JUnit によるテストファースト・デザイン

Object Day2001 XP workshop
2001/5/18 Kenji Hiranabe

JUnit を使った，テストファースト・デザイン（テストを先に実装してから，コードを実装する設計スタイル）のやり方を，簡単に説明します．例題は，じゃんけんゲームです．まずは，下の簡単なストーリーでやってみましょう．

- **じゃんけんプレーヤーはじゃんけんゲームに参加できる。**

このストーリーは簡単なので分割の必要はありませんが，もしストーリーが大きいようであれば，複数のタスクに分割して紙に書き出しましょう．

ここでは，このストーリーを Player クラスと Game クラスで実現するとして，簡単なテストを書いてみましょう．そう！　コードより先にテストを書くのです．

GameTest.java

```
import junit.framework.*;

public class GameTest extends TestCase {
    public GameTest(String name) {
        super(name);
    }
    public void testAddPlayer() {
        Game game = new Game();
        Player kenji = new Player("Kenji");
        game.addPlayer(kenji);
        assert(game.hasPlayer(kenji));
    }
}
```

JUnit を import し，TestCase を継承してテストを作成．

このコンストラクタは決まり事として必ず書く．

プレーヤーが参加する，というシンプルなテスト．

この部分が，真になって欲しいことの表明．

テストクラスの名前は，xxxTest が良いでしょう．public クラスとして作ります．テストメソッドは必ず test から始まる名前にし，アクセス制御は public にします．そうすることで，JUnit がテストメソッドを自動認識します．テストの中では assert を使い，真(true)になっていて欲しいことを表明します．

さて，これでコンパイルしてみましょう．IDE によっては自動的にコンパイルされるかもしれません.

```
C:¥> javac *.java
GameTest.java:8: シンボルを解釈処理できません。
シンボル: クラス Game
位置    : GameTest の クラス
        Game game = new Game();
         ^
GameTest.java:9: シンボルを解釈処理できません。
シンボル: クラス Player
位置    : GameTest の クラス
        Player kenji = new Player("Kenji");
         ^
…
```

ここでは，javac を直接 DOS 窓から叩いています.

エラー．これは当然ですね，Game や Player をまだ作成してませんから．まず，このテストをコンパイルできることを目標にして, Game クラスと Player クラスを実装しましょう.

Game.java

```
public class Game {
    public void addPlayer(Player p) {
    }
    public boolean hasPlayer(Player p) {
        return false;
    }
}
```

とりあえず，空.

とりあえず false としておく.

Player.java

```
public class Player {
    public Player(String name) {
    }
}
```

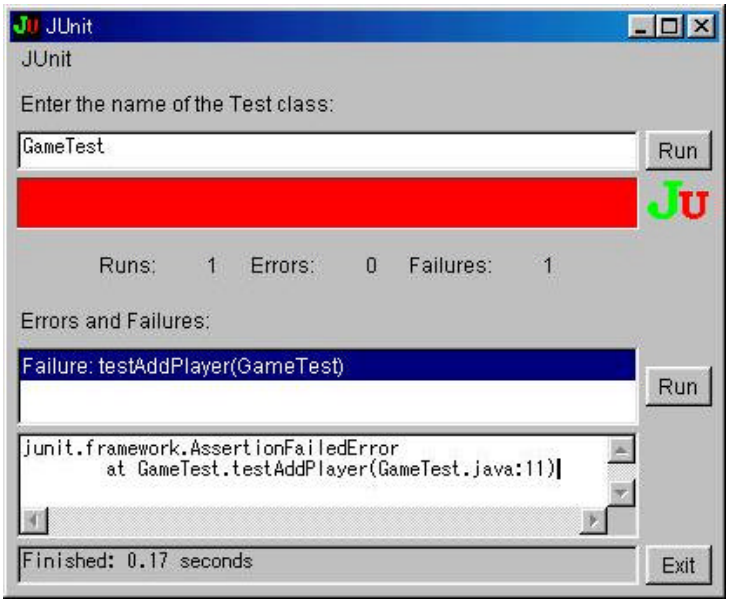
とりあえず，空.

