

JB0120: JAVA OOP: A GENTLE INTRODUCTION TO JAVA PROGRAMMING*

R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 3.0[†]

Abstract

This module provides a gentle introduction to Java programming.

1 Table of Contents

- Preface (p. 1)
 - General (p. 1)
 - Prerequisites (p. 2)
 - Viewing tip (p. 2)
 - * Figures (p. 2)
 - * Listings (p. 2)
- Discussion and sample code (p. 2)
 - Introduction (p. 2)
 - Compartments (p. 3)
 - Checkout counter example (p. 3)
 - Sample program (p. 5)
- Run the program (p. 7)
- Miscellaneous (p. 7)

2 Preface

2.1 General

This module is part of a sub-collection of modules designed to help you learn to program computers.
It provides a gentle introduction to Java programming.

*Version 1.1: Nov 16, 2012 7:25 pm +0000

[†]<http://creativecommons.org/licenses/by/3.0/>

2.2 Prerequisites

In addition to an Internet connection and a browser, you will need the following tools (*as a minimum*) to work through the exercises in these modules:

- The Sun/Oracle Java Development Kit (JDK) (See <http://www.oracle.com/technetwork/java/javase/downloads/index>¹)
- Documentation for the Sun/Oracle Java Development Kit (JDK) (See <http://download.oracle.com/javase/7/docs/api/>²)
- A simple IDE or text editor for use in writing Java code.

The minimum prerequisites for understanding the material in these modules include:

- An understanding of algebra.
- An understanding of all of the material covered in the earlier modules in this collection.

2.3 Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

2.3.1 Figures

- Figure 1 (p. 4) . A checkout counter algorithm.

2.3.2 Listings

- Listing 1 (p. 5) . Program named Memory01.
- Listing 2 (p. 6) . Batch file for Memory01.

3 Discussion and sample code

3.1 Introduction

All data is stored in a computer in numeric form. Computer programs do what they do by executing a series of calculations on numeric data. It is the order and the pattern of those calculations that distinguishes one computer program from another.

Avoiding the detailed work

Fortunately, when we program using a high-level programming language such as Java, much of the detailed work is done for us behind the scenes.

Musicians or conductors

As programmers, we are more like conductors than musicians. The various parts of the computer represent the musicians. We tell them what to play, and when to play it, and if we do our job well, we produce a solution to a problem.

¹<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

²<http://download.oracle.com/javase/7/docs/api/>

3.2 Compartments

As the computer program performs its calculations in the correct order, it is often necessary for it to store intermediate results someplace, and then come back and get them to use them in subsequent calculations later. The intermediate results are stored in memory, often referred to as RAM or *Random Access Memory*.

A mechanical analogy

We can think of random access memory as being analogous to a metal rack containing a large number of compartments. The compartments are all the same size and are arranged in a column. Each compartment has a numeric address printed above it. No two compartments have the same numeric address. Each compartment also has a little slot into which you can insert a name or a label for the compartment. No two compartments can have the same name.

Joe, the computer program

Think of yourself as a computer program. You have the ability to write values on little slips of paper and to put them into the compartments. You also have the ability to read the values written on the little slips of paper and to use those values for some purpose. However, there are two rules that you must observe:

- You may not remove a slip of paper from a compartment without replacing it by another slip of paper on which you have written a value.
- You may not put a slip of paper in a compartment without removing the one already there.

3.3 Checkout counter example

In understanding how you might behave as a human computer program, consider yourself to have a job working at the checkout counter of a small grocery store in the 1930s.

You have two tools to work with:

- A mechanical adding machine
- The rack of compartments described above

Initialization

Each morning, the owner of the grocery store tells you to insert a name in the slot above each compartment and to place a little slip of paper with a number written on it inside each compartment. (*In programming jargon, we would refer to this as initialization.*)

Each of the names on the compartments represents a type of grocery such as

- Beans
- Apples
- Pears

No two compartments can have the same name.

No compartment is allowed to have more than one slip of paper inside it.

The price of a can of beans

When you place a new slip of paper in a compartment, you must be careful to remove and destroy the one that was already there.

Each slip of paper that you insert into a compartment contains the price for the type of grocery identified by the label on the compartment.

For example, the slip of paper in the compartment labeled **Beans** may contain the value 15, meaning that each can of beans costs 15 cents.

The checkout procedure

As each customer comes to your checkout counter during the remainder of the day, you execute the following procedure:

- Examine each grocery item to determine its type.
- Read the price stored in the compartment corresponding to that type of grocery.
- Add that price to that customer's bill using your mechanical adding machine.

In programming jargon, we would say that as you process each grocery item for the same customer, you are *looping* . We would also say that you are executing a procedure or an *algorithm* .

When you have processed all of the grocery items for a particular customer, you would

- Press the TOTAL key on the adding machine,
- Turn the crank, and
- Present the customer with the bill.

A schematic representation of the procedure

We might represent the procedure in schematic form as shown in Figure 1 (p. 4) .

A checkout counter algorithm.

```

For each customer, do the following:

  For each item, do the following:
    a. Identify the type of grocery item
    b. Get the price from the compartment
    c. Add the price to accumulated total
  End loop on grocery items

  Present customer with a bill

End loop on a specific customer

```

Figure 1: A checkout counter algorithm.

