

Korea
Software
Technology
Association



과정명: JUnit 기반 Java Test Programming

훈련기간: 2008.06.09 ~ 2008.06.13

김영진(yjkim@nextree.co.kr)

박윤미(lympark@nextree.co.kr)

□ 교육 목표

- TDD(Test Driven Development)에 대한 이해 정립
- 단위 테스트 수행방법과 이를 지원하는 **jUnit** 에 대한 이해
- 다양한 요소에 대한 단위 테스트 실행 방법 적용

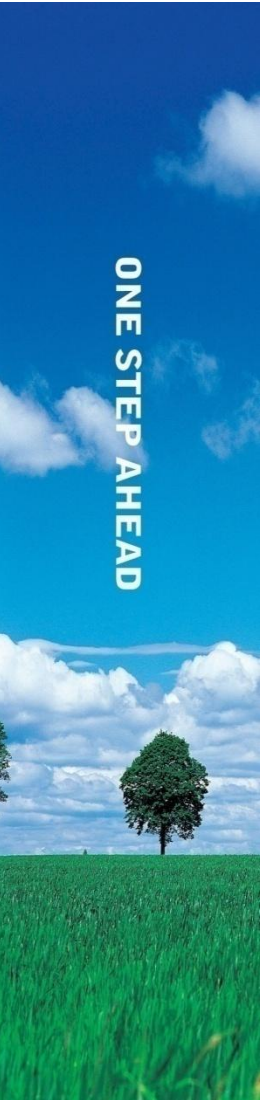
□ 교육 특징

- 강의와 실습을 병행하여 수강생의 명확한 이해를 도모함
- 숙달 및 깊은 이해가 필요한 부분은 실습 수행
- 현장 경험이 풍부한 강사진으로 구성
- 충분한 토의를 통한 지식 및 경험 공유

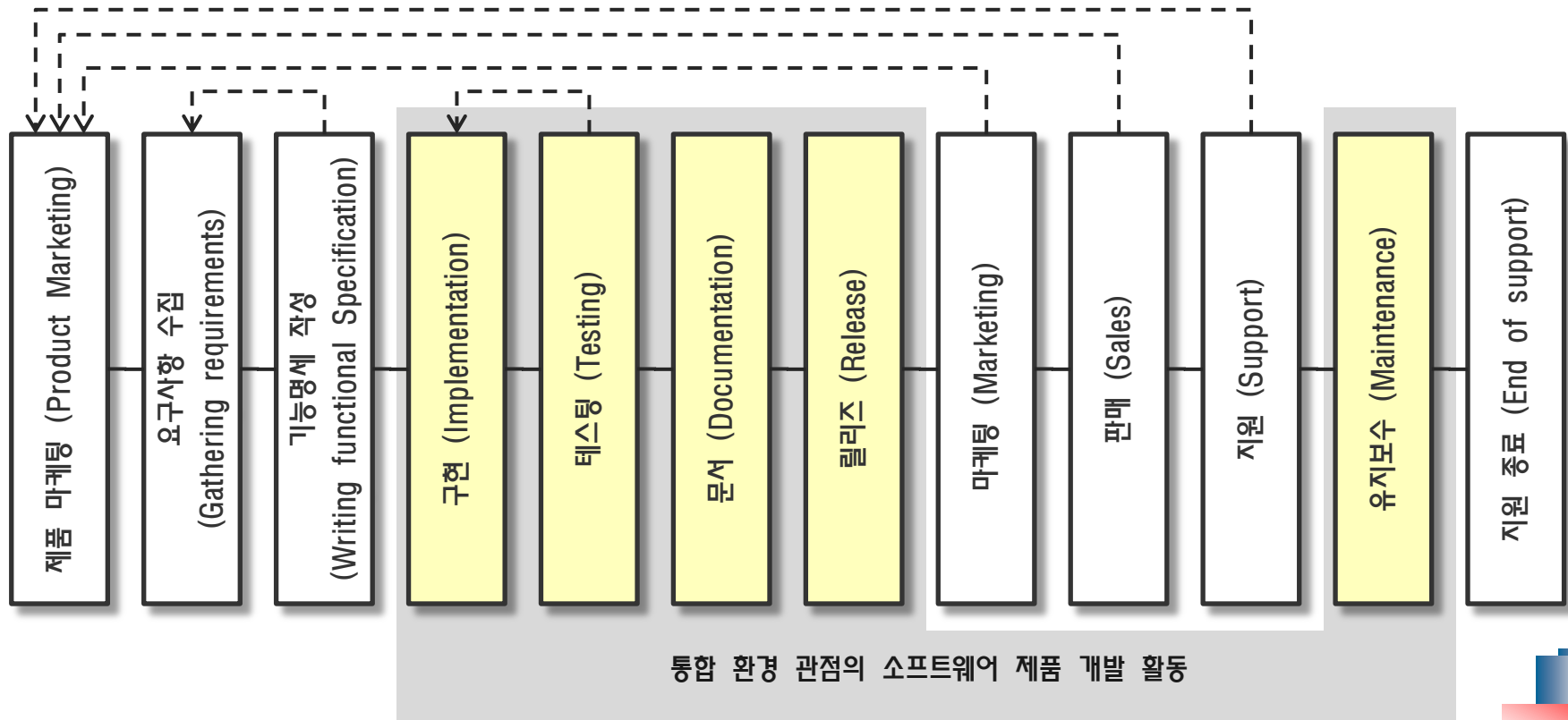
1일차 (월요일)	화	수	목	금
6월 9일	6월 10일	6월 11일	6월 12일	6월 13일
jUnit 개요	소프트웨어 테스트 와 jUnit 자동화	jUnit 테스트 전략	jUnit 테스트 심화 I	jUnit 테스트 심화 II

- 소프트웨어 개발
- 단위 테스트 개념 이해
- **JUnit**을 이용한 단위 테스트
- **TDD(Test Driven Development)**

ONE STEP AHEAD



- ❑ 소프트웨어 제품 생성은 다양한 활동(**Activities**)이 존재
- ❑ 활동은 프로젝트 개발과 유지보수 상황에서 여러 차례로 반복 되어 짐
- ❑ 일부 프로젝트에서는 연결된 모든 활동을 가지는 것은 아니지만 이러한 활동은 좋은 개발 환경을 지원하는 것 중 하나임



□ 소프트웨어 제품 개발 활동 단계별 내용

● 제품 마케팅 (Product marketing)

소프트웨어 제품에 대한 아이디어 정립을 포함하는 것으로 잠재적 고객이 실질적으로 원하는 성향을 파악하여 구성하도록 기술적 지식을 조화시킴

● 요구사항 수집 (Gathering requirements)

소프트웨어 제품이 명백히 수행할 것이 무엇인지 프로젝트 내의 모든 이해관계자가 명확히 의사소통 할 수 있도록 정보수집 하는 과정으로 문서, 프로토타입 그리고 제품이 사용될 수 있는 방법의 예제 등으로 기술

● 기능 명세 작성 (Writing functional specifications)

소프트웨어 제품이 요구사항을 수행할 수 있도록 만들기 위해 기능 명세를 기술

● 구현 (Implementation)

실제로 소프트웨어 제품을 생성하는 단계로서 의외로 개발에 필요한 시간이 전체 시간에 비해 적은 부분을 차지

● 테스트 (Testing)

요구사항과 구현된 것을 구체적으로 비교하는 곳으로 예상외로 전체 개발 시간 중에 많은 부분을 차지하지만 일정이 바쁠 때 일반적으로 가장 먼저 삭제됨

● 문서 (Documentation)

소프트웨어 제품을 사용하는 방법에 대한 기술로서 내부 문서는 제품의 유지보수를 위해 참여하게 될 개발자를 위해 기술함

● 릴리즈 (Release)

고객에게 소프트웨어 제품을 이관하고 개발된 제품이 외부 환경에서 적절히 작동되는지 확인하는 과정

● 마케팅 (Marketing)

소프트웨어 제품에 대한 시장을 생성하고 조절하는 것으로 시장 상황에 맞도록 제품을 변경하는 것도 포함

● 판매 (Sales)

소프트웨어 제품을 사용하도록 설득하고 이에 대한 요금을 지불하도록 기대하는 단계

● 지원 (Support)

고객의 질문에 대한 해결책이 문서에 없을 때 고객이 소프트웨어 제품을 사용할 수 있도록 돕는 과정으로 제품 사용에 필요한 라이선스에 대한 Contact point를 제공

● 유지보수 (Maintenance)

주로 상업적인 소프트웨어 제품에서 제품이 변경되는 경우와 같이 적절히 기능을 유지하도록 필요한 작업을 진행하여 고객의 수익을 보장할 수 있도록 도와주는 단계

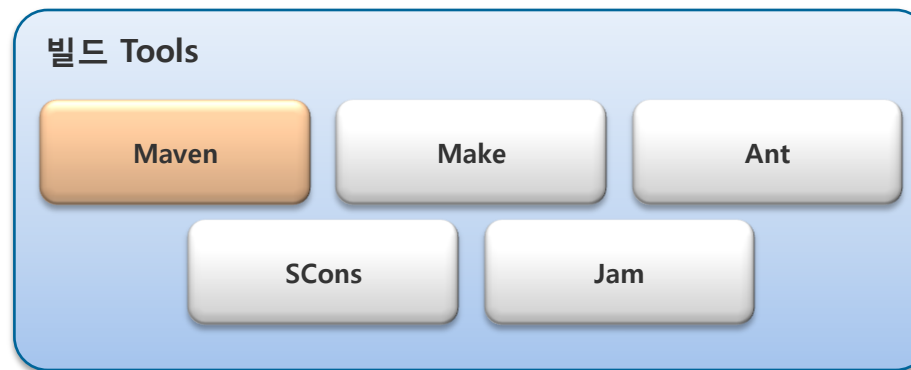
● 지원 종료 (End of support)

최후에는 소프트웨어 제품을 사거나 사용하는 사람이 없게 되는 단계로서 고객은 제품 지원이 종료될 것이라는 통보를 하게 되고 개발환경과 소스파일을 조심스럽게 보관하도록 필요한 단계

□ 저장된 소스파일(소스코드)로부터 소프트웨어 제품이 빌드

□ 빌드 툴

- 소스파일을 사용하여 제품을 생성
- 컴파일러와 같은 툴이 수행될 때 빌드 규칙이 적용됨
- 일반적으로, 빌드 파일과 같은 구성 파일 내에 빌드 규칙을 명시
- 소스파일 셋과 라이브러리와 같은 파일로부터 빌드될 수 있는 것을 인지
- 개발자가 소스파일을 변경했을 때 의존성이 있는 다른 부분도 재 빌드
- 동일한 소프트웨어 제품을 다른 플랫폼에서 빌드 되도록 해야 함



- 테스트 수행은 소프트웨어 제품이 어떻게 수행할 것인지 고객으로부터 피드백 받음
 - 개발자가 작성한 단위 테스트는 애플리케이션의 작은 부분을 체크
 - 시스템 테스트는 애플리케이션 전체에 대한 사용을 체크
- 테스트가 자동화되지 않으면 제품테스트 결과는 지연 또는 포기될 수 있음
- 테스트 환경은 제품 개발과 유지보수 시 매우 중요
 - 테스트를 쉽게 추가하고 수행할 수 있어야 함
 - 테스트 결과는 명확하게 이해할 수 있도록 해야 함



XP 방법론

"기능을 구현하기 전에 테스트를 먼저 작성하라"

UNIT TEST

□ 버그 트래킹 툴

- 일반적으로 제품의 테스트 수행에 대한 정보와 같은 내용을 사용
- 버그에 대한 정보를 데이터베이스에 저장하고 이에 대한 GUIs와 기록된 버그 정보의 변경 또는 입력을 위한 CLIs(Command-Line Interfaces)를 제공
- 여러 버그 정보에 대한 보고서를 생성
- 여러 버그들에 대한 작업을 진행할 때 팀의 작업흐름을 돕기 위해 여러 상태를 정의
 - 상태 예: Open → Accepted → Fixed → Closed
- 버그 트래킹 시스템
 - Spreadsheets
 - **Bugzilla**
 - GNATS
 - FogBugz
 - JIRA
 - TestTrack

버그에 대한 이슈

- 일부 다른 카테고리나 조건에 대한 변경 요구
- 제품의 심한 결함이나 이로 인한 사고
→ 버그는 이들과 연관관계를 가지고 있으면 가능하면 이를 피해야 한다.

ERROR!

- ❑ 테스트에 대한 잘못된 인식
- ❑ 테스트와 디버깅
- ❑ 프로젝트 테스트 시간
- ❑ 테스트 종류
- ❑ 객체지향 테스트(**Object-Oriented Testing**)의 전체 주기
- ❑ 단위 테스트란?
- ❑ 단위 테스트 경계

테스트에 대한 잘못된 인식

□ 개발자 테스트

- 테스트 케이스 작성은 불편하며 귀찮은 것이다.
- 진짜 코드를 만들 시간도 부족하다.
- 잘 되는 성공 케이스의 테스트가 성공하면 안심한다.
- 내가 만든 코드는 잘 돌아간다. (자만에 의한 테스트 생략)

□ 프로젝트에서의 테스트

- 프로젝트가 끝날 때에만 테스트 하면 된다.
- 테스트는 꼭 필요하지만, 일정에 압박이 오면 축소하거나 생략할 수도 있다.

큰일이군...
내일이 오픈인데, 에러가 많으니...
박대리!
밤샘 작업해서라도 오픈해야 해!!

테스트 짤 시간이 어딴어?
코딩 할 시간도 부족하단 말야!
...
난 테스트 따윈 필요없어!



□ 테스트(Test)

- 오류를 발견하기 위한 방법
- 요구사항 명세서에 했던 가정에 대한 타당성을 검토
- 요구된 소프트웨어 제품 기능의 타당성 검토

□ 디버깅(Debugging)

- 이미 발견된 오류의 원인을 진단하고 수정하는 방법

□ 테스트 vs 디버깅

- 대다수의 개발자들이 전체 테스트 시간의 많은 부분을 디버깅을 하는데 소요
- 선(先)테스트를 늘리면, 디버깅을 통한 오류 수정시간 단축 가능
- 에러가 발생 가능한 코드의 작은 부분들에 대한 테스트가 이미 존재한다면,
복잡한 프로그램의 오류도 쉽게 발견 가능

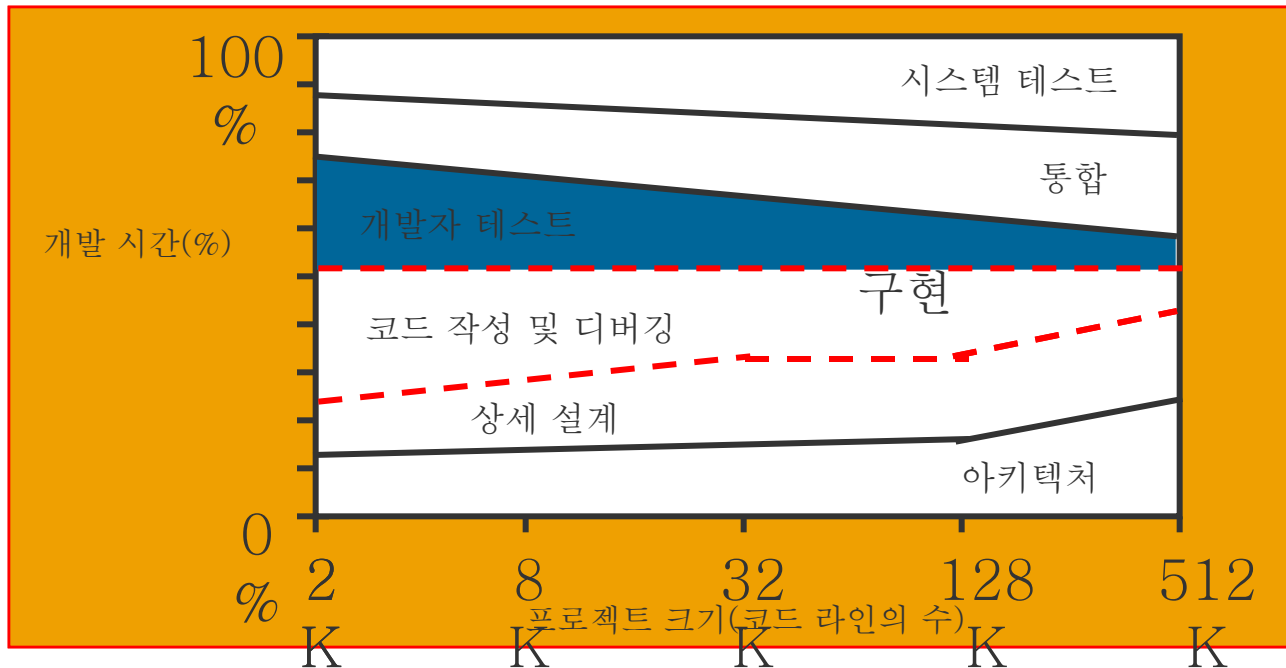
Q. 일반적인 프로젝트에서 개발자가 테스트하는데 얼마나 많은 시간을 보내야 하는가?

A. 일반적으로 인용되는 수치는 프로젝트 전체시간의 50%이다.

하지만, 이는 테스트와 디버깅을 합친 시간을 의미하며, 테스트가 더 적게 걸린다.

이는 반드시 소요되어야 하는 시간이라기 보다는 일반적으로 소요되는 시간을 의미하며, 개발자 단위 테스트 및 기능(시스템) 테스트를 모두 포함하는 시간이다.

(개발자 테스트는 8~25% 정도를 차지한다.)



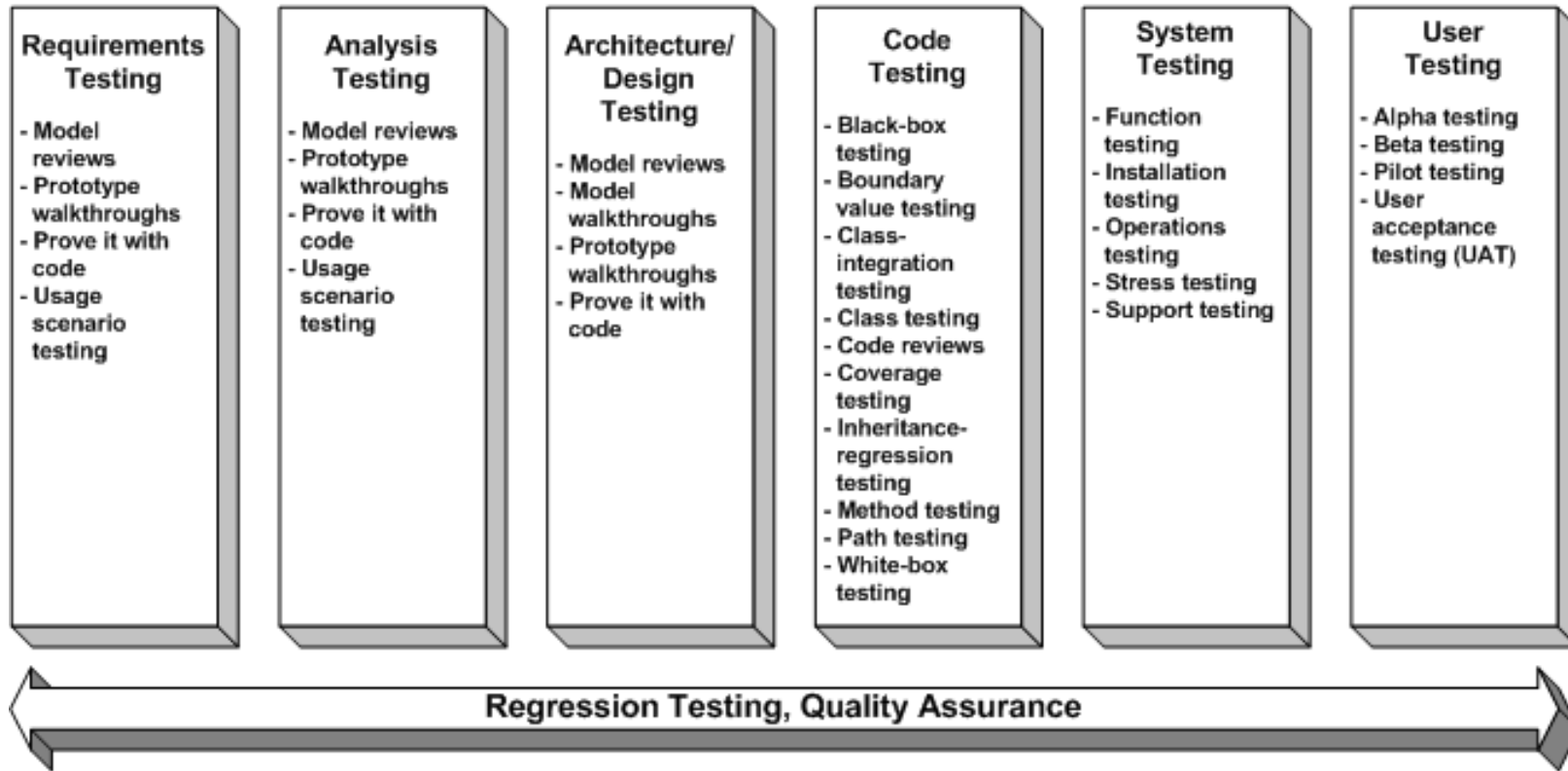
□ 코드 테스트 (Code Testing)

- 단위 테스트 (Unit Test)
 - 기능 테스트에 속함
 - 개발자에 의한 개발자를 위한 테스트
 - 필수적이긴 하지만, 검증 툴로서는 불충분함

□ 시스템 테스트 (System Testing)

- 기능(functional) 테스트
- 성능(performance) 테스트
- 스트레스(stress) 테스트
- 확장성(scalability) 테스트
- 가용성(availability) 테스트
- 보안(security) 테스트
- 인수(acceptance) 테스트

객체지향 테스트의 전체 주기 (1/3)



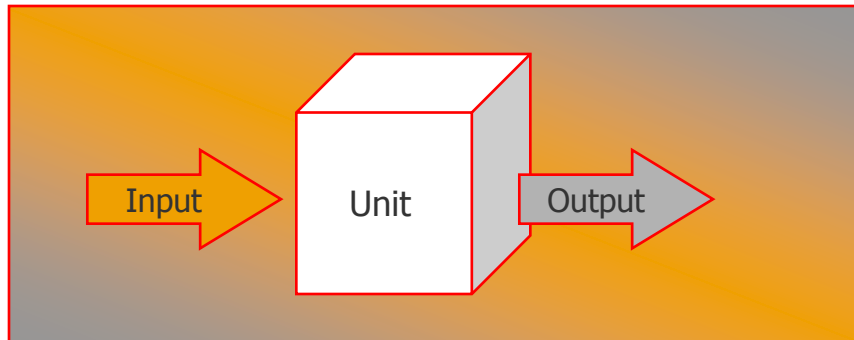
객체지향 테스트의 전체 주기 (2/3)

FLOOT Technique	Description
Black-box testing	Testing that verifies the item being tested when given the appropriate input provides the expected results.
Boundary-value testing	Testing of unusual or extreme situations that an item should be able to handle.
Class testing	The act of ensuring that a class and its instances (objects) perform as defined.
Class-integration testing	The act of ensuring that the classes, and their instances, form some software perform as defined.
Code review	A form of technical review in which the deliverable being reviewed is source code.
Component testing	The act of validating that a component works as defined.
Coverage testing	The act of ensuring that every line of code is exercised at least once.
Design review	A technical review in which a design model is inspected.
Inheritance-regression testing	The act of running the test cases of the super classes, both direct and indirect, on a given subclass.
Integration testing	Testing to verify several portions of software work together.
Method testing	Testing to verify a method (member function) performs as defined.
Model review	An inspection, ranging anywhere from a formal technical review to an informal walkthrough, by others who were not directly involved with the development of the model.
Path testing	The act of ensuring that all logic paths within your code are exercised at least once.
Prototype review	A process by which your users work through a collection of use cases, using a prototype as if it was the real system. The main goal is to test whether the design of the prototype meets their needs.

