

1. 도구 개요

JUnit

| | | | |
|-------------|--|--------|-------------------------------------|
| 소 개 | JUnit은 Java기반 테스트를 위한 프레임워크로, 단위모듈(ex: Method)이 정확히 구현되었는지를 확인할 수 있는 도구 | | |
| 주요기능 | Test case 생성 및 실행, 오류추적 | | |
| 카테고리 | Testing | 세부카테고리 | 테스트 설계 및 실행 |
| 커버리지 | Testing | 도구난이도 | 하 |
| 라이선스형태 / 비용 | Common Public License 1.0 / 무료 | 사전설치도구 | 자바 기반 IDE(Eclipse, NetBeans 등), JDK |
| 운영체제 | Windows, Linux, Mac OS X, UNIX | 도구버전 | 4.10 (2010. 10) |
| 특징 | <ul style="list-style-type: none"> • 메소드와 같은 단위 모듈 별 테스트를 가능케 함으로써 코드 품질을 보장 • 정확한 단위 테스트를 가능케 함으로써 통합 테스트 시 회귀결함(모듈통합에 의해 발생하는 결함)을 줄임 • 다른 모듈에 의존하지 않고, 원하는 모듈만 임의의 순서대로 수행할 수 있다 • JFeature(요구사항개발도구)와 통합되어 요구사항의 정확한 구현 비율을 알 수 있다 | | |
| 적용회사 / 프로젝트 | Samsung SDS Anyframe Java EE 프레임워크 내 단위 테스트 도구로 사용 | | |
| 관련 도구 | JCoverage , CppUnit, Continuous Testing Plugin, httpunit, TestLink, Jdepend | | |
| 제작사 | - | | |
| 공식 홈페이지 | http://www.junit.org | | |

2. 기능 요약

JUnit

Java기반 테스트를 위한 프레임워크로, 단위모듈(ex: Method)이 정확히 구현되었는지를 확인할 수 있는 도구

| 주요기능 | 지원내용 |
|------------|---------------|
| 테스트 범위 | 단위테스트 |
| TDD 환경 지원 | 지원 |
| 코드지원 | 지원 |
| Test Suite | 지원 |
| UI Report | 지원 |
| Code Trace | 지원, 실시간 추적 가능 |

3. 도구 실행 환경

JUnit

JAVA를 지원하는 IDE상에서 설치 및 구현이 가능

- 다양한 OS를 지원
 - Windows : Windows XP / Windows 7 (32, 64-bit 모두 지원)
 - Linux : 32, 64-bit 지원
 - Mac OS X : 32, 64-bit 지원
 - UNIX : 32, 64-bit 지원
- JDK, 자바 기반 도구(IDE 등)이 필요
 - 코드 및 플러그인 형태, 도구에 포함되어 있는 형태로 제공

JUnit

Java IDE (Eclipse, NetBeans 등)

JDK (Java development kit)

Windows / Linux / Mac OS / UNIX

4. 도구 설치 방법

세부 목차

JUnit

4.1 JUnit 다운받기

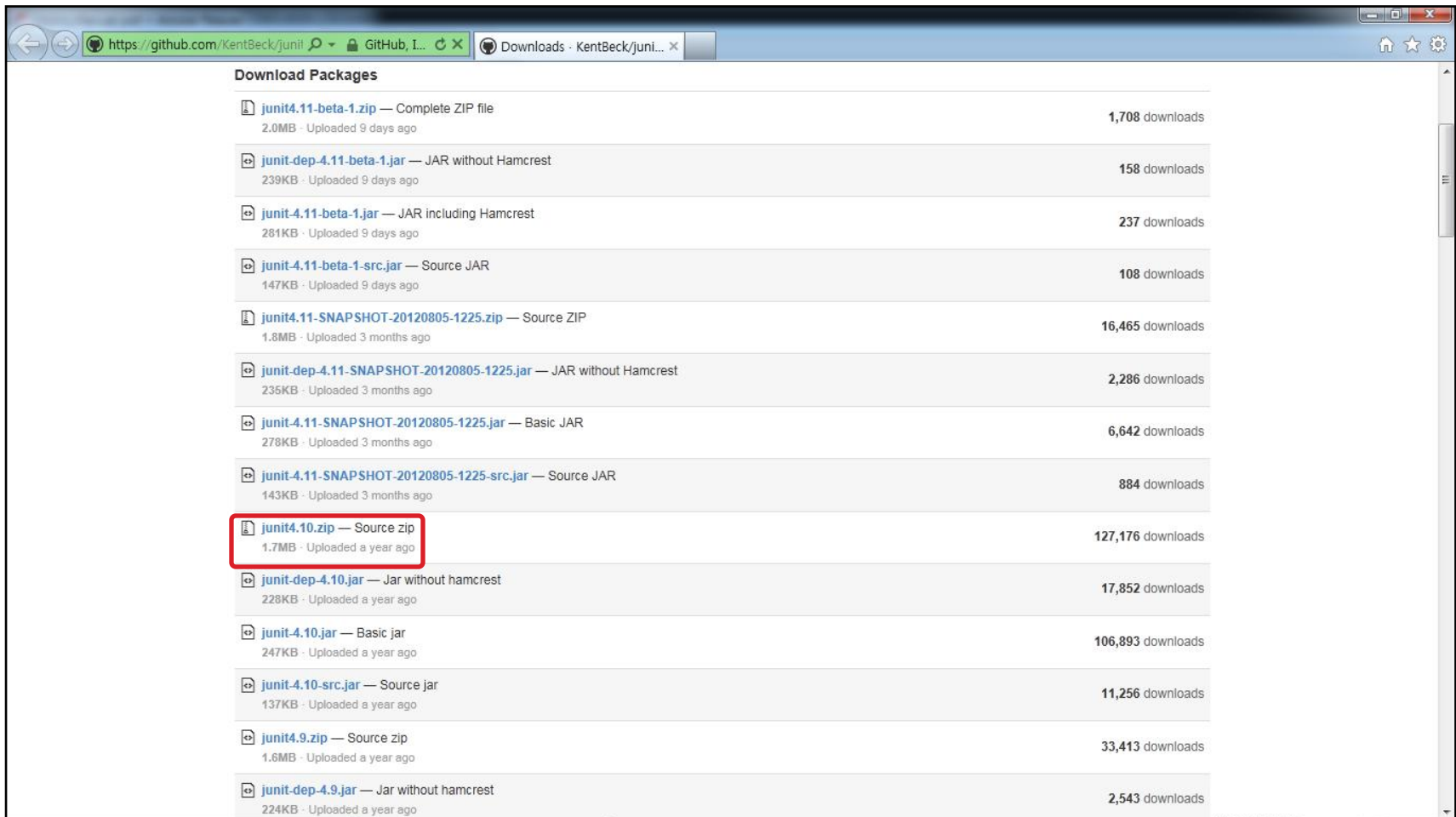
4.2 JUnit 설치하기

4. 도구 설치 방법

4.1 JUnit 다운받기

JUnit

- <https://github.com/KentBeck/junit/downloads>에서 JUnit을 다운받기
 - 대부분의 JAVA기반 IDE는 JUnit이 탑재(JUnit 미 탑재, 업데이트, 베타버전을 사용시 주로 다운)

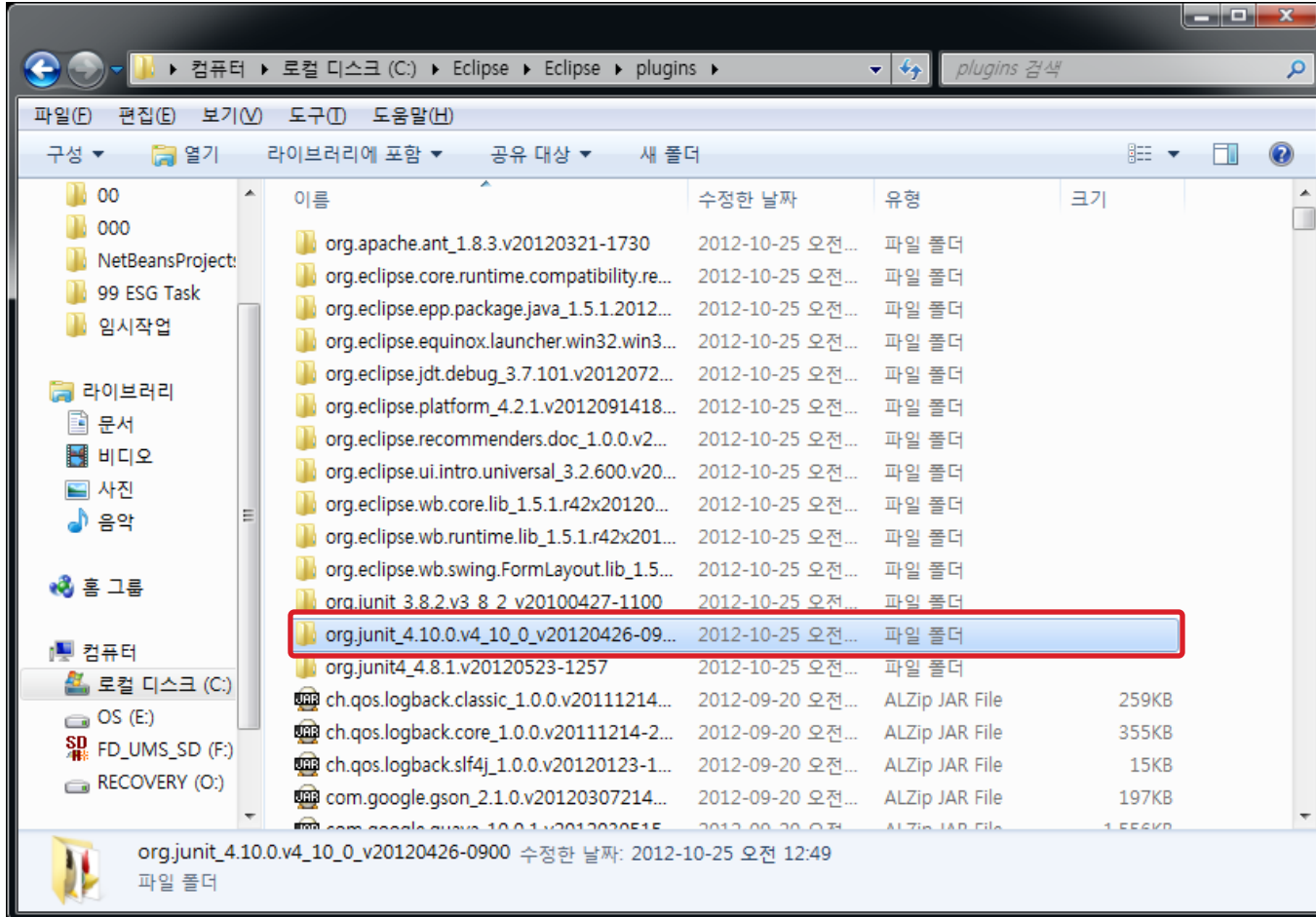


4. 도구 설치 방법

4.2 Jnit 설치하기

JUnit

- 압축을 풀고, junit4.10을 Eclipse가 설치된 폴더 내의 plugins 폴더에 복사
 - 본 매뉴얼에서는 Eclipse기준으로 설명



