking for knowledge with both your eyes and mind open.

> *The learning and knowledge that we have, is, at the most, but little compared with that of which we are ignorant.*
>
> —Plato

> *True knowledge exists in knowing that you know nothing. And in knowing that you know nothing, that makes you the smartest of all.*
>
> —Socrates

Readers are advised to use the API documentation for the Java programming language, as much as possible, while using this book. The Java API documentation is the place where you will find a complete list of documentation for everything available in the Java class library. You can download (or view) the Java API documentation from the official web site of Oracle Corporation at www.oracle.com. While you read this book, you need to practice writing Java programs yourself. You can also practice by tweaking the programs provided in the book. It does not help much in your learning process if you just read this book and do not practice by writing your own programs. Remember that "practice makes perfect," which is also true in learning how to program in Java.

Source Code and Errata

Source code and errata for this book may be downloaded from www.apress.com/source-code.

Questions and Comments

Please direct all your questions and comments for the author to ksharan@jdojo.com.

CHAPTER 1

Programming Concepts

In this chapter, you will learn

- The general concept of programming
- Different components of programming
- Major programming paradigms
- The object-oriented paradigm and how it is used in Java

What Is Programming?

The term "programming" is used in many contexts. We will discuss its meaning in the context of human-to-computer interaction. In the simplest terms, programming is the way of writing a sequence of instructions to tell a computer to perform a specific task. The sequence of instructions for a computer is known as a program. A set of well-defined notations is used to write a program. The set of notations used to write a program is called a programming language. The person who writes a program is called a programmer. A programmer uses a programming language to write a program.

How does a person tell a computer to perform a task? Can a person tell a computer to perform any task or does a computer have a predefined set of tasks that it can perform? Before we look at human-to-computer communication, let's look at human-to-human communication. How does a human communicate with another human? You would say that human-to-human communication is accomplished using a spoken language, for example, English, German, Hindi, etc. However, spoken language is not the only means of communication between humans. We also communicate using written languages or using gestures without uttering any words. Some people can even communicate sitting miles away from each other without using any words or gestures; they can communicate at thought level.

To have a successful communication, it is not enough just to use a medium of communication like a spoken or written language. The main requirement for a successful communication between two parties is the ability of both parties to understand what is communicated from the other party. For example, suppose there are two people. One person knows how to speak English and the other one knows how to speak German. Can they communicate with each other? The answer is no, because they cannot understand each other's language. What happens if we add an English-German translator between them? We would agree that they would be able to communicate with the help of a translator even though they do not understand each other directly.

Computers understand instructions only in binary format, which is a sequence of 0s and 1s. The sequence of 0s and 1s, which all computers understand, is called machine language or machine code. A computer has a fixed set of basic instructions that it understands. Each computer has its own set of instructions. For example, 0010 may be an instruction to add two numbers on one computer and 0101 is an instruction to add two numbers on another computer. Therefore, programs written in machine language are machine-dependent. Sometimes machine code is referred to as native code as it is native to the machine for which it is written. Programs written in machine language are very difficult, if not impossible, to write, read, understand, and modify. Suppose you want to write a program that

adds two numbers, 15 and 12. The program to add two numbers in machine language will look similar to the one shown below. You do not need to understand the sample code written in this section. It is only for the purpose of discussion and illustration.

```
0010010010   10010100000100110
0001000100   01010010001001010
```

The above instructions are to add two numbers. How difficult will it be to write a program in machine language to perform a complex task? Based on the above code, you may now realize that it is very difficult to write, read, and understand a program written in a machine language. But aren't computers supposed to make our jobs easier, not more difficult? We needed to represent the instructions for computers in some notations that were easier to write, read, and understand, so computer scientists came up with another language called an assembly language. An assembly language provides different notations to write instructions. It is little easier to write, read, and understand than its predecessor, machine language. An assembly language uses mnemonics to represent instructions as opposed to the binary (0s and 1s) used in machine language. A program written in an assembly language to add two numbers looks similar to the following:

```
li $t1, 15
add $t0, $t1, 12
```

If you compare the two programs written in the two different languages to perform the same task, you can see that assembly language is easier to write, read, and understand than machine code. There is one-to-one correspondence between an instruction in machine language and assembly language for a given computer architecture. Recall that a computer understands instructions only in machine language. The instructions that are written in an assembly language must be translated into machine language before the computer can execute them. A program that translates the instructions written in an assembly language into machine language is called an assembler. Figure 1-1 shows the relationship between assembly code, an assembler, and machine code.

