JUnitのススメ

統合開発環境eclipse上で行う
Javaのテスティングフレームワーク『JUnit』の利用方法

株式会社エスプランニング　瀬戸田　慎一

本日の流れ

- JUnitの概要
- JUnit導入のメリット・デメリット
- eclipseでの使用方法
- テストケースの書き方
- テストを考慮したクラス設計

JUnitとは

Javaの単体テスト向けフレームワーク

単体テストの目的

- 設計仕様を満たしている事を証明する
- 各処理単体の責任を明確にする
- 不具合を発見する
- システムをより優れたものに改良する

JUnitでのテスト

2. テストケースを作成・実装
3. テスト実施
4. 全テストが成功するまでテスト対象のコードを修正

JUnit導入のメリット

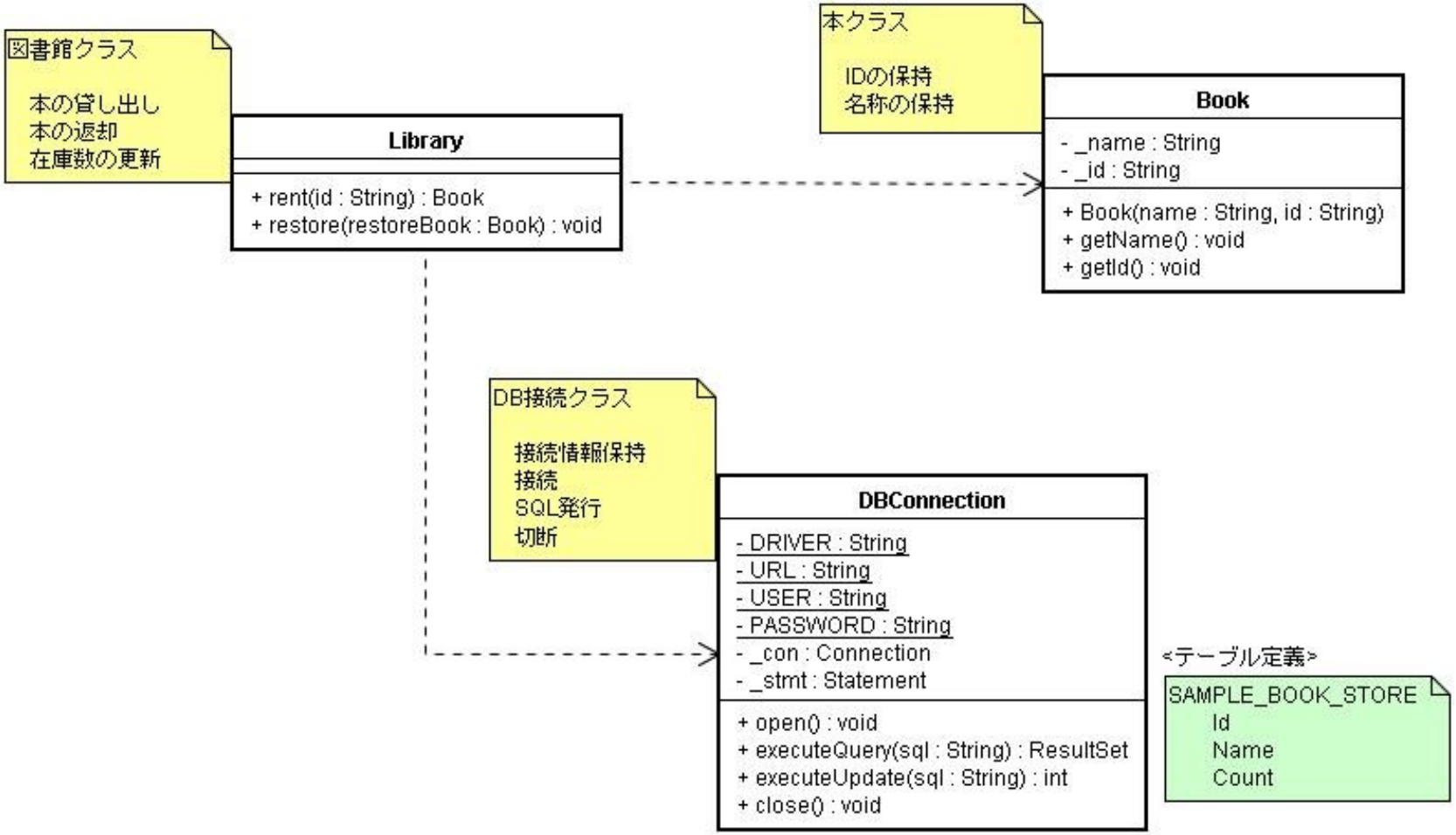
- テストを自動化しテスト工程の効率を上げる
- 曖昧な仕様を明確にする
- リファクタリングが怖くなくなる
- デグレードを防止できる

JUnit導入のデメリット

- ツールの機能や使い方を覚える必要がある
- テストケースを実装する作業が増える
- テストレベルの管理が必要
- 仕様変更にテストケースの変更作業が伴う
- 向かないテストがある