5. 도구 기능 소개

세부목차

JUnit

- 5.1 주요 기능
- 5.2 예제 소개
- 5.3 Test Case 작성하기
- 5.4 Test Suite 작성하기
- 5.5 테스트 실행 및 결과
- 5.6 테스트 실행을 위한 메뉴 소개

5. 도구 기능 소개

5.1 주요 기능

JUnit

- 테스트하고자 하는 메소드에 대해 Test Case를 만들 수 있다
- 일부의 특정 테스트 메소드를 실행하지 않거나 특정 테스트 메소드만 실행하고 싶을 때, 혹은 테스트 클래스를 한데 묶어서 실행하고 싶은 경우 Test Suite를 사용
- 단위테스트를 실행하고 테스트 결과를 빠르게 확인할 수 있다

```
import junit.framework.TestCase;

public class TrafficLightModelTest extends TestCase {

    public void testNewTrafficLight() {
        TrafficLightmodel a = new TrafficLightmodel();
        assertLights("R",a);
    }
    private void assertLights(String expected, TrafficLightmodel a){
        String actual = "";
        if(a.getRed())
            actual += "R";
        if(a.getYellow())
            actual += "Y";
        if(a.getGreen())
            actual += "G";
        assertEquals("Lights", actual, expected);
    }
}
```

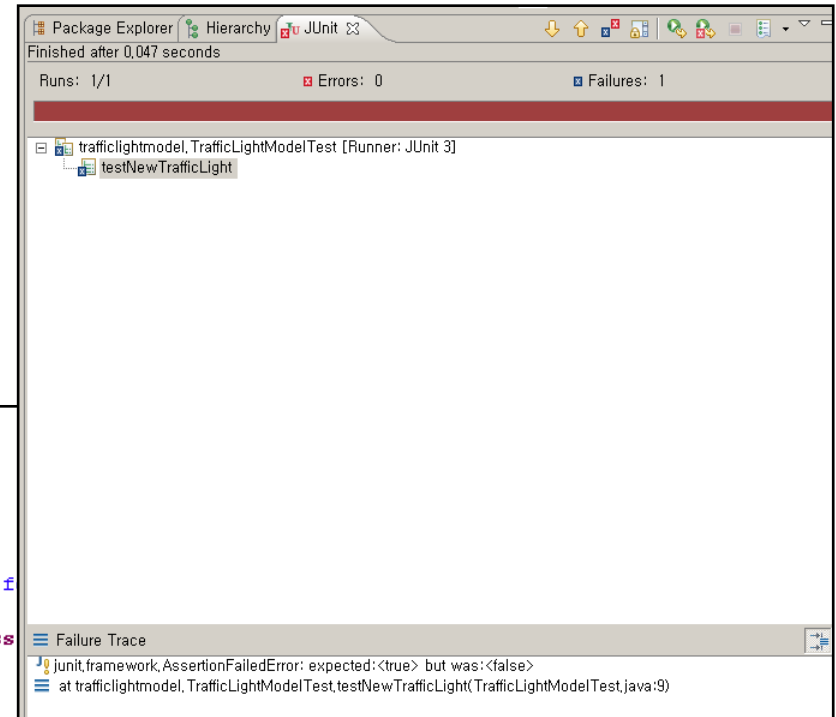
[Test Case 작성화면]

```
import junit.framework.Test;

public class AllTests {

    public static Test suite() {
        TestSuite suite = new TestSuite("Test f
        //$JUnit-BEGIN$
        suite.addTestSuite(CalculatorTest.class
        //$JUnit-END$
        return suite;
    }
}
```

[Test Shite 작성 화면]



[테스트 실행 및 결과 화면]

5. 도구 기능 소개

5.2 예제 소개

JUnit

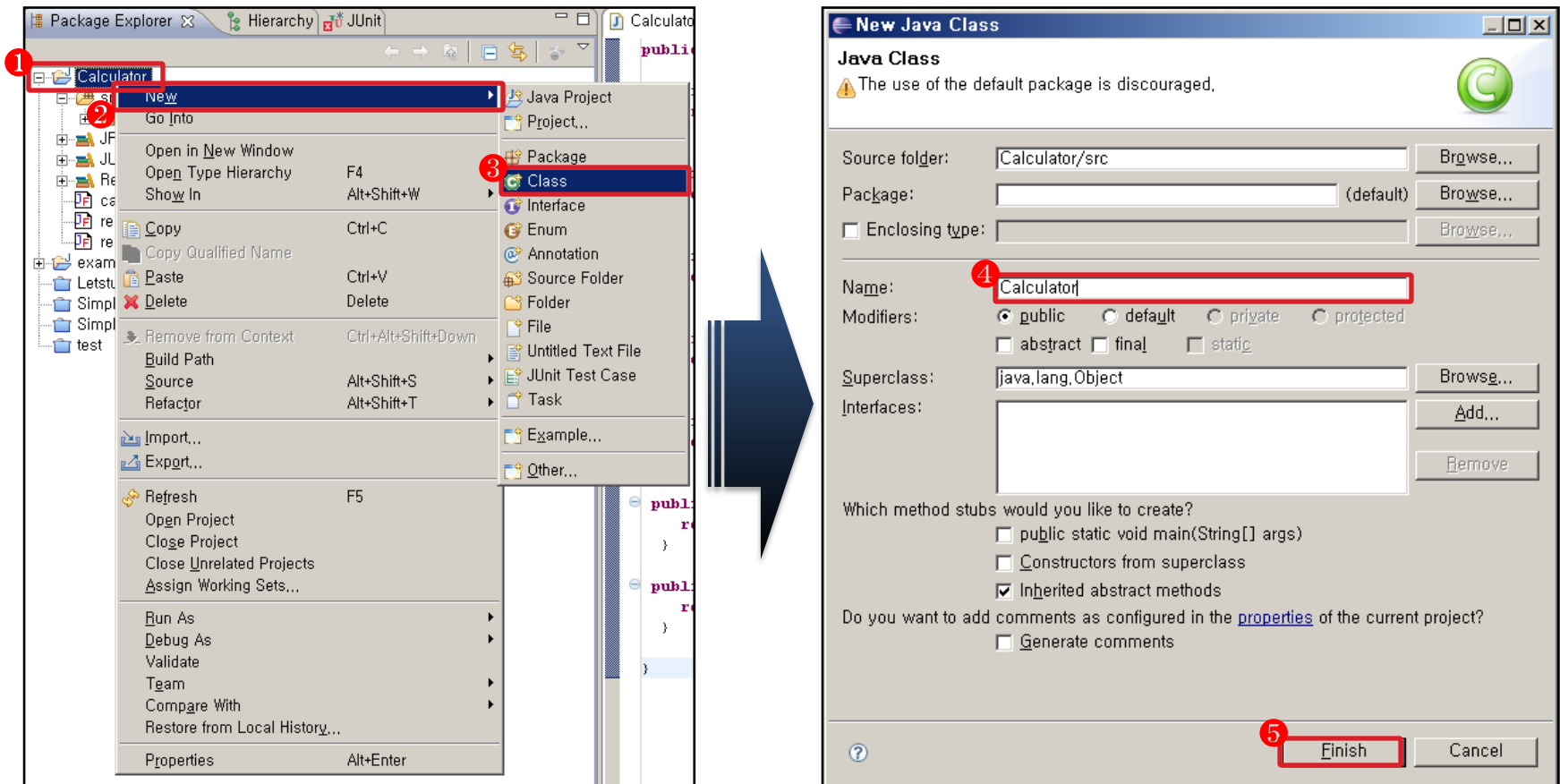
- JUnit의 기능을 소개하기 위해 계산기 프로젝트 예제를 사용
- 계산기는 다음과 같은 기능을 가지고 있다
 - sum : 두수의 더하기 연산을 수행
 - subtract : 두수의 빼기 연산을 수행
 - multiply : 두수의 곱하기 연산을 수행
 - divide : 두수의 나누기 연산을 수행
 - ceil : 지정된 숫자를 올림 하는 연산을 수행 (ex - 3.2의 ceil연산 : 4)
 - floor : 지정된 숫자보다 작거나 같은 가장 작은 정수를 double형태로 반환하는 연산을 수행 (ex - 1.2의 floor연산 : 1, 1.7의floor연산 : 1)
 - abs : 지정된 숫자의 절대값을 계산하고 반환하는 연산을 수행

5. 도구 기능 소개

5.3 Test Case 작성하기(1/8)

JUnit

- 테스트하고자 하는 클래스를 생성 : Calculator
 - 프로젝트선택 → 마우스 우 클릭 메뉴의 New → Class → Class이름 설정(Calculator) → Finish



