

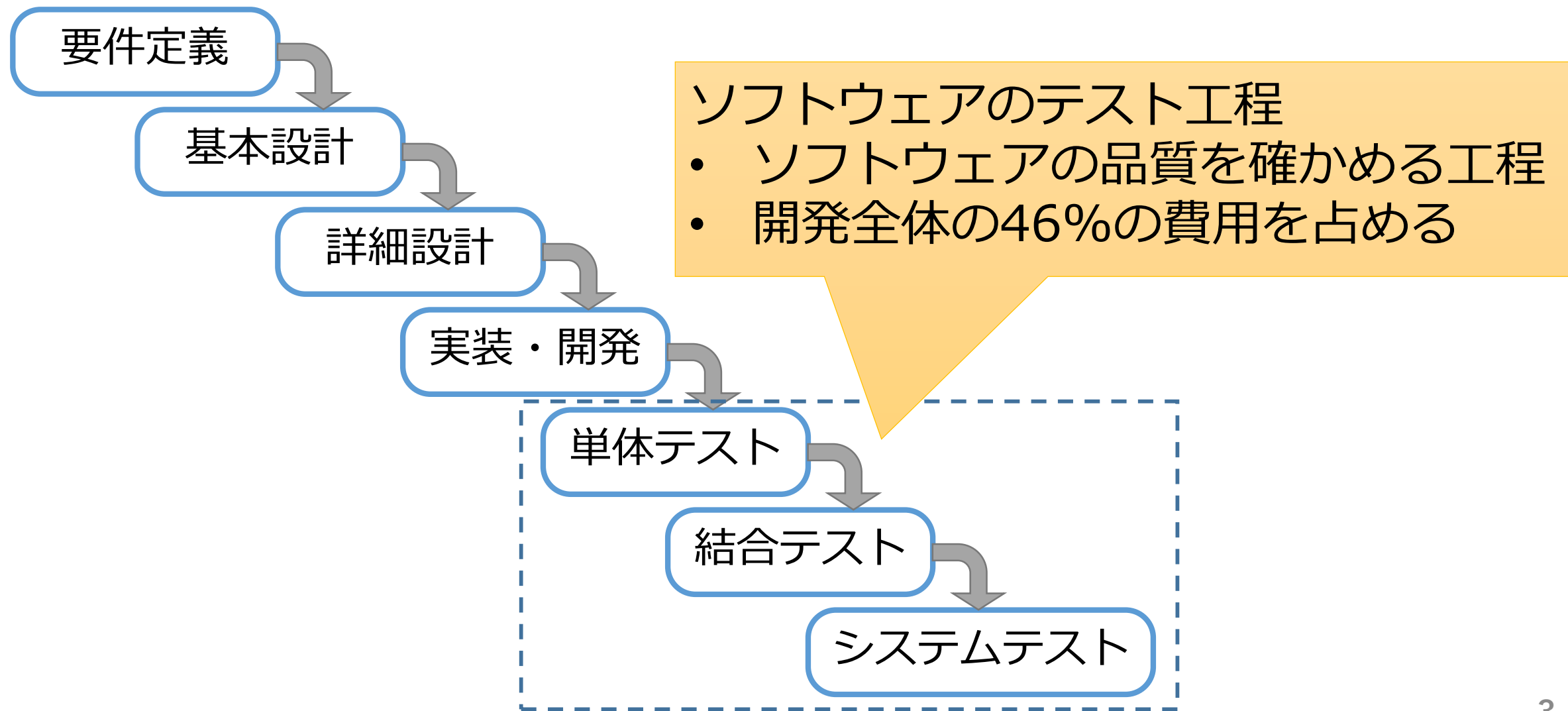
テストコード自動推薦 ツールに対する評価実験

NAIST SDLab 倉地亮介

本日の実験手順

1. ソフトウェアテストの説明 (5分)
2. テストコードの作成手順 (5分)
3. 事前練習 (10分)
4. 実践練習 (10分)
5. 評価実験 (最大60分)
6. アンケート調査 (10分)

ソフトウェアテストとは？



テストコードとは？

- 作成したプログラムを実行して、結果を確認すること
 - 単体テストでは、テスト対象を呼び出すテストコードを書いて実行

```
class Calculator{  
    int multiply(int x,int y)  
    {  
        return x * y;  
    }  
}
```

