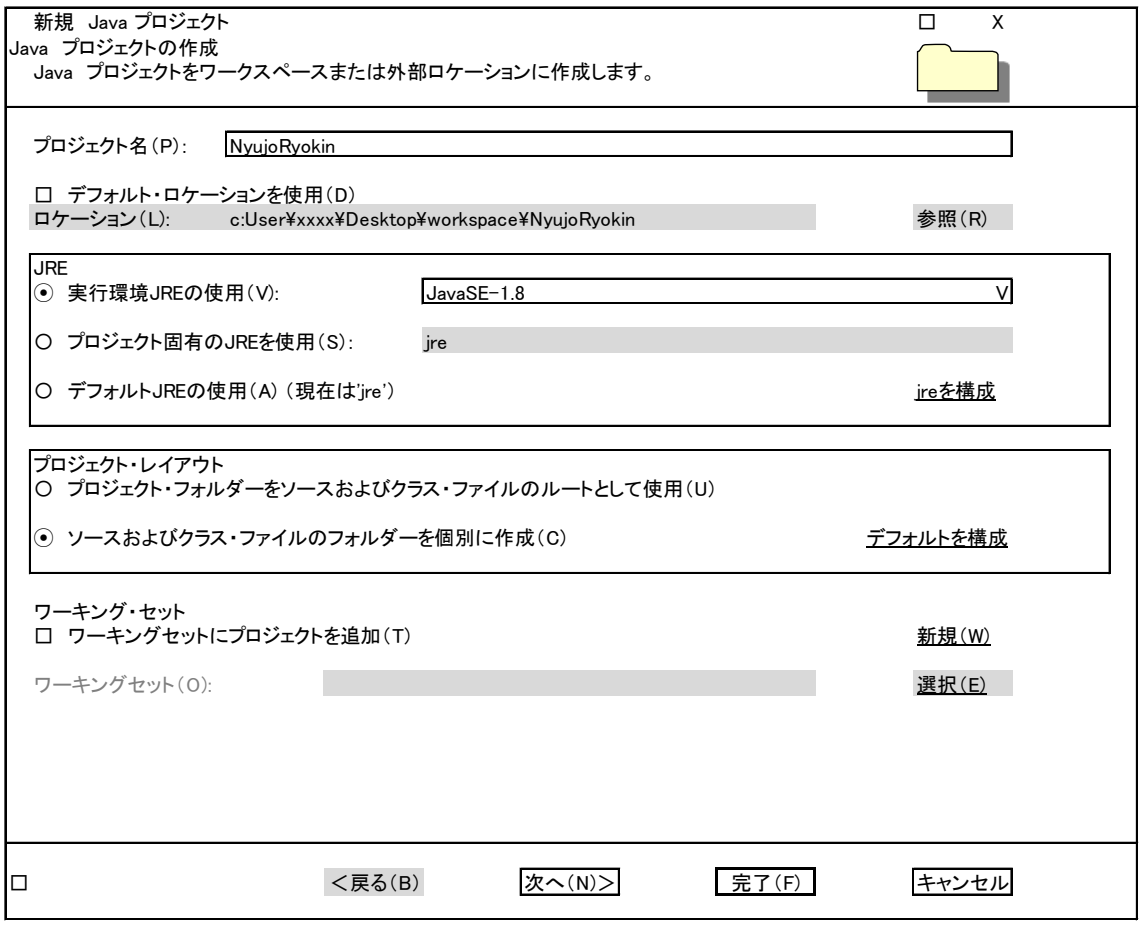
■演習プリント１

JUnit の操作演習

１．テスト用プロジェクト作成

最初にテスト用のプロジェクトを作成します。Eclipse 画面のファイル・メニューから、新規→Java プロジェクトを選択します。そこで、次の画面のようにプロジェクト名を入力して「完了」をクリックします。



そうすると、最初に次のようなコードが生成されます。(実際には、// のコメントはありません)

```
package junit.tutorial;          // チュートリアルを使用した例
public class Ryokin {            // クラスの名前の例
}
```

この原型に、必要なコードを追加してプロダクション用となる「テスト対象クラス」を作り上げます。

コーディングの順序として、先に「テストクラス」をコーディングし、そのテストに合格するように「テスト対象クラス」をコーディングしていく手法があります。この手法を「テスト駆動型開発」と呼びます。

