

# SOFTWARE VERIFICATION

## Tool Analysis

2005 | 1342 | 이도현  
2006 | 1000 | 안병욱  
2007 | 10069 | 전창완  
2009 | 2432 | 김다영

# INDEX



- | Index
- | Eclipse
- | JUnit
- | JDepend
- | Clover

# Eclipse

# Eclipse

## | Eclipse 소개

/ 1999년 시작된 Java Project

/ 2001년 IBM 오픈소스프로젝트

/ 자바로 작성된 자유 소프트웨어 (EPL License)

/ 다양한 언어를 지원하는 통합 개발환경

# Eclipse

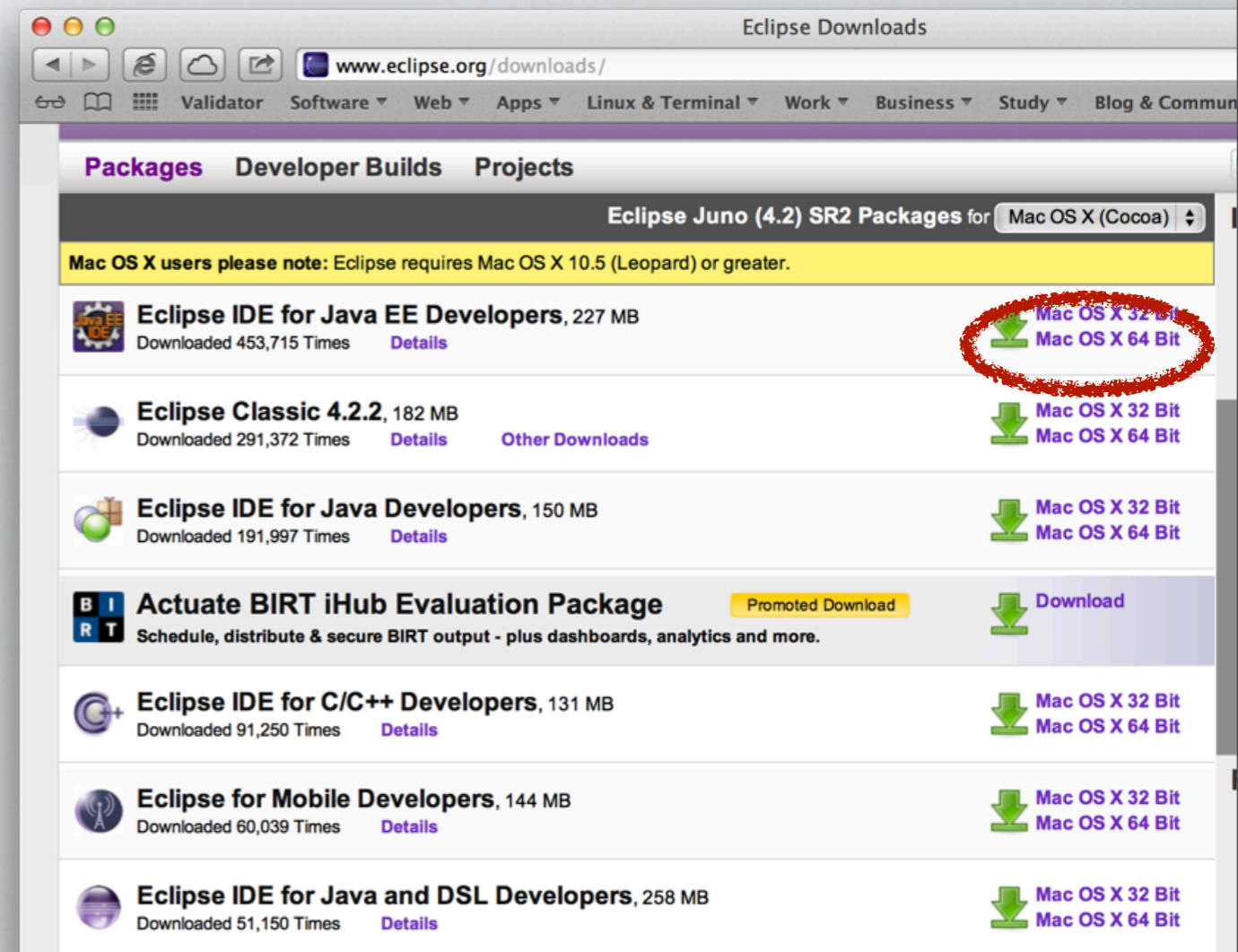
1 Eclipse 설치

/ <http://eclipse.org>

/ Windows 7 64 Bit 다운로드

/ Windows, C:\eclipse  
Mac, /Application/eclipse

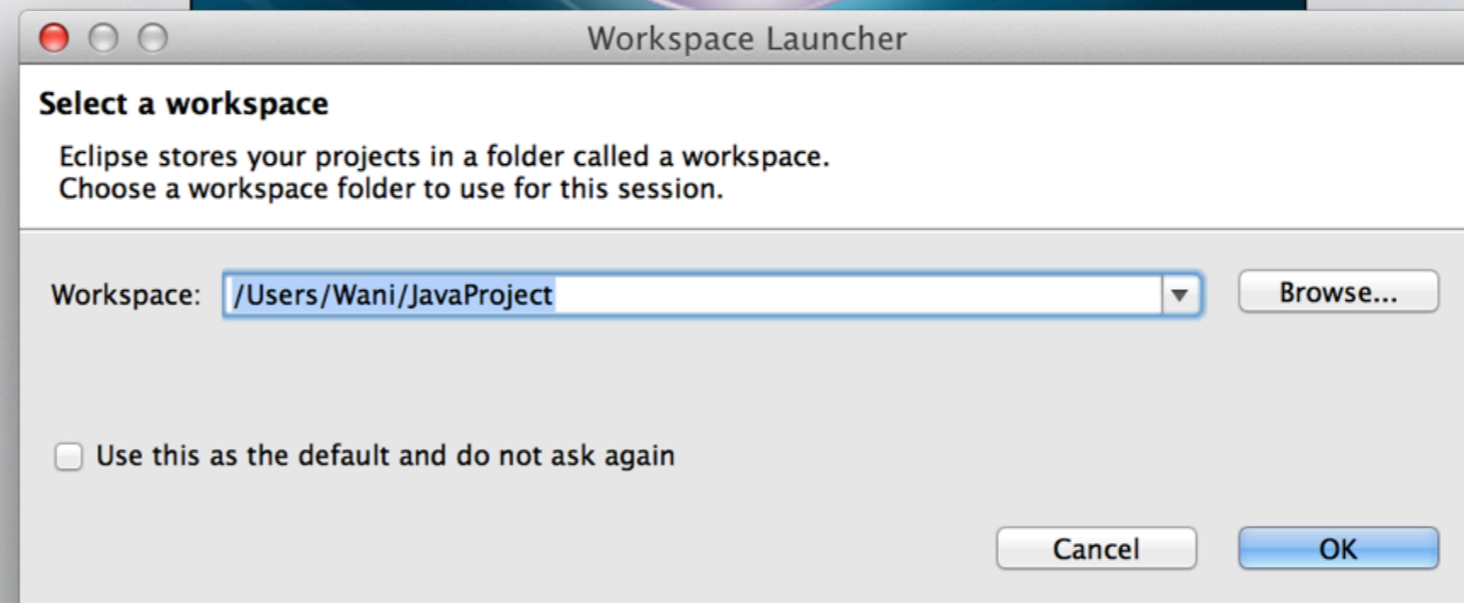
/ 압축해제



# Eclipse

| Eclipse 설치

/ Workspace 설정



# Eclipse

| Eclipse 설치

/ JDK 설치

/ <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Java SE Downloads

www.oracle.com/technetwork/java/javase/downloads/index.html

Validator Software Web Apps Linux & Terminal Work Business Study Blog & Co

Java Platform (JDK) 7u17 JavaFX 2.2.7 JDK 7 + N

Here are the Java SE downloads in detail:

**Java Platform, Standard Edition**

**Java SE 7u17**  
This release includes important security fixes. Oracle strongly recommends that all Java SE 7 users upgrade to this release.  
[Learn more](#)

**JDK**  
**DOWNLOAD**

**JDK 7 Docs**

- Installation Instructions
- ReadMe
- Release Notes
- Oracle License
- Java SE Products
- Third Party Licenses
- Certified System Configurations

**JDK 7 and JavaFX Demos and Samples**  
Demos and samples of common tasks and new functionality available on JDK 7. The source code provided with samples and demos for the JDK is meant to illustrate the usage of a given feature or technique and has been deliberately

**Demos and Samples**  
**DOWNLOAD**

7/91

# Eclipse

| Eclipse 설치

/ JDK 설치

/ OS에 맞춰서 다운로드

## Java SE Development Kit 7u17

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	106.65 MB	<a href="#">jdk-7u17-linux-i586.rpm</a>
Linux x86	92.97 MB	<a href="#">jdk-7u17-linux-i586.tar.gz</a>
Linux x64	104.78 MB	<a href="#">jdk-7u17-linux-x64.rpm</a>
Linux x64	91.71 MB	<a href="#">jdk-7u17-linux-x64.tar.gz</a>
Mac OS X x64	143.78 MB	<a href="#">jdk-7u17-macosx-x64.dmg</a>
Solaris x86 (SVR4 package)	135.39 MB	<a href="#">jdk-7u17-solaris-i586.tar.Z</a>
Solaris x86	91.67 MB	<a href="#">jdk-7u17-solaris-i586.tar.gz</a>
Solaris SPARC (SVR4 package)	135.92 MB	<a href="#">jdk-7u17-solaris-sparc.tar.Z</a>
Solaris SPARC	95.32 MB	<a href="#">jdk-7u17-solaris-sparc.tar.gz</a>
Solaris SPARC 64-bit (SVR4 package)	22.97 MB	<a href="#">jdk-7u17-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	17.59 MB	<a href="#">jdk-7u17-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	22.61 MB	<a href="#">jdk-7u17-solaris-x64.tar.Z</a>
Solaris x64	15.02 MB	<a href="#">jdk-7u17-solaris-x64.tar.gz</a>
Windows x86	88.75 MB	<a href="#">jdk-7u17-windows-i586.exe</a>
Windows x64	90.42 MB	<a href="#">jdk-7u17-windows-x64.exe</a>

