Javaのイケてるコード 残念なコード

テストしやすいコードを書く技術

Acroquest Technology
石田 浩司

今回は、「テストしやすいコード」とは何かを具体例で説明していきます。ソフトウエア開発におけるテストとは、プログラムにバグが隠れていないか探したり、プログラムが仕様通りに正しく動作するかどうかを検証する作業です。テストは、あまりクリエイティブな作業とは思えず、苦手意識を持っている人も多いのではないでしょうか。しかしソフトウェアの品質を保つために、テストは欠かせない作業なのです。

それでは、ソフトウエアはどのようにテストするのでしょうか。多くの場合は、テスト実施用のプログラムを作成して、ソフトウエア本体を呼び出してテストします。例えば、テストプログラムから、ソフトウエアに「A」という値を渡して(In)、期待通りに「B」という結果が返ってくるか(Out)を検証するという、InとOutに着目したテストを実施することが多いです。

また、ソフトウエアのテストは、ソフトウエアを構成する部品ごとにテストします。例えば自動車を作るとき、いきなりすべて組み立ててから動作確認することはありません。普通は、パーツごとにそれぞれ動作を確認しているでしょう。ソフトウエアも同様なのです。

それでは、サンプルコードを例にしてテストを実行してみましょう。**リスト1**のようにPostgreSQLのデータベースへ接続して、従業員リストを取得する機能を考えてみます。さらに、リスト1をテストする例として、getEmployeeListメソッドをテストしてみましょう。getEmployeeListメソッドを呼び出す口を作り、実際にgetEmployeeListメソッドを呼び出すだけでテストできそうです。単純に考えると、呼び出すコードは**リスト2**のようになるでしょう。

リスト1●テスト対象のサンプルプログラム

```
public class EmployeeSearch {
  public static void main(String[] args) {
    List<Employee> list =
      new EmployeeSearch().getEmployeeList(args[0]);
    System.out.println(list);
  }

  protected List<Employee> getEmployeeList (String groupId) {
    if (groupId == null || "".equals(groupId)) {
      return new ArrayList<Employee>();
    }

    // データベースにアクセスし、従業員リストを取得
    List<Employee> employeeList = new ArrayList<Employee>();
    ResultSet resultSet = null;
    PreparedStatement statement = null;
    Connection connection = null;

    // JDBCドライバの登録
    String driver = "org.postgresql.Driver";
    String url = "jdbc:postgresql://localhost/sampledb";
    String user = "test";
    String password = "xxxxxxxxxx";
    Class.forName(driver);

    // データベースとの接続
    connection = DriverManager.getConnection(url, user, password);
    String sql = "SELECT * FROM EMPLOYEE WHERE GROUP_ID =
      '" + groupId + "'";
    statement = connection.prepareStatement(sql);
    resultSet = statement.executeQuery(sql);

    (略)

    return employeeList;
  }
}
```

テストするメソッド

リスト2●getEmployeeListメソッドをテストするプログラム

```
public class EmployeeTest {
  public static void main (String[] args) {
    String[] args = { "group1" };
    EmployeeSearch.main(args);
  }
}
```

group1のIDを取得する

データベースの内容を書き出す

小分けにテストする

単純なテストコードを作成できたのですが、リスト2のコードには問題があります。処理の実行結果がSystem.out.printlnに出力されてしまうため、テストするたびに人間の目で結果をチェックしなければなりません。また、getEmployeeListメソッドの中でいくつもの処理があり、しかもテストにはデータベースを準備しなければなりません。これでは、テストの準備にも、実行結果の確認にも時間がかかってしまいます。テストは、いきなりコード全体をテストするのではなく、コードを部品ごとにテストした方が効率が良いのです。

そこで、リスト1のEmployeeSearchクラスをControllerとService、Daoの3つの部品に分けてみましょう(**リスト3**)。クラス分けにより、各メソッドがシンプルになります。クラス分けした後でEmployeeServiceクラスのgetEmployeeListメソッドをテストすることを考えてみます。テスト用に実行するクラスを作成し、対象となるメソッドを呼び出せば良さそうですね(**リスト4**)。

そこで、メインクラスにgetEmployeeListを実行するようなメソッドを用意しました。例として、データベースから取得した件数が10以外なら警告を発するようにしています。

リスト3●いきなり全体をテストするのではなく小分けにテストする

```
// Controllerクラス
public class EmployeeSearch {
  EmployeeService empService = new EmployeeService();
  public static void main(String[] args) {
    List<Employee> list =
      new EmployeeService().getEmployeeList(args[0]);
    System.out.println(list);
  }
}

// Serviceクラス
public class EmployeeService {
  EmployeeDao employeeDao;
  public List<Employee> getEmployeeList(String groupId) {
    if (groupId == null || "".equals(groupId)) {
      return new ArrayList<Employee>();
    }
    // データベースにアクセスし、従業員リストを取得
    List<Employee> employeeList = employeeDao.findByGroupId(groupId);
    return employeeList;
  }

  public void setEmployeeDao(EmployeeDao employeeDao) {
    this.employeeDao = employeeDao;
  }
}
(Daoクラスは省略)
```

モックを作成する

リスト4を記述すれば、確かにgetEmployeeListメソッドをテストできます。しかし、データベースを準備しなければならないという問題は残っています。簡単にテストするためには、テスト時にデータベースにアクセスしなくても良いように変更するべきです。

データベースアクセスの問題を解消するために「モック」という考え方があります。一般的にモックとは「外見が似た模型」という意味ですが、Javaの世界では元のクラスを継承して元のクラスと同じインタフェースを持ちながら、実際には中身の振る舞いが違うクラスのことを指します。今回の例でいえば、EmployeeDaoを継承して実際にはデータベースにアクセスしないようなクラスがモックになります。

リスト5では、元のEmployeeDaoクラスを継承して、findByGroupIdメソッドを上書き(オーバーライドと呼びます)するようなMockEmployeeDaoクラスを作成します。呼び出し側のテストクラスでは、EmployeeDaoの代わりにMo

リスト4●EmployeeServiceクラスのgetEmployeeListメソッドをテストするプログラム

```
public class EmployeeTest {
  public static void main (String[] args) {
    EmployeeService service = new EmployeeService();
    List<Employee> list = service.getEmployeeList("group1");
    if (list.size() == 10) {
      System.out.println("OK!");
    } else {
      throw new IllegalStatementException(
        "期待した通りの結果じゃない!");
    }
  }
}
```

IDがgroup1のカラムを入手する

例えばリストの長さが10でなければエラーを返す

リスト5●モックのプログラム

```
interface EmployeeDao {
  List<Employee> findEmployeeListByGroup(String groupId);
}

public MockEmployeeDao extends EmployeeDao {
  @Override
  public List<Employee> findByGroupId(String groupId) {
    // サンプルとして10件のデータを作成し、返す
    List<Employee> employeeList = new ArrayList<Employee>();
    for (int i = 0; i < 10; i ++) {
      Employee empoloyee = new Employee();
      employee.setName("SampleName");
      employee.setId(123);
      employeeList.add(employee);
    }
    return employeeList;
  }
}
```

データベースの代わりとなるデータを用意する