JB0160: JAVA OOP: HELLO WORLD*

Richard Baldwin

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

A traditional Hello World in Java provides interesting insights into the structure of a Java application.

1 Table of Contents

- Preface (p. 1)
- Viewing tip (p. 1)
  - Figures (p. 1)
  - Listings (p. 2)
- Introduction (p. 2)
- The Java version of Hello World (p. 3)
- Interesting code fragments (p. 4)
- General information (p. 5)
- Run the program (p. 5)
- Miscellaneous (p. 5)
- Complete program listing (p. 6)

2 Preface

It is traditional in introductory programming courses to write and explain a simple program that prints the text ***"Hello World"*** on the computer screen.

This module continues that tradition.

2.1 Viewing tip

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

2.1.1 Figures

- Figure 1 (p. ??) . How to compile and run a Java application.

*Version 1.1: Nov 17, 2012 5:17 pm -0600

†http://creativecommons.org/licenses/by/3.0/

2.1.2 Listings

- Listing 1 (p. 4) . Beginning of the class named hello1.
- Listing 2 (p. 4) . Beginning of the main method.
- Listing 3 (p. 4) . Display the string Hello World.
- Listing 4 (p. 4) . End of the class named hello1.
- Listing 5 (p. 6) . Complete program listing.

3 Introduction

This module introduces you to Java programming by presenting and discussing a traditional *Hello World* program.

Two approaches

Java programs can be written and executed in several different ways, including the following:

- Stand-alone application from the command line.
- Applet that runs under control of a Java-capable browser.

It is also possible in many cases to write applets, which can be run in a stand-alone mode from the command line, or can be run under control of a Java-capable browser. An example of such an applet will be presented in a future module.

Applets vs. applications

Programming an *"application"* in Java is significantly different from programming an *"applet."* Applets are designed to be downloaded and executed on-line under control of a browser.

Restrictions on applets

Their functionality of an applet is usually restricted in an attempt to prevent downloaded applets from damaging your computer or your data. No such restrictions apply to the functionality of a Java application.

Class definitions

All Java programs consist of one or more **class** definitions. In this course, I will often refer to the *primary* class definition for a Java *application* as the *controlling class* .

The main method

A stand-alone Java application requires a method named **main** in its *controlling class* .

An **Applet** does not require a **main** method. The reason that a Java **Applet** does not require a **main** method will be explained in a future module.

Getting started

Figure 1 (p. **??**) shows the steps for compiling and running a Java application.

1. Download and install the JDK from Oracle. Also consider downloading and installing the documentation, which is a separate download.

2. Using any editor that can produce a plain text file *(such as Notepad)*, create a source code file with the extension on the file name being .java This file contains your actual Java instructions. *(You can copy some sample programs from the early lessons in this collection to get started.)*

3. Open a command-line window and change directory to the directory containing the source file. It doesn't really matter which directory the source file is in, but I normally put my Java files in a directory all their own.

4. Assume that the name of the file is ***joe.java*** , just to have something definitive to refer to.

5. To compile the file, enter the following command at the prompt:

javac joe.java

6. Correct any compiler errors that show up. Once you have corrected all compiler errors, the **javac** program will execute and return immediately to the prompt with no output. At that point, the directory should also contain a file named ***joe.class*** and possibly some other files with a .class extension as well. These are the compiled Java files.

7. To run the program, enter the following command:
java joe
8. If your program produces the correct output, congratulations. You have written, compiled, and executed a Java application. If not, you will need to determine why not.

1. Download and install the JDK from Oracle. Also consider downloading and installing the documentation, which is a separate download.
2. Using any editor that can produce a plain text file *(such as Notepad),* create a source code file with the extension on the file name being .java This file contains your actual Java instructions. *(You can copy some sample programs from the early lessons in this collection to get started.)*
3. Open a command-line window and change directory to the directory containing the source file. It doesn't really matter which directory the source file is in, but I normally put my Java files in a directory all their own.
4. Assume that the name of the file is ***joe.java*** , just to have something definitive to refer to.
5. To compile the file, enter the following command at the prompt:
javac joe.java
6. Correct any compiler errors that show up. Once you have corrected all compiler errors, the **javac** program will execute and return immediately to the prompt with no output. At that point, the directory should also contain a file named ***joe.class*** and possibly some other files with a .class extension as well. These are the compiled Java files.
7. To run the program, enter the following command:
java joe
8. If your program produces the correct output, congratulations. You have written, compiled, and executed a Java application. If not, you will need to determine why not.

Here are the steps for compiling and running a Java application, based on the assumption that you are running under Wi

Figure 1: How to compile and run a Java application.

