Junit, Eclipse, Clover & JDepend

2013 spring software verification

Team 1

200711460 이상열

200711470 정재호

201111344 김재엽

201211350 박주광

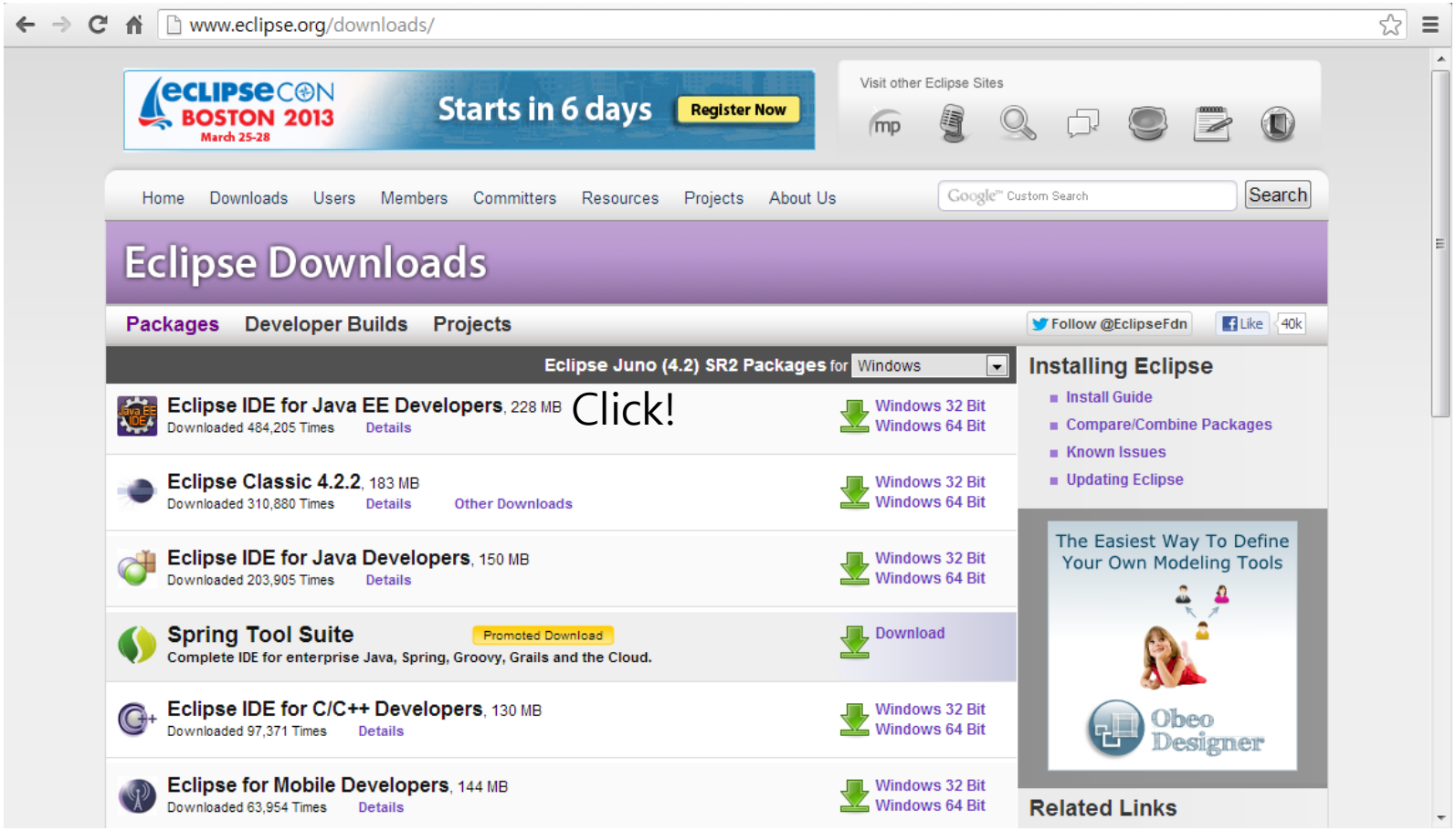
Contents

Eclipse

Eclipse

Eclipse

What is the Eclipse?



Eclipse

What is the Eclipse?

Eclipse is a multi-language software development environment comprising a baseworkspace and an extensible plug-in system for customizing the environment.

Eclipse

What is the Eclipse?

JUnit

JUnit

JUnit

What is Unit Test?

Unit Test is a method by which individual units of source code are tested to determine if they are fit for use.

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy.

Unit 테스트를 통하여 Source Code의 신뢰성을 높여준다.

JUnit

Benefit of Unit Test

Find problems Early

-Unit tests find problems early in the development cycle.

Facilitate Change

-The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.

Simplifies Integration

-Unit testing may reduce uncertainty in the units themselves and can be used if a bottom-up testing style approach.

JUnit

Benefit of Unit Test(Cont.)

Documentation

-Unit testing provides a sort of living documentation of the system.

Unit의 작동에 대한 적합/부적합함의 정보를 문서를 통해 제공받을 수 있다..

Design

-When software is developed using a test-driven approach, the unit test may take the place of formal design.

디자인과 맞지 않게 개발된 Unit일 경우 테스트를 통해 발견 할 수 있다.

JUnit

Limitation of Unit Test

Disable to catch all errors

-It cannot evaluate every execution path in any but the most trivial programs.

→ Multiple thread와 같은 경우에 테스트 결과의 신뢰성이 떨어진다.

Difficulty of setting up realistic and useful test

-If these initial conditions are not set correctly, the test will not be exercising the code in a realistic context

→ Test시 Unit들은 전체 시스템의 일부로 작동하도록 초기조건을 만족시켜야 하지만 부적절한 초기조건들로 인해 테스트의 정확성이 떨어질 수 있다.

JUnit

What is JUnit?

JUnit is a unit testing framework for the Java programming language.

And that has been important in the development of TDD(Test-Driven Development).

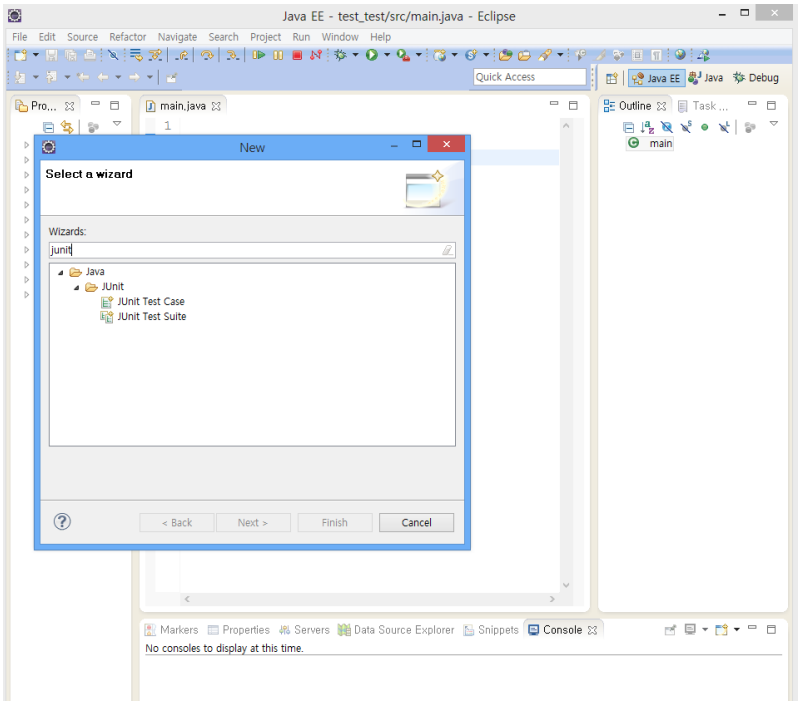
Developer(s)	Kent Beck, Erich Gamma, David Saff, Mike Clark (University of Calgary)
Stable release	4.11[1] / November 14, 2012; 3 months ago
Written in	Java
Operating system	Cross-platform
Type	Unit testing tool
License	Common Public License
Website	junit.sourceforge.net

In Wikipedia

JUnit

Install

Eclipse 2.1 버전 이상에서는 JUnit을 기본적으로 사용.



메뉴에서 File->New->Other 에서 Junit을 입력하여 확인 할 수 있다.

JUnit

Annotation

@Test	Unit test 대상 method를 정의
@Test(timeout)	테스트 시간을 예측할 때 사용 시간보다 길게 진행될 시 Fail
@Test(expected)	예외를 지정할 때 사용 예외가 발생하지 않을 시 Fail

@Ignore	테스트 하지 않을 method 앞에 작성. 이후에 오는 테스트를 무시한다.
@After @Before	각 단위 테스트 method의 실행 앞, 뒤에서 초기화 및 자원 정리
@AfterClass @BeforeClass	각 단위 테스트 class 수행 전, 후에 초기화 및 자원 정리

JUnit

Annotation(Cont.)

@RunWith	사용자가 지정한 러너를 통해 특정 클래스를 실행
@SuiteClasses	테스트 하고자 하는 다수의 클래스를 지정
@Parameters	다수의 parameter값을 테스트 하려고 할 때 자동으로 테스트를 실행

JUnit

Method

assertEquals (타입 expected, 타입 result)	예상값과 결과값을 매개변수로 지정하고 두 값이 같을 경우에 성공. 타입에는 int, long 등의 기본 자료형과 String, Object가 사용 가능
assertEquals (String Message, 타입 expected, 타입 result)	위에 assertEquals에서 예상 값과 결과 값이 같지 않을 경우 첫 번째 매개변수인 Message 값을 리턴
assertTrue (boolean condition)	condition이 true이면 성공, false이면 실패로 처리한다
assertFalse (boolean condition)	condition이 true이면 실패, false이면 성공으로 처리한다

JUnit

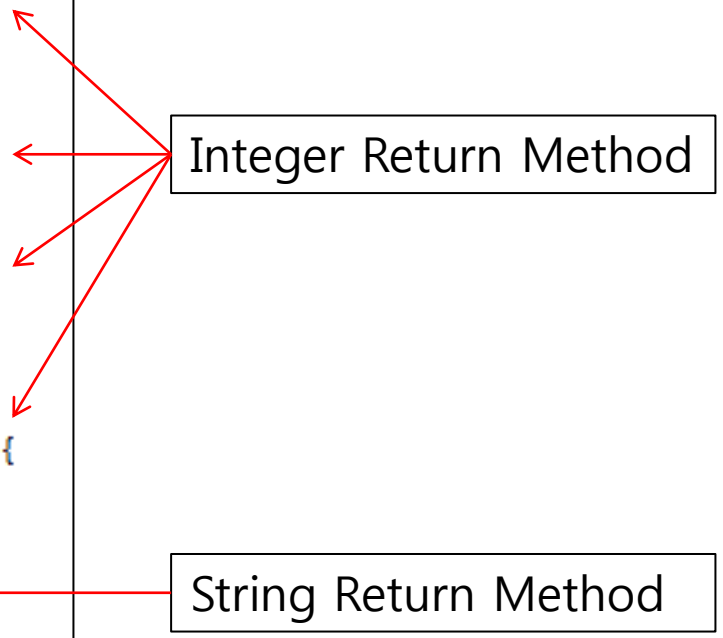
Method(Cont.)