JUnitを使ってみる

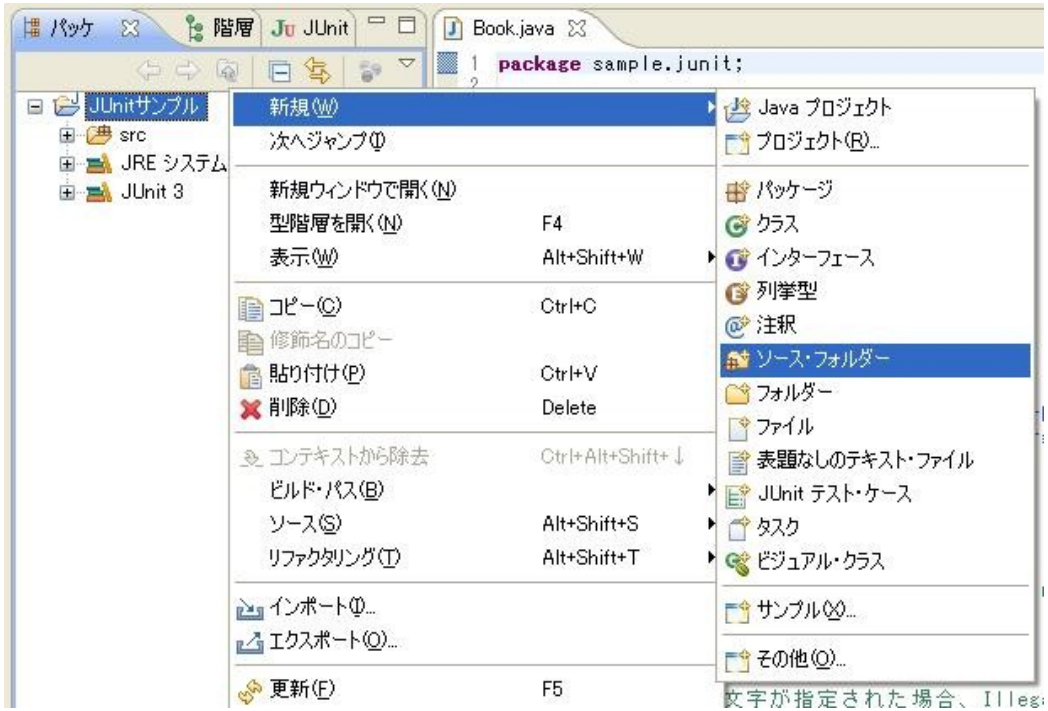
サンプルプログラム

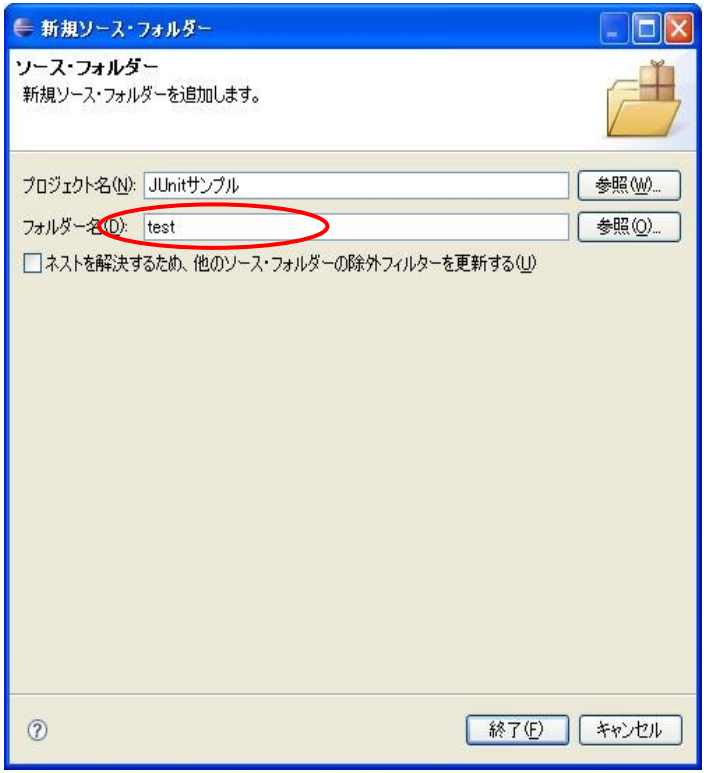

図書館クラス
本の貸し出し
本の返却
在庫数の更新
Library
+ rent(id : String) : Book
+ restore(restoreBook : Book) : void
本クラス
IDの保持
名称の保持
Book
- _name : String
- _id : String
+ Book(name : String, id : String)
+ getName() : void
+ getId() : void
DB接続クラス
接続情報保持
接続
SQL発行
切断
DBConnection
- DRIVER : String
- URL : String
- USER : String
- PASSWORD : String
- _con : Connection
- _stmt : Statement
+ open() : void
+ executeQuery(sql : String) : ResultSet
+ executeUpdate(sql : String) : int
+ close() : void
«テーブル定義»
SAMPLE_BOOK_STORE
Id
Name
Count

