

JUnit & Eclipse

201260053 Abbos Shomurodov
201260058 채승흠

TEAM 1
200711437 성하진
200511355 정용구
200911436 조성완

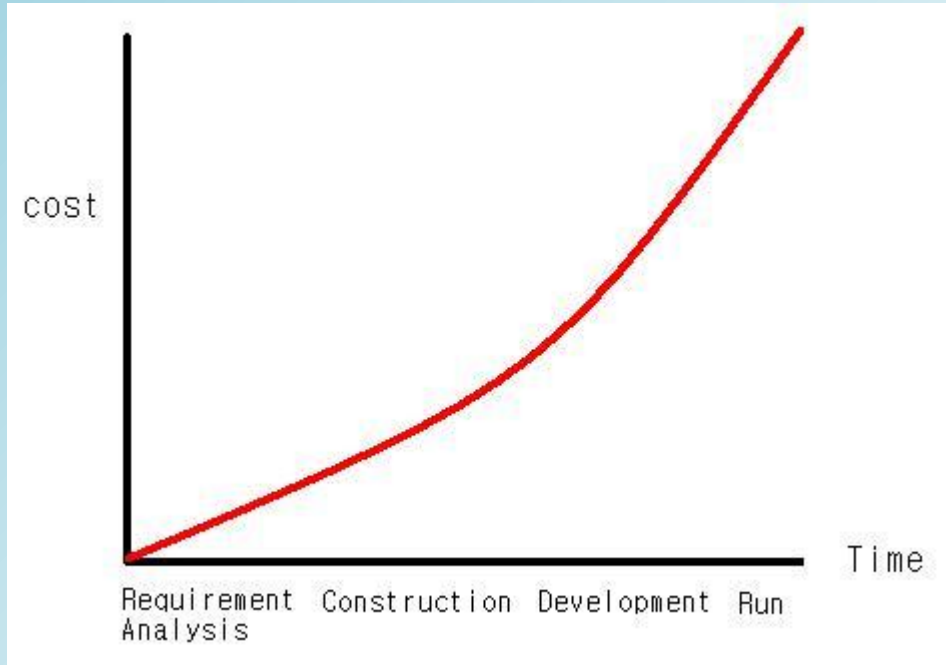
Contents

- 1. Software Testing Theory**
- 2. Eclipse Plug-in**
- 3. JUnit Practice**

Software Testing Theory

- 1.1 Importance of testing
- 1.2 Test Driven Development(TDD)
- 1.3 Extreme Programming(XP)
- 1.4 Unit Test
- 1.5 Structural Test
- 1.6 JUnit

Importance of Testing



- 프로그램 개발 도중, 결함을 발견한 시기가 늦어질수록 수정 비용이 늘어난다.
- 프로그램의 결함 유무를 점검하면서 개발하는, 유동적이며 변화에 기민한 개발 방법론이 필요

As find problems lately, the cost of modification grows.

Importance of Testing(cont.)

- **Timings of software testing**

- During development
- After organize the requirements
- Complete timing of coding



