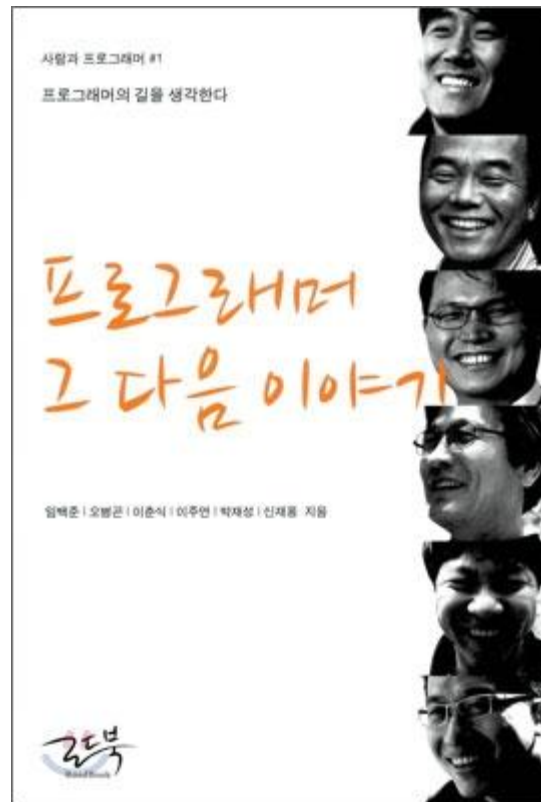


# 자바 웹 개발자 교육

—

<http://www.javajigi.net>

<http://www.slipp.net>



# 강의 소개

—



# 강의 목표

자바를 활용한 객체 지향 개발

Spring 프레임워크의 핵심 개념 이해

JPA(ORM)를 활용한 웹 애플리케이션 개발

Grails를 활용한 웹 애플리케이션 개발

# 오늘 학습 목표

메이븐 빌드 도구 이해

Test Driven Development(이하 TDD)