assertNotNull (Object obj)	객체가 null이 아니면 성공
assertNull (Object obj)	객체가 null이면 성공
assertSame (Object expected, Object result)	객체가 동일하면 성공, 다르면 실패
assertNotSame (Object expected, Object result)	객체가 다르면 성공, 같으면 실패
Fail() 관련 method	테스트가 실패일 경우를 쉽게 처리하는 것과 관련된 method

JUnit

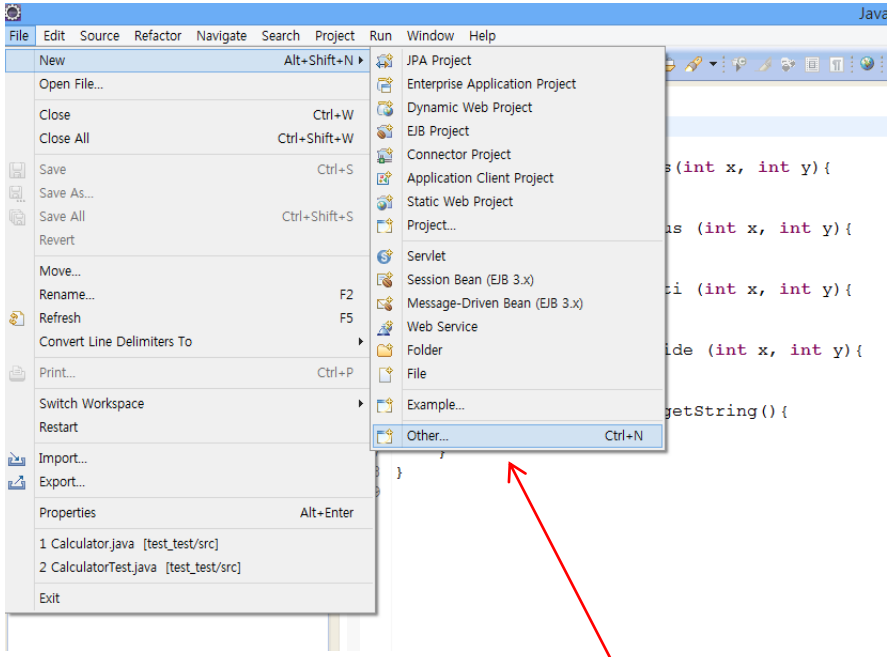
Example1

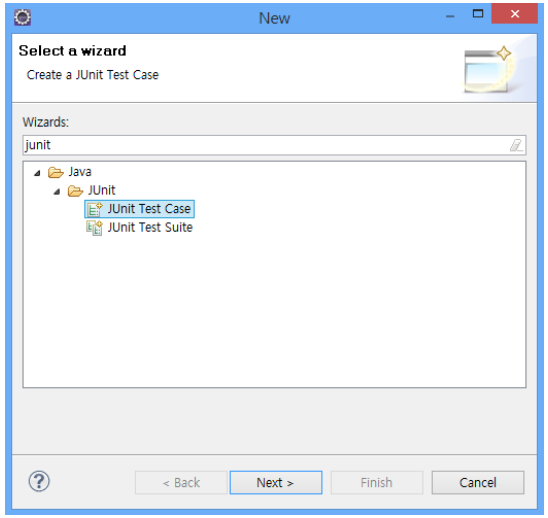
```
public class Calculator {
    public static int plus(int x, int y){
        return x + y;
    }
    public static int minus (int x, int y){
        return x - y;
    }
    public static int multi (int x, int y){
        return x * y;
    }
    public static int divide (int x, int y){
        return x/y;
    }
    public static String getString(){
        return "TEST";
    }
}
```



JUnit

Example1(Cont.)



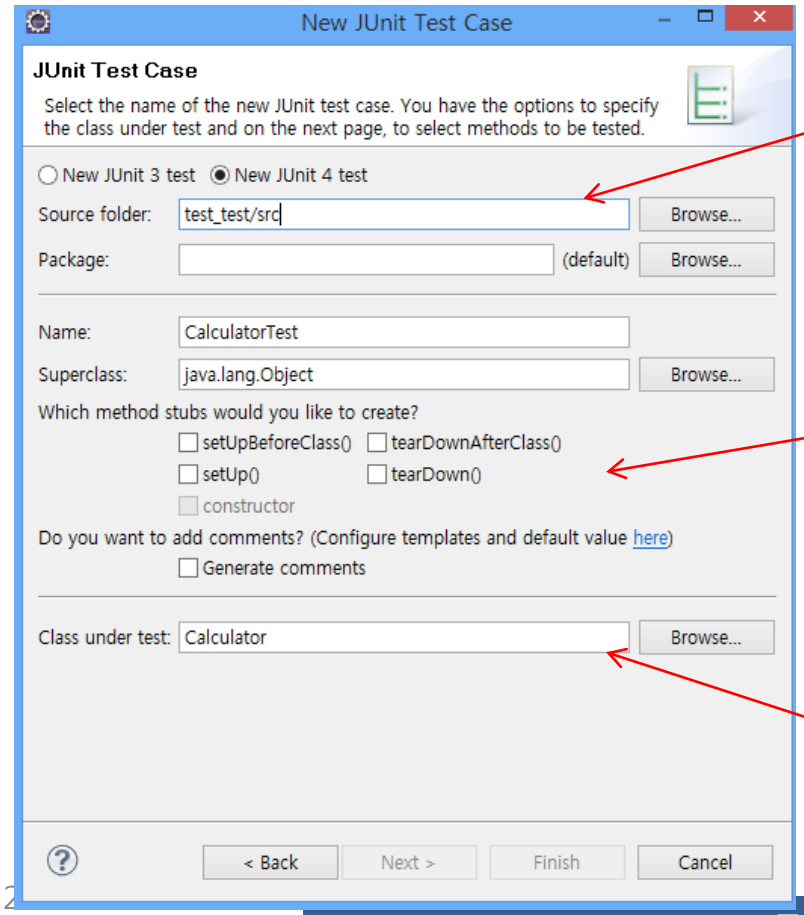


File – New – Other..을 Click

JUnit 검색 - JUnit Test Case 선택

JUnit

Example1(Cont.)



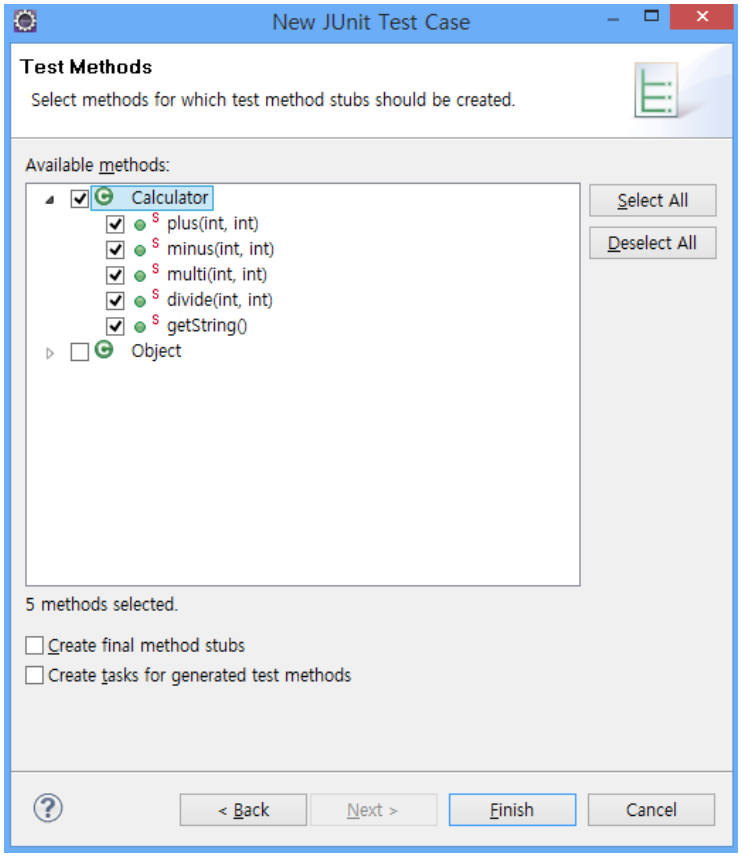
Test 할 Source code 경로

setUp() : 테스트 대상 클래스의 객체를 생성하거나 네트워크 연결, DB 연결 작업 등을 수행.
tearDown() : setUp과 정 반대의 기능. 객체의 제거, 네트워크 종료, DB연결 종료 등을 수행
...beforeClass() : 클래스의 시작 혹은 끝나고 실행되는 method

Test 할 Case

JUnit

Example1(Cont.)



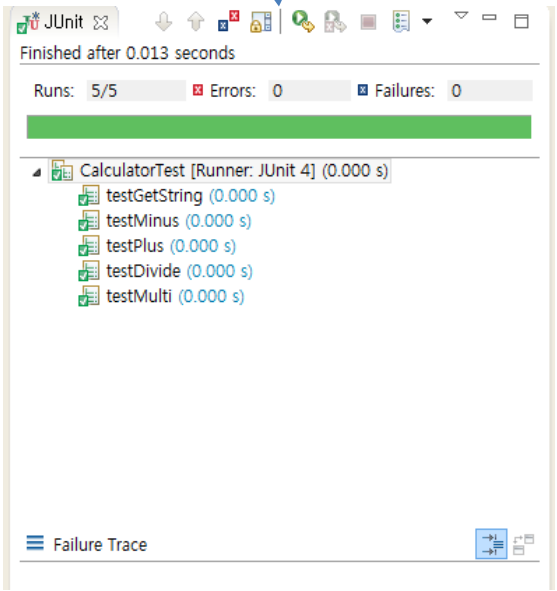

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {

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

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

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

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

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

JUnit

Example1(Cont.)