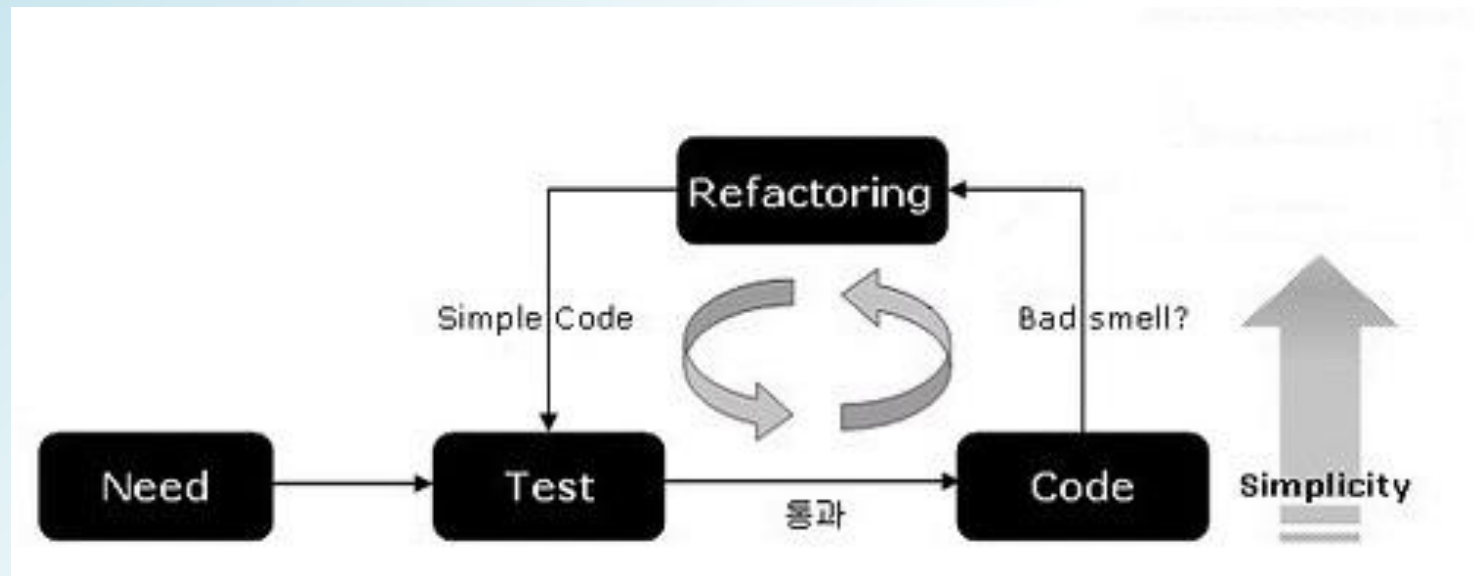
즉, 개발 내내 반복적으로 테스트

Test Driven Development

- Not a test technique, analysis technique
 - 테스트 기법이 아닌 분석, 설계 기술
- Related to the test-first programming concepts of XP
 - 테스트 중심의 빠른 개발 방법인 Extreme Programming과 관련
- Relies on the repetition of a short development cycle
 - 테스트-개발-테스트의 반복

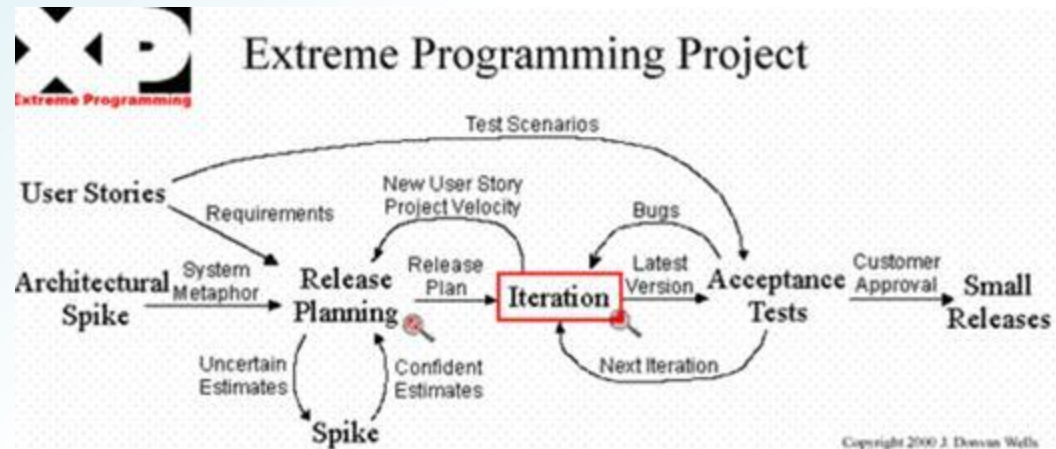
Test Driven Development(cont.)

- 일반적인 개발 과정
 - 디자인 → 개발 → 테스트
- TDD 개발 과정
 - 테스트 코드 작성 → 개발 → 리팩토링



Extreme Programming(XP)

- Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage.
- 불필요한 작업은 최소한으로 줄여, 단기간에 가능한 좋은 제품을 고객에게 제공하는 개발 방법론
- 빠른 개발과 테스트 중심의 프로젝트 진행
- 리팩토링 과정을 통해 적절한 기능의 추가와 보완 가능



Unit Test

- What is Unit

A unit is the smallest testable part of an application. In procedural programming a unit could be an entire module but is more commonly an individual function or procedure. In object-oriented programming a unit is often an entire interface, such as a class, but could be an individual method.

- Application의 가장 작은 단위인 Unit을 테스트(모듈, 객체 등)
- TDD concept에 따라, 개발 중 반복적으로 테스트 코드 작성
- Unit Test Framework : xUnit
- Black Box Test
 - Focus on testing functional requirements

Structural Test

- Structure based test
- tests internal structures or workings of an application, as opposed to its functionality
- Coverage 작성의 대표적인 예
 - Coverage = (수행 성공 대상/총 대상) * 100
- White Box Test
 - Focus on using internal knowledge of the software

xUnit

- XP style unit testing framework. xUnit uses a protocol between a front-end to display test results and a test driver linked into applications - removing the link time dependencies between the application and the graphics, formatting libraries.