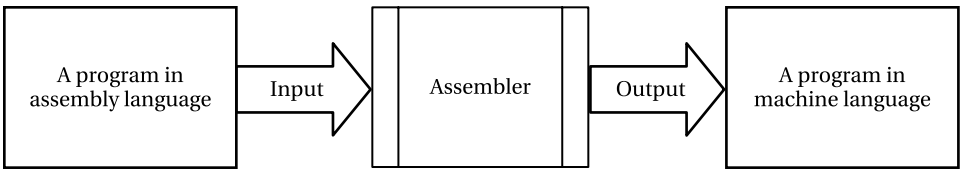

Figure 1-1. *The relationship between assembly code, assembler, and machine code*

Machine language and assembly language are also known as low-level languages. They are called low-level languages because a programmer must understand the low-level details of the computer to write a program using those languages. For example, if you were writing programs in machine and assembly languages, you would need to know what memory location you are writing to or reading from, which register to use to store a specific value, etc. Soon programmers realized a need for a higher-level programming language that could hide the low-level details of computers from them. The need gave rise to the development of high-level programming languages like COBOL, Pascal, FORTRAN, C, C++, Java, C#, etc. The high-level programming languages use English-like words, mathematical notation, and punctuation to write programs. A program written in a high-level programming language is also called source code. They are closer to the written languages that humans are familiar with. The instructions to add two numbers can be written in a high-level programming language, for example. Java looks similar to the following:

```
int x = 15 + 27;
```

You may notice that the programs written in a high-level language are easier and more intuitive to write, read, understand, and modify than the programs written in machine and assembly languages. You might have realized that computers do not understand programs written in high-level languages, as they understand only sequences of 0s and 1s. So there's a need for a way to translate a program written in a high-level language to machine language. The translation is accomplished by a compiler, an interpreter, or a combination of both. A compiler is a program that translates programs written in a high-level programming language into machine language. Compiling a program is an overloaded phrase. Typically, it means translating a program written in a high-level language into machine language. Sometimes it is used to mean translating a program written in a high-level programming language into a lower-level programming language, which is not necessarily the machine language. The code that is generated by a compiler is called compiled code. The compiled program is executed by the computer.

Another way to execute a program written in high-level programming language is to use an interpreter. An interpreter does not translate the whole program into machine language at once. Rather, it reads one instruction written in a high-level programming language at a time, translates it into machine language, and executes it. You can view an interpreter as a simulator. Sometimes a combination of a compiler and an interpreter may be used to compile and run a program written in a high-level language. For example, a program written in Java is compiled into an intermediate language called bytecode. An interpreter, specifically called a Java Virtual Machine (JVM) for the Java platform, is used to interpret the bytecode and execute it. An interpreted program runs slower than a compiled program. Most of the JVMs today use just-in-time compilers (JIT), which compile the entire Java program into machine language as needed. Sometimes another kind of compiler, which is called an ahead-of-time (AOT) compiler, is used to compile a program in an intermediate language (e.g. Java bytecode) to machine language. Figure 1-2 shows the relationship between the source code, a compiler, and the machine code.

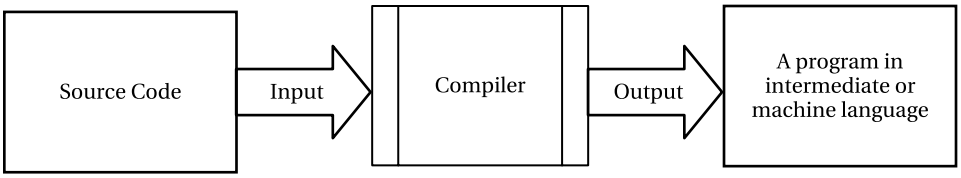

Figure 1-2. *The relationship between source code, a compiler, and machine code*

Components of a Programming Language

A programming language is a system of notations that are used to write instructions for computers. It can be described using three components:

- Syntax
- Semantics
- Pragmatics

The syntax part deals with forming valid programming constructs using available notations. The semantics part deals with the meaning of the programming constructs. The pragmatics part deals with the use of the programming language in practice.

Like a written language (e.g. English), a programming language has a vocabulary and grammar. The vocabulary of a programming language consists of a set of words, symbols, and punctuation marks. The grammar of a programming language defines rules on how to use the vocabulary of the language to form valid programming constructs. You can think of a valid programming construct in a programming language like a sentence in a written language. A sentence in a written language is formed using the vocabulary and grammar of the language. Similarly, a programming construct is formed using the vocabulary and the grammar of the programming language. The vocabulary and the rules to use that vocabulary to form valid programming constructs are known as the syntax of the programming language.