テストケース作成の前に・・・

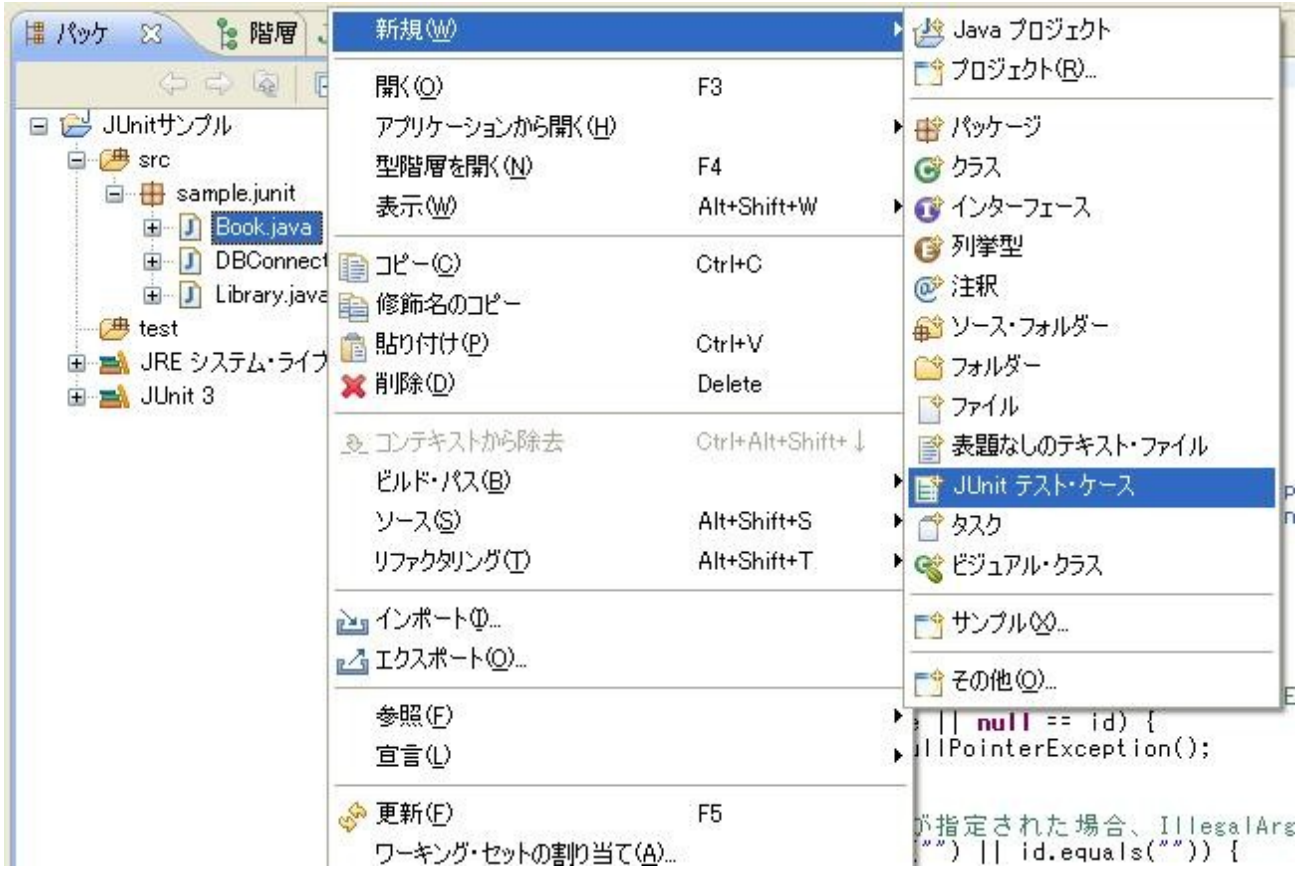
- テストケースの格納フォルダを決める
  ※今回はテスト用フォルダを作成

テストケース用フォルダの作成

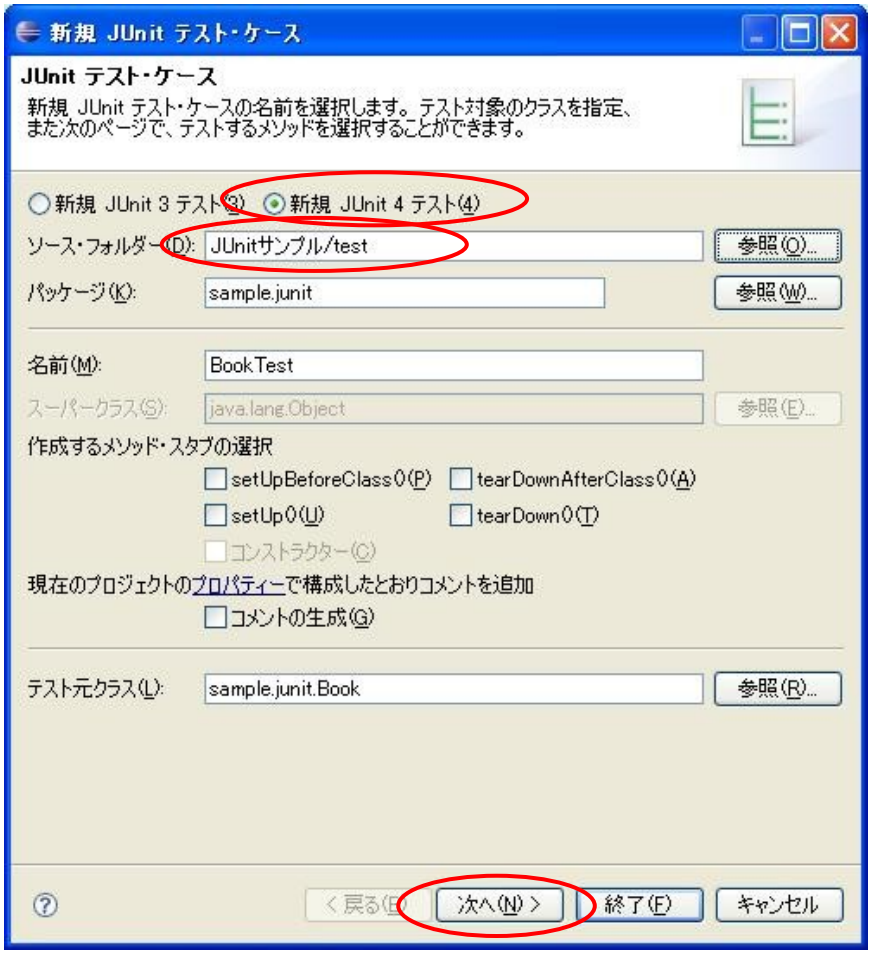


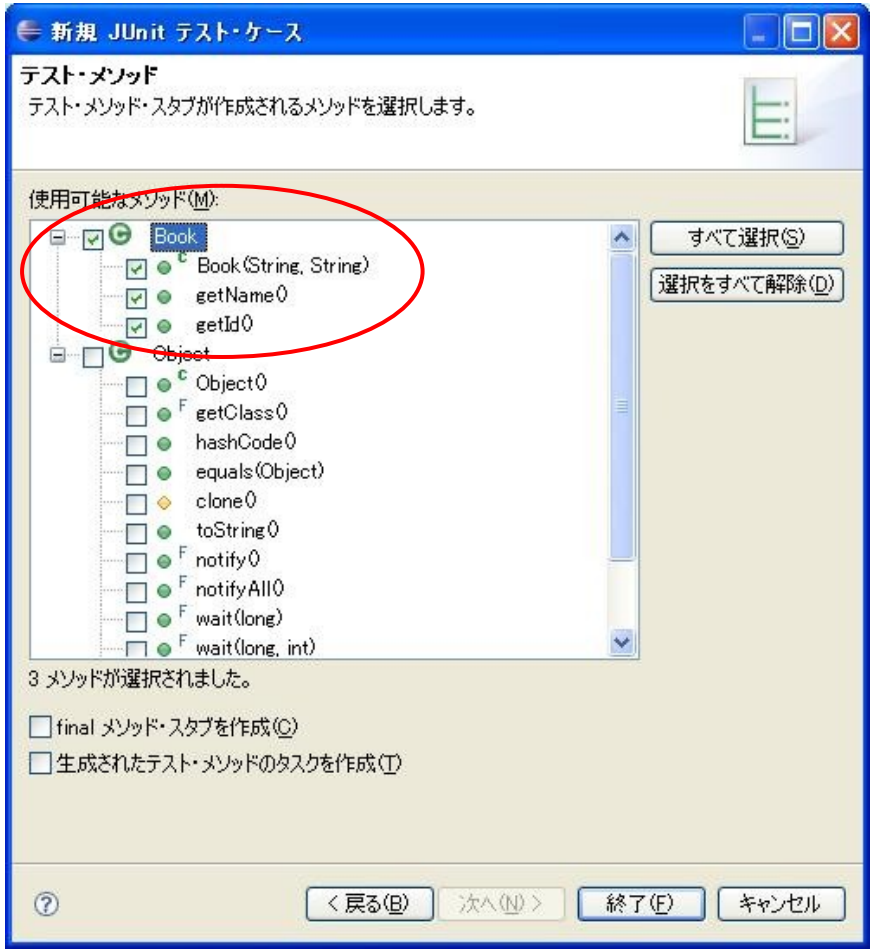


テストケース新規作成 1

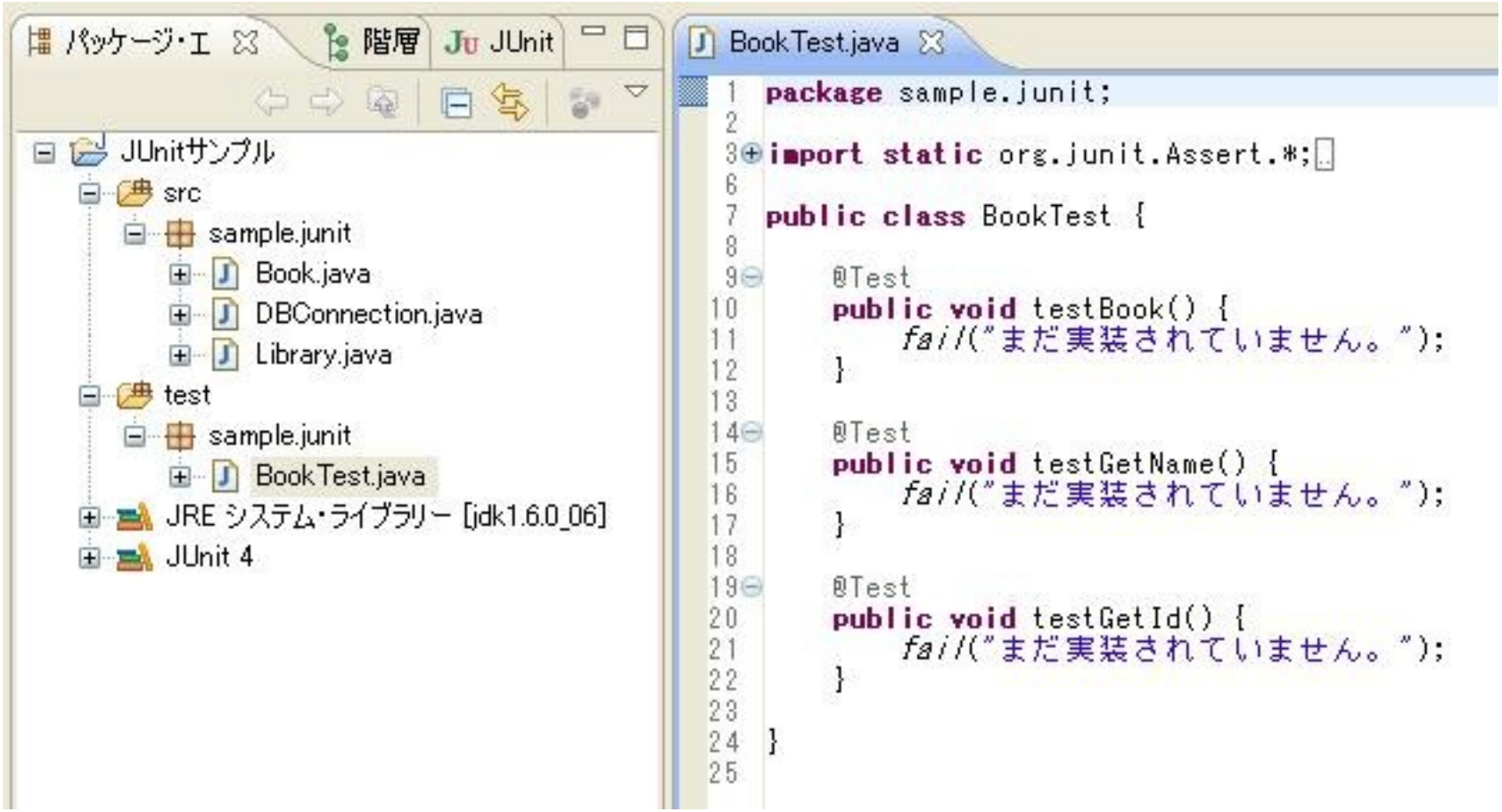


テストケース新規作成 2

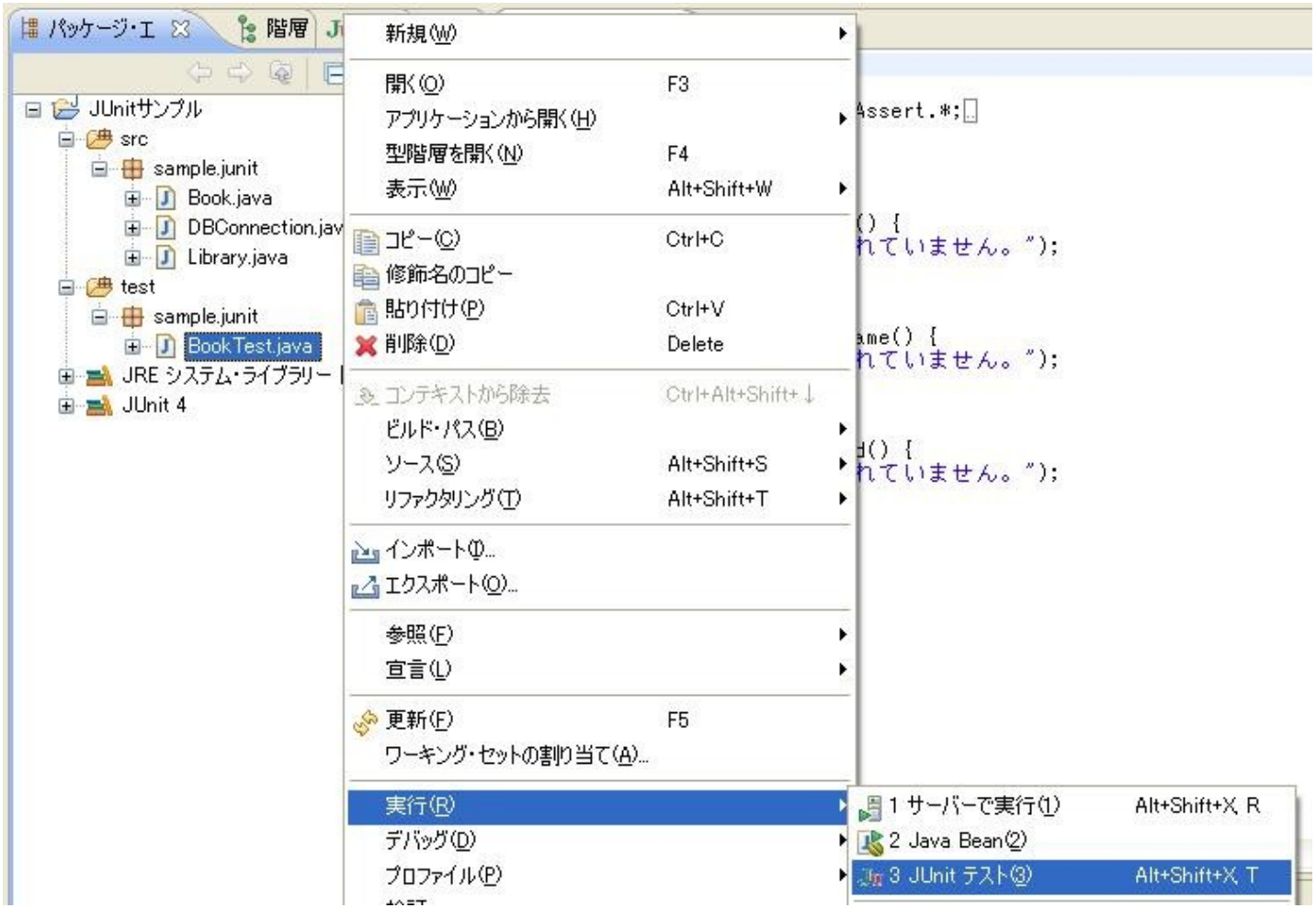




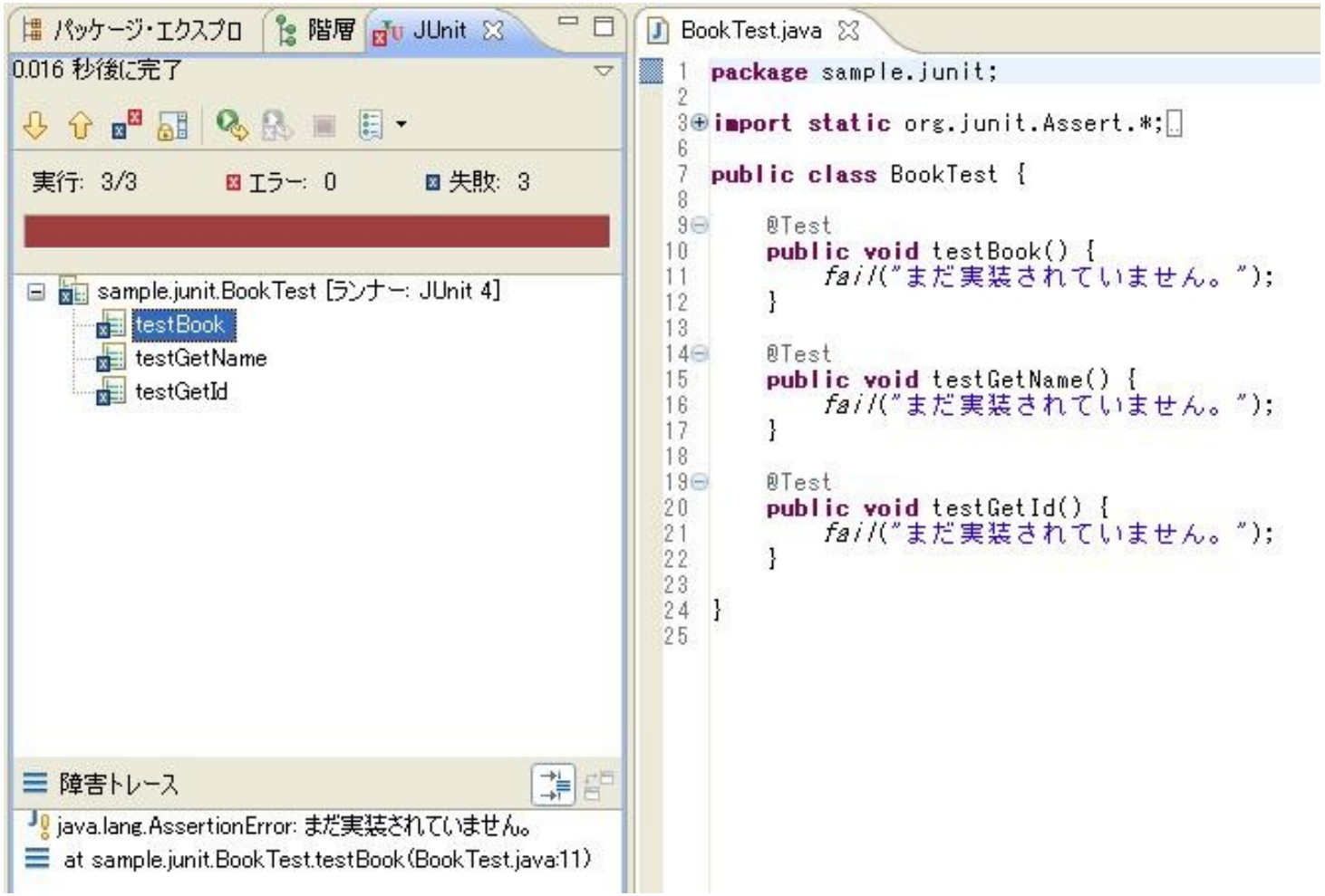
テストケース新規作成 3



テストを実施してみる


パッケージ・エ
階層
JUnitサンプル
src
sample.junit
Book.java
DBConnection.jav
Library.java
test
sample.junit
BookTest.java
JRE システム・ライブラリー
JUnit 4
新規(W)
開く(O) F3
アプリケーションから開く(H)
型階層を開く(N) F4