FLOOT Technique	Description
Prove it with code	The best way to determine if a model actually reflects what is needed, or what should be built, is to actually build software based on that model that show that the model works.
Regression testing	The acts of ensuring that previously tested behaviors still work as expected after changes have been made to an application.
Stress testing	The act of ensuring that the system performs as expected under high volumes of transactions, users, load, and so on.
Technical review	A quality assurance technique in which the design of your application is examined critically by a group of your peers. A review typically focuses on accuracy, quality, usability, and completeness. This process is often referred to as a walkthrough, an inspection, or a peer review.
Usage scenario testing	A testing technique in which one or more person(s) validate a model by acting through the logic of usage scenarios.
User interface testing	The testing of the user interface (UI) to ensure that it follows accepted UI standards and meets the requirements defined for it. Often referred to as graphical user interface (GUI) testing.
White-box testing	Testing to verify that specific lines of code work as defined. Also referred to as clear-box testing.

□ 단위(Unit)

- 테스트 가능한 최소의 소프트웨어 요소
- 하나의 단위는 대개 한 개발자가 책임을 짐
- 쉽게 예상 가능한 결과가 반환되는 수준의 코드 레벨을 가짐
 - 단위 코드의 결과가 예측 불가능하다면 리팩토링을 검토해 볼 것



(Unit Testing = White-Box Test)

□ 단위 테스트(Unit Test)

- 단위 코드 조각이 개발자가 의도한 대로 동작하는지 증명하기 위한 방법
- 단위 코드에서 에러 발생 소지가 있는 모든 부분을 테스트 해야 함
- 일반적으로 클래스의 **public method**를 테스트 수행

Q. 어디까지가 단위테스트인가?

A. 이 질문에 대한 논의는 지금까지 계속되어 왔으며, 앞으로도 프로젝트를 진행할 때마다 꾸준히 논의 될 것으로 예상된다.

앞에서 우리는 ‘쉽게 결과(output)가 예측 가능한 에러가 발생할 가능성이 있는 모든 public 메서드’ 정도로 단위 테스트의 범위를 정리해 보았다. 하지만 이런 정의는 조금은 애매 모호하다.

현실 세계의 프로젝트를 진행할 때에는 단위 테스트의 범위에 대한 ‘팀’ 내의 공유가 필요하다.

단위 테스트에 대한 범위를 정하고 모든 개발자가 단위 테스트를 수행하도록 유도한다면 궁극적으로는 애플리케이션의 높은 완성도를 기대할 수 있게 될 것이다.

(한 예로 ‘S사이트’에서는 모든 인터페이스에 대한 단위테스트 작성을 의무화 하기도 하였다.

물론, 개발자들은 인터페이스 외에도 수많은 단위 테스트를 가지고 있었다.)

위와는 다른 방향으로 단위테스트의 범주를 생각해본다면, 레거시 시스템(Legacy system)과의 연동이나 불안정한 데이터베이스와의 연결 또는 계속 변경중인 다른 팀에서 제공하는 인터페이스의 사용 등 단위 테스트가 어려운 다양한 환경에 놓일 경우의 단위 테스트를 수행해야 할 때가 있다.

이 때 환경에 대한 테스트를 하고자 하는 것이 아니라면 이러한 환경적인 에러 부분들은 개발자의 단위 테스트 범위에서는 제외하더라도, 단위에 대한 테스트를 위하여 모의 객체(Mock Object)를 활용함으로써 단위 테스트를 수행하는 방법이 있다.

단위 테스트 전략

- 단위 테스트의 목적
- 단위 테스트의 대상
- 좋은 테스트의 특징

단위 테스트의 목적

□ 왜(Why) 단위 테스트를 해야 하는가?

- 디버깅에 낭비하던 시간을 극적으로 줄일 수 있음
- 상대적으로 더 쉽고 비용이 적게 듦
- 좋은 설계를 이끌어 냄
- 신뢰 할 수 있는 시스템
- 단위 테스트를 코드가 의도한 바를 쉽게 파악할 수 있는 문서 또는 예제로 활용가능

단위 테스트의 대상 (1/2)

□ 테스트 영역

- 결과의 유효성 검사
- 단위(unit) 안에서 발생 가능한 모든 경우(case)에 대하여 테스트
- 버그가 자주 발생하는 경계조건(boundary)에 대한 테스트
- 역(Inverse) 관계를 확인하는 테스트
- 같은 결과를 내는 다른 알고리즘을 통한 교차 테스트(Cross-check)
- 시스템 부하, 네트워크 끊김 등 시스템에서 발생 가능한 에러에 대한 테스트
- 버그 수정으로 인해 발생 가능성이 있는 코드의 테스트



“나 지금 떨고있니?”

불안한 모든 코드를 테스트해라!

단위 테스트의 대상 (2/2)

□ 버그가 자주 발생하는 경계 조건

- 형식일치(conformance)
- 순서(Ordering)
- 범위(Range)
 - 1,000살(나이), 2007년 2월 29일(날짜)
- 참조(Reference)
 - 선언은 되었으나 정의되지 않은 전역변수의 사용
 - JNDI에 등록되지 않은 EJB 인터페이스의 호출
- 존재성(Existence)
 - null 값을 갖는 List의 size() 메서드 호출 (null, 0, “)
 - 종료된 세션의 접근
- 개체 수(Cardinality)
 - 0-1-n 규칙
- 시간(Time)
 - 상대 시간(시간적 순서)
 - 절대 시간(경과한 총 계산 시간)
 - 동시성(concurrency) 문제

□ 자동적(Automatic)

- 단위 테스트의 실행이 쉬워야 함
- 테스트는 스스로 검증할 수 있어야 함 (단정문의 사용)
- 테스트의 수동 단계를 자동화 해야 함 (Network,DB연결-모의 객체의 활용)

□ 철저함(Thorough)

- 문제가 될 수 있는 모든 것을 테스트

□ 반복 가능(Repeatable)

- 모든 테스트가 순서,시간에 관계없이 반복 수행 가능해야 함 (환경으로부터의 독립)

□ 독립적(Independent)

- 동시에 수행되는 테스트가 독립적인 상태를 유지해야 함
- 한 테스트가 다른 테스트에 의존하지 않아야 함

□ 전문적(Professional)

- 테스트 코드도 좋은 설계를 위한 전문적인 표준을 지켜 구현해야 함(캡슐화 유지, DRY(Don't Repeat Yourself) 원칙 지키기, 결합도 낮추기 등)

JUnit을 이용한 단위 테스트

- JUnit 개요
- JUnit 설치 및 실행
- JUnit을 이용한 단위 테스트
- JUnit 프레임워크의 이해
- 단정(Assertion) 메서드
- JUnit과 예외처리
- JUnit 4 이후 변화

□ 단위 테스트 프레임워크(JUnit)

- 1997년, Erich Gamma와 Kent Beck이 개발
- Kent Beck이 Smalltalk를 위해 개발한 프레임워크, SUnit의 후속
- 공개 소스 소프트웨어(상업적인 용도로 사용할 수 있음)
- IBM의 공개 라이선스 1.0 버전에서 배포되었고, SourceForge가 후원
- Java 개발에서 단위 테스트의 **de facto** 표준 프레임워크로 빠르게 자리잡았다

□ 프레임워크

- 프레임워크는 반-완성된(semi-complete) 어플리케이션
- 어플리케이션간에 공유할 수 있는 재사용 가능한 공통 구조를 제공
- 단순한 유틸리티를 제공하기 보다는 응집된 구조를 제공.

□ xUnit 프레임워크

- 모든 언어를 위한 표준 프레임워크로 자리잡고 있다.
- 예) ASP, C++, C#, Eiffel, Delphi, Perl, PHP, Python, REBOL, Smalltalk, Visual Basic 등.



Kent Beck
XP(Extreme Programming) 창시자
TDD 선구자



Erich Gamma
Design Pattern, (GoF)

Junit 설치 및 실행

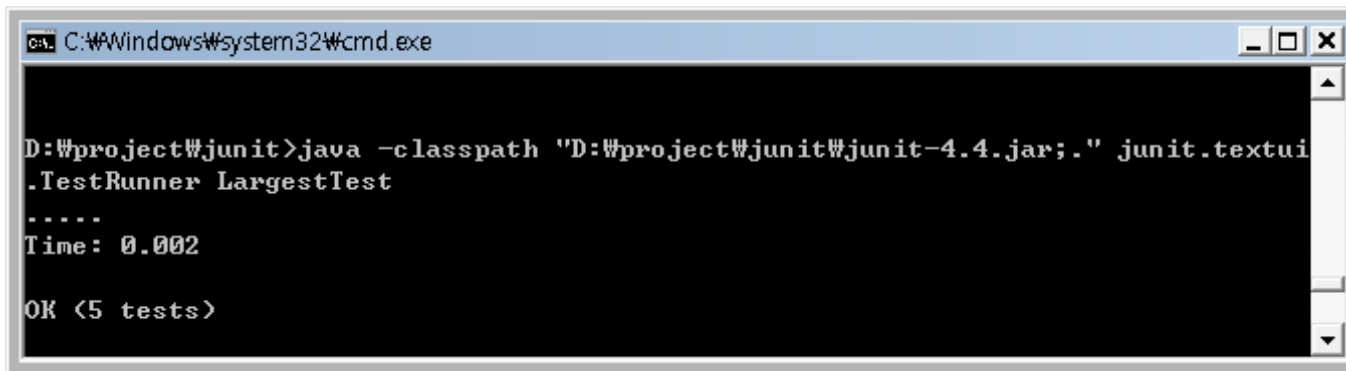
□ 설치

- Junit.org에서 최신 버전을 다운로드 (2008.06 현재 : released. JUnit4.4)
- junit.jar를 classpath내에 위치하여 사용
- 최근에는 Eclipse와 같은 IDE에 기본적으로 포함되어 배포되고 있음

□ 실행 – **Test Runner**

- Textual test runner
- Graphical test runner : Swing-based, AWT-based (최신 버전에는 미포함)
- IDE에서 제공하는 GUI를 이용

위와 같은 방식을 통해 JUnit 테스트를 실행시킬 수 있음



```

C:\Windows\system32\cmd.exe

D:\project\junit>java -classpath "D:\project\junit\junit-4.4.jar;." junit.textui
.TestRunner LargestTest
.....
Time: 0.002

OK (5 tests)
  
```


JUnit 프레임워크의 이해 (1/7)

□ JUnit 프레임워크의 핵심 class

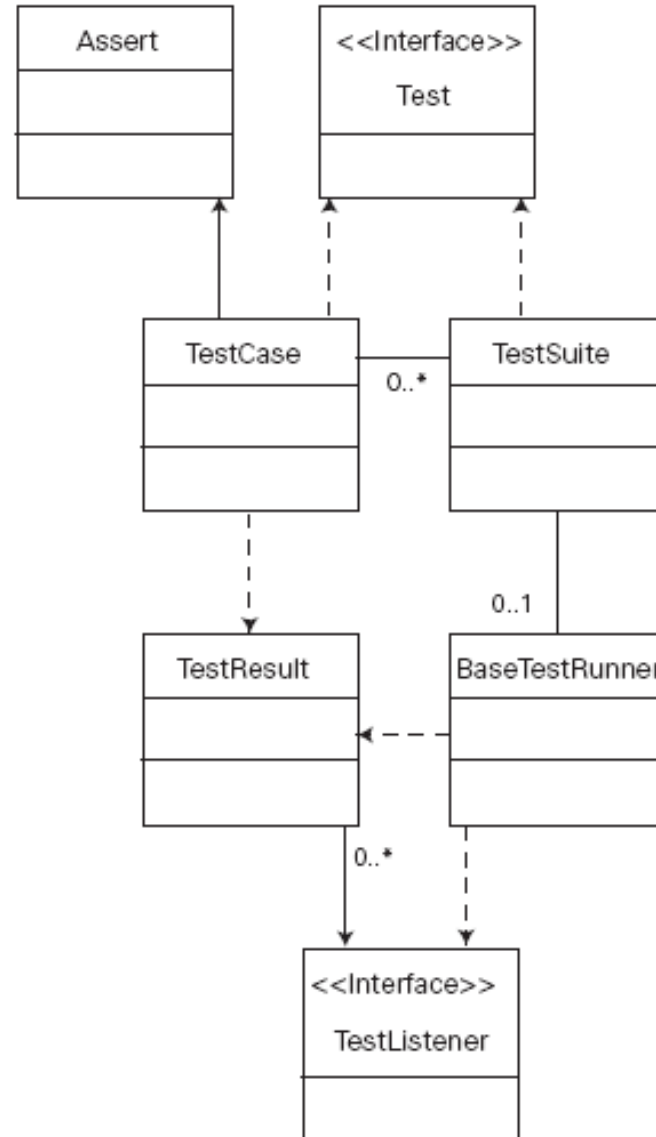


Junit 프레임워크의 이해 (1/7)

Class/Interface	책임
TestCase	TestCase는 복합적인 테스트를 수행하기 위하여 사용될 수 있는 환경을 정의한다.
TestSuite	TestSuite는 다른 test suites를 포함하는 일련의 test cases를 수행한다. 이는 Tests의 복합(composite)이다.
BaseTestRunner	test runner는 테스트를 개시(launch)하는 사용자 인터페이스이다. BaseTestRunner는 모든 test runner의 수퍼 클래스이다.
Assert	assert 메서드는 조건 성공 시에는 아무일 없지만, 조건 실패 시에는 예외를 던진다.
TestResult	TestResult는 테스트 동안 발생하는 에러나 실패(failure)를 모은다.
Test	Test는 실행되어 TestResult에 전해질 수 있다.
TestListener	TestListener는 테스트 동안 발생하는 에러나 실패와 함께 테스트의 시작과 끝을 포함하는 이벤트를 인지한다.

JUnit 프레임워크의 이해 (2/7)

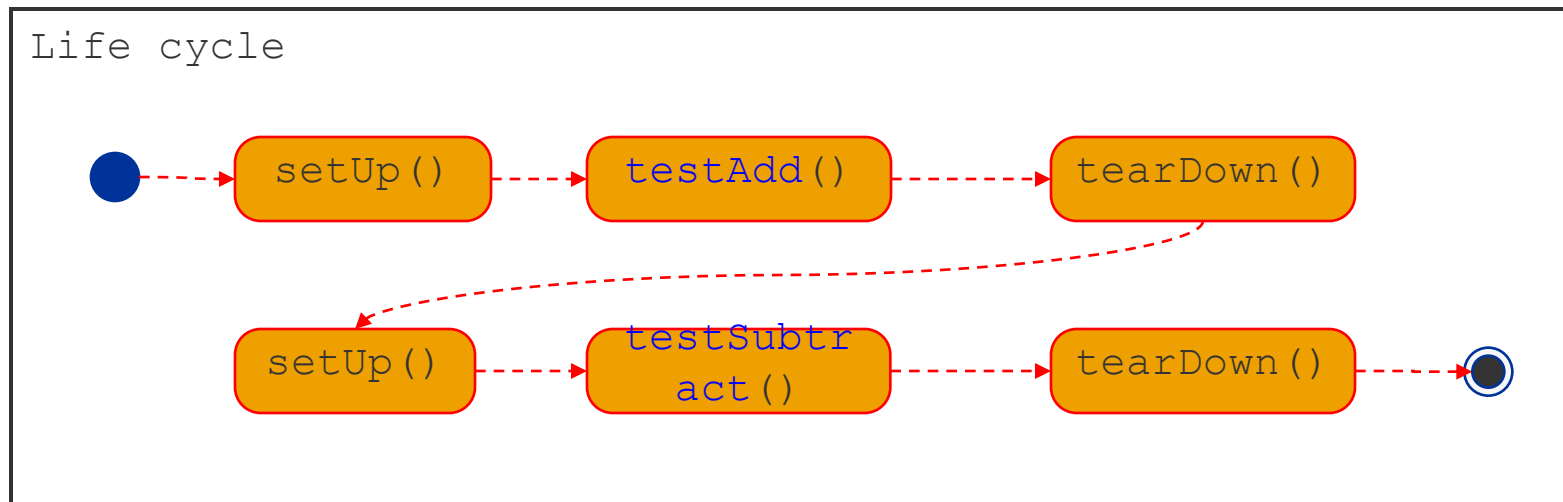
□ 핵심 클래스 다이어그램



Junit 프레임워크의 이해 (3/7)

□ TestCase

- TestCase를 상속하여 테스트 클래스를 정의
- 테스트 메서드는 `testXXX()`로 선언
- 테스트 메서드 내에서 단정(Assertion) 메서드를 사용하여 테스트의 성공 여부 확인
- `setUp()` -> `testXXX()` -> `tearDown()` 순으로 실행됨



JUnit 프레임워크의 이해 (4/7)

□ TestSuite

- 테스트의 Composite(조합)
- 테스트 자동화를 가능토록 하는 JUnit의 요소임
- 테스트 그룹을 반환하는 `suite()` 메서드를 통해 테스트 수행

```
public class AllTests {  
    public static Test suite() {  
        TestSuite suite = new TestSuite(JunitSampleTest.class);  
        suite.addTest(kr.co.nextree.tests.framework.AllTests.suite());  
        suite.addTest(kr.co.nextree.tests.runner.AllTests.suite());  
        suite.addTest(kr.co.nextree.tests.extensions.AllTests.suite());  
        return suite;  
    }  
}
```

JUnit 프레임워크의 이해 (5/7)

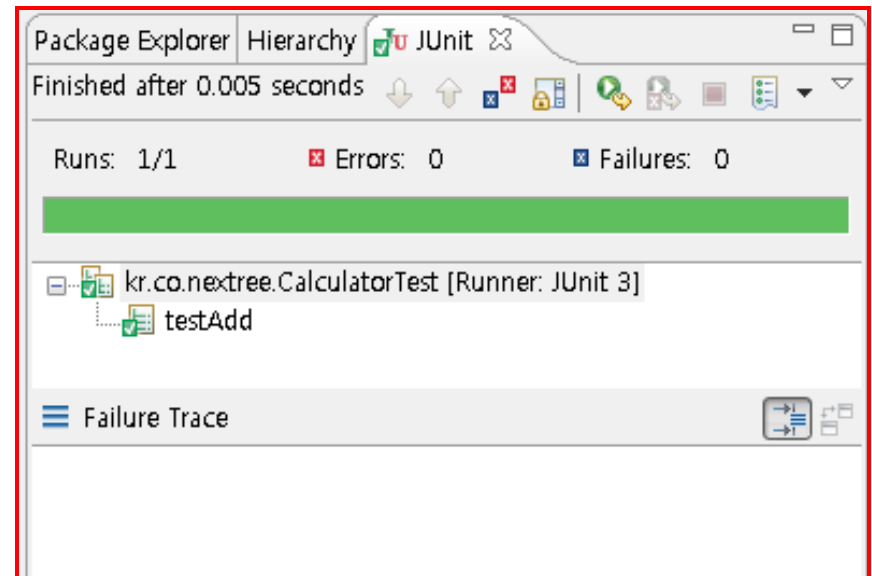
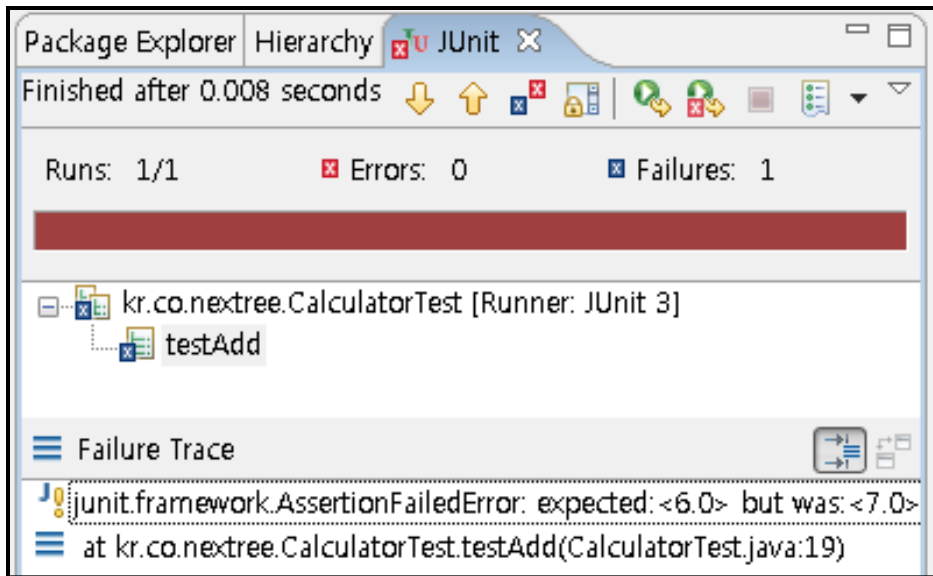
□ TestRunner

