

Java Developer's Journal

www.JavaDevelopersJournal.com

From COBOL or RPG to Java

by Dante Castiglione

This article, tailored for seasoned COBOL and RPG programmers, provides a jump start into the Java programming language and the concepts behind it; it may also be useful for anyone who is stepping into the Java arena for the first time.

Usually depicted as a language suitable only for those with a strong background in C or C++, in reality Java is a modern, eclectic language that builds on a variety of languages, including C and C++, and also borrows many features from Pascal, BASIC, and, yes, COBOL. In the last few years RPG has been moving closer to COBOL, so close in fact that many people say it's just COBOL written on an RPG specification sheet.

In the end, Java is just another language to learn and master. You did it with COBOL and/or RPG, you can do it again.

The difference is that with COBOL and RPG, the path to mastering them appears quite straightforward. You learn structured programming, some instructions, and then make your first attempt with a compiler. The more knowledgeable you become about a specific compiler, the more valuable you are as a programmer.

On the other hand, when facing Java programming, you get tangled in a jungle of unfamiliar terms and concepts: object-oriented programming, threads, sockets, HTML, browsers. And it looks like there's always moreŠand more.

That's just the beginning. In fact, you also need to learn about IP networks, but don't worry, it's not rocket science.

In the Beginning: The Birth of Java

Maybe you've noticed that many gadgets and appliances in your home or office contain some software; for example, a microwave has a piece of software in it to manage the clock and turn it on and off. This is an entire industry called *embedded software*. One of the problems in this field is that microchips change, and change fast. So software must be rewritten and tailored to work with the new processors.

James Gosling and his team came up with a new idea: make a language that's portable and able to run in as many platforms (please read environments) as needed by simply modifying the compiler for the new chipset.

This was not the first attempt to create a portable language. Do you know which one was the first? COBOL. The acronym, as you probably know, stands for COmmon Business-Oriented Language. The key word here is "Common" as it was truly a standardization effort, aimed at the write-once, run-anywhere concept so dear to Java. The C language was the other important try. The two of them failed in the long run. Compilers diverged, and soon there was no portability at all. If you're a COBOL programmer, you surely know what I'm talking about. RPG has better portability, but it's limited to a few IBM proprietary operating systems, so it doesn't count when looking at the industry as a whole.

The Java idea was good; the timing was not. The project failed. A few years later, the Web was in full fashion and Sun had this product ready.

They thought it was a good idea to download a program from the server, along with a Web page and its graphics (if any), and run the downloaded program in the client machine, no matter which hardware/operating system/ browser combination you were using. It was a good idea, but over time the language became more than that.

Let's Stop Talking and Start Coding

The simplest example may be the classic one, Hello World. There's some good news: in Java, you need to write fewer lines than in COBOL or even RPG to get something done.

```
/*
 * this is a remark
 */

class HelloWorld {

public static void main(String args[ ]) {

System.out.println("Hi, I'm your first Java program. Please call me class.");

} // this is the end of the 'main' method

} // this is the end of the class
```

Here you can see a couple of interesting things. First, a program in Java is called a *class* and routines are known as *methods*. They're almost the same; we'll talk about the differences later. In the code sample, the (one and only) method is called "main". This may remind you of the MAIN SECTION of a COBOL program (mandatory for some compilers). This method has a special "signature" (style of writing) that must be followed exactly if you want your program to be able to run from the command line; that's why you may find the second line a bit strange.

You'll also find that you can use the good old remarks in Java. In the code sample there's a multiline remark at the beginning (in fact, it can have one to *n* lines) that you may find familiar, as it looks like the mainframe or mini control language remarks, and even quite like the COBOL or RPG remarks. The Java toolkit comes with a tool called Javadoc that can be used to extract the remarks into a group of HTML pages, so they can be used as a basis for documenting an application. Look also at the "/" character group; it marks the beginning of an inline remark (like the ones signaling the end of the method and the end of the class).

Compiling

The tools you need for your first attempts at coding and running Java are free and easily downloadable from the official Web site, <http://java.sun.com/j2se>. Install them on your PC, Mac, or workstation. This is the best approach at first; once you feel comfortable with this environment, the next step is to try more sophisticated tools to develop and run an application. First you need to download the Java Development Kit (JDK) for your platform of choice (PC, Mac, Unix, Linux, etc.).

Once you've installed it, follow these instructions for a standard Windows PC (it's more or less the same for the other platforms):

1. Open a command (DOS) window.
2. Create a folder for your code (mkdir folder_name).

3. Type in the example using a text editor (it may be the Window's Notepad, WordPad, or other that you like).
4. Compile the example using the command `javac` (Java compile).

Note: Remember there are some rules for a Java source file:

1. The file must have the `.java` extension, so its name should look like `Class_name.java`.
2. `class_name` is in fact the name of the program, so it must be the same name you typed in the class signature, that is, when you declared the class typing `"class Class_name"`.

Testing

Once you complete the previous steps, you should have two files in your code folder: `Class_name.java` (containing your source code) and `Class_name.class` (holding your compiled code).