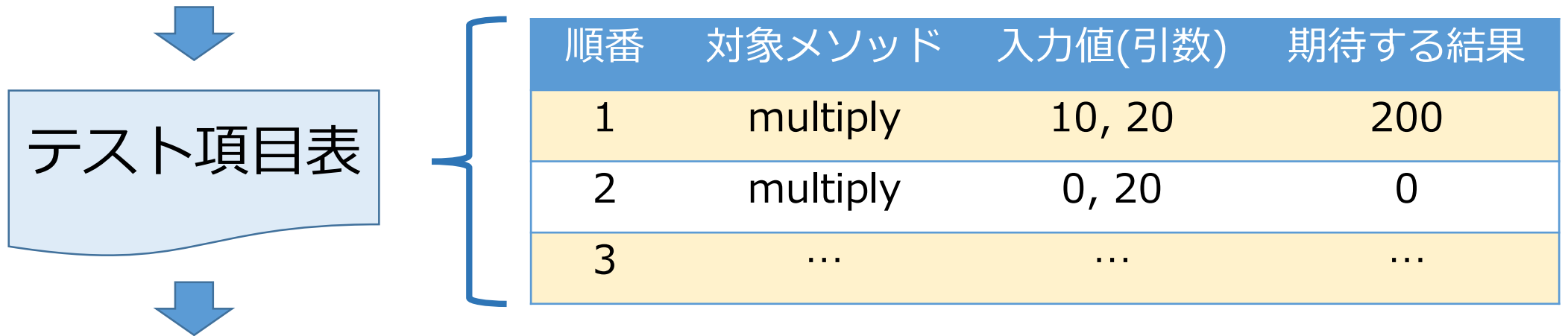
テスト対象コード

```
@Test  
void test(){  
    Calculator calc = new Calculator();  
    int result = calc.multiply(10,20);  
    assertEquals(200,result);  
}
```

テストコード

単体テストの作成手順

1. テスト項目(何を調べるか)を決める



2. テストコードを作成

3. テスト実施 (テストコード実行と結果の確認)

テストコードの書き方

テストコードの記述と実行

テストケースクラスの例

```
public class CalculatorTest {           // クラスCalculatorに対するテスト

    // テストメソッド。何をテストして、何を期待するのかを理解できる名前をつける
    @Test                               // テストメソッドを表すアノテーション
    public void testMultiply01() throws Exception {
        Calculator calc = new Calculator(); // テスト対象の準備
        int expected = 200;                // 期待する戻り値
        int actual = calc.multiply(10, 20); // テスト対象実行
        assertEquals(expected, actual);    // 実行結果の検証
    }

    @Test
    public void testMultiply02() throws Exception {
        // 以降、テスト項目毎にテストメソッドを作成
    }
}
```

テストケースクラスの見方

- 1テスト項目を1つのメソッドとして記述

テスト対象メソッドを呼び出す



出力が期待通りかどうか調べる

- 期待する結果かどうか調べるためのメソッド群 (assertXX)が用意されている

```
@Test    // @Testはテストメソッドを表すアノテーション
public void testMultiply01() throws Exception {
    int actual = new Calculator().multiply(10, 20);
    assertEquals(200, actual);
}
```


主なアサーションメソッド

メソッド名	説明
assertTrue(boolean)	引数がtrueかどうか検証する
assertFalse(boolean)	引数がfalseかどうか検証する
assertEquals(expected, actual)	2つの引数が同値であるかどうか検証する
assertSame(Object1, Object2)	2つの引数が同じオブジェクトかどうか検証する
assertNotSame(Object1, Object2)	2つの引数が異なるオブジェクトかどうか検証する
assertNull(Object)	引数がnullかどうか検証する
assertNotNull(Object)	引数がnullでないかどうか検証する
fail()	テストを強制的に失敗させる

本実験で主に用いる

テストカバレッジ

カバレッジの種類

カバレッジ = テストの網羅率

- プログラム全体の内、テスト中に実行された行の割合
- 「テスト中に一度も実行されない行がある」
 - その部分の品質は保証できない
 - (できる限り) カバレッジを100%に近づけるべき

