Massachusetts Institute of Technology

16.410-13 Principles of Autonomy and Decision Making

Jump Starting with Java

Introduction

This jumpstart shows you the basics of getting Java started, running simple programs, and simple editing and debugging. The jumpstart is oriented towards Windows installations, but can be adapted for Unix and other installations. Note that the Java SDK that is installed in the 16.410 computer lab is the Windows version.

Please note that this jumpstart will give you only the most rudimentary skills in Java. To get through the course, you will have to obtain a bit more information. The online Sun Tutorial and the Java in a Nutshell book are excellent resources. The Sun Tutorial can be found at http://java.sun.com/docs/books/tutorial/.

Hello World!

The following has been extracted from the "Hello World! for Microsoft Windows" tutorial (http://java.sun.com/docs/books/tutorial/getStarted/cupojava/index.html).

For additional practice, please try to compile and run the simple Factorial Example Program on page 7 of Java in a Nutshell. There are also many other examples in the book, as well as the online tutorial to help you get familiar with Java.

A Checklist

To write your first program, you'll need:

1. **The Java SE Development Kit 6 (JDK 6)**
2. **A text editor**

These two items are all that you will need in order to write your first application.

Creating Your First Application

Your first application, `HelloWorldApp`, will simply display the greeting "Hello world!" To create this program, you will:

- **Create a source file**

A source file contains code written in the Java programming language, which you and other programmers can understand. You can use any text editor to create and edit source files. Source files end with the extension `.java`.

- **Compile the source file into an executable file**

  The Java programming language *compiler* (`javac`) takes your (`.java`) source file and translates its text into instructions that the Java virtual machine can understand (`.class` files). The instructions contained within this file are known as *bytecodes*. `Javac` is passed the name of the .java file for the top-level class that implements the desired function.

- **Run the program**

  The Java application *launcher tool* (`java`) uses the Java virtual machine to run your application. Java is passed the name of the top-level class that implements the desired function. Its corresponding `.class` file is loaded, and the main method of the corresponding method is invoked. Note that the extension `.class` is not supplied.

Create a Source File

First, start your text editor. In a new document, type in the following code:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Save the code in a file with the name `HelloWorldApp.java`. The file needs to be saved in plan text format; it cannot contain any special formatting directives, such as those saved by default by WYSIWYG editors like Word. A commonly employed command-based editor is Gnu Emacs, a mature open source editor provided by the Free Software Foundation.

Compile the Source File into a .class File

Bring up a command window (known as a shell on Unix). You can do this from the **Start** menu by choosing **Command Prompt** (Windows XP), or by choosing **Run...** and then entering `cmd`. In Windows XP, by default **Command Prompt** can be found at Start> All Programs> Accessories>, while **Run** is found at Start>.

To compile your source file, change your current directory to the directory where your file is located, using `cd <directory-name>`. You can list the contents of the current directory, including subdirectories, using `dir`, You can move up a directory using “`cd ..`“ that is, by supplying double dot as the directory name.

At the prompt, type the following command and press **Enter**.

```
javac HelloWorldApp.java
```

The compiler has generated a bytecode file, `HelloWorldApp.class`. Now that you have a `.class` file, you can run your program, as described below.

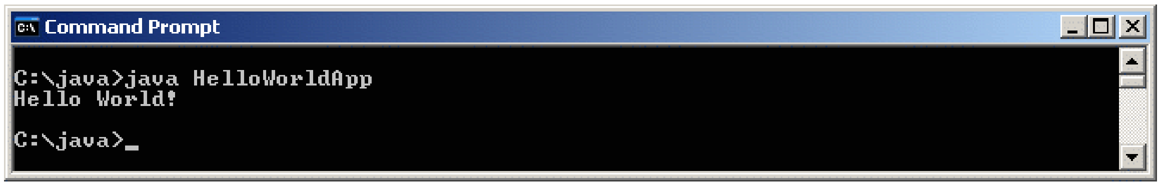
If you encounter problems with the instructions in this step, consult the Common Problems (and Their Solutions) at http://java.sun.com/docs/books/tutorial/getStarted/problems/index.html.