表示(W) Alt+Shift+W
コピー(C) Ctrl+C
修飾名のコピー
貼り付け(P) Ctrl+V
削除(D) Delete
コンテキストから除去 Ctrl+Alt+Shift+↓
ビルド・パス(B)
ソース(S) Alt+Shift+S
リファクタリング(T) Alt+Shift+T
インポート(I)...
エクスポート(O)...
参照(F)
宣言(L)
更新(F) F5
ワーキング・セットの割り当て(A)...
実行(R)
デバッグ(D)
プロファイル(P)
1 サーバーで実行(1) Alt+Shift+X, R
2 Java Bean(2)
3 JUnit テスト(3) Alt+Shift+X, T
Assert.*;
れていません。");
れていません。");
れていません。");

テスト結果


パッケージ・エクスプロ
階層
JUnit
0.016 秒後に完了
実行: 3/3
エラー: 0
失敗: 3
sample.junit.BookTest [ランナー: JUnit 4]
testBook
testGetName
testGetId
障害トレース
java.lang.AssertionError: まだ実装されていません。
at sample.junit.BookTest.testBook(BookTest.java:11)
BookTest.java
1 package sample.junit;
2
3 import static org.junit.Assert.*;
6
7 public class BookTest {
8
9 @Test
10 public void testBook() {
11 fail("まだ実装されていません。");
12 }
13
14 @Test
15 public void testGetName() {
16 fail("まだ実装されていません。");
17 }
18
19 @Test
20 public void testGetId() {
21 fail("まだ実装されていません。");
22 }
23
24 }
25

テストケースを実装する

JUnitで用意されているメソッド

assertEquals	引数同士が同じ値かを判別
assertTrue	引数がtrueかどうかを判別
assertFalse	引数がfalseかどうかを判別
assertNotNull	引数がNullでないかを判別
assertNull	引数がNullかどうかを判別
assertSame	引数同士が同じ参照かを判別
assertNotSame	引数同士が違う参照かを判別
fail	強制的に失敗になる

