The Keyword **this**

Within the body of a (nonstatic) method in Java, the keyword **this** is automatically defined as a reference to the instance upon which the method was invoked. That is, if a caller uses a syntax such as thing.foo(a, b, c), then within the body of method foo for that call, the keyword **this** refers to the object known as thing in the caller's context. There are three common reasons why this reference is needed from within a method body:

1. To store the reference in a variable, or send it as a parameter to another method that expects an instance of that type as an argument.

2. To differentiate between an instance variable and a local variable with the same name. If a local variable is declared in a method having the same name as an instance variable for the class, that name will refer to the local variable within that method body. (We say that the local variable *masks* the instance variable.) In this case, the instance variable can still be accessed by explicitly using the dot notation with **this** as the qualifier. For example, some programmers prefer to use the following style for a constructor, with a parameter having the same name as the underlying variable.

```
public Counter(int count) {
  this.count = count;   // set the instance variable equal to parameter
}
```

3. To allow one constructor body to invoke another constructor body. When one method of a class invokes another method of that same class on the current instance, that is typically done by using the (unqualified) name of the other method. But the syntax for calling a constructor is special. Java allows use of the keyword **this** to be used as a method within the body of one constructor, so as to invoke another constructor with a different signature.

   This is often useful because all of the initialization steps of one constructor can be reused with appropriate parameterization. As a trivial demonstration of the syntax, we could reimplement the zero-argument version of our Counter constructor to have it invoke the one-argument version sending 0 as an explicit parameter. This would be written as follows:

```
public Counter( ) {
  this(0);          // invoke one-parameter constructor with value zero
}
```

   We will provide a more meaningful demonstration of this technique in a later example of a CreditCard class in Section 1.7.