Run your HelloWorldApp Program

In the same directory, enter the following command at the prompt:

```
java HelloWorldApp
```

The next figure shows what you should now see:



```
C:\java>java HelloWorldApp
Hello World!

C:\java>_
```

If you encounter problems with the instructions in this step, consult the Common Problems (and Their Solutions) at http://java.sun.com/docs/books/tutorial/getStarted/cupojava/index.html.

Using the Eclipse SDK

Many people prefer to use an integrated software development kit (SDK). One SDK that you might find useful is Eclipse SDK 3.6.0, which can be downloaded from http://www.eclipse.org/downloads/. Some advantages of using this SDK include: the user interface is very nice, the SDK organizes your files into project folders, and the debugger is more visual and somewhat more user-friendly. There is, of course, an additional learning curve to get started. If you would like to use Eclipse, follow these directions to get started:

1. Download and unzip Eclipse to your Program Files folder; open Eclipse.
2. When asked, enter in the path of the directory in which you want to work or just accept the default folder.
3. You should see a window titled “Welcome” (If you don’t, click on the Help menu, and select Welcome). Click on the icon denoting the Tutorials link (as of 2007 the icon is a page with a check mark).

4. On the tutorials page, click on **Create a Hello World application**. This will bring up Eclipse with a cheat sheets window, typically on the right, that includes a Welcome tab and a set of tutorial steps.
5. Follow all tutorial steps on this Welcome tab.
6. Once you have a HelloWorld program running, try creating an error, for example, by deleting a semicolon. Run it again to see what the error looks like.
7. Then run it with the debug option (Run→Debug As→Java Application).
8. After you terminate the debugger, you can get back to the java perspective by going to Windows→Open Perspective→Java.
9. A more detailed tutorial can be found, by returning to the Welcome Overview page, clicking Java Development, clicking "Basic tutorial" in the resulting window, which in 2007 is entitled Java Development Overview, and following the tutorial directions.

Using JUnit

JUnit is an automated testing program that will be used to grade the problem sets. To install JUnit, go to http://www.junit.org/, and click on Download. Select a location, and unzip to C:\. If you use a different directory make sure it does not contain spaces, such as "Program Files". Refer to README.html in the JUnit director for installation instructions. You will need to set the CLASSPATH environment variable as instructed on the website (on Windows XP this can be found by going to Start> Control Panels>System>Advanced>Environment Variables>System Variables>). Don't confuse CLASSPATH with the PATH environment variable, they are separate variables. Next try to run the sample test programs at the end of this tutorial to see that everything is working. In README.html, click Getting Started, and then Test Infected – Programmers Love Writing Tests. This gives you a brief introduction to JUnit.

Create GetHelloWorldString Class

Create GetHelloWorldString class, in which the `get()` method returns a string "Hello World!".

```
public class GetHelloWorldString {
      public String get() {
            return ("Hello World!");
      }
}
```

Save the file as GetHelloWorldString.java.

Create a JUnit Test for GetHelloWorldString Class

Create a GetHelloWorldStringTest class, in which testGet1() and testGet2() methods tests the get() method of GetHelloWorldString.

```
import org.junit.*;
import static org.junit.Assert.*;

public class GetHelloWorldStringTest {
	@Test
	public void testGet1()
	{
		GetHelloWorldString getHelloWorldString = new
GetHelloWorldString();
		assertEquals(getHelloWorldString.get(),"Hello World!");
	}
	@Test
	public void testGet2()
	{
		GetHelloWorldString getHelloWorldString = new
GetHelloWorldString();
		assertTrue(getHelloWorldString.get() == "Hello World!");
	}
}
```

Save the file as GetHelloWorldStringTest.java.

Compile

Compile the two files by running:

```
javac GetHelloWorldString.java GetHelloWorldStringTest.java
```

The compiler has generated bytecode files GetHelloWorldString.class and GetHelloWorldStringTest.class. Now that you have .class files, you can test your program.

Test

Test by running:

```
> java org.junit.runner.JUnitCore GetHelloWorldStringTest
```

If successful, the output should read:

```
JUnit version 4.4
..
Time: 0.01

OK (2 tests)
```

☺ Have Fun Programming in Java!!! ☺