- 쉽고 빠른 테스트를 위해 JUnit은 test runner를 제공
- 수행 및 결과에 대한 통계를 제공
- 초기에는 Text/Swing/AWT Test runner를 제공하였으나, 현재는 Text 콘솔만 제공
- test runner 실행 시 JUnit 프레임워크의 행동
 - TestSuite를 생성
 - TestResult를 생성
 - 테스트 메서드를 수행 (testXXX())
- JUnit 모토: 오류 없는 코드를 위하여 초록막대를 유지해야 함

JUnit 프레임워크의 이해 (6/7)

□ TestResult

- TestCase의 수행 결과(성공 or 실패)의 세부 내용을 수집
- TestRunner는 테스트의 결과를 보고하기 위하여 TestResult를 사용함
- 테스트 결과
 - 성공 – 초록막대
 - 실패 – 빨간막대



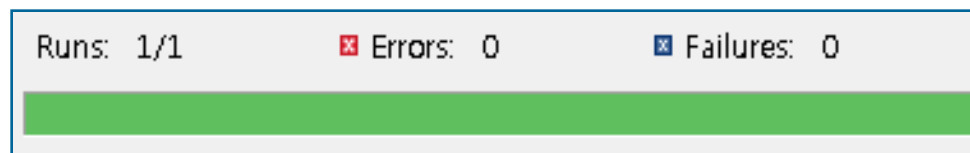
Junit 프레임워크의 이해 (7/7)

□ TestListener

- TestResult에 접근하는 객체를 도와주고, 유용한 보고서를 작성하기 위하여 TestListener 인터페이스를 제공
 - **addError** : 에러 발생시 호출됨
 - **addFailure** : 실패 발생시 호출됨
 - **startTest** : 테스트 시작 시 호출됨
 - **endTest** : 테스트 종료 시 호출됨
- TestRunner는 TestListener를 구현(implement)한 객체임

□ 실패와 에러

- **실패(Failure)** : 테스트에 의해 예견될 수 있는 상태
 - 단정 메소드의 결과과 ‘거짓’ 인 경우
 - Assert 객체의 fail을 호출한 경우
- **에러(error)** : 테스트에 의해 예견되지 못하는 상태
 - RuntimeException 등과 같이 단정문에 의한 예외가 아닌 경우



□ JUnit의 Assertion 메서드

- 테스트 개발을 쉽고 빠르게 도와주는 **Assert** 클래스의 메서드
- 조건이 다음의 설명과 다를 경우 **AssertionFailedError** 발생
- **Assert**의 결과가 “초록막대” 가 되도록 테스트 작성

Method	설명
assertTrue	조건이 참인지 판정
assertFalse	조건이 거짓인지 판정
assertEquals	두 개의 객체가 같은지 판정
assertSame	두 개의 객체가 동일한 객체를 참조하는지 판정
assertNotSame	두 개의 객체가 동일한 객체를 참조하지 않는지 판정
assertNull	객체가 null인지 판정
assertNotNull	객체가 null이 아닌지 판정
fail	강제 예외처리 – AssertionFailedError를 던짐

단정(Assertion) 메서드 (2/3)

❑ **assertEquals** vs **assertSame**

- assertEquals : Object 객체의 equals 메소드로 비교하여 참/거짓 판단

```
(expected == null && actual == null)  
(expected != null && expected.equals(actual))  
(expected instanceof String && actual instanceof String)
```

- assertSame : 동일한 객체를 참조하는가에 따라 참/거짓 판단

```
(expected == actual)
```

단정(Assertion) 메서드 (3/3)

□ 단정 메서드 활용 예

```
assertTrue(true);  
assertFalse(false);  
assertNotNull(new Object());  
assertNull(null);  
  
assertEquals(12L, 12L);  
assertEquals(new Long(12), new Long(12));  
assertEquals("Capacity", 12.0, 11.99, 0.1);  
assertEquals((Object) null, (Object) null);  
assertEquals("a", "a");  
  
assertSame(12, 12);  
assertSame(12L, 12L);  
assertSame(new Long(12), new Long(12)); // fail  
assertSame(12.0, 11.99); // fail  
assertSame((Object) null, (Object) null); // success  
assertSame("a", new String("a")); // fail
```

Junit을 이용한 단위 테스트 (1/7)

□ TestCase

```
import junit.framework.TestCase;

public class CalculatorTest extends TestCase {
    protected void setUp() throws Exception {
        super.setUp();
    }
    protected void tearDown() throws Exception {
        super.tearDown();
    }
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```

junit.framework.TestCase 를 상속

testXXX 메서드가 실행되기 전에 수행.
테스트를 위한 사전 작업을 담당.

testXXX 메서드가 실행된 후 수행.
테스트 종료 후 작업을 담당.

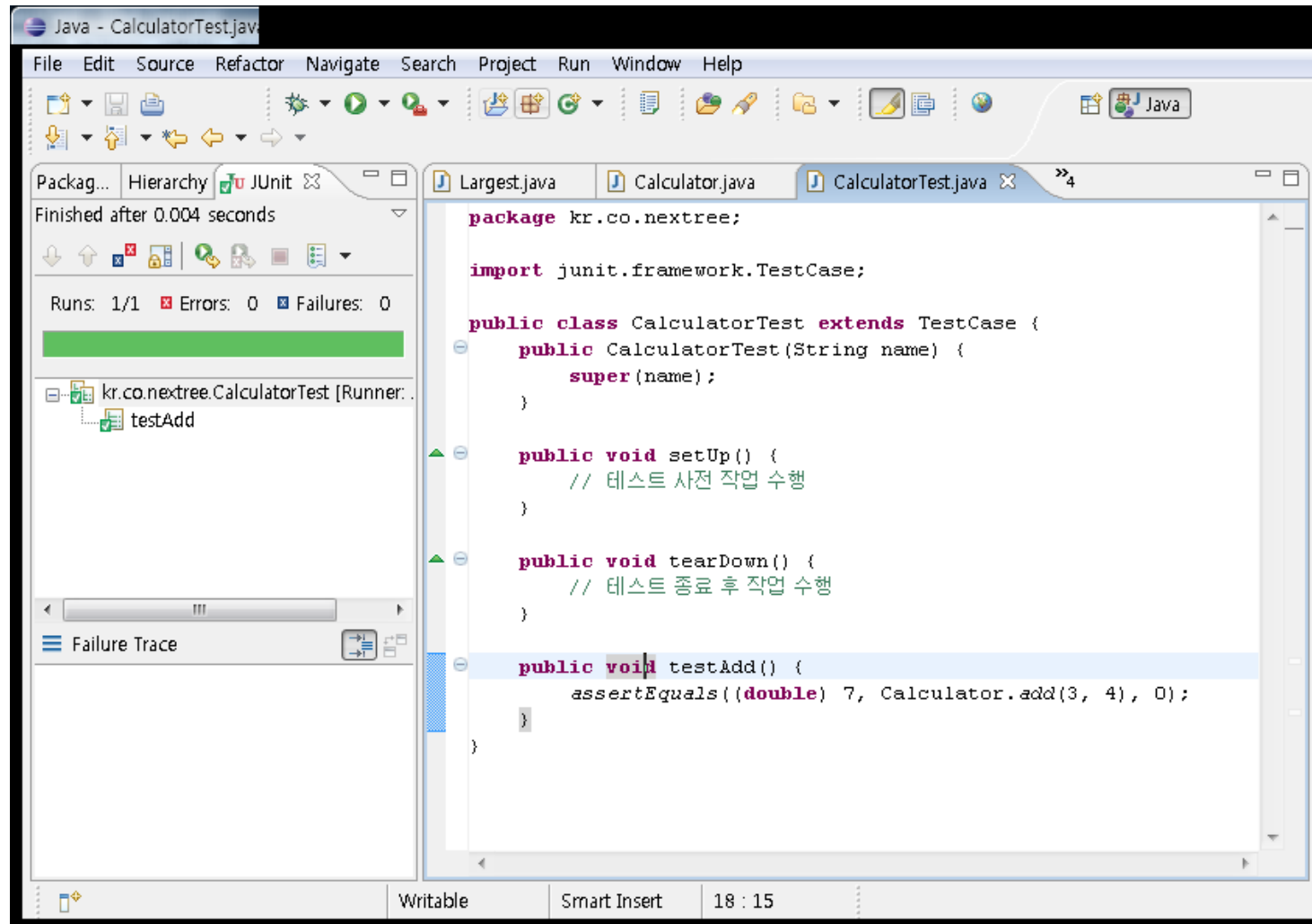
자동으로 수행될 단위 테스트 메서드.
testXXX()의 형식을 따름.

단위 테스트를 수행할 객체의 인스턴스를 생성하고 결과를 얻음

단정 메서드를 통해 테스트 결과가 정상적으로 수행되었는지 확인

Junit을 이용한 단위 테스트 (2/7)

□ TestCase 수행 결과 (CalculatorTest)



Junit을 이용한 단위 테스트 (3/7)

□ 자동(**automatic**) 스위트(**suite**) 수행하기(1)

- 디폴트 **TestSuite**는 **test**로 시작되는 모든 메서드를 찾아냄
- 내부적으로 디폴트 **TestSuite**는 각각의 **testXXX** 메서드를 위하여 **TestCase**의 인스턴스를 생성
- 호출된 메서드명은 **TestCase** 생성자로서 전달되어, 각각의 인스턴스는 고유한 식별자를 갖음

```
public static Test suite()
{
    return new
        TestSuite(TestCalculator.class);
}
```

①

TestCalculator에 대한 디폴트 **TestSuite**는 코드상에서 이와 같이 나타날 수 있다.

Junit을 이용한 단위 테스트 (4/7)

□ 자동 스위트 수행하기(2)

- 수동(manual) 스위트 – 메소드 단위 테스트 케이스 실행

```
public TestCalculator(String name) {  
    super(name);  
}  
  
public static Test suite()  
{  
    TestSuite suite = new TestSuite();  
    suite.addTest(new TestCalculator("testAdd"));  
    return suite;  
}
```

Junit을 이용한 단위 테스트 (5/7)

□ TestSuite

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class CalculatorTestSuite {
    static public Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(CalculatorTest);
        suite.addTestSuite(MultiplicationTest);
        return suite;
    }

    static public Test suite() {
        TestSuite suite = new TestSuite(CalculatorTest.class);
        suite.addTestSuite(MultiplicationTest);
        return suite;
    }
}
```

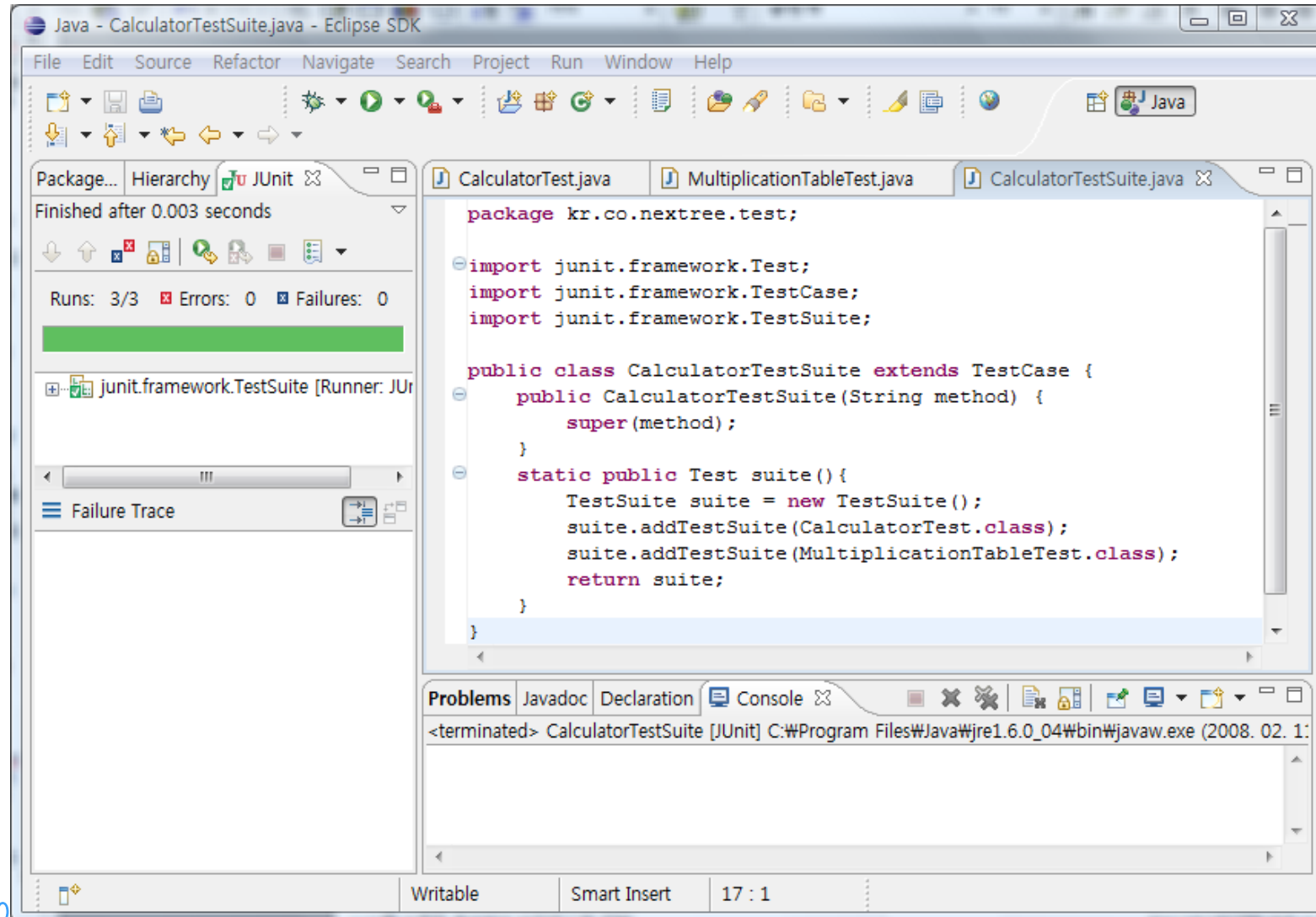
suite() 메서드를 통해 테스트를 수행할 Tests 또는 Suites 을 호출.
(하나 이상의 Test/Suite 를 수행 가능)

Default TestSuite 설정.
(Default TestSuite는 각각의 testXXX 메서드를 위하여 TestCase의 인스턴스를 생성함)

테스트 대상 TestCase/TestSuite를 추가.
(반드시 junit.framework.Test의 구현체 이어야 함)

Junit을 이용한 단위 테스트 (6/7)

□ TestSuite 수행 결과 (CalculatorTestSuite)



Java - CalculatorTestSuite.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package... Hierarchy JUnit

Finished after 0.003 seconds

Runs: 3/3 Errors: 0 Failures: 0

junit.framework.TestSuite [Runner: JUnit4Runner]

Failure Trace

```

package kr.co.nextree.test;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class CalculatorTestSuite extends TestCase {
    public CalculatorTestSuite(String method) {
        super(method);
    }

    static public Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(CalculatorTest.class);
        suite.addTestSuite(MultiplicationTableTest.class);
        return suite;
    }
}
    
```

Problems Javadoc Declaration Console

<terminated> CalculatorTestSuite [JUnit] C:\Program Files\Java\jre1.6.0_04\bin\javaw.exe (2008. 02. 1:

Writable Smart Insert 17 : 1

Junit을 이용한 단위 테스트 (7/7)

□ 자신만의 **TestSuite** 구동하기 – 모든 테스트 실행

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
import junitbook.sampling.TestDefaultController;

public class TestAll
{
    public static Test suite()
    {
        TestSuite suite = new TestSuite("All tests from part 1");
        suite.addTestSuite(TestCalculator.class);
        suite.addTestSuite(TestDefaultController.class);
        return suite;
    }
}
```

□ 테스트에서 발생하는 예상된 예외에 대한 처리

```
public void testForException() {
    try {
        sortMyList(null);
        fail("Should have thrown an exception");
    }
    catch (RuntimeException e) {
        assertTrue(true);
    }
}

public void sortMyList(List<String> lists) {
    if (lists.isEmpty()) throw new RuntimeException();
}
```

Errors : 0 , Failures : 0

예상되는 예외를 **catch**. 테스트가 정상 통과 되도록 단정문을 기술

□ 예상치 못한 예외에 대한 처리

```
public void testDetail() throws FileNotFoundException {
    FileInputStream in = new FileInputStream("data.txt");
    // ...
}
```

Errors : 1 , Failures : 0

예측 불가능한 예외는 테스트가 실패하도록 내버려 둬

JUnit 4 이후 변화

비교대상	JUnit 3.8 이전	JUnit 4.0 이후
JDK 5.0 Paradigm 도입	-	JDK 5.0 이상
		Annotation 사용
		Assertion Method를 static import 사용
Test Class	junit.framework.TestCase를 상속	TestCase를 상속받지 않음
Set up, tear down	setUp(), tearDown()	@Before, @After (다수의 메서드 지정 가능)
One-time set up, tear down	-	@BeforeClass, @AfterClass
Test Method	testXXX()로 정의	@Test @Test (expected= <i>Error.class</i>)
JUnit 4.4 이후	-	assertThat([value], [matcher statement]); assumeThat() 추가

어노테이션(Annotation)

- 소스 코드에 메타데이터를 표현하기 위한 표현 양식
- **Java5** 이전에는 **XML**을 사용하여 메타 데이터를 표현함
 - 소스와 메타정보가 분리되어 관리
 - 이를 극복하기 위해 **Xdoclet**같은 기술이 출현함
- **Java5**부터 코드 내 어노테이션 도입
 - 소스레벨의 메타정보를 참조할 수 있게 됨

```
@SupressWarning
Public List<Map >readAccountsByAccountId(String accountId) {
    try {
        return (List<Map>) sqlMapClient.queryForList( "selectAccountsByAccountId" , accountId);
    }
    catch(Throwable e) {
        e.printStackTrace();
        return null;
    }
}
```

JUnit 3.8 예제

```
package kr.co.nexttree.version;

import junit.framework.TestCase;
import kr.co.nexttree.Member;

public class Junit38Test extends TestCase {
    public Junit38Test(String name) {
        super(name);
    }

    public void setUp() throws Exception {
        super.setUp();
    }

    public void tearDown() throws Exception {
        super.tearDown();
    }

    public void testName() {
        String name = "Yoonmi";
        Member member = new Member();
        member.setName(name);
        assertEquals(name, member.getName());
    }
}
```

□ TestCase 만들기

```
public class Junit4Test {
    @Before
    public void runBeforeTest() {}
    @Before
    public void runBeforeTest2() {}
    @After
    public void runAfterTest() {}
    @Test
    public void name() {
        assertThat(x, is(3));
        assertThat(responseString,
            either(containsString("color")).or(containsString("colour")));
        assertThat(myList, hasItem("3"));
        assumeThat(File.separatorChar, is('/'));
    }
}
```

TestCase를 상속받지 않음

@ Annotation의 사용

Test Method

□ TestSuite 만들기

```
public class AllTests {  
    public static Test suite() {  
        TestSuite suite = new TestSuite(JunitSampleTest.class);  
        suite.addTest(kr.co.nextree.tests.framework.AllTests.suite());  
        suite.addTest(kr.co.nextree.tests.runner.AllTests.suite());  
        suite.addTest(kr.co.nextree.tests.extensions.AllTests.suite());  
        return suite;  
    }  
}
```

```
@RunWith(Suite.class)  
@SuiteClasses({Junit4.class, XXX.class})  
public class TestAll {  
}
```


□ 기타 기능들

```
public class Junit4Test {
    @BeforeClass
    public void runBeforeTest() {}
    @AfterClass
    public void runAfterTest() {}

    @Test(timeout=1)
    public void name() {
        ...
    }

    @Test(expected=IndexOutOfBoundsException.class)
    public void name2() {
        ...
    }

