

## Basics of Java Programming (A Work In Progress)

### What Constitutes a Java Program?

A Java program consists of one or more source files.

Each source file is called <class name>.java, where <class name> is the name of the class contained in that source file and .java is the extension that identifies the file as a Java source code file. In general, each source file contains one class. The name of the class must match the name of the file (without the extension) *exactly*.

To execute a Java program, you first need to compile the source code into byte code. Byte code files have the name <class name>.class. If you're working from the shell, this is done using the javac command (where "c" is for compiler). Usually we will be compiling using the Xcode IDE. Once compiled, the program is executed using the Java interpreter (a.k.a. the Java Virtual Machine or JVM). From the shell, the JVM is invoked with the "java" command. Again we will usually be doing this with the Build and Go or Debug options of the Xcode IDE.

A Java program must have a routine called main(), which is the starting point for program execution. main() will always look something like this:

```
public static void main (String args[]) {
    // insert code here...
    System.out.println("Hello World!");
}
```

### Source Code Formatting

Keep in mind the following when writing Java source code:

- Java is case sensitive ("foo", "Foo", and "fOO" are all considered to be different in Java).
- Every line of code in Java must end in a semi-colon (;).
- Java doesn't care about white space (line breaks, tabs, etc.) but consistent and thoughtful use of white space makes code much easier to read and is part of good programming style.
- You can create comments in Java using // for a single-line comment or /\* \*/ for a block comment. Examples:

```
// This comment is just this one line
/* This comment includes many lines. Code inside is inactive
   ("commented out")
       int x;
       x = 5;
*/
```

## Basics of Variables and Assignment

Like almost every programming language, Java has variables and an assignment statement to put values into a variable. There are two classifications of variables in Java: primitive types and reference variables.

The primitive types we are interested in are int, double, and Boolean, for integer, floating point, and true/false values respectively. An example of declaring and assigning to primitive types:

```
int x;           // declare an integer variable x
x = 3;          // assign the value 3 to x
double y;       // declare a floating point variable y
y = 0.3;        // assign the value 0.3 to y
boolean b;      // declare a boolean variable b
b = (x < y);    // b will be assigned false, because
                // the expression x < y evaluates to false
```

A reference variable either points to an object or is null (null is a keyword for a reference variable that is empty). An example of declaring and using reference variables:

```
Foo f1;
Foo f2;
f1 = new Foo("I'm a foo!"); // create a new Foo and have f1
                             // point to it
f2 = f1;                     // f2 now points to the same Foo as f1
f1 = null;                   // now f1 doesn't point to anything but
                             // f2 still points to the Foo created before
```

## Basics of Classes

Java is an object-oriented programming (OOP) language and the fundamental unit for organizing code is the class. A class has properties (a.k.a. member data) which contain characteristics and data (in general, nouns or adjectives) and methods (code) for actions (verbs). Properties and methods together are called, collectively, members.

Here is an example of a partial class for holding a complex number in rectangular form.

```
public class RectComplexNumber {
    double a;    // the real part of the complex number
    double b;    // the complex (i) part of the complex number

    // constructor—create a new complex number with given values
    public RectComplexNumber(double aInput, double bInput) {
        a = aInput;
        b = bInput;
    }

    // add this complex number to addend and return the result
    public RectComplexNumber add(RectComplexNumber addend) {
        return new RectComplexNumber(a + addend.a, b + addend.b);
    }

    // etc.
}
```

## Using Built-In Classes and Packages

The Java Developer's Kit (JDK) comes with a huge amount of built-in Java code that can be used to build applications. However, for a Java class to use this code it must identify the package that contains that code. This is done at the top of the source file using the import keyword. Example:

```
import java.util.*;      // gain access to all the code in the
                        // java.util package (* means all)
import java.io.File;    // gain access to the File class of
                        // the java.io package
```

## Public vs. Private

Like all OOP languages, Java has mechanisms to support encapsulation. In general, we declare as public all properties and methods we want code outside our package to use. Otherwise the default is that all code in the same package can access the members. If we only want the code in that particular class to access members, we declare them as private.

Good encapsulation style suggests that all properties have non-public or private access. In other words, the "outside world" should only be able to access or modify properties by calling methods. There are, however, exceptions.

## Basic Flow of Control

There are two main flow-of-control constructs: branches (if-then) and loops. The Java syntax for a branch is

```
if (x > 10000) {
    System.out.println("x is big!");
} else if (x > 100) {
    System.out.println("x is medium.");
} else {
    System.out.println("x is small.");
}
```

Note that the code inside the curly brackets for each if or else can be arbitrarily complex and thus might contain additional branches.

Below is a code fragment that illustrates the basic Java syntax for a loop:

```
int i; // loop variable
int total; // accumulator

// this loop adds up the integers from 1 to MAX

for ( i = 1; i++; i <= MAX) {
    total = total + i;
}
```