単体テストの内容

- パラメータ
- 閾値
  - 閾値と等価、閾値を超える、閾値未満など
- 文字列
  - 空文字、半角、全角など
- 桁数
- 例外処理
- etc…

テストケースの書き方

- 具体的に書く
  - 一目でテストの意味がわかるように
  - 投入値・期待値・結果を具体的に
  - 「わかりにくいが短いコード」よりも「具体的で長い（程度にもよるが）コード」
- テスト範囲に関係ない処理は抽象化
  - 投入値や条件の指定が長くなる箇所

テストケースの実装
（Bookクラス　コンストラクタ 1）

<class Book>

コンストラクタ

```
23    public Book(String name, String id) {
24        // 引数にnullが指定された場合、NullPointerExceptionをthrow
25        if (null == name || null == id) {
26            throw new NullPointerException();
27        }// else {}
28
29        // 引数に空文字が指定された場合、IllegalArgumentExceptionをthrow
30        if (name.equals("") || id.equals("")) {
31            throw new IllegalArgumentException();
32        }// else {}
33
34        _name = name;
35        _id = id;
36    }
```

<class BookTest>

コンストラクタの

テスト

```
15    /**
16     * コンストラクタのテスト1<br>
17     * 第1引数に"testName"、第2引数に"testId"を指定した場合、
18     * インスタンスが生成される事
19     */
20    @Test
21    public void testBook1() {
22        Book test = new Book("testName", "testId");
23
24    }
```

テストケースの実装
（Bookクラス　コンストラクタ 2）

<class Book>

コンストラクタ

```
23    public Book(String name, String id) {
24        // 引数にnullが指定された場合、NullPointerExceptionをthrow
25        if (null == name || null == id) {
26            throw new NullPointerException();
27        }// else {}
28
29        // 引数に空文字が指定された場合、IllegalArgumentExceptionをthrow
30        if (name.equals("") || id.equals("")) {
31            throw new IllegalArgumentException();
32        }// else {}
33
34        _name = name;
35        _id = id;
36    }
```

<class BookTest>

コンストラクタの

テスト

```
26    /**
27     * コンストラクタのテスト2<br>
28     * 第1引数にnullを指定した場合、NullPointerExceptionを発生する事
29     */
30    @Test(expected=NullPointerException.class)
31    public void testBook2() {
32        Book test = new Book(null, "testId");
33    }
```

テスト範囲に関係ない処理の抽象化
例）リフレクション

```
 8 public class ReflectionUtil {
 9
10     /**
11      * 指定されたインスタンスの指定されたフィールド名に指定値を格納<br>
12      * 処理内でエラー発生の場合、スタックトレース出力の上、テストを強制エラー
13      *
14      * @param instance 値を格納するインスタンス
15      * @param fieldName フィールド名
16      * @param param 格納値
17      */
18     public static void setField(Object instance, String fieldName, Object param) {
19         try {
20             Class cls = instance.getClass();
21             Field field = cls.getDeclaredField(fieldName);
22             field.setAccessible(true);
23             field.set(instance, param);
24         } catch (SecurityException e) {
25             e.printStackTrace();
26             fail();
27         } catch (IllegalArgumentException e) {
28             e.printStackTrace();
29             fail();
30         } catch (NoSuchFieldException e) {
31             e.printStackTrace();
32             fail();
33         } catch (IllegalAccessException e) {
34             e.printStackTrace();
35             fail();
36         }
37     }
38 }
```

テストケースの実装
（Bookクラス　getName）ReflectionUtilを利用

<class Book>

getName () メソッド

```
38      /**
39       * 名称を返却
40       *
41       * @return 名称
42       */
43      public String getName() {
44          return _name;
45      }
```

<class BookTest>

getName () メソッドのテスト

```
82      /**
83       * getName()のテスト1
84       * フィールド"_name"に格納された値"testName"を取得できる事
85       */
86      @Test
87      public void testGetName1() {
88          Book test = new Book("test", "test");
89          // フィールド"_name"に"testName"を格納
90          ReflectionUtil.setField(test, "_name", "testName");
91
92          assertEquals("testName", test.getName());
93      }
```

Libraryクラス　rentメソッド

```
19    public Book rent(String id) throws Exception {
20
21        // 引数のnullチェック
22        if (null == id) {
23            throw new NullPointerException();
24        }// else {}
25
26        Book book = null;
27        DBConnection db = null;
28        ResultSet rs = null;
29        try {
30            // DB接続インスタンス生成
31            db = new DBConnection();
32            // DB接続
33            db.open();
34
35            // SELECT文発行
36            String selectSql = "SELECT Count "
37                             + "FROM SAMPLE_BOOK_STORE "
38                             + "WHERE Id = '" + id + "'";
39            rs = db.executeQuery(selectSql);
40
41            // 在庫数取得
42            int count = rs.getInt("Count");
43            // 本の名称取得
44            String name = rs.getString("Name");
```

テストケースの実装
（Libraryクラス　rent）

<class LibraryTest>

rent () メソッドのテスト

void sample.junit.LibraryTest.testRent1()

rent()のテスト1

SAMPLE_BOOK_STORE		
Id	Name	Count
testId	testName	1

上記条件にて引数に"testId"を指定した場合、
ID:"testId",名称:"testName"がセットされたBookインスタンスを返却する事

```
13      /**
14       * rent()のテスト1<br>
15       * <table border=1>
16       * <tr><th colspan=3>SAMPLE_BOOK_STORE</th></tr>
17       * <tr><td>Id</td><td>Name</td><td>Count</td></tr>
18       * <tr><td>testId</td><td>testName</td><td>1</td></tr>
19       * </table>
20       * 上記条件にて引数に"testId"を指定した場合、
21       * ID:"testId",名称："testName"がセットされたBookインスタンスを返却する事
22       */
23      public void testRent1() {
24          Library test = new Library();
25          Book actualBook = null;
26          try {
27              actualBook = test.rent("testId");
28          } catch (Exception e) {
29              fail();
30          }
31
32          assertNotNull(actualBook);
33          assertEquals("testId", actualBook.getId());
34          assertEquals("testName", actualBook.getName());
35      }
```

DBにアクセスする処理があると・・・

- いろいろな問題が浮上する
  - 本番環境のDBではテスト不可能
  - テスト用DBの用意
  - テストをする為の作業が発生
    - テストに合わせたデータをセット
    - 他テストチームとの調整
  - etc…