5. 도구 기능 소개

5.3 Test Case 작성하기(2/8)

JUnit

- Calculator 클래스의 코드를 작성합니다
 - 계산기의 주요 기능을 구현

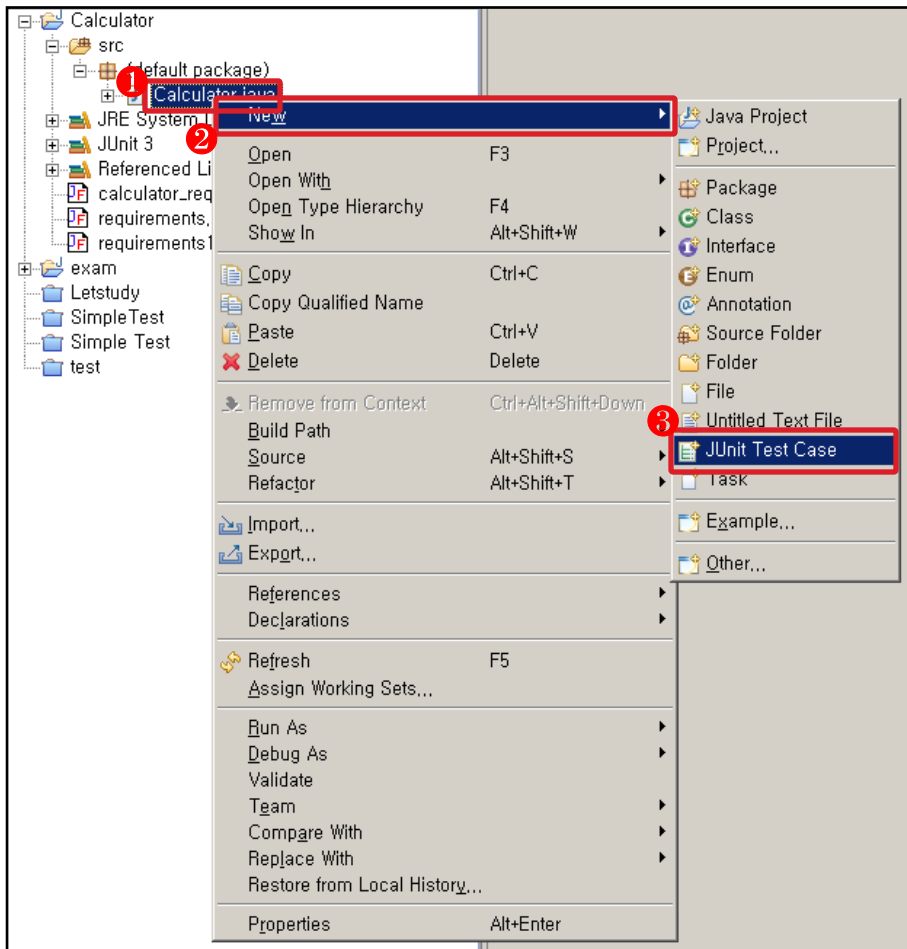
```
public class Calculator {  
  
    public static double sum(double num1, double num2) {  
        return num1 + num2;  
    }  
  
    public static double subtract(double num1, double num2) {  
        return num1 - num2;  
    }  
  
    public static double multiply(double num1, double num2) {  
        return num1 * num2;  
    }  
  
    public static double divide(double num1, double num2) {  
        return num1 / num2;  
    }  
  
    public static double ceil(double num1) {  
        return Math.ceil(num1);  
    }  
  
    public static double floor(double num1) {  
        return Math.floor(num1);  
    }  
  
    public static double abs(double num1) {  
        return Math.abs(num1);  
    }  
}
```

5. 도구 기능 소개

5.3 Test Case 작성하기(3/8)

JUnit

- 테스트하고자 하는 클래스에 대해 Test Case를 생성
 - 테스트를 실행할 클래스 선택 → 마우스 우 클릭 메뉴의 New-> JUnit Test Case



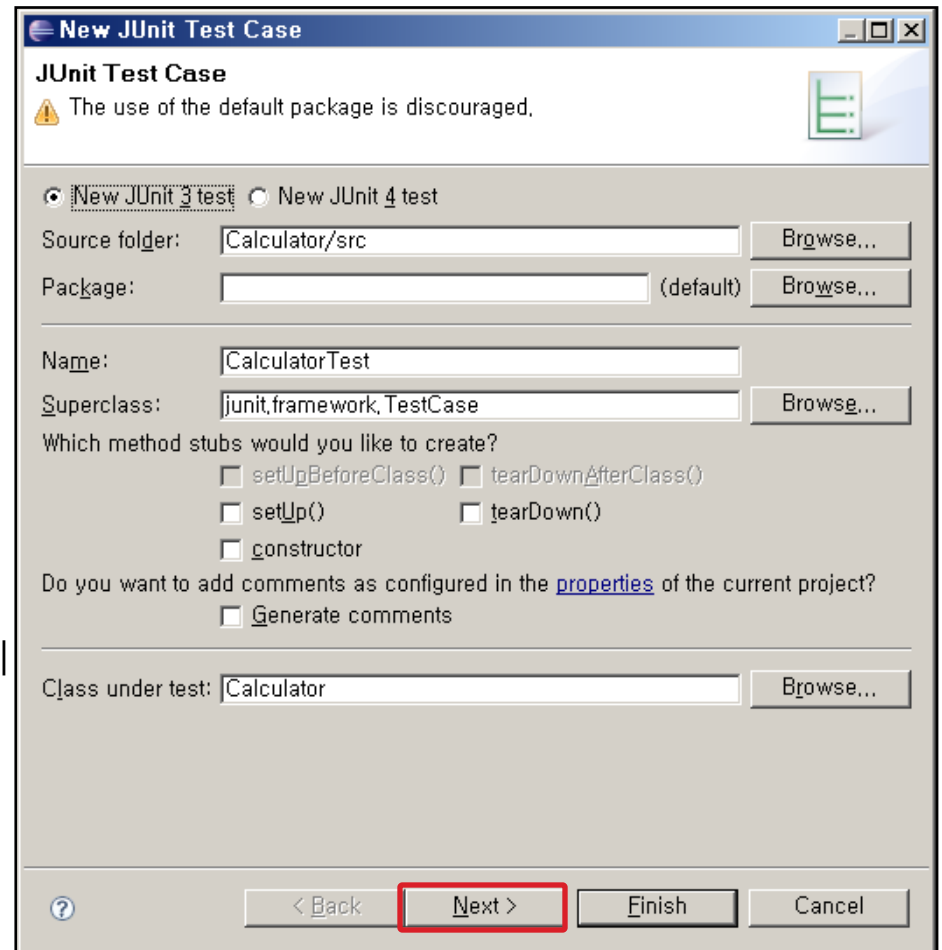
5. 도구 기능 소개

5.3 Test Case 작성하기(4/8)

JUnit

- Source folder, Package, Name, Superclass등을 그대로 두고 NEXT

- Source folder
 - > Test Case 클래스파일이 저장되는 폴더
- Package
 - > Java의 패키지를 의미
- Name
 - > Test Case 클래스의 이름을 의미
(기본값 : 테스트대상클래스명 + Test)
- Superclass
 - > Test Case 클래스가 상속받을 클래스를 의미
(기본값 : junit.framework.TestCase)

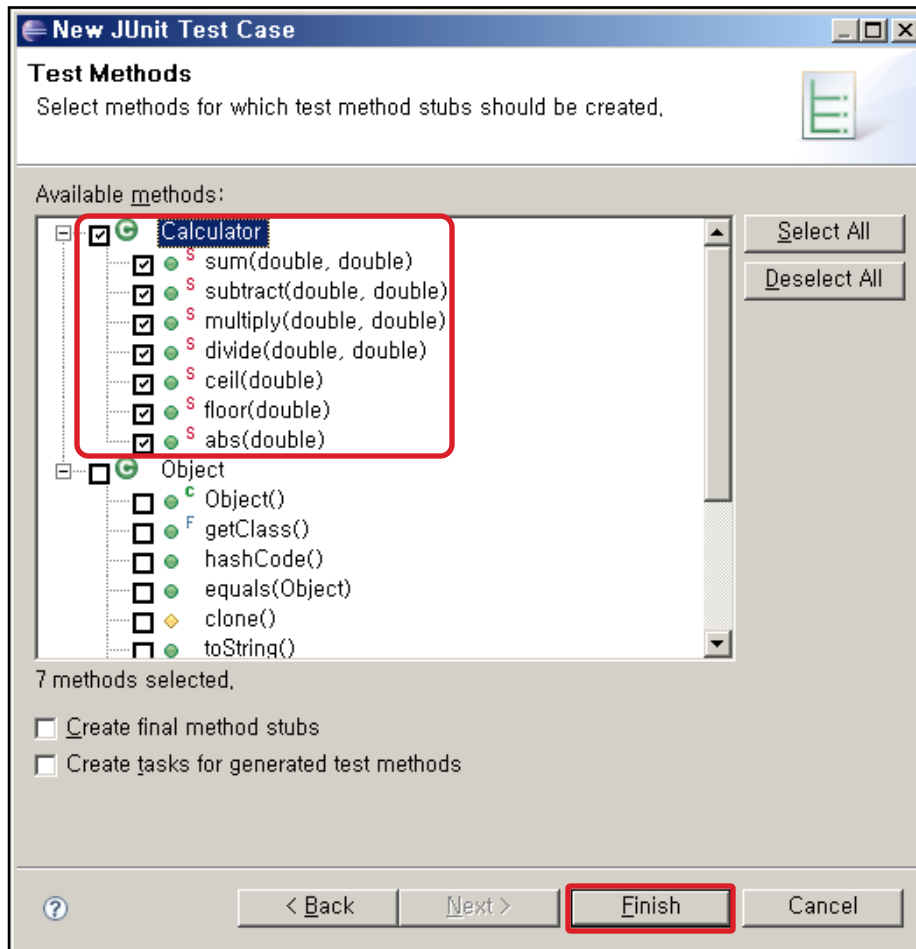


5. 도구 기능 소개

5.3 Test Case 작성하기(5/8)

JUnit

- Calculator클래스 내의 테스트하고자 하는 메소드 체크 → Finish



5. 도구 기능 소개

5.3 Test Case 작성하기(6/8)

JUnit

- 테스트하고자 하는 메소드의 내용을 구현
 - 테스트 케이스 메소드를 구현하기 : 오른쪽의assert 메소드 이용
 - JUnit에서 가장 많이 이용되는 단정(assert) 메소드
 - > assertEquals(x, y) : x와y가 같으면 테스트 통과
 - > assertFalse(b) : b가 false이면 테스트 통과
 - > assertTrue(b) : b가 true이면 테스트 통과
 - > assertNull(o) : 객체o가 null이면 테스트 통과
 - > assertNotNull(o) : 객체o가 null이 아니면 테스트 통과
 - > assertEquals(ox, oy) : ox와 oy가 같은 객체를 참조하고 있으면 테스트 통과
 - > assertEquals(ox, oy) : ox와oy가 같은 객체를 참조하고 있지 않으면 통과
 - > assertEquals : 테스트를 바로 실패처리

```
import junit.framework.TestCase;

public class CalculatorTest extends TestCase {

    public void testSum() {
        fail("Not yet implemented");
    }

    public void testSubtract() {
        fail("Not yet implemented");
    }

    public void testMultiply() {
        fail("Not yet implemented");
    }

    public void testDivide() {
        fail("Not yet implemented");
    }

    public void testCeil() {
        fail("Not yet implemented");
    }

    public void testFloor() {
        fail("Not yet implemented");
    }

    public void testAbs() {
        fail("Not yet implemented");
    }
}
```