➡ Unit Test에 사용하기 위한 Framework

▷ xUnit 종류

Name	JUnit	NUnit	CUnit	CppUnit	PHPUnit
Language	Java	.NET	C	C++	PHP

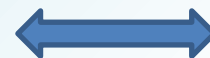
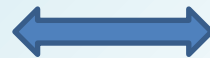
JUnit

- What is JUnit
 - Unit testing framework for the Java programming language.
 - JUnit has been important in the development of TDD, and is one of a family of unit testing frameworks collectively known as xUnit.
- ➔ Java에서 쓰는 xUnit
- Automation of code execution and results after
 - 테스트 코드는 개발자가 작성
 - XML과 HTML 로 출력되는 결과를 다른 포맷으로 변환 및 저장 가능
- Eclipse에 내장(www.junit.org에서 download 가능)

JUnit 3 & JUnit 4

JUnit 3

- Method는 Test로 시작
- TestCase를 상속받음
- setUp(), tearDown()로 Fixture 일일이 지정



JUnit 4

- @Test로 Method 지정가능
- 독립 클래스로 작성
- @Before, @After로 직관적인 표현

JUnit 4

- Annotation

- @Before: 테스트 전에 실행할 메소드 지정
- @After: 테스트 후에 실행할 메소드 지정
- @BeforeClass: 테스트 전에 한 번만 실행할 메소드 지정
- @AfterClass: 테스트 후에 한 번만 실행할 메소드 지정
- @Test(timeout=100): 제한시간 설정
- @Ignore: 테스트 제외

- 직관적인 테스트 코드 작성 가능

@Test

```
public void verifyGoodZipCode() throws Exception{  
    Matcher mtcher = this.pattern.matcher(phrase);  
    boolean isValid = mtcher.matches();  
    assertEquals("Pattern did not validate zip code", isValid, match);  
}
```

JUnit 4(cont.)

- JUnit Methods

- assertEquals - 같은지 비교
- assertNull - null값을 리턴하는지 비교
- assertNotNull - 인자로 넘겨받은 객체가 null인지 판정하고 반대인경우 실패로 처리
- assertEquals - expected 와 actual이 같은 객체를 참조하는지 판정하고 그렇지 않다면 실패로 처리
- assertEquals - expected 와 actual이 서로 다른 객체를 참조하는지 판정하고, 만약 같은 객체를 참조한다면 실패로 처리
- assertTrue - boolean 조건이 참인지 판정
- assertFalse - boolean 조건이 거짓인지 판정
- fail - 테스트를 바로 실패 처리

Eclipse Plug-in

1.1 Ant

1.2 Subclipse

1.3 TPTP

Ant

- 빌드 자동화 툴
 - Eclipse에 내장
 - XML형태의 Ant문서 작성(*.ant)
 - 컴파일, 폴더 분류, FTP업로드, 파일 압축 등의 작업 일괄 처리 가능
- FTP사용
 - Apache에서 ftp라이브러리 download
 - 해당 라이브러리를 ant Class path에 등록
 - Xml파일에 ftp 정보 작성