# 개발 환경

—



JDK 6.0 이상

Eclipse Java EE Developers 4.3 버전

Maven 최신 버전

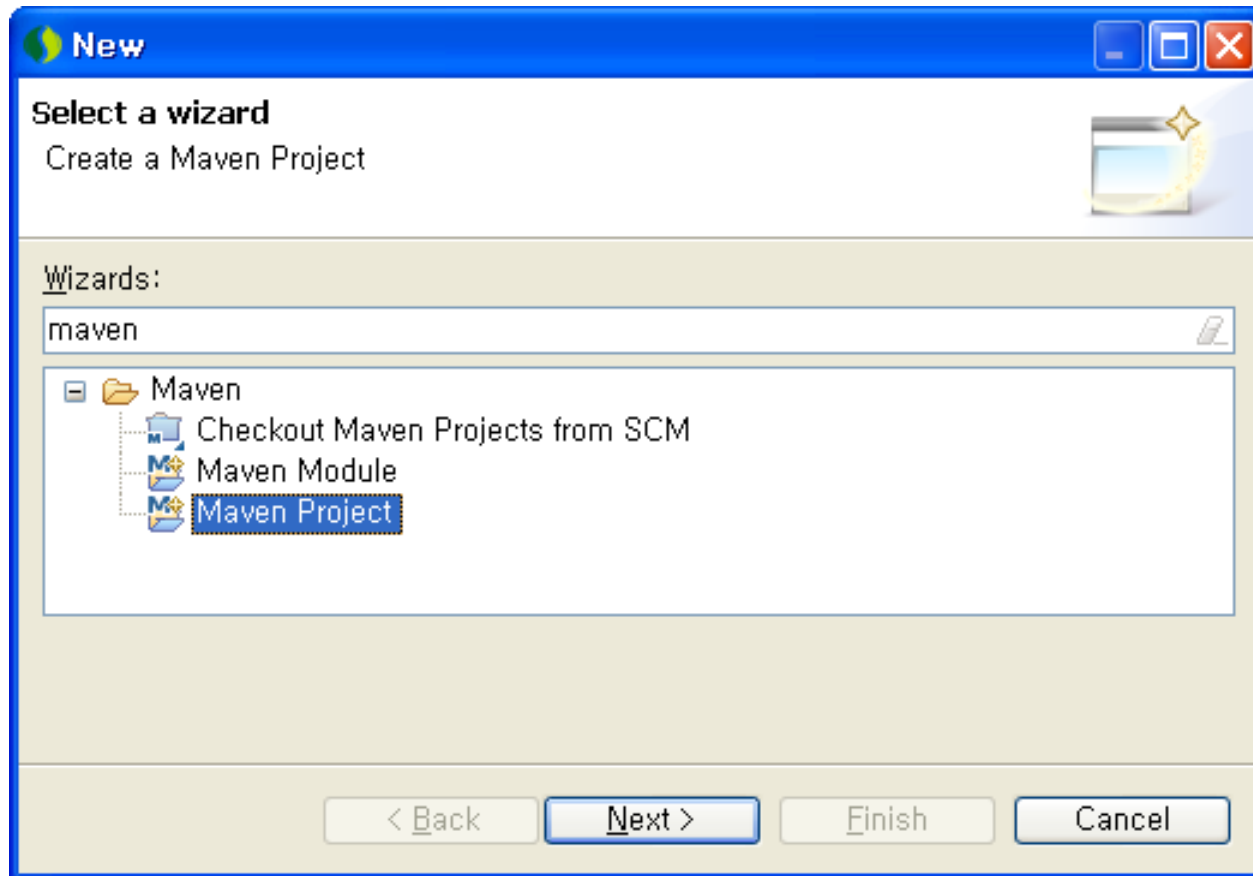
# 메이븐

-

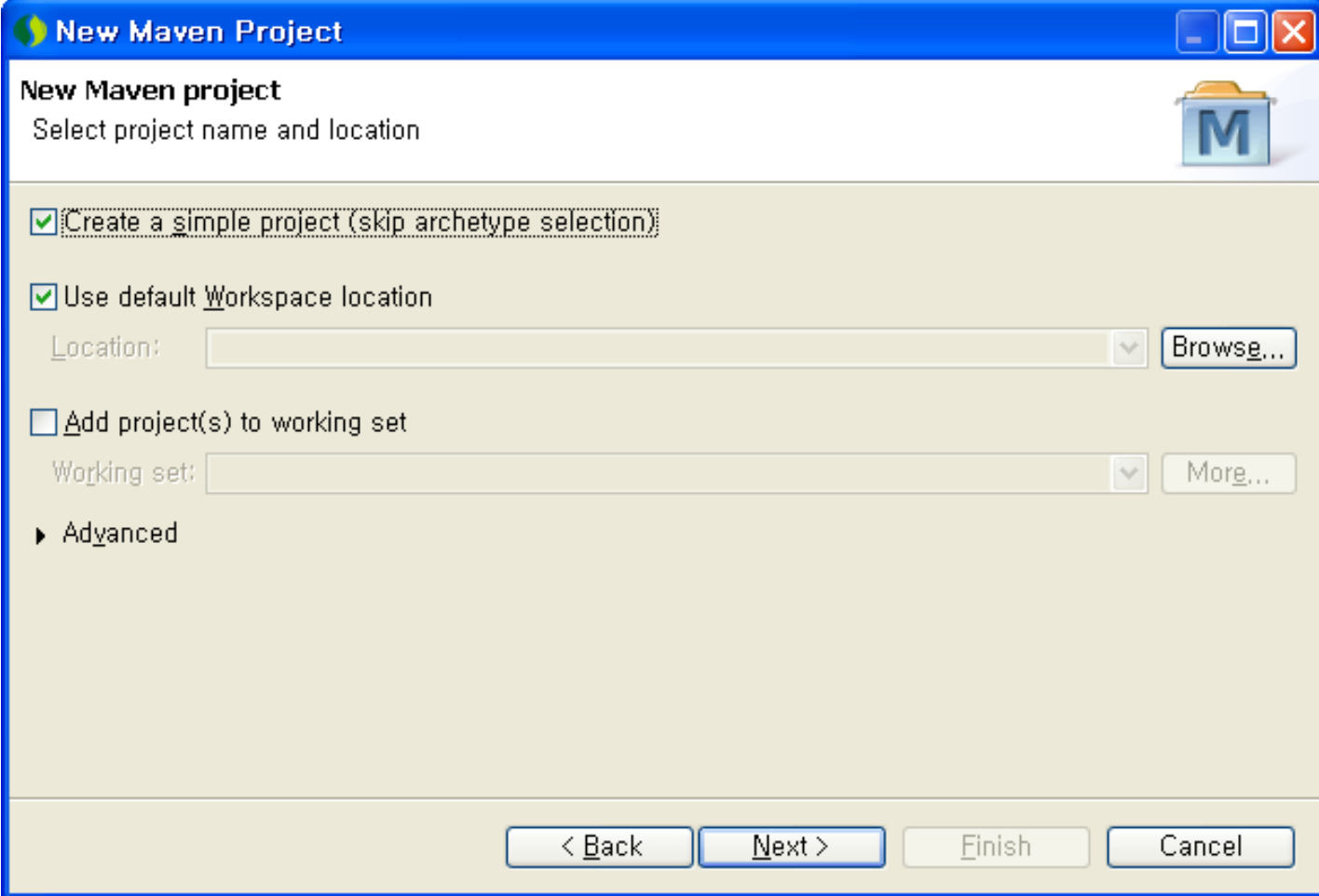




Ctrl + N 단축키 >> Maven으로 검색 >> Maven project 선택 >> Next



Create a simple project(…) 체크박스 선택 >> Next



The image shows a 'New Maven Project' dialog box with a blue title bar. The main area has a light beige background. At the top, it says 'New Maven project' and 'Select project name and location'. There is a small icon of a folder with an 'M' on it. Below this, there are three checked checkboxes: 'Create a simple project (skip archetype selection)', 'Use default Workspace location', and 'Add project(s) to working set'. The 'Location:' field is empty, and the 'Working set:' field is also empty. There are 'Browse...' and 'More...' buttons next to these fields. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

**New Maven Project**

New Maven project  
Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location:  Browse...

☐ Add project(s) to working set

Working set:  More...

► Advanced

< Back Next > Finish Cancel

프로젝트 생성

groupId : net.slipp, artifactId : calculator 입력 >> Finish

**New Maven Project**

New Maven project  
Configure project

**Artifact**

Group Id: net.slipp

Artifact Id: calculator

Version: 0.0.1-SNAPSHOT

Packaging: jar

**Parent Project**

Group Id:

Artifact Id:

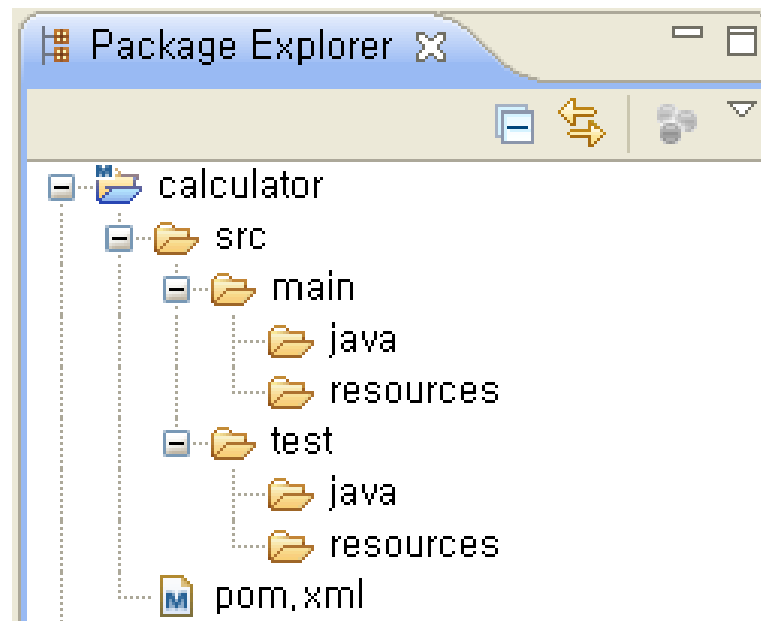
Version: Browse... Clear

► Advanced

< Back Next > Finish Cancel

프로젝트 생성

## Maven project 기본 디렉토리 구조



# 메이븐 설치

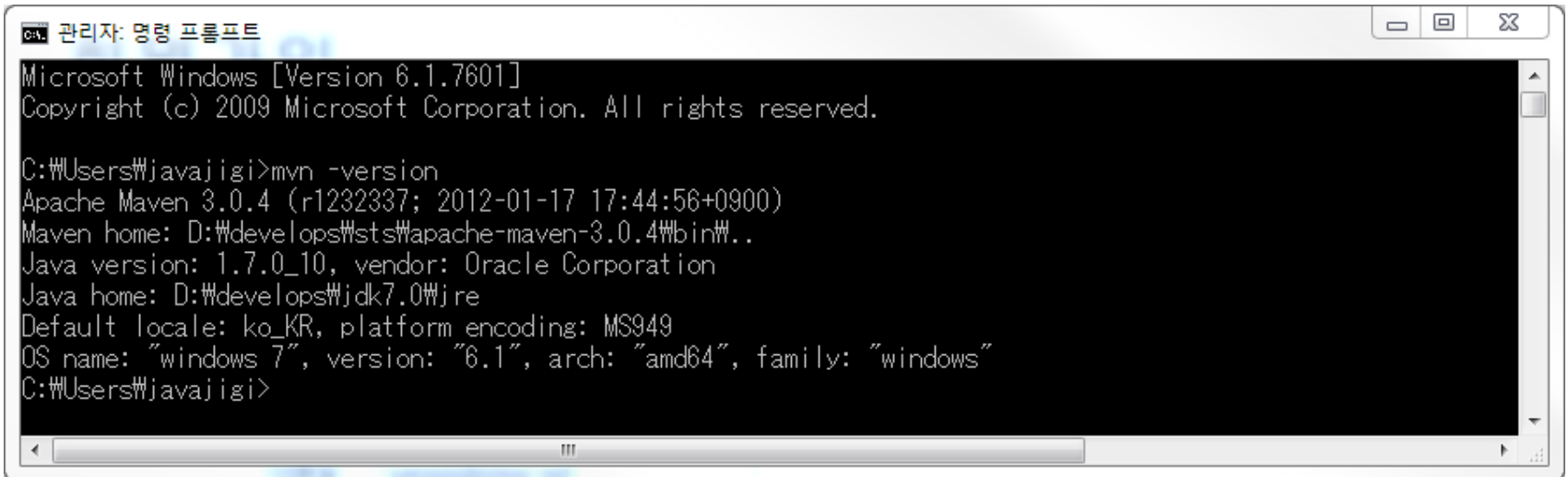
<http://maven.apache.org>에서 최신 버전 다운로드(3.0.4)

압축을 풀면 설치 완료

시스템 환경 변수에 MAVEN\_HOME 설정

PATH에 MAVEN\_HOME/bin 추가

명령 프롬프트에서 “mvn -version” 실행하여 버전 확인



```
관리자: 명령 프롬프트
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\javajigi>mvn -version
Apache Maven 3.0.4 (r1232337; 2012-01-17 17:44:56+0900)
Maven home: D:\develops\sts\apache-maven-3.0.4\bin\..
Java version: 1.7.0_10, vendor: Oracle Corporation
Java home: D:\develops\jdk7.0\jre
Default locale: ko_KR, platform encoding: MS949
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
C:\Users\javajigi>
```