    @Ignore("이 함수는 아직 완성되지 않았음")
    @Test
    public void name2() {
        ...
    }
}
```

프로젝트에서의 테스트

- ❑ 테스트 코드의 위치
- ❑ 테스트 빈도
- ❑ 회귀 테스트와 테스트 자동화
- ❑ **Test-Driven Development(TDD)**

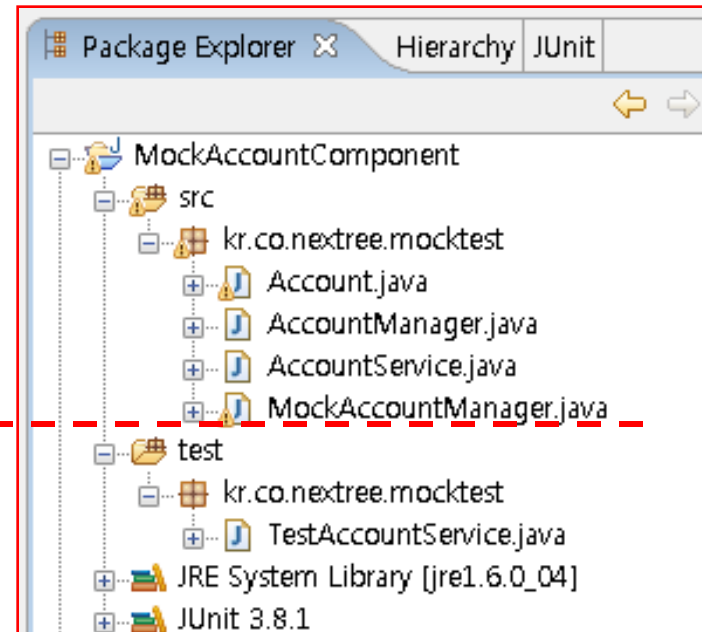
Q. 중요 기능 부분이 private, default, protected 접근 권한자를 갖는다면 어떻게 테스트 할 것인가?

A. ‘제대로 동작 안하고 테스트 안 된 코드를 잘 캡슐화하는 것보다는, 제대로 동작하고 테스트된 캡슐화를 깨뜨리는 것이 더 낫다’ 는 의견이 일반적이다.

□ **protected** 메서드의 접근이 가능한 테스트 코드의 위치

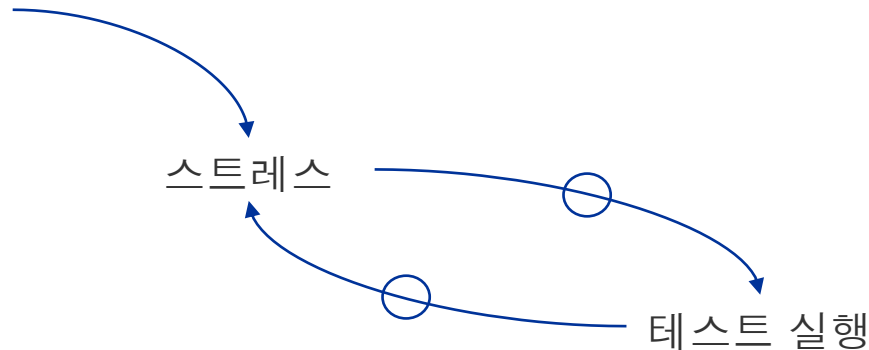
- 제품코드와 테스트코드를 같은 디렉터리에 위치시키는 방법
 - 테스트코드가 제품코드와 분리되지 않아 지저분하며, 배포판 생성시에 문제 될 수도 있음

- 제품코드와 테스트코드를 같은 패키지, 다른 소스 디렉터리에 위치시키는 방법



□ 자주, 일찍, 자동화된 테스트

- 신규 메서드 추가 시
- 프로그램의 버그 수정 시
- 성공적으로 컴파일 할 때마다
- 버전 관리 시스템에 체크인할 때마다
- 끊임없이
 - 따로 배정된 특정 컴퓨터가 자동으로 하루 종일
 - 주기적으로 또는 버전 관리 시스템에 체크인 할 때마다 전체 빌드 & 테스트 수행



회귀 테스트와 테스트 자동화

□ 회귀(regression) 테스트

- 모듈을 수정한 후에 동작에 이상이 없는지 확인하는 테스트

□ 자동화된 테스트

- 회귀 테스트를 관리하기 위한 유일한 현실적인 방법
- 개발자가 수동적으로 같은 테스트를 반복 수행하다 보면 실수를 범하게 되는데, 이를 막기위한 방법으로 자동화된 테스트의 수행이 필요
- 문제를 초기에 발견 가능하며 안정성을 높임
- XP에서 시스템의 결함을 줄이기 위한 방법으로 테스트 우선 개발과 빈번한 테스트를 가능하게 하는 자동화된 테스트를 강조

Test-Driven Development(TDD)

□ TDD의 정의

- 테스트 코드를 먼저 작성하고, 그 테스트를 통과하는 실제코드를 단계적으로 만들어 가면서 중복되는 것들은 삭제하는 개발 방법론

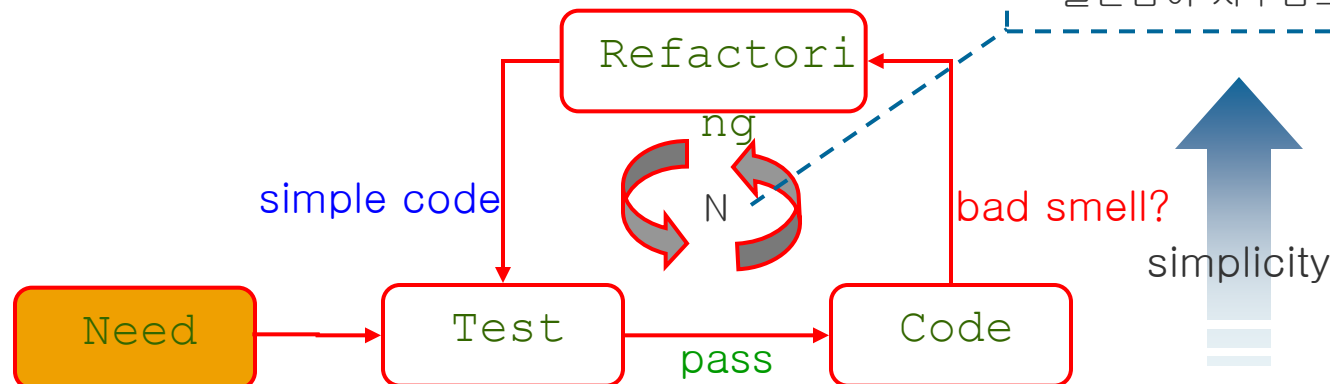
□ TDD의 목적

- Simple code

“Clean code that works!(작동하는 깨끗한 코드)” – Ron Jeffries

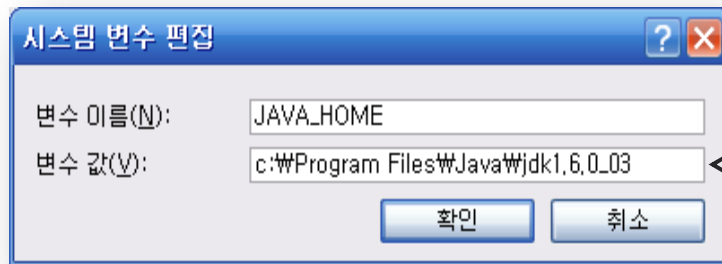
□ TDD의 흐름 (TDD development cycle)

- 가장 간단한 코드가 만들어질 때까지 (중복이 없을 때까지)
- 불안함이 지루함으로 변할 때까지

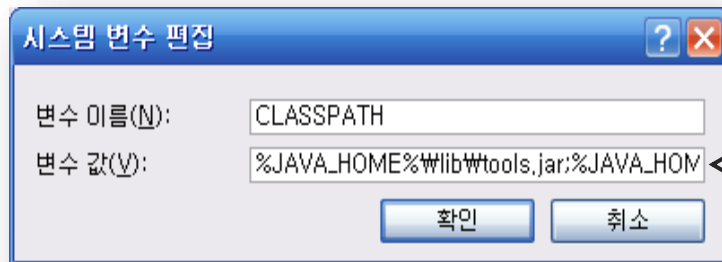


JDK (The Java SE Development Kit) (1/2)

- ❑ JDK는 버전 **6**을 사용 (**Update 3**)
- ❑ 다운로드 링크: <http://java.sun.com/javase/downloads/index.jsp>
- ❑ 설치 후 환경변수 설정
 - 내 컴퓨터 > 속성 > 고급 > 환경변수에 JAVA_HOME 및 CLASSPATH속성 등록



----- 다운로드 후 설치한 **JDK** 디렉토리 경로를 등록한다.



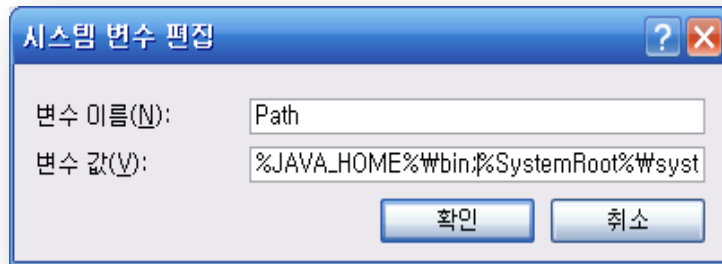
----- 설치한 **JDK** 라이브러리 경로를 클래스패스에 추가한다.

·
`%JAVA_HOME%| lib| rt.jar;`
`%JAVA_HOME%| lib| dt.jar;`
`%JAVA_HOME%| lib| tools.jar`

JDK (The Java SE Development Kit) (2/2)

□ 설치 후 환경변수 설정

- 또한 Path 변수 값에 %JAVA_HOME%\ bin을 맨 앞에 추가



- DOS 창에서 환경변수 설정을 확인

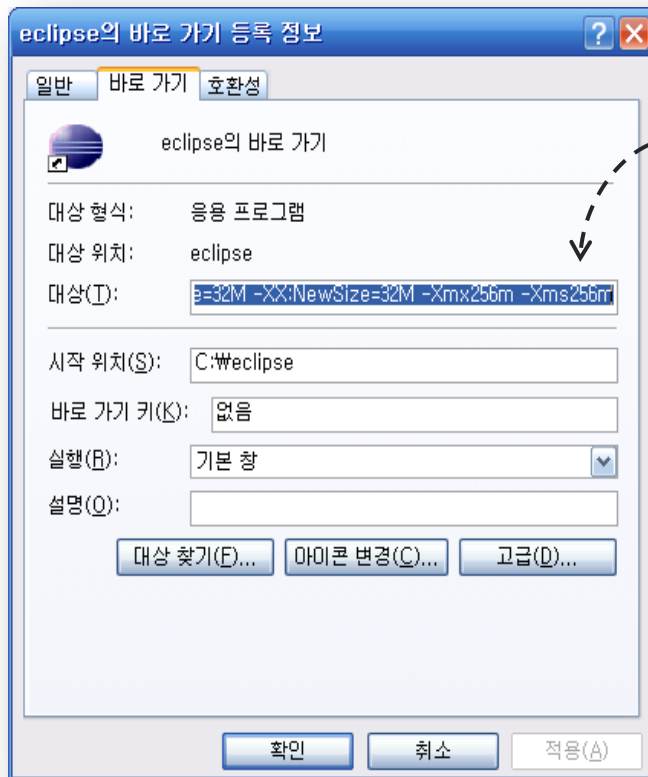
```
C:\Documents and Settings\Administrator>java -version
java version "1.6.0_03"
Java(TM) SE Runtime Environment (build 1.6.0_03-b05)
Java HotSpot(TM) Client VM (build 1.6.0_03-b05, mixed mode, sharing)
```

```
C:\Documents and Settings\Administrator>echo %CLASSPATH%
c:\Program Files\Java\jdk1.6.0_03\lib\tools.jar;c:\Program Files\Java\jre1.6.0_03\lib\rt.jar;c:\hsqldb\lib\hsqldb.jar
```

```
C:\Documents and Settings\Administrator>PATH
PATH=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\Softex\OmniPass;C:\Program Files\ESTsoft\ALZip;c:\Program Files\Java\jdk1.6.0_03\bin;C:\maven-2.0.7\bin;C:\Program Files\DM Computer Solutions\UltraEdit-32;C:\hsqldb\bin;C:\Program Files\ESTsoft\ALZip
```


Eclipse IDE for Java Developers

- ❑ IDE로 Eclipse를 사용할 경우 버전 3.2 이상 (3.3 Europa 권장)
- ❑ 다운로드 링크: <http://www.eclipse.org/downloads/>
- ❑ 하드디스크의 적당한 위치에 압축 해제
- ❑ **eclipse.exe** 바로가기 아이콘 생성 후 실행환경 설정



대상에 다음과 같은 옵션을 추가한다.

C:\eclipse | eclipse.exe -data D:\ Projects and Developing | Workspace | KOSTA | EJB3 -vmargs -Xverify:none -XX:+UseParallelGC -XX:PermSize=20M -XX:MaxNewSize=32M -XX:NewSize=32M -Xmx256m -Xms256m

- data 옵션은 eclipse 실행 시 대상 workspace를 지정하는 역할 담당한다. 만약 -data 옵션이 없을 경우에는 직접 workspace를 지정할 수 있다.

나머지 옵션들은 eclipse 실행 시 JVM에 할당하는 메모리 설정과 관련이 있다.

주의사항)) Eclipse 다운로드 사이트에서
Eclipse IDE for Java Developers – Windows (78MB)
Eclipse IDE for Java EE Developers – Windows (126MB)
 둘 중 하나를 선택해서 Download 받는다.

2. JUnit 자동화

ONE STEP AHEAD

- ☐ **Ant**란 무엇인가?
- ☐ **Ant**를 이용한 빌드 및 테스트
- ☐ **Maven**개요
- ☐ **Maven** 시작하기
- ☐ **Maven**에서 **Junit** 테스트 실행하기
- ☐ **Eclipse** 소개
- ☐ **Eclipse**에서 테스트 구동하기

왜 자동화인가?

- 효과적인 **Unit** 테스트를 위해서 단위테스트는 개발 과정의 한 부분이 되어야 함
- 한 프로젝트는 수백, 수천 개의 클래스를 포함
- 결국 자동화는 필수

- ❑ **Ant** 소개
- ❑ **Ant** 설치 및 실행
- ❑ **Ant** 구성
- ❑ **Ant**로 **Junit** 실행하기

□ Java 기반의 자동화 빌드 툴(Build Tool)

- **Another Neat Tool**(또 다른 멋진 도구) 의 축약
- Unix의 make와 비슷함
- make는 Shell 스크립트, Ant는 XML로 스크립트 작성
- O/S 독립적
- 1998년에 Project가 시작됨
- 현재 1.7.0 버전, <http://ant.apache.org/manual/index.html>

□ 왜 Ant를 사용하나?

- 대부분의 프로젝트는 다량의 클래스를 포함
- 클래스패스 추가, 소스 컴파일, jar파일 생성, 테스트 일괄 실행에 사용할 수 있는 일관된 컴파일 및 테스트 방법이 필요
- 쉽고 간편한 도구가 요구됨

□ Ant의 특성

- 어플리케이션의 컴파일 및 테스트의 용이성
- 자바 어플리케이션의 빌드를 위한 **de facto** 표준
- 빌드파일(Buildfile)이라는 XML 문서를 통해 환경설정
 - 디폴트 이름은 build.xml
 - 프로젝트에 적용할 각 작업을 서술

Ant 설치 및 실행

□ 설치

- Ant.apache.org 에서 최신 버전을 다운로드 (2008.06 현재 : 1.7.0)
- 최근에는 Eclipse와 같은 IDE에 기본적으로 포함되어 배포되고 있음
- Binary 다운로드 후 압축 해제

□ 환경설정

- 환경 변수 설정 - Ant설치 디렉토리를 ANT_HOME으로 지정
- 시스템 패스 설정 - %ANT_HOME%\ bin
- junit 사용을 위해 /lib/에 junit.jar 추가

□ 실행

- 커맨드 라인 입력 사용
- IDE에서 제공하는 GUI를 이용

위와 같은 방식을 통해 JUnit 테스트를 실행시킬 수 있음

□ Project와 Target Element로 구성, Task로 표현

```
<project>

  <target name="clean">
    <delete dir="build"/>
  </target>

  <target name="compile">
    <mkdir dir="build/classes"/>
    <javac srcdir="src" destdir="build/classes"/>
  </target>

  <target name="jar">
    <mkdir dir="build/jar"/>
    <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
      <manifest>
        <attribute name="Main-Class" value="oata.HelloWorld"/>
      </manifest>
    </jar>
  </target>

</project>
```


□ Ant 빌드 파일 만들기

- XML을 이용한 빌드파일 구성
- 빌드 파일은 하나의 프로젝트와 여러 개의 타겟 엘리먼트로 구성됨
- 타겟은 타스크 엘리먼트를 포함

□ 빌드파일(Build File)

- 하나의 빌드파일은 하나의 프로젝트와 연관
- build.xml의 최상위 xml 태그는 “project”.
- 디폴트 Target을 갖을 수 있어 어떠한 매개변수가 없어도 Ant의 실행이 가능.

□ Project 엘리먼트

애틀리뷰트	설명	필수 여부
Name	프로젝트를 대표하는 이름	아니요
default	초기치 작업 타겟 지정 없이 수행할 경우 default로 지정 된 타겟을 수행	아니요
Basedir	프로젝트에서 사용할 기준 폴더 설정	아니요

□ Target 엘리먼트

- 빌드의 대상이 되는 엘리먼트
- 하나 이상이 존재해야 함
- 하나의 타겟은 다른 타겟에서 참조(의존) 가능

애트리뷰트	설명	필수 여부
Name	타겟의 이름	예
depends	이 타겟이 의존하는 타겟의 이름, 콤마로 분리하여 여러 개를 지정 가능	아니요
if	이 타겟이 실행되기 위해서 설정돼야 하는 프로퍼티 이름	아니요
unless	이 타겟이 실행되기 위해서 설정되면 안되는 프로퍼티 이름	아니요
description	이 타겟에 대한 설명	아니요

□ Target 엘리먼트 - 계속

```
<target name="A"/>  
<target name="B" depends="A"/>  
<target name="C" depends="B"/>  
<target name="D" depends="C,B,A"/>
```

```
<target name="myTarget" depends="myTarget.check" if="myTarget.run">  
<echo>Files foo.txt and bar.txt are present.</echo>  
</target>
```

```
<target name="myTarget.check">  
<condition property="myTarget.run">  
<and>  
    <available file="foo.txt"/>  
    <available file="bar.txt"/>  
</and>  
</condition>  
</target>
```

□ Task 엘리먼트

- Target 엘리먼트 내에서 수행되는 코드 조각
- 여러 개의 애트리뷰트(아규먼트)를 가질 수 있음

```
<name attribute1="value1" attribute2="value2" ... />
```

- Ant는 많은 수의 내장(built-in) 타스크 및 선택(optional) 타스크지원
- 사용자가 직접 커스텀 (custom)타스크를 제작할 수도 있음

❑ Built-in Tasks

Task 이름	설명
Java	Java 클래스 실행
Javac	Java 소스파일 컴파일
Jar	Jar 파일 생성
Javadoc	JavaDoc 생성
Copydir	디렉토리 카피
Copyfile	파일 카피
Mkdir	디렉토리 생성
Mail	메일 전송

□ Optional-Taks

Task 이름	설명
Junit	Junit testcase 실행
JUnitReport	Junit Task에서 생성된 결과 XML파일을 합병하여 통합 결과 문서 생성
PropertyFile	프로퍼티 파일을 조작할 수 있는 기능 제공
EJB Tasks	EJB 컴파일, DD생성 등에 관련된 다양한 타스크 제공
FTP	FTP 서버에 접속하여 파일을 주고 받을 수 있음

□ Property 엘리먼트

- 프로젝트 내에서 전역적으로 사용하기 위한 값을 세팅
- 다량의 **Target**이 같은 설정을 공유할 때 유용
- 명시적인 설정을 캡슐화하며 빌드파일 전체에 걸쳐 재사용이 가능
- **Property** 설정 값은 불변
 - 앞에서 이미 설정된 값은 나중에 재설정 되더라도 반영되지 않음
- 빌드파일의 새로운 환경으로의 적용을 용이하게 함

□ Property 엘리먼트 - 계속

애트리뷰트	설명	필수 여부
name	프로퍼티 이름	아니요
value	프로퍼티에 세팅할 값	name 속성 사용시, 셋 중 하나는 필수
location	파일 경로	
refid	다른 곳에서 정의된 객체에 대한 참조	
resource	프로퍼티 파일의 리소스 이름	Name 속성 사용하 지 않을 경우, 넷 중 하나는 필수
file	프로퍼티 파일의 파일 이름	
url	프로퍼티 파일의 URL	
enviroment	환경변수를 읽을 때 사용하는 PREFIX	
classpath	리소스를 찾을 때 사용하는 클래스패스	아니요
classpathref	리소스를 찾을 때 사용하는 클래스패스 참조값	아니요
prefix	File, resource를 사용해서 로드되는 프로퍼티에 적 용하기 위한 prefix	아니요

□ Property 엘리먼트 - 예제

```
<property name="foo.dist" value="dist"/>
```

```
<property file="foo.properties"/>
```

```
<property url="http://www.mysite.com/bla/props/foo.properties"/>
```

```
<property resource="foo.properties"/>
```

```
<property file="${user.home}/.ant-global.properties"/>
```

```
<property environment="env"/>
```

```
<echo message="Number of Processors = ${env.NUMBER_OF_PROCESSORS}"/>
```

```
<echo message="ANT_HOME is set to = ${env.ANT_HOME}"/>
```

□ Project, Property 설정 예제

- 프로젝트 이름 : sampling
- 디폴트 타겟 : test
- Build.properties 파일을 기본 프로퍼티로 포함
- 각종 파일 위치 설정을 위한 프로퍼티