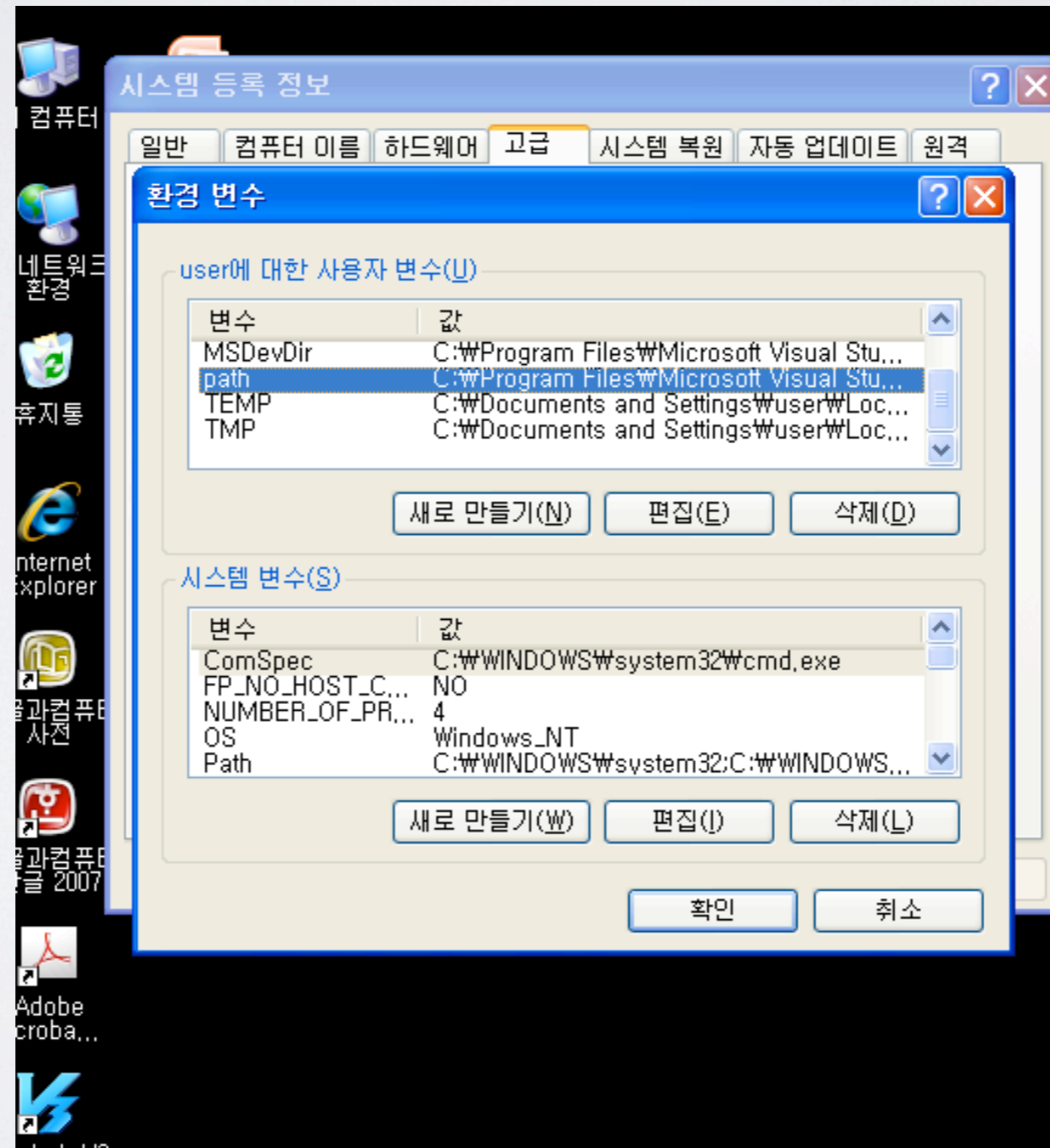


# Eclipse

┆ Eclipse 설치

┆ JDK 설치

┆ 환경변수설정

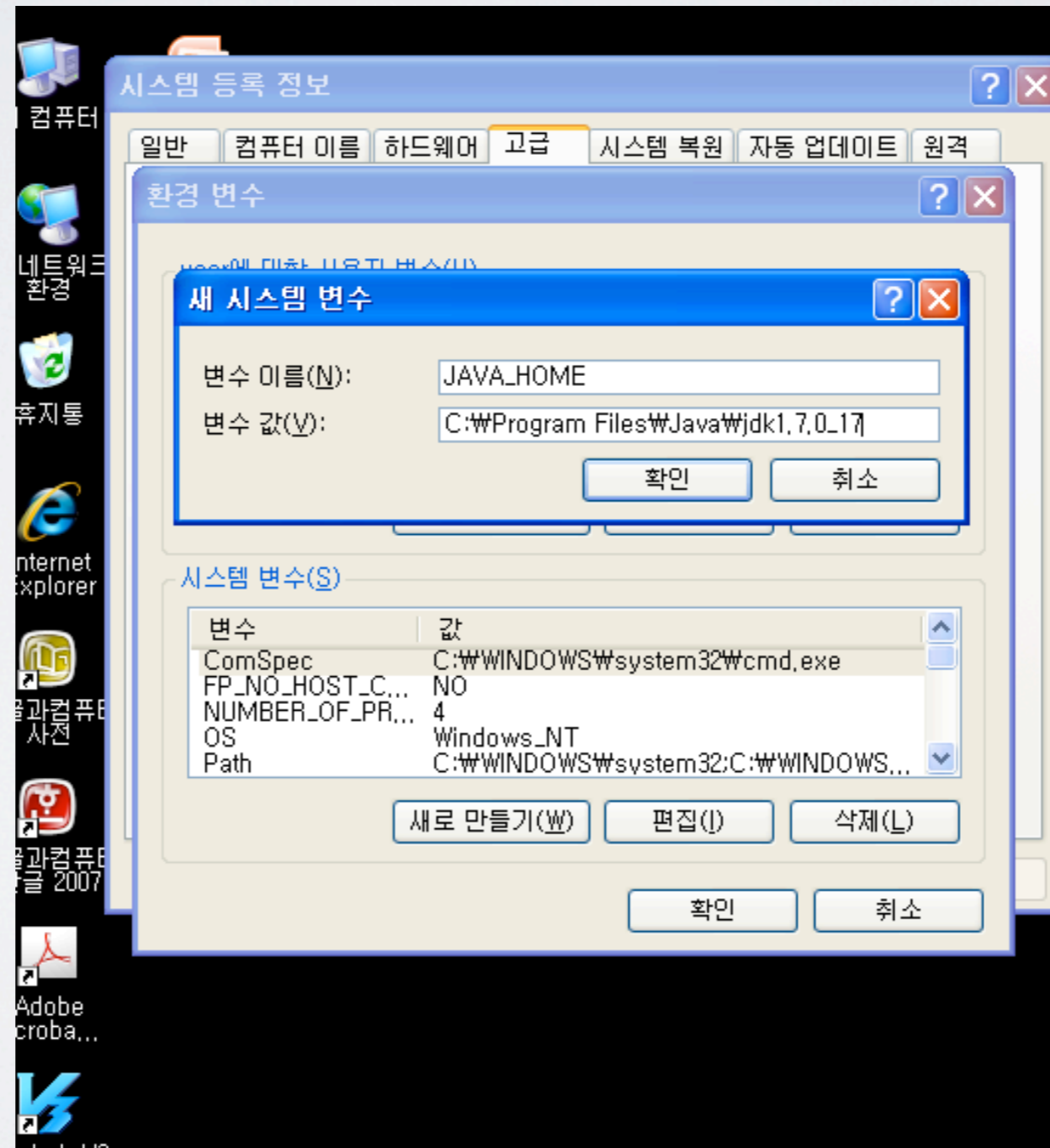


# Eclipse

┆ Eclipse 설치

┆ JDK 설치

┆ 환경변수설정

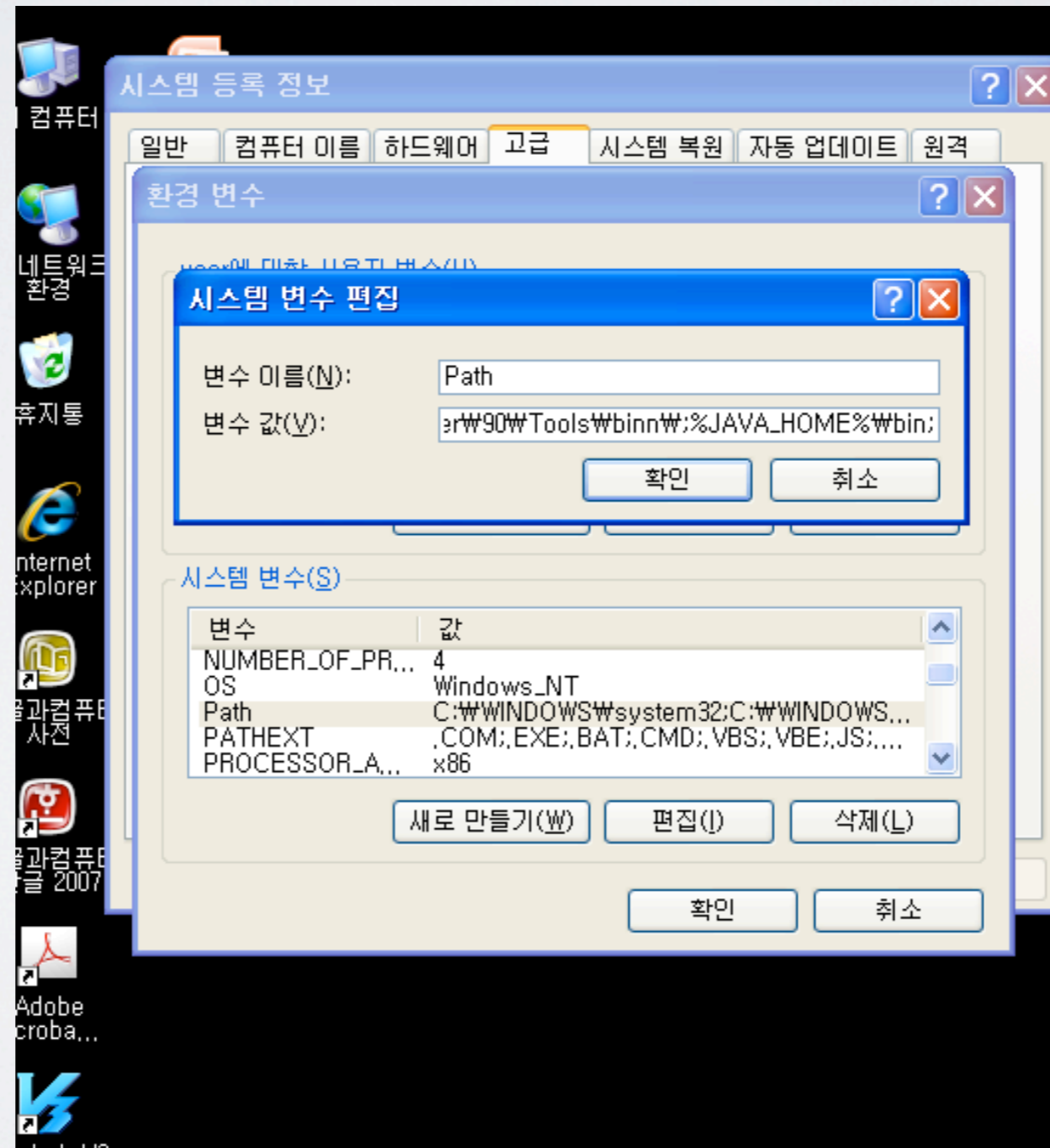


# Eclipse

· Eclipse 설치

· JDK 설치

· 환경변수설정

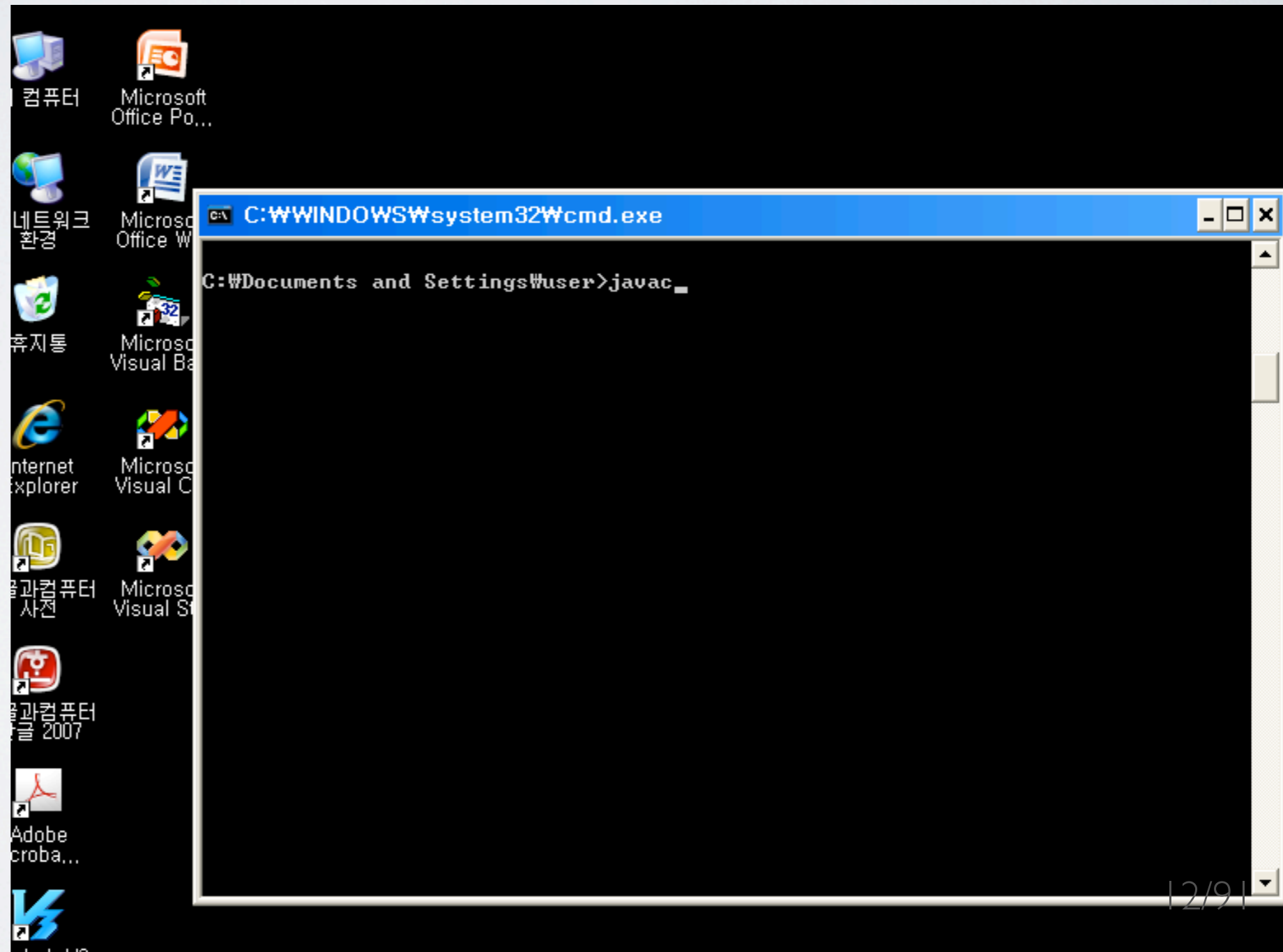


# Eclipse

| Eclipse 설치

/ JDK 설치

/ 환경변수설정

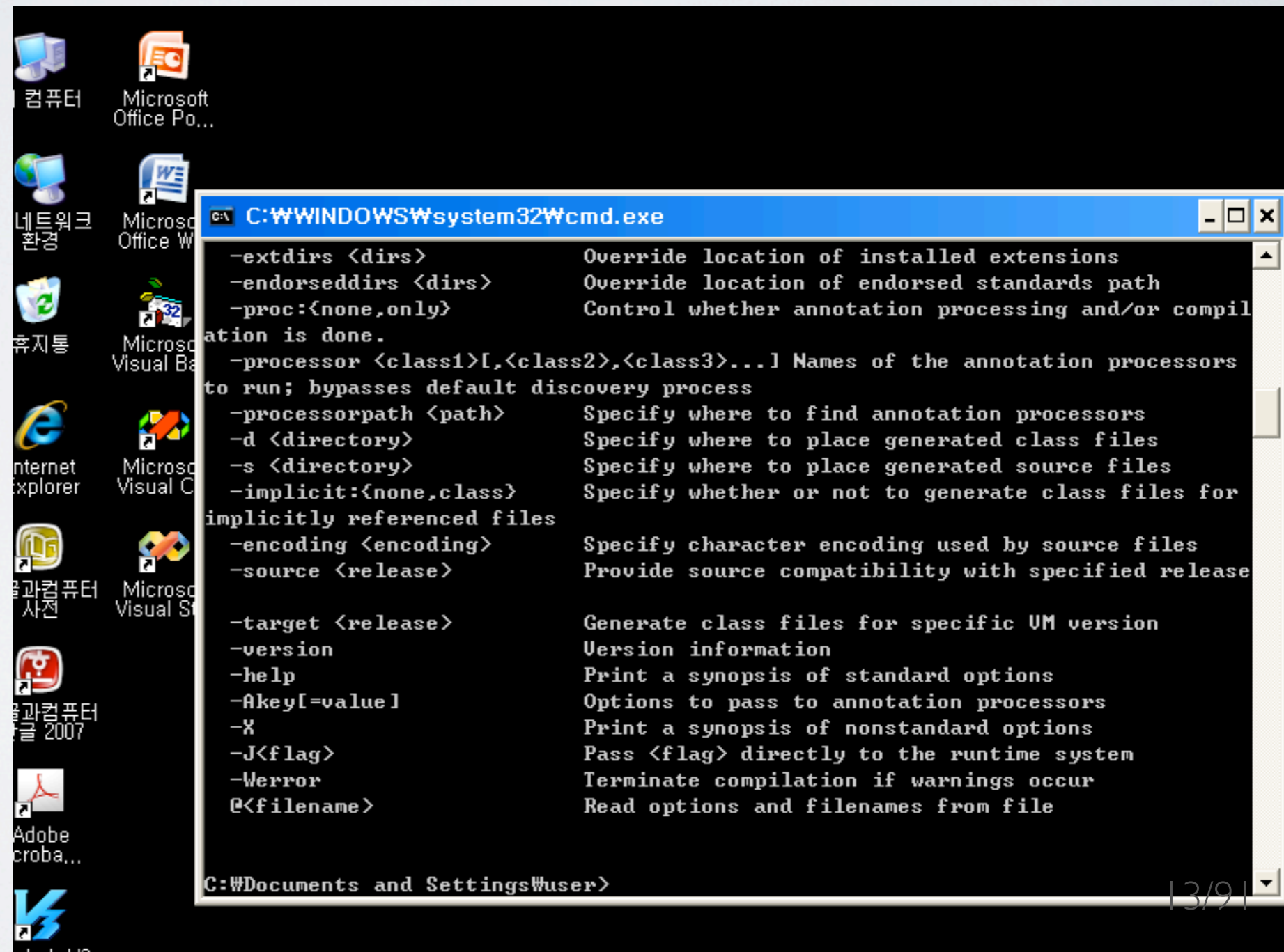


# Eclipse

| Eclipse 설치

/ JDK 설치

/ 환경변수설정



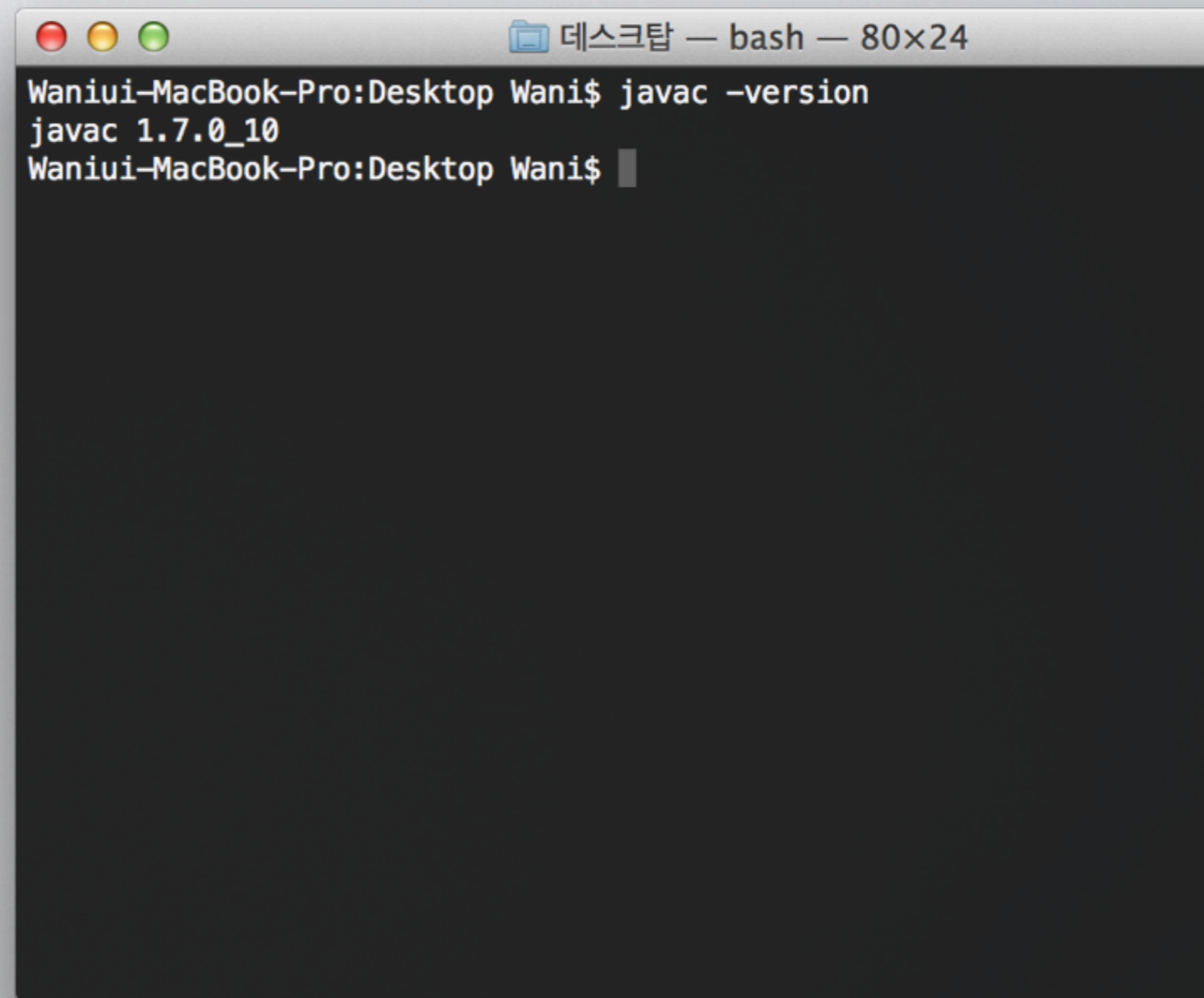
# Eclipse

## | Eclipse 설치

/ JDK 설치

/ 환경변수설정

/ Mac의 경우 자동설치

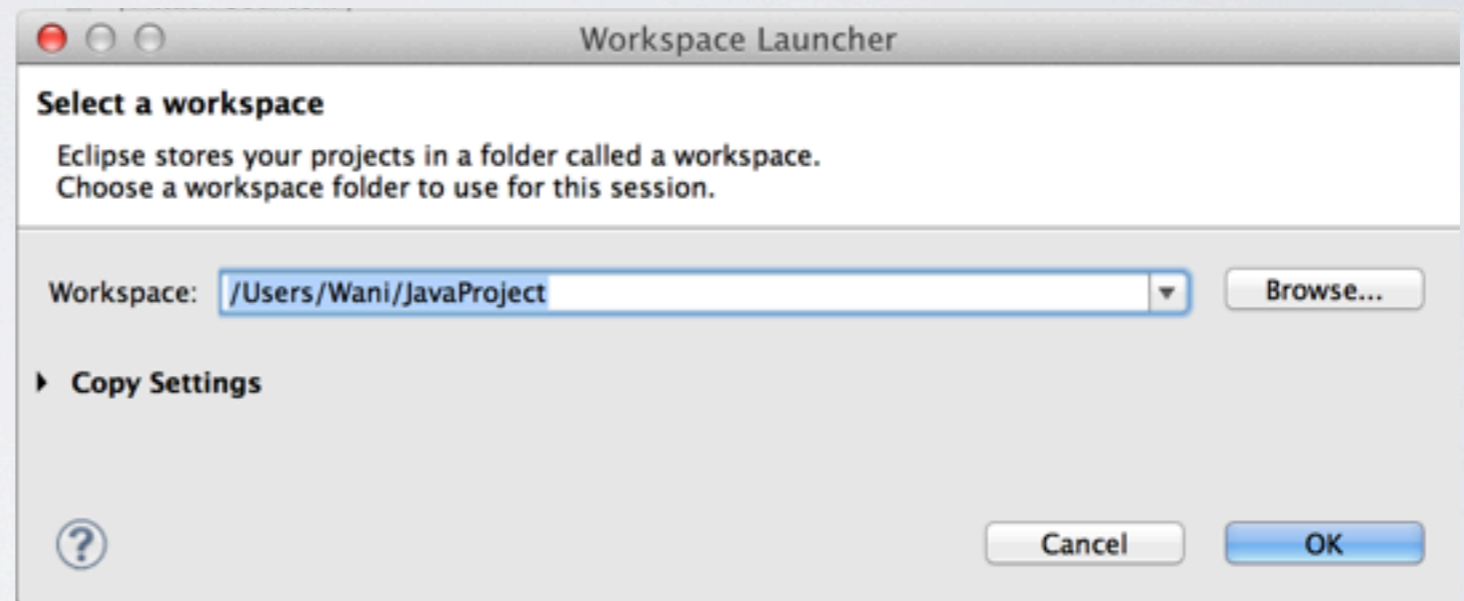


```
데스크탑 — bash — 80x24
Waniui-MacBook-Pro:Desktop Wani$ javac -version
javac 1.7.0_10
Waniui-MacBook-Pro:Desktop Wani$
```

# Eclipse

## ! Eclipse 사용법

/ 이클립스를 실행하면 처음에 Workspace를 설정하여야 한다.



/ 원하는 경로를 입력한 후 OK버튼을 누른다.