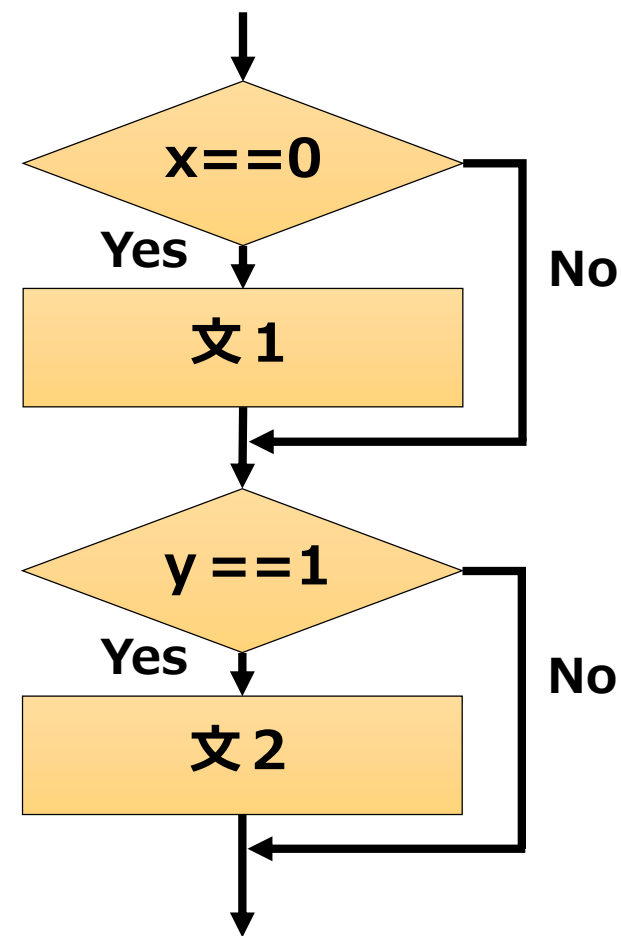
カバレッジの種類

- 命令網羅 C0

- 全ての命令を最低一回実行
- フローチャートの全節点通過
 - ✓ ($x=0$, $y=1$) を入力

- 分岐網羅 C1

- 全ての条件分岐について、then、else いずれも最低一回以上実行
- フローチャートの全辺通過
 - ✓ ($x=0$, $y=1$) と ($x=1$, $y=2$) を入力