```
<project name="sampling" default="test">  
  <property file="build.properties"/>  
  <property name="src.dir" location="src"/>  
  <property name="src.java.dir" location="${src.dir}/java"/>  
  <property name="src.test.dir" location="${src.dir}/test"/>  
  <property name="target.dir" location="target"/>  
  <property name="target.classes.java.dir"  
    location="${target.dir}/classes/java"/>  
  <property name="target.classes.test.dir"  
    location="${target.dir}/classes/test"/>
```

[...]

□ Target, Task 설정 예제

```
<target name="compile.java">
  <mkdir dir="${target.classes.java.dir}"/>
  <javac destdir="${target.classes.java.dir}">
    <src path="${src.java.dir}"/>
  </javac>
</target>
<target name="compile.test" depends="compile.java">
  <mkdir dir="${target.classes.test.dir}"/>
  <javac destdir="${target.classes.test.dir}">
    <src path="${src.test.dir}"/>
    <classpath>
      <pathelement location="${target.classes.java.dir}"/>
    </classpath>
  </javac>
</target>
<target name="compile" depends="compile.java,compile.test"/>
```

Ant로 Junit 테스트 실행하기

□ Junit Task 이해하기

- Junit 테스트 프레임워크를 이용하여 테스트를 구동하기 위한 타스크
- Junit 3.0 이후버전부터 사용가능
- Junit 타스크를 사용하기 위해서는 **junit.jar**가 적절히 설정되어야 함

애트리뷰트	설명	필수 여부
printsummary	각 테스트케이스에 대한 한 줄 통계를 제공 {on,off, withOutAndErr}	기본값- off
fork	개별 VM에서 테스트 실행	기본값 - off
forkmode	얼마나 많은 개별 VM을 생성할 것인지 여부 {preTest, preBatch, once}	기본값 - preTest
haltonerror	테스트 도중 에러가 발생하면 테스트 중지	기본값 - off
errorproperty	에러 이벤트를 기록하기 위한 프로퍼티 이름	아니요
haltonfailure	테스트가 실패하면 테스트 중지	아니요
failureproperty	실패 이벤트를 기록하기 위한 프로퍼티 이름	아니요
filtertrace	에러/실패 스택 트레이스에서 junit, ant 스택 제거	기본값 - on

Ant로 Junit 테스트 실행하기

□ Junit Task 이해하기 - 계속

애트리뷰트	설명	필수 여부
timeout	테스트케이스 실행이 주어진 시간 내에 끝나지 않으면 취소, fork가 켜져있을 때만 적용	아니요
maxmemory	Fork된 VM에서 사용할 수 있는 최대 메모리, fork가 켜져있을 때만 적용	아니요
jvm	자바 VM을 실행하기 위해 사용하는 명령어 fork가 켜져있을 때만 적용	기본값 - java
dir	VM을 실행하는 디렉토리	아니요
newenviroment	새 환경 변수가 설정되면 이전 환경 변수는 무시	기본값 - false

Ant로 Junit 테스트 실행하기

□ Junit Task 이해하기 - 계속

�트리뷰트	설명	필수 여부
showoutput	테스트 출력을 Ant 로깅 시스템으로도 출력	아니요
outputtoformatters	테스트 출력을 테스트 포맷터로 출력	아니요
tempdir	임시 파일 저장 위치	아니요
reloading	각 테스트케이스마다 새로운 클래스로더를 사용할지 여부	기본값 - true
clonevm	Ant가 실행되는 VM과 새로 만들어지는 VM이 모든 프로퍼티값과 클래스패스값을 똑같이 사용하도록 함	기본값 - false

Ant로 Junit 테스트 실행하기

□ Junit 실행을 위한 빌드파일 설정

```
<target name="test" depends="compile">
  <junit printsummary="yes" haltonerror="yes" haltonfailure="yes"
        fork="yes">
    <formatter type="plain" usefile="false"/>
    <test name="junitbook.sampling.TestDefaultController"/>
    <classpath>
      <pathelement location="${target.classes.java.dir}"/>
      <pathelement location="${target.classes.test.dir}"/>
    </classpath>
  </junit>
</target>
```


Ant로 Junit 테스트 실행하기

□ 실행 결과

```

C:\WINDOWS\system32\cmd.exe

C:\Java\Test\junitbook\sampling>ant
Buildfile: build.xml

compile.java:

compile.test:

compile:

test:
[junit] Running junitbook.sampling.TestDefaultController
[junit] Tests run: 5, Failures: 0, Errors: 0, Time elapsed: 0.09 sec
[junit] Testsuite: junitbook.sampling.TestDefaultController
[junit] Tests run: 5, Failures: 0, Errors: 0, Time elapsed: 0.09 sec

[junit] Testcase: testAddHandler took 0 sec
[junit] Testcase: testProcessRequest took 0.01 sec
[junit] Testcase: testProcessRequestAnswersErrorResponse took 0 sec
[junit] Testcase: testGetHandlerNotDefined took 0 sec
[junit] Testcase: testAddRequestDuplicateName took 0 sec

BUILD SUCCESSFUL
Total time: 1 second
C:\Java\Test\junitbook\sampling>_
  
```

Ant로 Junit 테스트 실행하기

□ JunitReport Task 이해하기

- Optional Ant Task
- 각 테스트케이스 실행결과(XML)를 Merge하여 HTML형태로 출력
- 실행결과를 XSLT를 이용하여 변경하기 때문에 XSL이 필요함

애틀리뷰트	설명	필수 여부
Tofile	보고서 파일 이름	아니요
toDir	보고서를 출력할 디렉토리	아니요

□ Reprot 엘리먼트 이해하기

- JunitReport의 하위 엘리먼트

애틀리뷰트	설명	필수 여부
format	생성되는 보고서 형식,{noframes, frames}	아니요
styledir	스타일시트가 위치한 디렉토리 Junit-frames.x니, junit-noframe.xsl	아니요
toDir	변환 결과 파일이이 위치할 디렉토리	아니요

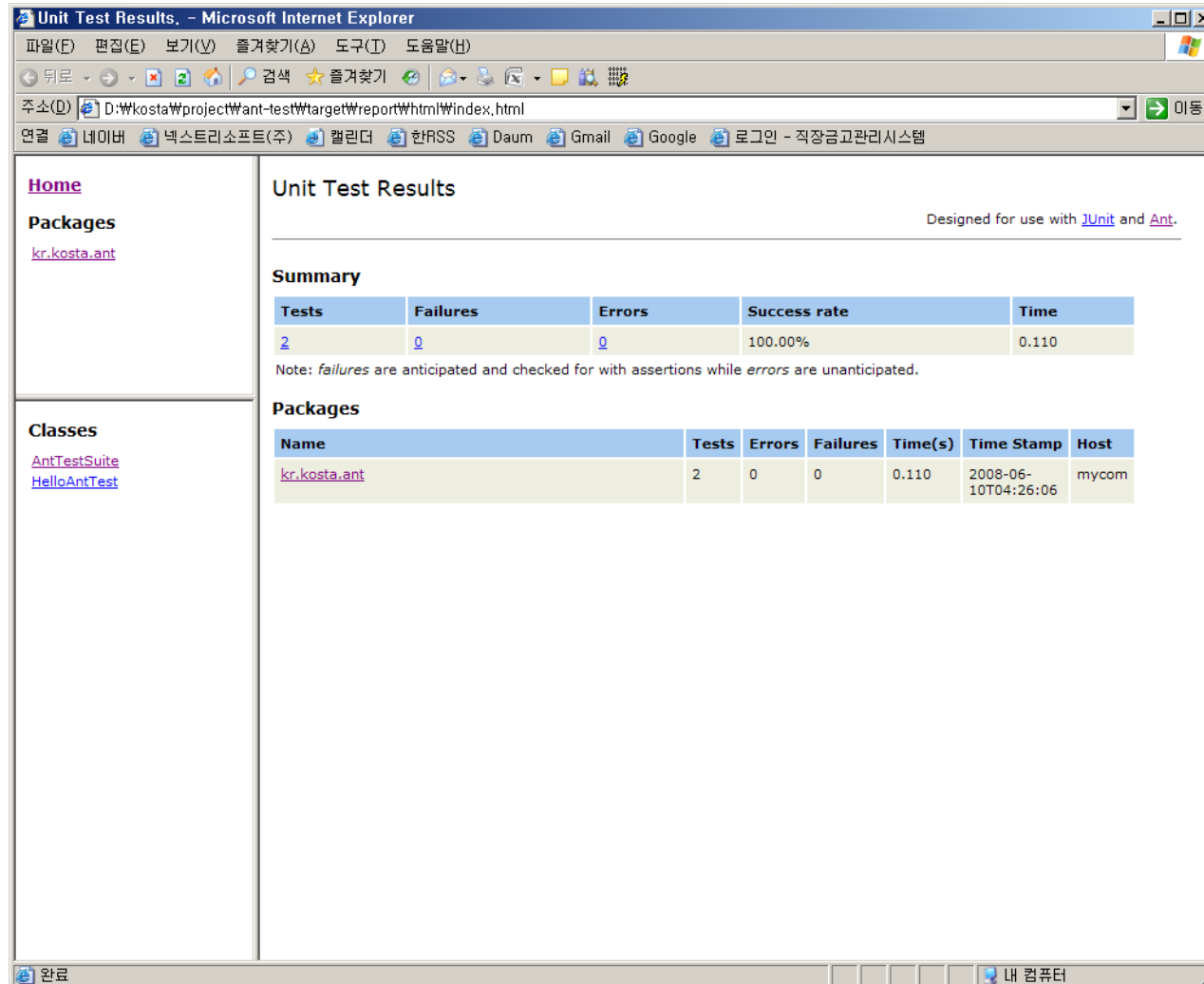
Ant로 Junit 테스트 실행하기

□ JunitReport - 예제

```
<target name="report" depends="test">
  <mkdir dir="${target.report.dir}/html"/>
  <junitreport todir="${target.report.dir}">
    <fileset dir="${target.report.dir}">
      <include name="TEST-*.xml"/>
    </fileset>
    <report todir="${target.report.dir}/html"/>
  </junitreport>
</target>
```

Ant로 Junit 테스트 실행하기

□ JunitReport - 예제 실행 결과



Unit Test Results. - Microsoft Internet Explorer

주소(D) D:\kosta\project\Want-test\target\report\html\index.html

Unit Test Results

Designed for use with [JUnit](#) and [Ant](#).

Summary

Tests	Failures	Errors	Success rate	Time
2	0	0	100.00%	0.110

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
kr.kosta.ant	2	0	0	0.110	2008-06-10T04:26:06	mycom

Home

Packages

[kr.kosta.ant](#)

Classes

[AntTestSuite](#)

[HelloAntTest](#)

완료

내 컴퓨터

- 개요
- 메이븐 정의
- 이점

□ Maven은 소프트웨어 프로젝트에 대해 광범위한 접근방법을 제공

- 컴파일, 배포, 문서, 팀 협력
- 재사용에 대한 촉진 및 프로젝트 빌드 이외의 많은 업무를 수행하는데 필요한 추상화된 메커니즘을 제공

Apache 사이트의 메이븐 소개

메이븐은 유대인 단어로 지식의 축적(accumulator of knowledge)라는 뜻으로, 원래는 Jakarta Turbine 프로젝트에서 빌드 프로세스를 단순화 시키려고 시작했던 작업임. 모두 약간씩 다른 Ant 빌드 파일과 CVS에 체크인되어야 하는 JAR 파일을 각각 가지는 몇 가지 프로젝트가 있었음. 프로젝트들을 빌드하는 일관화된 방법과 프로젝트가 어떠한 것들로 구성되어 있는지 명확한 정의와 프로젝트 정보를 정리하는 손쉬운 방법과 여러 프로젝트들에서 JAR를 공유하는 방법에 대해서 표준적인 방법을 원했음.

그 결과로 자바 기반 프로젝트를 빌드하고 관리하는 데에 사용될 수 있는 도구가 필요했음. 자바 개발자들이 더 쉽게 하루 작업을 수행하고 자바 기반 프로젝트에 대한 범용적으로 이해를 하는데 돕는 데에 maven이 사용되기를 희망함.

□ 프로젝트 관리 프레임워크 *project management framework*

- 프로젝트를 관리하고 표현하는 데에 사용되는
 - 여러 표준들
 - 레파지토리 형식
 - 몇 가지 소프트웨어
- 프로젝트 산출물을 빌드, 테스트, 배포하기 위한 표준적인 생명 주기를 정의
- 메이븐의 표준을 준수하는 모든 프로젝트에 대해서
 - 프로젝트 간 빌드 로직 *cross-cutting build logic*을 손쉽게 재사용 가능하게 함.
- 메이븐 적용 프로젝트의 이점
 - 프로젝트들 간의 관계를 빌드하고 이러한 관계들을 조사하고 레포트하는 시스템 빌드가 더 쉬워짐
 - 메이븐의 표준 형태는 프로젝트를 프로그래밍 하는데 있어서 “시맨틱 웹” *semantic web* 과 같은 종류를 가능하게 함.
 - 메이븐의 표준과 중앙 레파지토리는 프로젝트들에 대한 새로운 네이밍 시스템을 정의하고 있음.
 - 외부 의존성을 추가하고 새로운 컴포넌트를 만들기가 더 쉬워짐.

□ ASF Apache Software Foundation 의 프로젝트

- 메이븐 이전의 프로젝트 개발
 - 컴파일, 배포, 웹 사이트 생성에 대해서 서로 다른 접근 방식을 취함.
 - 수많은 좋은 프로젝트가 제각각 다른 빌드 체계를 가지고 있기 때문에 이를 사용하는 또 다른 프로젝트에서는 서로 다른 방식의 빌드 체계를 사용해서 또 다른 빌드 체계를 만들어 냄.
 - 소프트웨어를 빌드 하는데 공통적인 접근방식이 없다는 의미는
 - 새로운 프로젝트마다 기존의 다른 프로젝트의 빌드 체계를 **copy-paste** 한다는 의미임.
 - 빌드 재사용의 **copy-paste**는 우수한 품질의 소프트웨어를 개발하는 중앙 작업으로부터 빌드 체계를 유지할 수많은 작업이 필요함을 의미함.
 - 다른 빌드 체계를 가진 프로젝트로의 진입 장벽은 매우 높음.
- Turbine 프로젝트
 - 메이븐이 효율적인 프로젝트 관리 도구로 사용됨.
 - 서브 컴포넌트 간 자유롭게 이동이 가능함.
 - 개발자가 한 프로젝트가 어떻게 빌드 되는지를 이해하면, 다른 프로젝트로 이동해서 동일한 과정을 겪을 필요가 없음.
 - 컴파일, 테스트, 패키징, 문서 생성, 메트릭 및 리포트 생성, 배포 등을 메이븐 방식으로 처리
- 앤트와의 관계
 - 메이븐은 단순한 빌드 도구가 아니며 앤트의 대체가 아님.
 - 앤트가 빌드를 스크립팅하는 기능을 제공하는 반면에 메이븐은 공통의 빌드 전략을 재사용함으로써 프로젝트 관리를 촉진시키기 위한 패턴에 대한 응용임.
 - 메이크 파일과 앤트는 메이븐을 대체할 수 있음.

□ 표준화된 선언적인 방법

- 메이븐을 사용하는 프로젝트
 - 투명성, 재사용성, 유지보수성, 이해도가 향상됨
 - 소프트웨어 프로젝트에 대한 동일한 접근방법을 가능하게 함
 - 어느 한 메이븐 프로젝트를 빌드하면, 다른 프로젝트 모두에도 동일하게 적용이 가능함
- POM *Project Object Model*
 - 메이븐 프로젝트의 구조와 내용을 설명하고 있음
 - 개발자들이 자신의 빌드 프로세스를 만드는 것이 아니라, 표준적인 방법을 제공함
 - Goal과 Dependency를 선언, 메이븐의 기본 구조와 플러그인 기능을 사용
 - 프로젝트 관리에 대한 많은 작업과 빌드 과정(컴파일, 테스트, 조립, 인스톨)이 POM과 적절한 플러그인으로 위임
 - 개발자들은 개별 플러그인이 어떻게 동작하는지를 이해할 필요가 없음
- 기능
 - 빌드 프로세스에 대한 시간을 절약하고, 더 투명하게 만듦
 - 소스 파일, 문서, 결과에 대한 위치를 표준화시킴
 - 프로젝트 문서에 대한 공통적인 형태를 제공
 - 공유된 저장소로부터 프로젝트 dependency를 뽑아냄
 - 소프트웨어 프로젝트에 대한 폭넓은 모델
 - 선언적인 모델 (POM)과 상호작용하는 도구(장치)

□ 메이븐으로부터 얻는 장점

- 일관성 *coherence*
 - 조직에게 최상의 기법으로 표준화시키는 것을 가능하게 함
 - 보다 덜 불투명한 표준적인 모델에 고착화시킴
- 재사용성 *reusability*
 - 메이븐은 재사용성의 기반하게 만들어짐
 - 전체 업계의 최상의 기법을 효율적으로 재사용하게 함
- 민첩성 *agility*
 - 컴포넌트 뿐만 아니라 빌드 로직 재사용성의 장벽을 낮춰줌
 - 컴포넌트를 만들고 여러 프로젝트 빌드로 통합하는 것을 더 수월하게 함
 - 기존의 빌드 체계로 만들어진 프로젝트 간의 이동이 별도의 학습없이 가능함
- 유지보수성 *maintainability*
 - 빌드에 대한 노력보다 어플리케이션 구성에 대해 초점을 맞출 수 있음
 - 일관되고 공통적인 모델을 사용함으로써 유지보수가 더 안정됨

□ 패턴의 원칙으로부터 영감을 받음

- 설정에 대한 형식 *Convention Over Configuration*
- 빌드 로직의 재사용
- 선언적인 실행
- Dependency의 일관된 구조

Christopher Alexander의 사상

패턴은 문제와 그 해결책에 대한 통찰과 경험을 교환하는 공통적인 언어를 만드는데 도움을 준다.

원칙 – 설정에 대한 형식 (1/2)

□ 이해가 빠른 기본 전략

- 대부분의 공통 작업을 기본 뼈대를 통해 작업
- ROR *Ruby on Rails* 진영의 사상
 - “설정에 대한 형식”
 - 표준적인 형식은
 - 다른 이들과의 의사소통을 수월하게 해주면, 적은 노력으로 빨리 어플리케이션을 개발하도록 해줌.

ROR의 창시자 David Heinemeier Hansson의 말

레일은 독선적인 소프트웨어이다. 레일은 중요한 위치에서 소프트웨어의 기존 이상이 놓이는 것 으로부터 피한다. 그러한 이상 중에 한가지가 유연성 *flexibility*으로, 가능한 한 많은 접근 방법을 시도하는 것을 노력해야 하고, 다른 것과 대비해서 개발의 한 형태에 대해 판단을 하지 말아야 하는 개념이다. 레일이 그러하며, 이것이 왜 그렇게 작용하는지를 내가 믿는 바이다.

레일을 사용하면, 어플리케이션 레벨에서 유연성을 얻기 위해서 인프라 수준에서의 유연성을 사용한다. 만일 레일에 내장된 황금 경로를 따라서 작업한다면, 어플리케이션 레벨에서 더 많이, 더 빠르고, 더 잘 작업할 수 있는 생산성 측면에서 많은 보상을 받게 된다.

독선적인 소프트웨어의 한가지 특징은 “설정에 대한 형식”이라는 개념이다. 만일 클래스가 단수 이고 테이블이 복수이다 같은 기본적인 형식을 준수한다면, 그러한 연결을 설정할 필요 없이 보 상을 받게 된다. 클래스는 저장을 위해서 어떤 테이블을 사용할지를 자동으로 알게 된다. 그러한 예제는 수없이 많으며, 모든 것들이 매일 사용함으로써 큰 차이를 만드는데 추가된다.

원칙 – 설정에 대한 형식 (2/2)

□ 세가지 주요 형식

- 프로젝트에 대한 표준 디렉터리 형태
 - 디렉터리 구성을 표준화함으로써 프로젝트 구조의 일관성을 보장함.
- 프로젝트 당 하나의 주요 결과물
 - 관심에 따라서 클라이언트, 서버, 유틸리티 관련 프로젝트로 나눌 수 있음.
 - 복잡성을 관리할 수 있게 하며, 적응성 *adaptability*, 유지보수성 *maintainability*, 확장성 *extendibility*, 재사용성 *reusability*과 같은 품질 요소를 관리할 수 있게 함.
 - 모듈화가 재사용성을 강화시킴.
- 표준적인 명명 *naming* 규칙
 - 프로젝트의 주요 결과물에 대한 표준적인 명명 규칙
 - 규칙은 명확성과 쉬운 이해력을 제공함.
 - commons-loggin-1.2.jar
 - 이름으로 인해서 해당 jar의 정보를 쉽게 알 수 있음.

원칙 – 빌드 로직의 재사용

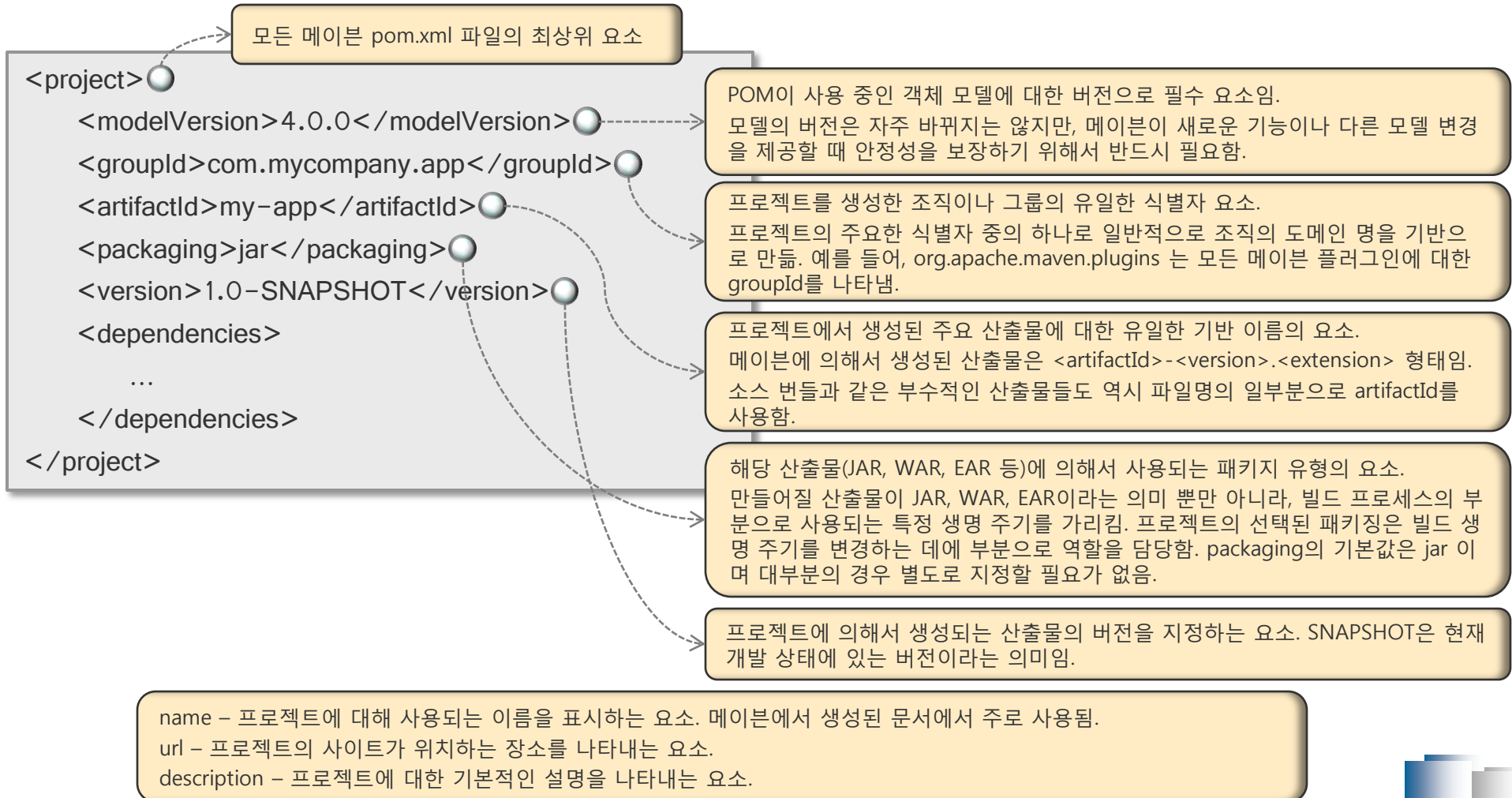
□ 빌드 로직의 캡슐화

- 플러그인이라는 일관된 모듈 사용
 - 잘 정의된 형태로 플러그인 실행을 조정하는 프레임워크
 - 소스 코드를 컴파일하는 플러그인, 테스트를 실행하는 플러그인, JAR를 생성하는 플러그인, Javadoc 문서를 생성하는 플러그인 등
 - 메이븐에서 이루어지는 모든 것은 플러그인의 실행 결과임.
 - 플러그인은 메이븐에 있는 모든 것에 대한 주요한 빌딩 블록임.
 - 플러그인 실행은 **POM** 내의 플러그인 설정에 있는 선언적인 형태로 빌드 생명 주기에 의해서 조정됨.

□ 프로젝트 객체 모델 (**POM** *Project Object Model*)

- 단일 프로젝트에 대한 설명 *description*
 - POM은 메이븐의 통용되는 화폐와도 같음.
 - 메이븐에서 실행을 주도함.
 - 모델 기반 실행
 - XML 문서
 - POM을 통해서 컴파일, 테스트, 기본적인 문서 생성이 가능함.
 - 최상위 *super* POM
 - 메이븐이 실행하는 모든 기본적인 형식을 가짐.
 - 자바의 `java.lang.Object` 클래스에 비유
 - 메이븐의 모든 POM들은 최상위 POM을 내부적으로 부모를 가짐.
 - 중요한 기본적인 정보를 포함하고 있음.
 - 생성된 POM에서 이러한 정보에 대해서 반복할 필요가 없음.

□ 단순한 POM 예제



원칙 – 일관성 있는 dependency 구조 (1/3)

□ POM의 dependency 예제

■ Dependency

- 레파지토리에 위치한 특정 산출물 *artifact*에 대한 참조
 - Dependency를 충족시키기 위해서 어떤 레파지토리를 찾아야 하고 dependency의 관련 된 사항을 알아야 함.
 - groupId, artifactId, version으로 유일하게 식별됨.
- 메이븐의 dependency는 선언적 *declarative*임.
 - POM에서 제공되는 의존관계 정보를 가지고 내부 의존관계 메커니즘에 추가함.
 - 논리적인 의존관계에 초점을 맞춤.
 - 메이븐에서 가장 쉽게 이해되고 강력한 기능 중의 하나
- 가용한 모든 원격 레파지토리에서 검색
 - 일치하는 산출물이 있으면,
원격 레파지토리에서 로컬 레파지토리로 복사

■ 두 가지 유형의 레파지토리

- 로컬, 원격
- 통상 로컬 레파지토리를 검색하고,
선언된 의존관계가 로컬에 없는 경우
접근이 가능한 모든 원격 레파지토리를 검색

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

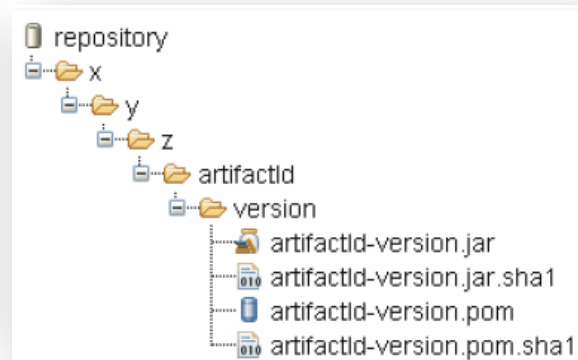
원칙 – 일관성 있는 dependency 구조 (2/3)

□ 로컬 메이븐 레파지토리

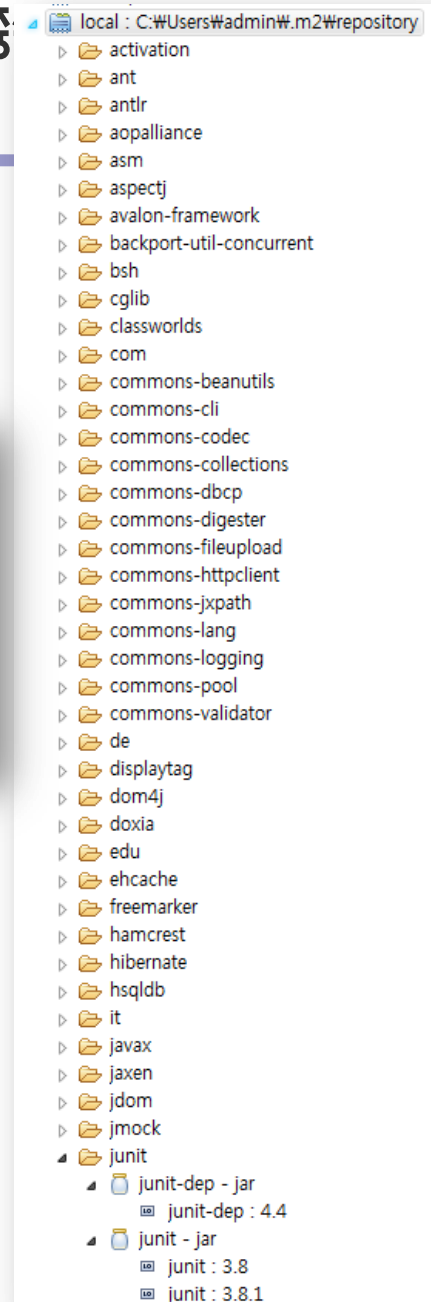
- 위치
 - ~/.m2/repository



일반적인 형태의 레파지토리 구조



groupId가 x.y.z와 같은 형식의
레파지토리 구조



원칙 – 일관성 있는 dependency 구조 (3/3)

□ 의존관계 산출물 위치

- 의존관계의 산출물 검색 절차
 - 로컬 레파지토리에 산출물에 대한 경로를 생성
 - junit의 경우, ~/m2/repository/junit/junit/3.8.1/
 - 해당 파일이 없는 경우, 원격 레파지토리로부터 가지고 옴.
 - 기본적으로 <http://www.ibiblio.org/maven2> 에서 중앙 메이븐 레파지토리로부터 산출물을 가지고 옴.
 - 프로젝트 POM이 하나 이상의 원격 레파지토리를 포함하면 POM에 정의되어 있는 순으로 각각의 원격 레파지토리로부터 산출물을 다운로드 함.
 - 일단 동일한 의존관계를 다른 POM에서 사용하는 경우, 로컬 레파지토리의 동일한 위치에서 산출물을 사용함.

□ 성공적인 기술

- 부담을 강요하는 대신에 부담을 없애줌
 - 유용한 기술은 특정 작업에만 집중하도록 하면서 복잡성을 그 이면에 감추어서 작업하게 함.
- 개발 절차를 간소화시킴
 - 여러 표준들, 레파지토리, 프레임워크, 소프트웨어
 - 프로젝트 관리에 집중된 소프트웨어를 만들고 이는 역동적이고 활동적인 오픈 소스 진영임.
 - **jar** 파일이나 스크립트들을 단순히 다운받는 것을 넘어서 다음 단계로 소프트웨어 개발을 적용하게끔 프로세스를 적용함.

메이븐 시작하기

- ❑ 메이븐 사용 준비
- ❑ 첫 번째 프로젝트 생성
- ❑ 어플리케이션 소스 컴파일
- ❑ 테스트 소스 컴파일과 단위 테스트 실행
- ❑ 로컬 레파지토리에 패키징 및 설치
- ❑ 클래스패스 리소스 처리
- ❑ 메이븐 플러그인 사용

□ 메이븐 설치 및 설정

- <http://maven.apache.org/download.html>
- 설정 파일
 - ~/.m2/settings.xml
- 실행
 - mvn -version