5. 도구 기능 소개

5.3 Test Case 작성하기(7/8)

JUnit

- 테스트하고자 하는 메소드의 내용을 구현
 - Sum의 테스트 예시
 - > Calculator.sum(a,b)를 통해서 두수를 더한 값을 반환
 - » 예시 값으로 임의의 값 사용 – a:4, b:5
 - » assertTrue를 통해 result가9 인지 확인

```
Public void testSum()
{
    double result;
    result = Calculator.sum(4,5);
    assertTrue("The result should be 9.",(result ==9));
}
```


5. 도구 기능 소개

5.3 Test Case 작성하기(8/8)

JUnit

- 테스트하고자 하는 메소드의 내용을 구현
 - 전체 메소드에 대해서 구현

```
import junit.framework.TestCase;

public class CalculatorTest extends TestCase {

    public void testSum() {
        double result;
        result = Calculator.sum(4, 5);
        assertTrue("The result should be 9.", (result == 9));
    }

    public void testSubtract() {
        double result;
        result = Calculator.subtract(5, 4);
        assertTrue("The result should be 1.", (result == 1));
    }

    public void testMultiply() {
        double result;
        result = Calculator.multiply(4, 5);
        assertTrue("The result should be 20.", (result == 20));
    }

    public void testDivide() {
        double result;
        result = Calculator.divide(15, 3);
        assertTrue("The result should be 5.", (result == 5));
    }

    public void testCeil() {
        double result;
        result = Calculator.ceil(3.6);
        assertTrue("The result should be 4.", (result == 4));
    }

    public void testFloor() {
        double result;
        result = Calculator.floor(3.6);
        assertTrue("The result should be 3.", (result == 3));
    }

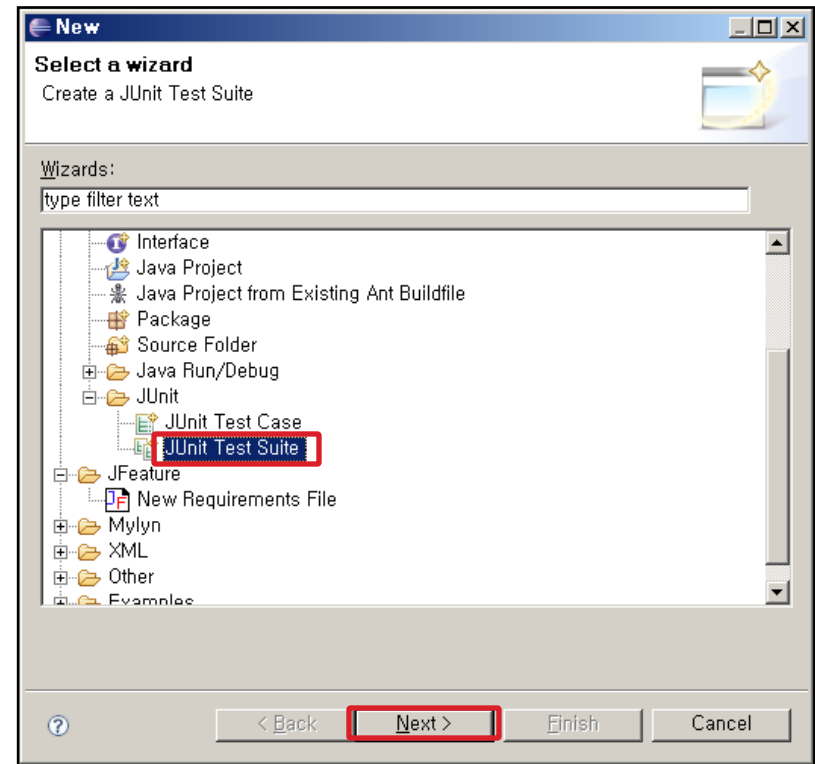
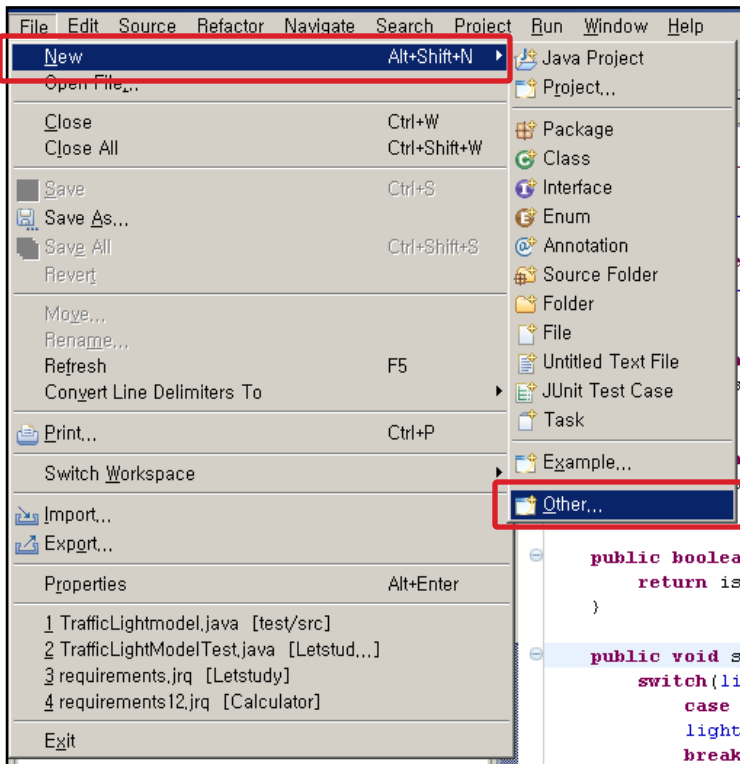
    public void testAbs() {
        double result;
        result = Calculator.abs(10);
        assertTrue("The result should be 10.", (result == 10));
    }
}
```

5. 도구 기능 소개

5.4 Test Suite 작성하기(1/3)

JUnit

- 메뉴바의 File → New → Other → JUnit선택 → JUnitTestSuite → Next
 - Test클래스를 한데 묶어서 테스트(본 예시), 특정 메소드만 실행하거나 하지 않을 때 사용

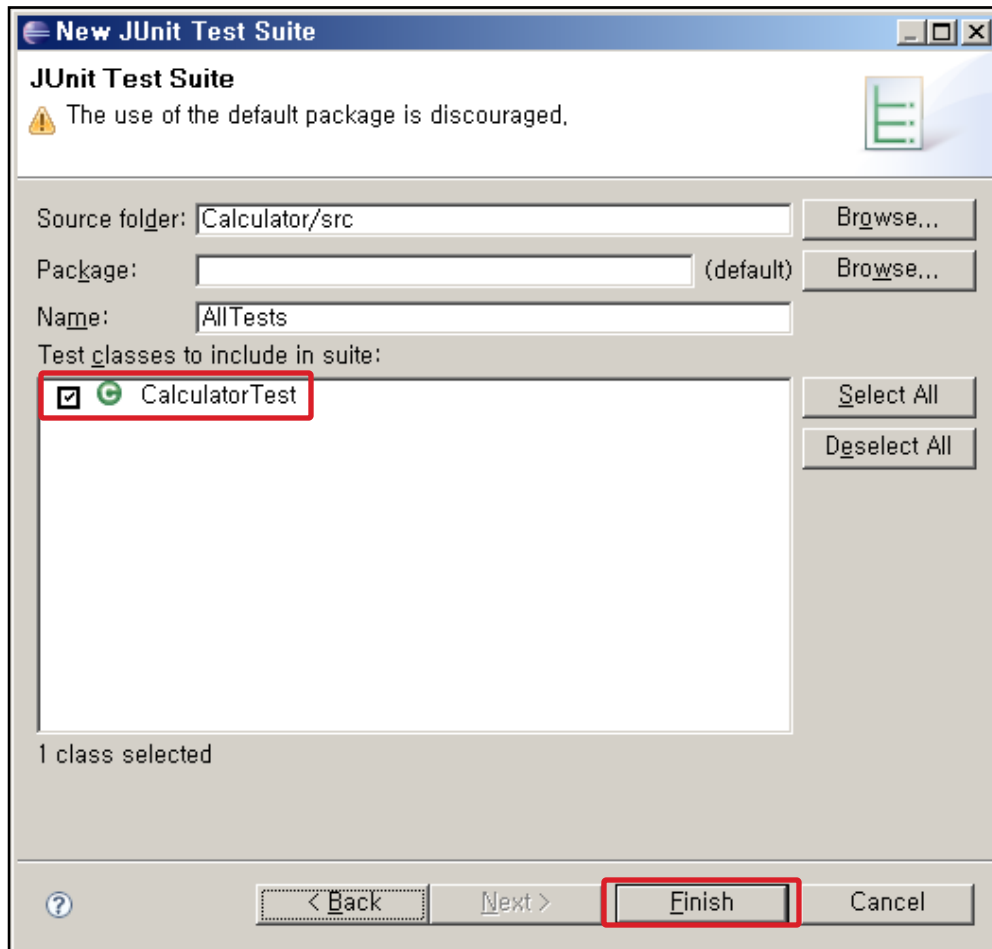


5. 도구 기능 소개

5.4 Test Suite 작성하기(2/3)

JUnit

- 테스트할 클래스를 선택 → Finish



5. 도구 기능 소개

5.4 Test Suite 작성하기(3/3)

JUnit

- Test Suite 작성 결과

```
import junit.framework.Test;

public class AllTests {

    public static Test suite() {
        TestSuite suite = new TestSuite("Test for default package");
        //$JUnit-BEGIN$
        suite.addTestSuite(CalculatorTest.class);
        //$JUnit-END$
        return suite;
    }
}
```