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


public class CalculatorTest {

    @Test
    public void testPlus() {
        int result = Calculator.plus(1, 2);
        assertEquals(result,3);
    }
    @Test
    public void testMinus() {
        int result = Calculator.minus(2, 1);
        assertEquals(result,1);
    }
    @Test
    public void testMulti() {
        int result = Calculator.multi(1, 2);
        assertEquals(result,2);
    }
    @Test
    public void testDivide() {
        int result = Calculator.divide(2, 1);
        assertEquals(result,2);
    }
    @Test
    public void testGetString() {
        assertEquals("TEST", Calculator.getString());
    }
}
```

assertEquals를 이용한 Testing. 성공 시 초록색으로 표시

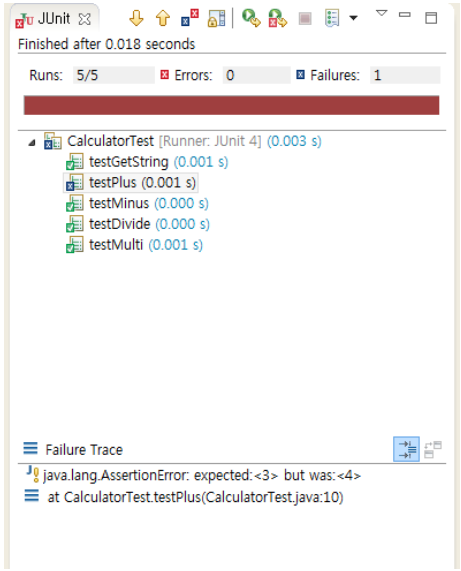


JUnit

Example1(Cont.)

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


public class CalculatorTest {

    @Test
    public void testPlus() {
        int result = Calculator.plus(1, 2);
        assertEquals(result,4);
    }
    @Test
    public void testMinus() {
        int result = Calculator.minus(2, 1);
        assertEquals(result,1);
    }
    @Test
    public void testMulti() {
        int result = Calculator.multi(1, 2);
        assertEquals(result,2);
    }
    @Test
    public void testDivide() {
        int result = Calculator.divide(2, 1);
        assertEquals(result,2);
    }
    @Test
    public void testGetString() {
        assertEquals("TEST", Calculator.getString());
    }
}
```

assertEquals로 Test 실패 시 빨간색으로 나타내고 해당 Unit을 Fail로 표시.



Error 메세지

JUnit

Example2

```
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.BeforeClass;
import org.junit.Before;
import org.junit.AfterClass;
import org.junit.After;

public class CalculatorTest {

    static private int i, j;

    @BeforeClass
    public static void ExeBeforeTest() {
        i = 3;
        j = 2;
        System.out.println("Execute @BeforeClass");
    }

    @Before
    public void testPlus() {
        int result = Calculator.plus(1, 2);
        assertEquals(result,3);
        System.out.println("Execute @Before");
    }

    @Test
    public void TestMinus() {
        int k = i - j;
        assertEquals(1, k);
        System.out.println("Execute @TestMinus");
    }
```

BeforeClass는 Test 내에서 가장 먼저 1회 실행된다.

Before는 Unit Test 앞부분에서 Test마다 1회 실행된다.

JUnit

Example2(Cont.)

```
    @Test
    public void TestPrint1() {
        System.out.println("Execute @Test_1");
    }

    @Test
    public void TestPrint2() {
        System.out.println("Execute @Test_2");
    }

    @After
    public void TestMulti() {
        int k = i * j;
        assertEquals(6, k);
        System.out.println("Execute @After");
    }

    @AfterClass
    public static void ExeAfterTest() {
        System.out.println("Execute @AfterClass");
    }
}
```

After는 Unit Test 뒷부분에서 Test마다 1회 실행된다.

AfterClass는 Test 내에서 가장 마지막에 1회 실행된다.

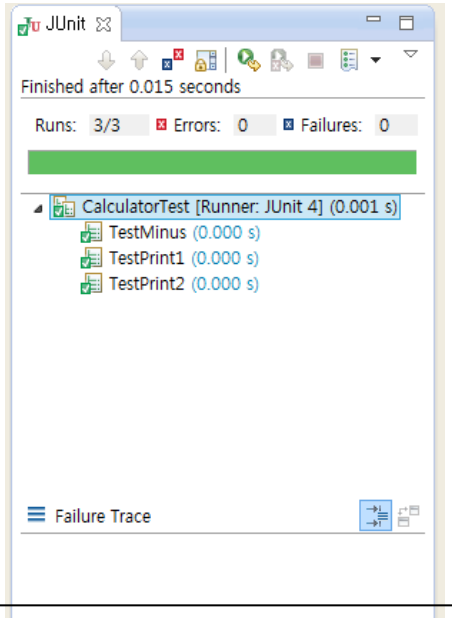
JUnit

Example2(Cont.)

Console | Markers | Propertie

<terminated> CalculatorTest [JUnit] C:₩Proc

```
Execute @BeforeClass
Execute @Before
Execute @TestMinus
Execute @After
Execute @Before
Execute @Test_1
Execute @After
Execute @Before
Execute @Test_2
Execute @After
Execute @AfterClass
```



각각의 Annotation에 대한 실행 수와 실행 순서를 Print 문을 통해 확인 할 수 있다.

@Test Annotation을 제외한 다른 Annotation의 경우 Testing은 이뤄지지 않는다.

JUnit

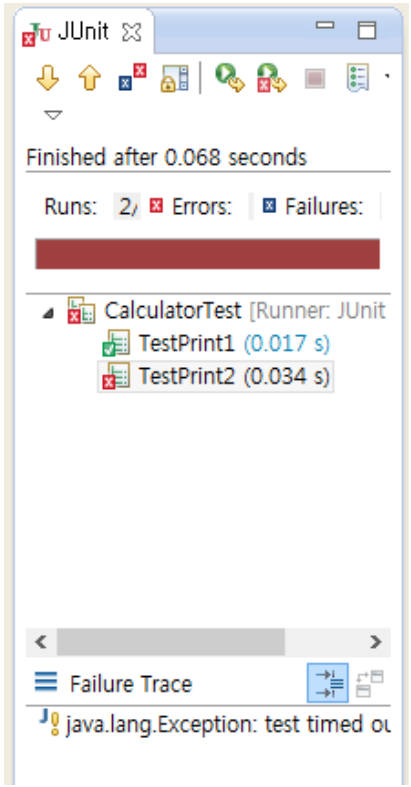
Example3

Use @Test(timeout) Annotation

```
import org.junit.Test;

public class CalculatorTest {

    @Test(timeout = 30)
    public void TestPrint1() throws InterruptedException {
        System.out.println("Execute @Test_1");
        Thread.sleep(20);
    }

    @Test(timeout = 30)
    public void TestPrint2() throws InterruptedException {
        System.out.println("Execute @Test_2");
        Thread.sleep(50);
    }

}
```

Timeout 값이 Sleep 값 보다 적다.
-Success

Timeout 값이 Sleep 값 보다 크다.
-Fail



JUnit

Example4

How to test a method that doesn't return anything?

```
@Test
public void testCollectionAdd() {
    Collection collection = new ArrayList();
    assertEquals(0, collection.size());
    collection.add("itemA");
    assertEquals(1, collection.size());
    collection.add("itemB");
    assertEquals(2, collection.size());
}
```

Unit 별로 Return 되는 값들이 아닌 method 내에서 발생되는 변화들을 값으로 가져와서 Testing 하는 방법이 있다.

이 외에 'MockObjects'와 같은 접근 방법들이 존재한다.

Clover

Clover

Clover

What is code coverage?

Code coverage is the percentage of code which is covered by automated tests.

Code coverage can be part of a feedback loop in the development process.

Clover

Benefit of code coverage

code coverage highlights aspects of the code which may not be adequately tested and which require additional testing.

→ Code coverage를 높이기 위해서 Unit test code를 더 많이 작성해야 한다.

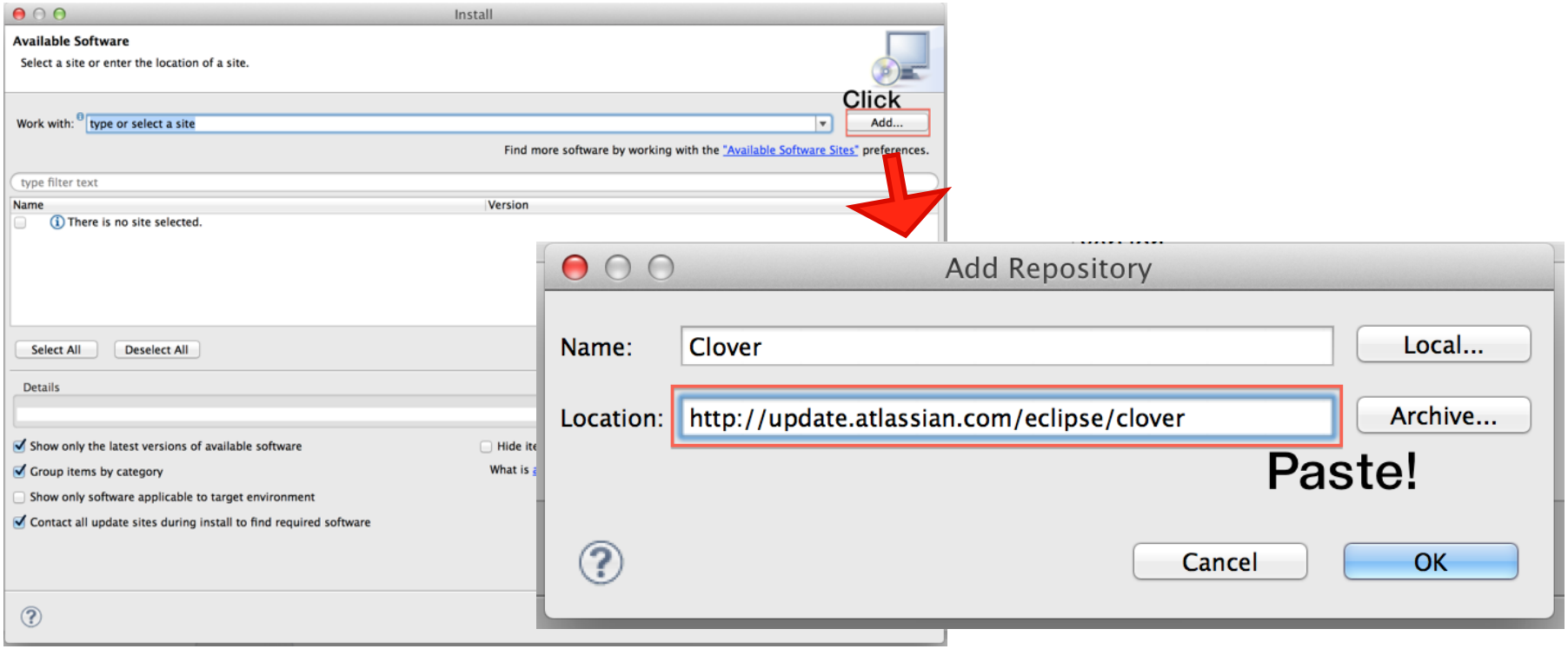
Coverage measurement helps to avoid test entropy.

→ 기존의 code가 변경되더라도 code coverage를 확인함으로써 test가 정확히 실행되었는지 확인할 수 있다.

Clover

How to install Clover?

