12.1 Introduction

In the last chapter we discussed the problems facing the tester of an object-oriented system (and in particular a Java system). We also briefly considered some approaches that overcome these problems. In this chapter we will consider the current best practice in testing object-oriented systems.

12.2 Class and Instance Sides

As was stated in the last chapter, the basic unit of test in Java is the class. However, there are two sides to a class: one is referred to as the class side and the other the instance side. The class side can be tested directly by sending messages to the class. However, the instance side can only be tested by creating instances of the class. That is, although you define the instance's methods in the class, you must test them using an instance. An important point to remember is that it may be necessary to use both the class and instances of the class together to adequately test both the class side and the instance side. For example, let us assume we have a class such as that illustrated in Listing 12.1.

```
public class Database {
    static String Driver = "sun.jdbc.odbc.JdbcOdbcDriver";
    static void setDriver(String s) {
        Driver = s;
    }
    public void databaseConnect(String url, String userid, String password) {
        Class.forName(Driver);
        Connection con = DriverManager.getConnection(url, userid, password);
        ...
    }
}
```

J. E. Hunt et al., *Key Java*

© Springer-Verlag London Limited 1998
In this situation, it is possible to test the class side method setDriver() directly by sending a message to the class in isolation. However, it is not possible to test the instance method databaseConnect() without considering the range of values which may be returned by Driver. In this case, it is the string "sun.jdbc.odbc.JdbcOdbcDriver" which is returned; however, any subclass could redefine this static variable, thus allowing any one of a range of strings specifying different database drivers to be returned. As the databaseConnect() method may be inherited by other classes, we cannot guarantee which class will receive the getConnection() message.

12.2.1 Testing Methods in a Class

Each method in a class should first be tested in isolation. However, once all the individual methods in the class have been tested, threads through the methods in the class should be identified. Postulating scenarios of normal and exceptional usage (which may have been produced when the class was being designed) can do this. By tracing the results of these scenarios through the class it is possible to identify threads of execution among the methods in the class. An example of a thread of execution is illustrated in Figure 12.1. This thread was obtained by considering a scenario in which a person object has a birthday. This leads to the person object being sent the message birthday.

The associated method birthday is presented below:

```java
public void birthday() {
    System.out.println("A happy birthday to " + name);
    this.incrementAge();
    System.out.println("I am now " + age);
}
```

From the scenario that a person might have a birthday and therefore receive the message birthday, and from examining the source code of birthday (and incrementAge) we might draw the thread of execution presented in Figure 12.1.