# Eclipse

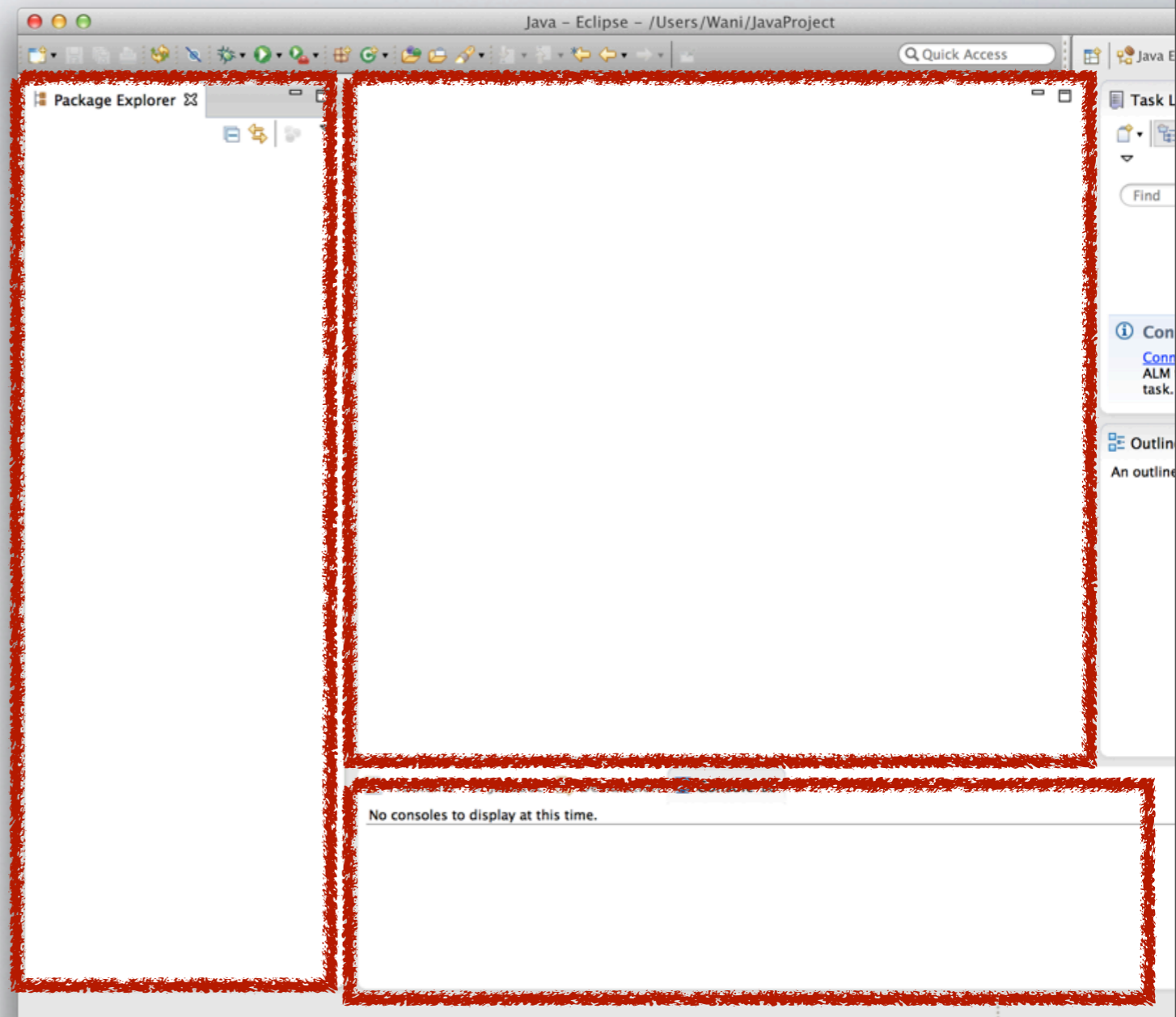
| Eclipse 사용법

/ Package Explorer

/ Editor

/ Console

/ 크게 3가지 영역으로 구성



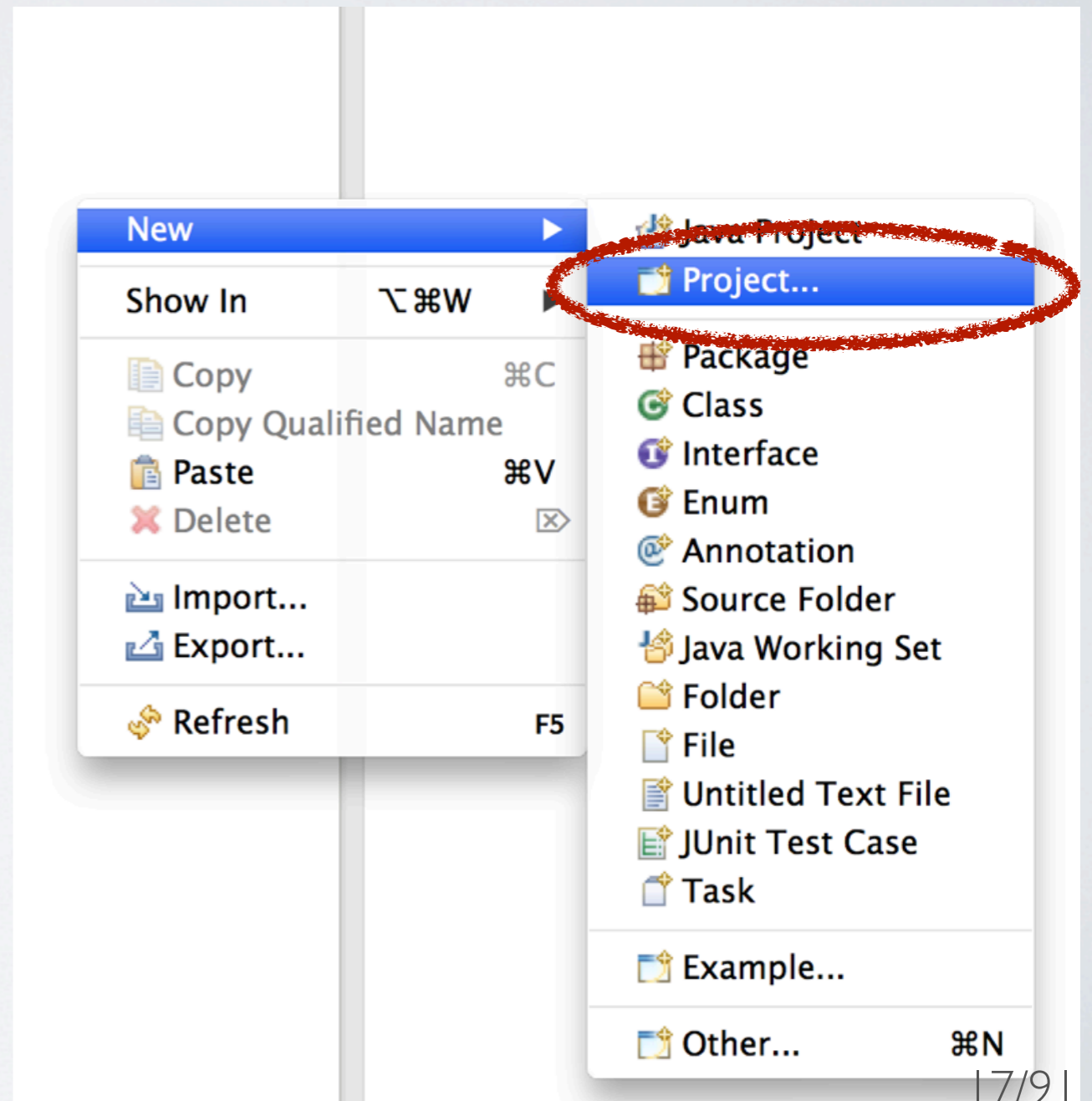


# Eclipse

l Eclipse 사용법

/ Package Explorer에서 우클릭,

/ New -> Project

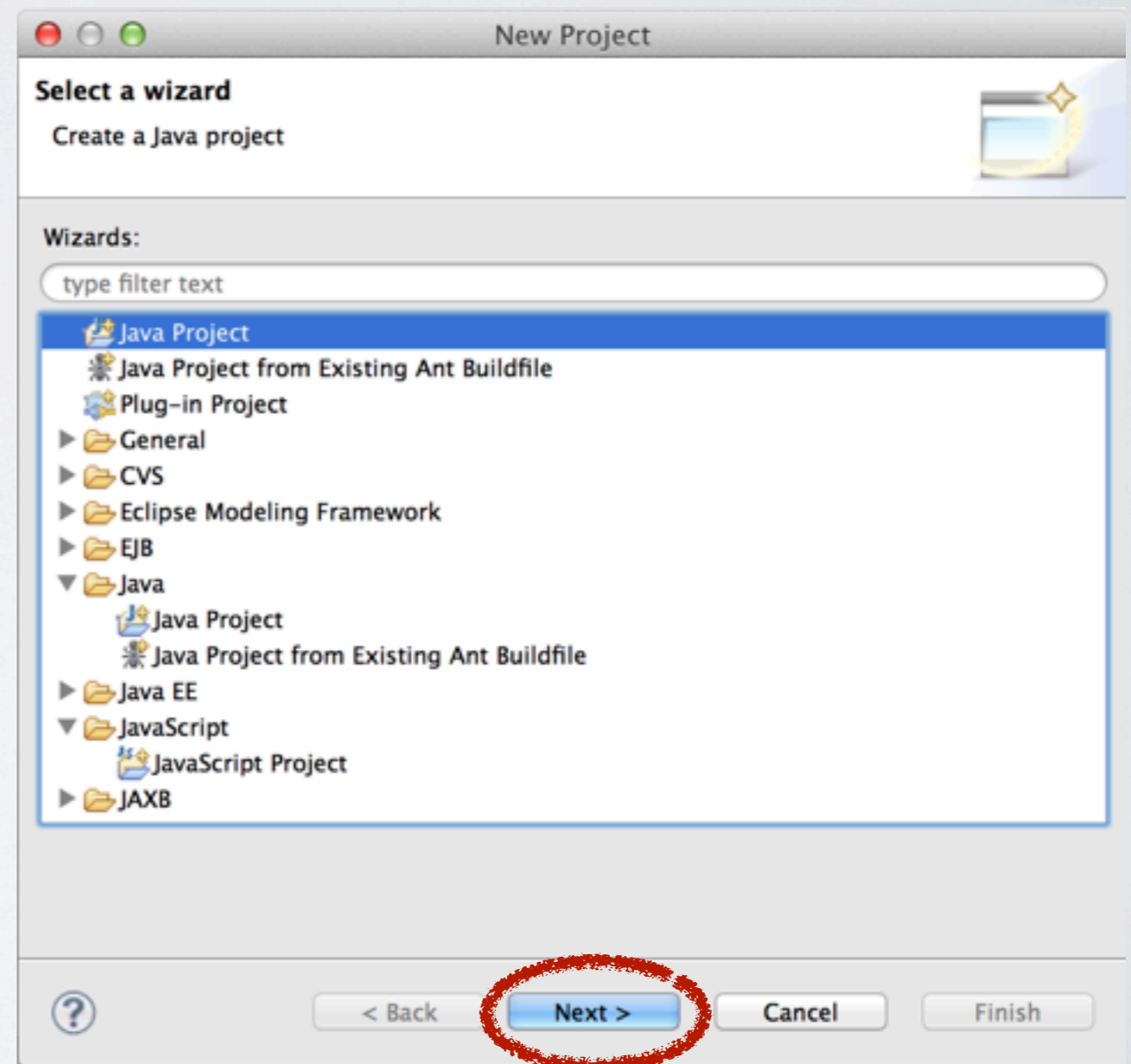


# Eclipse

| Eclipse 사용법

/ Java Project 선택

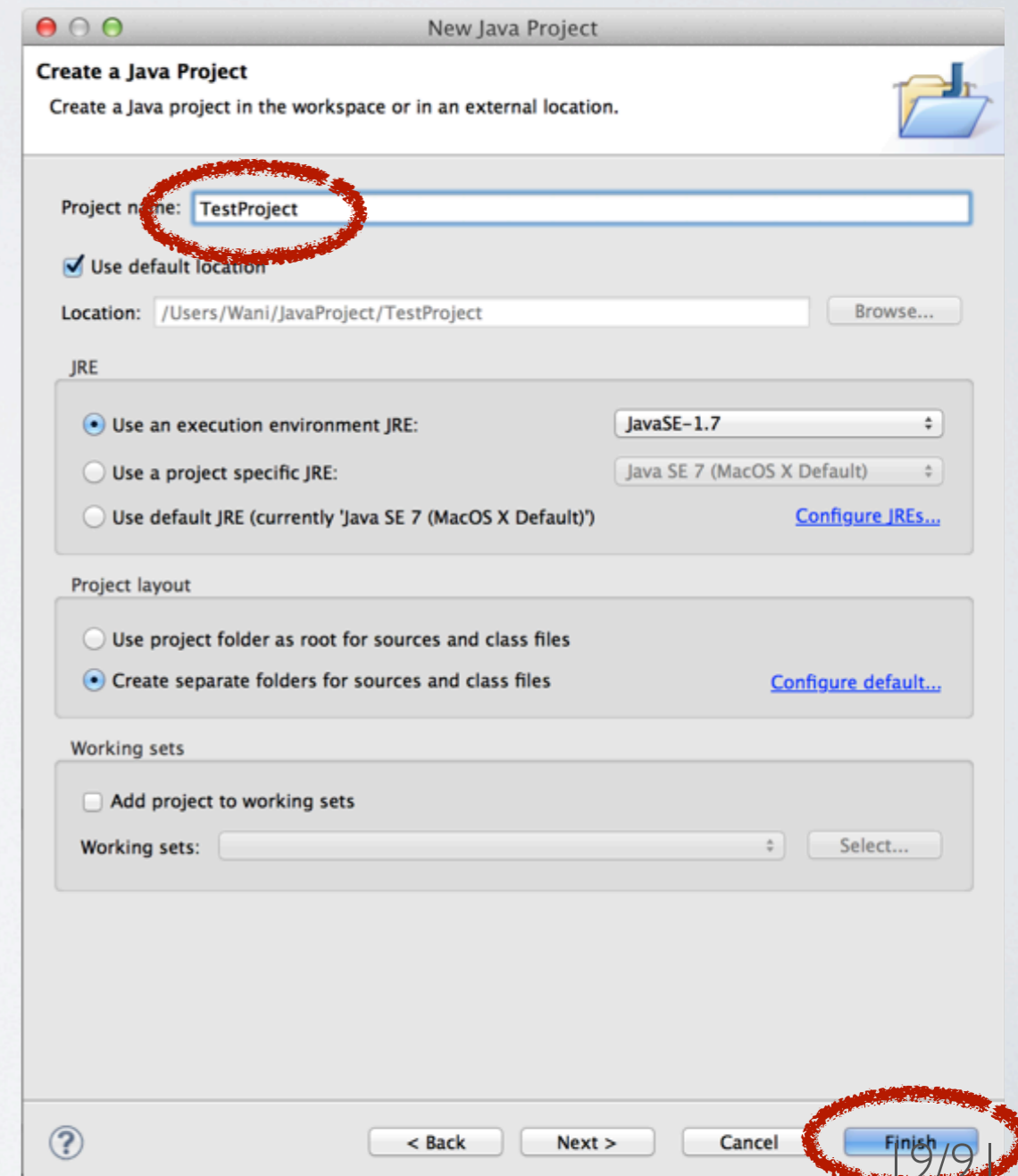
/ Next 클릭



# Eclipse

## Eclipse 사용법

/ Project name에 원하는 제목을 적고 Finish 클릭.

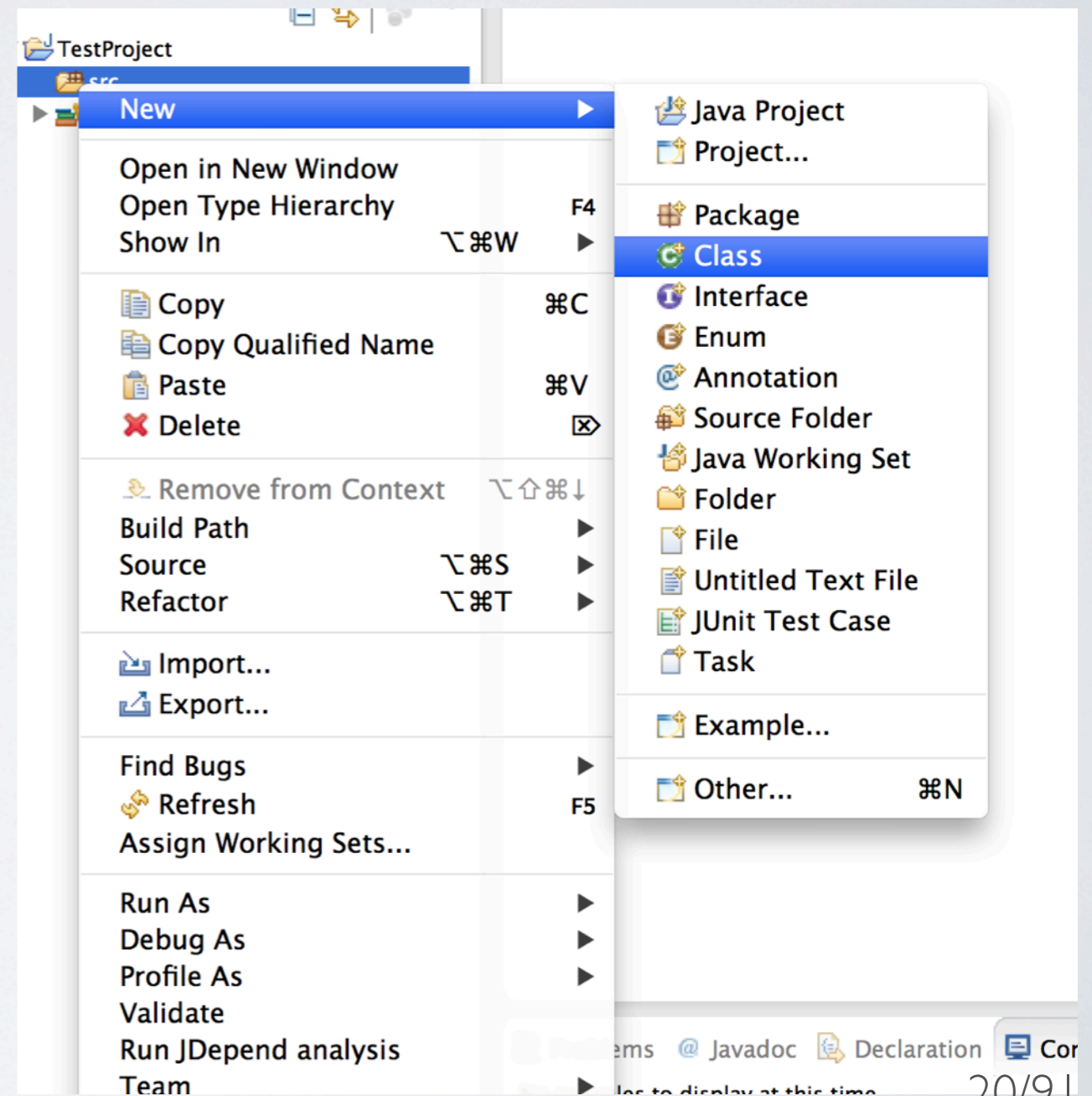


# Eclipse

| Eclipse 사용법

/ src에서 우클릭

/ New -> Class 클릭



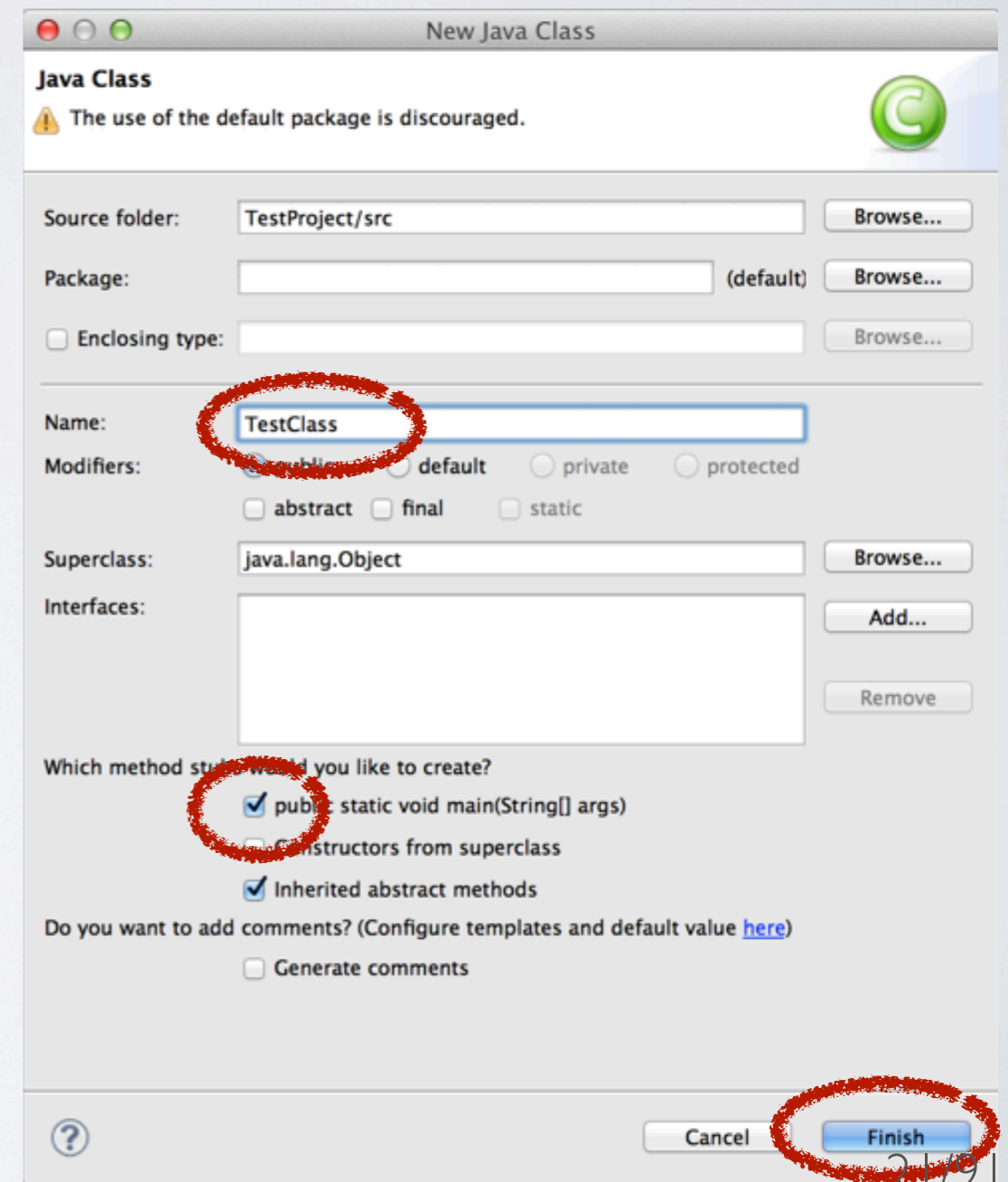
# Eclipse

1 Eclipse 사용법

/ Class Name,

/ public static ... 체크

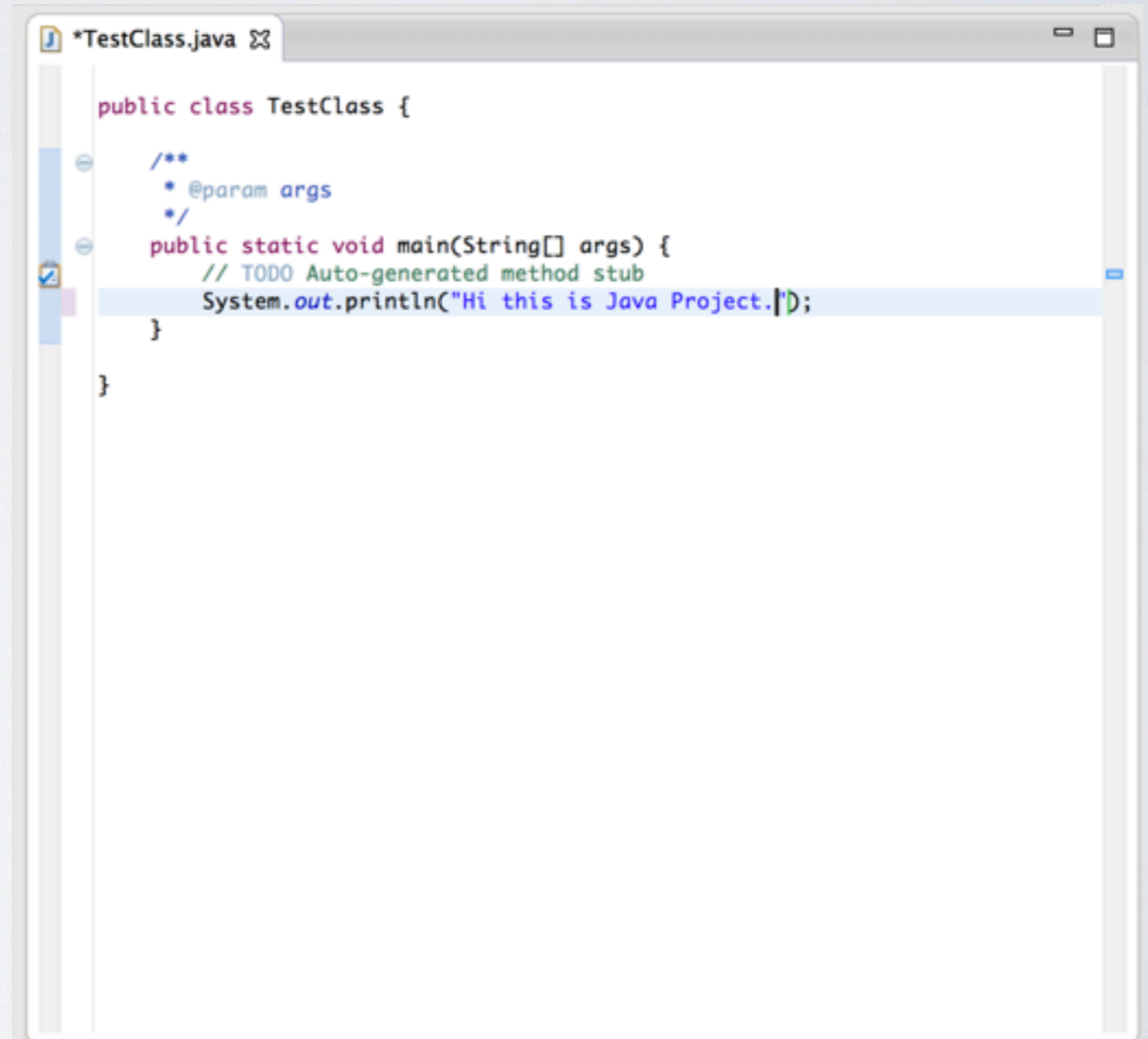
/ Finish 클릭



# Eclipse

| Eclipse 사용법

/ 다음과 같이 작성

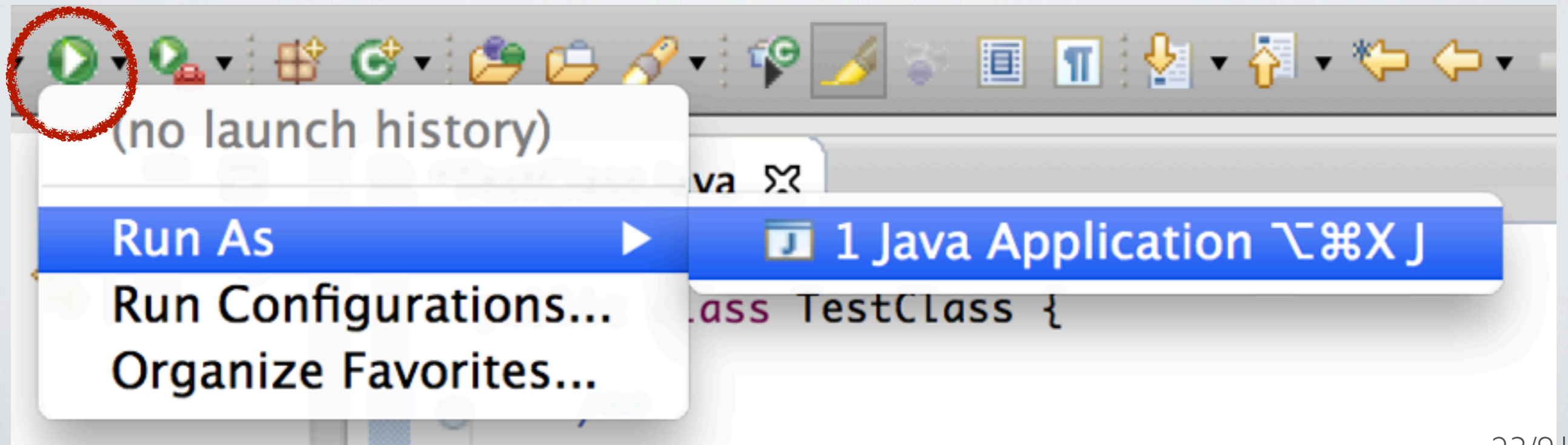


```
*TestClass.java ✕  
  
public class TestClass {  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        System.out.println("Hi this is Java Project.");  
    }  
}
```

# Eclipse

| Eclipse 사용법


/' Run As -> Java Application



# Eclipse

| Eclipse 사용법

/ Console 영역에서 내용 확인



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, and Console. The Console output displays the following text:

```
<terminated> TestClass [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_10.jdk/Contents/Home/bin/  
Hi this is Java Project.
```



# JUnit

# JUnit

## | 개요

- / Java를 위한 오픈 소스 테스트 프레임워크
- / 콘솔 환경에서 명령행으로도 실행 가능
- / Eclipse에는 기본 탑재되어 있음 (설치 필요없음)

# JUnit

## ┆ 단위 및 기능 테스트

### ┆ 단위테스트(Unit Test)란?

┆ 프로그램의 기본 단위가 제대로 동작하는지 테스트 하는 것.