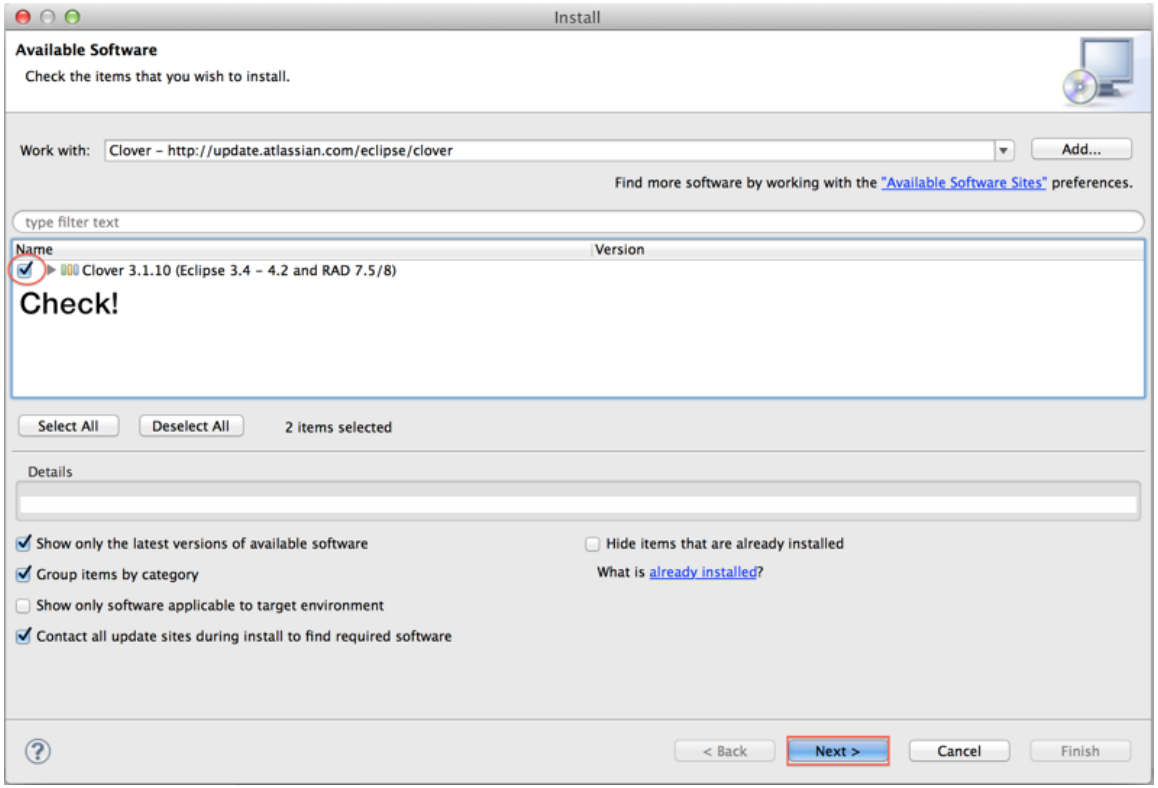
Eclipse에서 help->Install new software... 선택!



Clover

How to install Clover?

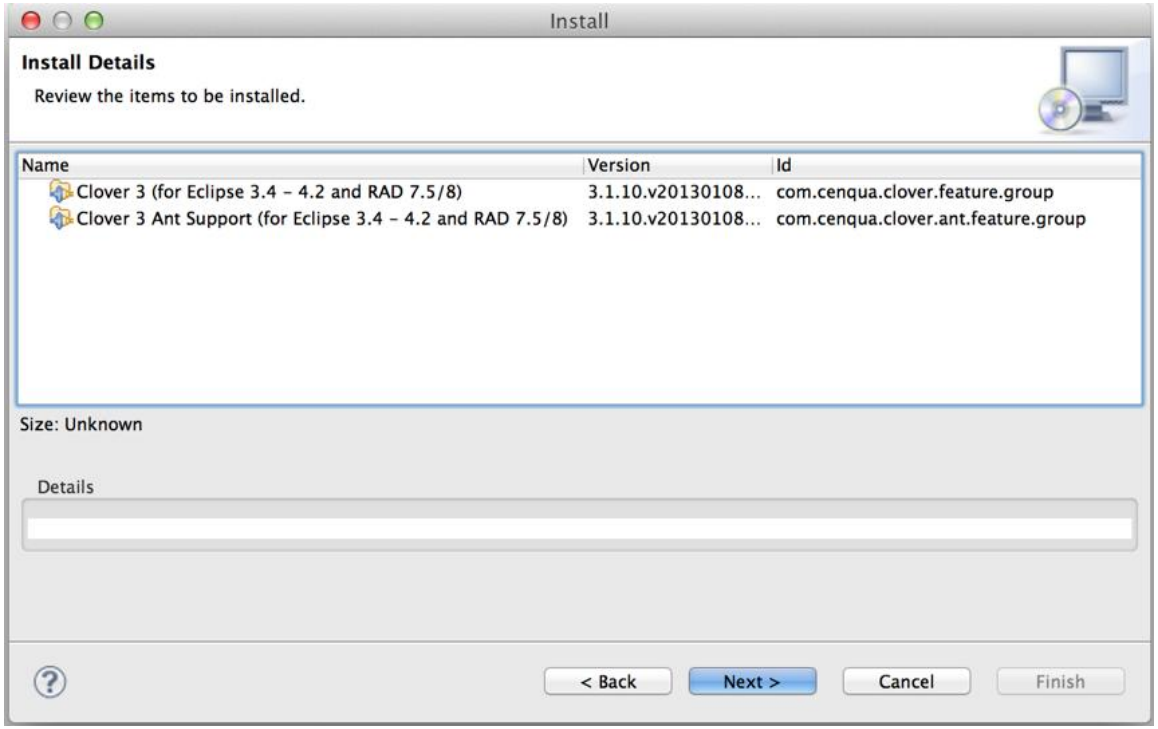
Clover를 선택하고 Next버튼을 Click!



Clover

How to install Clover?

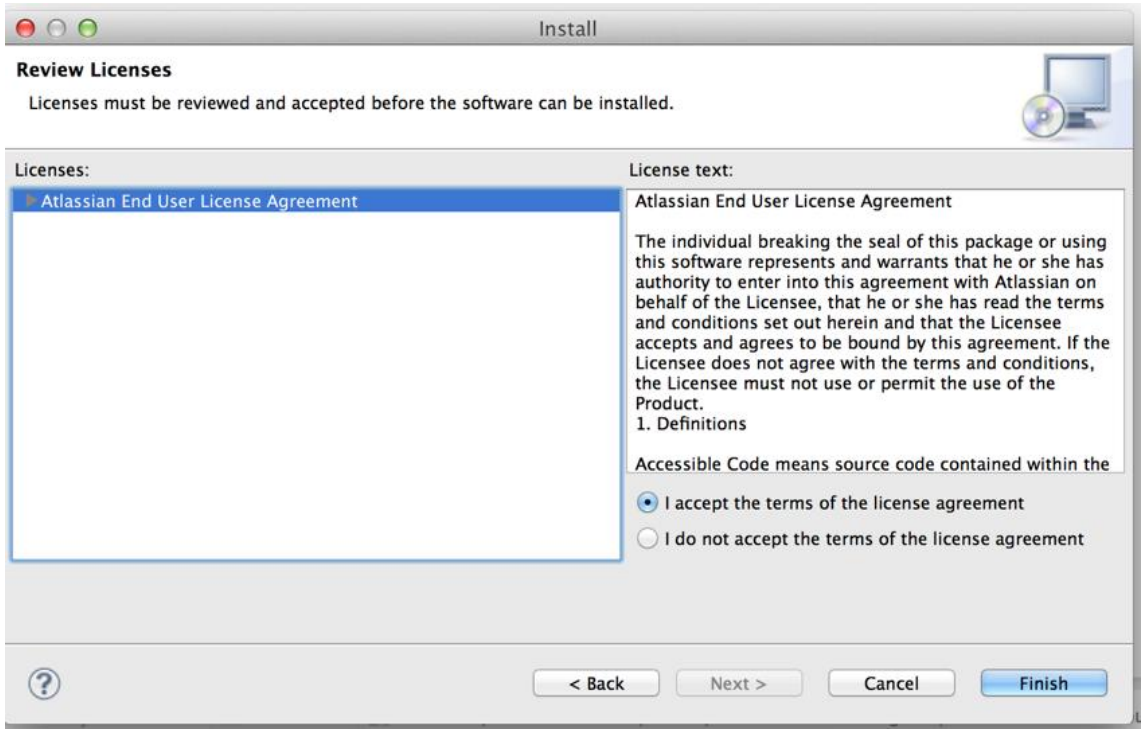
제대로 Clover설치 하는지 패키지를 보고 Next버튼을 Click!



Clover

How to install Clover?

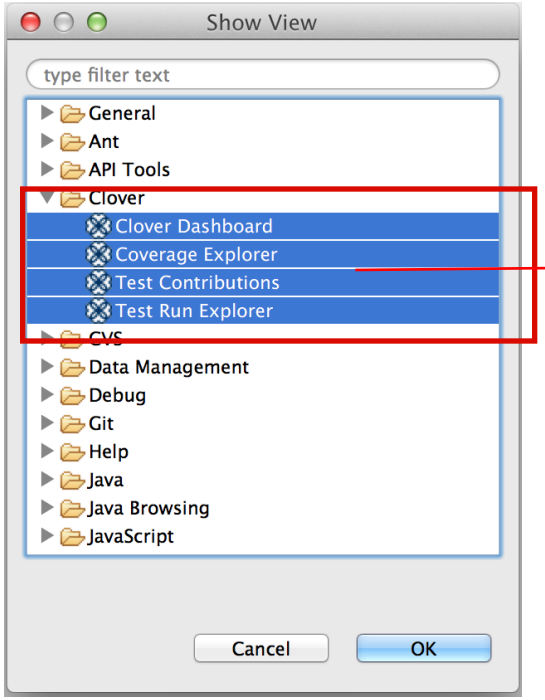
License 동의를 하고 Finish버튼을 Click!



Clover

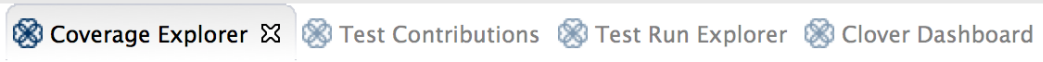
How to show Clover interface?

Eclipse menu에서 Window->Show View->Other... Click!



Clover 폴더에 있는 모든 Clover Interface를 선택하고 OK버튼을 Click!!

아래와 같이 탭이 생기면 OK..!

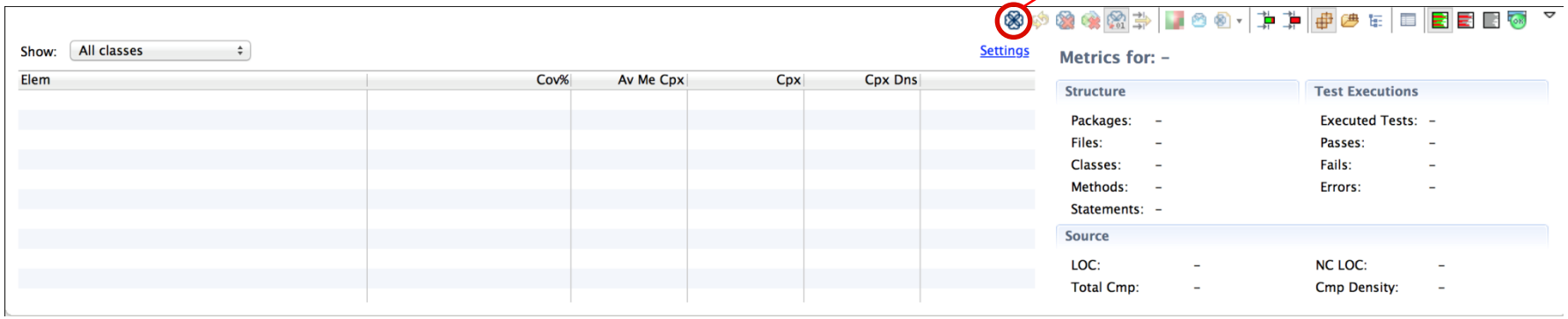


Clover

How to use Clover?