```
<settings>
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.mycompany.com</host>
      <port>8080</port>
      <username>your-username</username>
      <password>your-password</password>
    </proxy>
  </proxies>
</settings>
```

방화벽이 있는 경우 proxy를 통한
메이븐 원격 레파지토리 접근

```
<settings>
  <mirrors>
    <mirror>
      <id>maven.mycompany.com</id>
      <name>My Company's Maven Proxy</name>
      <url>http://maven.mycompany.com/maven2</url>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>
</settings>
```

내부에 메이븐 proxy가 있는 경우

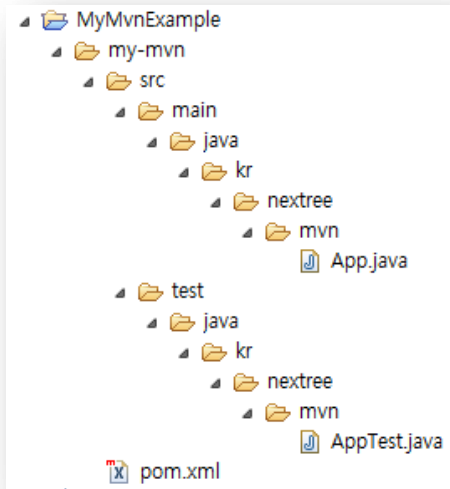
□ 메이븐 archetype

- Archetype은 동일한 종류의 모든 다른 것들이 만들어지는 원래의 패턴 혹은 모델을 의미함.
 - 프로젝트의 템플릿
- Archetype을 사용한 메이븐 프로젝트 생성

```
D:\Works\temp\maven_example>mvn archetype:create -DgroupId=kr.nexttree.mvn -DartifactId=my-mvn
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: archetype
[INFO] artifact org.apache.maven.plugins:maven-archetype-plugin: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-archetype-plugin/2.0-alpha-2/maven-archetype-plugin-2.0-alpha-2.pom
3K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/archetype/maven-archetype/2.0-alpha-2/maven-archetype-2.0-alpha-2.pom
16K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-archetype-plugin/2.0-alpha-2/maven-archetype-plugin-2.0-alpha-2.jar
64K downloaded
[INFO] -----
[INFO] Building Maven Default Project
[INFO] task-segment: [archetype:create] (aggregator-style)
[INFO] -----
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-core/2.0.8/maven-core-2.0.8.pom
6K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven/2.0.8/maven-2.0.8.pom
11K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-settings/2.0.8/maven-settings-2.0.8.pom
2K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-model/2.0.8/maven-model-2.0.8.pom
3K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/wagon/wagon-http/2.0.8/wagon-http-2.0.8.pom
637b downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/maven-plugin-reporting/2.0.8/maven-plugin-reporting-2.0.8.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/wagon/wagon-http/2.0.8/wagon-http-2.0.8.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/kr/nexttree/mvn/wagon-http-sha1/2.0.8/wagon-http-sha1-2.0.8.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/reporting/maven-plugin-reporting/2.0.8/maven-plugin-reporting-2.0.8.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/reporting/maven-plugin-reporting/2.0.8/maven-plugin-reporting-2.0.8.jar
1K downloaded
Downloading: http://repo1.maven.org/maven2/net/sourceforge/jchardet/jchardet/1.0/jchardet-1.0.jar
25K downloaded
[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus.velocity.ContextClassLoaderResourceLoader'.
[INFO] Setting property: velocimacro.messages.on => 'false'.
[INFO] Setting property: resource.loader => 'classpath'.
[INFO] Setting property: resource.manager.logwhenfound => 'false'.
[INFO] [archetype:create]
[INFO] Defaulting package to group ID: kr.nexttree.mvn
[INFO] artifact org.apache.maven.archetypes:maven-archetype-quickstart: checking for updates from central
[INFO] -----
[INFO] Using following parameters for creating OldArchetype: maven-archetype-quickstart:RELEASE
[INFO] -----
[INFO] Parameter: groupId, Value: kr.nexttree.mvn
[INFO] Parameter: packageName, Value: kr.nexttree.mvn
[INFO] Parameter: package, Value: kr.nexttree.mvn
[INFO] Parameter: artifactId, Value: my-mvn
[INFO] Parameter: basedir, Value: D:\Works\temp\maven_example
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] ***** End of debug info from resources from generated POM *****
[INFO] OldArchetype created in dir: D:\Works\temp\maven_example\my-mvn
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 54 seconds
[INFO] Finished at: Wed Mar 26 20:12:40 KST 2008
[INFO] Final Memory: 7M/13M
[INFO] -----
```

첫 번째 프로젝트 생성 (2/2)

□ Archetype 생성 결과



```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>kr.nexttree.mvn</groupId>
  <artifactId>my-mvn</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>my-mvn</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
  
```


□ 네 가지 기본 원칙

- 메이븐의 모든 부분에 원칙들이 스며들어 있음.

```
D:\Works\temp\maven_example\my-mvn>mvn compile
[INFO] Scanning for projects...
[INFO] Building my-mvn
[INFO] task-segment: [compile]
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Compiling 1 source file to D:\Works\temp\maven_example\my-mvn\target\classes
[INFO] BUILD SUCCESSFUL
[INFO] Total time: 5 seconds
[INFO] Finished at: Wed Mar 26 20:40:03 KST 2008
[INFO] Final Memory: 2M/7M
```

“설정에 대한 형식” 원칙

기본적으로 메이븐 소스는 /src/main/java에 위치함. 이 기본값은 최상위 POM으로부터 상속받음. POM에 소스의 위치나 컴파일 결과에 대한 위치(target/classes)를 별도로 지정할 필요가 없음.

“빌드 로직 재사용” 원칙

표준 컴파일러 플러그인은 어플리케이션 소스를 컴파일하는데 사용되는 도구임. 컴파일러 플러그인에 캡슐화된 동일한 빌드 로직은 수많은 프로젝트에서 일관적으로 수행될 것임.

메이븐을 처음 설치한 경우, 컴파일러 플러그인은 탑재되지 않음. 명령어를 처음 실행하면, 메이븐은 모든 플러그인과 해당 명령어를 수행하는데 필요한 관련된 의존관계를 다운로드 받음.

이 후에 동일한 명령어를 다시 실행 시 이미 다운로드 받은 플러그인으로 실행하여 좀 더 빨리 실행됨.

메이븐의 표준 형식을 사용하면 POM에 기술되는 내용이 매우 적어지며, 매우 적은 노력으로 많은 일을 수행할 수 있음.

테스트 소스 컴파일과 단위 테스트 실행

□ 단위 테스트 실행

```
D:\Works\temp\maven_example\my-mvn>mvn test
[INFO] Scanning for projects...
[INFO] Building my-mvn
[INFO] task-segment: [test]
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Compiling 1 source file to D:\Works\temp\maven_example\my-mvn\target\test-classes
[INFO] [surefire:test]
Downloading: http://repo1.maven.org/maven2/org/apache/maven/surefire/surefire-junit/2.4.2/surefire-junit-2.4.2.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/surefire/surefire-junit/2.4.2/surefire-junit-2.4.2.jar
14K downloaded
[INFO] Surefire report directory: D:\Works\temp\maven_example\my-mvn\target\surefire-reports

-----
T E S T S
-----
Running kr.nexttree.mvn.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.252 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 10 seconds
[INFO] Finished at: Wed Mar 26 21:26:32 KST 2008
[INFO] Final Memory: 4M/11M
[INFO] -----
```

메이븐은 테스트를 실행하는데 필요한 의존관계들과 플러그인들을 다운로드 받음.
테스트를 컴파일하고 실행하기 이전에 메이븐은 메인 소스를 컴파일함.

로컬 레파지토리에 패키징 및 설치 (1/3)

□ 패키징과 설치

```
D:\Works\temp\maven_example\my-mvn>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] Building my-mvn
[INFO] task-segment: [package]
[INFO]
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory: D:\Works\temp\maven_example\my-mvn\target\surefire-reports

T E S T S

Running kr.nexttree.mvn.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.103 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] Building jar: D:\Works\temp\maven_example\my-mvn\target\my-mvn-1.0-SNAPSHOT.jar
[INFO] BUILD SUCCESSFUL
[INFO] Total time: 6 seconds
[INFO] Finished at: Wed Mar 26 21:37:09 KST 2008
[INFO] Final Memory: 6M/11M
[INFO] -----
```

packaging 요소가 jar 로 세팅되었기 때문에 JAR파일로 묶여짐.
~/.m2/repository 밑에 jar 파일이 만들어짐.

```
D:\Works\temp\maven_example\my-mvn>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] Building my-mvn
[INFO] task-segment: [install]
[INFO]
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory: D:\Works\temp\maven_example\my-mvn\target\surefire-reports

T E S T S

Running kr.nexttree.mvn.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.103 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] Building jar: D:\Works\temp\maven_example\my-mvn\target\my-mvn-1.0-SNAPSHOT.jar
[INFO] [install:install]
[INFO] Installing D:\Works\temp\maven_example\my-mvn\target\my-mvn-1.0-SNAPSHOT.jar to C:\Users\admin\m2\repository\kr\nexttree\my-mvn\1.0-SNAPSHOT\my-mvn-1.0-SNAPSHOT.jar
[INFO] BUILD SUCCESSFUL
[INFO] Total time: 7 seconds
[INFO] Finished at: Wed Mar 26 21:39:22 KST 2008
[INFO] Final Memory: 6M/12M
[INFO] -----
```

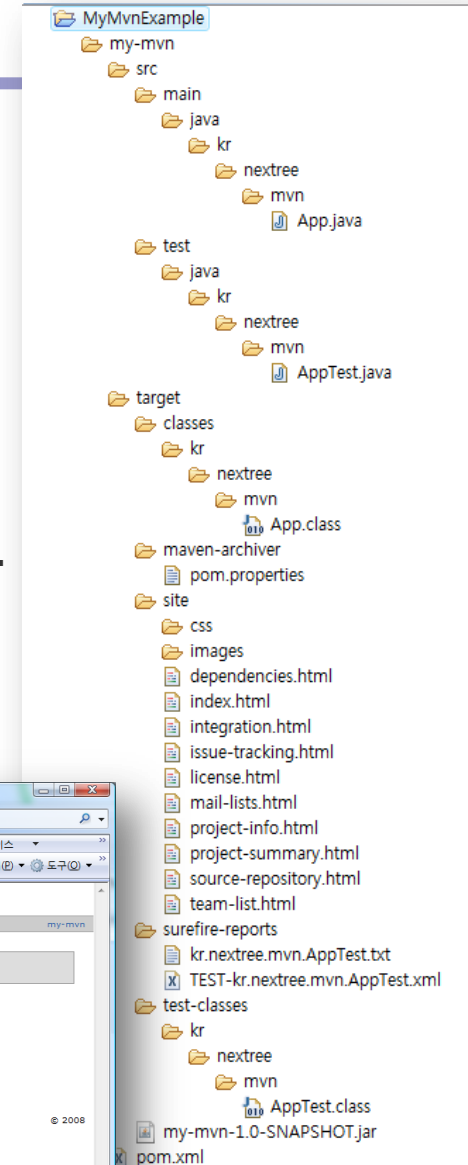
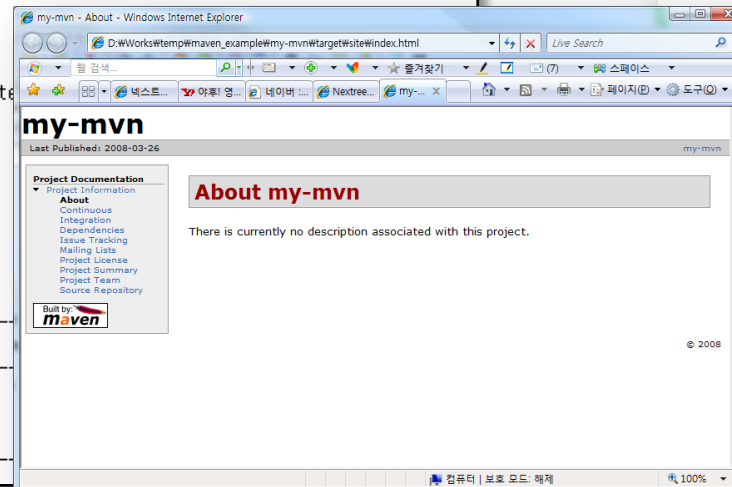
□ Surefire 플러그인

- 포함되는 파일
 - `**/*Test.java`, `**/Test*.java`, `**/*TestCase.java`
- 제외되는 파일
 - `**/Abstract*Test.java`, `**/Abstract*TestCase.java`
- 가장 단순한 POM
 - 수많은 메이븐 플러그인을 통해서 여러 가지 일을 수행함.