DBに依存しないテストを行うには

- DB接続生成部分をメソッドに割る
- モック(擬似)オブジェクトを作成する

DB接続部分をメソッドに割る

<class Library>rent () メソッド

変更前

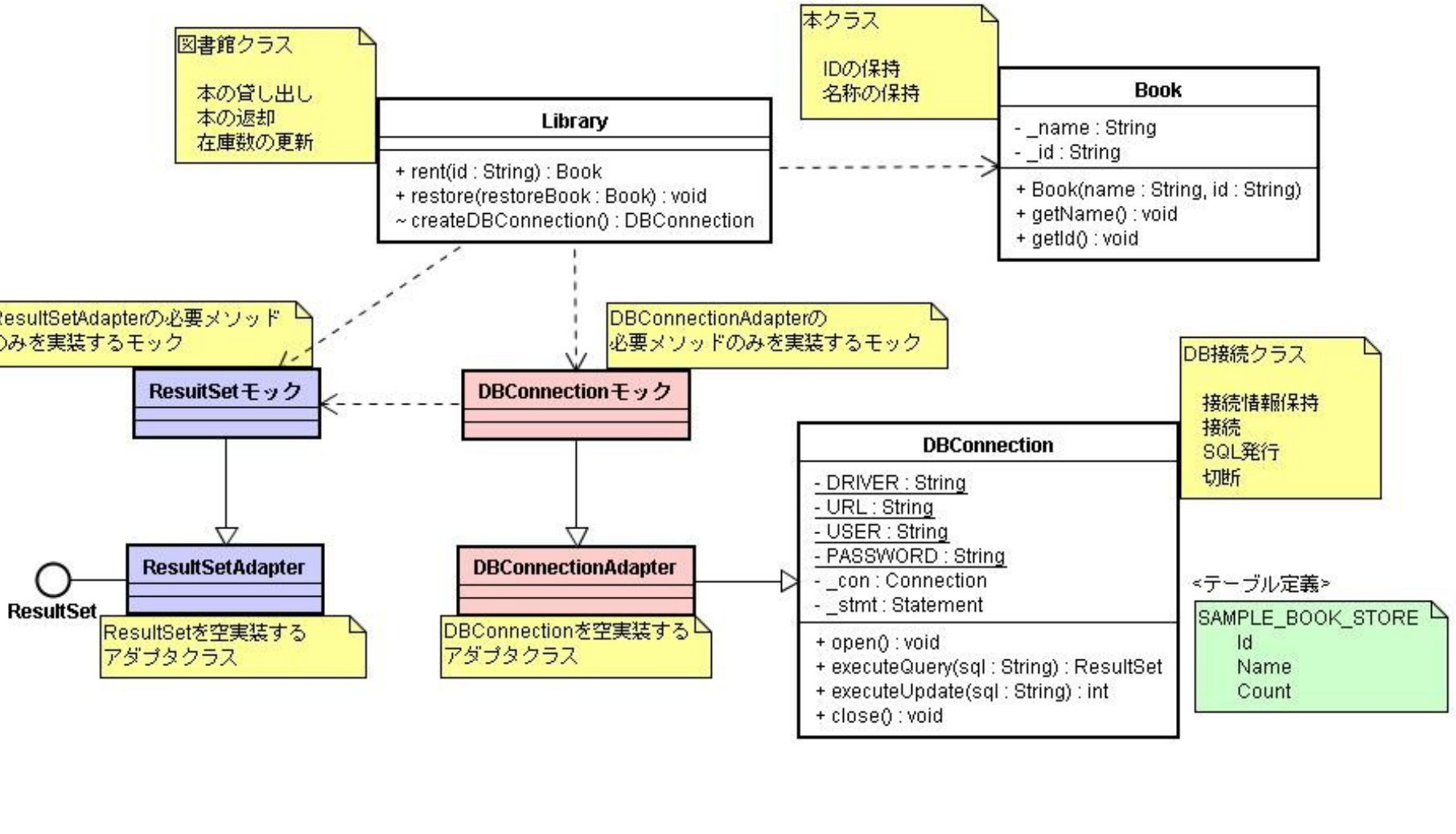
```
26  Book book = null;
27  DBConnection db = null;
28  ResultSet rs = null;
29  try {
30      // DB接続インスタンス生成
31      db = new DBConnection();
32      // DB接続
33      db.open();
```

変更後

```
25  Book book = null;
26  DBConnection db = null;
27  ResultSet rs = null;
28  try {
29      // DB接続インスタンス生成
30      db = createDBConnection();
31      // DB接続
32      db.open();
```

```
101 /**
102  * DBConnectionを返却する
103  *
104  * @return DB接続インスタンス
105  */
106 DBConnection createDBConnection() {
107     return new DBConnection();
108 }
```

モックオブジェクトを作成する



テスト範囲に関係ない処理の抽象化
例）生成メソッドを用意する

<class LibraryTest>　rent () メソッドのテスト

メンバ変数を追加

```
31      /** DB接続モックオブジェクト */
32      private DBConnectionAdapter _dbConnection = null;
```