5. 도구 기능 소개

5.5 테스트 실행 및 결과 (1/2)

JUnit

- JUnit 테스트의 실행 : Test Case선택 → RUN → Run As → JUnit Test
 - JUnit Test 실행화면 : 모든 단위테스트에 성공(녹색막대), 실패(붉은색 막대) 표시

```
import junit.framework.TestCase;

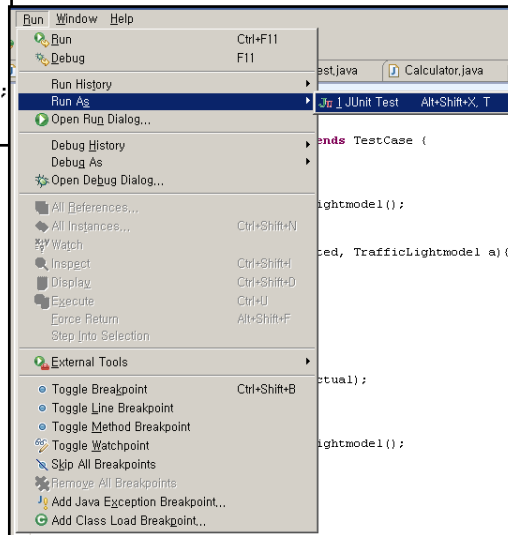
public class CalculatorTest extends TestCase {

    public void testSum() {
        double result;
        result = Calculator.sum(4, 5);
        assertTrue("The result should be 9.", (result == 9));
    }

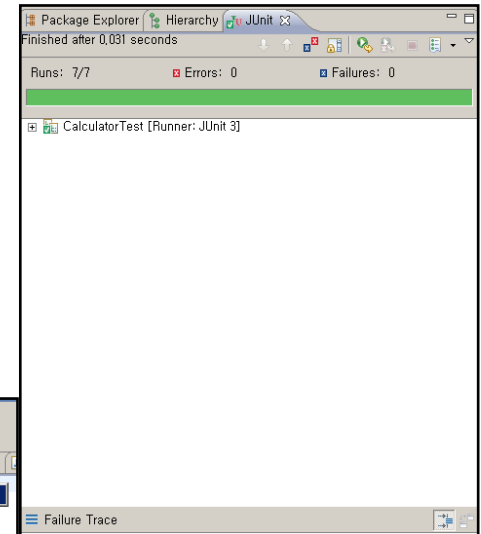
    public void testSubtract() {
        double result;
        result = Calculator.subtract(5, 4);
        assertTrue("The result should be 1.", (result == 1));
    }

    public void testMultiply() {
        double result;
        result = Calculator.multiply(4, 5);
        assertTrue("The result should be 20.", (result == 20));
    }
}
```

[Test Case 작성 화면]



[JUnit 실행 과정]



[JUnit Test 실행 화면]

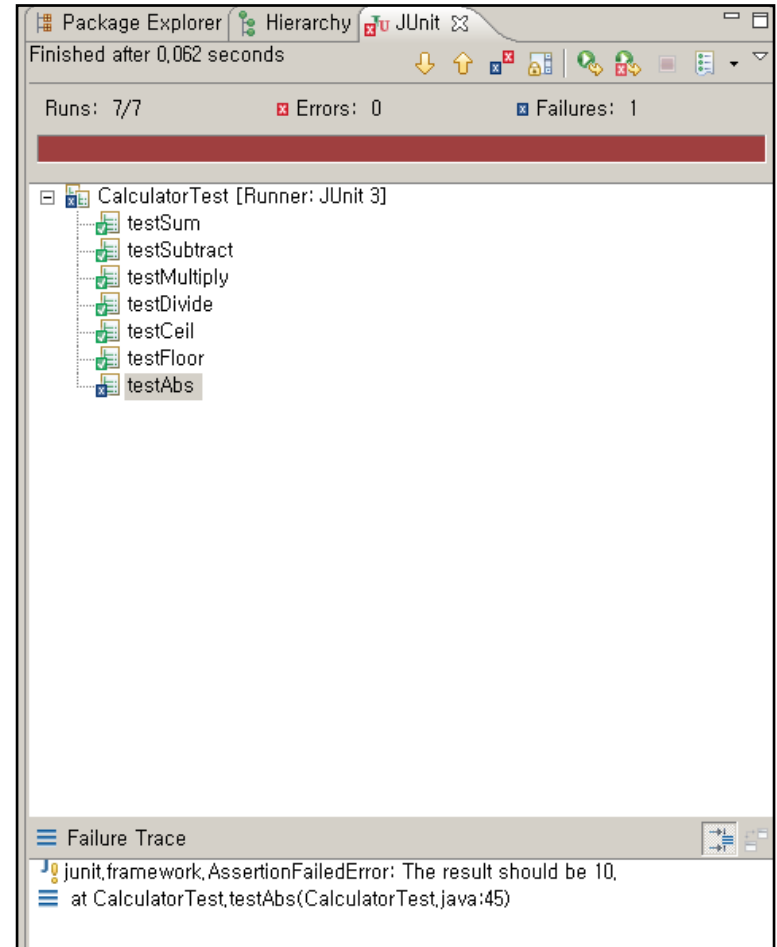


5. 도구 기능 소개

5.5 테스트 실행 및 결과 (2/2)

JUnit

- JUnit 테스트 실행화면
 - 테스트 메소드 옆에 성공한 테스트는 v 표, 실패한 테스트의 목록은 x 표로 표시
 - 상단메뉴
 - > Runs : 실행된 테스트의 개수
 - > Errors : 예외상황이 발생한 테스트의 개수
 - > Failures: 실패한 테스트의 개수
 - 상단 막대
 - > 빨간 막대 : 테스트실패
 - > 초록 막대 : 테스트성공
 - Failure Trace
 - > 실패 부분의 추적정보를 보임
 - > 실패한 테스트의 메소드 및 위치를 확인
 - > 실패메시지 더블클릭 : 소스코드 위치로 이동



5. 도구 기능 소개

5.6 테스트 실행을 위한 메뉴 소개

JUnit

- Tool bar 기능소개



| 항 목 | 기 능 |
|---|---|
|  Nest Failed Test | 테스트 메소드 목록에서 실패한 테스트 메소드 목록간에 하위 이동을 수행 |
|  Previous Failed Test | 테스트 메소드 목록에서 실패한 테스트 메소드 목록간에 상위 이동을 수행 |
|  Show Failures Only | 실패 항목만 보임 |
|  Scroll lock | 스크롤바를 고정 |
|  Return Test | 테스트를 다시 실행 |
|  Return Test (Failures First) | 테스트를 다시 실행 |
|  Test Run History | 테스트 실행 히스토리를 보여줌 |
|  Stop JUnit Test Run | JUnit Test실행을 멈춤 |

6. 도구 활용 예제

세부 목차

JUnit

- 6.1 예제 소개
- 6.2 프로젝트 생성
- 6.3 테스트 케이스 생성
- 6.4 테스트 수행

6. 도구 활용 예제

6.1 예제 소개

JUnit

- 예제 시스템 : 교통 신호등 프로그램 구현
 - 교통신호등 요구사항 : TDD(Test Driven Development)을 이용한 예제
 - 계산기 요구사항 : 빨강, 노랑, 초록 불을 가지고 있는 교통신호등 프로그램을 개발한다