２．テストクラスの作成

「テスト対象クラス」とは別のソース・フォルダーに「テストクラス」を作成します。

- Eclipse から「テスト対象クラス」を開いておいて「ファイル」⇒「新規」⇒「JUnit テストケース」を選択します。
- ダイアログが開かれるので、「パッケージ」にはクラスを格納する場所の名前を入力し、「名前」の箇所に、これから作成する「テストクラス」の名称（ここでは NyujoRyokin）を記入します。
- 「テスト元クラス」に「テスト対象クラス」を指定して「完了」をクリックします。

そうすると、次のようなテストクラスの原型が自動的に作成されます。（コメントの部分は自動生成ではありません）

```
package junit.tutorial;                    // チュートリアルを使用した例
import static org.junit.Assert.*;
import org.junit.Test;
public class NyujoRyokin {  // クラスの名前の例
  @Test                              // Test　アノテーション
  public void test() {
          fail("Not yet implemented");
  }
            }
```

この原型に、必要なコードを追加してテストコードを作り上げます。

このまま「Run（実行)」させると、コードに記述されている”Not yet implemented（まだ実装されていません）”のメッセージが出されてエラーとなり、「赤色バー」の上部に「Runs（実行）:1/1　Error（エラー）:0　 Fail（失敗）:1」が表示されます。この時点では、これは正常です。英語版には（）内の日本語はありません。

３．テストメソッド

次にテスト仕様書に従って「テストクラス」のソフトウェアに「テストメソッド」を定義します。これがテストコードであり、「テスト対象クラス」の振る舞いを検証する機能となります。テストコードは、機能単位であったり条件単位であったりします。例えば、ひとつのクラスにおいて、次のように二つの乗算の結果をテストすることができます。これは非常に簡単な例ですが、境界値テストなどによく使われます。

```
Package ・・・・
import static org.junit.Assert.*;

省略

@Test
Public void 乗算 12 テスト() {
  Calculator calc = new Calculator();
  int expected =12                    // 期待値
  int actual = calc.multiply(3,4);    // 3 x 4
  assertThat(actual, is(expected));   // 計算結果と期待値の比較
}
```

```
省略

@Test
Public void 乗算35テスト() {
   Calculator calc = new Calculator();
   int expected =35                    // 期待値
   int actual = calc.multiply(5,7);    // 5 x 7
   assertThat(actual, is(expected));   // 計算結果と期待値の比較
}
以下省略
```

４．アノテーション

「@Test」は Java で使われるアノテーションのひとつです。アノテーションは Java5 から導入された言語仕様ですが、クラス、メソッド、変数などについて注釈を加える役割を担っています。実際に JUnit で使われる主なアノテーションを表にしました。

アノテーション	意味
@Test	テストメソッドであることを宣言する。二つの属性をもつ expected;例外の検証テストにおいて期待される例外クラスを指定する timeout; 実行時間がかかりすぎる場合に失敗させる
@Ignore	テストの実行から除外する。一時的に停止させる場合に使用する
@Before	テスト実行前に処理する
@After	テスト実行後に処理する
@BeforeClass	テスト実行前に一度だけ処理する
@AfterClass	テスト実行後に一度だけ処理する

さらに、JUnit では値を比較する手法として Assert クラスに assertThat メソッドや Matcher API を活用します。JUnit の以前のバージョンには assertEquals や assertTrue などもありましたが、現在は assertThat と fail メソッドのみで十分です。assertThat は二つの引数をもっています。ひとつ目は実測値で、ふたつ目は期待値です。JUnit では基本的には static インポートを使うことを想定しているのですが、static インポートを使わないで記述すると次のようになります。

```
Assert.assertThat(1 + 2, CoreMatchers.is(3));
```

これは、１＋２＝３であることを検証するためのコードです。
CoreMatchers API を活用して static インポートを使えば、次のように簡単で自然言語に近くなります。

```
assertThat(1 + 2, is(3));
```

fail メソッドは、無条件にテストを失敗させるために使われます。

```
@Test
public void 失敗のテスト()　{
        fail (“失敗しました”);
          }
```

このように、fail メソッドからは直ちに AssertionError （この例では“失敗しました”）が送出されて失敗させます。