1．JUnitでテストするコード

4行目のメソッドに対して、別にテストクラスを新規作成してJUnitでテストを行います。

テストクラスでは、以下のテストメソッドを記述します。

・引数に1をセットしたら「赤」が返ってくること

・引数に2をセットしたら「青」が返ってくること

・引数に1,2以外をセットしたら「1or2を入力して下さい」が返ってくること

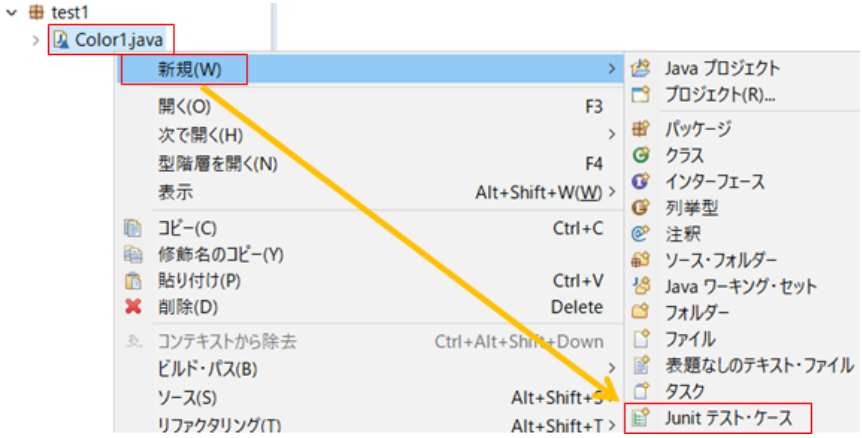
```
package test1;

class Color1 {


        if (i == 1) {
            return "赤";

        } else if (i == 2) {
            return "青";

        } else {
            return "1or2を入力して下さい";
        }
    }
}
```

2．テストクラスを作成する

テストクラスを作成します。

１．テストするクラスを右クリックし「新規」→「JUnitテスト・ケース」をクリックします。

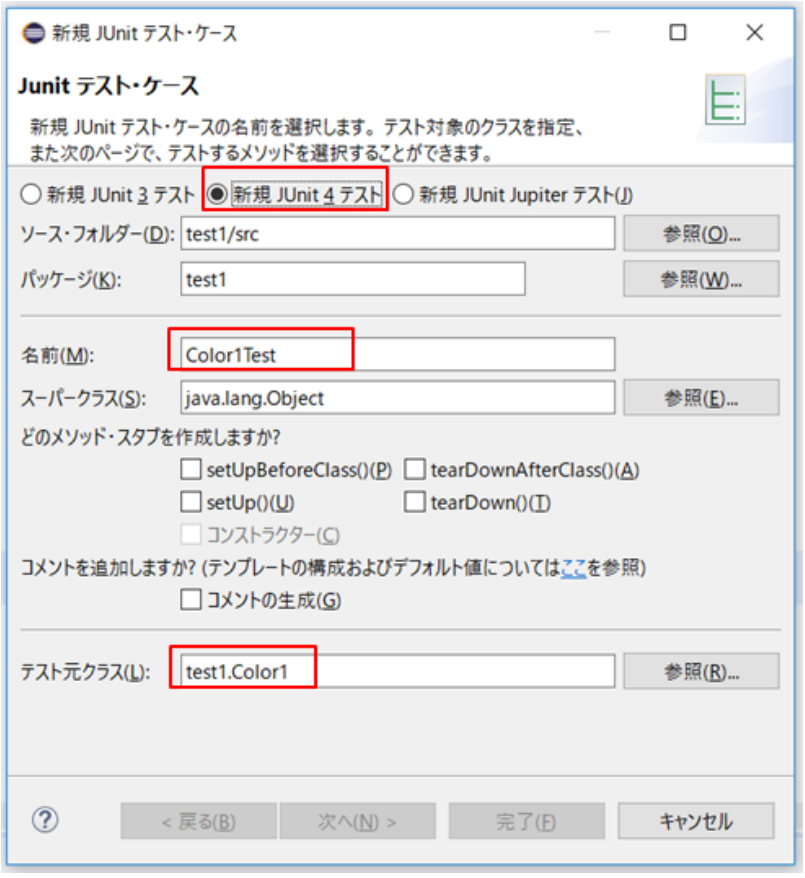
※ない場合は、「新規」→「その他」→「Java」→「JUnit」を選択してJUnitテスト・ケースを選択します。