신호등의 처음 상태는 빨간 불

신호등의 상태 변화는 빨간 불, 빨간 불과 노란 불 동시에, 초록 불, 노란 불, 다시 빨간 불 순

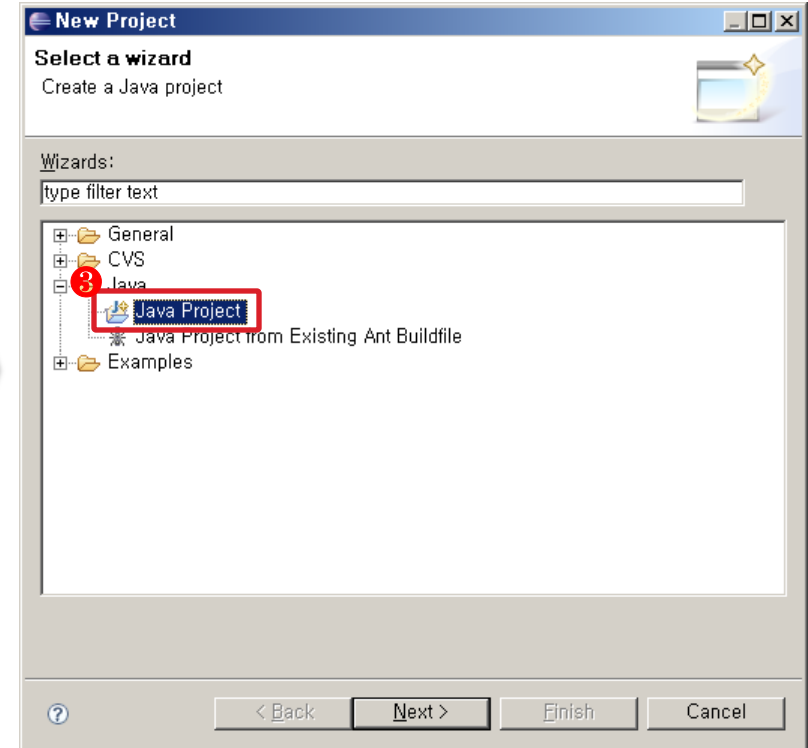
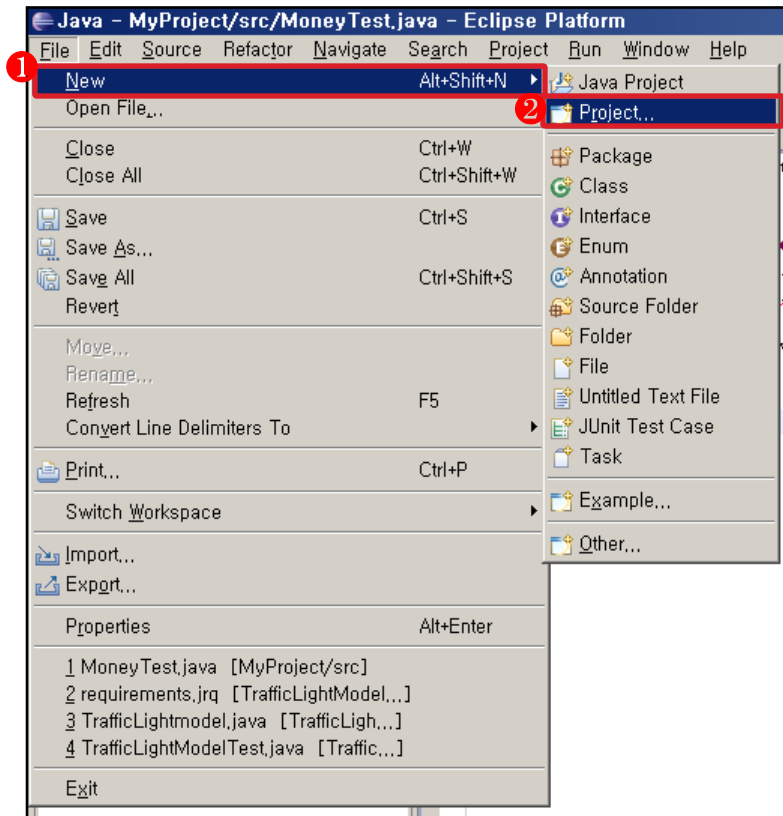
- TDD는 정의된 요구사항을 기반으로 Test Case를 먼저 만들고, 이를 기반으로 구현
- TDD는 처음부터 프로그램을 만드는 것이 아닌, 오류를 가정하고 오류를 해결해 나감
 - > 점차적으로 안정 된 프로그램을 만들어 가는 방식

6. 도구 활용 예제

6.2 프로젝트 생성 (1/2)

JUnit

- 요구사항 : "신호등의 처음 상태는 빨간 불"을 만족하는 코드 작성
 - File → New → Project → Java Project → Next

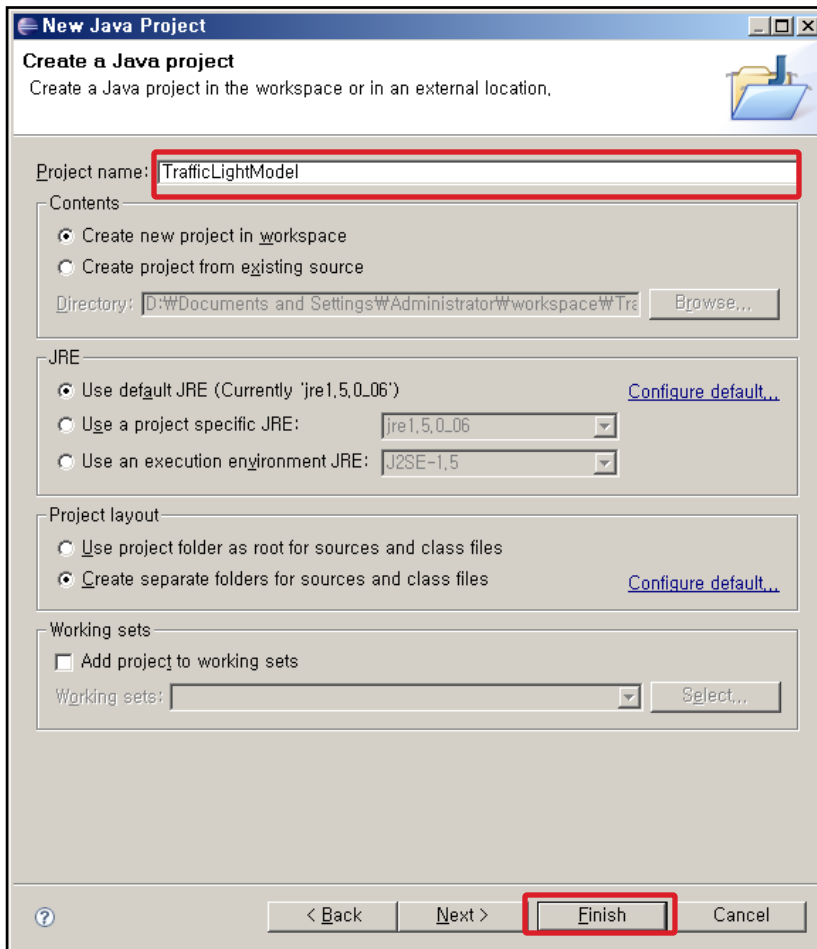


6. 도구 활용 예제

6.2 프로젝트 생성 (2/2)

JUnit

- Project 만들기 : TrafficLightModel
 - Project name에 TrafficLightModel → Finish



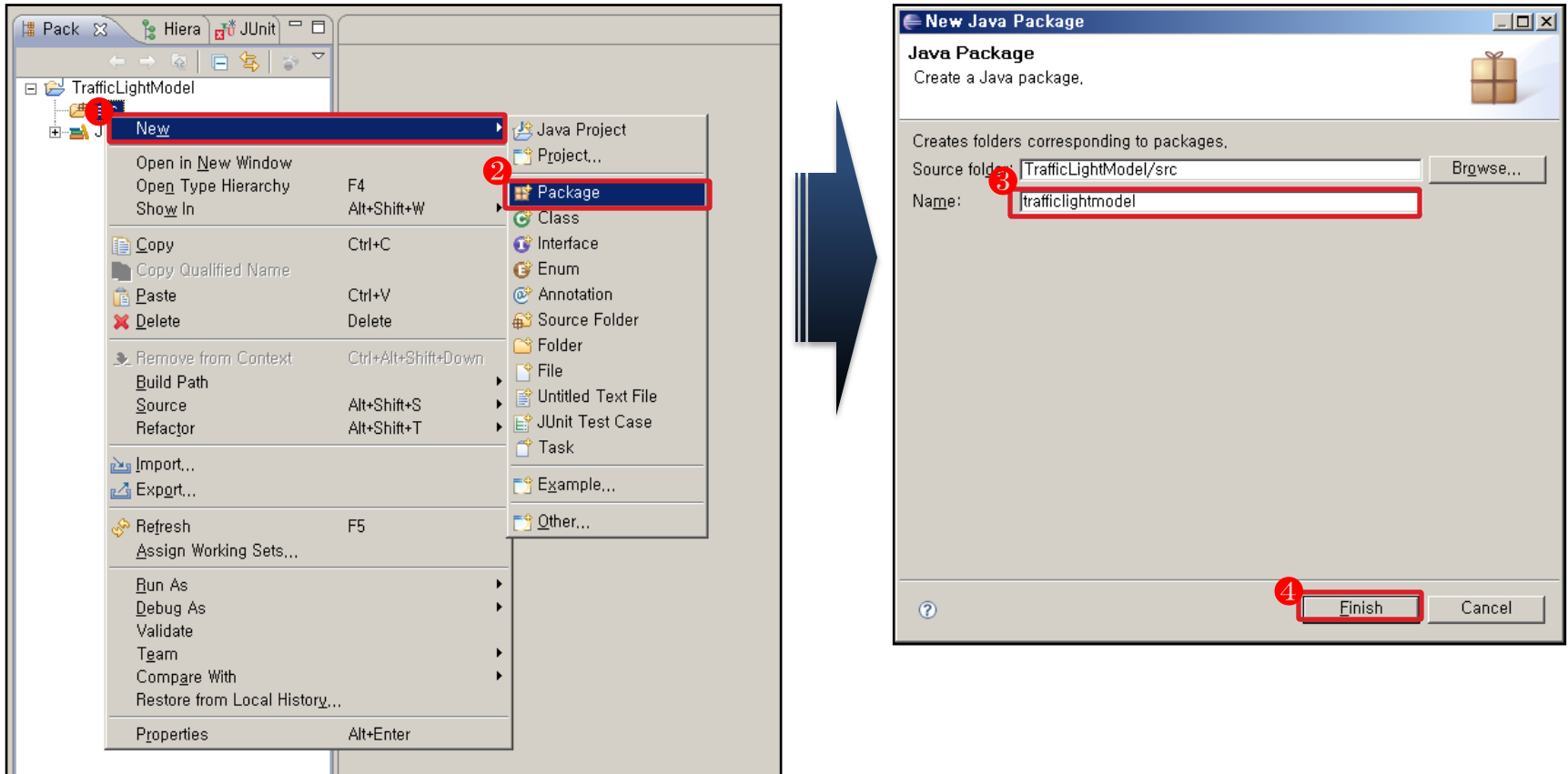
6. 도구 활용 예제

6.3 테스트 케이스 생성 (1/5)

JUnit

- 패키지 생성 : TrafficLightmodel

– src 선택 → 오른쪽 버튼 클릭 후 Package → Package이름을 Name으로 설정 → Finish



6. 도구 활용 예제

6.3 테스트 케이스 생성 (3/5)