練習問題

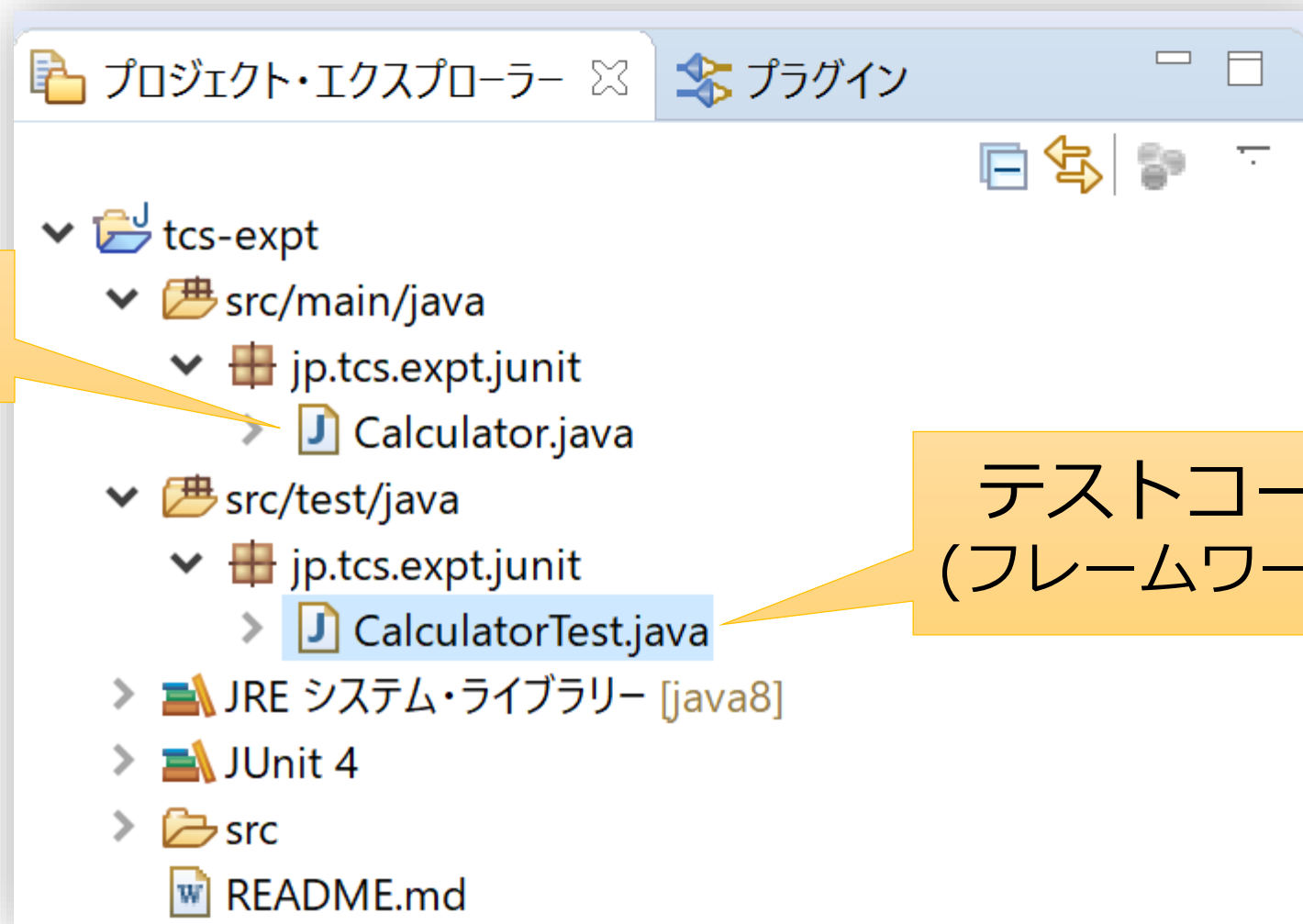
テストコードの記述と実行

事前練習 (10分)

- 簡単なテストケースを作成
 - 開発環境 : Eclipse oxygen
 - テスティングフレームワーク : JUnit4
 - テスト対象コード : Calculator.java
 - テスト対象コードはフォルダ **src/main/java** の下にある
 - テストコードはフォルダ **src/test/java** の下にある

練習プロジェクトの確認

テスト対象



テストコード置き場
(フレームワーク生成済み)

Eclipseの見方

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer showing a project structure with packages like `jp.tcs.expt.junit`. The main editor displays two files: `Calculator.java` and `CalculatorTest.java`.

Calculator.java (Test Target Code):

```
1 package jp.tcs.expt.junit;
2
3 public class Calculator {
4     /**
5      * 2引数の積を返す.
6      * @return xとyの積
7      */
8
9     public int multiply(int x, int y) {
10         return x * y;
11     }
12 }
13
14
```

CalculatorTest.java (Test Code):

```
1 package jp.tcs.expt.junit;
2
3 import static org.junit.Assert.*;
4
5
6 public class CalculatorTest {
7
8     @Test
9     public void test() {
10         fail("まだ実装されていません");
11     }
12 }
13
14
15
```

Test Execution Results:

要素	カバレッジ	カバー命令	未実行命令	合計命令数
tcs-expt	97.2 %	212	6	218

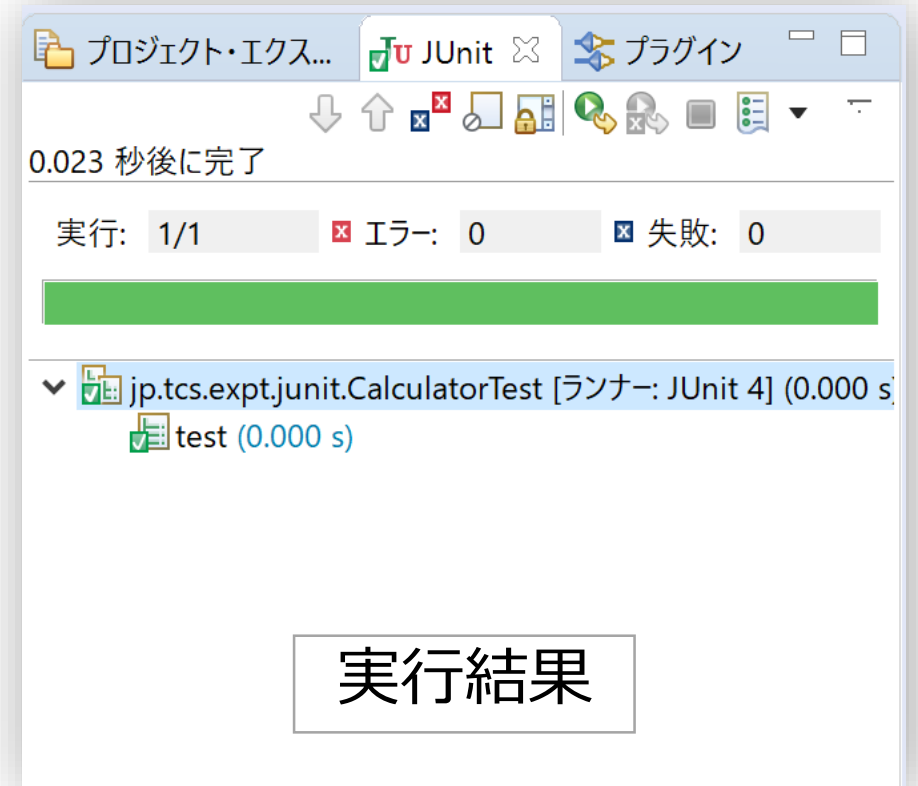
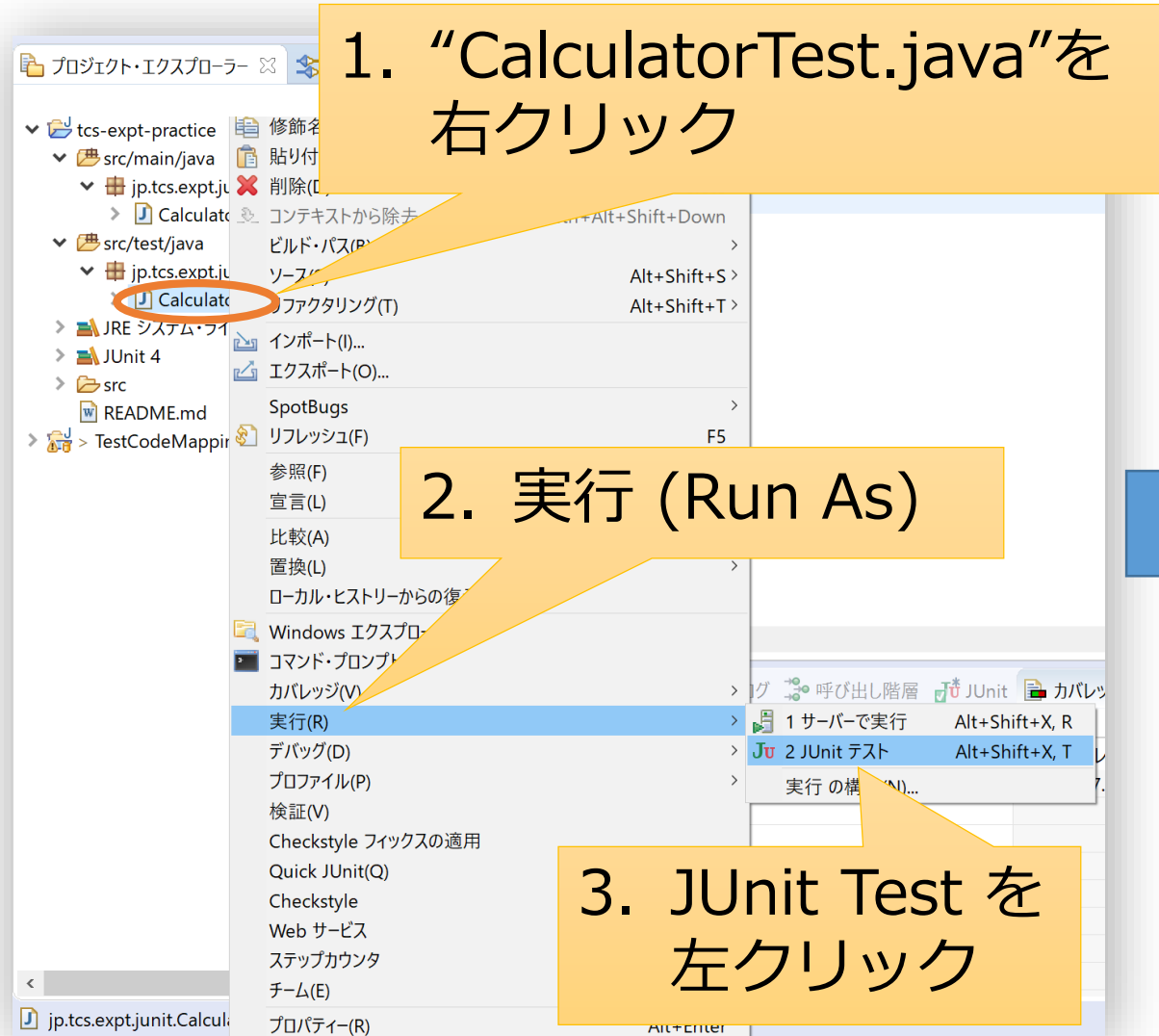
Annotations in the image:

- テスト対象コード** (Test Target Code) points to the `Calculator` class in `Calculator.java`.
- テストコード (未実装)** (Test Code (Not Implemented)) points to the `test` method in `CalculatorTest.java`.
- テストの実行結果** (Test Execution Results) points to the coverage table at the bottom.

テストコードの記述

```
public class CalculatorTest {  
  
    @Test  
    public void test() {  
        Calculator calc = new Calculator(); // テスト対象の準備  
        int result1 = calc.multiply(10, 20);  
        int result2 = calc.multiply(20, 30);  
        assertEquals(200, result1);  
        assertEquals(600, result2);  
    }  
}
```

テストの実行・結果表示



実践練習（10分間）

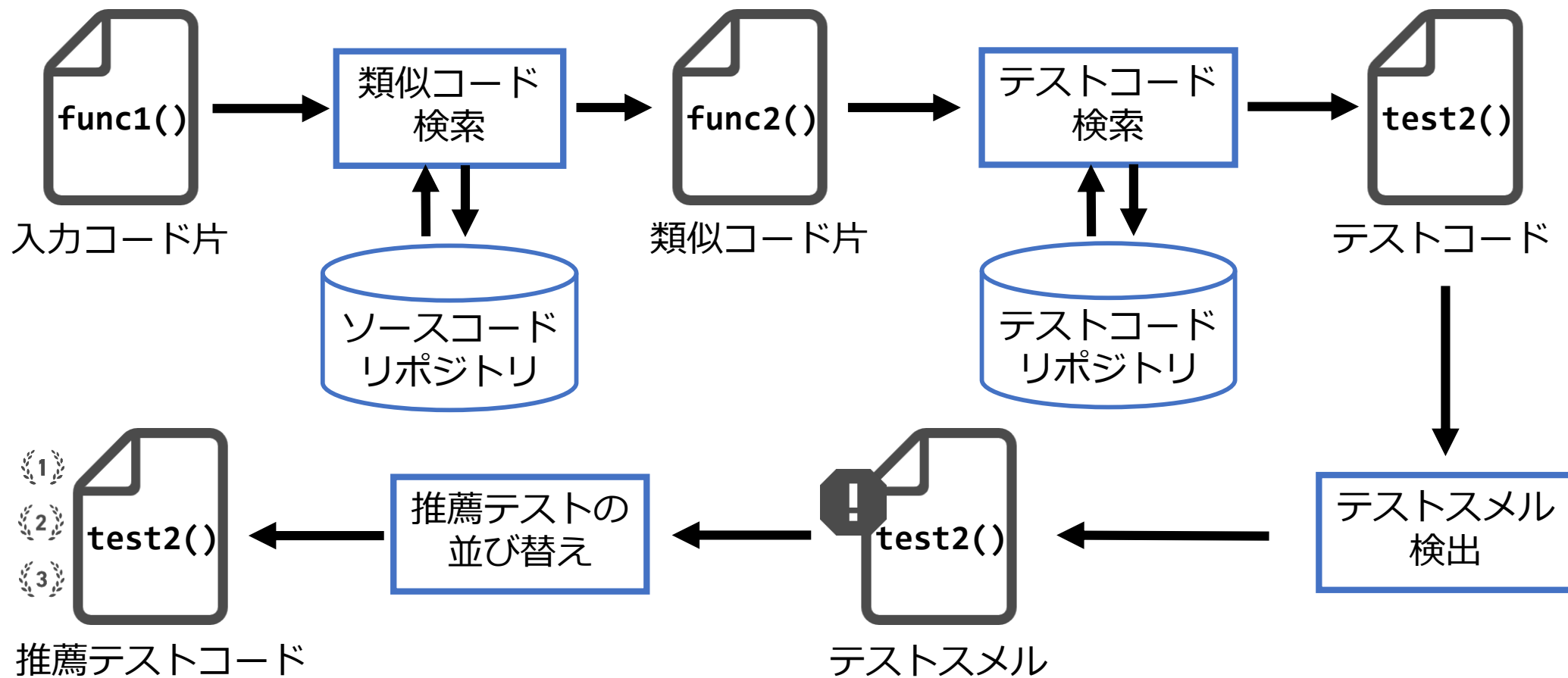
- フォルダ **src/main/java** の下ある “CalcMax.java” に対するテストコードを作成してください
 - “CalcMax.java”の仕様
 - 3つの引数(int 型) a, b, c を入力としての最大値を返す
 - テストコードの練習のため、少し分かりにくいコードにしてある
- ポイント
- ✓ 他の人から見ても読みやすいテストコードを書く
 - ✓ 漏れのない必要十分なテストの量がどれだけか考える