# 메이븐 eclipse 플러그인 설치

<http://www.slipp.net/wiki/pages/viewpage.action?pagelId=13271090> 문서를 참고

# Ant vs Maven

Jdbc vs Spring Jdbc

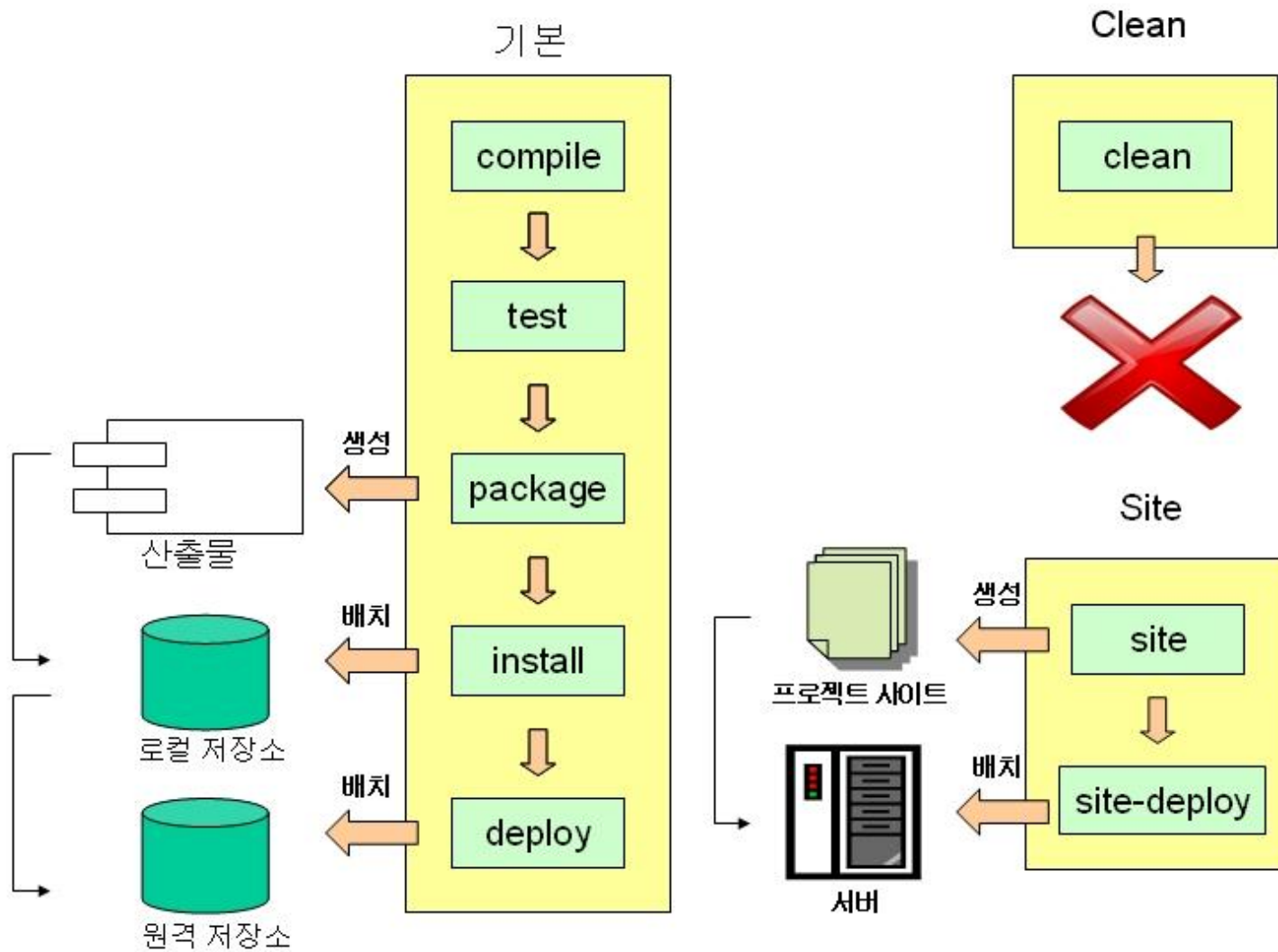


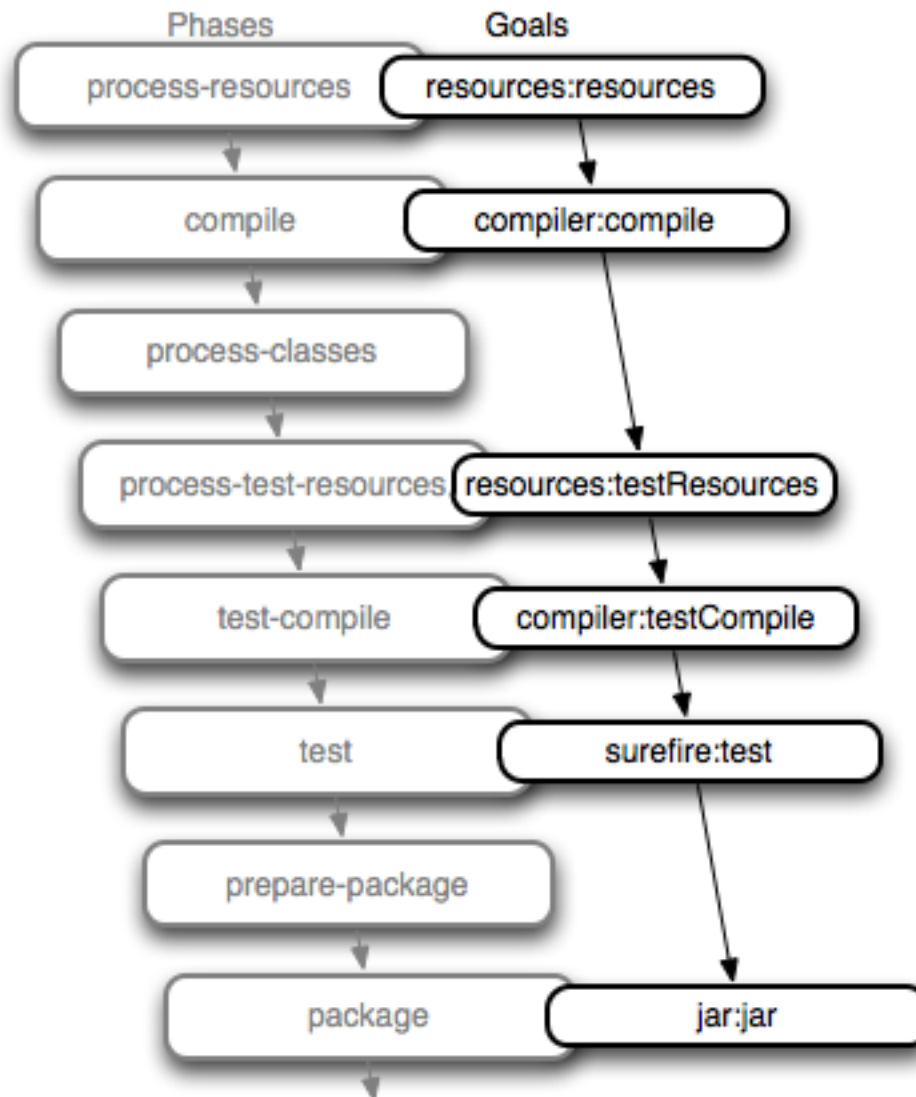
# Convention over Configuration

## 설정정보다는 관례

# 메이븐 기본 설정 파일

pom.xml





Note: There are more phases than shown above, this is a partial list

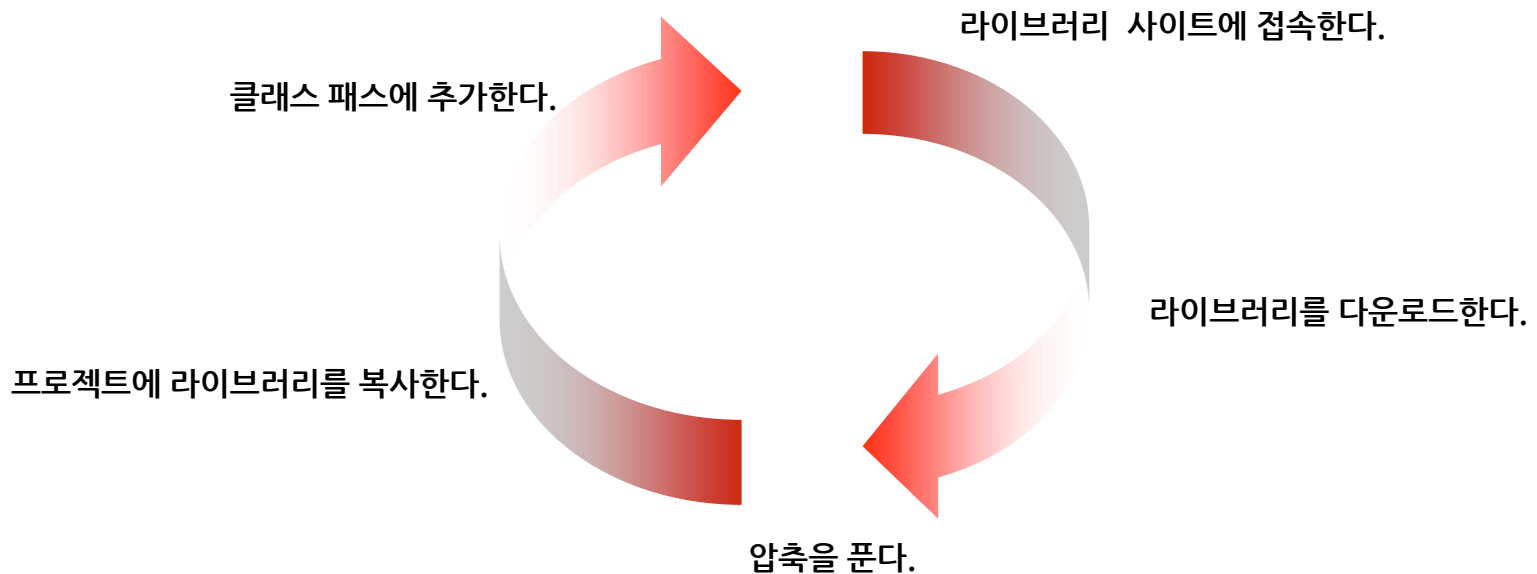
페이즈

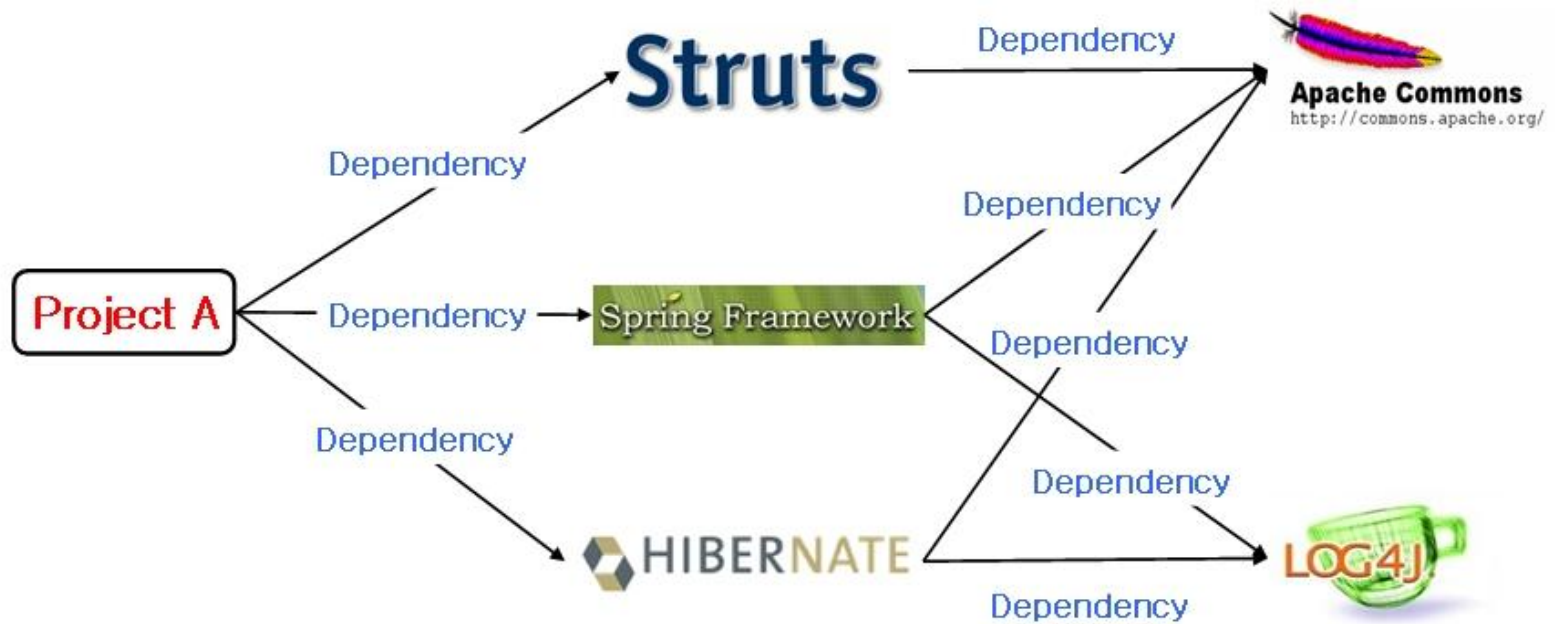
골

compile

compiler:compile

# 과거의 의존성 라이브러리 관리





```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

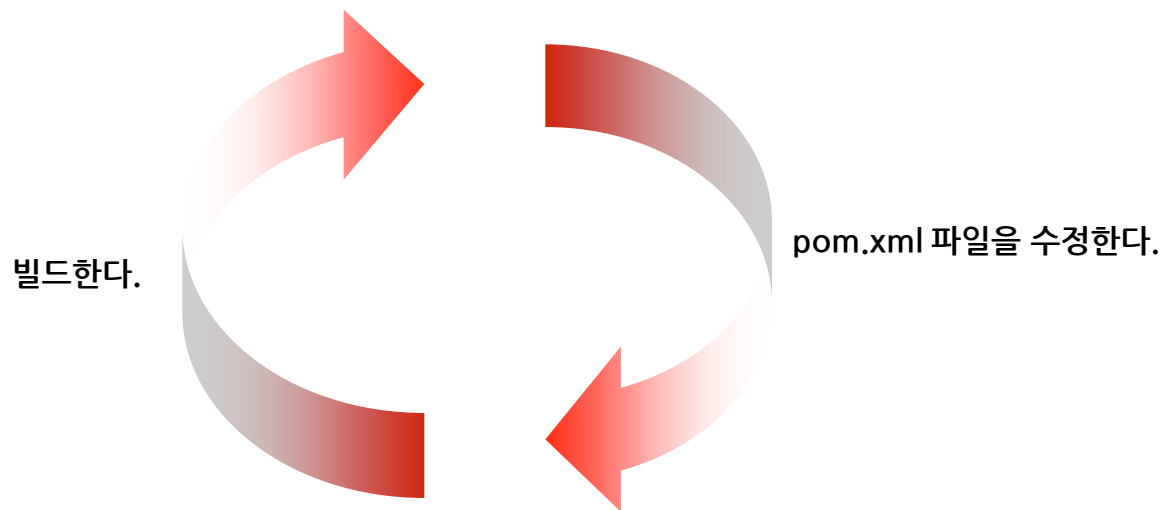
USER\_HOME/.m2/repository/junit/junit/3.8.1/junit-3.8.1.jar



```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

**mvn eclipse:clean eclipse:eclipse**

# Maven에서 의존성 라이브러리 관리



# Spring 프레임워크 설치

<http://mvnrepository.com> 접근한다.

org.springframework로 검색

spring-jdbc 모듈과 spring-web 모듈 추가한다.

mvn eclipse:clean eclipse:eclipse

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-jdbc</artifactId>  
  <version>3.2.3.RELEASE</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-web</artifactId>  
  <version>3.2.3.RELEASE</version>  
</dependency>
```

## 공통 값을 변수로 추출

```
<properties>  
  <org.springframework.version>3.2.3.RELEASE</org.springframework.version>  
</properties>
```

`${org.springframework.version}` 로 접근한다.

# webapps 경로 변경

```
<plugin>  
  <artifactId>maven-war-plugin</artifactId>  
  <configuration>  
    <warSourceDirectory>webapp</warSourceDirectory>  
  </configuration>  
</plugin>
```

## wtp에 배포할 수 있도록 설정

```
<plugin>
  <artifactId>maven-eclipse-plugin</artifactId>
  <version>2.6</version>
  <configuration>
    <wtpversion>2.0</wtpversion>
    <wtpContextName>/</wtpContextName>
  </configuration>
</plugin>
```

# Java compiler 버전 설정

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.3.2</version>
  <configuration>
    <source>1.6</source>
    <target>1.6</target>
    <encoding>utf-8</encoding>
  </configuration>
</plugin>
```