```

D:\Works\temp\maven_example\my-mvn>mvn site
[INFO] Scanning for projects...
[INFO]
[INFO] Building my-mvn
[INFO] task-segment: [site]
[INFO]
[INFO] Setting property: classpath.resource.loader.class => 'org.codehaus.plexus.velocity.ContextClassLoaderResourceLoader'.
[INFO] Setting property: velocimacro.messages.on => 'false'.
[INFO] Setting property: resource.loader => 'classpath'.
[INFO] Setting property: resource.manager.logwhenfound => 'false'.
[INFO] [site:site]
[INFO] artifact org.apache.maven.skins:maven-default-skin: checking for updates from central
[INFO] Generating "Continuous Integration" report.
[INFO] Generating "Dependencies" report.
[INFO] Generating "Issue Tracking" report.
[INFO] Generating "Project License" report.
[INFO] Generating "Mailing Lists" report.
[INFO] Generating "About" report.
[INFO] Generating "Project Summary" report.
[INFO] Generating "Source Repository" report.
[INFO] Generating "Project Team" report.
[INFO]
[INFO] BUILD SUCCESSFUL
[INFO]
[INFO] Total time: 16 seconds
[INFO] Finished at: Wed Mar 26 21:51:04 KST 2008
[INFO] Final Memory: 13M/26M
[INFO]
    
```

프로젝트 문서 생성



로컬 레파지토리에 패키징 및 설치 (3/3)

□ 기타

```
D:\Works\temp\maven_example\my-mvn>mvn clean
[INFO] Scanning for projects...
[INFO] Building my-mvn
[INFO] task-segment: [clean]
[INFO] [clean:clean]
[INFO] Deleting directory D:\Works\temp\maven_example\my-mvn\target
[INFO] BUILD SUCCESSFUL
[INFO] Total time: 2 seconds
[INFO] Finished at: Wed Mar 26 22:05:10 KST 2008
[INFO] Final Memory: 2M/6M
```

시작 이전의 기존 빌드 데이터와 같이 target 디렉터리 제거

```
D:\Works\temp\maven_example\my-mvn>mvn idea:idea
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'idea'.
[INFO] artifact org.apache.maven.plugins:maven-idea-plugin: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-idea-plugin/2.1/maven-idea-plugin-2.1.pom
2K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-idea-plugin/2.1/maven-idea-plugin-2.1.jar
40K downloaded
[INFO] Building my-mvn
[INFO] task-segment: [idea:idea]
[INFO] [idea:idea]
[INFO] Preparing idea:idea
[INFO] No goals needed for project - skipping
[INFO] [idea:idea]
[INFO] jdkName is not set, using [java version1.6.0_03] as default.
[INFO] BUILD SUCCESSFUL
[INFO] Total time: 7 seconds
[INFO] Finished at: Wed Mar 26 22:07:47 KST 2008
[INFO] Final Memory: 2M/6M
```

IntelliJ 의 IDEA 에 대한 프로젝트 생성

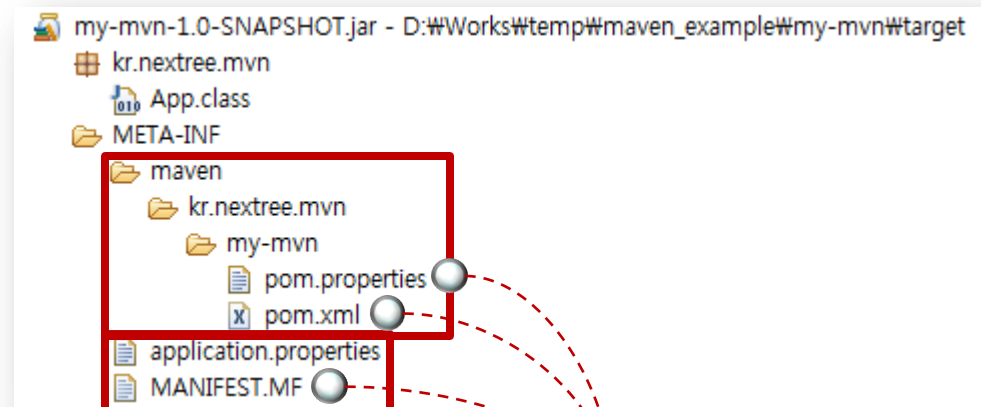
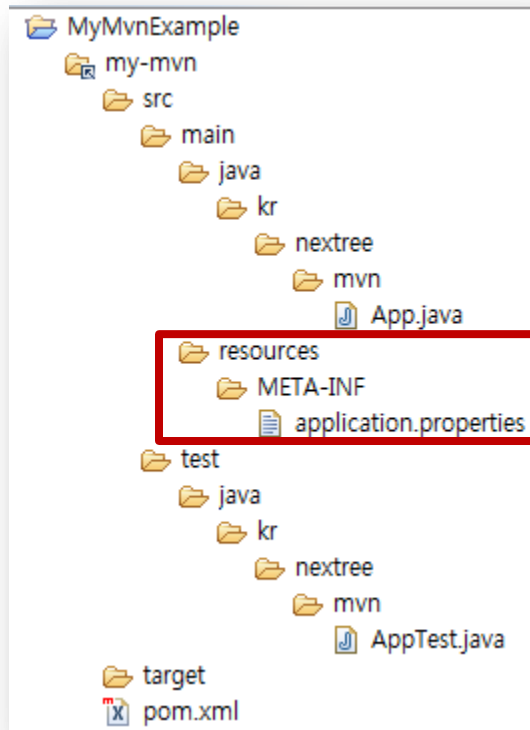
```
D:\Works\temp\maven_example\my-mvn>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'eclipse'.
[INFO] artifact org.apache.maven.plugins:maven-eclipse-plugin: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-eclipse-plugin/2.5/maven-eclipse-plugin-2.5.pom
6K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-eclipse-plugin/2.5/maven-eclipse-plugin-2.5.jar
177K downloaded
[INFO] -----
[INFO] Building my-mvn
[INFO] task-segment: [eclipse:eclipse]
[INFO] -----
[INFO] Preparing eclipse:eclipse
[INFO] No goals needed for project - skipping
Downloading: http://repo1.maven.org/maven2/org/apache/maven/shared/maven-oss/0.2.0/maven-oss-0.2.0.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/biz/aQute/bndlib/0.0.203/bndlib-0.0.203.pom
886b downloaded
Downloading: http://repo1.maven.org/maven2/biz/aQute/bndlib/0.0.145/bndlib-0.0.145.pom
886b downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-interactivity-jline/1.0-alpha-5/plexus-interactivity-jline-1.0-alpha-5.pom
772b downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-interactivity/1.0-alpha-5/plexus-interactivity-1.0-alpha-5.pom
482b downloaded
Downloading: http://repo1.maven.org/maven2/jline/jline/0.9.1/jline-0.9.1.pom
145b downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-interactivity-api/1.0-alpha-5/plexus-interactivity-api-1.0-alpha-5.pom
430b downloaded
Downloading: http://repo1.maven.org/maven2/org/eclipse/core/resources/3.3.0-v20070604/resources-3.3.0-v20070604.pom
1K downloaded
Downloading: http://repo1.maven.org/maven2/org/codehaus/plexus/plexus-interactivity-jline/1.0-alpha-5/plexus-interactivity-jline-1.0-alpha-5.jar
5K downloaded
Downloading: http://repo1.maven.org/maven2/jline/jline/0.9.1/jline-0.9.1.jar
45K downloaded
Downloading: http://repo1.maven.org/maven2/biz/aQute/bndlib/0.0.145/bndlib-0.0.145.jar
111K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/shared/maven-oss/0.2.0/maven-oss-0.2.0.jar
12K downloaded
Downloading: http://repo1.maven.org/maven2/org/eclipse/core/resources/3.3.0-v20070604/resources-3.3.0-v20070604.jar
662K downloaded
[INFO] [eclipse:eclipse]
[INFO] Using as WTP server : null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Using source status cache: D:\Works\temp\maven_example\my-mvn\target\my-mvn-eclipse-cache.properties
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "my-mvn" to D:\Works\temp\maven_example\my-mvn.
[INFO] -----
Sources for some artifacts are not available.
Please run the same goal with the -DdownloadSources=true parameter in order to check remote repositories for sources.
List of artifacts without a source archive:
o junit:junit:3.8.1

Javadoc for some artifacts is not available.
Please run the same goal with the -DdownloadJavadocs=true parameter in order to check remote repositories for javadoc.
List of artifacts without a javadoc archive:
o junit:junit:3.8.1

[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 28 seconds
[INFO] Finished at: Wed Mar 26 22:09:37 KST 2008
[INFO] Final Memory: 5M/11M
```

Eclipse 에 대한 프로젝트 생성

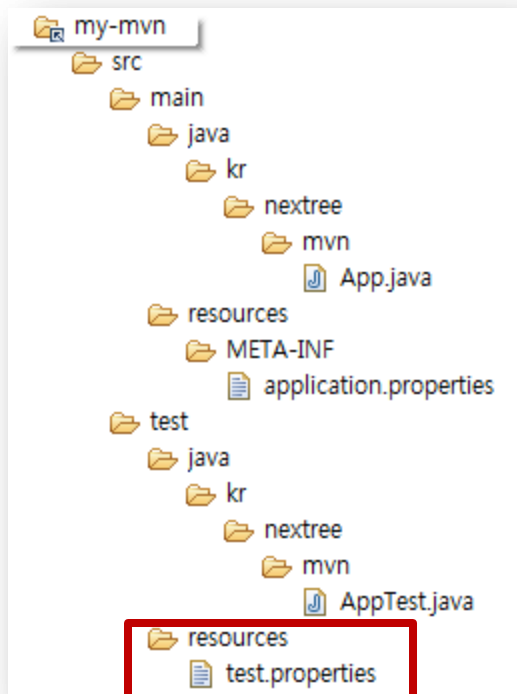
- **JAR** 파일에 리소스 패키징
 - 표준 디렉터리 구조 사용
 - src/main/resources



메이븐에서 JAR 생성시 표준적으로 생성됨.
MANIFEST.MF 파일은 없는 경우 생성.

□ 테스트 클래스패스 리소스 처리

- 단위 테스트 클래스패스에 리소스 추가
 - src/test/resources



단위 테스트에서 테스트 시 필요한 리소스 접근

[...]

// 리소스 추출

```
InputStream is = getClass().getResourceAsStream("/test.properties");
```

// 리소스를 사용

[...]

□ 클래스패스 리소스 필터링

- 빌드 시점에만 필요한 값을 가지는 리소스
 - \${<property_name>} 과 같은 형태의 값 세팅
 - property는 pom.xml 이나, settings.xml 에 정의된 값
 - 외부 properties 파일이나 system property에 정의된 값

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
...
<dependencies>
...
</dependencies>
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
</project>
```

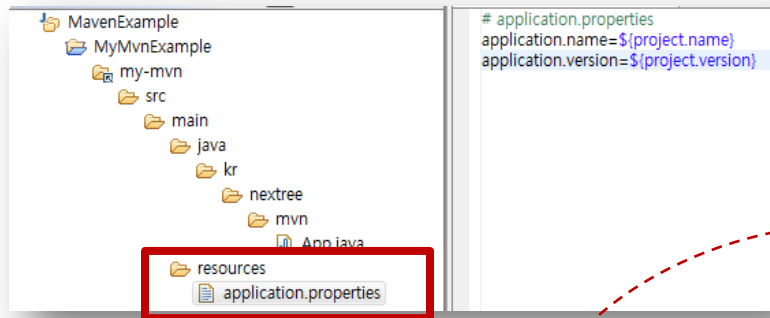
filtering 값을 true로 오버라이딩 함.
default 로는 false 로 처리됨.

pom.xml 파일

□ 클래스패스 리소스 필터링

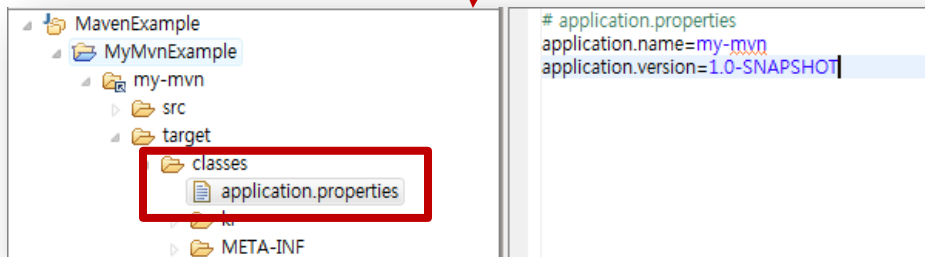
■ 메이븐 내부 변수 사용

- `${project.name}`, `${project.version}`, `${project.build.finalName}`



mvn process-resources 수행

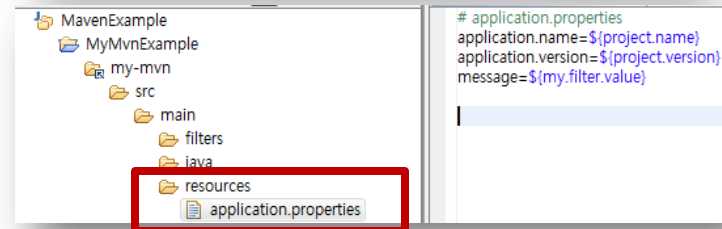
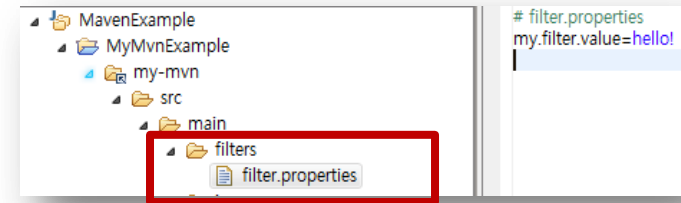
```
[INFO] Scanning for projects...
[INFO] Attempting to resolve a version for plugin: org.apache.maven.plugins:maven-resources-plugin using meta-version: LATEST
[INFO] Using version: 2.2 of plugin: org.apache.maven.plugins:maven-resources-plugin
[INFO] Using default encoding to copy filtered resources.
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 1 second
[INFO] Finished at: Sat Mar 29 15:36:20 KST 2008
[INFO] Final Memory: 2M/4M
[INFO] -----
```



□ 클래스패스 리소스 필터링

- 필터링 파일 지정
 - src/main/filters/filter.properties

```
...
<build>
  <filters>
    <filter>src/main/filters/filter.properties</filter>
  </filters>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
</project>
```



```
...
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
</build>
<properties>
  <my.filter.value>hello</my.filter.value>
</properties>
</project>
```

Application.name=my-mvn
Application.version=1.0-SNAPSHOT
Message=hello!

클래스패스 리소스 처리 (6/7)

□ 클래스패스 리소스 필터링

▪ 기타 변수 사용하기

- `${java.version}`
- `${command.line.prop}`

➤ `mvn process-resources "-Dcommand.line.prop=hello again"`

```
# application.properties
java.version=${java.version}
command.line.prop=${command.line.prop}
```

□ 바이너리 리소스 필터링 방지

- 이미지 파일
 - src/main/resources/images

```

...
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
      <excludes>
        <exclude>images/**</exclude>
      </excludes>
    </resource>
    <resource>
      <directory>src/main/resources</directory>
      <includes>
        <include>images/**</include>
      </includes>
    </resource>
  </resources>
</build>
</project>
    
```

filtering 시에만 images 폴더의 파일들은 제외됨.

메이븐 플러그인 사용 (1/2)

□ 기본적인 플러그인 기능을 수정하거나 추가하는 경우

■ 플러그인 사용

```

...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.0</version>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>

```

groupId 가 지정되어 있지 않으면, 메이븐은 org.apache.maven.plugins 나 org.codehaus.mojo.groupId 로 검색함.
POM이나 settings.xml 에 groupId 를 지정할 수 있음.

version을 지정하지 않으면 지정된 플러그인의 가장 최근에 배포 버전을 사용하려 함.

기존 플러그인의 속성 중에 JDK 버전 정보를 바꿔줌.

configuration 은 모든 goal의 주어진 파라미터에 적용됨.

메이븐 플러그인 사용 (2/2)

□ 플러그인의 **configuration** 옵션 확인

▪ mvn help:describe 사용

```
D:\Works\study\강의자료\통합환경>mvn help:describe -DgroupId=org.apache.maven.plugins -DartifactId=maven-compiler-plugin -Dfull=true
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'help'.
[INFO] artifact org.apache.maven.plugins:maven-help-plugin: checking for updates from central
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-help-plugin/2.0.2/maven-help-plugin-2.0.2.pom
3K downloaded
Downloading: http://repo1.maven.org/maven2/org/apache/maven/plugins/maven-help-plugin/2.0.2/maven-help-plugin-2.0.2.jar
23K downloaded
[INFO]
[INFO] Building Maven Default Project
[INFO] task-segment: [help:describe] (aggregator-style)
[INFO]
[INFO] [help:describe]
[INFO] Plugin: 'org.apache.maven.plugins:maven-compiler-plugin:2.0.2'
-----
Group Id: org.apache.maven.plugins
Artifact Id: maven-compiler-plugin
Version: 2.0.2
Goal Prefix: compiler
Description:

Maven Plugins

Mojos:
-----
Goal: 'compile'
-----
Description:
Compiles application sources

Implementation: org.apache.maven.plugin.CompilerMojo
Language: java
Bound to Phase: compile

Parameters:
-----
[0] Name: basedir
Type: java.io.File
Required: true
Directly editable: false
Description:
The directory to run the compiler from if fork is true.
-----
```

Maven을 이용한 Junit 테스트

□ Maven으로 Junit 실행하기

- 메이븐은 기본적으로 JUnit 테스트를 지원
- Maven에 test 디렉토리에 TestCase, TestSuite를 작성해 넣기만 하면 됨(CoC)
- 테스트 작성 후 mvn test 실행

```

C:\> 선택 명령 프롬프트 - mvn test
[INFO] Surefire report directory: D:\work\NEC\NECMAA\target\surefire-reports

-----
T E S T S
-----
Running kr.go.nec.mutual.service.CodeServiceTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec
Running kr.go.nec.mutual.service.ZipCodeServiceTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.579 sec
Running kr.go.nec.mutual.domain.PayTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec
Running kr.go.nec.mutual.service.FinanceCalculatorServiceTest
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec
Running kr.go.nec.mutual.service.LoanOnceServiceTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.344 sec
Running kr.go.nec.mutual.service.calculator.roundsum.GoalSavingCalculatorTest
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathCon
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathCon
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathCon
ERROR [main] <GoalSavingCalculatorTest.java:56> - 8146673
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathCon
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathCon
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathCon
ERROR [main] <GoalSavingCalculatorTest.java:74> - 2273722
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015 sec
Running kr.go.nec.mutual.util.PageCriteriaTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec
Running kr.go.nec.mutual.util.DateUtilTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.016 sec
Running kr.go.nec.mutual.service.ExcelPrintServiceTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.875 sec
Running kr.go.nec.mutual.service.RateServiceTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.328 sec
Running kr.go.nec.mutual.service.SecedeIntegrationServiceTest
data = <investPreAmount : '50100', investPreInterest : '745', investCurAmount : '200000', investCurInterest
Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.828 sec
Running kr.go.nec.mutual.util.NecDateTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec
Running kr.go.nec.mutual.service.UserServiceTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.625 sec
Running kr.go.nec.mutual.service.InvestStatisticsServiceTest
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.969 sec
Running kr.go.nec.mutual.service.ACAccountServiceTest

```

Eclipse에서 테스트 실행하기

- ❑ Eclipse 소개
- ❑ Eclipse에서 Junit 실행하기

□ 통합 개발 환경(IDE)