JUnit

- Test Case 클래스에 테스트 할 내용을 추가

```
TrafficLightModelTest.java ✕  
  
package trafficlightmodel;  
  
import junit.framework.TestCase;  
  
public class TrafficLightModelTest extends TestCase {  
  
}
```



```
TrafficLightModelTest.java ✕  
  
import junit.framework.TestCase;  
  
public class TrafficLightModelTest extends TestCase {  
  
    public void testNewTrafficLight() {  
        TrafficLightmodel a = new TrafficLightmodel();  
        assertEquals(true, a.getRed());  
        assertEquals(false, a.getYellow());  
        assertEquals(false, a.getGreen());  
    }  
  
}
```

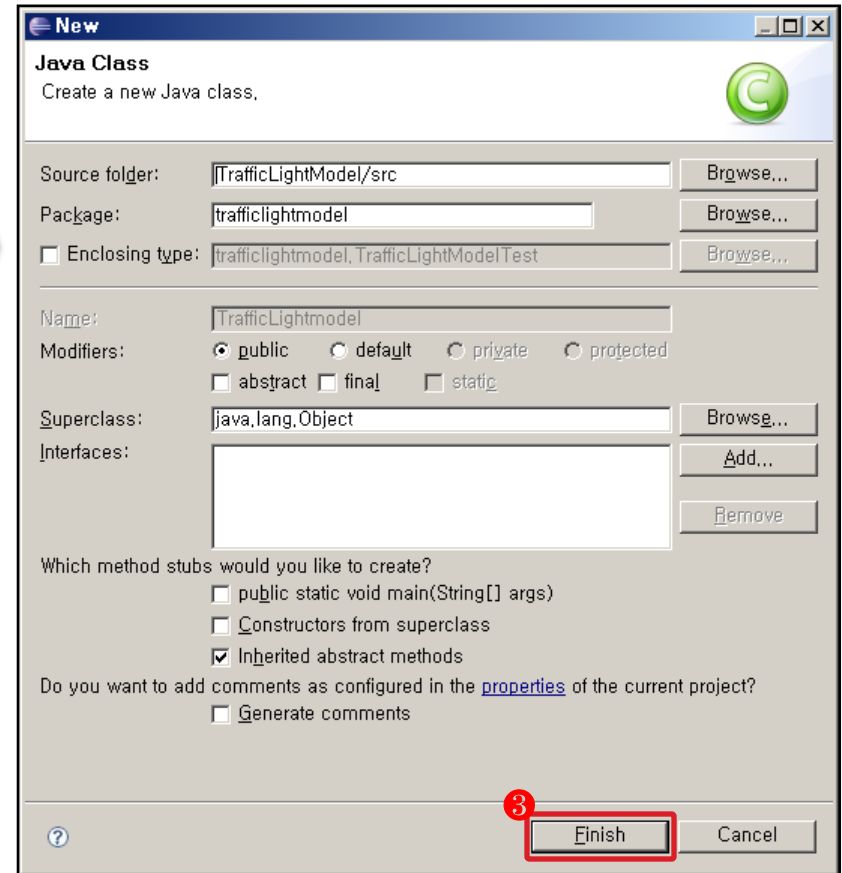
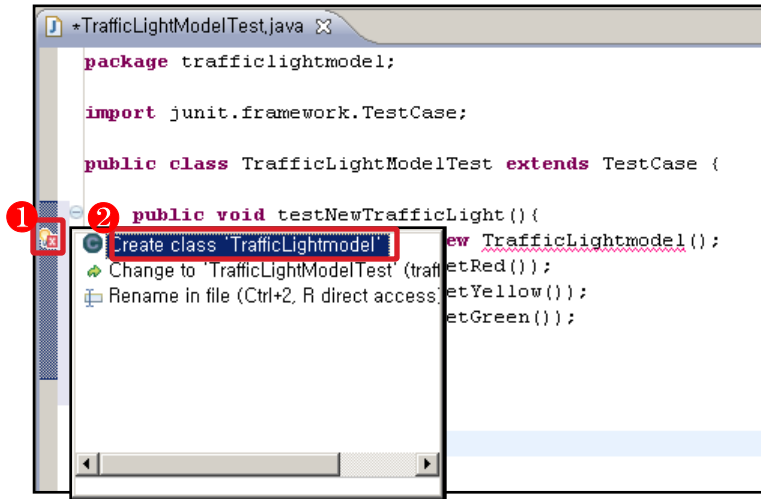
- “신호등의 처음 상태는 빨간 불”이라는 요구사항을 구현하기 위해, 테스트 케이스 중 assertEquals(true, a.getRed())를 통해서 검증
- assertEquals(false, a.getYellow())와 assertEquals(false, a.getGreen())은 false로 설정함으로써 초기값이 노란색과 녹색이 아님을 검증

6. 도구 활용 예제

6.3 테스트 케이스 생성 (4/5)

JUnit

- 클래스 생성 : TrafficLightmodel
 - TrafficLightmodel클래스가 존재하지 않기 때문에 좌측화면과 같은 오류 발생
 - Java 에디터에서 툴팁(전구) 아이콘 클릭 → Create class "TrafficLightmodel"을 선택

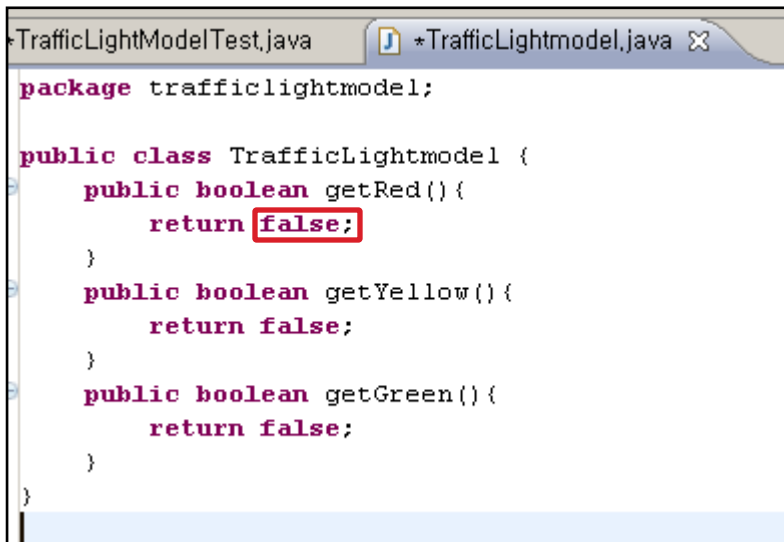


6. 도구 활용 예제

6.3 테스트 케이스 생성 (5/5)

JUnit

- TrafficLightmodel 클래스의 내용 작성 : 의도적 오류
 - TestCase 클래스에서 getRed()를 true로 설정하였기 때문에, 요구사항이 정확히 구현되었는지 확인하기 위해서 Test 대상메소드를 false로 설정
 - False로 설정하였기에 이 메소드는 에러를 발생
 - 발생한 에러를 통해 getRed()가 잘못 구현되었음을 알고, 그 결과를 바탕으로 getRed()를 수정



```
package trafficlightmodel;

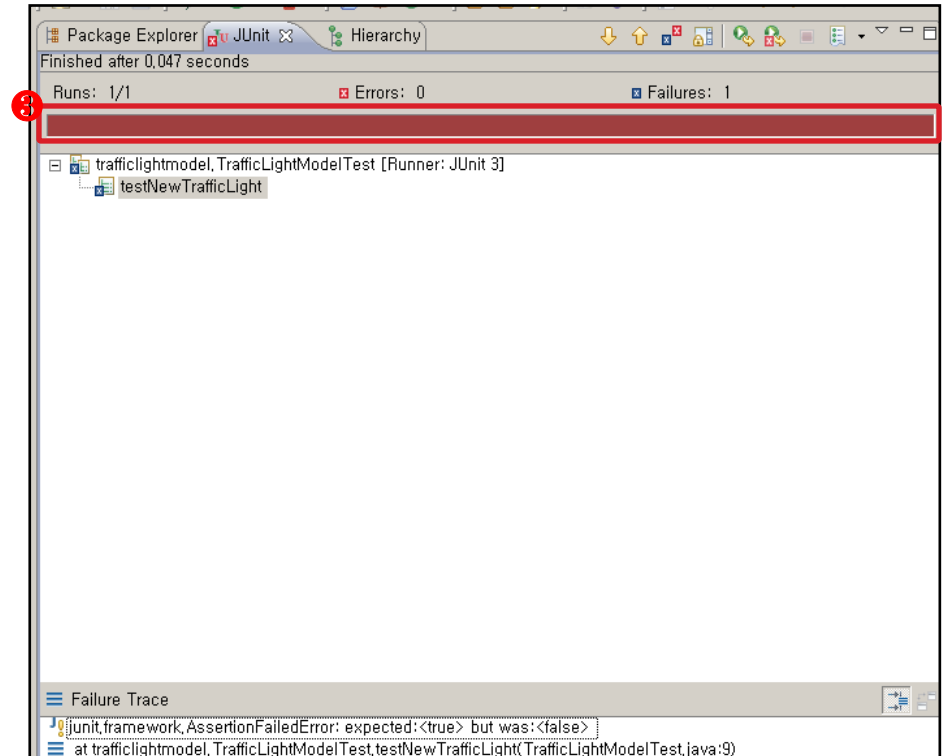
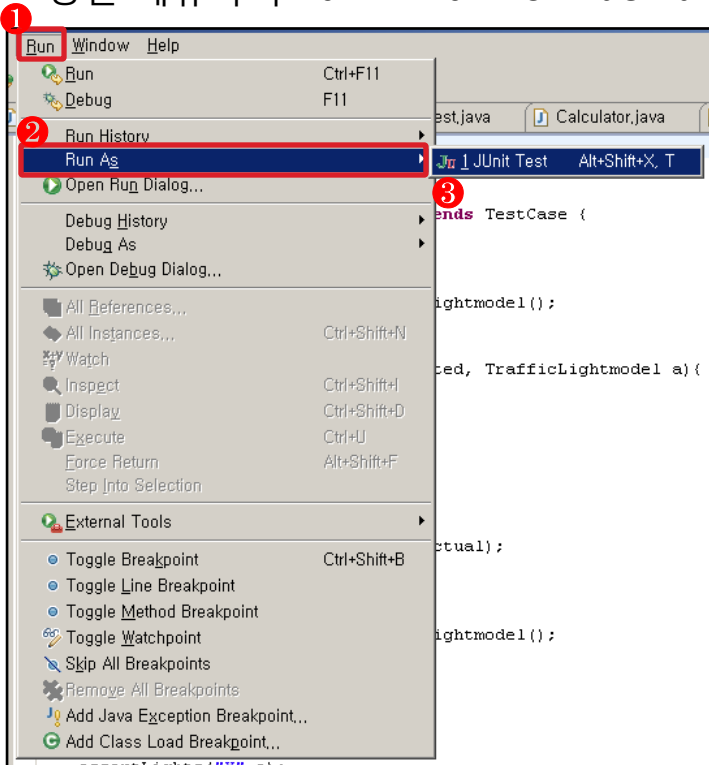
public class TrafficLightmodel {
    public boolean getRed(){
        return false;
    }
    public boolean getYellow(){
        return false;
    }
    public boolean getGreen(){
        return false;
    }
}
```