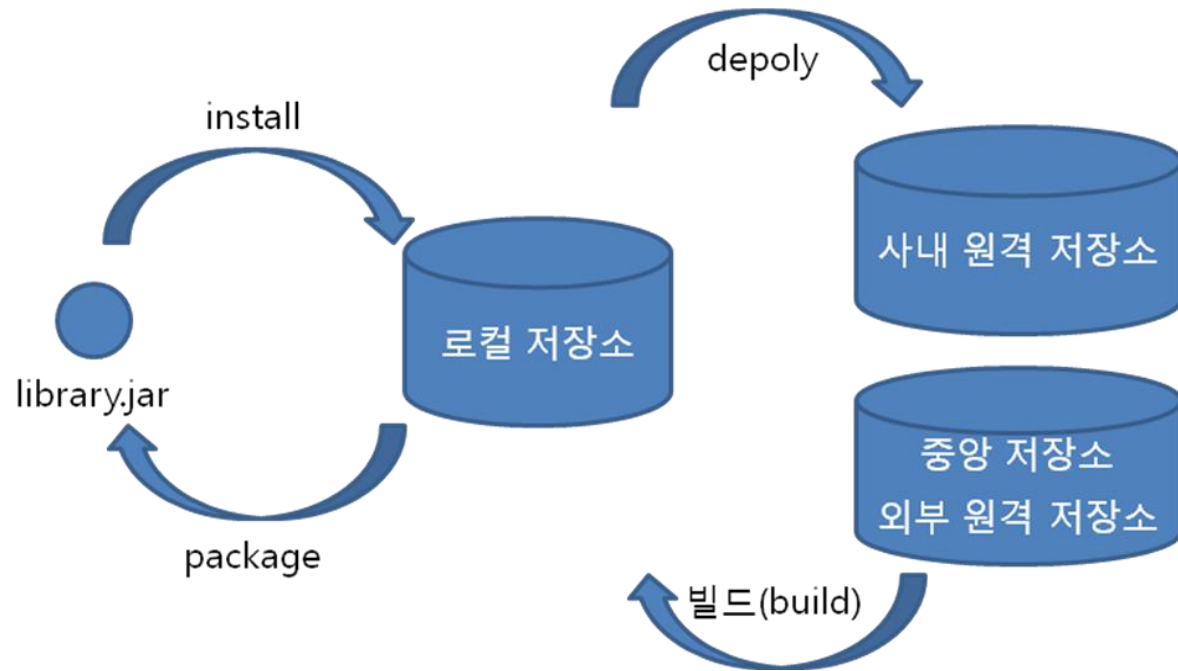
## slipp-user 프로젝트로 실습

- <https://github.com/javajigi/slipp-user>에 접근해 url 복사
- <http://www.slipp.net/wiki/pages/viewpage.action?pageId=4161582> 문서 참고해 이클립스와 github 연동
- slipp-user 프로젝트 import
- mvn ecilpse:clean eclipse:eclipse
- <http://www.slipp.net/wiki/pages/viewpage.action?pageId=5177373> 문서 참고해 WTP에 프로젝트 배포
- 테스트



로컬 기본 저장소 : C:\Documents and Settings\\${user.id}\.m2

Central : <http://repo1.maven.org/maven2>



Maven 중앙 저장소 : <http://repo1.maven.org/maven2/>

라이브러리 검색 : <http://mvnrepository.com/>

참고 서적 : 자바 세상의 빌드를 이끄는 메이븐, 박재성



# TDD - 초간단 계산기 구현

—

## 사칙연산이 가능한 계산기

- 덧셈(add)
- 뺄셈(subtract)
- 곱셈(multiply)
- 나눗셈(divide)

## Production Code

```
public class Calculator {  
    int add(int i, int j) {  
        return i + j;  
    }  
  
    int subtract(int i, int j) {  
        return i - j;  
    }  
  
    int multiply(int i, int j) {  
        return i * j;  
    }  
  
    int divide(int i, int j) {  
        return i / j;  
    }  
}
```

## Test Code

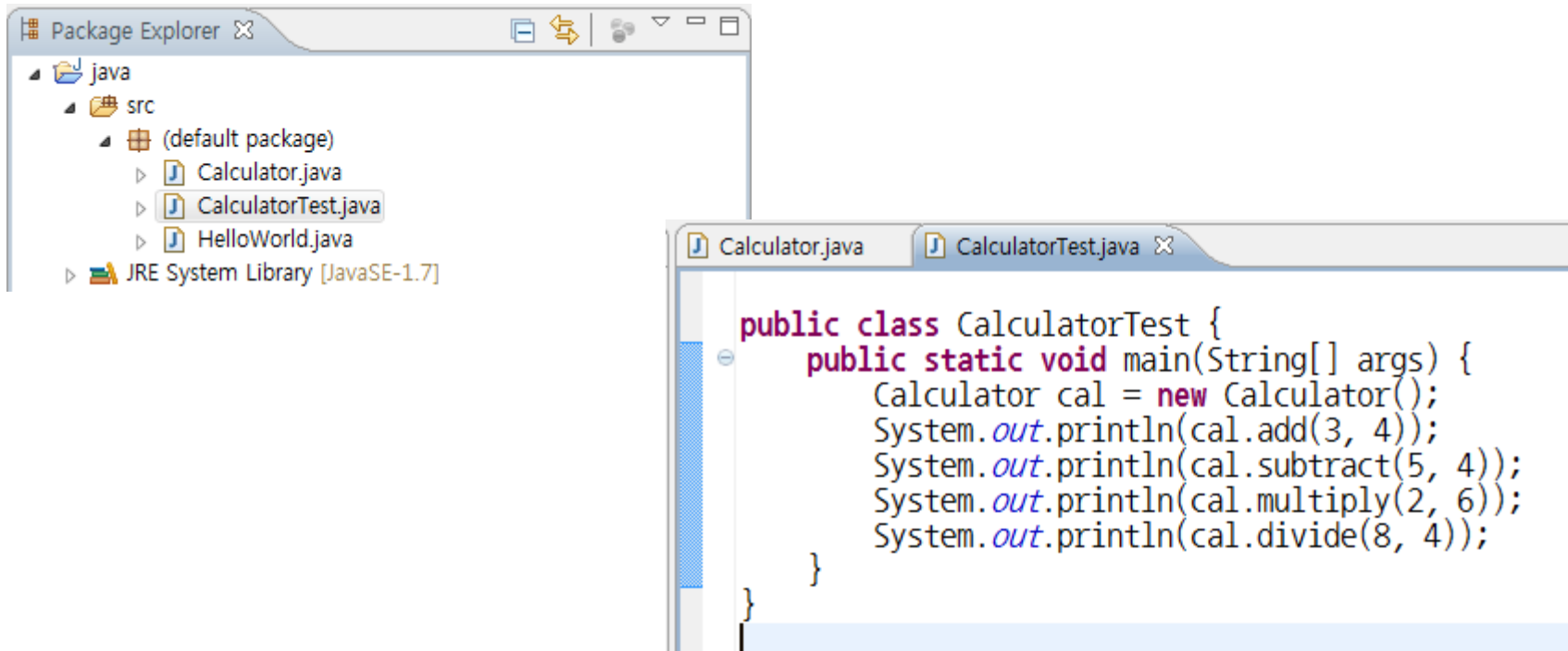
```
public static void main(String[] args) {  
    Calculator cal = new Calculator();  
    System.out.println(cal.add(3, 4));  
    System.out.println(cal.subtract(5, 4));  
    System.out.println(cal.multiply(2, 6));  
    System.out.println(cal.divide(8, 4));  
}
```

# main method의 문제점

Production code와 Test Code가 클래스 하나에 존재한다. 클래스 크기가 커짐. 복잡도 증가함.  
Test Code가 실 서비스에 같이 배포됨.

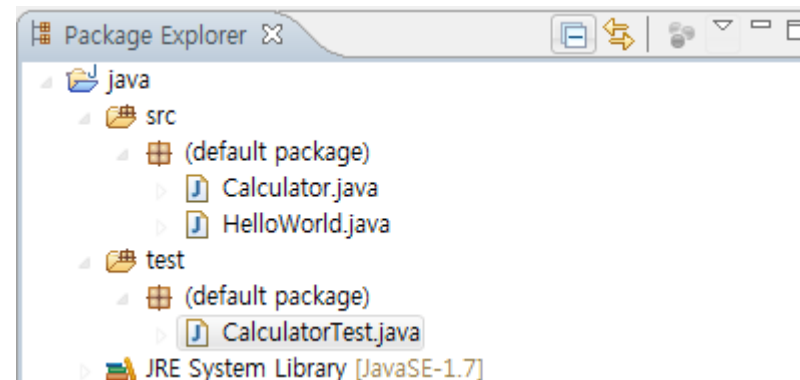
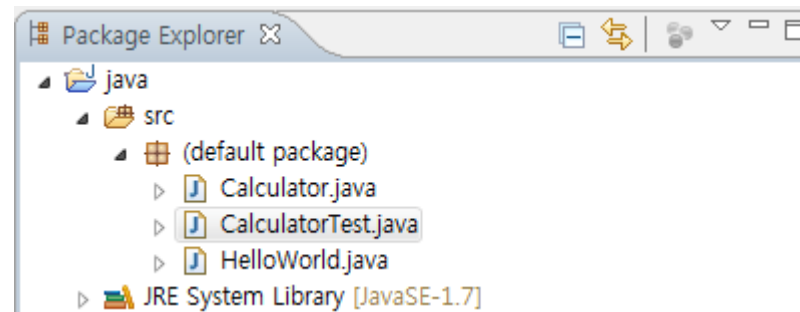
# 해결책

Production code와 Test Code가 클래스 하나에 존재한다.  
클래스 크기가 커짐. 복잡도 증가함.