Subclipse

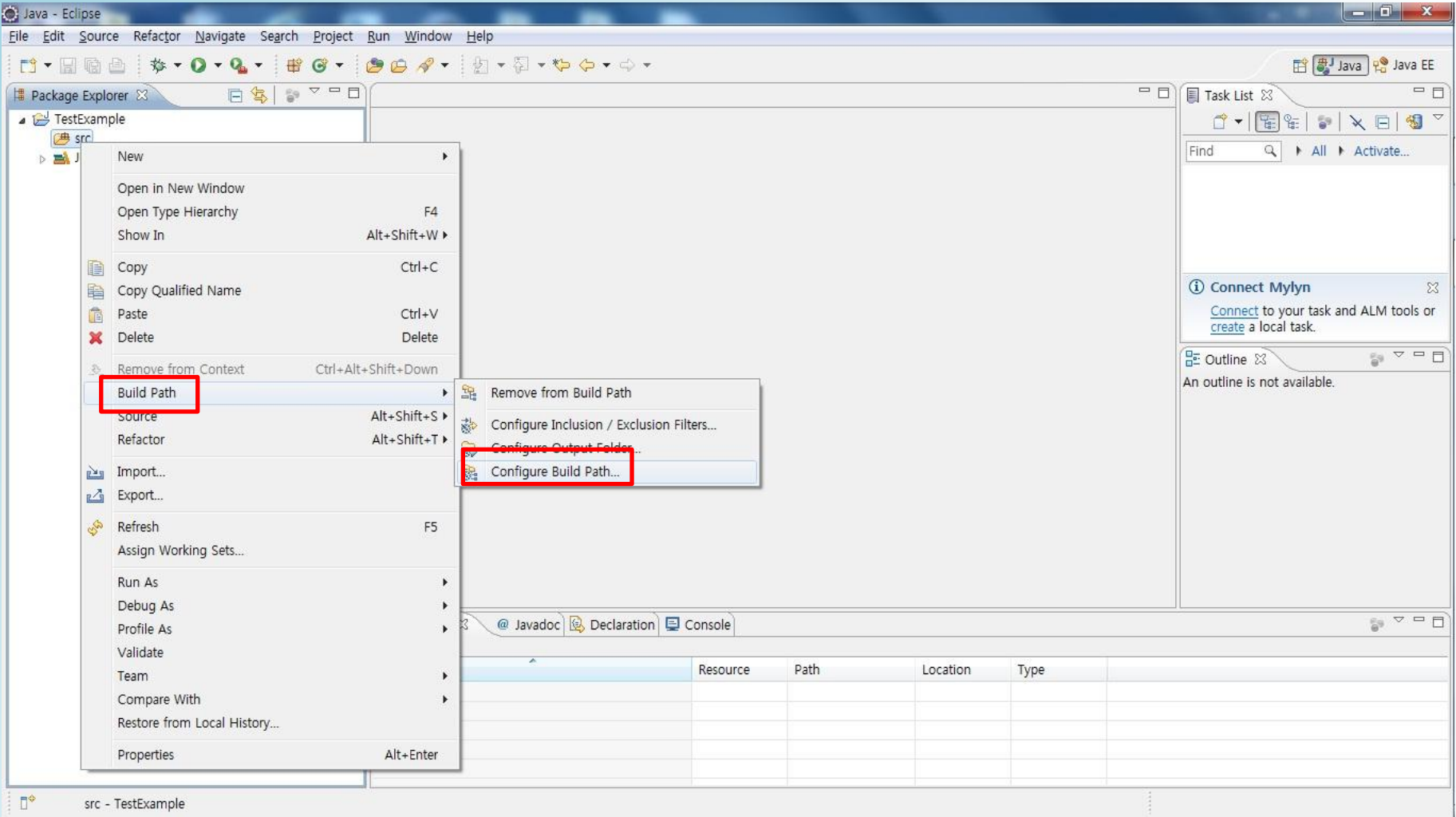
- SVN(Subversion)
 - Software versioning and revision control system
 - 버전 관리 시스템
- Eclipse IDE에서 SVN을 사용하기 위한 Eclipse Plug-in