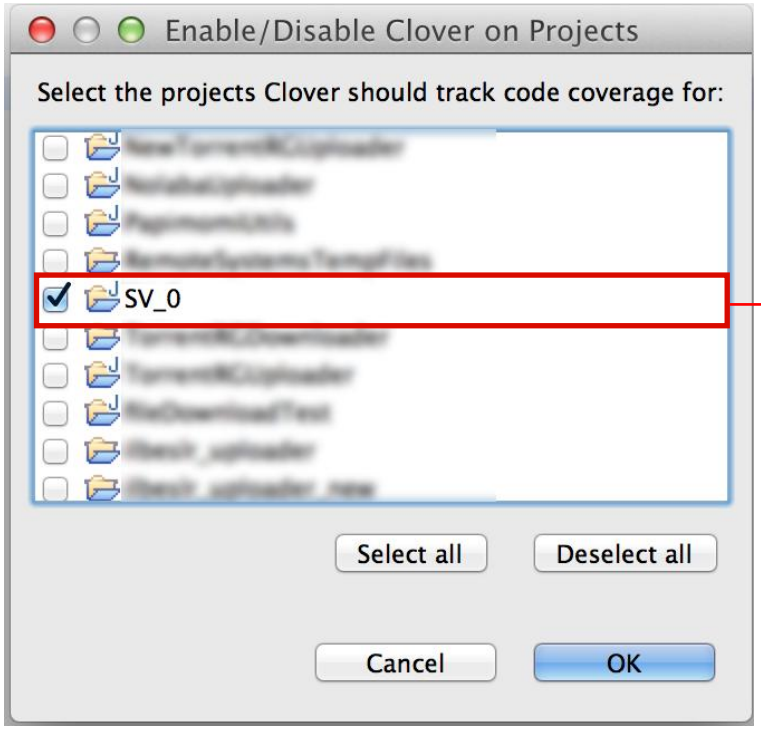
이 버튼을 Click한다

Coverage Explorer 탭을 선택하면 아래와 같이 나온다.



Clover

How to use Clover?



Code Coverage할 Project를 선택한다.

Clover

How to use Clover?

```
package myPack;
public class Calc {
    public int plus(int a, int b) {
        return a+b;
    }
    public int minus(int a, int b) {
        if(a > b) {
            return a-b;
        } else {
            return b-a;
        }
    }
    public int mul(int a, int b) {
        return a*b;
    }
    public int division(int a, int b) {
        return a*b;
    }
}
```

UnitTest Code

```
package UnitTest;
import static org.junit.Assert.assertArrayl
public class CalcTest {
    @Test
    public void testPlus() {
        Calc calc = new Calc();
        assertEquals(5, calc.plus(3, 2));
        assertEquals(10, calc.plus(5, 5));
    }
    @Test
    public void testMul() {
        Calc calc = new Calc();
        assertEquals(6, calc.mul(3, 2));
        assertEquals(6, calc.mul(2, 3));
    }
    @Test
    public void testMinus() {
        Calc calc = new Calc();
        assertEquals(1, calc.minus(3, 2));
    }
    @Test
    public void testDivision() {
        Calc calc = new Calc();
        assertEquals(2, calc.division(4, 2));
        assertEquals(3, calc.division(6, 2));
        assertEquals(8, calc.division(24, 3));
    }
}
```

Clover

How to use Clover?

Elem	Cov%	Av Me Cpx	Cpx	Cpx Dns
SV_0	0.0%	1.0	6.0	0.6
myPack	0.0%	1.0	4.0	1.0
Calc.java	0.0%	1.0	4.0	1.0
MainWindow.java	0.0%	-	0.0	-
UnitTest	0.0%	1.0	2.0	0.3
CalcTest.java	0.0%	1.0	2.0	0.3

UnitTest를 시작한다. 그럼 아래와 같이 테스트 코드가 얼마나 Cover됐는지 Percentage로 나타난다.

Elem	Cov%	Av Me Cpx	Cpx	Cpx Dns
SV_0	75.0%	1.0	6.0	0.6
The name of the Java element with code coverage	50.0%	1.0	4.0	1.0
Calc.java	50.0%	1.0	4.0	1.0
UnitTest	100.0%	1.0	2.0	0.3
CalcTest.java	100.0%	1.0	2.0	0.3

Clover

How to use Clover?

```
package myPack;
public class Calc {
2   public int plus(int a, int b) {
2       return a+b;
2   }
1   public int minus(int a, int b) {
1       if(a > b) {
1           return a-b;
1       } else {
0           return b-a;
1       }
1   }
2   public int mul(int a, int b) {
2       return a*b;
2   }
1   public int division(int a, int b) {
1       return a*b;
1   }
}
```

좌측과 같이 테스트가 통과한 부분은 초록색으로 나타난다.

테스트 시에 검사되지 않은 부분은 빨간색으로 나타난다.

노란색은 unit test가 failure된 부분을 말한다

Clover

How to use Clover?

Green	Coverage from passing tests or due to execution outside tests (e.g. main() method).
Yellow	Failed test coverage (where coverage has only been caused by one or more failing tests and no passing tests).
Grey	Filtered out code.
Red	Code with no coverage.
Squiggly Red Lines	Partial branch coverage (caused when only one part of a branch has been covered).

Clover

What is Clover Report?

코드가 Complexity하고 low coverage하면 높은 risk에 직면하게 될 수 있다. 그걸 알아보는 것이 바로 Report이다.

Clover에서 제공해주는 Report를 보고 프로젝트의 안정성을 확인 할 수 있다.

Clover에선 Cloud Report, Treemap Report를 기본적으로 지원하며 HTML, PDF, XML로 Report를 작성하여 공유할 수 있도록 해준다.

Clover

What is Clover Report?

Coverage Cloud Reports

복잡성(Complexity)은 크지만 적게 Coverage된 코드를 해결하는 데 가장 좋은 방법이다.

Package Risks와 Quick Wins항목이 있다.

Coverage가 되지 않거나 Complexity한 클래스를 쉽게 찾아서 코드를 열어 확인할 수 있다.

Clover

What is Clover Report?

Coverage Cloud Reports - Package Risks

가장 복잡하고 커버가 적은 클래스를 강조해준다.
-> Sub-Package의 class를 포함하거나 제외시킬수도 있다.

Metric	Attribute
Average Method Complexity	Font Size
% Coverage	Font Color

Clover

What is Clover Report?

Coverage Cloud Reports - Quick Wins

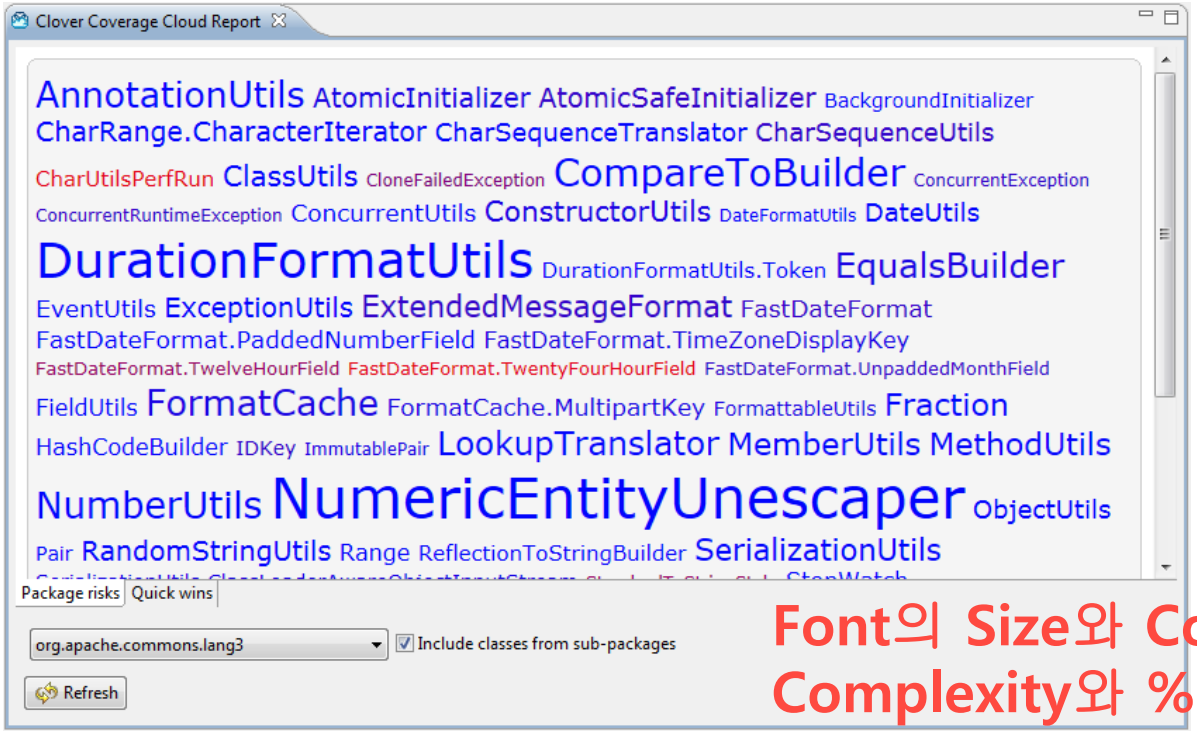
리스크가 가장 큰 클래스 순으로 강조해준다.
-> Sub-Package의 class를 포함하거나 제외시킬수도 있다.

Metric	Attribute
Number of Elements	Font Size
Number of Elements Untested	Font Color

Clover

What is Clover Report?

Coverage Cloud Reports



Font의 Size와 Color를 통해서
Complexity와 % Coverage를 알 수 있다.

Clover

What is Clover Report?

Coverage Treemap Reports

복잡성과 Coverage된 Project 또는 Package를 쉽게 볼 수 있다.

Package(labeled)로 나누고 그 안에 class(unlabeled)로 쪼개져서 보여준다.

Package 또는 Class의 크기는 복잡성을 나타낸다.

색깔로 code coverage의 정도를 나타낸다.

Clover

What is Clover Report?