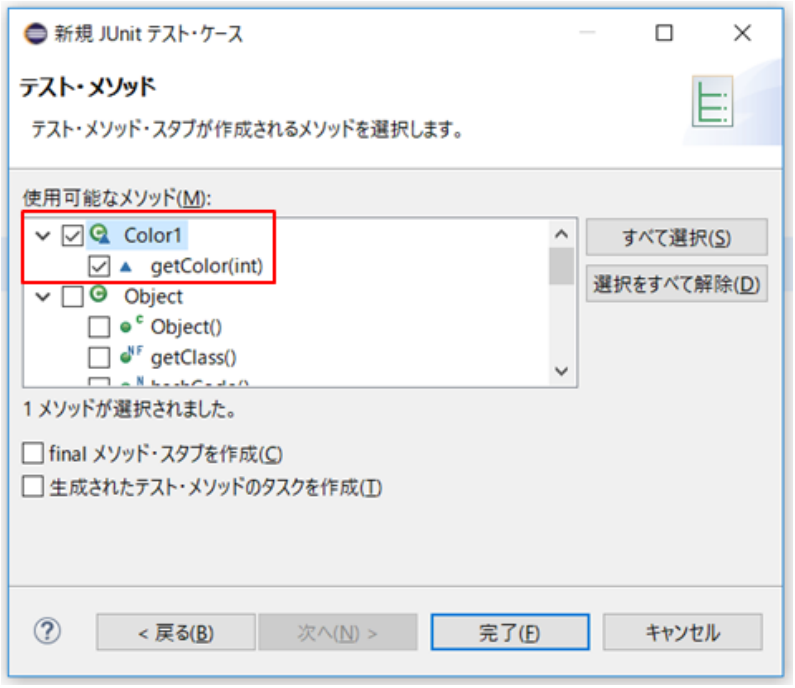
2．「新規JUnit4テスト」を選択します。

「名前」はクラス名をつけます。クラス名はテストクラスとわかるようにTestの文字を入れます。

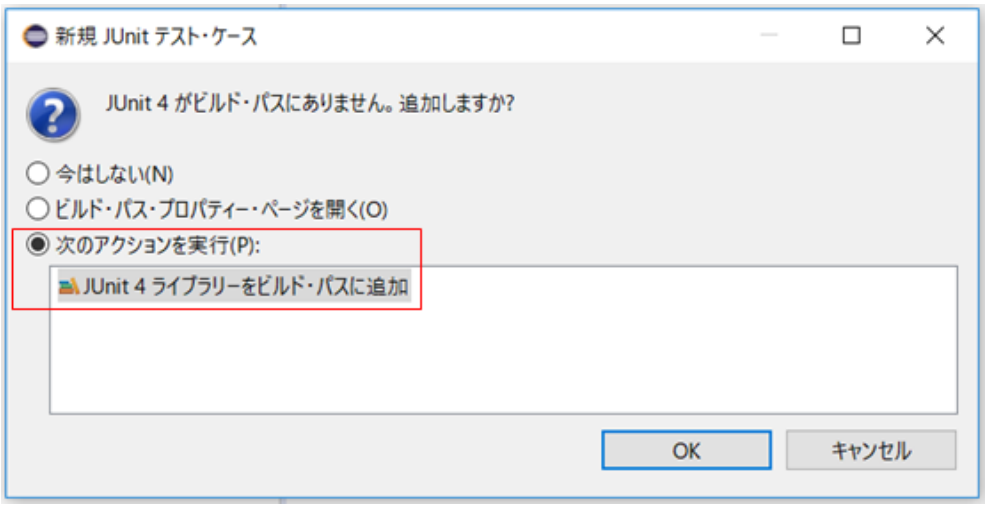
「テスト元クラス」は上記手順で右クリックしたクラスが入っています。「次へ」を押します。



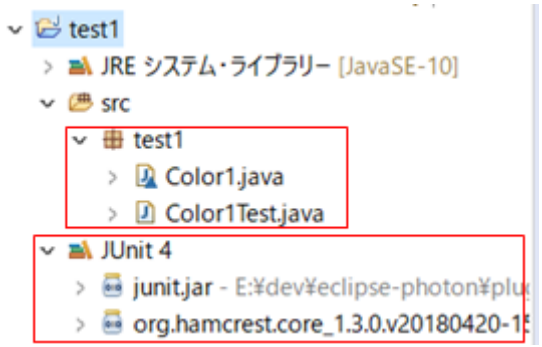
3．テストするクラスとメソッドにチェックを入れ、「完了」を押します。



4．JUnit4ライブラリがビルドパスに追加されていない場合、以下のダイアログが表示されます。「JUnit4ライブラリーをビルド・パスに追加」の上にある「次のアクションを実行」を選択して「OK」を押します。