Libraryインスタンス生成メソッド追加

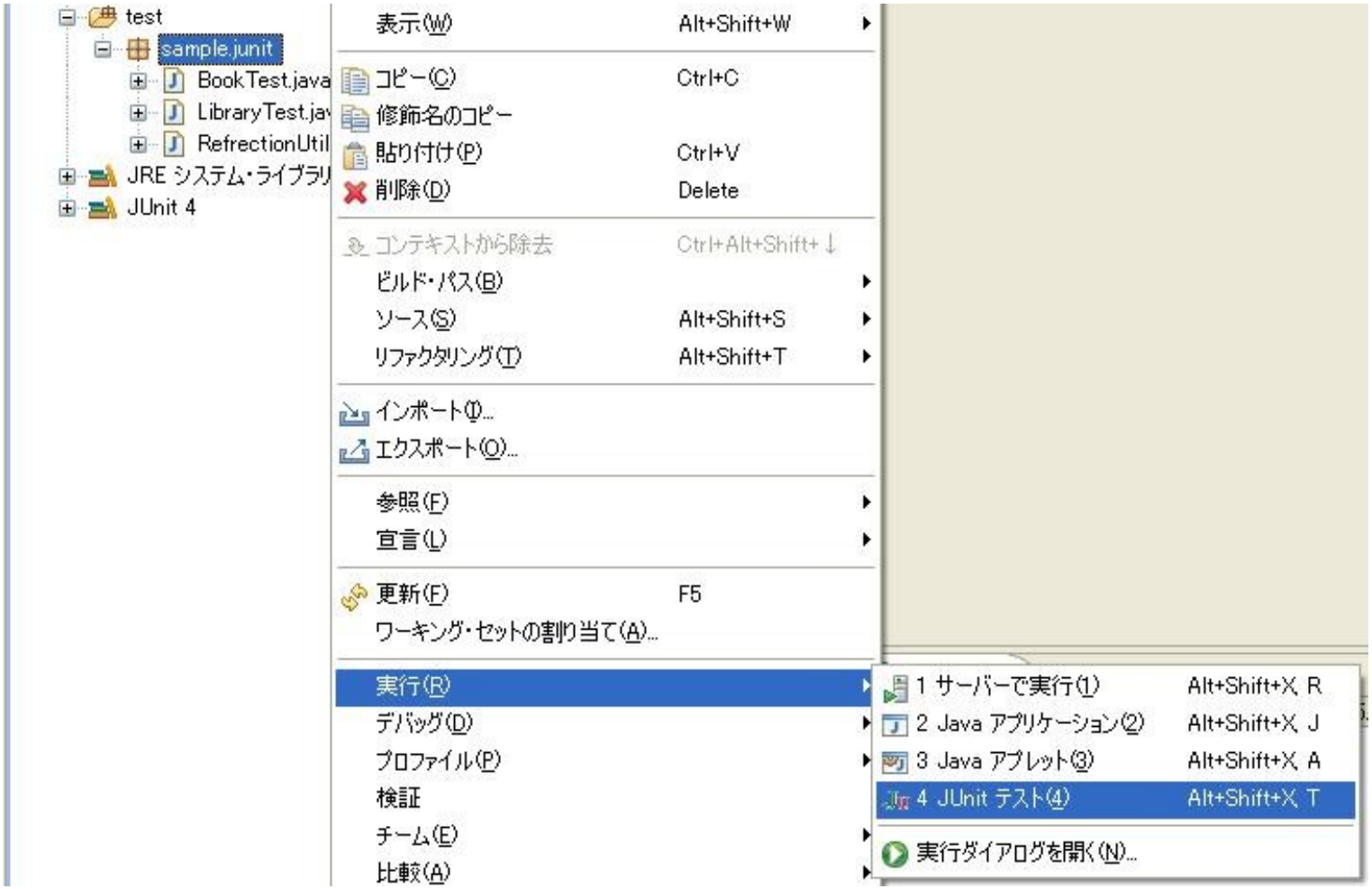
```
79      /**
80       * DB接続モックオブジェクトを利用するLibraryインスタンスを生成
81       *
82       * @return DB接続モックオブジェクトを利用するLibraryインスタンス
83       */
84      private Library createLibrary() {
85          return new Library() {
86              @Override
87              DBConnection createDBConnection() {
88                  return _dbConnection;
89              }
90          };
91      }
```

モックオブジェクトを実装する

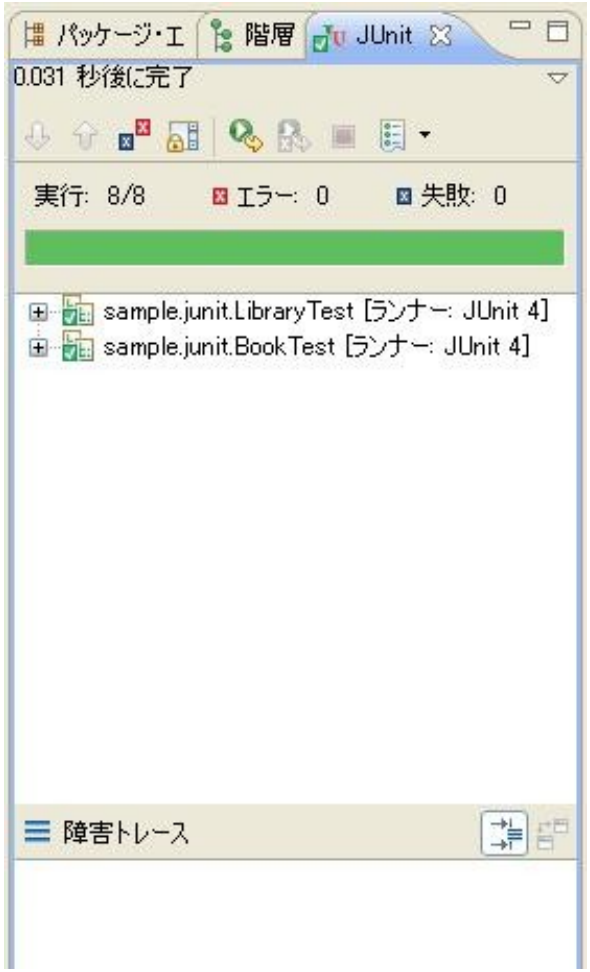
<class LibraryTest>　rent () メソッドのテスト

```
44    @Test
45    public void testRent1() {
46        // モックDB接続の振る舞いを定義
47        _dbConnection = new DBConnectionAdapter() {
48            @Override
49            public ResultSet executeQuery(String sql) throws Exception {
50                return new ResultSetAdapter() {
51                    @Override
52                    public int getInt(String columnLabel)
53                            throws SQLException {
54                        return 1;
55                    }
56
57                    @Override
58                    public String getString(String columnLabel)
59                            throws SQLException {
60                        return "testName";
61                    }
62                };
63            }
64        };
65
66        Library test = createLibrary();
67        Book actualBook = null;
68        try {
69            actualBook = test.rent("testId");
70        } catch (Exception e) {
71            fail();
```

テスト実施


test
sample.junit
BookTest.java
LibraryTest.jav
RefrectionUtil
JRE システム・ライブラリ
JUnit 4
表示(W) Alt+Shift+W
コピー(C) Ctrl+C
修飾名のコピー
貼り付け(P) Ctrl+V
削除(D) Delete
コンテキストから除去 Ctrl+Alt+Shift+↓
ビルド・パス(B)
ソース(S) Alt+Shift+S
リファクタリング(T) Alt+Shift+T
インポート(I)...
エクスポート(O)...
参照(F)
宣言(L)
更新(F) F5
ワーキング・セットの割り当て(A)...
実行(R)
デバッグ(D)
プロファイル(P)
検証
チーム(E)
比較(A)
1 サーバーで実行(1) Alt+Shift+X, R
2 Java アプリケーション(2) Alt+Shift+X, J
3 Java アプレット(3) Alt+Shift+X, A
4 JUnit テスト(4) Alt+Shift+X, T
実行ダイアログを開く(N)...

テスト結果


パッケージ・エ
階層
JUnit
0.031 秒後に完了
実行: 8/8
エラー: 0
失敗: 0
sample.junit.LibraryTest [ランナー: JUnit 4]
sample.junit.BookTest [ランナー: JUnit 4]
障害トレース

JUnitを上手に利用して
効果的で効率の良いテストを