Coverage Treemap Reports

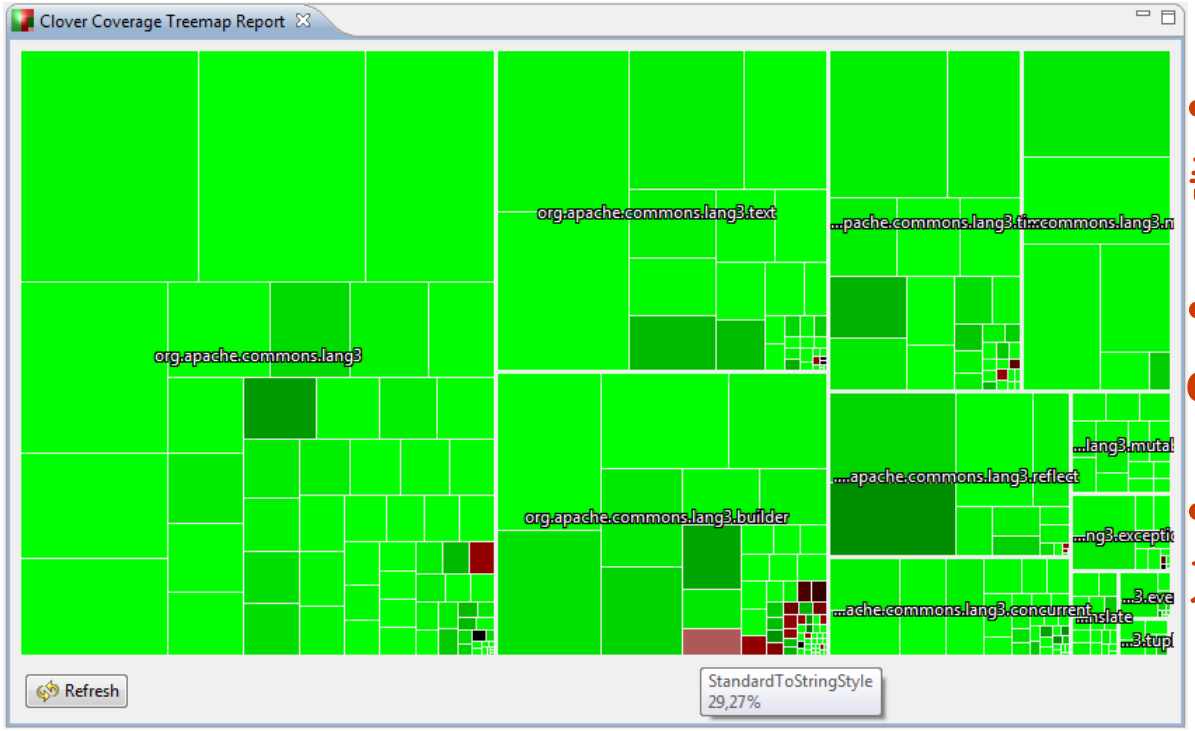
Treemap에 보이는 사각형 박스 색깔의 의미

Bright green (most covered)
Dark green (more coverage)
Black (around 50% coverage)
Dark Red (little coverage)
Bright Red (uncovered)

Clover

What is Clover Report?

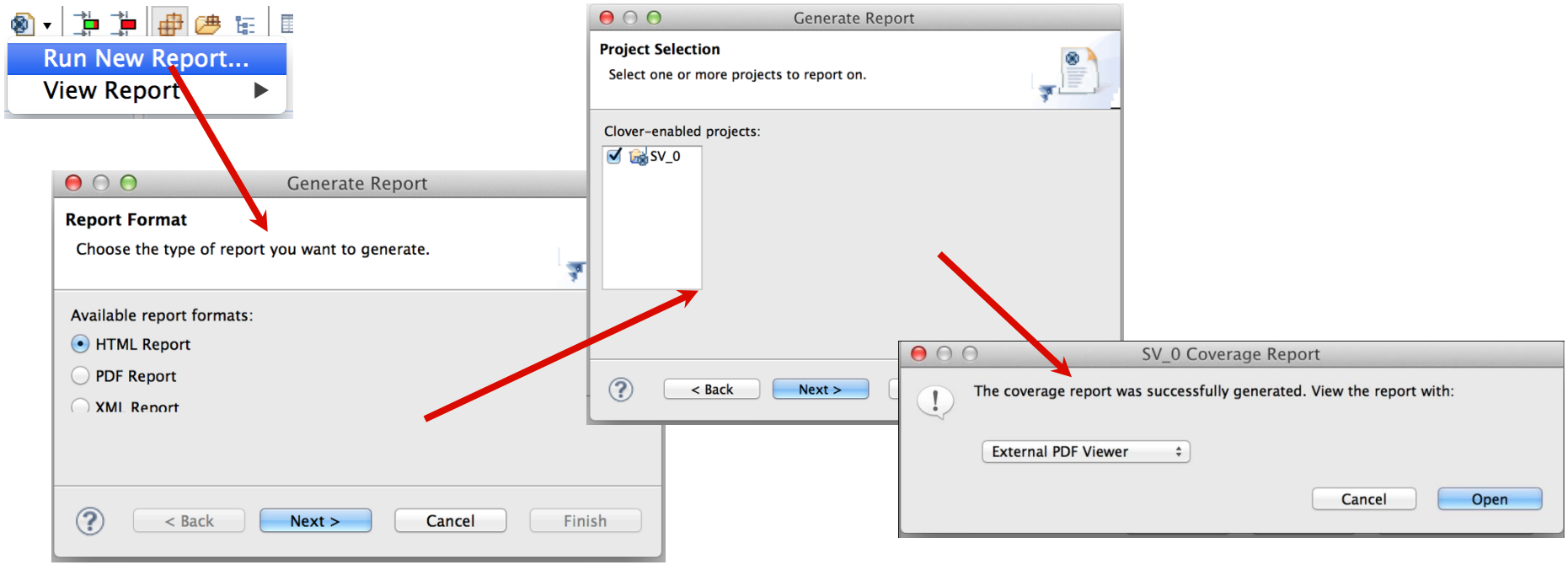
Coverage Treemap Reports



- **Package**별로 출력을 해 준다.
- 각 사각형의 크기는 **complexity**함을 보여줌
- 클래스의 **label**을 출력하지 않는다.

Clover

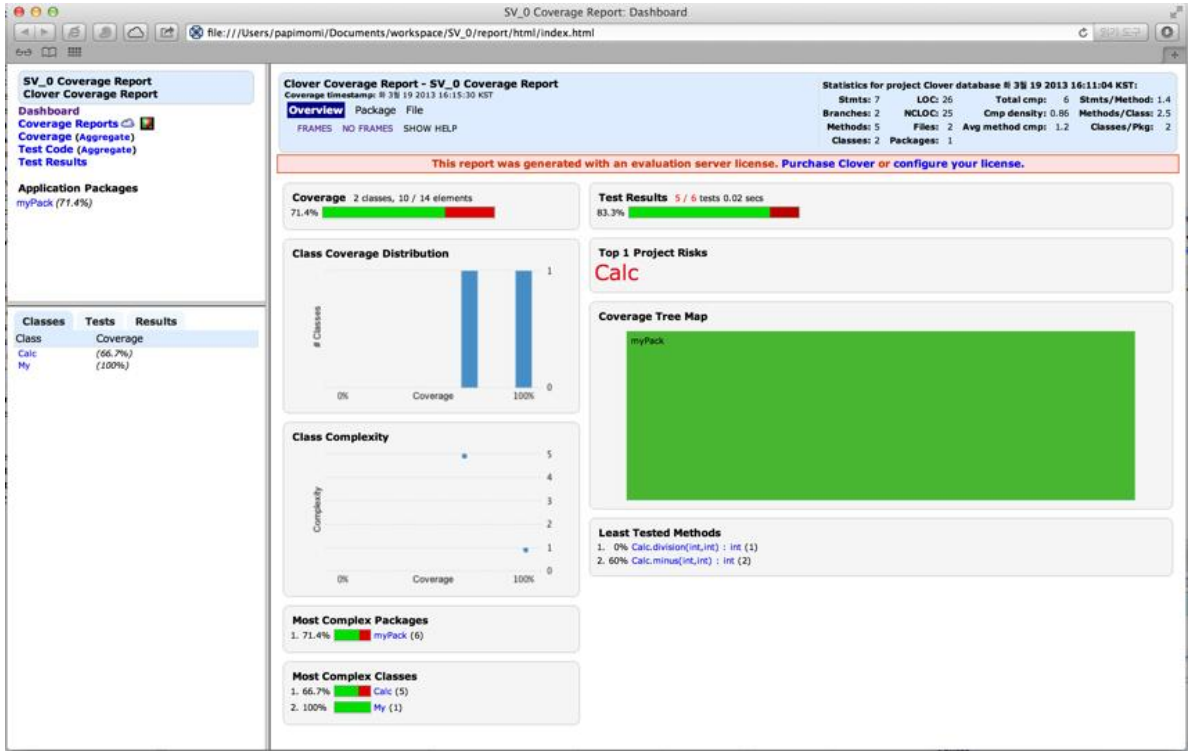
What is Clover Report?



Clover

What is Clover Report?

About Generated Reports



JDepend

JDepend

JDepend

What is Cycle Dependency?

Cycle dependencies are natural in many domain models where certain objects of the same domain depend on each other.

→ Cycle dependencies는 소프트웨어 프로그램 내에서 원치 않는 문제를 일으킬 수 있다. 예를 들어, Domino effect와 memory leask 등이 있다.

In very large software designs, software engineers may lose the context and inadvertently introduce circular dependencies. There are tools to analyze software and find unwanted circular dependencies

→ JDpend를 사용하여 Cycle Dependency를 줄이겠다.

JDepend

Benefits of JDepend

The goal of using JDepend is to ultimately invert package dependencies such that low-abstraction packages depend upon high-abstraction packages.

⇒ JDepend를 통해 Cycle dependency를 발견하여 design quality 높일 수 있다.

Package dependency cycles can be easily identified by reviewing the textual reports of dependency cycles.

⇒ Package간의 cycle dependency를 확인할 수 있다.

JDepend

JDepend is not perfect!

Cyclic dependency detection may not report all cycles reachable from a package.

Java interfaces are treated as equals with Java abstract classes.

⇒ JDepend를 통해 모든 Cycle dependency를 발견하지 못할 수도 있다.

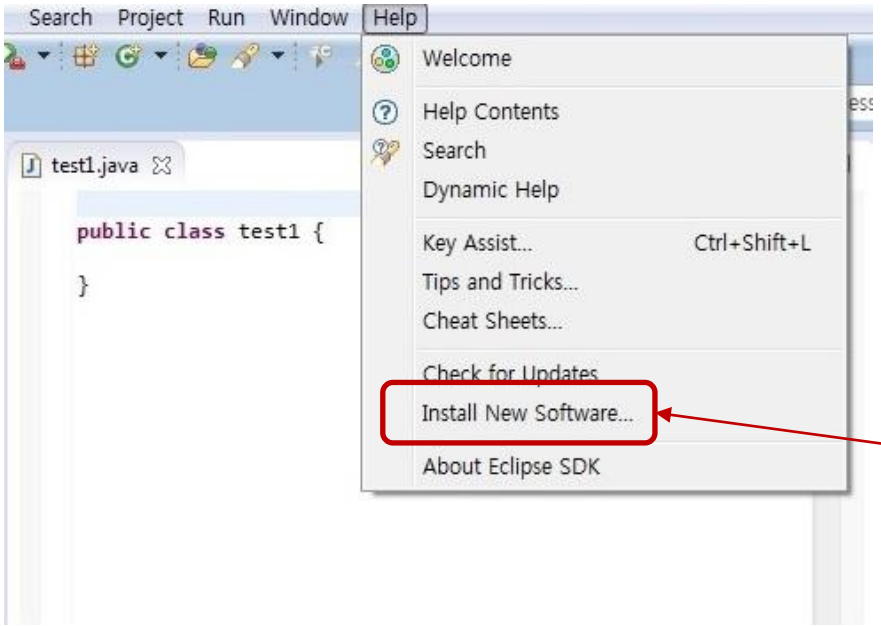
The design quality metrics generated by JDepend are imperfect.