4 The Java version of Hello World

The class file

Compiled Java programs are stored in *"bytecode"* form in a file with an extension of **class** where the name of the file is the same as the name of the *controlling class (or other class)* in the program.

The main method is static

The **main** method in the controlling class of an application must be *static* , which results in **main** being a *class* method.

Class methods can be called without a requirement to instantiate an object of the class.

When a Java application is started, the *Java Virtual Machine or JVM (an executable file named java.exe)* finds and calls the **main** method in the class whose name matches the name of the class file specified on the command line.

Running an application

For example, to start the JVM and run a Java application named **hello1** , a command such as the following must be executed at the operating system prompt:

java hello1

This command instructs the operating system to start the JVM, and then instructs the JVM to find and execute the java application stored in the file named **hello1.class** . *(Note that the .class extension is not included in the command (p. 4) .)*

This sample program is a Java application named **hello1.java** .

When compiled, it produces a class file named **hello1.class** .

When the program is run, the JVM calls the **main** method defined in the *controlling class* .

The **main** method is a *class* method.

Class methods can be called without a requirement to instantiate an object of the class.

The program displays the following words on the screen:

Hello World

5 Interesting code fragments

I will explain this program code in fragments. A complete listing of the program is provided in Listing 5 (p. 6) .

The code fragment in Listing 1 (p. 4) shows the first line of the class definition for the controlling class named **hello1** . *(I will discuss class definitions in detail in a future module.)*

Listing 1: Beginning of the class named hello1.

```
class hello1 { //define the controlling class
```

The code fragment in Listing 2 (p. 4) begins the definition of the **main** method. I will also discuss method definitions in detail in a future module.

Listing 2: Beginning of the main method.

```
public static void main(String[] args){
```

The fragment in Listing 3 (p. 4) causes the string **Hello World** to be displayed on the command-line screen.

The statement in Listing 3 (p. 4) is an extremely powerful statement from an object-oriented programming viewpoint. When you understand how it works, you will be well on your way to understanding the Java version of Object-Oriented Programming (OOP).

I will discuss this statement in more detail later in a future module.

Listing 3: Display the string Hello World.

```
    System.out.println("Hello World");
```

Listing 4 (p. 4) ends the **main** method and also ends the class definition for the class named **hello1** .

Listing 4: End of the class named hello1.

```
    }//end main
}//End hello1 class
```

The complete program listing

As mentioned earlier, a complete listing of the program is provided in Listing 5 (p. 6) near the end of the module.

6 General information

This program illustrates several general aspects of Java programming.

Overall skeleton of java program

The overall skeleton of any Java program consists of one or more class definitions.

All methods and variables must be defined inside a **class** definition. There can be no freestanding methods or global variables.

File names and extensions

The name of the *controlling class* should be the same as the name of the source file that contains it.

Files containing source code in Java have an extension of *java* .

The main method

The controlling class definition for an application must contain the **main** method.

The primary class file

The file produced by compiling the file containing the controlling class has the same name as the controlling class, and has an extension of **class** .

Many class files may be produced

The java compiler produces a separate file for every class definition contained in an application or applet, even if two or more class definitions are contained in the same source file.

Thus, the compilation of a large application can produce many different *class* files.

What are jar files?

A feature known as a *jar* file can be used to consolidate those class files into a single file for more compact storage, distribution, and transmission. Such a file has an extension of **jar** .

The main method is static

The controlling class for a Java application must contain a ***static*** method named **main** .

When you run the application using the JVM, you specify the name of the *class* file that you want to run.

The JVM then calls the **main** method defined in the *class* file having that name. This is possible because a *class method* can be called without a requirement to instantiate an object of the class.

The **main** method defined in that class definition controls the flow of the program.

7 Run the program

I encourage you to copy the code from Listing 5 (p. 6) . Compile the code and execute it. 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.

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Jb0160: Java OOP: Hello World
- File: Jb0160.htm
- Originally published: 1997
- Published at cnx.org: November 17, 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.

9 Complete program listing

A complete listing of the program discussed in this module is provided in Listing 5 (p. 6) .

Listing 5: Complete program listing.

```
    /*File hello1.java Copyright 1997, R.G.Baldwin
This is a Java application program .

When compiled, this program produces the class named:

hello1.class

When the Java interpreter is called upon the application's
controlling class using the following statement at the
command line:

java hello1

the interpreter starts the program by calling the main
method defined in the controlling class. The main method is
a class method which can be called without the requirement
to instantiate an object of the class.

The program displays the following words on the screen:

Hello World

*******************************************************/
class hello1 { //define the controlling class
  //define main method
  public static void main(String[] args){
    //display text string
    System.out.println("Hello World");
  }//end main
}//End hello1 class.
```

-end-