┆ Java 기본단위인 클래스와 메서드를 테스트 함.

### ┆ 기능테스트(Functional Test)란?

┆ 소프트웨어 전체가 제대로 동작하는지 테스트 하는 것.

┆ 전체 소프트웨어를 사용자 입장에서 잘 동작하는지 테스트 하는 것.

┆ 별도 테스트 팀이 수행, 개발과는 다른 도구와 기술 사용.

# JUnit

## | 사용법

/ 간단한 소스를 입력

/ add() a+b를 a=b로 오탁냄

```
package kr.wani.project.junit;

public class BasicOperations {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

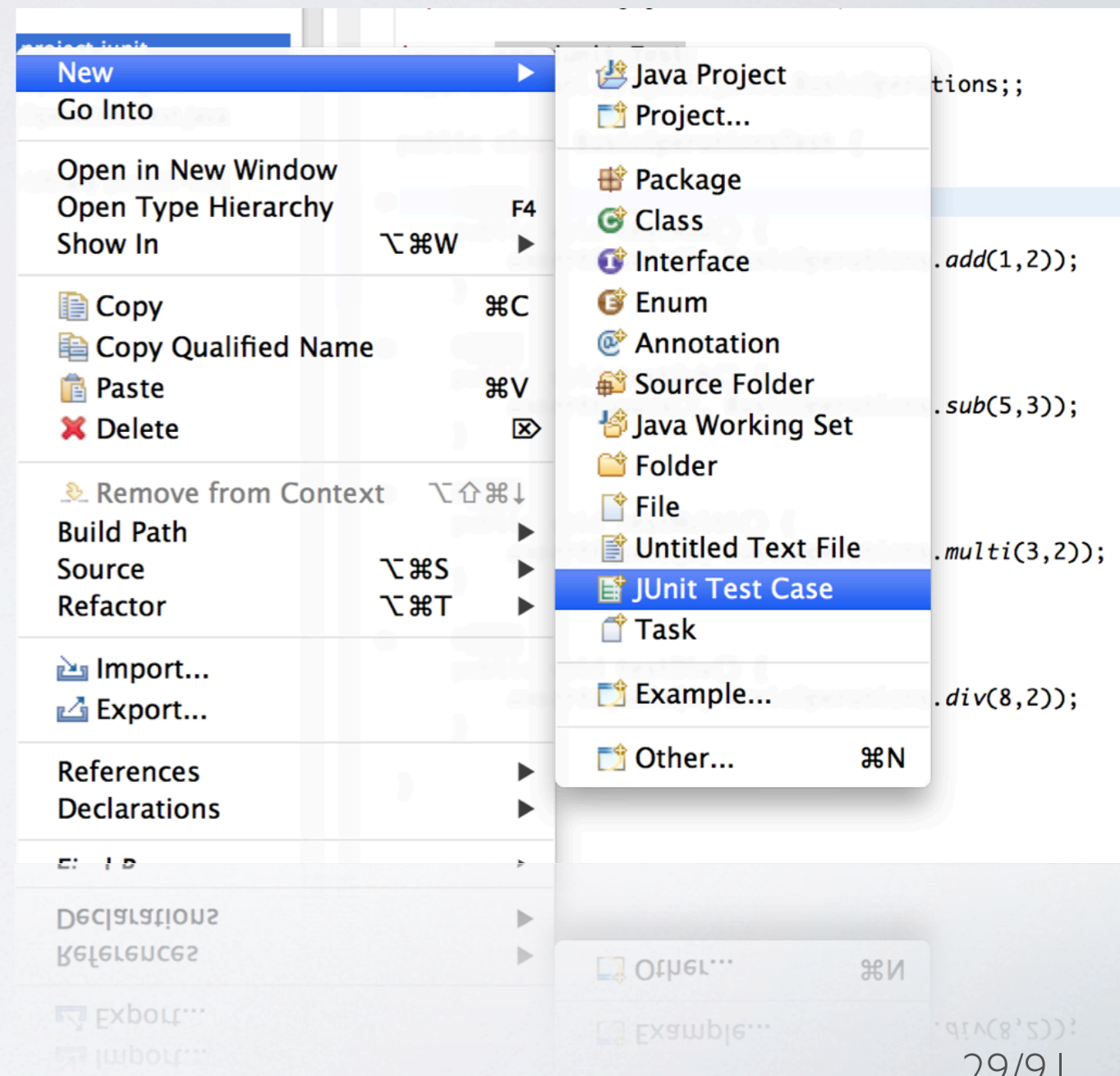
    public static int add(int a, int b) { return a = b; }
    public static int sub(int a, int b) { return a - b; }
    public static int multi(int a, int b) { return a * b; }
    public static int div(int a, int b) { return a / b; }

}
```

# JUnit

## 사용법

New -> JUnit Test Case



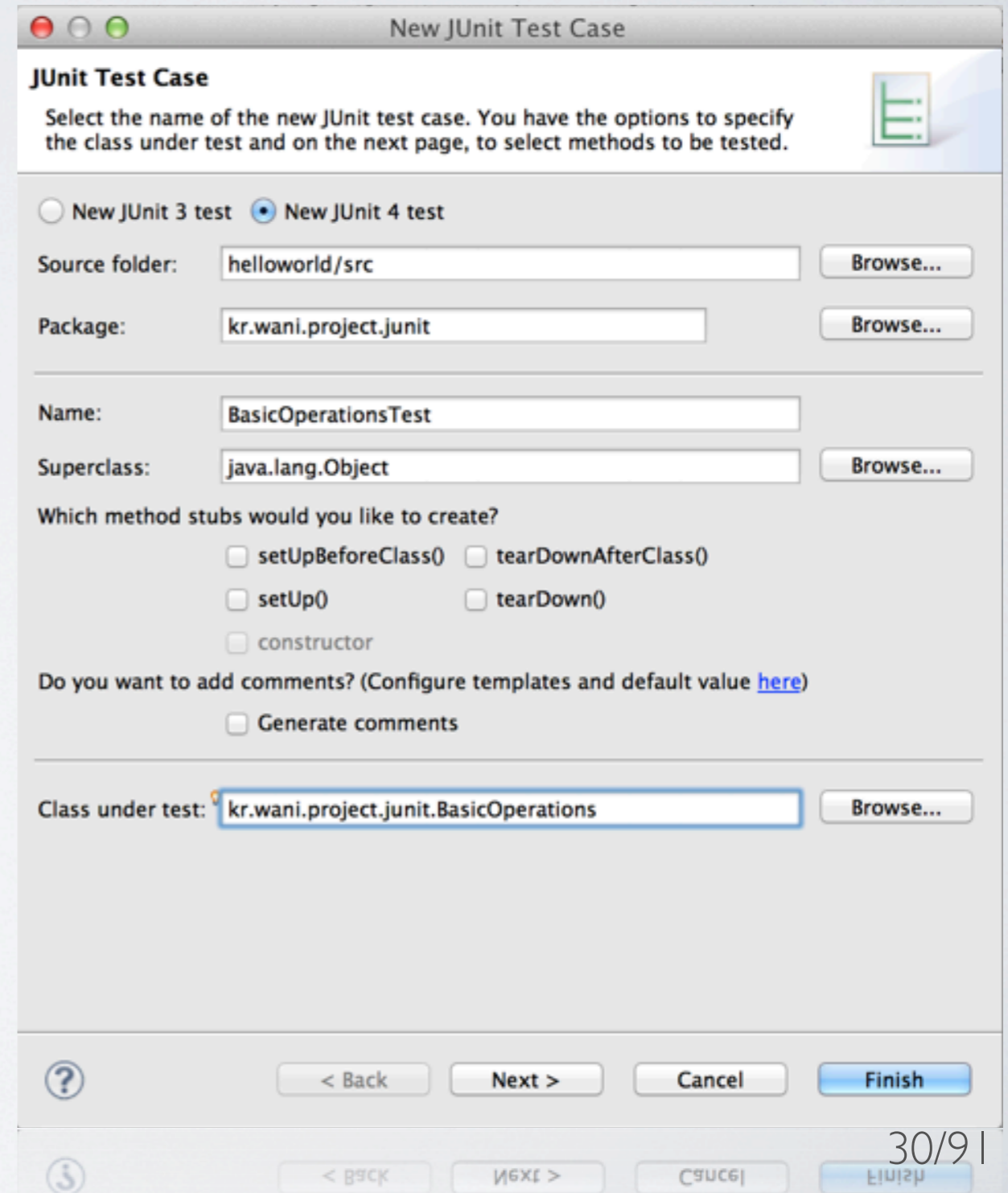
# JUnit

## 사용법

/ 다음과 같이 작성,

/ setUpBeforeClass(),  
tearDownAfterClass()의 경우  
클래스가 실행될때 각각 한번  
씩 실행됨

/ setUp(), tearDown()의 경우 각  
테스트 메서드가 실행되기 전  
후에 한번씩 실행됨



# JUnit

| 사용법

/ 다음과 같이 생성됨

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a project named 'helloworld' with a source folder 'src' containing a package 'kr.wani.project.junit'. Inside this package, there are two files: 'BasicOperations.java' and 'BasicOperationsTest.java'. Below the source folder, there is a 'test' folder containing 'JRE System Library [javaSE-1.7]', 'JUnit 4', 'JDepend', and 'solr-4.2.0'. The main editor window shows the code for 'BasicOperationsTest.java'. The code is as follows:

```
package kr.wani.project.junit;

import static org.junit.Assert.*;

public class BasicOperationsTest {

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

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

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

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

At the bottom of the IDE, there are tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console shows a message: '<terminated> Test [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0\_10'. The page number '31/91' is visible in the bottom right corner of the IDE window.

```

package kr.wani.project.junit;

import static org.junit.Assert.*;

import org.junit.Test;
import kr.wani.project.junit.BasicOperations;;

public class BasicOperationsTest {

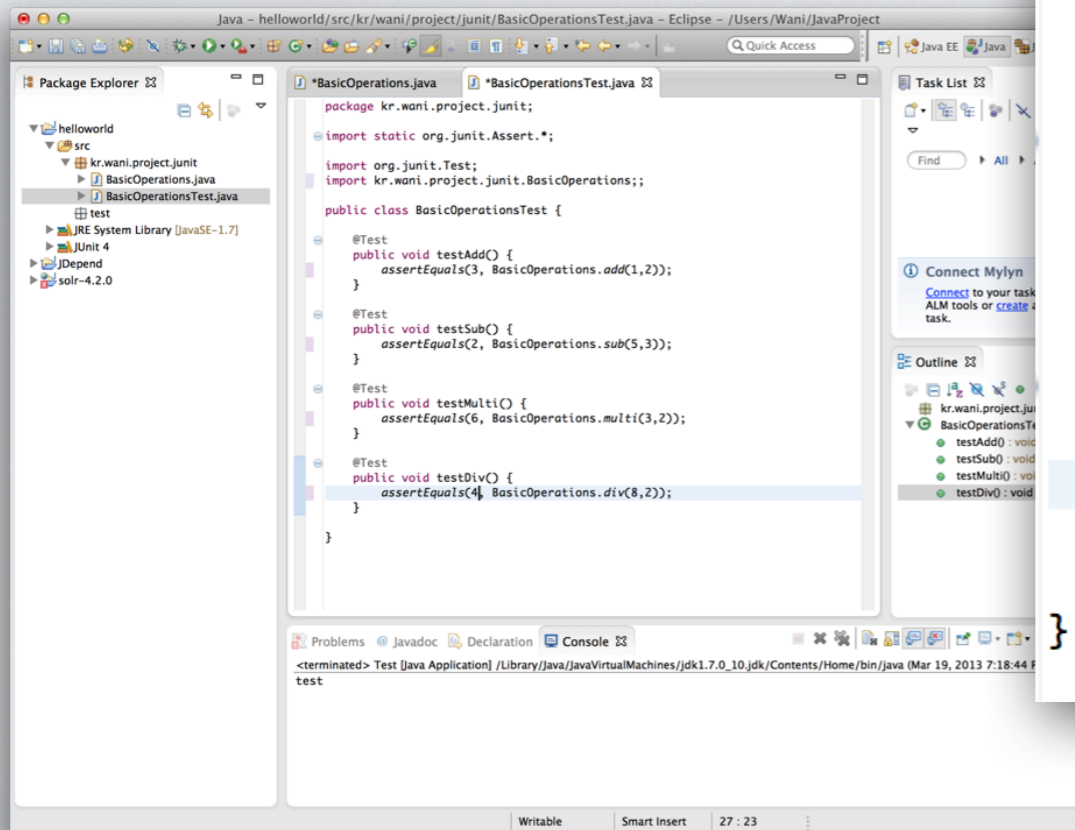
    @Test
    public void testAdd() {
        assertEquals(3, BasicOperations.add(1,2));
    }

    @Test
    public void testSub() {
        assertEquals(2, BasicOperations.sub(5,3));
    }

    @Test
    public void testMulti() {
        assertEquals(6, BasicOperations.multi(3,2));
    }

    @Test
    public void testDiv() {
        assertEquals(4, BasicOperations.div(8,2));
    }
}

```



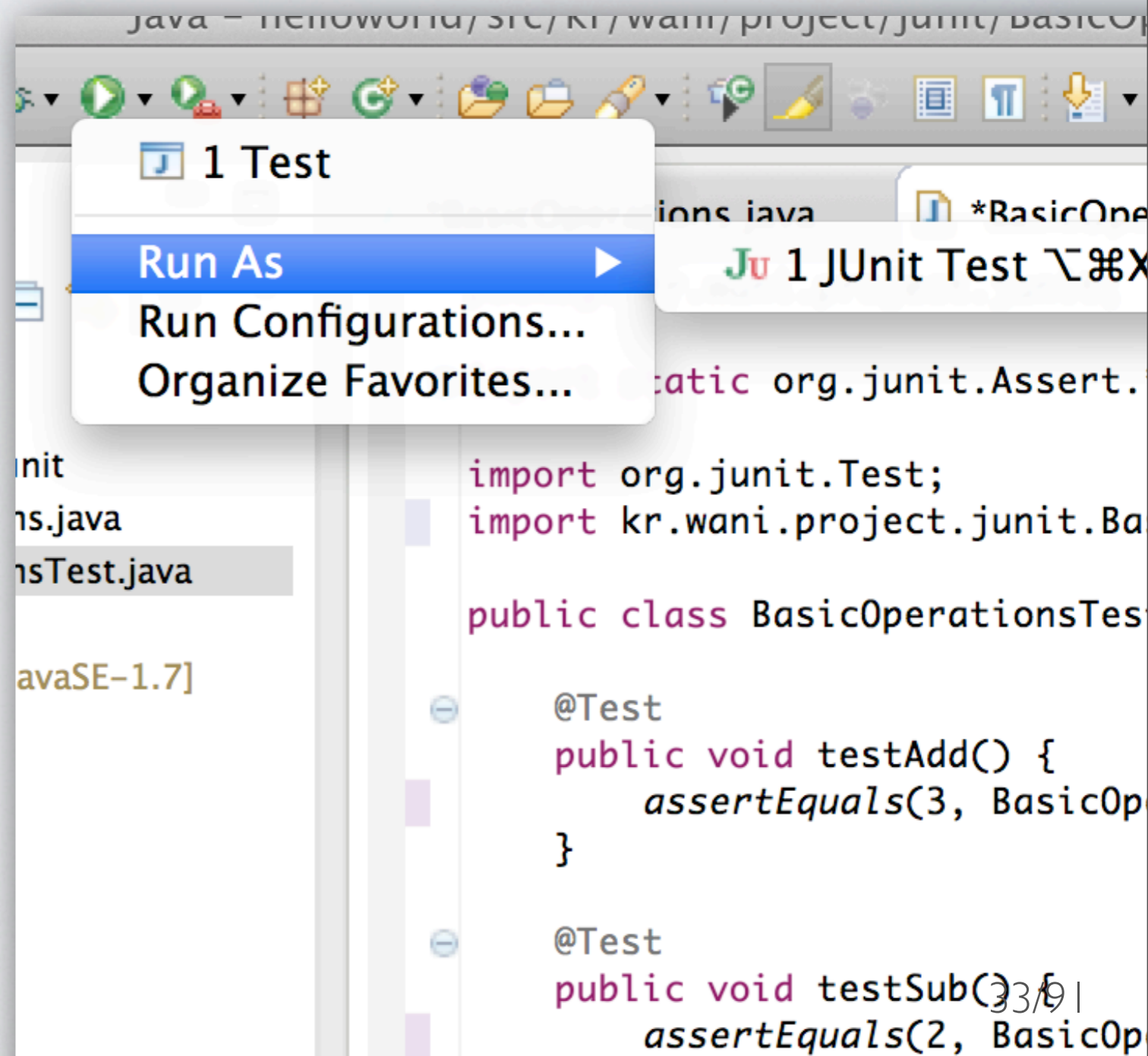
위와 같이 작성



# JUnit

사용법

Run As -> JUnit Test



# JUnit

The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows a test run summary: "Finished after 0.019 seconds", "Runs: 4/4", "Errors: 0", and "Failures: 1". The test results list includes testAdd (0.002 s), testSub (0.001 s), testMulti (0.000 s), and testDiv (0.000 s). The testDiv method is circled in red with a "Click!" label. The main editor shows the source code for BasicOperationsTest.java, with the testDiv method highlighted in blue. The console at the bottom shows a failure message: "java.lang.AssertionError: expected:<3> but was:<4>".

```
package kr.wani.project.junit;  
  
import static org.junit.Assert.*;  
  
import org.junit.Test;  
import kr.wani.project.junit.BasicOperations;  
  
public class BasicOperationsTest {  
  
    @Test  
    public void testAdd() {  
        assertEquals(3, BasicOperations.add(1,2));  
    }  
  
    @Test  
    public void testSub() {  
        assertEquals(2, BasicOperations.sub(5,3));  
    }  
  
    @Test  
    public void testMulti() {  
        assertEquals(6, BasicOperations.multi(3,2));  
    }  
  
    @Test  
    public void testDiv() {  
        assertEquals(4, BasicOperations.div(8,2));  
    }  
}
```

Failure Trace:  
java.lang.AssertionError: expected:<3> but was:<4>  
at kr.wani.project.junit.BasicOperationsTest.testDiv()

# JUnit

The screenshot shows the Eclipse IDE interface. The main editor displays the code for `BasicOperationsTest.java`. The `testDiv` method is highlighted with a red circle and the text "Highlight!!" is written next to it. The console shows a failure trace for `testDiv`.

```
package kr.wani.project.junit;
import static org.junit.Assert.*;
import org.junit.Test;
import kr.wani.project.junit.BasicOperations;

public class BasicOperationsTest {
    @Test
    public void testAdd() {
        assertEquals(3, BasicOperations.add(1,2));
    }

    @Test
    public void testSub() {
        assertEquals(2, BasicOperations.sub(5,3));
    }

    @Test
    public void testMulti() {
        assertEquals(6, BasicOperations.multi(3,2));
    }

    @Test
    public void testDiv() {
        assertEquals(4, BasicOperations.div(8,2));
    }
}
```

JUnit Test Results:

- testAdd (0.002 s)
- testSub (0.001 s)
- testMulti (0.000 s)
- testDiv (0.000 s)

Failure Trace:

```
java.lang.AssertionError: expected:<3> but was:<4>
    at kr.wani.project.junit.BasicOperationsTest.testAdd(BasicOperationsTest.java:12)
```

# JUnit

## | 사용법

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

/ 틀린 부분 수정 후 다시 Run As -> JUnit Test

# JUnit

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

- Package Explorer:** Shows the project structure with 'JUnit' and 'BasicOperationsTest' visible. A green progress bar indicates successful execution.
- Code Editor:** Displays the source code for `BasicOperationsTest.java`. The code includes imports for `org.junit.Assert` and `org.junit.Test`, and defines four test methods: `testAdd()`, `testSub()`, `testMulti()`, and `testDiv()`. Each method uses `assertEquals` to verify the results of corresponding operations in `BasicOperations`.
- Task List:** Contains a 'Connect Mylyn' notification.
- Outline:** Lists the test methods: `testAdd() : void`, `testSub() : void`, `testMulti() : void`, and `testDiv() : void`.
- Console:** Shows the output: `<terminated> BasicOperationsTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.7.0_10.jdk/Contents/Home/bin/java (Mar 19, 2013 7:56:47 PM)`

# JUnit

## | 구성요소

### / Annotation List

@Before	Test method가 실행되기 전에 실행되는 method
@After	Test method가 실행되고 난 후에 실행되는 method
@Test	JUnit의 test class를 생성할 때 'extends TestCase'를 없애준다. method앞의 'test'라고 붙여줘야 하는 불편함을 없애준다. TestCase를 확장하지 않았기 때문에 'assert*' method를 사용하기 위해서는 'import static org.junit.Assert.*;' static imports해준다.
@Test (expected Exception.class)	@Test의 expected에 class method 안의 try, catch에서 발생한 예외를 정의해준다. 정의를 해준 예외는 method안에서 try, catch해 줄 필요가 없다.
@Test (timeout=5000)	@test의 timeout은 method에 시간적인 제한을 줄 수 있다. 성능test시 일정 시간만 실행 후에 종료를 할 수 있다.
@Ignore	@Ignore를 통해서 test method실행시 해당 method를 제외할 수 있다. 제외하는 이유를 적어서 추후에 직관적으로 확인 할 수도 있다.

# JUnit

## | 구성요소

### / Class

Assert	테스트하려는 조건을 명시한다. assert 메서드는 조건이 만족되면 아무일도 없었다는 듯이 조용히 지나가며, 만족되지 못하면 예외를 던진다.
Test	@Test 애노테이션이 부여된 메서드로, 하나의 테스트를 뜻한다. JUnit은 먼저 메서드를 포함하는 클래스의 인스턴스를 만들고, 애노테이션된 메서드를 찾아 호출한다.
Test Class	@Test 메서드를 포함한 클래스이다.
Suite	Suite는 여러 테스트 클래스를 하나로 묶는 수단을 제공한다.
Runner	러너는 테스트를 실행시킨다.

# JUnit

## | 구성요소

### / Method

<code>assertEquals(primitive expected, primitive actual)</code>	두 개의 기본형 변수의 값이 같은지 검사
<code>assertEquals(Object expected, Object actual)</code>	두 개의 객체 값이 같은지 검사
<code>assertSame(Object expected, Object actual)</code>	두 개의 객체가 같은지 검사
<code>assertNotSame(Object expected, Object actual)</code>	두 개의 객체가 다른지 검사
<code>assertNull(Object object)</code>	객체가 NULL인지 검사
<code>assertNotNull(Object object)</code>	객체가 NULL이 아닌지 검사
<code>assertTrue(Boolean condition)</code>	조건문이 true인지 검사
<code>assertFalse(Boolean condition)</code>	조건문이 false인지 검사



# JDepend

# JDepend

- | 개요
  - ' 패키지 의존성과 설계 품질의 객체 지향식 측정을 통해, 패키지를 분석하고 관리할 수 있도록 지원하는 도구.
- | 주요기능
  - ' 패키지 의존성 측정
- | 관련도구
  - ' Eclipse Metrics
- | 제작사
  - ' Clarkware Consulting, Inc (<http://andrei.gmxhome.de/jdepend4eclipse/>)

# JDepend

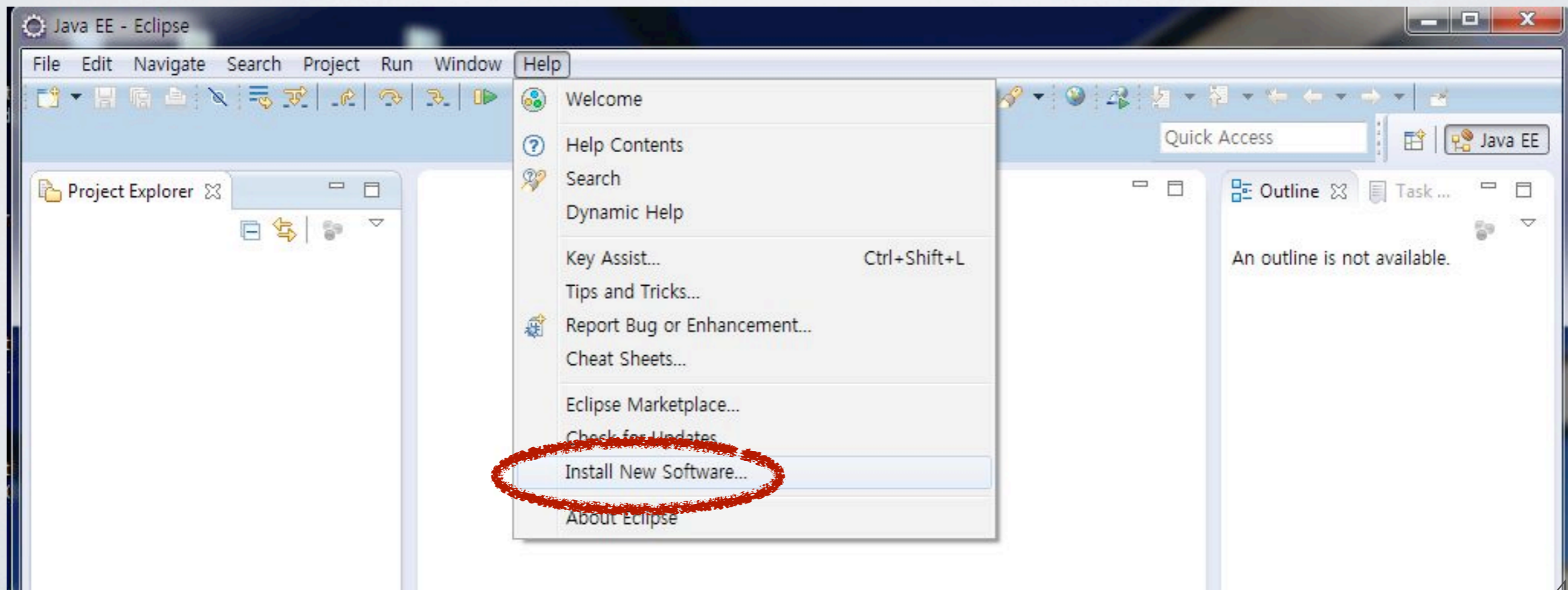
## | 특징

- / 패키지 별로 의존성 측정가능
- / 패키지 의존성과 관련된 데이터 품질을 수치화하여 표현
- / 수치화된 의존성 정보를 텍스트 형태로 제공
- / 수치화된 데이터 품질을 그래프로 표현
- / 의존성 정보를 XML 형식 파일로 저장가능

# JDepend

## | 설치방법

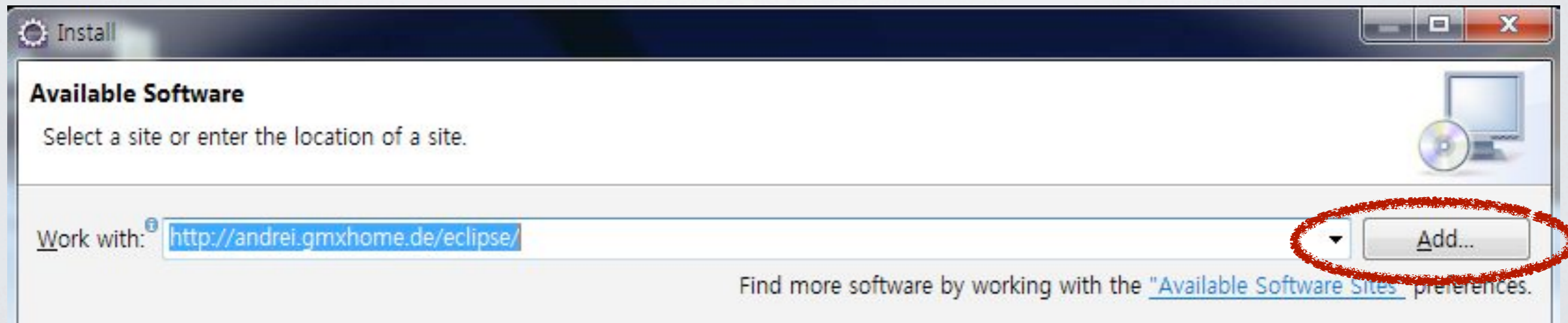
/ Help -> Install New Software



# JDepend

| 설치방법

/' Add 클릭

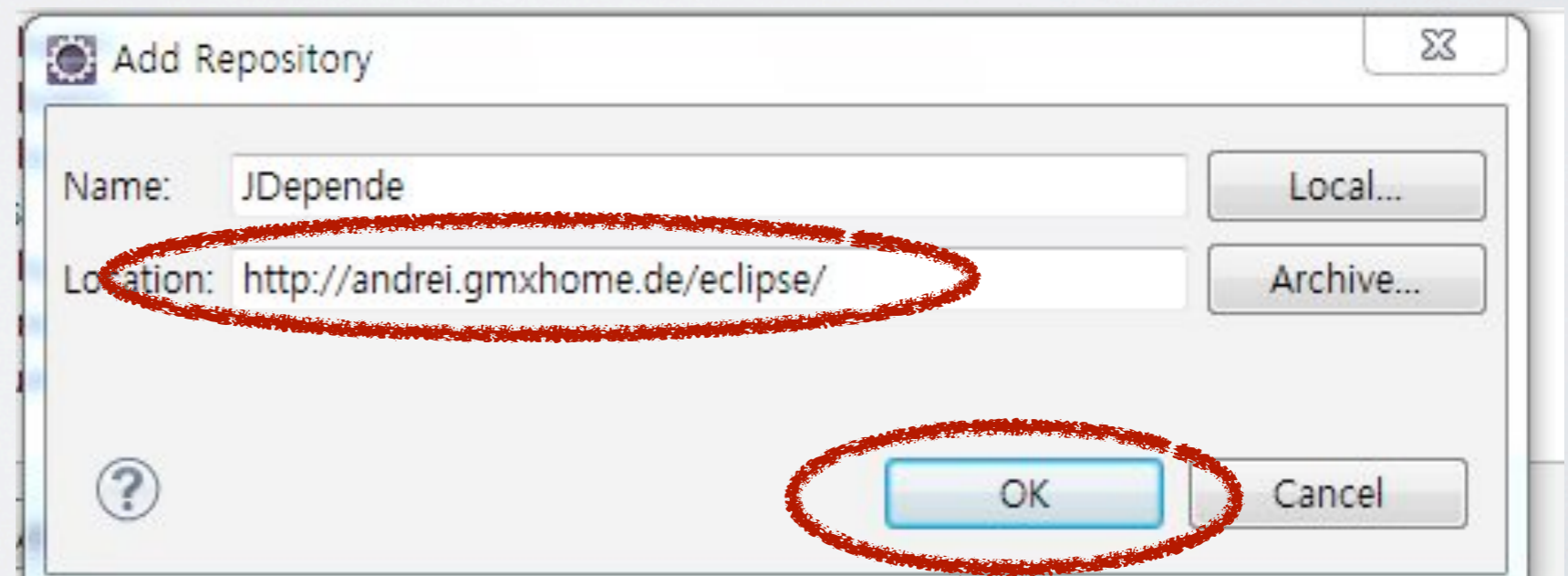


# JDepend

## | 설치방법

/ Location : <http://andrei.gmxhome.de/eclipse/>

/ Name은 아무거나 상관없음, OK 클릭.



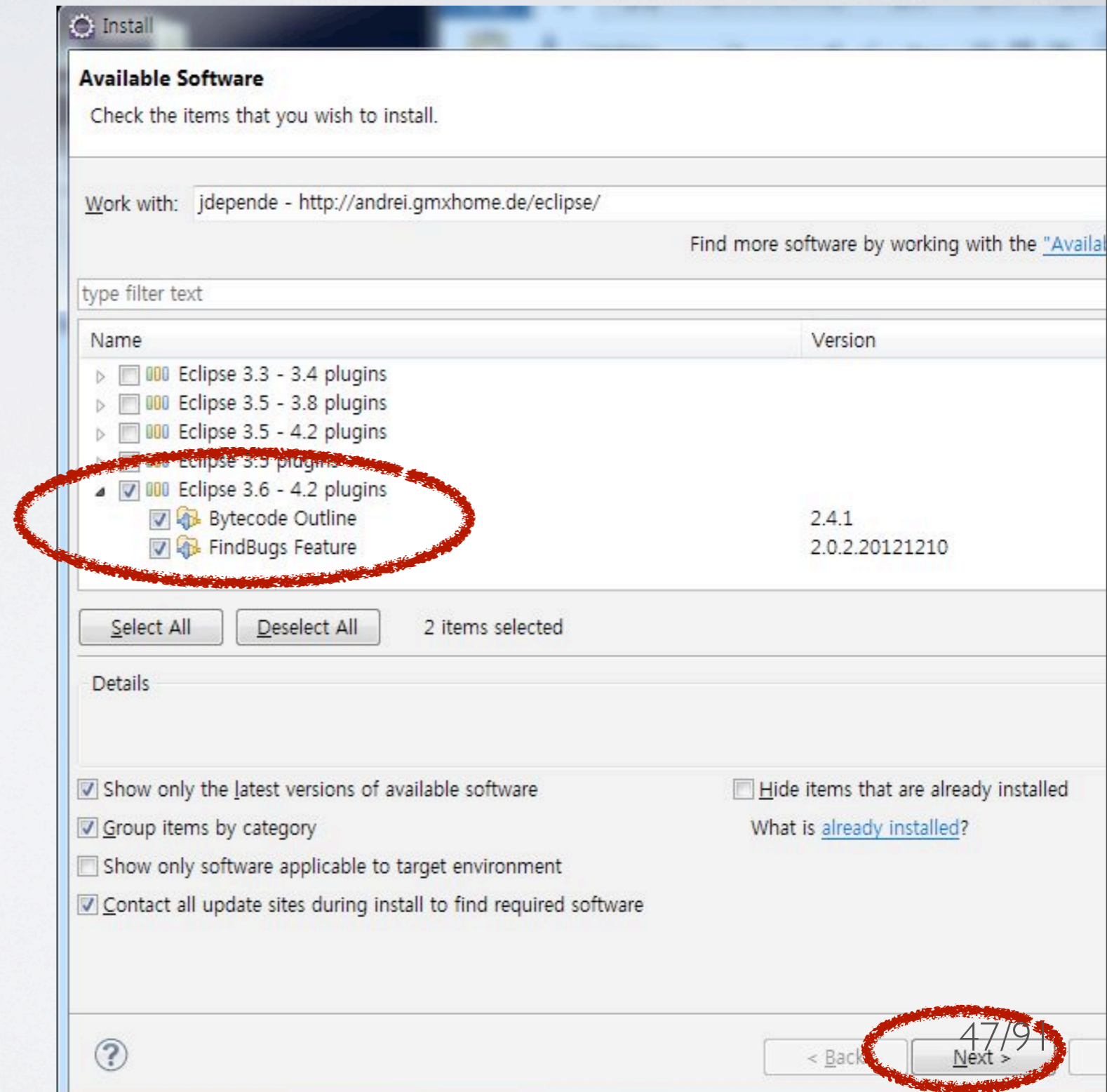
# JDepend

| 설치방법

/ Eclipse Version Select

/ Checkbox Check

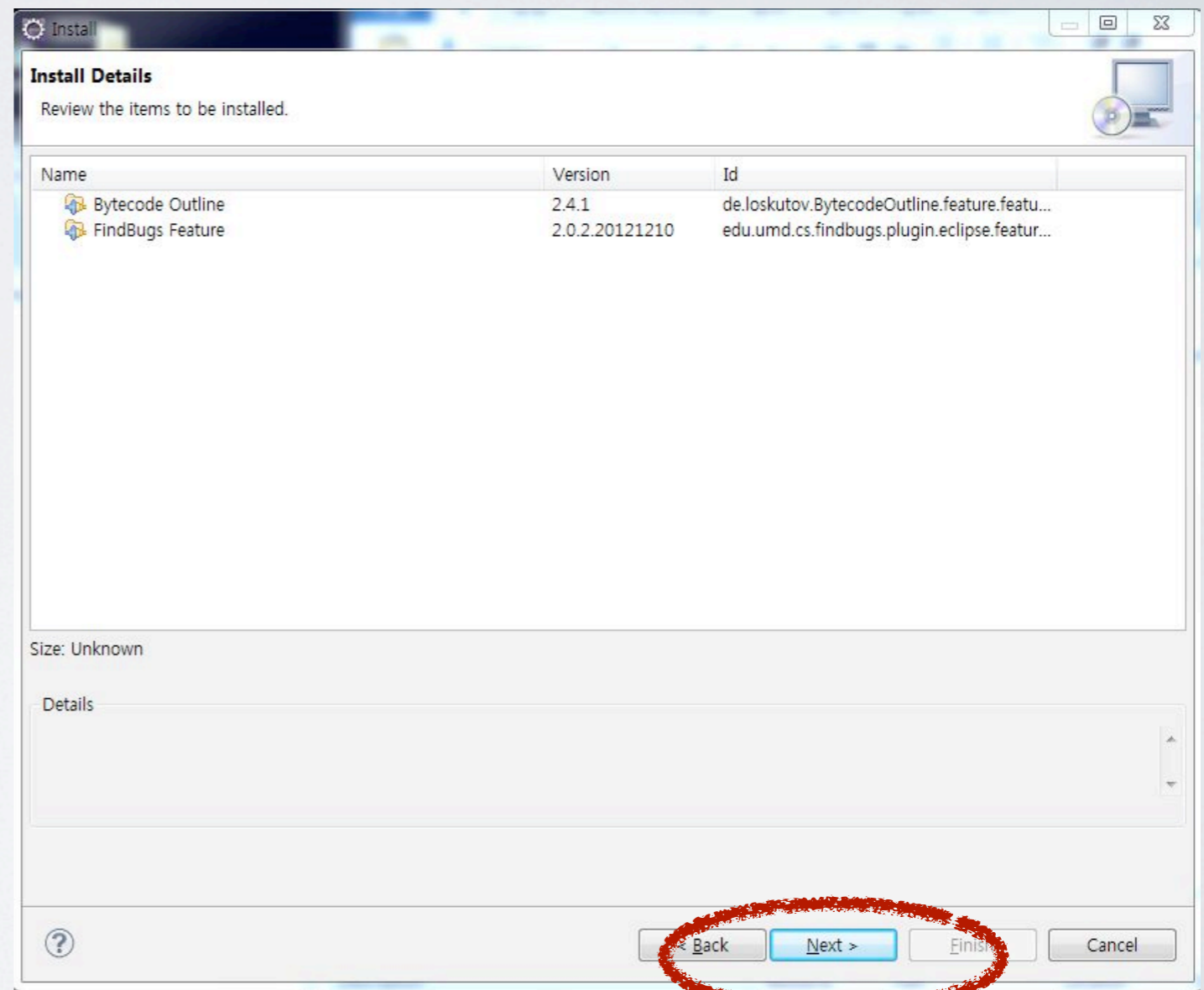
/ Next



# JDepend

| 설치방법

/ Next



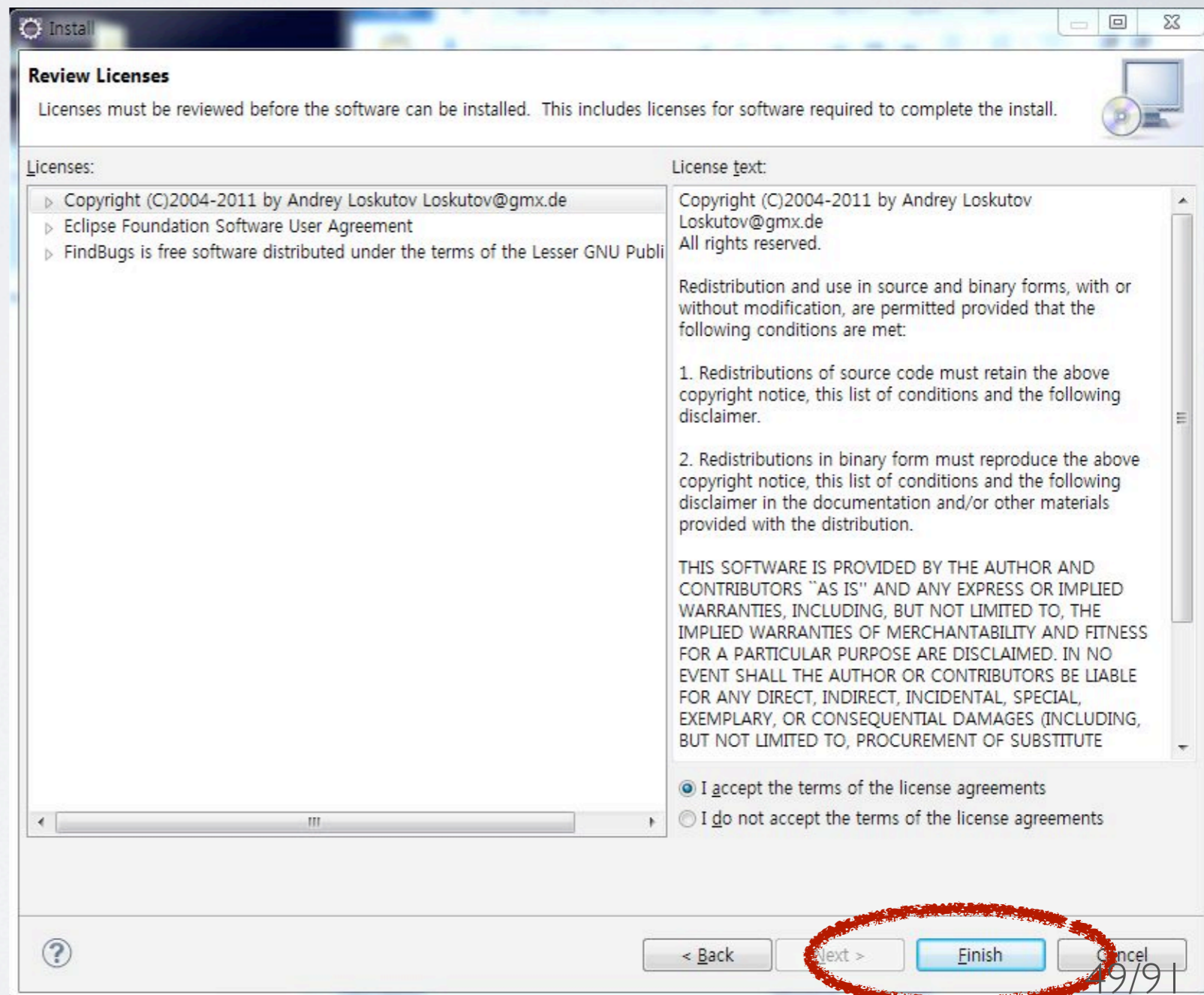


# JDepend

## 설치방법

/ Check, I accept ..

/ Finish

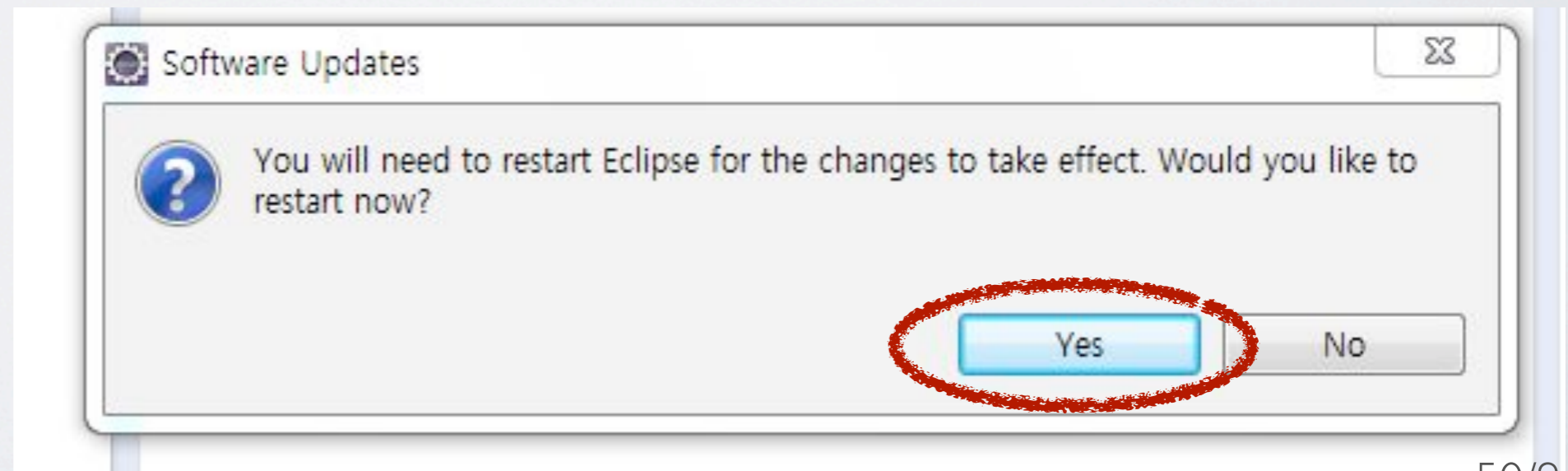
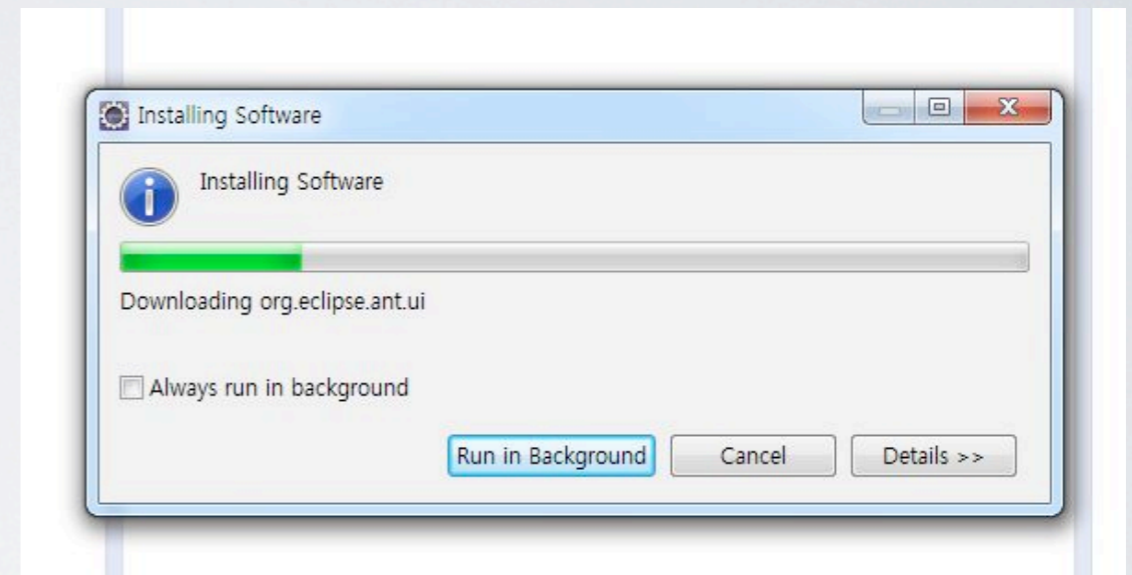


# JDepend

## | 설치방법

/ Waiting ...

/ Finish, then Click Yes.

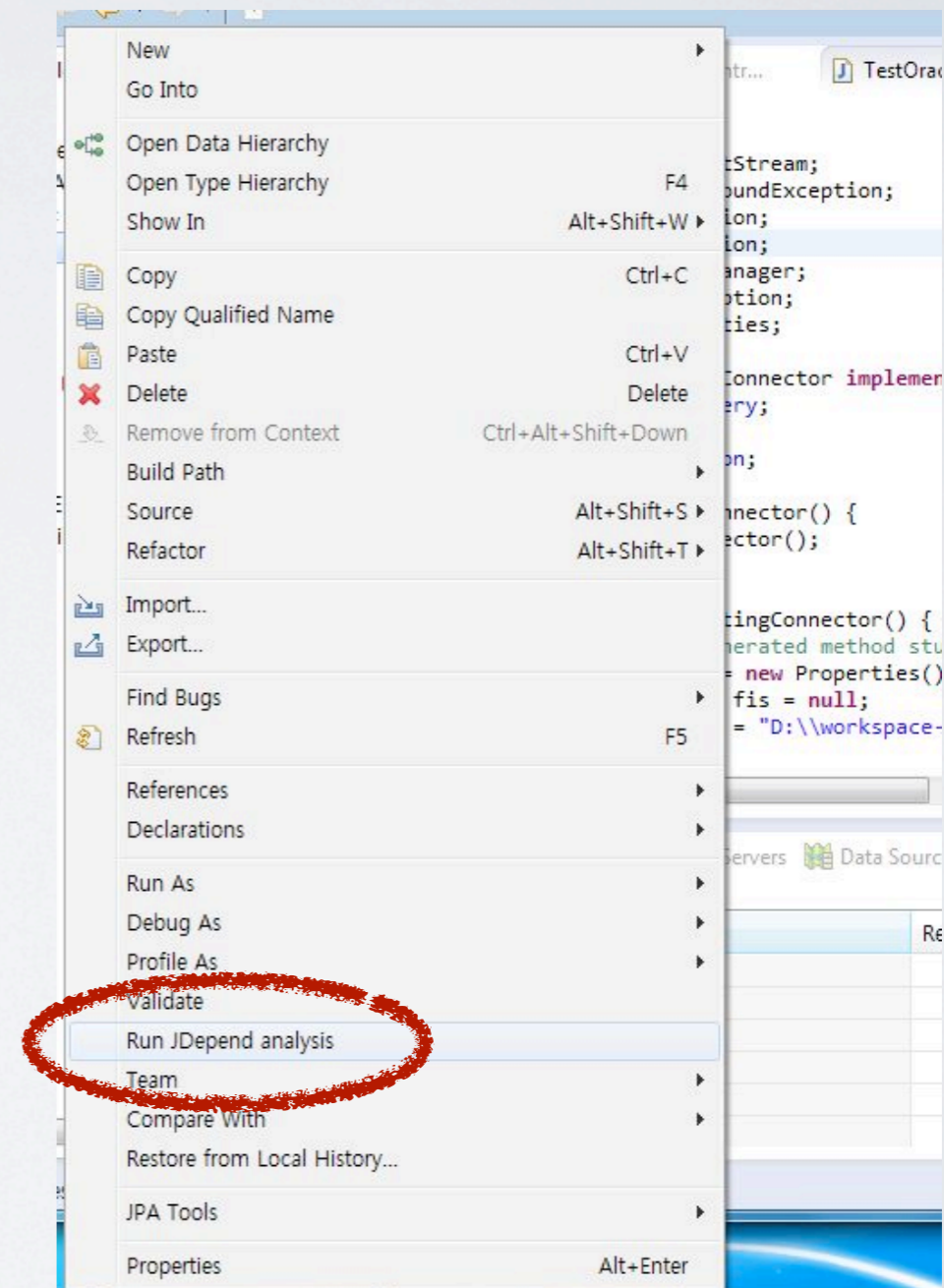


# JDepend

## 사용방법

/ 원하는 패키지 선택

/ Run JDepend analysis 클릭



# JDepend

## 사용방법

/ Packages

/ Dependencies

/ Metrics

/ 3개의 창 확인가능.

The screenshot displays the Eclipse IDE with the JDepend plugin. The 'Packages' view on the left shows a tree structure for the 'jdepend' package containing several Java files. The 'Dependencies' view on the right shows a table of dependencies for the selected object(s). The 'Metrics' view at the bottom left shows a graph for 'Instability ->'. The 'Dependencies' view includes the following tables:

Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
jdepend	4	2	0	0	0.33	0.00	0.66	

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
---------	---------	---------	----------	----------	---	---	---	--------

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
---------	---------	---------	----------	----------	---	---	---	--------

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
---------	---------	---------	----------	----------	---	---	---	--------

# JDepend

## 사용방법

### Packages View

여러개의 패키지를 포함하고 있는 경우 이곳에 모든 패키지 정보가 표시 됨

하나의 패키지 또는 여러개의 의존성 정보를 확인할 수 있음

The screenshot displays the Eclipse IDE with the JDepend plugin. The Packages view is highlighted with a red border, showing a tree structure of packages under 'jdepend'. The Dependencies view shows a table with columns for Package, CC(concr.cl.), AC(abstr.cl.), Ca(aff.), Ce(eff.), A, I, D, and Cycle!. The Metrics view shows a line graph with a red line labeled 'Instability ->'.

Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
jdepend	4	2	0	0	0.33	0.00	0.66	

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!

# JDepend

사용방법

Dependencies View

선택한 패키지의 의존성 정보를 수치화하여 표현

The screenshot shows the Eclipse IDE with the JDepend plugin. The 'Dependencies' view is active, displaying a table of metrics for the selected package 'jdepend'. The table includes columns for various dependency types and their counts, as well as calculated stability and cycle metrics.

Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
jdepend	4	2	0	0	0.33	0.00	0.66	

Below the main table, there are three empty tables for 'Packages with cycle', 'Depends upon - efferent dependencies', and 'Used by - afferent dependencies'. The 'Metrics' view at the bottom left shows a graph for 'Instability ->'.

# JDepend

사용방법

Dependencies View

수치에 대한 정의

CC	Concrete Class Interface, Abstract Class를 제외한 Concrete class의 갯수.
AC	Abstract Class 추상클래스나 Interface의 갯수. 확장성의 척도.
Ca	Afferent Couplings 현재 패키지의 클래스에 의존하고 있는 패키지의 수. 책임의 척도가 됨.
Ce	Efferent Couplings 현재 패키지의 클래스들이 의존하고 있는 패키지의 수. 독립성의 척도가 됨.
A	Abstractness ( $A = AC/CC+AC$ ) 추상화 정도를 나타내며, 0~1 사이의 값을 가짐. 0은 완전 구체적인 패키지, 1은 추상적인 패키지를 나타낸다.
I	Instability ( $I = Ce/(Ce + Ca)$ ) 변화에 대한 안정성을 나타내며, 0~1 사이의 값을 가짐. 0은 외부 변화에도 끄떡없는 패키지이며, 1은 작은 변화에도 쉽게 흔들릴 수 있는 패키지를 나타낸다.
D	Distance To Main Sequence Main Sequence로부터의 거리를 나타내며, 0은 Main Sequence와 가깝고, 1은 먼 상태. Main Sequence란 이상적인 패키지로 완전 추상적이면서 안정적이거나 완전 구체적이면서 불안정한 패키지를 나타낸다.
Cycle	Package Dependency Cycle 패키지들 상호 간에 의존성을 가지고 있을 때 발생 안 좋은 상황이기 때문에 경고 아이콘으로 보여짐

# JDepend

사용방법

Metrics View

Dependencies 에서  
수치화된 정보를 그  
래프로 보여줌

The screenshot shows the Eclipse IDE with the JDepend plugin. The 'Dependencies' view is active, displaying a table of metrics for the 'jdepend' package. The 'Metrics' view is also visible, showing a graph of 'Instability'.

Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
jdepend	4	2	0	0	0.33	0.00	0.66	

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!



# JDepend

## 사용방법

### Metrics View

#### 추상화되어 있는 패키지

Ce는 없고 Ca가 발생하는 것이 좋습니다. 즉 안정성이 높은 상태로 유지되는 것이 바람직합니다.

#### 구체화되어 있는 패키지

Ca는 없고 Ce가 발생하는 것이 좋습니다. 외부 패키지와의 의존관계를 가질 경우 Concrete Class 보다는 Interface를 통하여 의존관계를 가지는 것이 좋은 설계입니다. 즉, 구체화되어 있는 패키지는 안정성이 낮은 상태로 유지되는 것이 좋습니다.

### Main Sequence

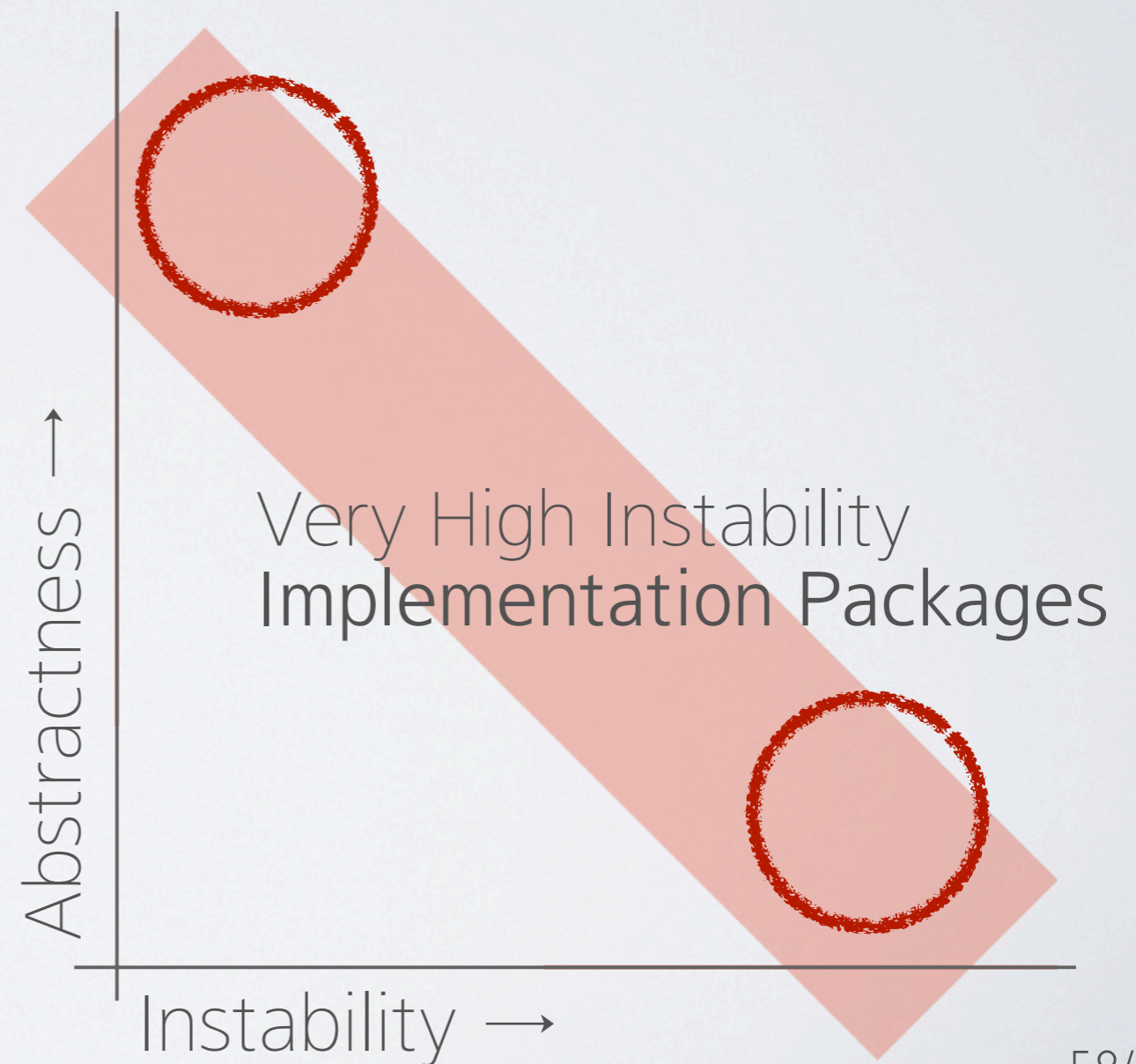
이상적인 패키지로 완전 추상적이면서 안정적이거나, 완전 구체적이면서 불안정한 패키지를 나타냅니다. D값은 A와 I의 종합이 적절하게 잘 유지되도록 고려할 때 판단할 수 있는 좋은 근거가 될 수 있습니다.

# JDepend

| 사용방법

/ Metrics View

Very High Stability  
Interface Packages



# JDepend

## 사용방법

Dependencies view table:

Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
jdepend	4	2	0	0	0.33	0.00	0.66	

Metrics view:

Instability ->

/ Show JDepend output, 클릭

# JDepend

## | 사용방법

/ Dependencies에서  
수치화된 정보를  
Console에서 텍스트로  
확인 가능