6. 도구 활용 예제

6.4 테스트 수행 (1/3)

JUnit

- 테스트를 수행
 - 상단 메뉴바의 Run → Run As → JUnit Test

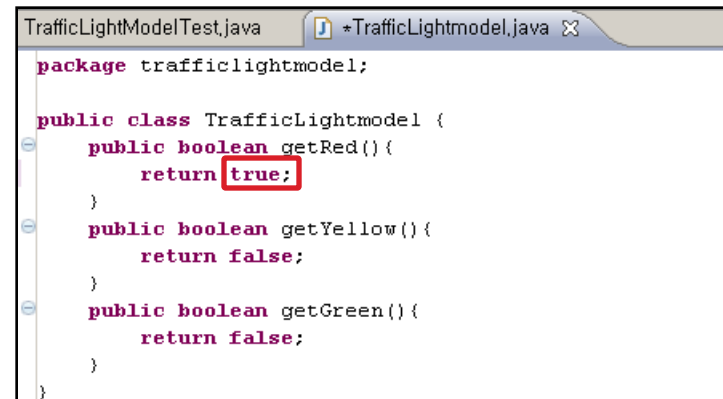
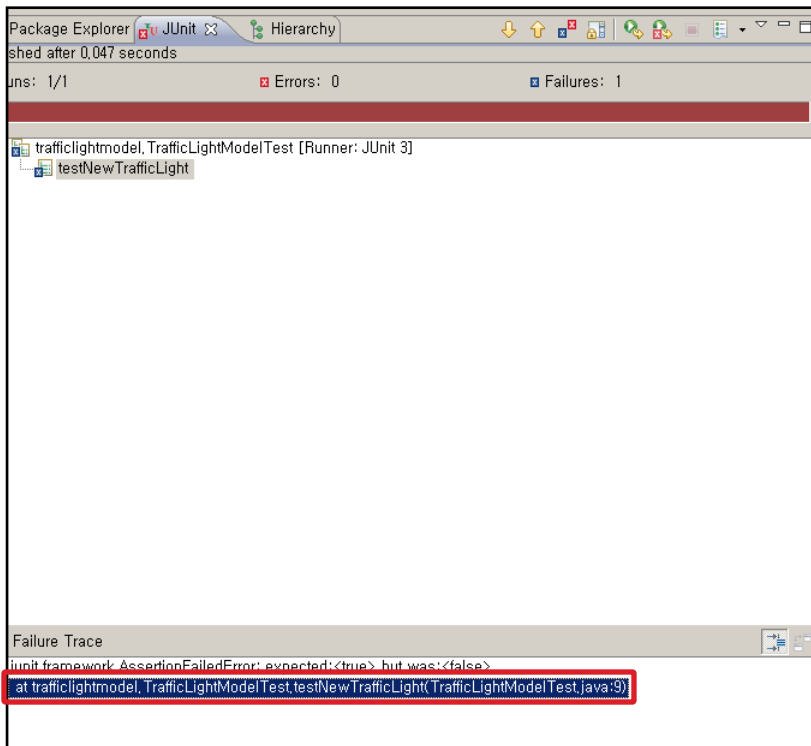


6. 도구 활용 예제

6.4 테스트 수행 (2/3)

JUnit

- 오류를 수정
 - JUnit뷰의 하단 Failure Trace부분을 더블클릭
 - > 테스트케이스getRed() 호출 부분에 문제가 발생했다는 것을 알 수 있음
 - 커서를 접근 메소드 위로 옮겨 F3키를 눌러 구현부분 TrafficLightModel.java로 이동
 - 최초 요구사항인 "신호등의 초기 상태는 빨간 불"를 만족시키기 위해 getRed()의 리턴값이 true 이여야 하므로 현재의 잘못된 리턴값인 false를 true로 변경

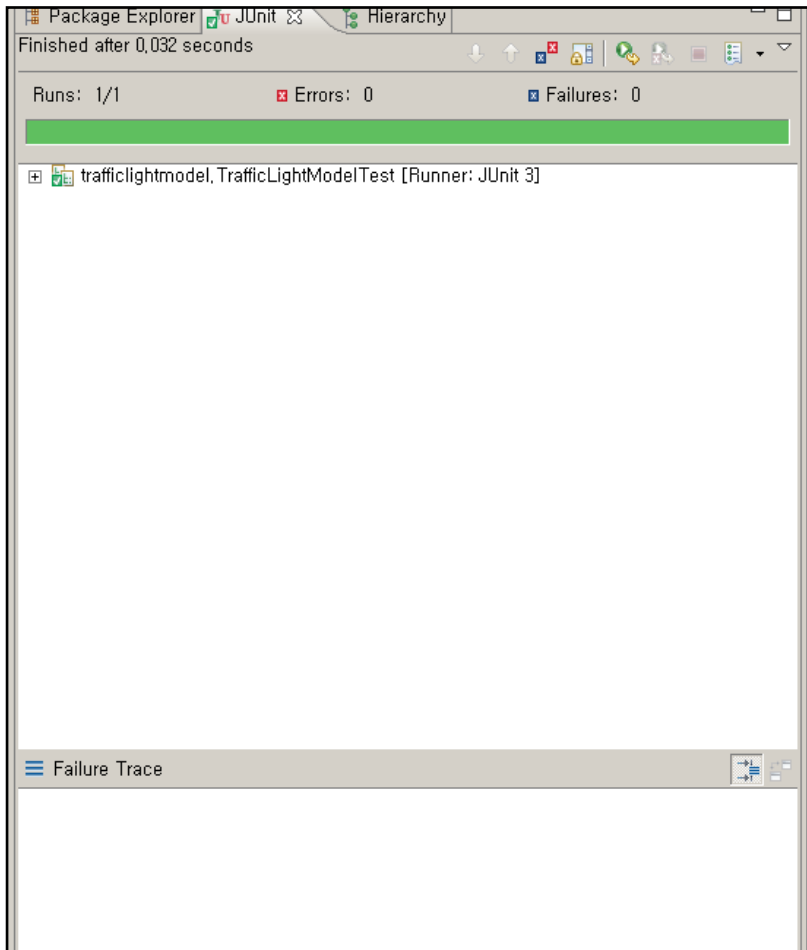


6. 도구 활용 예제

6.4 테스트 수행 (3/3)

JUnit

- 테스트를 재 실행 : Run → Run As → JUnit Test
 - 실패 메시지 미 발생, 녹색 막대로 이상 없음을 표시



7. FAQ

JUnit

질문1) TDD(Test Driven Development) 활동이 중요한 이유가 무엇입니까?

➡ 답변1 : 개발 초기 단계에서 발견된 결함은 수정이 용이하고 시간과 비용을 감소시키는 반면, 개발 후기 단계에서 발견된 결함들을 수정하기에는 비용과 시간이 많이 소비되고, 프로젝트가 실패될 확률이 높아지기 때문입니다.

질문2) 단위 테스트가 무엇입니까?

➡ 답변2 : 단위테스트는 단위모듈(ex : 메소드)에서 문제발생소지가 있는 모든 부분에 대해서 테스트하는 작업입니다.

8. 도구 평가

JUnit

- 활용성
 - 대부분의 자바를 지원하는 도구에서 사용가능
- 범용성
 - 뛰어난 성능과 넓은 활용범위에 의해 대부분의 도구에 기능 탑재
- 호환성
 - 자바를 지원하는 모든 도구에 호환
 - 도구의 버전을 가리지 않음
 - 도구가 가능한 환경이면 어떤 환경에서도 작동가능
- 성능
 - 빠른 작동성능, 도구에 같이 로드 되어도 성능상 문제 없음
- 기타
 - 뛰어난 기능으로 대부분의 도구에 기본탑재
 - 기본 탑재된 플러그인 이더라도, 필요 시 버전업이 가능하며 여러 버전을 동시에 설치 가능

도구평가 의견

- JUnit은 Eclipse에 기본으로 플러그인이 포함되어 있어 설치가 용이
- Test Suite를 제공함으로써 여러 Test Case 클래스를 한번에 실행 시킬 수 있음
- 대부분 도구에서 지원하고 있기 때문에, 도구 교육시간이 단축

9. 용어 정리

JUnit

본 매뉴얼에서 사용하고 있는 용어의 정리

| | |
|------------|--|
| Test Case | 특정한 프로그램부분 및 경로를 실행해 보거나 특정한 요구사항에 준수하는지를 확인하기 위해 개발된 입력 값, 실행조건, 그리고 예상된 결과로 이루어져 있는 하나의 테스트 세트 |
| Test Suite | 일정한 순서에 의하여 수행될 개별 테스트들의 집합, 또는 패키지. Suite는 응용분야나 우선순위, 내용에 연관 |
| TDD | 자동화된 테스트로 개발을 이끌어가는 개발방식을 테스트주도개발이라 칭함. TDD는 분석기술이며, 설계기술이며, 개발의 모든 활동을 구조화하는 기술 |
| 단위테스트 | 단위테스트는 단위코드에서 문제발생소지가 있는 모든 부분을 테스트하는 작업 |