TPTP

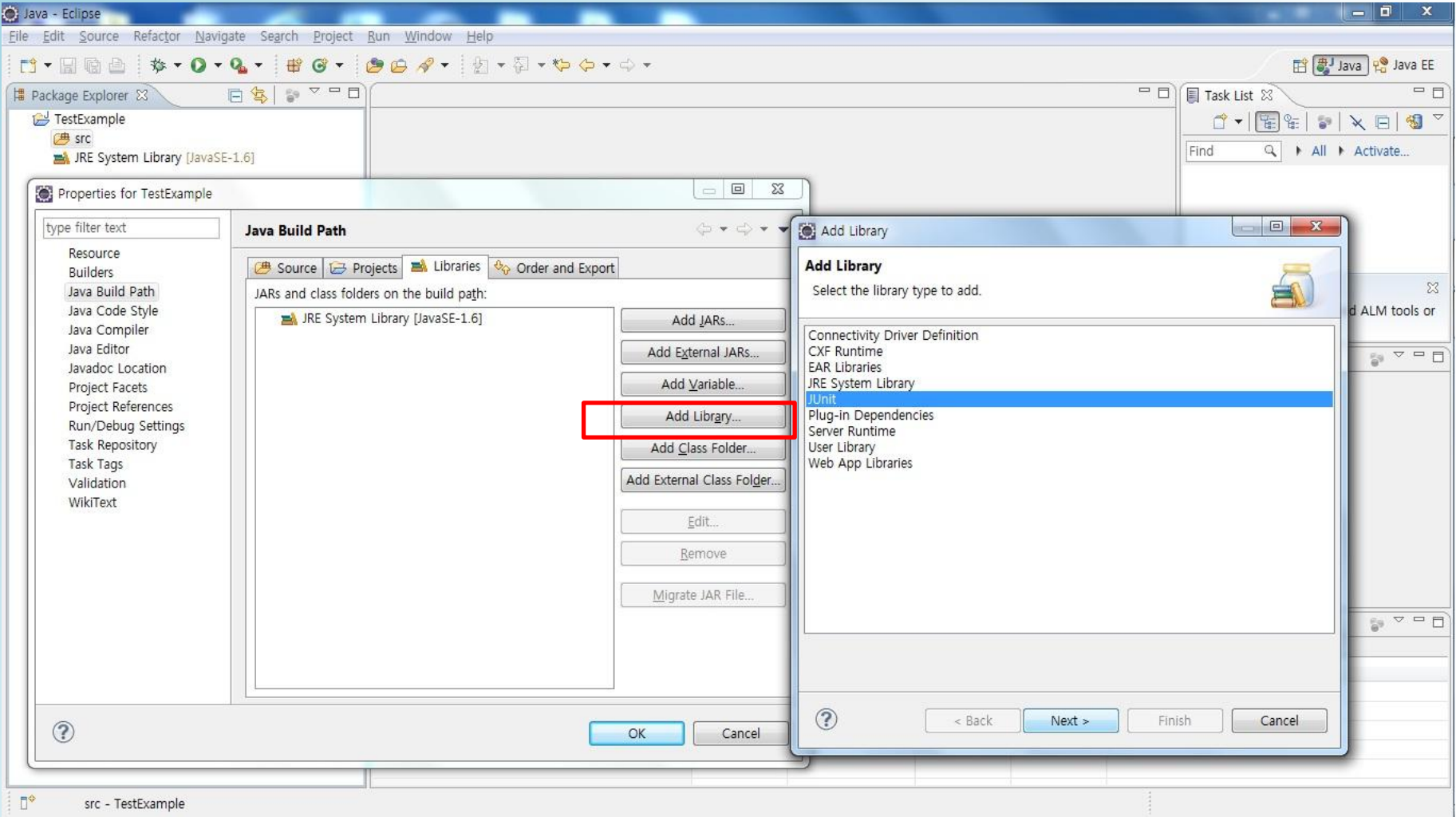
- Test & Performance Tools Platform
 - Eclipse에서 진행중인 프로젝트 및 성능 관련 툴을 위한 플랫폼
- 모니터링, 테스트 자동화, 프로파일 등의 기능
- Garbage Collector로 제거되지 않는 객체를 수정 가능
- 높은 메모리 점유율 및 지나친 수행시간을 가지는 소스를 찾아 수정 가능

JUnit Practice

JUnit Set-up



JUnit Set-up(cont.)



Coding(class 작성)

The screenshot displays the Eclipse IDE interface. The main editor window shows the source code for `Calculator.java` in the `example` package. The code defines a `Calculator` class with four static methods: `add`, `sub`, `mul`, and `div`. The `add` method is currently selected and highlighted in blue.

```
package example;

public class Calculator {

    public static int add(int a, int b) {
        return a+b;
    }

    public static int sub(int a, int b) {
        return a-b;
    }

    public static int mul(int a, int b) {
        return a*b;
    }

    public static int div(int a, int b) {
        return a/b;
    }

}
```