```
Dependencies Console Package Explorer
JDepend
-----
- Package: jdepend
-----

Stats:
  Total Classes: 6
  Concrete Classes: 4
  Abstract Classes: 2

  Ca: 0
  Ce: 0

  A: 0.33
  I: 0
  D: 0.67

Abstract Classes:
  jdepend.iTestDBConnector
  jdepend.iTestDBQuery

Concrete Classes:
  jdepend.TestDBControl
  jdepend.TestDBKindMap
  jdepend.TestMain
  jdepend.TestOracleConnector

Depends Upon:
  Not dependent on any packages.

Used By:
  Not used by any packages.

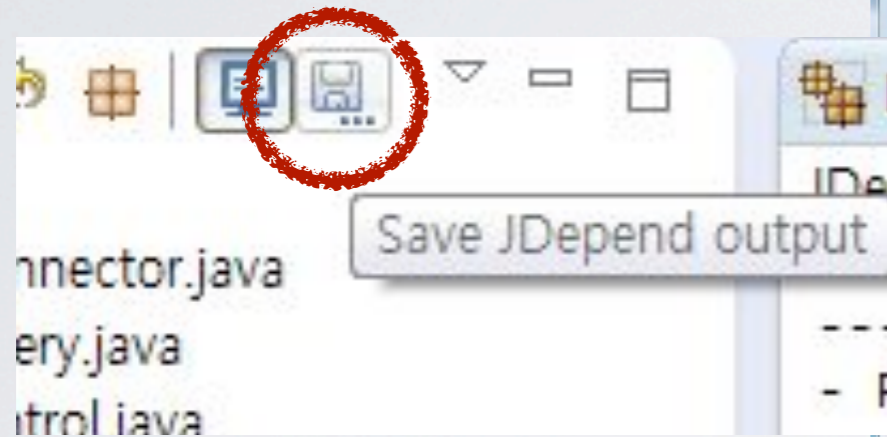
-----
- Package Dependency Cycles:
-----

-----
- Summary:
-----

Name, Class Count, Abstract Class Count, Ca, Ce, A, I, D, V:
jdepend 6 2 0 0 0 33 0 67 1
```

# JDepend

## 사용방법



Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
jdepend	4	2	0	0	0.33	0.00	0.66	

Selected object(s)

Packages with cycle

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
---------	---------	---------	----------	----------	---	---	---	--------

Depends upon - efferent dependencies

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
---------	---------	---------	----------	----------	---	---	---	--------

Used by - afferent dependencies

Package	CC(c...	AC(a...	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
---------	---------	---------	----------	----------	---	---	---	--------

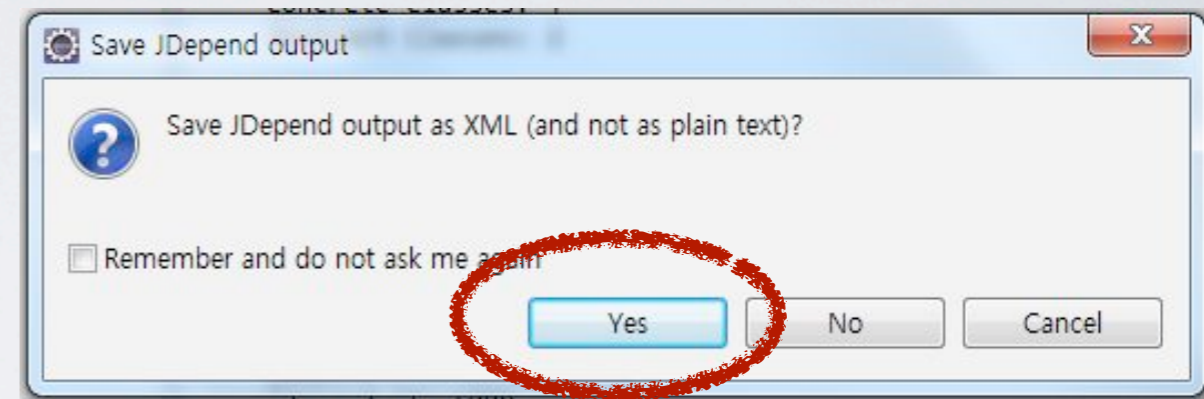
/ Dependencies 정보를 XML로 보여줌

/ Save JDepend output 클릭

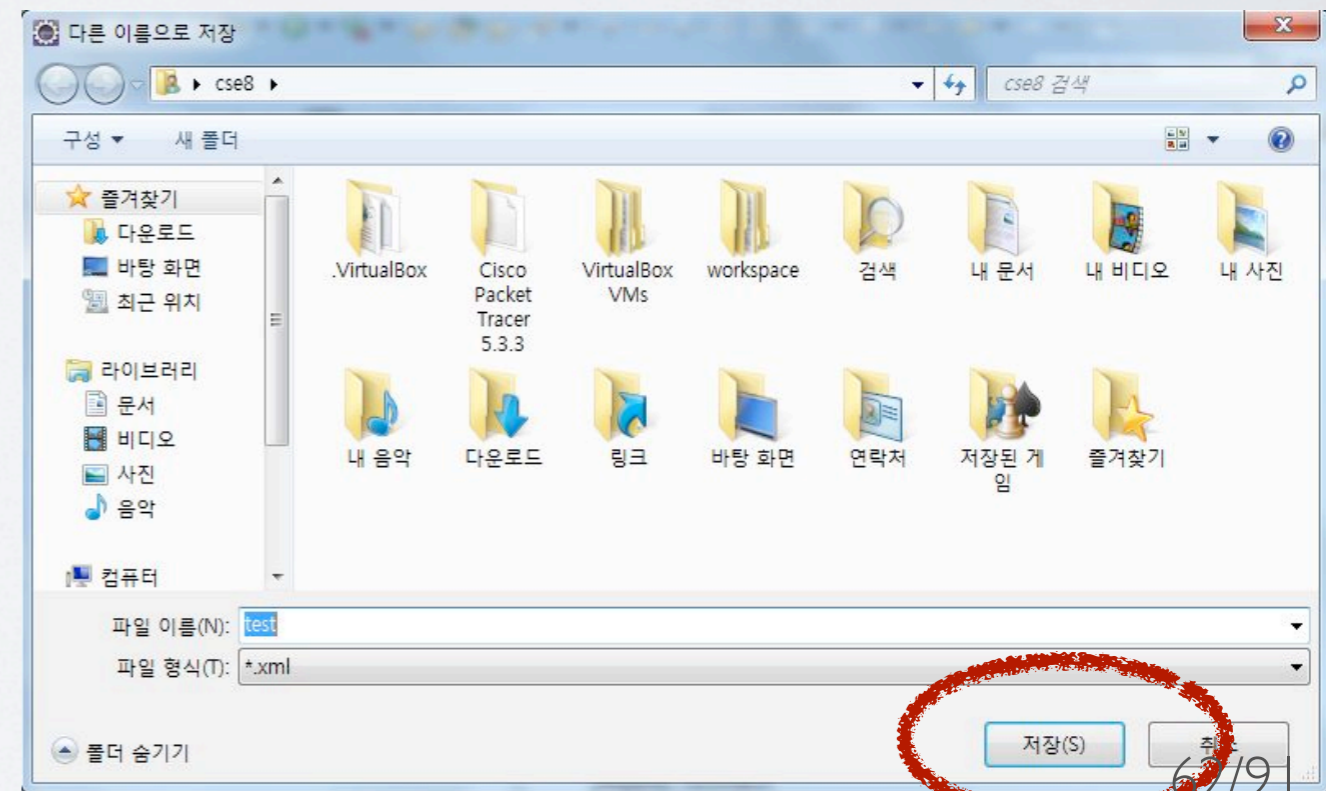
# JDepend

| 사용방법

/ Yes 클릭



/ 저장경로 선택 후 저장



# JDepend

## 사용방법

XML로 저장된  
Dependencies 정보 확인