To run a Java class (program) use the following command:

```
java Class_name
```

This brings the Java runtime environment to the computer memory, and will in turn load your class (program) and execute it. This is the same thing you do when invoking a COBOL or RPG program from a control language script. After this, your Java class can load other classes and transfer control to them; again, this is quite similar to the concept of a run unit in COBOL or RPG. In Java, this is such a natural thing to do that it doesn't even have a name (i.e., there's no "run unit").

Achieving Portability in Java

When you compile a Java program, the compiler first checks your source code for errors, then it translates the source into "Java bytecodes" or just "bytecodes" machine-like instructions not targeted to any specific platform (operating system and hardware). During runtime, there's an interpreter that reads the bytecodes and causes the execution of instructions suitable for the target processor (microchip). Think of this as the runtime provided with some PC COBOL compilers (like RM-COBOL from Ryan-McFarland or the old Microsoft COBOL). The bytecodes are said to be instructions for a "virtual machine." Think of the virtual machine as a processor that's well defined, but that you won't find at a hardware store; it's implemented in software, on top of existing microprocessors.

You'll find two different packages on the Java Web site: the JDK that I mentioned before, and the Java Runtime Environment (JRE). The latter provides only the runtime, without the programmer's tools, and is useful for deploying applications.

Data and Objects

In COBOL you use data by declaring variables. It's not so different in Java. As a matter of fact, it's better. Java is a modern language, so it builds on the strengths of its predecessors.

And there is something good: you can have basic data types as well as complex ones. Maybe you'll think that this is no news, that you actually use complex data types in COBOL by defining groups of data items in the working storage section. Oh, yes, I'm sure you do it, but here you can have executable code perpetually linked with each complex data type that you define.

This is not a minor thing. It brings up a whole new palette of resources and tricks and ways to do things. In fact, this new possibility is so cool it's called *object-oriented programming*.

An object is just that: a group of fields plus related code, all in a package that can't be (legally) opened. This may not be the best theoretical definition (and maybe it's not even a good one) but I can assure you, from my teaching experience, that it enables you to grasp the concept.

Listing 1 provides a simple program that:

1. Receives parameters from the command line
2. Translates the text-based parameters into numbers
3. Summarizes the numbers
4. Prints the total using the system console

Let's analyze it in more detail. First, look at the `main()` method signature. Inside the parentheses you'll find a data definition, "`String args[]`." This is an array of text parameters received from the command line. It's a mandatory thing for any Java class (program) that wants to be able to run from the command line. You must accept an unspecified number of text parameters from the command line or the class won't be able to run from it. Note that this program needs two parameters to run properly. It won't do any damage to pass more, but if you pass none, or just one, you'll get an error message.

After the method signature you'll find two data definitions. One is a simple, basic variable called a primitive data type in Java. Other object-oriented languages force you to treat absolutely everything as an object; that may be a good thing to do from a theoretical point of view, but in everyday, commercial, production programming practice, it's really a pain in the neck. There's no sense in using an object when all you need to do is count from 1 to 10. (You can use an object if you're a purist, or if you just want to; this is my opinion shared by many people.)

The next one is the definition for using an object. Note that primitive datatype names are in lowercase, and class names must be written with an initial uppercase letter. Java is very case-sensitive, and that's one of the minor things you'll find annoying at first. But remember, COBOL was good because it was very precise and it required a lot of order from the programmer; Java shares the same philosophy, so I guess you'll learn to like it. Using code from other programmers is probably the most useful technique for a professional programmer; in an object-oriented language it's surely the most important technique!

After the definition, the object needs to be initialized. Do so by invoking a special method called a *constructor* you can conceptually compare it to that frequently used COBOL routine for opening the files; there's no exact example for RPG-loving programmers since we know that RPG opens the files automatically. However, if you're an RPG programmer who knows your compiler, you'll remember that there is one thing called *first-cycle processing* implicit in the RPG runtime cycle where files get opened.

So, we created a program and it's running in memory. We defined a simple data item and declared an object in memory. The Java environment went to disk, found the class definition, and instantiated an object into memory using that class as a model. I guess this is enough preparation; now let's do our work.

We transfer control from our object (program) to the other object (program) by invoking a method (subroutine) in it. That's what the line `c = b.parseInt(args[0])` does. It calls a method called `parseInt` (short for parse integer) in object `b`. Object `b` is equal to the class `Integer`, so it has all the methods in this class.

I told you there were some differences between a method and a subroutine. It's time to outline them.

Subroutines are like methods in that you can transfer control to both of them. In COBOL or RPG, you can transfer control to any other subroutine in the same program, but if you transfer control to another program, you can't choose the subroutine. It will execute the whole program from the beginning. In Java, you're free. You can execute any method (subroutine) in your class (program) or any method in any other class (if you've previously loaded it into memory; anyway, there is an exception to this).