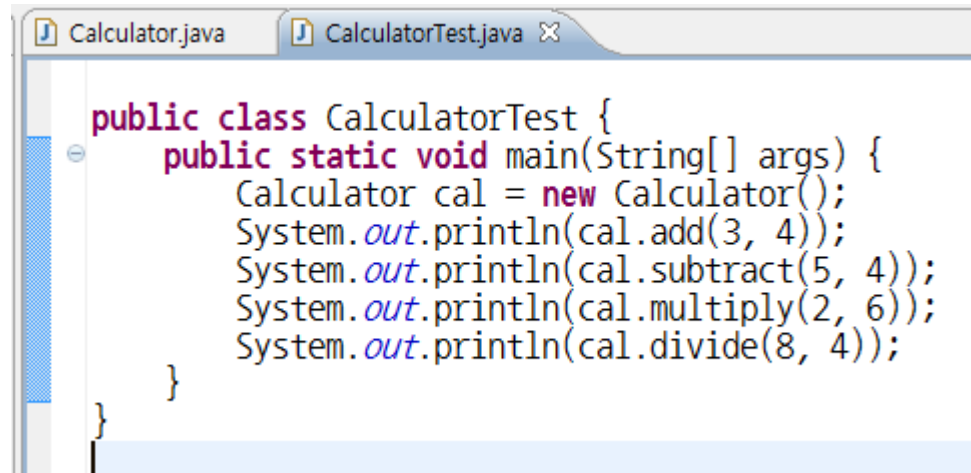
# 해결책

Test Code가 실 서비스에 같이 배포됨.





# 문제점 찾기



```
Calculator.java  CalculatorTest.java X
public class CalculatorTest {
    public static void main(String[] args) {
        Calculator cal = new Calculator();
        System.out.println(cal.add(3, 4));
        System.out.println(cal.subtract(5, 4));
        System.out.println(cal.multiply(2, 6));
        System.out.println(cal.divide(8, 4));
    }
}
```

main method 하나에서 여러 개의 기능을 테스트 함. 복잡도 증가  
method명을 통해 테스트 의도를 드러내기 힘들다.

# 해결책

```
CalculatorTest.java ✕  
  
public class CalculatorTest {  
    public static void main(String[] args) {  
        add();  
        subtract();  
        multiply();  
        divide();  
    }  
  
    private static void add() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.add(3, 4));  
    }  
  
    private static void subtract() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.subtract(5, 4));  
    }  
  
    private static void multiply() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.multiply(2, 6));  
    }  
  
    private static void divide() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.divide(8, 4));  
    }  
}
```

# 문제점 찾기

CalculatorTest.java

```
public class CalculatorTest {  
    public static void main(String[] args) {  
        add();  
        subtract();  
        multiply();  
        divide();  
    }  
  
    private static void add() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.add(3, 4));  
    }  
  
    private static void subtract() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.subtract(5, 4));  
    }  
  
    private static void multiply() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.multiply(2, 6));  
    }  
  
    private static void divide() {  
        Calculator cal = new Calculator();  
        System.out.println(cal.divide(8, 4));  
    }  
}
```

# main method의 문제점

테스트 복잡도가 증가하면 테스트 간에 영향을 미친다.

테스트 결과를 사람이 수동으로 확인

# 해결책

main() 메소드를 통해 해결할 수 없어 junit이라는 도구가 등장함.

# JUnit 3

## 상속 메소드 이름 규칙

- extends TestCase
- public void test\*
- setUp, tearDown
- 상위 클래스의 assert를 활용

메소드의 실행 순서를 명확히 이해해야 함.

# JUnit 4

## Annotation

- @Test
- @Before, @After
- @BeforeClass, @AfterClass : static 메소드 앞에만 가능
- 상위 클래스의 assert를 활용

초간단 계산기를 TDD로 다시 구현한다.



# 메소드 실행 순서

```
public class SlippTest {  
    @Before  
    public void setup() {  
    }  
  
    @Test  
    public void test1() throws Exception {  
    }  
  
    @Test  
    public void test2() throws Exception {  
    }  
  
    @After  
    public void teardown() {  
    }  
}
```

# Hamcrest matcher

`assertEquals(expected, actual)` 대신 `assertThat(actual, is(expected))`;

보다 영어 같은 표현, `expected`와 `actual`을 덜 헷갈리게

`static import` 활용이 점점 더 중요해진다.

## TDD 원칙 1

실패하는 단위 테스트를 작성할 때까지 실제 코드를 작성하지 않는다.

## TDD 원칙 2

컴파일은 실패하지 않으면서 실행이 실패하는 정도로만 단위 테스트를 작성한다.

## TDD 원칙 3

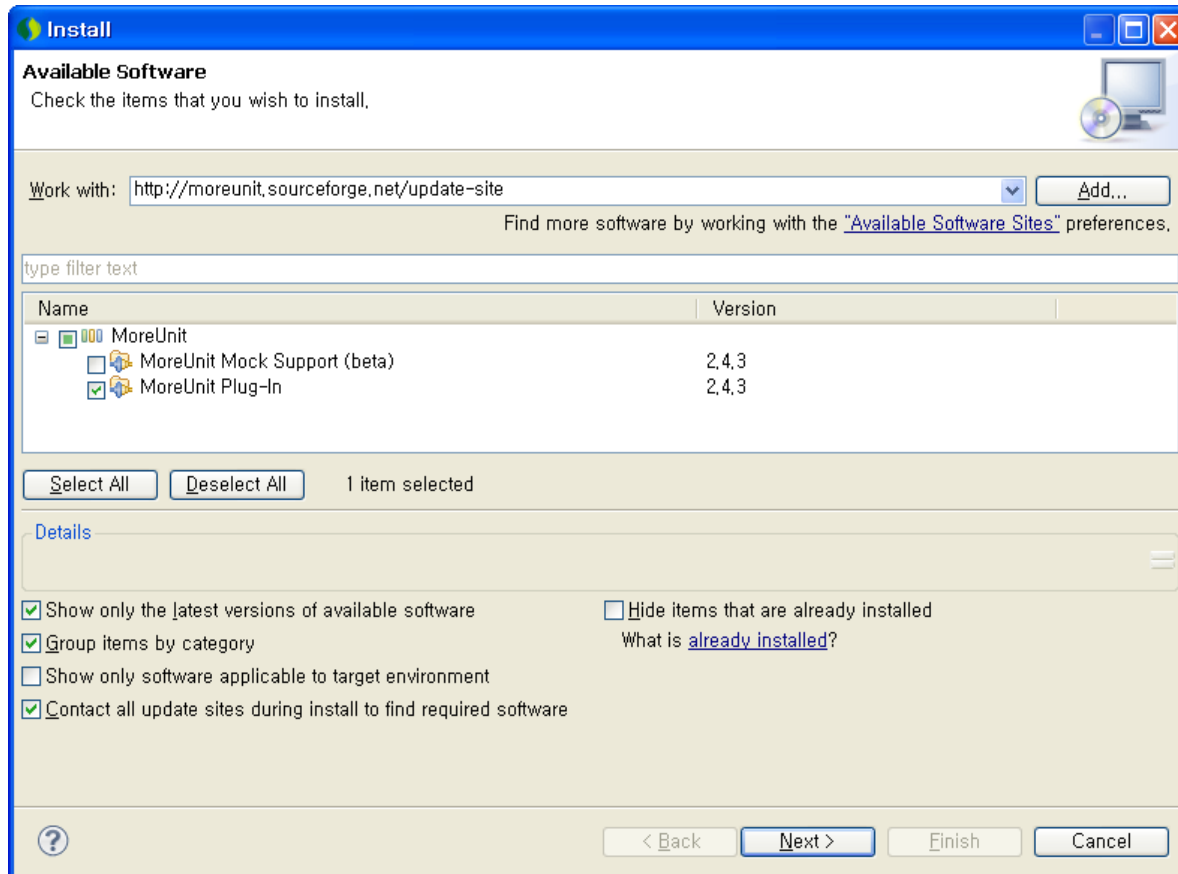
현재 실패하는 테스트를 통과할 정도로만 실제 코드를 작성한다.