テストコード自動推薦ツール

ツールの概要・使い方


テストコード推薦ツール

- テストコード自動推薦ツールの仕組み



テストコード推薦ツールの使い方1

Test Code Searcher

 Code Fragment

テスト対象のコード片を入力

SEARCH

テストコード推薦ツールの使い方2

Test Code Searcher Report

入力コード

Clone Pairs 1 : 50.0%

類似コード

Input Code	Similarity Code
<pre>! public void setNorthMember(Canvas member) { if (member == null) { ! north.setHeight(0); ! north.setMembers(); } else { member.setWidth("100%"); ! north.setHeight(1); ! north.setMembers(member); } }</pre>	<pre>! public void setSouthMember(Canvas member) { if (member == null) { ! south.setHeight(0); ! south.setMembers(); } else { member.setWidth("100%"); ! south.setHeight(1); ! south.setMembers(member); } }</pre>

テストスメル有り

Test Suite					
Assertion Roulette	Conditional Test Logic	Default Test	Eager Test	Exception Handling	Mystery Guest
Lines 155 - 160 of 360- Innovations_VaadinSmartGWT/VaadinSmartGWT/org.vaadin.smartgwt/src/test/java/org/vaadin/smartgwt/server/layout/BorderLayoutTest.java					
<pre>public void test_resizesSouthHolderToNothingIfMemberIsNotNull() { borderLayout.setSouthMember(new Canvas()); borderLayout.setSouthMember(null); assertEquals(Integer.valueOf(0), getSouthHolder().getAttributeAsInt("height")); }</pre>					

テストコード

評価実験

評価実験の概要

- 1人で3つのコード片に対してテストコードを書いてもらいます
 - 1回目は、何も使わずテストコードを書く(20分)
 - 2回目は、テスト推薦ツールを使って書く(20分)
 - 3回目は、何も使わないまたはツールを利用して書く(20分)
 - ※ 1回目に推薦ツールを使う場合もある
 - アンケート回答
- 調査すること
 - 自身で納得するまでのテストコード作成時間(最大20分)
 - テストコードの品質(テストスメルの有無)

実験問題の割り当て

実験者	 Aさん		 Bさん		 Cさん		 Dさん		 Eさん		 Fさん		 Gさん		 Hさん	
	問題	ツール	問題	ツール	問題	ツール	問題	ツール	問題	ツール	問題	ツール	問題	ツール	問題	ツール
1回目	問題1		問題1	○	問題2		問題2	○	問題3		問題3	○	問題1	○	問題1	
2回目	問題2	○	問題3		問題1	○	問題3		問題1	○	問題2		問題2		問題3	○
3回目	問題3		問題2	○	問題3		問題1	○	問題2		問題1	○	問題3	○	問題2	

事前資料のご確認
ありがとうございました