第 1 回 テスト ゼミ

ゼミ担当者　：　牧野 浩之，雨宮 明日香，橋本 篤
指導院生　：　千野 晋平，山崎 弘貴
開催日　：　2006 年 5 月 19 日

ゼミ内容: 本ゼミでは，プログラミング演習に向けて，テストの概念について部分を学ぶ．またテストで必要となる JUnit について学ぶ．

1　はじめに

本ゼミでは，システム開発におけるテスト工程についての説明を行い，なぜテストを行うのか，またその目的は何であるかを明らかにする．システム開発におけるテストには，単体テスト，結合テスト，統合テスト及び機能テストといったものがあるが，本ゼミでは単体テストに焦点をあてて説明を行う．そして，Eclipse に備わっているテスト実行環境である JUnit の使い方についての説明を行う．

2　テストとは

システム開発では，プログラムを実装するだけではなく記述したソースコードにバグがないかどうかを検討し，システムの品質を確保する必要がある．そこでシステムの品質を保つためにテストという工程を行う．テストの目的は以下の通りである．

- バグを発見する
  一つずつバグを発見し，一つずつ修正することで，システムの品質は確保される．
- 仕事が完了したことを確認する
  システム開発の一つ一つの工程が完了したことを，誰でもわかるような明瞭な形で示す．
- システムをより優れたものに改良する
  テストによりシステムの現在の状態を把握して，改良することができる．

テスト工程の中で一番重要なものが単体テストである．なぜなら，開発の早い段階でバグが見つかり，それ以降の工程をスムーズに運ぶことができるからである．バグの発見以外の単体テストの特徴は以下の通りである．

- ソースコードの満たすべき機能やプログラムの設計仕様を示す
- クラスがいつ完成したか，判断基準を与える

3　JUnit

Eclipse には JUnit というテスト実行環境が備わっている．JUnit は Java プログラム単体テスト用のフレームワークのことで，Java のプログラム単位であるクラス毎に単体テストを行う．そして作成したテストケースとプログラムの実行結果が合っているかをチェックする機能を提供する．JUnit によって共通のテストフレームワークをもつことで，他人のテストプログラムの修正を容易にする．仕様に合致する単体テストを先に書いて、その単体テストに通るようなコードを書いていく．プログラムに対する要求が複雑化しているため，人の手によるテストの精度には 限界がある．ゆえに，テストプログラムを作成し，自動化テストを行うことが望ましいといえる．テスト仕様が作成できれば，プログラムのテストを自動化することができる．また，JUnit では複数のテストケースをまとめて実行することも可能である．更に，テスト仕様があれば，プログラミング完成のゴールとすることができる．そして，テストケースの何パーセントをクリアしたかを視覚的に捉え，完成までの目安にすることができる．JUnit には次のメリットがある．

- テストが簡単に実装できる
  JUnit はテスト用のクラスを継承してテストクラスを作成しているので，最小限のコーディングで単体テストを構築できる．
- テストが統一的に作成できる
  フレームワークという枠組みがあることで，統一された方法でテストを行うことができる．
- テストと実装コードの分離
  JUnit では実装クラス「Sample」に対して，テストクラス「SampleTest」を作成する．このことにより，テストコードで実装コードを分離することができる．

4　JUnit の実行手順

4.1　JUnit のテストクラス作成

Eclipse を使用して，JUnit のテストクラスの作成方法について解説する．以下の手順に従ってテストクラスを作成する．但し，プロジェクト名，クラス名は自由である．

1. Java プロジェクトを作成する．

2. 作成したプロジェクトに新しいクラス（AddNum）を作成する．このとき，「public satic void(String[] args)」と「継承された抽象メソッド」にチェックを入れる．(Fig. 1 を参照)

Fig. 1　クラスの作成

3. パッケージエクスプローラでクラス（AddNum）を右クリックし，「新規」→「JUnit Test Case」を選択する．(Fig. 2 を参照)