# 효율적인 TDD를 위한 설정

-

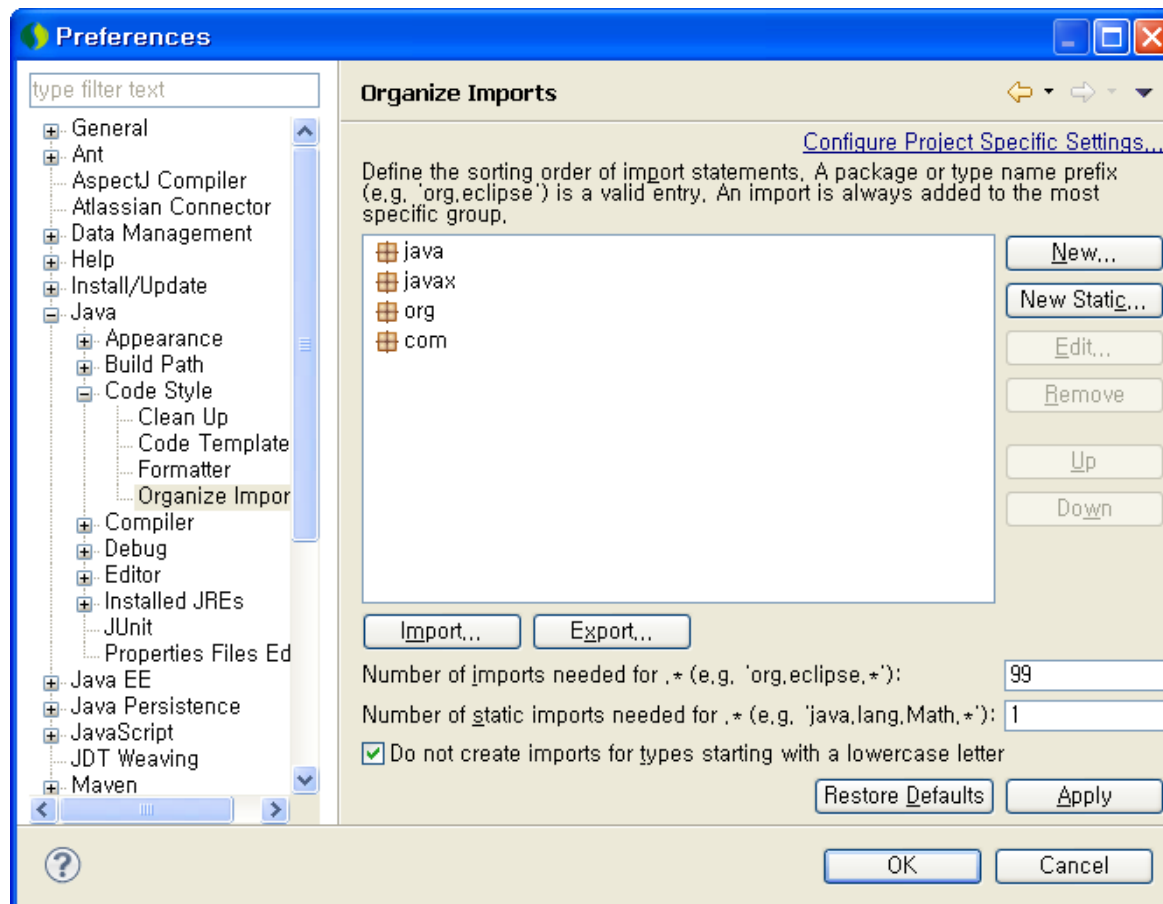
# Eclipse - More Unit 플러그인

- 테스트 코드와 실제 코드 사이를 단축키로 이동할 수 있다. (Ctrl + J)
- <http://moreunit.sourceforge.net>
- update url : <http://moreunit.sourceforge.net/update-site>
- Help >> Install New Software ... 에서 update url 활용해 설치



# Eclipse - static import 1

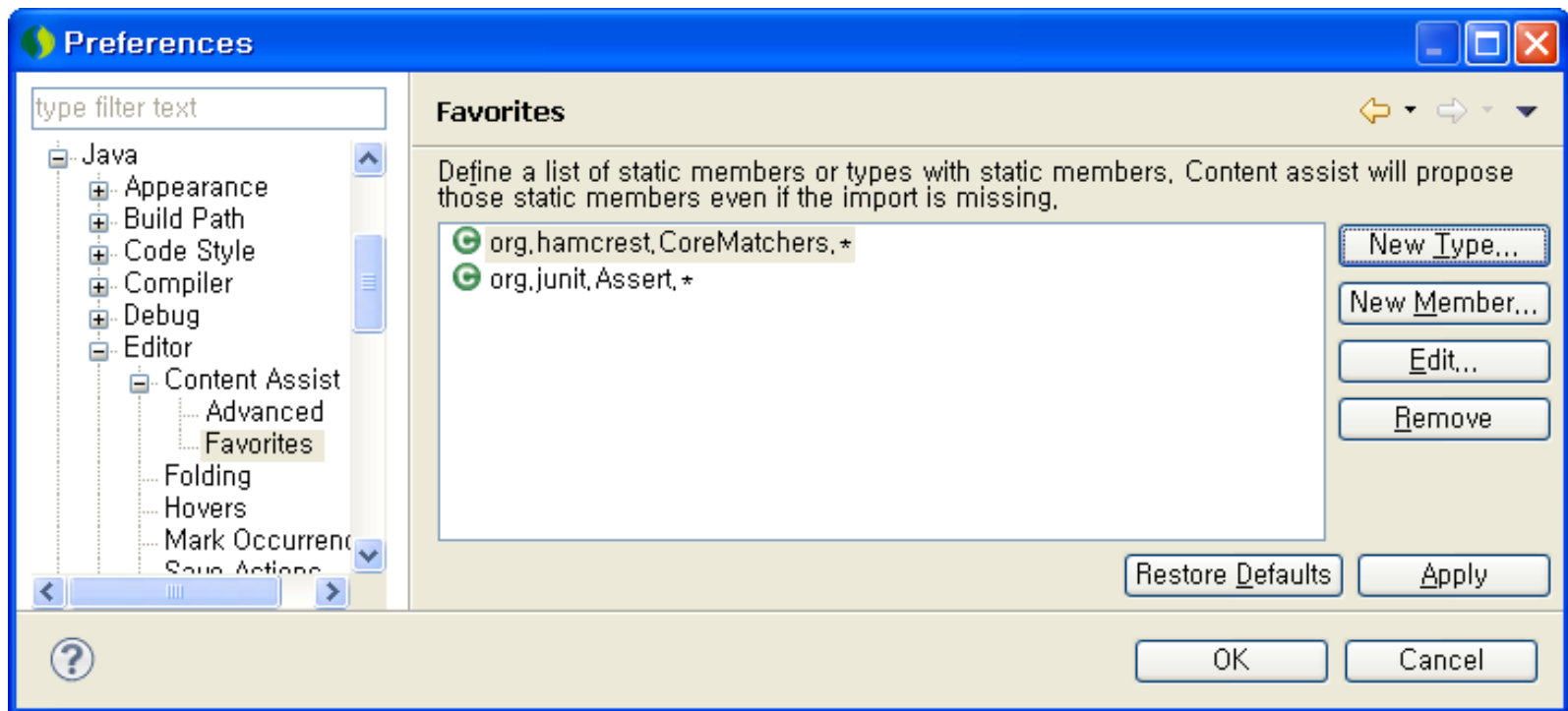
- Organize import해도 static import의 \*가 풀리지 않도록 설정
- Windows >> Preferences >> java >> Code Style >> Organize import의 Number of static imports...를 1로 설정