The other important difference is that you can't pass data to a subroutine; it will look for it in the program's data definitions. But you can pass data to a method; put the data (or a reference to it) between the parentheses following the method name. And a method can also give back data. Let's look at that strange line again:

```
c = b.parseInt(args[0])
```

```
<--- <-----
```

Output Input

As you can see, the method receives the first occurrence in the array `args[]` as a parameter. That's the easy part. The one thing you'll find a bit weird is that the method invocation is to the right of an assignment expression (the "=" sign). In COBOL or RPG you must use a variable to do this. Write the subroutine so that it puts the output value in the variable, then move that variable to another. In Java, you can do that in one easy step. The output of the method `parseInt` is (as its name implies) an integer. And that integer goes to variable `c`, also an integer.

To end the program we invoke a method called `println` (meaning print line) in a special system object. This object doesn't need to be instantiated and initialized, because it's a *static method*. Conceptually this is not very different from writing to the system console in COBOL or RPG; just the code is different.

This is it: Java is easier than it looks at first glance. You can try it at home or at work on a PC; the same code will run on any platform, so you're leveraging your abilities. Java will program almost anything. Once you can code and hack the simple examples, the next best thing is to concentrate on learning one API (set of classes) that you find interesting, understandable, and profitable. Profitability depends on where you live and who your clients are; some will find that AWT (the windowing API) can get them their first real work in Java. I usually teach the servlet API, because it's kind of easy to find some work to do for online applications.

Resources

- **The Java Tutorial:** <http://java.sun.com/docs/books/tutorial/>
- **New to Java Programming Center:** <http://developer.java.sun.com/developer/onlineTraining/new2java/>
- **The Java technology zone:** www-105.ibm.com/developerworks/education.nsf/dw/java-onlinecourse-bytitle
- **Java sites for mainframe and medium platforms:** www.ibm.com/s390/java, www.ibm.com/as400/java
- **The Java Boutique:** <http://javaboutique.internet.com/>

Author Bio

Dante Castiglione has written more than 30 articles for speciality magazines and newspapers in Latin America, and is the author of *Building Intranets*. He has taught at seminars and conferences in Argentina and participated in the IETF group defining geographic location and privacy standards.

Source Code

Listing 1

```
/*  
Numbers  
An example for CoBOL or RPG programmers  
*/  
class Numbers {  
public static void main(String args[ ]) {  
int c; // a simple integer variable for holding a number  
Integer b; // a complex data type, an Integer with associated code  

```

Reader Feedback

[PERCobol](#)

Posted by [Brian Sullivan](#) on Oct 3 @ 06:07 PM

Anyone who is considering COBOL and Java in the same sentence should take a look at PERCobol, the Cobol compiler for the Java environment. It supports full integration between Cobol and Java including even EJBs in Cobol. The current release 3.10 includes an IDE that even allows debugging both Cobol and Java simultaneously.

<http://www.legacyj.com>

Evaluation download and documentation are available from the website.

(Yes, I do work for the company, so I'll leave off here; just use the website if interested in more information.)

[\(read & respond...\)](#)

[One RPG programmer's POV](#)

Posted by [Doug Smith](#) on Oct 4 @ 02:24 PM

I first heard about Java back in 1995 from a short article in Business Week magazine. Not so odd, considering that most RPG programmers code for the AS/400, and have a considerable amount of business process knowledge. By and large, we are average programmers who produce above average work, given the excellent platform from IBM. In the years that followed I struggled to learn the language, reading dozens of books, coding toy classes, and keeping my day job. My first paid project was doing a web app with JSP. Along the way I passed Sun's Java Pgmr Cert exam, just don't ask me for my score. My peers tried their hands at Java, but most gave up. The problem was mainly emotional - why shoul...

[\(read & respond...\)](#)

[Composite Applications](#)

Posted by [Mike Gilbert](#) on Oct 21 @ 08:53 AM

Putting aside for the moment the business reasons for moving to Java, I would like to comment on the approach advocated here. Having spent considerable years programming in procedural languages (FORTRAN and COBOL), my experience in moving to OO (Smalltalk and Java) is that you MUST tackle the principles of OO before diving into syntax by example. This is fundamental to the structure of the programs you write - and to the nature of the pre-packaged services that you will certainly use. As to the why, my experience in the industry today suggests that many IT shops are taking a practical approach to leveraging the strengths of Java AND COBOL. Both are highly portable - and complementary in ...

[\(read & respond...\)](#)

[not just syntax](#)

Posted by [Lissa Klein](#) on Mar 7 @ 12:18 PM

The problem is much bigger than syntax. The COBOL programmer does not have to understand the systems software for MVS, DB2, IMS, NDM, etc. In the Java world you have to be able to write that code and deploy it. The vocabulary for networks and servers is completely foreign to the general mainframe applications programmers. Unless you are on a large project with lots of support, don't expect to just create your own running modules in Java.

[\(read & respond...\)](#)

All Rights Reserved
Copyright © 2003 SYS-CON Media
E-mail: info@sys-con.com

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. SYS-CON is independent of Sun Microsystems, Inc. SYS-CON, JDJEdge 2002 International Java Developer Conference or Java Developer's Journal is not affiliated with Sun Microsystems, Inc.