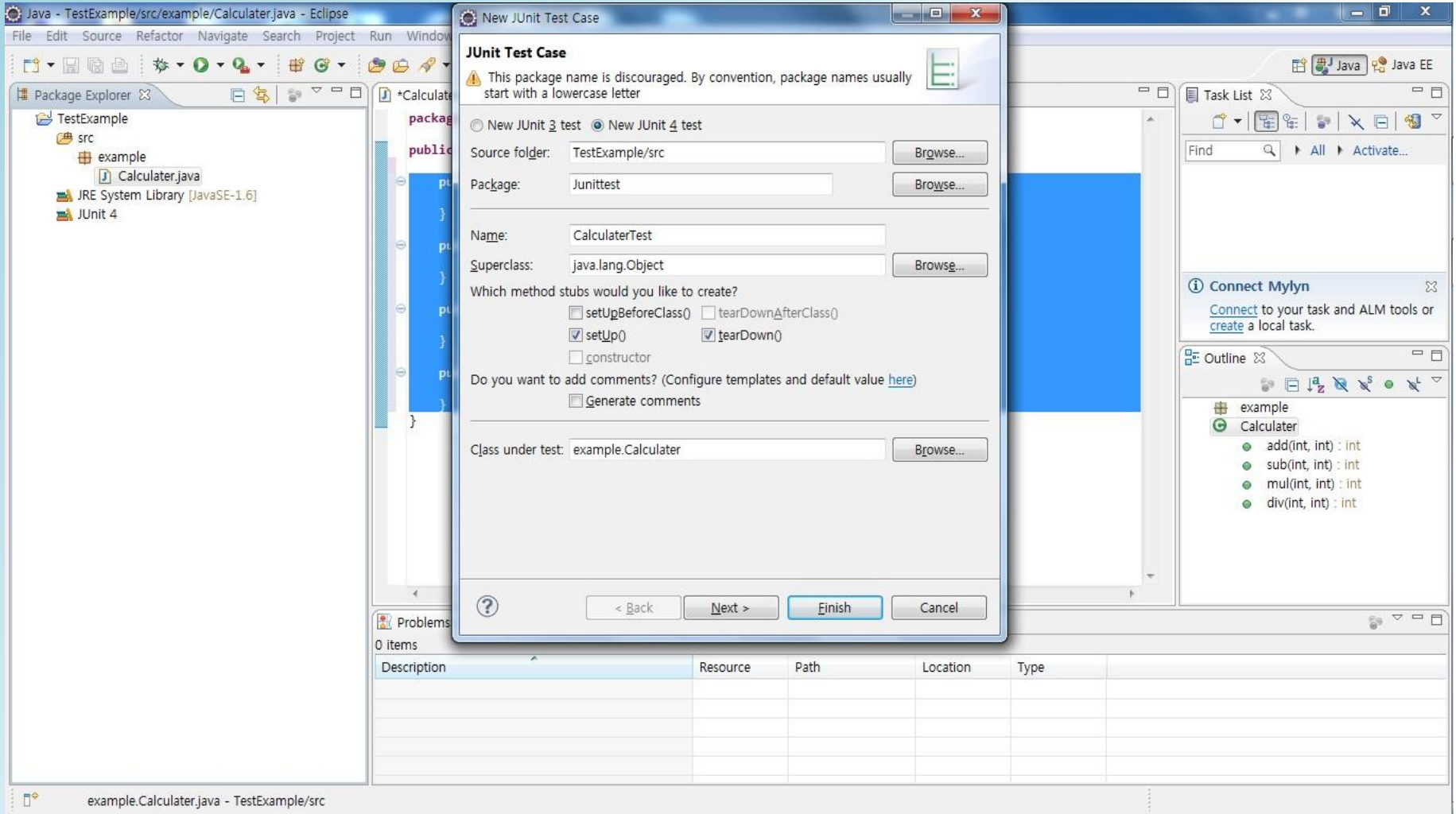
The Package Explorer on the left shows the project structure: `TestExample` contains `src`, `example` (with `Calculator.java`), `JUnittest`, `JRE System Library [JavaSE-1.6]`, and `JUnit 4`.

The Outline view on the right shows the class hierarchy: `example` contains `Calculator`, which has four methods: `add(int, int) : int`, `sub(int, int) : int`, `mul(int, int) : int`, and `div(int, int) : int`.

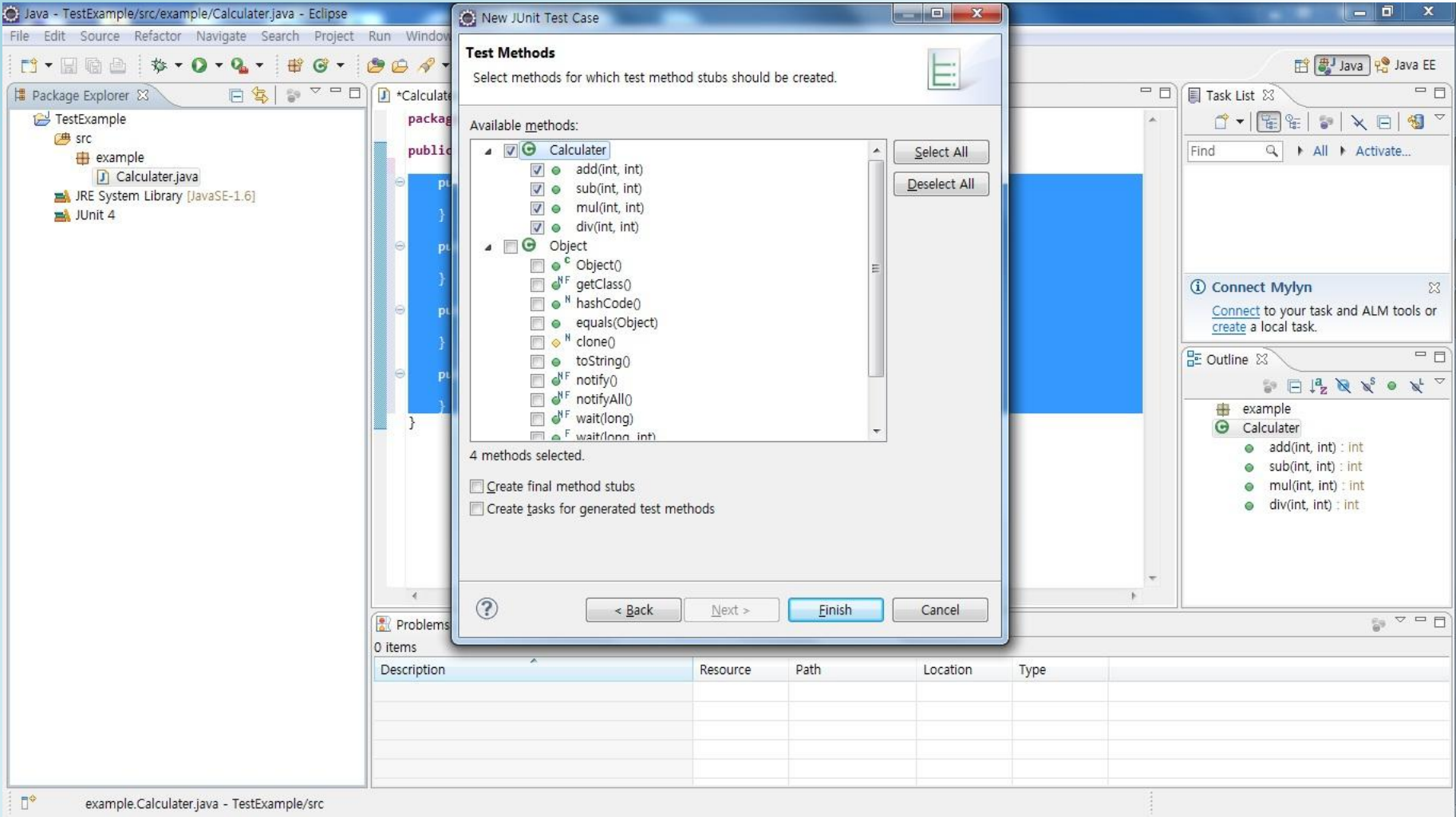
The Console view at the bottom shows 0 items.