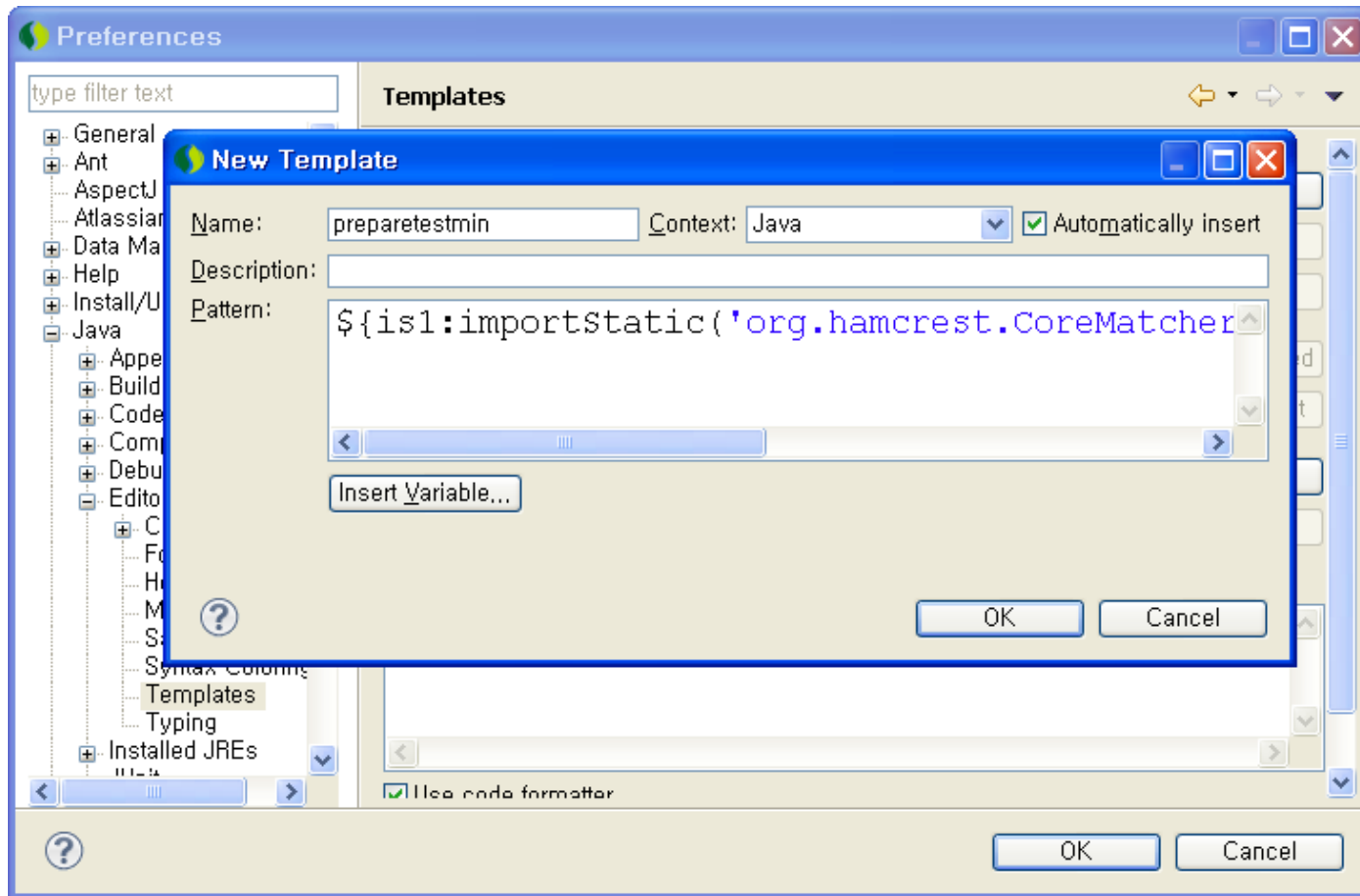
# Eclipse - static import 2

- 자주 쓰는 static import 등록
- Windows >> Preferences >> Java >> Editor >> Content Assist >> Favorites
  - org.junit.Assert.\*
  - org.hamcrest.CoreMatchers.\*



# Eclipse - static import 3

- 자주 쓰는 static import를 java template에 등록
- Windows >> Preferences >> Java >> Editor >> Templates
  - `${is1:importStatic('org.hamcrest.CoreMatchers.*')}${is2:importStatic('org.junit.Assert.*')}`



# Eclipse - TDD 자주 쓰는 단축키

- 코드 생성
  - Ctrl + 1 : Quick fix
  - Ctrl + Shift + O : import 절을 정리
- 실행
  - Alt + Shift + X, T : junit 으로 실행
- 이동
  - Ctrl + J : 테스트 코드와 실제 코드 이동(moreUnit 설치되어 있는 경우)
  - Ctrl + Q : 마지막으로 편집했던 곳으로 돌아감
  - Ctrl + T : 인터페이스에서 구현 클래스 찾을 때
  - Ctrl + Shift + 위, 아래 방향키 : method 단위로 커서 이동. method 하나만 실행할 때 사용하기 좋다.

# Eclipse - TDD 자주 쓰는 단축키

- 리팩토링
  - Alt + Shift + R : 이름 바꾸기
  - Alt + Shift + L : Local 변수 추출
  - Alt + Shift + V : 이동
  - Alt + Shift + M : 메소드 추출

# TDD - 문자열 계산기 구현

-

실습

빈 문자열을 입력할 경우 0을 반환해야 한다.  
(예 : “”)

힌트

```
if (text.isEmpty()) {  
  
}
```

## 실습

숫자 하나를 문자열로 입력할 경우 해당 숫자를 반환한다.  
(예 : “1”)

## 힌트

```
int number = Integer.parseInt(text);
```

## 실습

숫자 두개를 쉼표(,) 구분자로 입력할 경우 두 숫자의 합을 반환한다.(예 : “1,2”)

## 힌트

```
String[] numbers = text.split(",");
```



세 개 이상의 숫자를 쉼표(,) 구분자로 입력할 경우 모든 숫자의 합을 반환한다.(예 : “1,2,3”)

## 실습

구분자를 쉼표(,) 이외에 New Line을 사용할 수 있다.  
(예 : “1,2\n3”)

## 힌트

```
String[] tokens= text.split(",|\n");
```

## 실습

“//”를 사용해 커스텀 구분자를 지정할 수 있다.  
(예 : “//;\\n1;2;3”)

## 힌트

```
Matcher m = Pattern.compile("//(.)\\n(.*)").matcher(text);  
m.find();  
String customDelimiter = m.group(1);  
String[] tokens= m.group(2).split(customDelimiter);
```

음수를 전달할 경우 RuntimeException 예외가 발생해야 한다. (예 : “-1,2,3”)

## 실습 - 리팩토링

- indent depth를 2단계에서 1단계로 줄여라.
- else를 사용하지 마라.
- method가 한 가지 일만 하도록 최대한 작게 만들어라.

# 과제

-



## 목표

볼링점수를 계산하는 프로그램을 작성

## 볼링점수계산

Strike인 경우는 다음 두 번 투구 수의 점수를 합한다. 따라서 이후 두 번 더 투구할 때까지 strike한 프레임의 점수는 계산되지 않는다.  
Spare인 경우는 다음 한번 투구 수의 점수를 합한다. 따라서 이후 한 번 더 투구할 때까지 spare한 프레임의 점수는 계산되지 않는다.  
마지막 프레임의 경우는 위의 두 가지 조건을 만족하기 위해서 Strike이면 2번, Spare면 한번의 투구가 가능하다.

## 요구사항

볼링게임(BowlingGame)클래스의 인스턴스를 만들면 새 게임이 시작한 것으로 간주한다.(명시적인 start는 필요없음)  
현재 몇번째 프레임의 몇번째 투구(첫번~세번째)를 할 차례인 조회해 볼 수 있다. 게임이 끝났으면 GameOverException을 던진다.  
(Frame번호 + 그 프레임의 시도횟수)  
현재까지 진행된 프레임결과와 각 프레임 점수를 보여준다. 확정되지 않은 점수는 표시하지 않아도 된다.

한번 투구를 하는 메소드(roll)를 만들고 쓰러뜨린 핀의 수를 파라미터로 넘긴다. 게임이 끝났으면 GameOverException을 던진다.

기호)

Stike : X

Spare : /

Gutter: -

그외 : 0~9

볼링 한 게임에 대한 점수판을 구현해야 한다.

볼링 게임을 진행한다고 가정하고 볼링 핀을 입력하면 다음과 같은 결과 화면을 볼 수 있어야 한다.

아래 화면은 게임이 끝났을 때의 결과 화면이고, 게임이 진행 중에도 아래와 같은 점수 판을 볼 수 있어야 한다.

1	2	3	4	5	6	7	8	9	10
9 / 8 -	X	X	8 F	X	8 1	9 / 8 1	X	9 /	
18	26	54	72	80	99	108	126	135	155



# 참고 문서 - 객체 지향 생활체조

- <http://elaia.tistory.com/3> 문서 참고
- 더 자세한 내용은 “소트웍스 앤솔러지, 마틴파울러 외 지음, 위키북스” 책의 83~96 페이지 참고. 박재성 교수 방에 책 있으니 복사해서 참고

## 추가 요구사항

게임을 시작할 때 플레이어 수를 지정할 수 있다.

각 플레이어 별로 게임 점수를 확인할 수 있다.

	2	1	2	3	4	5	6	7	8	9	10	HDP
1		9 / - /	X	X	-5	6 -	-8	8 -	X	36		
1	10	30	50	65	70	76	84	92	111	120		120
2		8 /	X	3 /	7 /	8 /	X	72	X	3 -	6 -	
2	20	40	57	75	95	114	123	136	139	145		145
3		61	51	9 -	36	33	X	7 -	-9	X	16	
3	7	13	22	31	37	54	61	70	87	94		94
										359	359	TH

## 과제 제출 방법

Github에 저장소를 만들고 소스 코드를 Commit한다.

7월 26일까지 완료한다.

Github URL을 [javajigi@nhn.com](mailto:javajigi@nhn.com)으로 공유한다.