```
<?xml version="1.0"?>
- <JDepend>
  - <Packages>
    - <Package name="jdepend">
      - <Stats>
        <TotalClasses>6</TotalClasses>
        <ConcreteClasses>4</ConcreteClasses>
        <AbstractClasses>2</AbstractClasses>
        <Ca>0</Ca>
        <Ce>0</Ce>
        <A>0.33</A>
        <I>0</I>
        <D>0.67</D>
        <V>1</V>
      </Stats>
      - <AbstractClasses>
        <Class sourceFile="iTestDBConnector.java"> jdepend.iTestDBConne
        <Class sourceFile="iTestDBQuery.java"> jdepend.iTestDBQuery </C
      </AbstractClasses>
      - <ConcreteClasses>
        <Class sourceFile="TestDBControl.java"> jdepend.TestDBControl </
        <Class sourceFile="TestDBKindMap.java"> jdepend.TestDBKindMap
        <Class sourceFile="TestMain.java"> jdepend.TestMain </Class>
        <Class sourceFile="TestOracleConnector.java"> jdepend.TestOracle
      </ConcreteClasses>
      <DependsUpon> </DependsUpon>
      <UsedBy> </UsedBy>
    </Package>
  </Packages>
  <Cycles> </Cycles>
</JDepend>
```

# JDepend

## | 결론

- / 현재패키지의 의존성 및 Cycle Dependency의 문제점을 찾기 쉽다.
- / 개발완료가 아닌 개발 중에도 문제점을 체크할 수 있어 사용이 용이하다.
- / 자바의 경우 외부 라이브러리 의존관계가 복잡, 그에 대한 해결책은 내가 만들고 있는 시스템 내부 의존관계에 대한 정립, 아키텍처를 가진 설계와 접근, 검증과정의 도입이라고 볼 수 있다. 그 때 패키지 레벨의 의존관계에 대한 분석을 통해 시스템 아키텍처를 검토할 때 많은 도움을 주는 툴이다.



# JDepend

## | 결론

- / 의존관계 분석에서 중요한 것은 **순환의존관계**이다. 이는 매우 안좋은 설계이며 **단방향 의존성**이 중요한 이유는 다음과 같다.
  - / 순환참조는 코드를 악화시킨다.
  - / 패키지 재사용에 제한이 생긴다.
  - / B를 사용하려면 A를 컴파일해야하는데, A는 다시 B를 컴파일해야 하는 상황이 발생한다.
- / 이러한 문제를 해결하기 위해선 패키지들을 개념적인 모듈로 묶어야 한다. 또한 모듈을 만들어 낼 땐 개념적인 경계와 더불어 배포도 고려해야한다.

# Clover

# Clover

## | Test Coverage

/ 소프트웨어의 각 부분을 얼마나 테스트 실행했는지 분석한 값

/ 신규/개편 개발이나 서스테이닝 개발을 할 때 새로 추가하거나 수정한 소스 코드는 코드 완성도를 높이기 위해 테스트 코드를 작성하여 개발자 테스트를 수행해야 한다. 개발자가 자신이 작성한 코드를 충분히 테스트하지 않아서 QA단계에서 문제가 발견되면 그 때 다시 개발자가 일일이 코드를 수정해야 하고 전체적인 일정에 차질이 올 수 있다. 이러한 문제점을 해결하기 위해 테스트를 충분히 했는지 확인할 수 있는 방법이 **Test Coverage**이다.

# Clover

- | Test Coverage

- / 명세기반 Test Coverage

- / 가장 간단한 측정법, 시스템 명세중 테스트한 명세의 비율을 계산

- / 테스트한 명세 개수 / 전체 명세 개수 \* 100

- / 코드기반 Test Coverage(Code Coverage)

- / 명세 내용과 실제 구현내용이 달라 명세기반 Test Coverage가 큰 의미를 갖지 못하는 경우 사용되며, 동적 테스트 방법의 하나로서 실제로 동작하는 소스 코드가 기본이며 언제나 명확한 값을 얻을 수 있음

- / 테스트된 코드 구성요소 개수 / 전체코드 구성 요소 개수 \* 100

# Clover

## | Test Coverage

/ 명세기반 Test Coverage만으로는 신뢰성이 떨어지고, Code Coverage만으로는 코드로 표현할 수 없는 데이터베이스 스키마 등 코드 외적인 요소는 측정할 수 없으므로 시스템의 안정성을 보장할 수 없다.

/ 따라서, 두가지 Test Coverage를 함께 사용하여야 한다.

# Clover

## Code Coverage의 유형

### 기본적인 방법

Statement Coverage

Decision Coverage

Condition Coverage

Multiple Condition Coverage

Condition/Decision Coverage

Modified Condition/Decision Coverage

Path Coverage

# Clover

- | Code Coverage의 유형

- / 그 밖의 방법들

- / Function Coverage

- / Call Coverage

- / Linear Code Sequence and Jump (LCSAJ) Coverage

- / Data Flow Coverage

- / Object Code Branch Coverage

- / Loop Coverage

- / Race Coverage

- / Relational Operator Coverage

- / Weak Mutation Coverage

- / 각 유형에 따라서 같은 코드 구성 요소를 보더라도 다른 결과가 나오므로 필요에 따라서 적절한 유형을 선택해야 함.

# Clover

## Code Coverage 측정 도구

주로 소스코드를 컴파일하기 전에 Coverage를 측정하기 위한 계측코드를 삽입하고 테스트하면서 계측코드가 얼마나 실행되었는지 측정한다. 따라서 언어마다 사용하는 도구가 다르다.

### Java

Cobertura, EMMA, Clover ...

### C/C++

GCOV, BullsEyeCoverage ...



# Clover

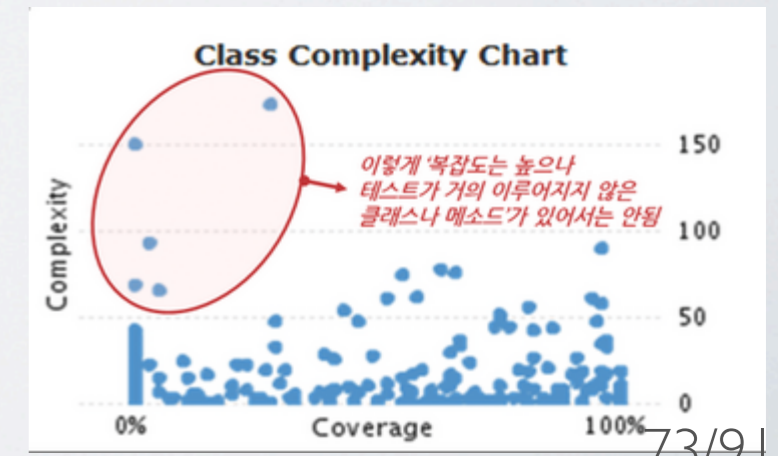
## 개요

/ <http://atlassian.com/software/clover>

/ Atlassian이라는 애자일 지원 도구 전문업체가 개발

/ 소스코드에 직접 계측코드를 삽입하여 Code Coverage를 측정, Eclipse, Ant, Maven 등과 쉽게 통합할 수 있다.

/ 분석 결과 중 하나로 클래스의 복잡도와 Code Coverage 분석 결과를 함께 보여주는 그래프.



# Clover

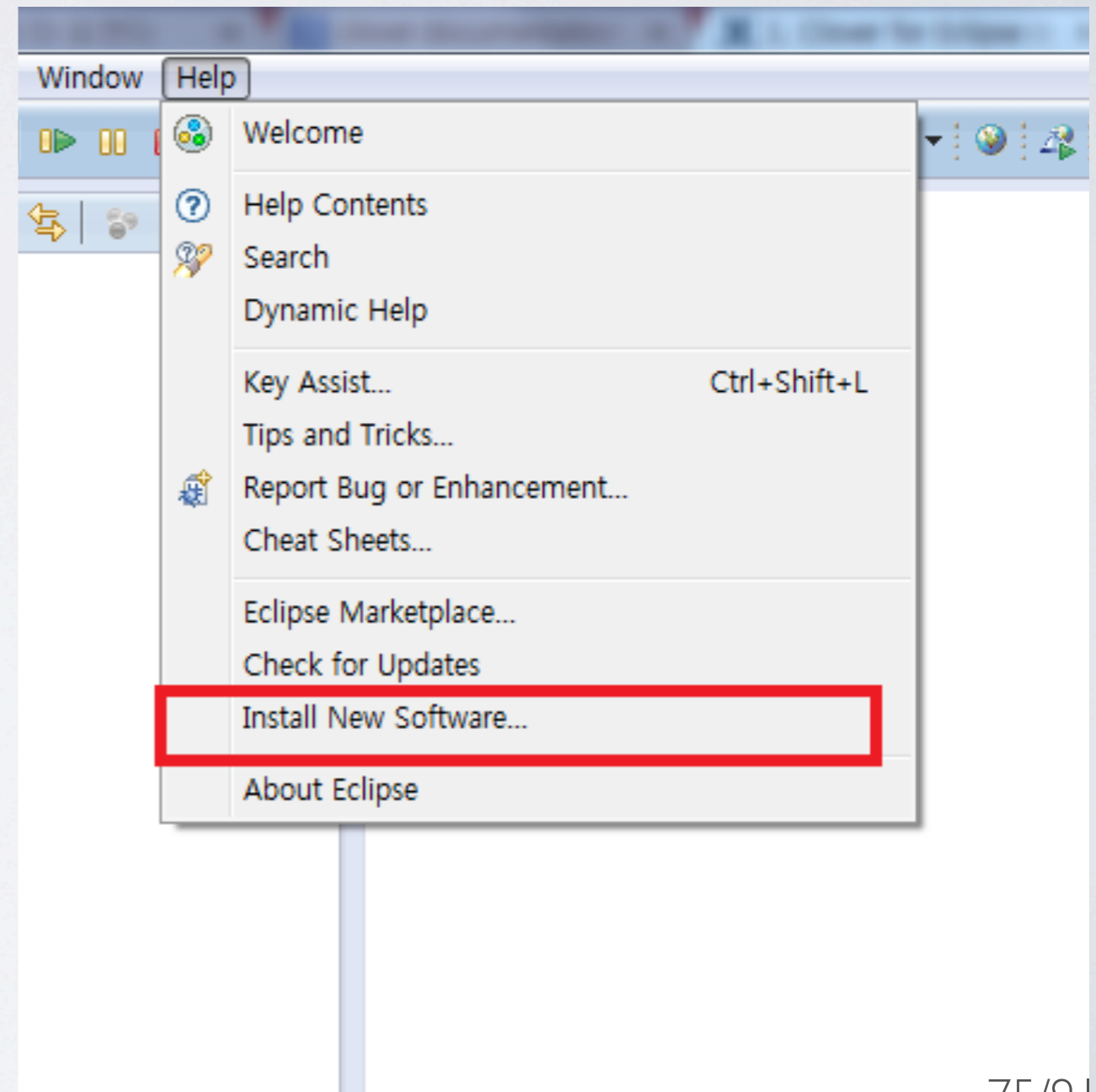
## | 동작 원리

- / 컴파일러 호출을 가로채어 소스코드에 계측 코드를 삽입
- / 자바 컴파일러로 컴파일
- / 계측 코드를 삽입할 때 clover.db라는 파일 생성
- / 최종 바이너리 실행중에 clover.db에 실행된 코드를 기록
- / 실행 이후 clover.db의 내용을 html 또는 xml 등의 형식으로 출력

# Clover

설치과정

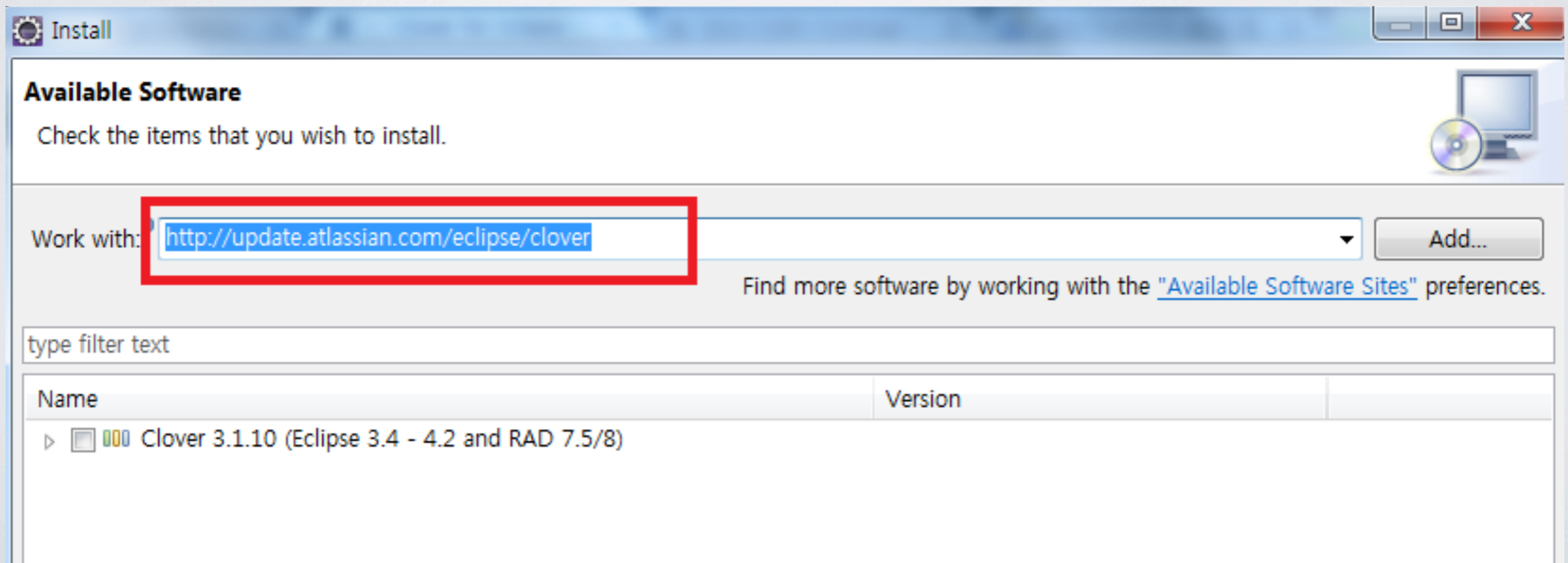
Help -> Install New Software 선택



# Clover

## 설치과정

Work With에 <http://update.atlassian.com/eclipse/clover> 입력 후 Add



# Clover

## 설치과정

설치할 Clover Plugin 선택 후 Next

type filter text

Name	Version
<input checked="" type="checkbox"/> Clover 3.1.10 (Eclipse 3.4 - 4.2 and RAD 7.5/8)	
<input checked="" type="checkbox"/> Clover 3 (for Eclipse 3.4 - 4.2 and RAD 7.5/8)	3.1.10.v20130108000000
<input checked="" type="checkbox"/> Clover 3 Ant Support (for Eclipse 3.4 - 4.2 and RAD 7.5/8)	3.1.10.v20130108000000

Show only software applicable to target environment  
 Contact all update sites during install to find required software

? < Back **Next >** Finish Cancel

# Clover

## 설치과정

/ I accept .. 선택 후 Finish 클릭

/ 그 후 Restart now 클릭

the use of the Product.

1. Definitions

Accessible Code means source code contained within the Product that is unprotected and accessible under this agreement.

Atlassian means Atlassian Pty. Ltd (ABN 53 102 443 916) of 173-185 Sussex Street, Sydney, New South Wales 2000 Australia.

Authorised Machine means a single installation of a copy of the Product on a single physical computer.

Authorised Server Node means a single installation of a copy of the Product within a J2EE application server on a single physical server, which is either stand alone or within a

I accept the terms of the license agreement

I do not accept the terms of the license agreement

< Back   Next >   Finish   Cancel

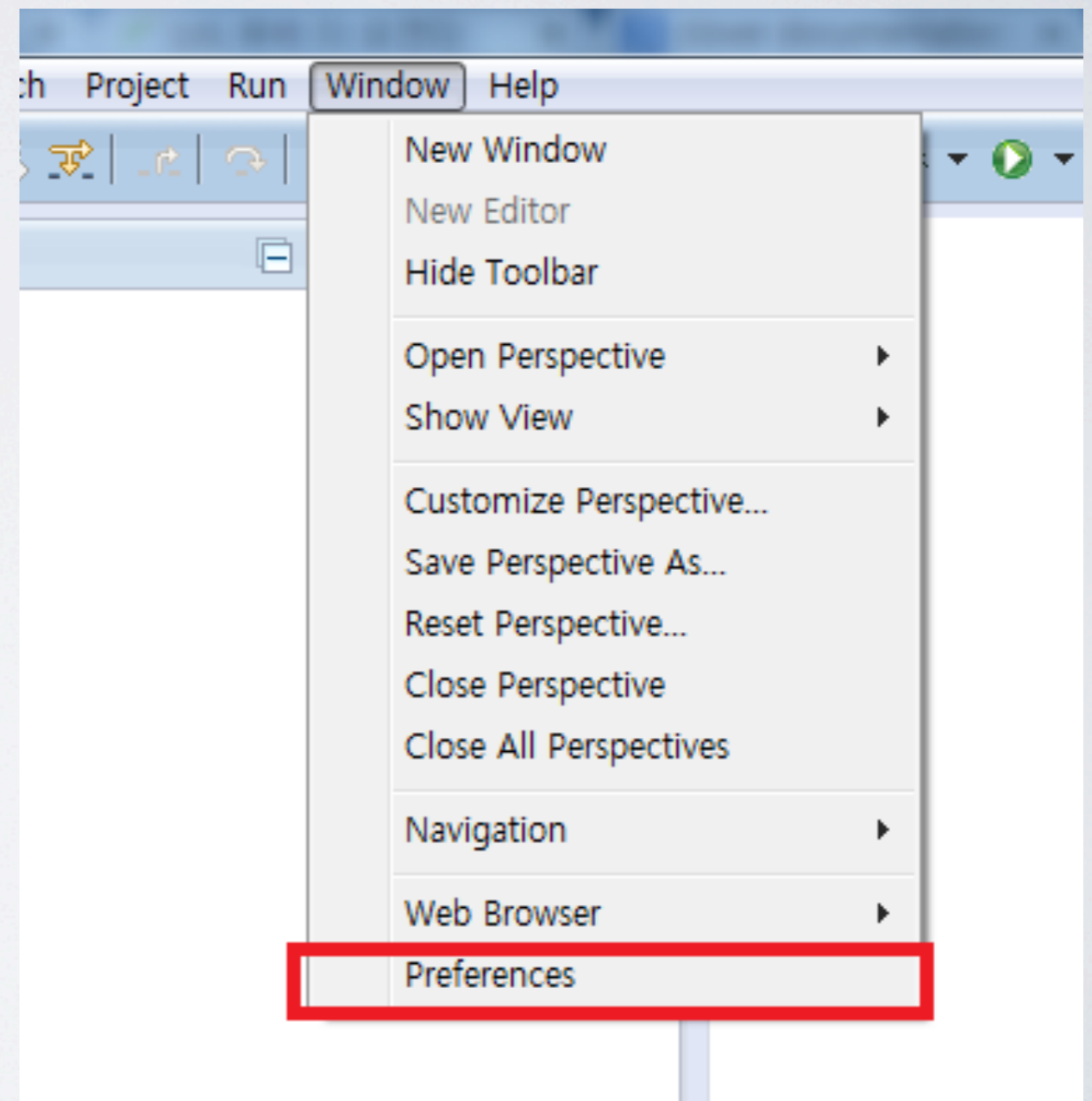
# Clover

## 설치과정

/ License Key 입력 (필요시에)

/ Window -> Preferences  
-> Clover -> License

/ (기본으로 30일 체험판을 제공해준다)



# Clover

## 설치과정

License Key 입력하고, OK 클릭

Ant

**Clover**

License

Test Optimization

Data Management

Help

Install/Update

Java

Java EE

Java Persistence

JavaScript

Mylyn

Plug-in Development

Remote Systems

Run/Debug

Server

Team

Summary

Status: You have 29 day(s) before your license expires.

Type: Clover: Evaluation

Message: Clover Evaluation License registered to Clover Plugin.

SID: BIUI-98NQ-ANCQ-5BMW

License Text

Insert the contents of your Clover license below.

You can [generate an evaluation license](#) (registration required) or you can [retrieve your existing license](#) on the Atlassian website..

Paste

Load...



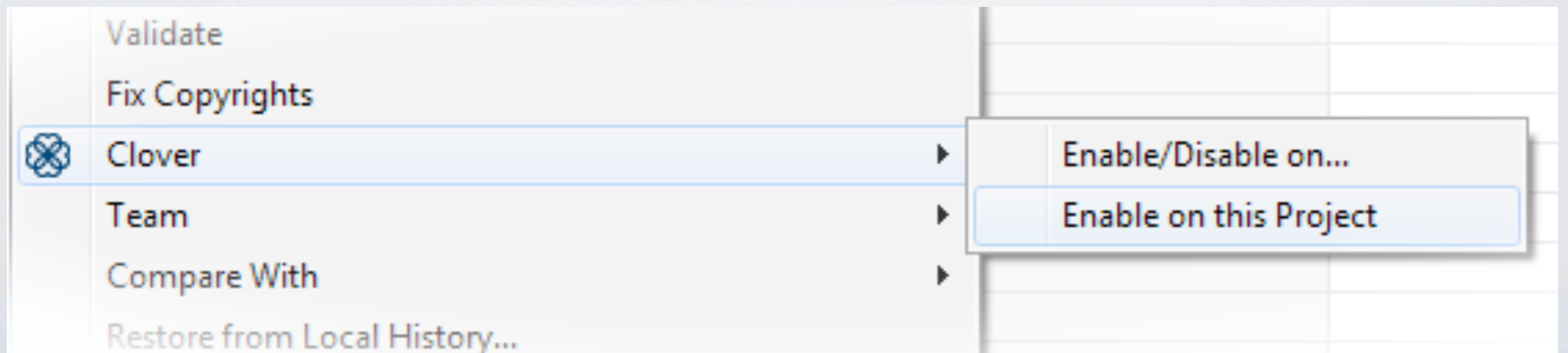
# Clover

## 사용법

/ Project의 오른쪽 클릭

/ Clover

/ Enable on this Project



# Clover

## 사용법

/ 기본적으로 4가지 항목에 대한 것을 보여준다.

/ Coverage Explorer, Test Run Explorer, Test Contributions, Clover Dashboard

The screenshot displays the Clover Coverage Explorer interface. The table below shows the coverage data for the selected package and its classes.

Elem	Cov%	Av Me Cpx	Cpx
Moneybags	0,0%	1,4	80,0
com.cenqua.samples.money	0,0%	1,4	80,0
IMoney.java	0,0%	-	0,0
Money.java	0,0%	1,3	18,0
MoneyBag.java	0,0%	2,1	36,0
MoneyBagTest.java	0,0%	1,0	25,0
MoneyTest.java	0,0%	1,0	1,0

The right-hand side of the interface shows the following metrics for the selected package:

**Structure**

- Packages: -
- Files: 5
- Classes: 5
- Methods: 57
- Statements: 165
- Branches: 46

**Test Executions**

- Executed Tests: 0
- Passes: 0
- Fails: 0
- Errors: 0

**Source**

- LOC: 479
- Total Cmp: 80
- Avg Method Cmp: 1,4
- NC LOC: 318
- Cmp Density: 0,5

# Clover

## Coverage Explorer

프로젝트의 통계가 표시된다.

The screenshot displays the Clover Coverage Explorer interface. The main table shows the following data:

Elem	Cov%	Av Me Cpx	Cpx
Moneybags	94,8%	1,4	80,0
com.cenqua.samples.money	94,8%	1,4	80,0
IMoney.java	0,0%	-	0,0
Money.java	89,4%	1,3	18,0
Money	89,4%	1,3	18,0
MoneyBag.java	94,4%	2,1	36,0
MoneyBagTest.java	97,8%	1,0	25,0
MoneyTest.java	100,0%	1,0	1,0
MoneyTest	100,0%	1,0	1,0
testAdd()	100,0%	-	1,0

On the right, the 'Metrics for: Moneybags' panel provides a summary:

- Structure:** Packages: 1, Files: 5, Classes: 5, Methods: 57, Statements: 165, Branches: 46
- Test Executions:** Executed Tests: 23, Passes: 23, Fails: 0, Errors: 0
- Source:** LOC: 479, Total Cmp: 80, Avg Method Cmp: 1,4, NC LOC: 318, Cmp Density: 0,5

# Clover

## Coverage Dashboard

프로젝트에 대한 개요를 가지고 있고, 초점을 맞춰야 할 코드 영역에 대한 정보를 찾을 수 있는 곳.

### Coverage

Test Coverage의 비율에 대한 정보

### Test Results

테스트 결과에 대한 정보

### Most Complex Packages

가장 복잡성을 가진 Package를 보여준다

### Most Complex Classes

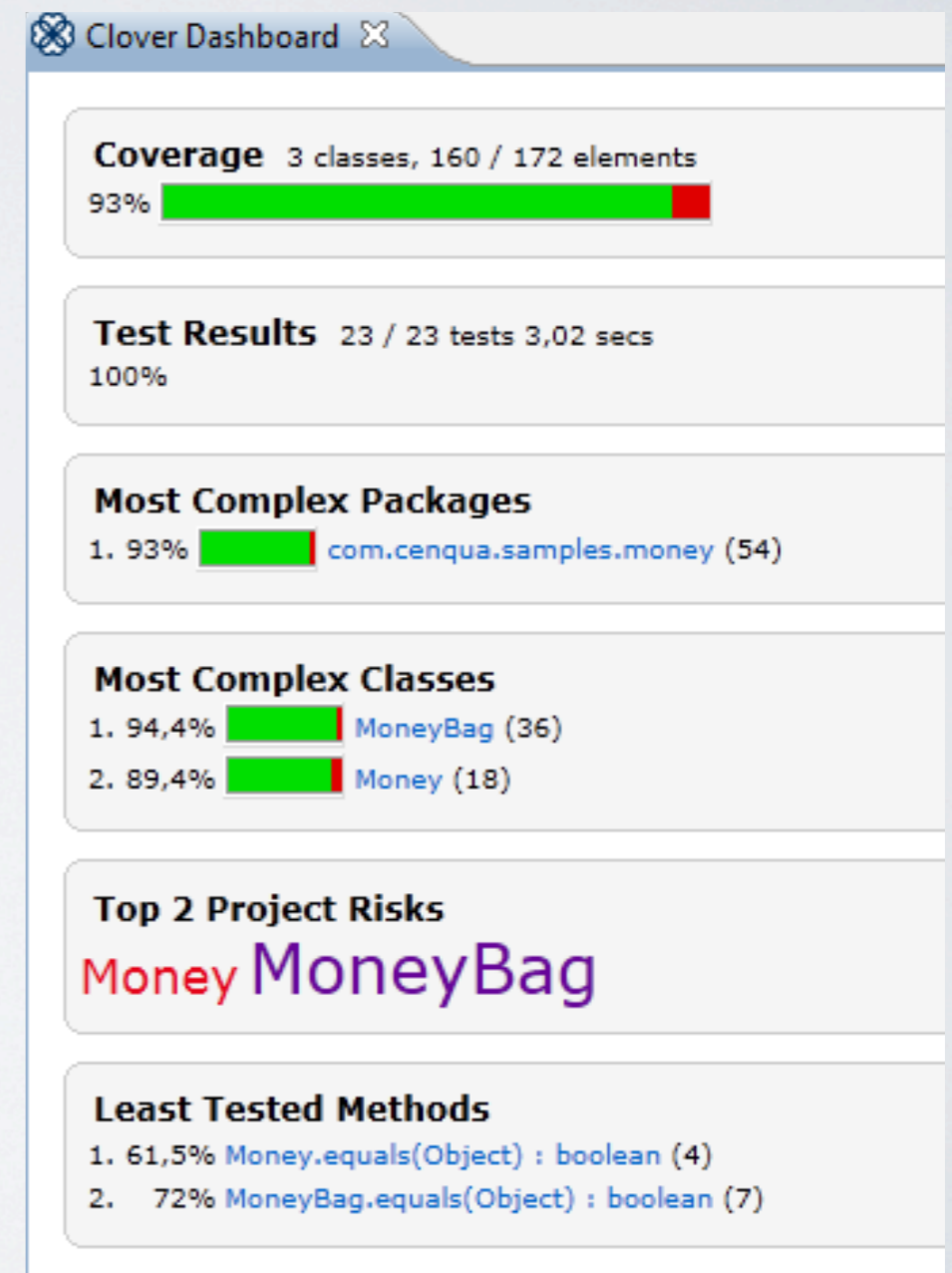
가장 복잡성을 가진 Class를 보여준다

### Top Project Risks

가장 복잡하고 적은 Test Coverage를 가진 Class를 보여준다

### Least Tested Methods

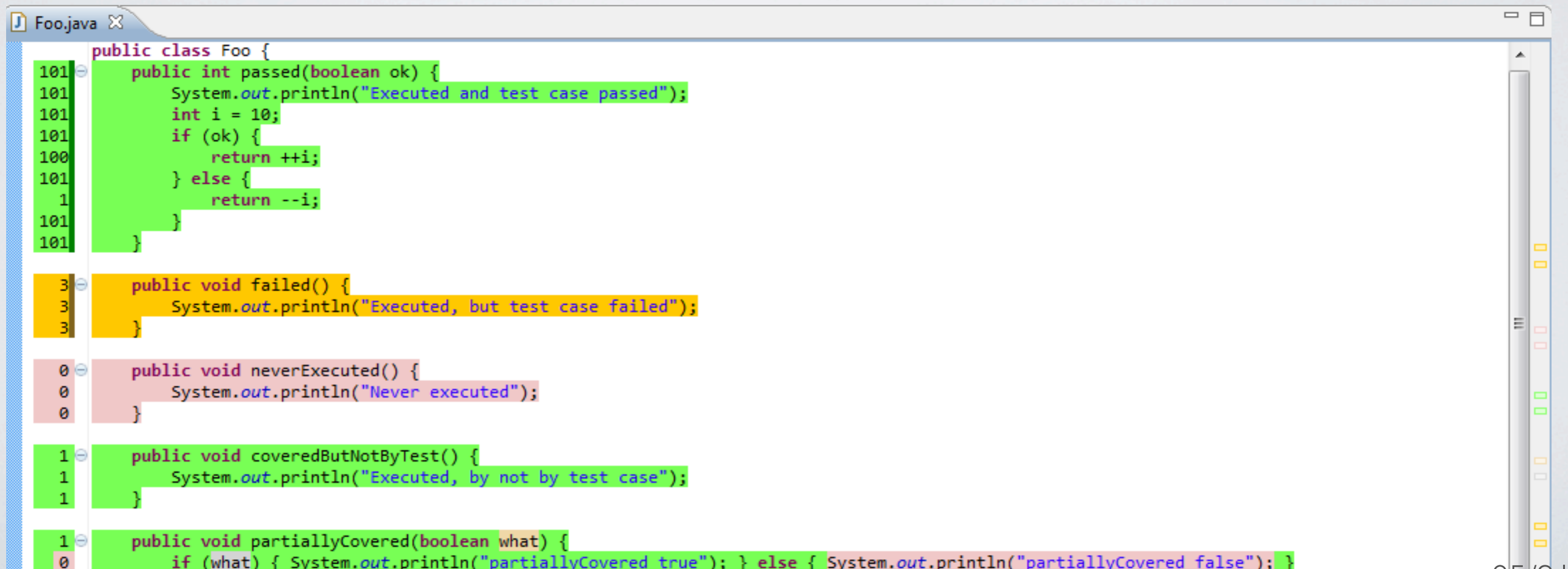
가장 낮은 Test Coverage의 Methods을 보여준다



# Clover

## Java Editor

프로젝트의 자바소스 코드편집기에 색상으로 주석을 추가한다.