The JDepend does not collect source code complexity metrics.

⇒ JDepend의 결과만으로 좋은 디자인을 만들 수 있는 것은 아니다.

JDepend

Install



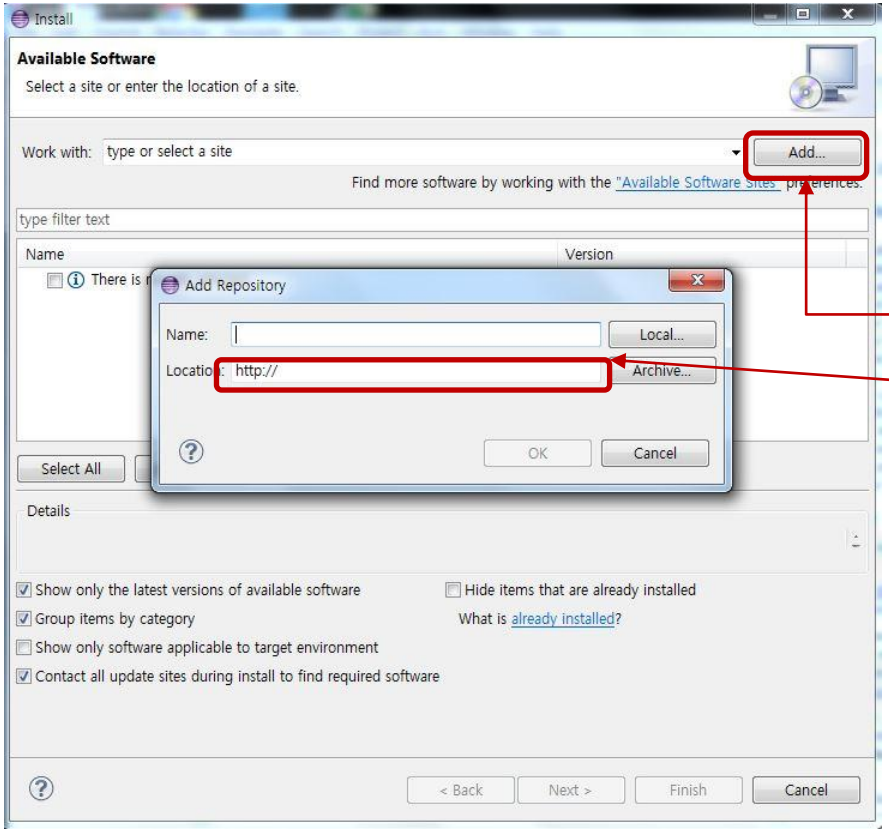
Step1

상단의 'Help'버튼을 클릭

Install New Software... 버튼 클릭

JDepend

Install



Step2

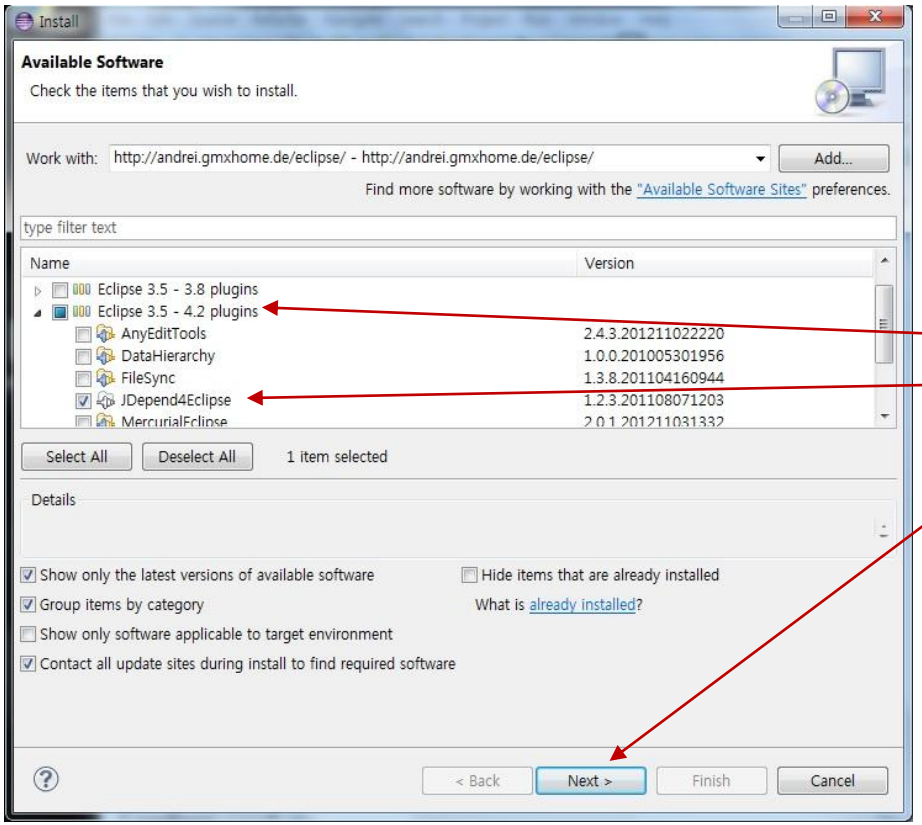
'Add'버튼 클릭.

주소창에 다음 주소를 입력:

http://andrei.gmxhome.de/eclipse

JDepend

Install



Step3

Eclipse 3.5 – 4.2 plugins 클릭

JDpence4eclipse체크

'Next' 버튼을 클릭.

JDepend

Index and Analysis

CC - (Concrete Class Count)
The number of concrete classes in the package.

AC - (Abstract Class Count)
The number of abstract classes (and interfaces) in the package

Ca - (Afferent Couplings)
The number of other packages that depend upon classes within the package is an indicator of the package's responsibility.

현재 Package의 클래스들에 종속성을 가지는 Package의 개수를 나타낸다.
(이 Package를 참조하는 Package의 수)

JDepend

Index and Analysis

Ce - (Efferent Couplings)

The number of other packages that the classes in the package depend upon is an indicator of the package's independence.

현재 Package의 클래스들이 종속하고 있는 Package의 개수를 나타낸다.
(이 Package가 참조하는 Package의 수)

A - (Abstractness)

The ratio of the number of abstract classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package.

0~1사이의 값을 가지며 1에 가까울 수록 추상적인 Package를 나타낸다.

JDepend

Index and Analysis

I - (Instability)

The ratio of efferent coupling (Ce) to total coupling (Ce + Ca) such that I = Ce / (Ce + Ca). This metric is an indicator of the package's resilience to change.

→ 0~1사이의 값을 가지며 1에 가까울 수록 불안정한 Package를 나타낸다.

Cycle (If the package contains a dependency cycle)

Package dependency cycles are reported along with the hierarchical paths of packages participating in package dependency cycles.

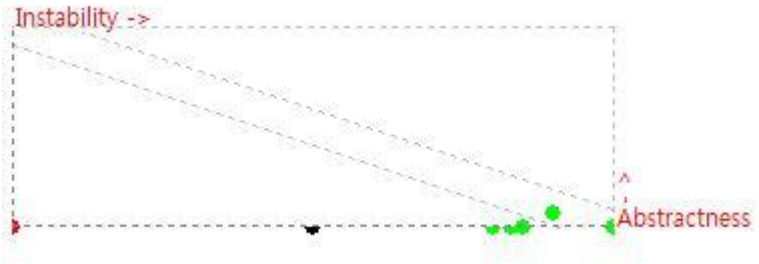
→ Package dependency가 발견 될 경우 발생한다.

JDepend

Index and Analysis

D(Distance from the Main Sequence)

The perpendicular distance of a package from the idealized line A + I = 1. This metric is an indicator of the package's balance between abstractness and stability.

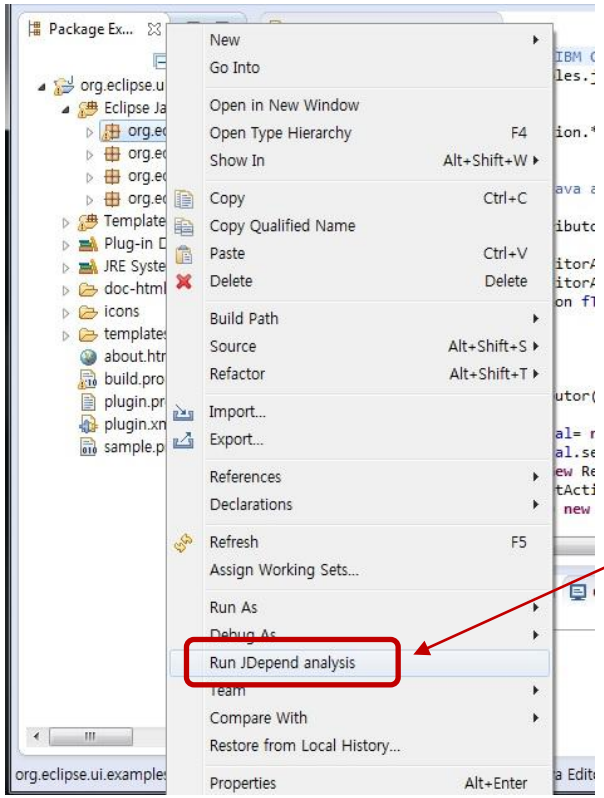


Main Sequence는 이상적인 Package로 추상화가 잘 이루어져 있으며 안정적인 Package를 의미한다.

다음 왼쪽의 결과 그래프가 D(Distance from the Main Sequence)를 의미한다.

JDepend

How to run JDepend?