5．フォルダとファイル構成は以下のようになります。



６．作成したテストクラスのコードは以下のようになります。

8行目のfailは、テストを失敗させるメソッドです。

```
package test1;
import static org.junit.Assert.*;
import org.junit.Test;

public class Color1Test {
    @Test
    public void testGetColor() {

    }
}
```

３．テストクラスを修正する

テストクラスを以下のように修正します。

```
package test1;
import static org.junit.Assert.*;
import org.junit.Test;


public class Color1Test {

    @Test
    public void testGetColor1() {
        Color1 c1 = new Color1();
        String t1 = c1.getColor(1);
```

```
            assertThat(t1,is("赤"));
        }
        @Test
        public void testGetColor2() {
            Color1 c1 = new Color1();

            assertThat(t1,is("青"));
        }
        @Test
        public void testGetColor3() {
            Color1 c1 = new Color1();

            assertThat(t1,is("1or2を入力して下さい"));
        }
    }
```

9,15,21行目は、テストメソッドとわかるようにtestの文字を入れます。

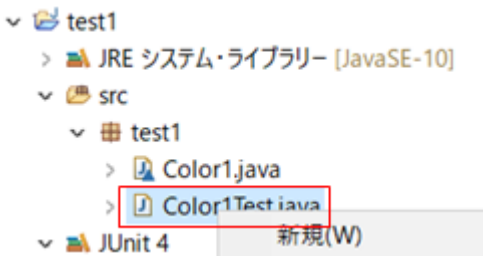
テストを行うメソッドごとに@Testアノテーションをつけます。

assertThatのisメソッドを使用するため4行目を手動で追加(コピペ)します。
12,18,24行目は、メソッドの戻り値が想定の値かどうかを確認しています。
比較するメソッドはassertEqualsではなく、Junit4.4で追加されたassertThatが良いです。
可読性が上がります。