```
Foo.java x
public class Foo {
101 public int passed(boolean ok) {
101     System.out.println("Executed and test case passed");
101     int i = 10;
101     if (ok) {
100         return ++i;
101     } else {
1     return --i;
101     }
101 }

3 public void failed() {
3     System.out.println("Executed, but test case failed");
3 }

0 public void neverExecuted() {
0     System.out.println("Never executed");
0 }

1 public void coveredButNotByTest() {
1     System.out.println("Executed, by not by test case");
1 }

1 public void partiallyCovered(boolean what) {
0     if (what) { System.out.println("partiallyCovered true"); } else { System.out.println("partiallyCovered false"); }
```

# Clover

## Java Editor

- / 초록색  
Test를 합격하거나 Main Method처럼 외부 Test를 하는 라인
- / 노란색  
실패한 Test Coverage
- / 회색  
제외된 코드
- / 빨간색  
Coverage가 없는 Code
- / 물결표시 붉은선  
부분 분기 Coverage(분기의 한부분만 포함되어있으면 발생)

```
Foo.java x
public class Foo {
101 public int passed(boolean ok) {
101     System.out.println("Executed and test case passed");
101     int i = 10;
101     if (ok) {
100         return ++i;
101     } else {
1     return --i;
101     }
101 }

3 public void failed() {
3     System.out.println("Executed, but test case failed");
3 }

0 public void neverExecuted() {
0     System.out.println("Never executed");
0 }

1 public void coveredButNotByTest() {
1     System.out.println("Executed, by not by test case");
1 }

1 public void partiallyCovered(boolean what) {
0     if (what) { System.out.println("partiallyCovered true"); }
1 }

1 public void partiallyCoveredWithFailed(boolean what) {
0     if (what) { System.out.println("partiallyCoveredWithFailed"); }
1 }
```

# Clover

## Test Run Explorer

최근에 실행된 테스트를 탐색할 수 있는 곳으로 개별 테스트, 테스트 클래스, 패키지 또는 전체 프로젝트에 의한 Code Coverage를 볼 수 있다

The screenshot shows the Test Run Explorer and Coverage Contribution panels in an IDE. The Test Run Explorer panel displays a table of test results, and the Coverage Contribution panel displays a table of code coverage data.

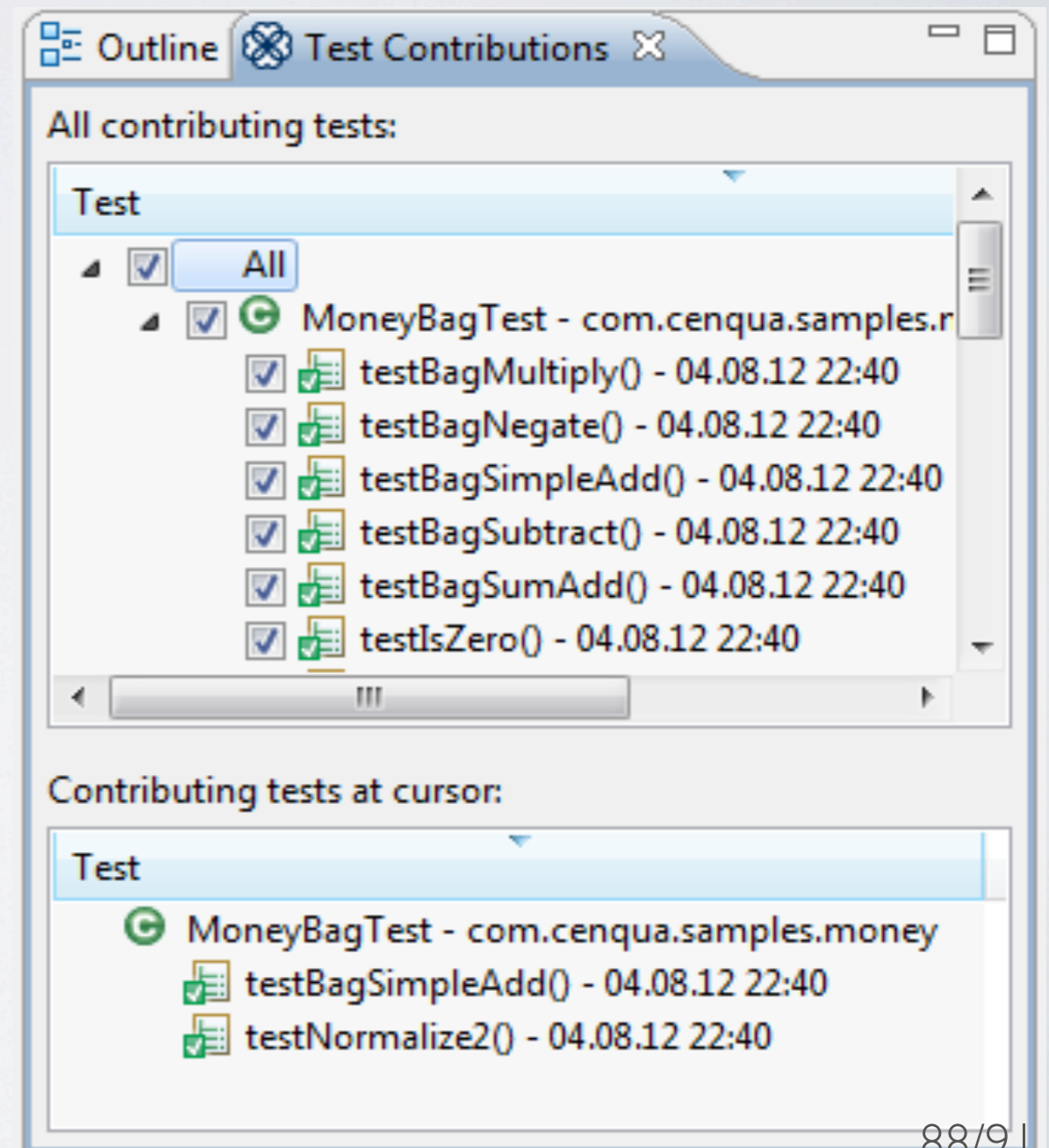
Test	Start	Rslt	Time	Message
Moneybags				
MoneyBagTest.testBagMultiply()	05.09.12 11:35	PASS	0.001s	
MoneyBagTest.testBagNegate()	05.09.12 11:35	PASS	0.001s	
MoneyBagTest.testBagNotEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSimpleAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSubtract()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testBagSumAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testIsZero()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMixedSimpleAdd()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyBagEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyBagHash()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyEquals()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testMoneyHash()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize2()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize3()	05.09.12 11:35	PASS	0.0s	
MoneyBagTest.testNormalize4()	05.09.12 11:35	PASS	0.001s	

Class	Contrib%	Uniq%
Money	27,7%	0,0%
Money	100,0%	0,0%
amount	100,0%	0,0%
appendTo	100,0%	0,0%
currency	100,0%	0,0%
isZero	100,0%	0,0%
multiply	100,0%	0,0%
MoneyBag	48,0%	8,8%
appendMoney	41,2%	0,0%
contains	66,7%	0,0%
create	100,0%	0,0%
equals	56,0%	0,0%
findMoney	100,0%	0,0%
isZero	100,0%	0,0%
multiply	100,0%	100,0%
simplify	60,0%	0,0%

# Clover

## Test Contributions

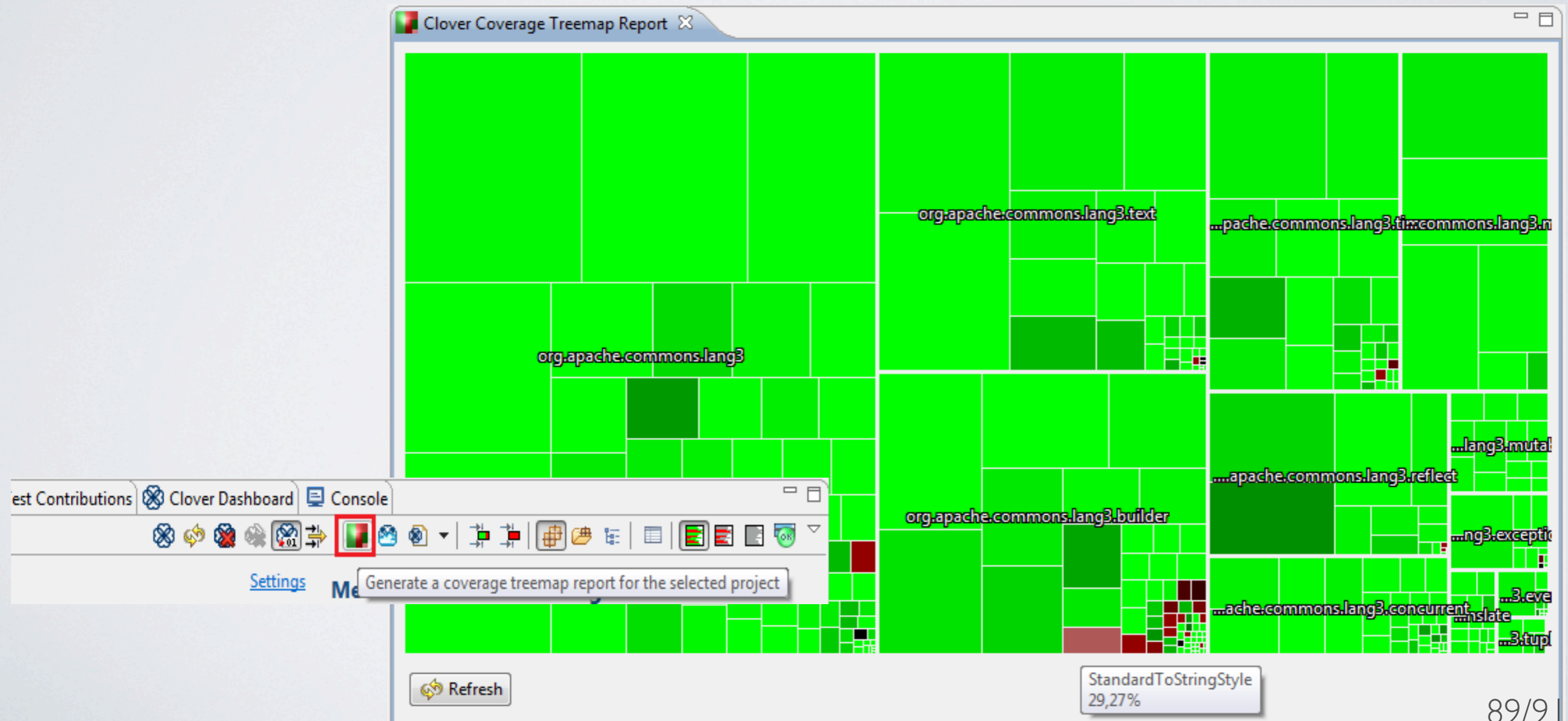
- Unit Test와 선택한 자바소스파일에 대한 Coverage를 보여준다.





# Clover

## Coverage Treemap Report



# Clover

## ┆ Coverage Treemap Report

┆ 복잡성과 Code Coverage에 의한 Class와 Package를 동시에 비교하여 보여준다.

┆ Treemap은 Package에 의해 나눈후, 다시 한번 Class에 의해 나누어 진다. 패키지와 클래스의 크기는 복잡성을 보여준다. (더 큰 사각형은 더 복잡하다는 것을 의미)

┆ 색상은 Coverage의 수준을 보여준다. (초록은 가장 Coverage가 잘 되었다는 것을 의미하고 붉은색은 그 반대이다.)

exit;\_