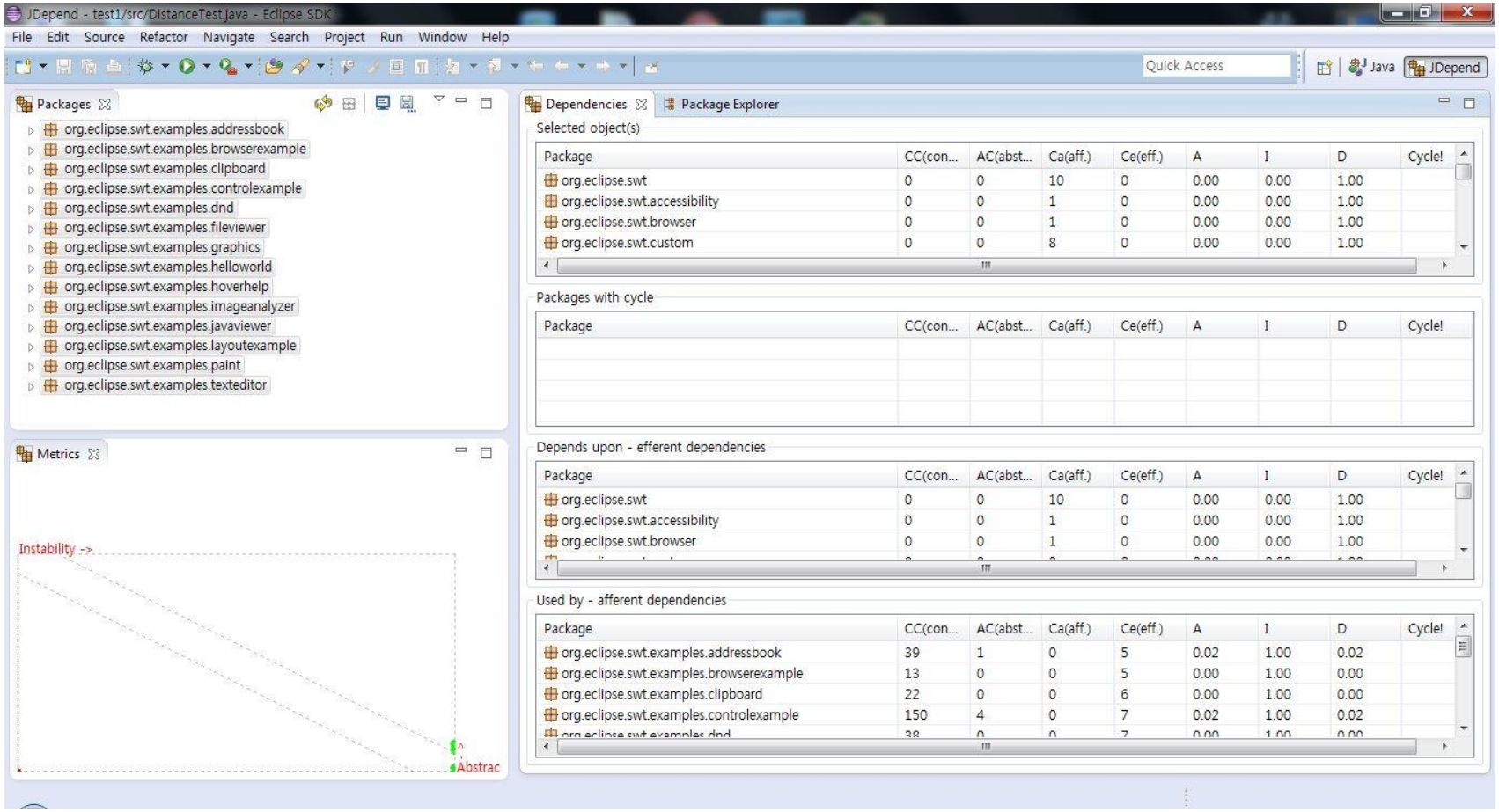


JDepend로 분석하고자 하는 소스 파일 또는 프로젝트 파일을 '우클릭'

Run JDepend analysis를 클릭.

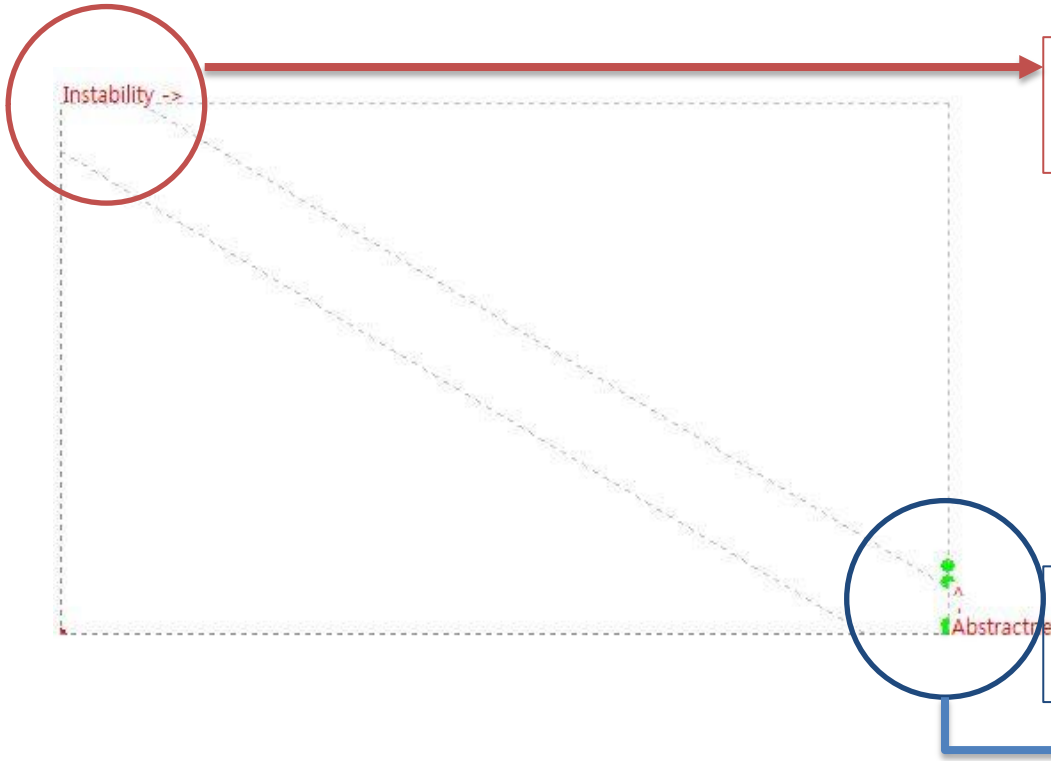
JDepend

How to run JDepend?



JDepend

Example

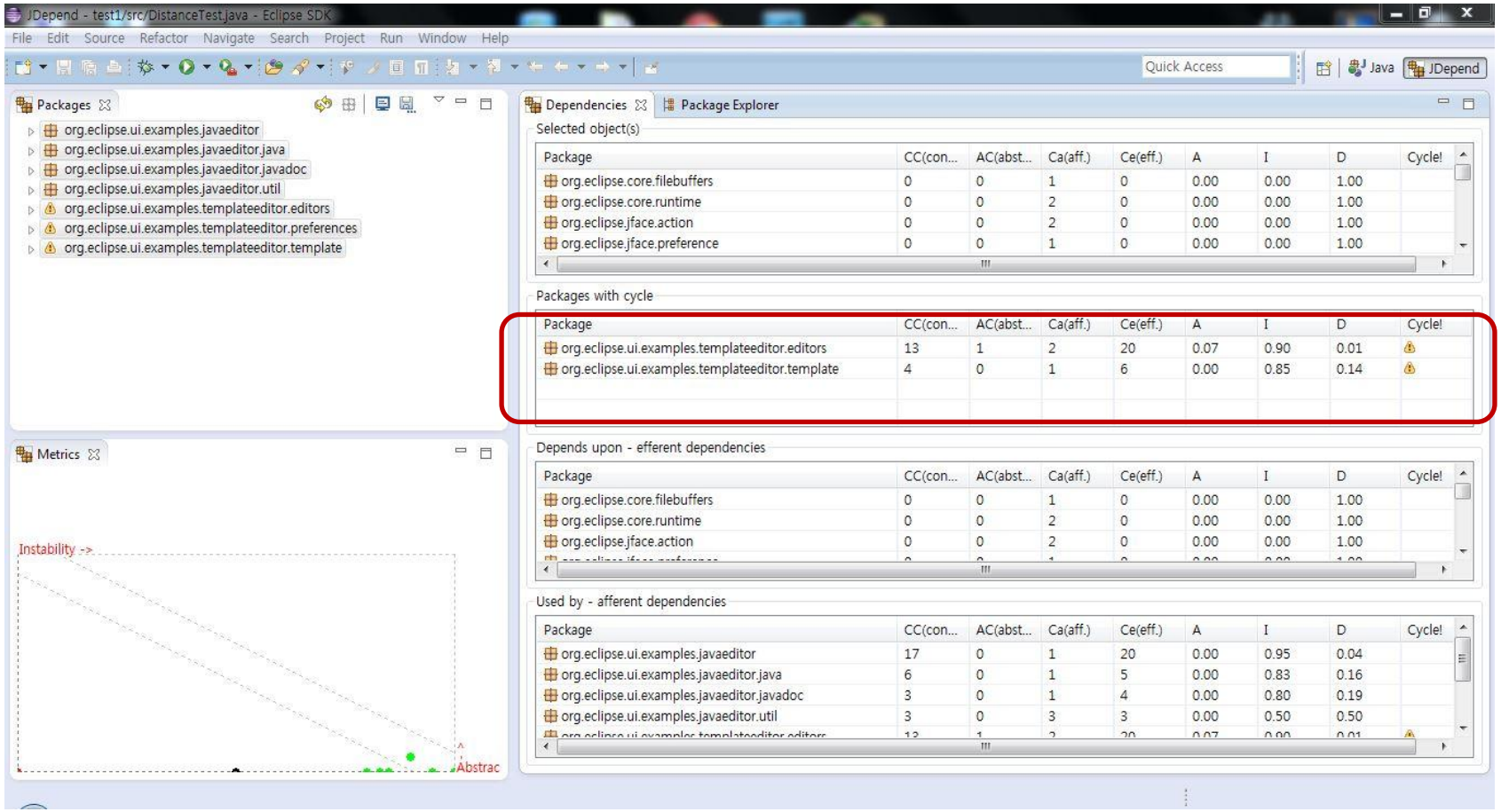


불안정하며 추상화가 거의 이루어지지 않은 Package

안정적이며 추상화가 잘 이루어진 Package

JDepend

Example



JDepend

Example

이전 예제와 달리 Main sequence에서 떨어진 Package 들을 발견할 수 있다.

Reference

Reference

Reference

Eclipse

http://en.wikipedia.org/wiki/Eclipse_(software)

http://www.eclipse.org/downloads/

Reference

JUnit

http://junit.org/

https://github.com/junit-team/junit/wiki

http://en.wikipedia.org/wiki/Unit

http://en.wikipedia.org/wiki/Junit

http://junit.sourceforge.net/doc/faq/faq.htm#tests_4

피터 타치브, [Junit in Action], (인사이트, 2011)

이상민, [자바 개발자도 쉽고 즐겁게 배우는 테스팅 이야기], (한빛미디어, 2009)

유석문 외 9인, [NHN은 이렇게 한다! 소프트웨어 품질관리], (위키북스, 2011)

채수원, [테스트 주도 개발 TDD 실천법과 도구], (한빛미디어, 2010)

Reference

Clover

https://confluence.atlassian.com/display/Clover/About+Code+Coverage

http://en.wikipedia.org/wiki/Coverage

Reference

JDepend

http://www.clarkware.com/software/JDepend.html#uses

http://en.wikipedia.org/wiki/Circular_dependency

http://www.onjava.com/pub/a/onjava/2004/01/21/jdepend.html

http://blog.benelog.net/2208368

Q & A

Thank You!!