Transitioning into object-oriented programming using Java

Have you been wanting to learn an object-oriented language? Now's your chance. Get started with this simple guide, and you'll be on the road to OOP.

By Jeff Hanson

Making the transition from procedural programming to object-oriented (O-O) development? Want to break into the exploding world of Java? Don't feel alone. Your situation is shared by thousands of developers. In this series, we will walk you step-by-step through the object-oriented development process using the Java programming language.

What does it mean for a language to be object-oriented? To be considered truly object-oriented, a programming language should support the following features:

- Encapsulation—Hiding an implementation
- Polymorphism—The ability to have the same message sent to different objects and have each different object respond as desired to the message
- Inheritance—The ability to extend existing classes to form specialized classes that inherit state and behavior from the original class
- Dynamic binding—The ability to send messages to objects without having to know their specific type when you write your code

Let’s take a look at how Java supports these features and how it offers additional features to make the transition from procedural programming to object-oriented development a relatively easy experience.

Object-oriented features in Java

Java is an object-oriented programming (OOP) language released by Sun Microsystems in the mid-nineties. The current Java Development Kit (JDK) can be downloaded from Sun. Java is an interpreted language, which means that the source code is compiled into a portable format that will later be interpreted by a virtual machine each time it’s run, and it’s object-oriented from the ground up.

Java hides a lot of the complexities and pitfalls of traditional object-oriented programming languages, such as C++ or Object Pascal, from the programmer. For example, there are no pointers in Java, reference types are automatically cleaned up for the programmer, and variables are automatically initialized to a known default state. With the exception of primitive data types, everything in Java is an object, and object encapsulations are even offered for the primitive types when needed.

Introducing objects

Objects are software programming entities that represent real-life objects, such as bank accounts, computer users, buttons on user interfaces, window menus, etc. Objects are defined by their state and their behavior. For example, a bank account has a state, such as the current balance, the current owner, the minimum balance allowed, and so on, and it has behavior, such as withdraw, deposit, balance, etc.

An object's state is defined by variables known only by the object. Java calls these variables fields or members. Fields are private to the object unless explicitly made available to other classes by keywords defining their scope. We will leave our discussion about scope for later.

An object's behavior is defined by its operations. In Java, these operations are known as methods. Methods can change the state of an object, create new objects, perform utility functions, and so on.

Classes

A class is an entity that defines how an object will behave and what the object will contain when the object is constructed, or instantiated. Classes serve as templates that one or more objects can be created from. Here's the declaration of the ubiquitous HelloWorld application using Java's object-oriented concepts:

```
public class HelloWorld
{
        private String helloMsg = "Hello World!";

        public static void main(String[] args)
        {
            HelloWorld   hw = new HelloWorld();
        }

        public HelloWorld()
        {
            // Display our "Hello World" message
            System.out.println(helloMsg);
       }
}
```

The above example defines a template from which real HelloWorld objects can be created. You will also notice a strange block of code beginning with the line `public static void main(String[] args)`. This block of code is a method used as the main entry point into our HelloWorld program, and this is a typical example of how all Java application entry points are defined. Notice that even this main entry point is encapsulated within a class. In this case, we encapsulate it within the HelloWorld class. The above example demonstrates the definition of one class, HelloWorld, one field, `helloMsg`, and two methods, main and HelloWorld.

The HelloWorld method is a special kind of method known as a constructor. We'll discuss the details and differences of regular methods, constructors, and static methods in a future article.

In Java, all source code for a particular class is declared in a source file that has the same name as the class and has the extension .java. The Java compiler takes your source files and translates them into a platform-independent, binary format, called bytecodes, and stores these bytecodes in classes with the same name as the class definition and an extension of .class. You end up with one class file for each class.

Compiling and running our example

Once you've downloaded the JDK from Sun's Web site and installed it on your machine, you are ready to compile and run Java programs. To compile and run our example, copy and paste the HelloWorld class into your favorite text editor, save the file as `HelloWorld.java`, and then, at a command-line prompt, change the current directory to the directory containing this file. At this point, you can compile the program by entering the following command at the command-line prompt:

Windows:

```
        <Your JDK directory>\bin\javac HelloWorld.java
```

UNIX or Linux:

```
        <Your JDK directory>/bin/javac HelloWorld.java
```

This should create a new file, in the same directory, named `HelloWorld.class`. To run the program, enter the following command at the command-line prompt:

Windows:

```
        <Your JDK directory>\bin\java HelloWorld
```

UNIX or Linux:

```
        <Your JDK directory>/bin/java HelloWorld
```

You should see output on the screen that says Hello World!

Summary

We have touched the surface of object-oriented programming using the Java programming language. Next time, we'll dissect our example, add to it, and discuss more about classes, objects, and some of the other basic concepts of object-oriented programming and their implementation using Java.