さあ，コンパイルしてみましょう．

```
C:¥> javac *.java
C:¥>
```

OK です．コンパイルが通りました．では早速テスト．JUnit を起動します．

```
C:¥> java junit.ui.TestRunner GameTest
```



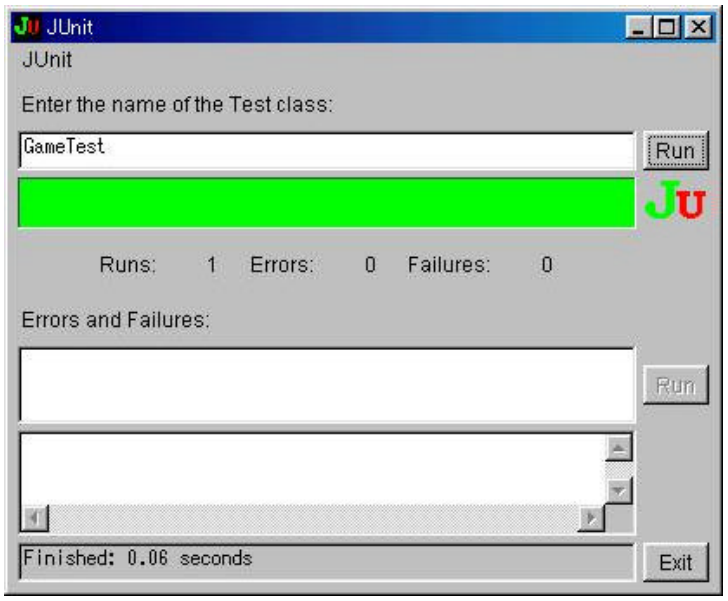
赤のバーが出て，失敗していることを示します．走ったテストは１件，失敗が１件です．GameTest.java の 11 行目で失敗しています．これも当然ですね．Player と Game には実装がありませんから．では，このテストをパスさせる実装を加えましょう．

Game.java

```
import java.util.Vector;
public class Game {
    Vector players = new Vector();
    public void addPlayer(Player p) {
        players.add(p);
```

```
    }
    public boolean hasPlayer(Player p) {
        return players.contains(p);
    }
}
```

Vector を使って，ごくシンプルに実装してみました．再度コンパイルし，JUnit の Run ボタンを押します．．JUnit は自動的に新しくなったクラスをローディングしてくれます．



バーがグリーンになり，テストが通ったことを示します．OK．これでできました．最初に紙に書いたストーリーをペンで消します．

- ~~じゃんけんプレーヤーはじゃんけんゲームに参加できる。~~ **OK!**

これは重要な作業です．これによってささやかな達成感を味わいながら前進しましょう．

では，テストファースト・デザインの復習です．

1. ストーリーが大きければタスクに分割．紙に書く．
2. ストーリーやタスクを表現するテストを書く．
3. テストがコンパイルできるまで実装．

4. テストを走らせる．→　失敗を確認.
5. テストをパスさせるように実装を加える.
6. テストを走らせる．失敗なら５へ戻る.
7. テストが成功したら，最初に書いたストーリーやタスクをペンで消す.

テストをわざと失敗させることも重要です．ちゃんとテストが機能していることを確認する意味があります．さあ，どんどんテストを追加し，実装をしていきましょう.

最後に，テストを書くにあたって幾つか知っておくと便利なことを書いておきます.

- 同値テストには assert(a.equals(b))や assert(a==b)でなく, assertEquals(a,b)を使おう.

assertEquals()は失敗した場合に，より分かりやすいエラーメッセージを出してくれます.
２つの値の型としては，int などの組み込み型はすべて使えますし，Object を継承する型も使えます．その場合は内部で equals()が呼び出されます．この assertEquals()メソッドは，テストクラスが TestCase を継承しているので，メソッドのどこでも使えます.

- AssertEquals の引き数順序は，AssertEquals(期待される値，実際の値).

こう書く習慣をつけましょう．エラーメッセージが分かりやすくなります.

- オブジェクト参照が等しい場合には，assertSame()を使おう.

これも，メッセージが見やすくなります．Object を引数に取る AssertEquals()が equals()を呼び出すのに対して，assertSame()は==を使って参照の同一性をテストします.

- toString()をうまく使おう.

Object を引数に取る assertEquals()が失敗すると，２つのオブジェクトの toString()の値がエラーメッセージに表示されます．エラーの理由が分からなければ，toString()を実装してみましょう.

- null，非 null のテスト.

参照の null や非 null のテストには，assertNull(), assertNotNull()が便利に使えます.

- 到達してはならないコードには，fail()を使おう.

Exception を出すことを確認するテストなどでは，ここに来ちゃまずいんだけど，という場所があります．そこには，fail()と書きましょう．そこに到達するとエラーになります.

では，**Enjoy Testing！**