The status bar at the bottom indicates the current state: Writable, Smart Insert, and 5 : 1.

JUnit Test Case 작성(cont.)



JUnit Test Case 작성(cont.)



JUnit Test Case 작성(cont.)

The screenshot displays the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with 'TestExample' containing 'src', 'example', 'Junittest', and 'CalculatorTest.java'.
- Editor:** Displays the source code for 'CalculatorTest.java' in the 'Junittest' package. The code includes imports for JUnit and the 'Calculator' class, and defines five test methods: 'setUp()', 'tearDown()', 'testAdd()', 'testSub()', 'testMul()', and 'testDiv()'. Each test method currently contains a 'fail("Not yet implemented");' statement.
- Task List:** Shows a search bar and 'Connect Mylyn' options.
- Outline:** Provides a hierarchical view of the code elements, including imports and the 'CalculatorTest' class with its methods.
- Bottom Panel:** Includes 'Problems', 'Javadoc', 'Declaration', and 'Console' tabs.
- Status Bar:** Shows 'Writable', 'Smart Insert', and '40 : 2'.

```
package Junittest;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import example.Calculator;

public class CalculatorTest {

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testAdd() {
        fail("Not yet implemented");
    }

    @Test
    public void testSub() {
        fail("Not yet implemented");
    }

    @Test
    public void testMul() {
        fail("Not yet implemented");
    }

    @Test
    public void testDiv() {
        fail("Not yet implemented");
    }
}
```

JUnit Test Case 작성(cont.)

The screenshot displays the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with 'TestExample' and 'JUnit' packages.
- Code Editor:** Contains the implementation of `CalculatorTest.java`. The test methods `testAdd()`, `testSub()`, `testMul()`, and `testDiv()` are highlighted with red boxes. The code includes annotations for `@Before`, `@After`, and `@Test`.
- Task List:** Shows a search bar and a 'Connect Mylyn' button.
- Outline:** Lists the methods of the `CalculatorTest` class: `setUp() : void`, `tearDown() : void`, `testAdd() : void`, `testSub() : void`, `testMul() : void`, and `testDiv() : void`.
- Problems/Console:** Shows '0 items' in the Problems view and tabs for '@ Javadoc', 'Declaration', and 'Console'.

```
public class CalculatorTest {  
    @Before  
    public void setUp() throws Exception {  
    }  
  
    @After  
    public void tearDown() throws Exception {  
    }  
  
    @Test  
    public void testAdd() {  
        assertEquals(3, Calculator.add(1,2));  
    }  
  
    @Test  
    public void testSub() {  
        assertEquals(3, Calculator.sub(5,2));  
    }  
  
    @Test  
    public void testMul() {  
        assertEquals(6, Calculator.mul(3,2));  
    }  
  
    @Test  
    public void testDiv() {  
        assertEquals(3, Calculator.div(6,2));  
    }  
}
```


Testing(cont.)