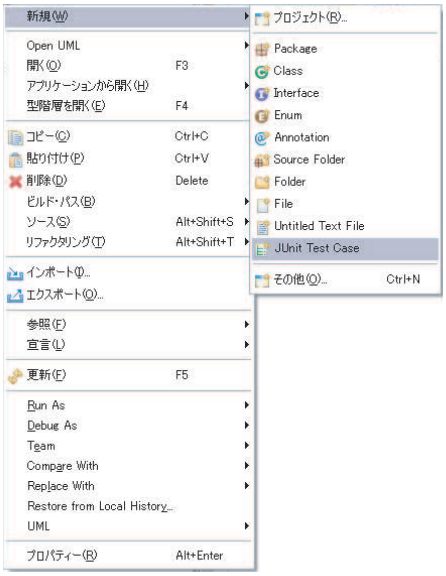

Fig. 2　JUnit Test Case の作成

4. Fig. 3 の新規テストケース画面が出てくるので「はい」をクリックする．

5. Fig. 4 のように JUNIT ライブラリ”junit.jar”へのパスがビルドされる．更に Fig. 5 の新規 JUnit テスト・ケース画面に移るので「終了」をクリックする．

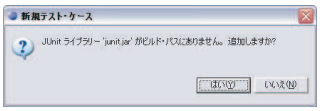

Fig. 3　新規テストケース画面

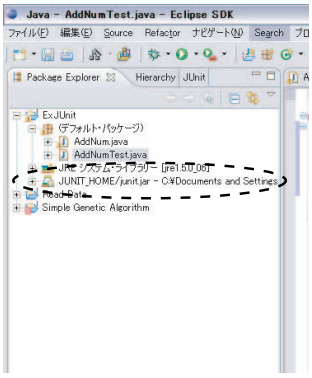

Fig. 4　JUnit.jar へのパス

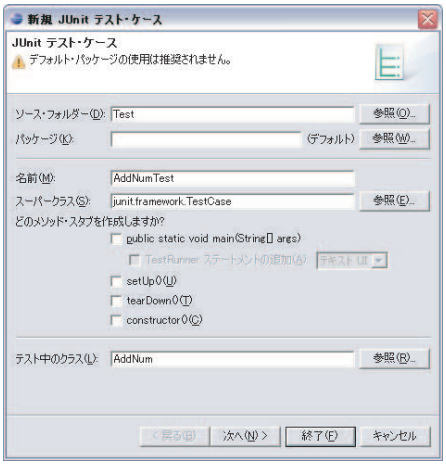

Fig. 5　テストの作成

以上の処理でテストクラスの作成が終了である．

4.2　JUnit の実行方法

Eclipse から JUnit を実行する方法について解説する．前節で作成したクラスとテストクラスを利用して，以下の手順に従って JUnit を実行する．なお，以下にテストクラスと暮らすのソースを記述する．

1. AddNumTest.java

```
import junit.framework.TestCase;

public class AddNumTest extends TestCase {
    public void testAddNum() {
        int tn1 = 17;
        AddNum check = new AddNum();
        int test = check.addNum(8, 9)
        assertEquals("計算ミス", tn1, test);
    }
}
```

2. AddNum.java

```
public class AddNum{
    public int addNum(int a, int b){
        int answer = a + b;
        return answer;
    }
}
```

3. テストプログラムを右クリックし,「実行」→「JUnit テスト」を選択する. (Fig. 6 を参照)

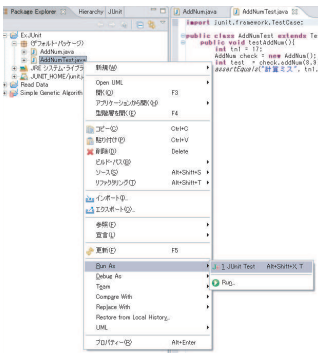

Fig. 6　JUnit の実行

4. JUnit ビューが起動し，JUnit の結果が Fig. 7 のように全体画面の左に表示される.

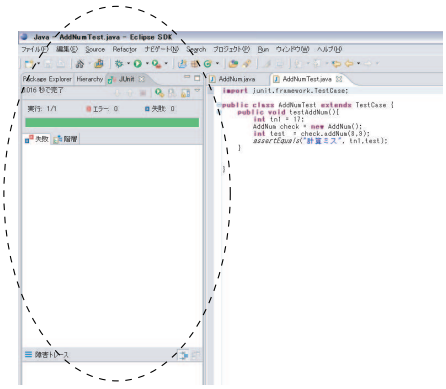

Fig. 7　JUnit の実行結果

5. 緑のバーが表示されればテストの成功である．赤のバーの場合はテストの失敗である．失敗すると Fig. 8 のようにエラーメッセージが表示される.