4．テストクラスを実行する

テストクラスを実行します。

1．テストクラスを右クリックしダイアログを開きます。

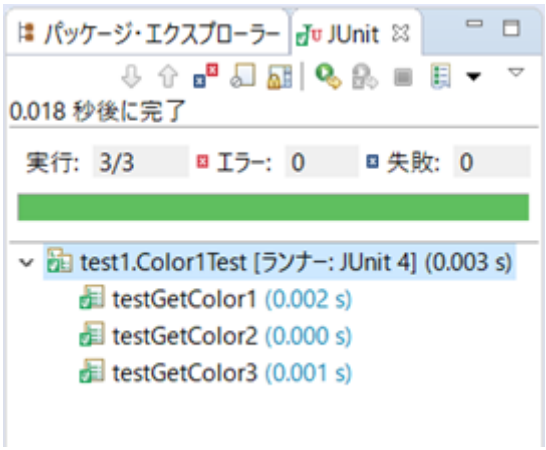


２．ダイアログの下にある「実行」>「JUnitテスト」をクリックするとJUnitテストが実行されます。

３．想定どうりの結果のため成功しました。

成功だと緑色の帯が表示されます。

test1.Color1Testの横の印をクリックすると下にメソッドが展開されます。

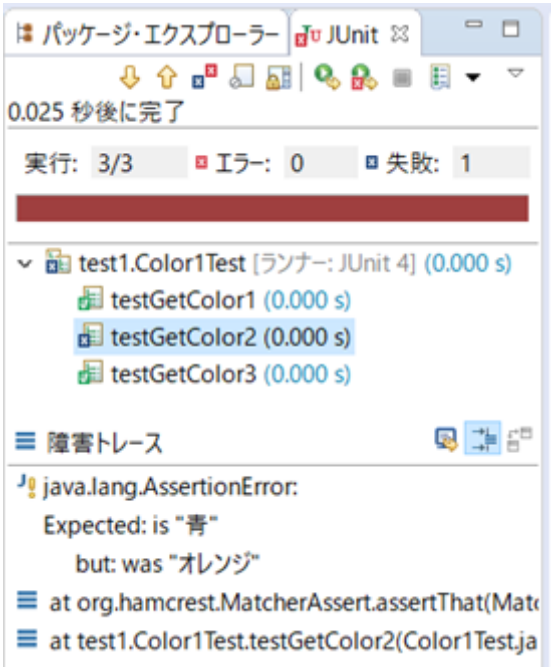


４．想定が違った場合、赤の帯が表示されます。

対象のメソッドを選択すると障害トレースに内容が表示されます。

この例では、testGetColor2を選択しました。

→テストクラスは「青」を想定していましたが、結果は「オレンジ」でした。



5．例外をテストする方法

例外をテストする方法です。

6行目は、強制的にNullPointerExceptionの例外を投げるようにしています。

テストクラスでは、引数に1をセットしたらNullPointerExceptionになることを確認します。

```
package test1;

class Color1 {
    String getColor(int i) {
        if (i == 1) {

        } else {
            return "１以外です";
        }
    }
}
```

以下はテストクラスです。

```
package test1;
import static org.junit.Assert.*;
```

```
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;
import static org.hamcrest.CoreMatchers.*;

public class Color1Test {
    //テストメソッド　その１

    public void testGetColor1() {
        Color1 c1 = new Color1();
        String t1 = c1.getColor(1); //エラーなし
    }
    //テストメソッド　その２
    @Test
    public void testGetColor2() {

            Color1 c1 = new Color1();
            String t1 = c1.getColor(1); //エラーになる

            assertThat(expected.getMessage(), equalTo("テスト１"));
        }
    }
    //テストメソッド　その３
    @Rule
    public ExpectedException ee1 = ExpectedException.none();
    @Test
    public void testGetColor3() {


            Color1 c1 = new Color1();
            String t1 = c1.getColor(1); //エラーなし
    }
}
```

テストメソッドその１

10行目は、@Testの後にexpectedとNullPointerException.classを指定しています。

次のテストでは、NullPointerExceptionが発生する想定になります。

13行目は、引数に1をセットしていますが、JUnitでは成功になります。

10行目の@Test以下を削除すると、13行目は失敗になります。

テストメソッドその２

16-24行目は、上記と同じくNullPointerExceptionになることを想定しています。
20行目でNullPointerExceptionが発生するので21行目のcatchに投げられ22行目でJUnitの成功になります。

テストメソッドその３

26行目は、@Ruleアノテーションをつけています。
28-34行目は、上記と同じくNullPointerExceptionになることを想定しています。
30,31行目で例外と文言を指定しています。
33行目はエラーは発生しません。JUnitで成功になります。

６．コンストラクタでセットされた値をテストする方法

コンストラクタでセットされた値をテストする方法です。

6-8行目は、コンストラクタで4行目のメンバ変数に値をセットしています。
テストクラスでは、インスタンス生成時に、想定した値が4行目のメンバ変数にセットされていることを確認します。

```
package test1;

class Color1 {



        this.name1 = "色は、" + name + "です";
    }
    public String getName() {
        return name1;
    }
}
```

以下はテストクラスです。

```
package test1;
import static org.junit.Assert.*;
import org.junit.Test;
import static org.hamcrest.CoreMatchers.*;

public class Color1Test {

    @Test
    public void testGetColor1() {
        Color1 c1 = new Color1("赤");

    }
}
```

10行目は、コンストラクタの引数を指定しています。

11行目は、JUnitで成功になります。

7．JUnit4で使用するアノテーション(@Before,@After)

JUnit4で使用するアノテーションです。

```
package test1;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Ignore;
import org.junit.Test;

public class Test1Test {

    public static void testBeforeClass() {
        System.out.println("BeforeClass");
    }

    public static void testAfterClass() {
        System.out.println("AfterClass");
    }

    public void testBefore() {
        System.out.println("Before");
```

```
    }

    @After
    public void testAfter() {
        System.out.println("After");
    }
    @Test
    public void test1() {
        System.out.println("test1");
    }

    @Ignore
    public void test2() {
        System.out.println("test2");
    }
    @Test
    public void test3() {
        System.out.println("test3");
    }
}
```

10行目の@BeforeClassは、テストクラス単位で最初に1回実行されます。

14行目の@AfterClassは、テストクラス単位で最後に1回実行されます。

18行目の@Beforeは、テストメソッドの前に実行されます。

22行目の@Afterは、テストメソッドの後に実行されます。

30行目の@Ignoreは、テストメソッドを実行しません。

実行結果の流れは以下のようになります。

実行されるテストメソッドが2個あるのでBeforeとAfterは2回実行されます。

BeforeClass

Before

test1

After

Before

test3

After

AfterClass