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with 'JUnittest' and 'CalculatorTest' under the 'src' folder.
- JUnit Runner:** Displays the test results: 'Finished after 0.015 seconds', 'Runs: 4/4', 'Errors: 0', and 'Failures: 0'. A list of tests is shown: 'testAdd (0.000 s)', 'testSub (0.000 s)', 'testMul (0.000 s)', and 'testDiv (0.000 s)'. A 'Failure Trace' section is also visible but empty.
- Code Editor:** Shows the source code for 'CalculatorTest.java':

```
public class CalculatorTest {  
  
    @Before  
    public void setUp() throws Exception {  
    }  
  
    @After  
    public void tearDown() throws Exception {  
    }  
  
    @Test  
    public void testAdd() {  
        assertEquals(3, Calculator.add(1,2));  
    }  
  
    @Test  
    public void testSub() {  
        assertEquals(3, Calculator.sub(5,2));  
    }  
  
    @Test  
    public void testMul() {  
        assertEquals(6, Calculator.mul(3,2));  
    }  
  
    @Test  
    public void testDiv() {  
        assertEquals(3, Calculator.div(6,2));  
    }  
}
```
- Task List:** Contains a 'Connect Mylyn' notification.
- Outline:** Shows the class hierarchy: 'org.junit.Before', 'org.junit.Test', 'example.Calculator', and 'CalculatorTest' with its methods: 'setUp() : void', 'tearDown() : void', 'testAdd() : void', 'testSub() : void', 'testMul() : void', and 'testDiv() : void'.
- Problems:** Shows '0 items'.
- Console:** Shows a table with columns: 'Description', 'Resource', 'Path', 'Location', and 'Type'.

Testing(cont.)

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with 'TestExample' containing 'src', 'example', 'Junittest', and 'CalculatorTest.java'.
- Main Editor:** Displays the code for 'Calculator.java'. The 'add' method is highlighted with a red box:

```
package example;

public class Calculator {

    public static int add(int a, int b) {
        return a+b;
    }

    public static int sub(int a, int b) {
        return a-b;
    }

    public static int mul(int a, int b) {
        return a*b;
    }

    public static int div(int a, int b) {
        return a/b;
    }
}
```
- Task List:** Contains a search bar and a 'Connect Mylyn' button.
- Outline:** Shows the class hierarchy: 'example' > 'Calculator' with methods: 'add(int, int) : int', 'sub(int, int) : int', 'mul(int, int) : int', and 'div(int, int) : int'.
- Problems:** Shows '0 items' in a table with columns: Description, Resource, Path, Location, Type.

example.Calculator.java - TestExample/src

Testing(cont.)

The screenshot displays the Eclipse IDE interface during a JUnit test run. The Package Explorer on the left shows the test results for `Junittest.CalculatorTest`, indicating that all 4 tests passed successfully. The main editor shows the source code for `CalculatorTest.java`, which includes methods for `setUp`, `tearDown`, and four test methods: `testAdd`, `testSub`, `testMul`, and `testDiv`. The `testAdd` method is highlighted in the Failure Trace, showing an `AssertionError` where the expected value was `<3>` but the actual value was `<-1>`. The Outline view on the right shows the class structure, including the `CalculatorTest` class and its methods.

```
public class CalculatorTest {  
    @Before  
    public void setUp() throws Exception {  
    }  
    @After  
    public void tearDown() throws Exception {  
    }  
    @Test  
    public void testAdd() {  
        assertEquals(3, Calculator.add(1,2));  
    }  
    @Test  
    public void testSub() {  
        assertEquals(3, Calculator.sub(5,2));  
    }  
    @Test  
    public void testMul() {  
        assertEquals(6, Calculator.mul(3,2));  
    }  
    @Test  
    public void testDiv() {  
        assertEquals(3, Calculator.div(6,2));  
    }  
}
```

Description	Resource	Path	Location	Type
0 items				

References

<자바 개발자도 쉽고 즐겁게 배우는 테스팅 이야기> 이상민

<http://blog.naver.com/1jongrak?Redirect=Log&logNo=50132272876>

<http://searchstory.tistory.com/268>

<http://xprogramming.com/book/whatisxp/>

<http://blog.naver.com/softgear?Redirect=Log&logNo=100002991166>

<http://en.wikipedia.org/wiki/>

<http://www.extremeprogramming.org/>

<http://www.junit.org/>

<http://blog.naver.com/nachaos?Redirect=Log&logNo=110030350613>

<http://www.ibm.com/developerworks/kr/library/tutorial/j-junit4/>

<http://www.eclipse.org>