Fig. 8　JUnit の実行結果（失敗例）

今回作成したプログラムは簡単な足し算を行っているプログラム (AddNum) であり，そのプログラムが正しい結果を返しているか判定するのがテストプログラム (AddNumTest) である．今回の場合は緑のバーが表示され，テストが成功したことが分かる.

4.3　JUnit のテストメソッド

前節のテストプログラム (AddNumTest) は，得たい結果と AddNum.java で生成される結果が一致するかを JUnit の assertEquals メソッドで判断している．JUnit には数個のテスト用メソッドが用意されている．以下の Fig. 9 に JUnit のメソッドを示す.

メソッド	判定	メッセージ
assertEquals(データ型 arg1, データ型 arg2)	expected、arg2の値が同じ値	なし
assertEquals(String message, データ型 arg1, データ型 arg2)	arg1、arg2の値が同じ値	あり
assertTrue(boolean arg1)	arg1がtrue	なし
assertTrue(String message, boolean arg1)	arg1がtrue	あり
assertFalse(boolean arg1)	arg1がfalse	なし
assertFalse(String message, boolean arg1)	arg1がfalse	あり
assertNull(Object arg1)	arg1がnull	なし
assertNull(String message, Object arg1)	arg1がnull	あり
assertNotNull(Object arg1)	arg1がnullではない	なし
assertNotNull(String message, Object arg1)	arg1がnullではない	あり
assertSame(Object arg1, Object arg2)	arg1 == arg2がtrue	なし
assertSame(String message, Object arg1, Object arg2)	arg1 == arg2がtrue	あり
assertNotSame(Object arg1, Object arg2)	arg1 == arg2がfalse	なし
assertNotSame(String message, Object arg1, Object arg2)	arg1 == arg2がfalse	あり
fail()	テスト強制失敗	なし
fail(String message)	テスト強制失敗	あり

Fig. 9　JUnit のメソッド

5 課題

以下に示されているテストクラスのプログラムを作成せよ.

```
import junit.framework.TestCase;

public class CrossoverTest extends TestCase {

  //テストメソッドの宣言
   public void test() {
     String[] geneX = {"11010011", "11001010",
                       "10101101", "11101001",
                       "01001110", "10111011",
                       "00111010", "00011101",
                       "11101101"};

     String[] geneY = {"10101010", "10110101",
                       "01011010", "11010101",
                       "11101101", "00010101",
                       "00010110", "11101010",
                       "00110011"};

     int crosspoint = 4;


     String[] gene = { "11011010", "11000101",
                       "10101010", "11100101",
                       "01001101", "10110101",
                       "00110110", "00011010",
                       "11100011"};


     int[] crosspoint = {3,5}

     String[] gene = {"11001011", "11010010",
                       "10111101", "11110001",
                       "01001110", "10110011",
                       "00110010", "00001101",
                       "11110101"};


    //MakeGene メソッド呼び出し
     for(int i=0; i<9; i++){
       out[i] = Crossover.MakeGene
                  (geneX[i], geneY[i], crosspoint);
      //確認
       assertEquals("誤り", gene[i], out[i]);
     }
   }
}
```

参考文献

1) 単体テストと JUnit,
   http://www.techscore.com/tech/Others/JUnit/1.html
   .

2) @ IT:連載:快適な XP ドライビングのすすめ 第 4 回,
   http://www.atmarkit.co.jp/fjava/devs/xpd04/xpd04.html
   .

3) Java の道 : Eclipse(JUnit の利用),
   http://www.javaroad.jp/opensource/js$_e$*clipse*9.*htm*.