Common programming activities

This procedure illustrates the three activities commonly believed to be the fundamental activities of any computer program:

- sequence
- selection
- loop

Sequence

A sequence of operations is illustrated by the three items labeled a, b, and c in Figure 1 (p. 4) because they are executed in sequential order.

Selection

The process of identifying the type of grocery item is often referred to as *selection* . A selection operation is the process of selecting among two or more choices.

Loop

The process of repetitively examining each grocery item and processing it is commonly referred to as a *loop* . In the early days of programming, for a programming language named FORTRAN, this was referred to as a *do loop* .

An algorithm

The entire procedure is often referred to as an *algorithm* .

Modifying stored data

Sometimes during the day, the owner of the grocery store may come to you and say that he is going to increase the price of a can of Beans from 15 cents to 25 cents and asks you to take care of the change in price.

You write 25 on a slip of paper and put it in the compartment labeled Beans, being careful to remove and destroy the slip of paper that was previously in that compartment. For the rest of the day, the new price for Beans will be used in your calculations unless the owner asks you to change it again.

Not a bad analogy

This is a pretty good analogy to how random access memory is actually used by a computer program.

Names versus addresses

As a programmer using a high-level language such as Java, you usually don't have to be concerned about the numeric addresses of the compartments.

You are able to think about them and refer to them in terms of their names. (*Names are easier to remember than numeric addresses*). However, deep inside the computer, these names are cross-referenced to addresses and at the lowest level, the program works with memory addresses instead of names.

Execute an algorithm

A computer program always executes some sort of procedure, which is often called an *algorithm* . The algorithm may be very simple as described in the checkout counter example described above, or it may be very complex as would be the case for a spreadsheet program. As the program executes its algorithm, it uses the random access memory to store and retrieve the data that is needed to execute the algorithm.

Why is it called RAM?

The reason this kind of memory is called *RAM* or *random access memory* is that it can be accessed in any order.

Sequential memory

Some types of memory, such as a magnetic tape, must be accessed in sequential order. This means that to get a piece of data (*the price of beans, for example*) from deep inside the memory, it is necessary to start at the beginning and examine every piece of data until the correct one is found.

Combination random/sequential

Other types of memory, such as disks, provide a combination of sequential and random access. For example, the data on a disk is stored in tracks that form concentric circles on the disk. The tracks can be accessed in random order, but the data within a track must be accessed sequentially starting at a specific point on the track.

Sequential memory isn't very good for use by most computer programs because access to each particular piece of data is quite slow.

3.4 Sample program

Listing 1 (p. 5) shows a sample Java program that illustrates the use of memory for the storage and retrieval of data.

Listing 1: Program named Memory01.

```
//File Memory01.java
class Memory01 {
    public static void main(String[] args){
        int beans;
        beans = 25;
        System.out.println(beans);
    }//end main
} //End Memory01 class
```

Listing 2 (p. 6) shows a batch file that you can use to compile and run this program.

Listing 2: Batch file for Memory01.

```
echo off
cls

del *.class

javac -cp .; Memory01.java
java -cp .; Memory01

pause
```

Using the procedure that you learned in the *Getting Started*³ module, you should be able to compile and execute this program. When you do, the program should display 25 on your computer screen.

Variables

You will learn in a future lesson that the term *variable* is synonymous with the term *compartment* that I have used for illustration purposes in this lesson.

The important lines of code

The use of memory is illustrated by the three lines of code in Listing 1 (p. 5) that begin with **int** , **beans** , and **System** . We will ignore the other lines in the program in this module and learn about them in future modules.

Declaring a variable

A memory compartment (*or variable*) is set aside and given the name **beans** by the line that begins with **int** in Listing 1 (p. 5) .

In programmer jargon, this is referred to as *declaring a variable* . The process of declaring a variable

- causes memory to be set aside to contain a value, and
- causes that chunk of memory to be given a name.

That name can be used later to refer to the value stored in that chunk of memory or variable.

This declaration in Listing 1 (p. 5) specifies that any value stored in the variable must be of type **int** . Basically, this means that the value must be an integer. Beyond that, don't worry about what the *type* means at this point. I will explain the concept of type in detail in a future module.

Storing a value in the variable

A value of 25 is stored in the variable named **beans** by the line in Listing 1 (p. 5) that begins with the word **beans** .

In programmer jargon, this is referred to as *assigning a value to a variable* .

³<http://cnx.org/content/m45137/latest/>

From this point forward, when the code in the program refers to this variable by its name, **beans** , the reference to the variable will be interpreted to mean the value stored there.

Retrieving a value from the variable

The line in Listing 1 (p. 5) that begins with the word **System** reads the value stored in the variable named **beans** by referring to the variable by its name.

This line also causes that value to be displayed on your computer screen. However, at this point, you needn't worry about what causes it to be displayed. You will learn those details in a future module. Just remember that the reference to the variable by its name, **beans** , reads the value stored in the variable.

The remaining details

Don't be concerned at this point about the other details in the program. They are there to make it possible for you to compile and execute the program. You will learn about them in future modules.

4 Run the program

I encourage you to run the program that I presented in this lesson to confirm that you get the same results. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

5 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: Housekeeping material

- Module name: JJb0120: Java OOP: A Gentle Introduction to Java Programming
- File: Jb0120.htm
- Published: November 16, 2012

NOTE: Disclaimers: Financial : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-