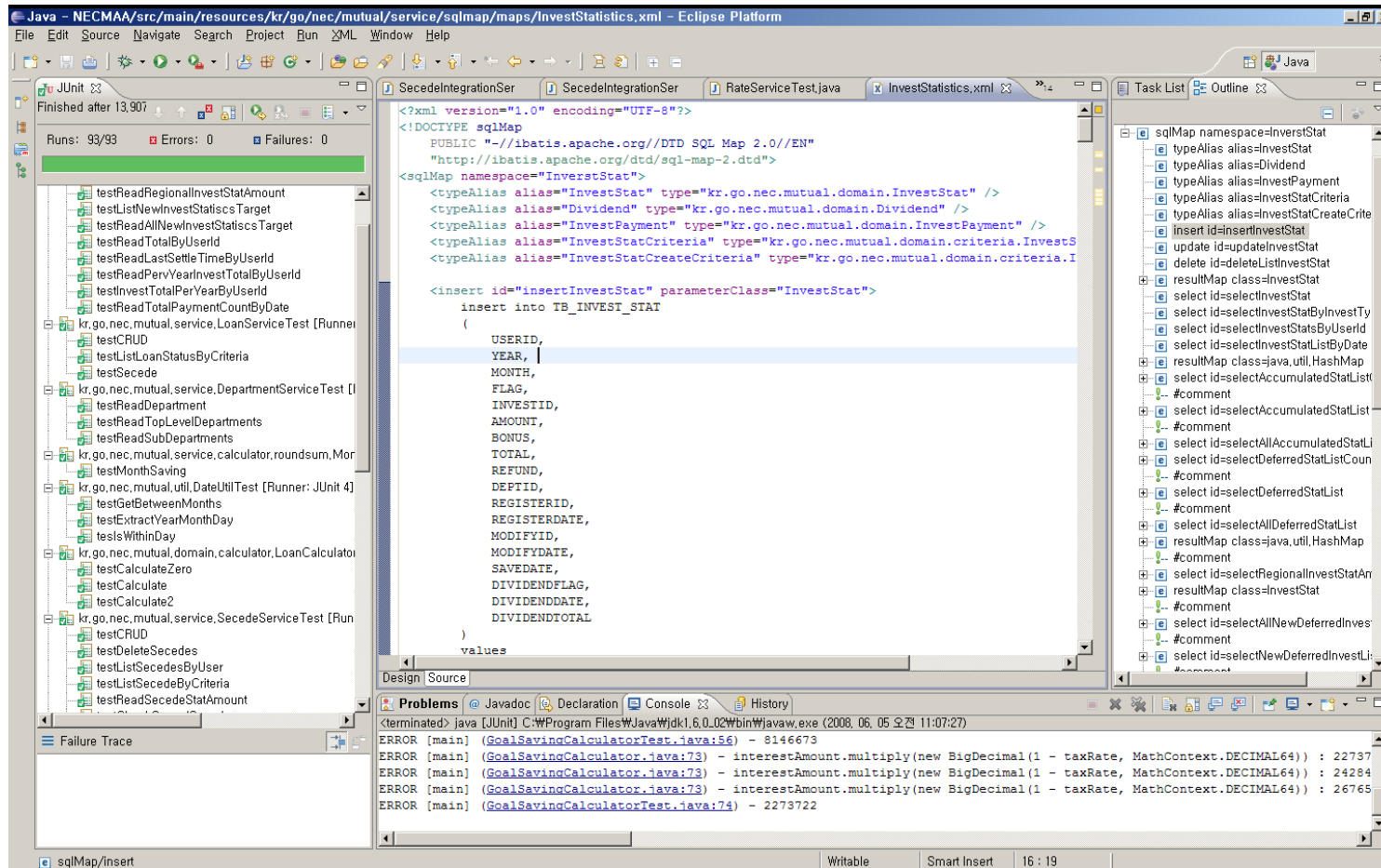
- 현재 가장 널리 사용되고 있는 통합 개발 환경
- 다양한 개발 도구를 통합할 수 있는 개발 환경 플랫폼
- Java이외의 다양한 언어로 개발할 수 있는 언어 중립적인 개발 환경

□ De-Facto 개발 도구

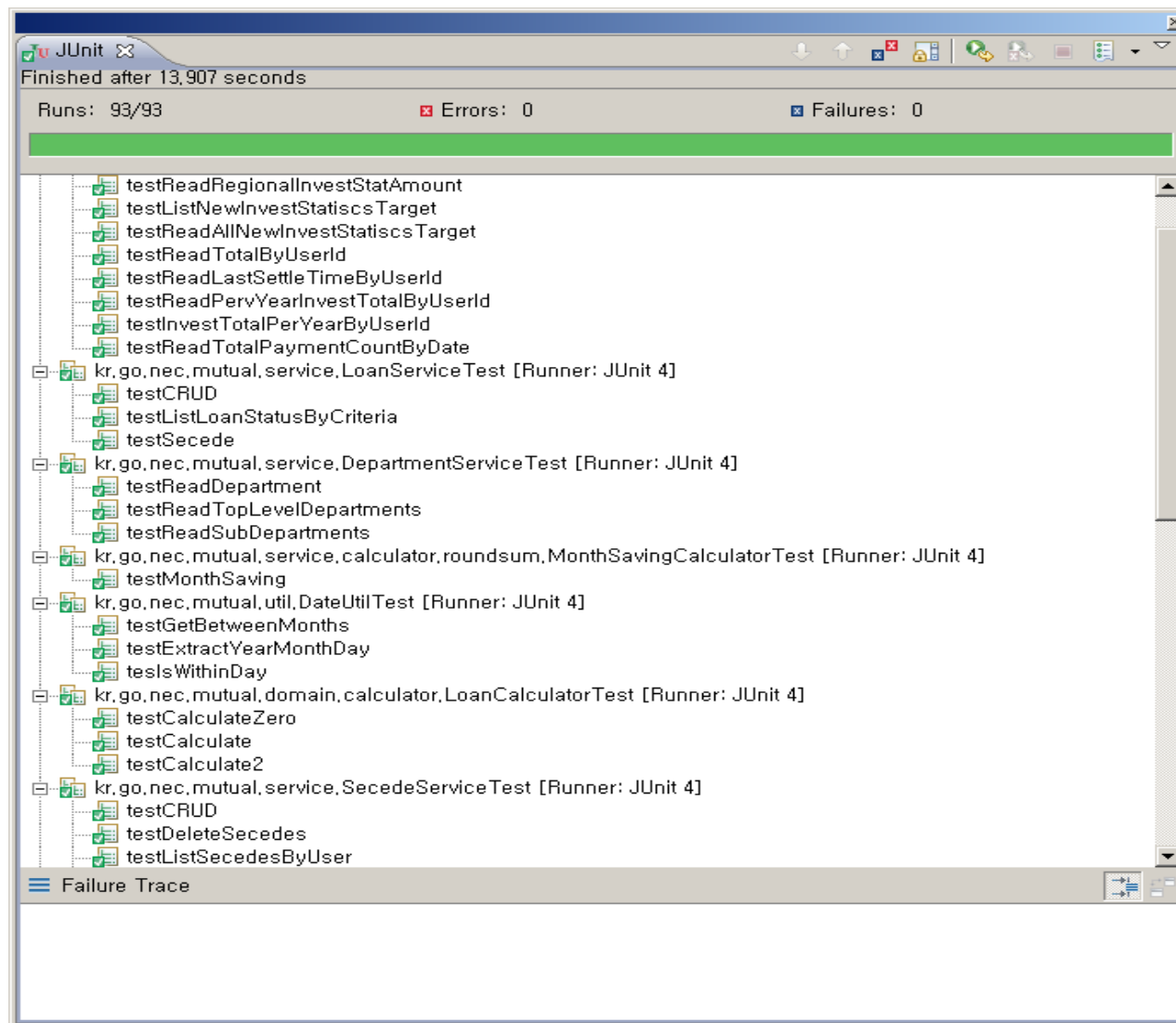
- 사실상의 자바 개발환경 표준임
- Eclipse에서 사용할 수 있는 수많은 플러그인 존재
- junit도 Eclipse 플러그인을 제공함

Eclipse에서 JUnit 실행하기

- ❑ Eclipse에 JUnit 플러그인이 기본 탑재되어 있음
- ❑ TestCase가 모여있는 디렉토리에서 JUnit Test를 실행하면 OK



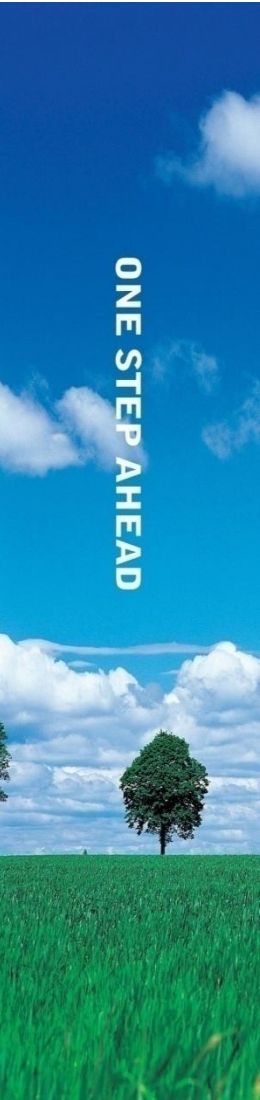
Eclipse에서 JUnit 실행하기



3. JUnit 테스트 전략

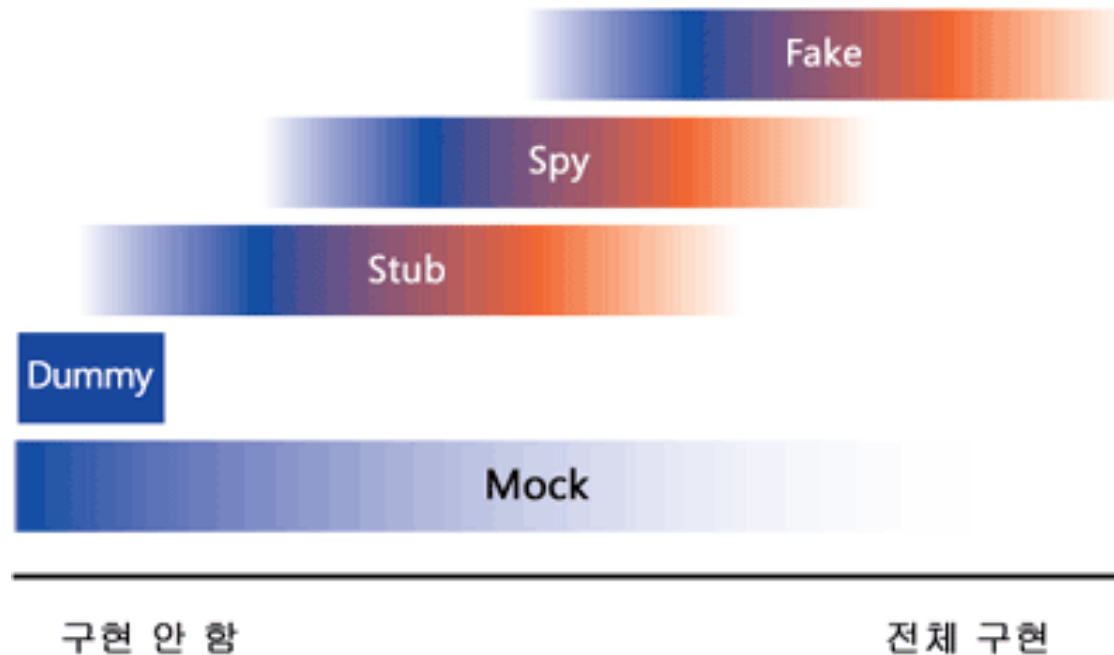
- ❑ **Test Double**
- ❑ **Stub** 테스트
- ❑ 모의객체(**Mock Object**) 활용한 테스트
- ❑ **Cactus**를 이용한 인-컨테이너 테스트
- ❑ 단위 테스트 품질 향상을 위한 툴 소개
 - 코드 커버리지 분석기 - Coverlipse
 - 패키지 의존성 분석기 - JDepend
 - 코딩 표준 분석기 - CheckStyle

ONE STEP AHEAD



□ Test Double

- Gerard Meszaros의 테스트 분류 기준
- 대용 객체를 Dummy, Stub, Mock, Fake 같은 모두 다른 이름으로 부름
- 스텐트맨이라 하는 대역을 stunt double, 실제 객체의 대신은 Test Double



❑ Dummy

- 객체의 전달에만 사용되고 실제로 이를 사용하지 않는 것이다. 대개 매개변수 목록을 채우는데 쓰임

❑ Fake

- 객체는 실제로 동작하도록 구현되지만, 보통 빠른 구현을 위해 실제환경과는 조금 다르게 구현
- 실제 사용할 RDBMS 대신 Memory DB를 사용하는 것이 적절한 예

❑ Stub

- 테스트를 위해 미리 준비한 응답만을 제공하는 것
- 그 외의 상황에 대해서는 정상적으로 작동하지 못하는 것이 일반적

❑ Mock

- 스펙을 통해 정의된 응답을 받게 되어 있고, 다른 응답을 받게 되면 예외를 발생하도록 구현되어서, 응답에 대한 확인 기능을 수행하는 역할

- ❑ **Stub** 소개
- ❑ **HTTP Connection** 예제
- ❑ **Stub** 의 선택
- ❑ 웹 리소스 스터빙하기
- ❑ **HTTP Connection** 스터빙하기

□ Stub 이란?

- 실제 코드의 행위를 흉내내기 위한 매커니즘
- 아직 가용하지 않은 코드를 제외한 부분을 테스트할 수 있도록 해줌
- 실제 코드의 일부를 독립적으로 테스트할 수 있게 해줌

□ 언제 **Stub**을 사용하나?

- 코드가 너무 복잡하거나 변경에 취약하여 변경을 가할 수 없을 때
- 다른 두 시스템의 통합 테스트같이 복잡도가 높은 테스트를 할 때

□ 단점

- 만들기 대체로 어려움, 스텝을 디버깅해야 하는 경우도 있음
- 상황마다 다른 전략이 필요

HTTP Connection 예제

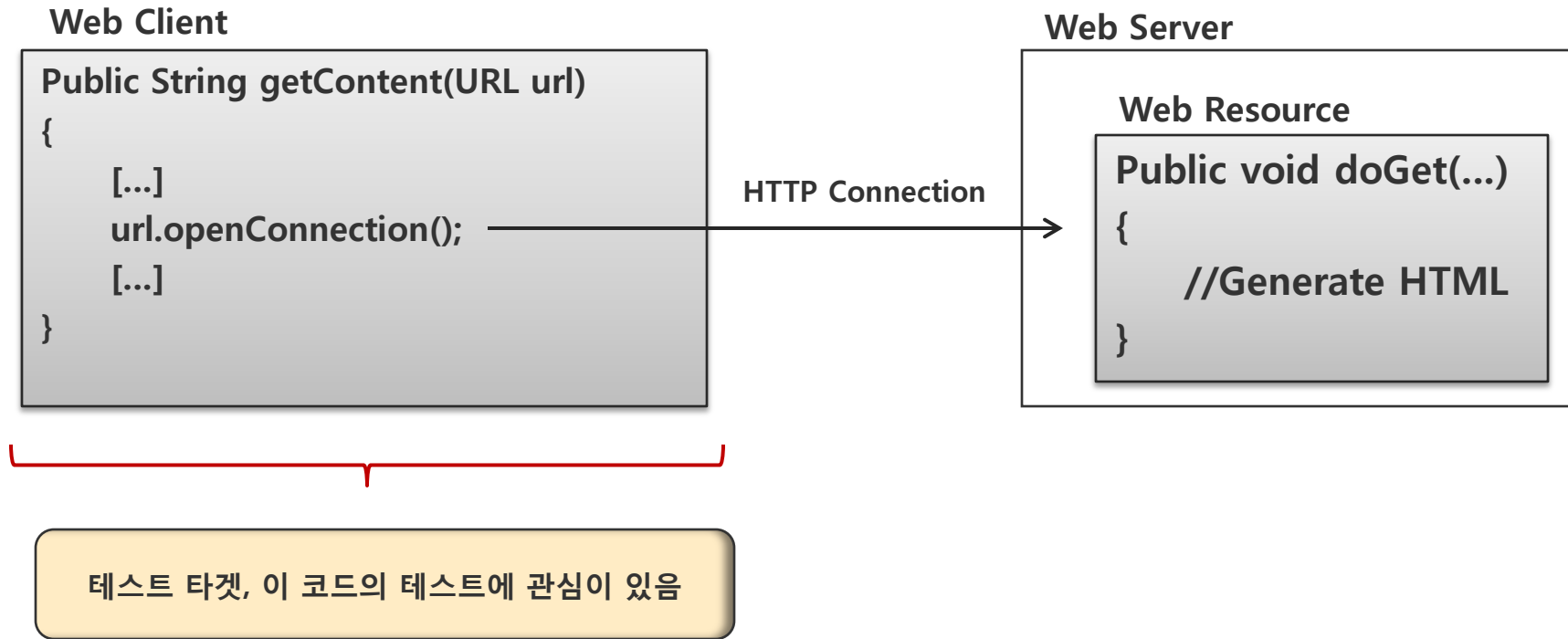
□ 테스트할 코드 조각

```
public class WebClient
{
    public String getContent(URL url)    {
        StringBuffer content = new StringBuffer();
        try {
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setDoInput(true);
            InputStream is = connection.getInputStream();
            byte[] buffer = new byte[2048];
            int count;
            while (-1 != (count = is.read(buffer))) {
                content.append(new String(buffer, 0, count));
            }
        }
        catch (IOException e) { return null; }
        return content.toString();
    }
}
```

HTTP Connection 예제

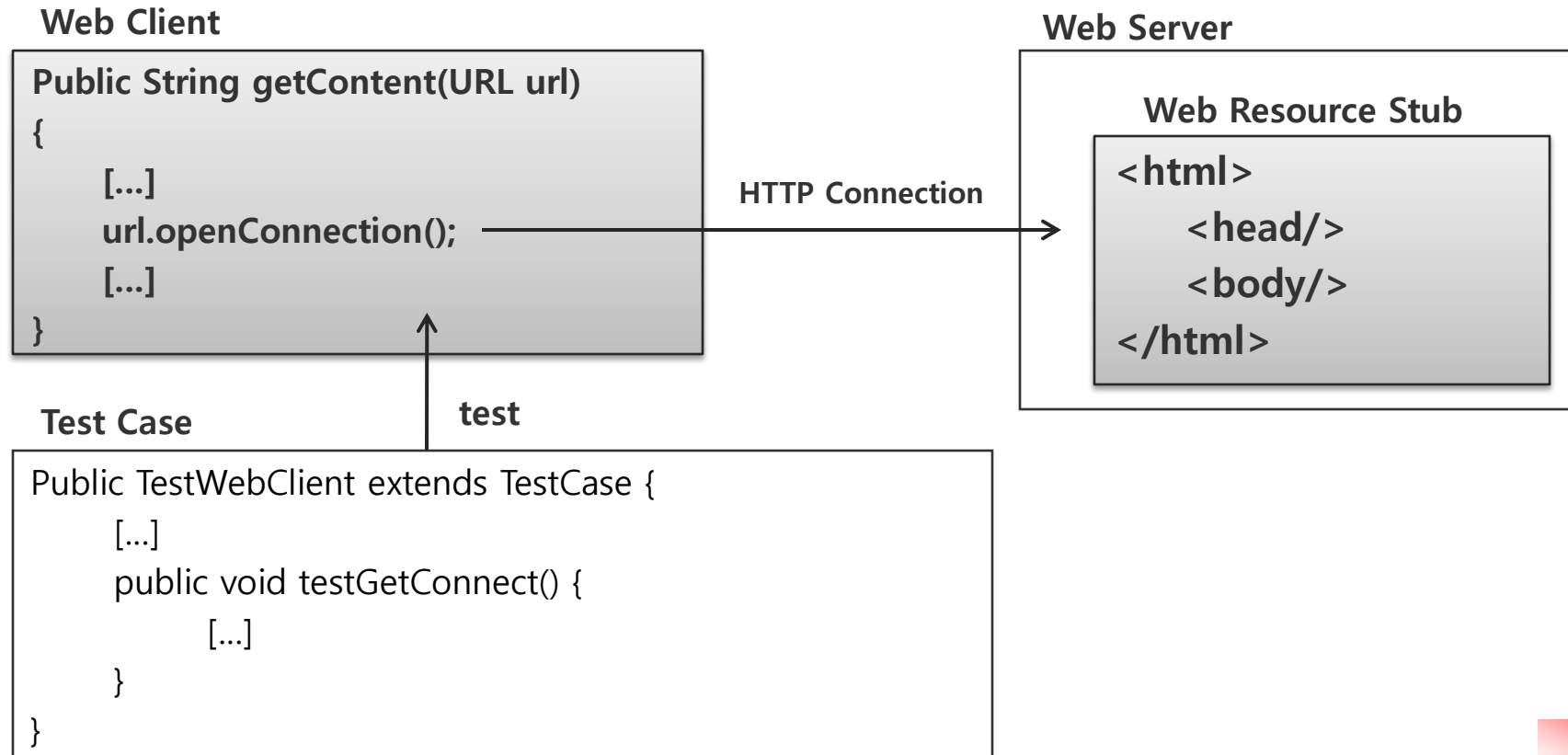
□ 테스트 목표

- 웹 서버에서 접속해서 콘텐츠를 읽어오는 코드를 테스트 하려 함
- 이 코드를 테스트하기 위해서는 어떤 방법이 존재할까?



□ 제약 조건

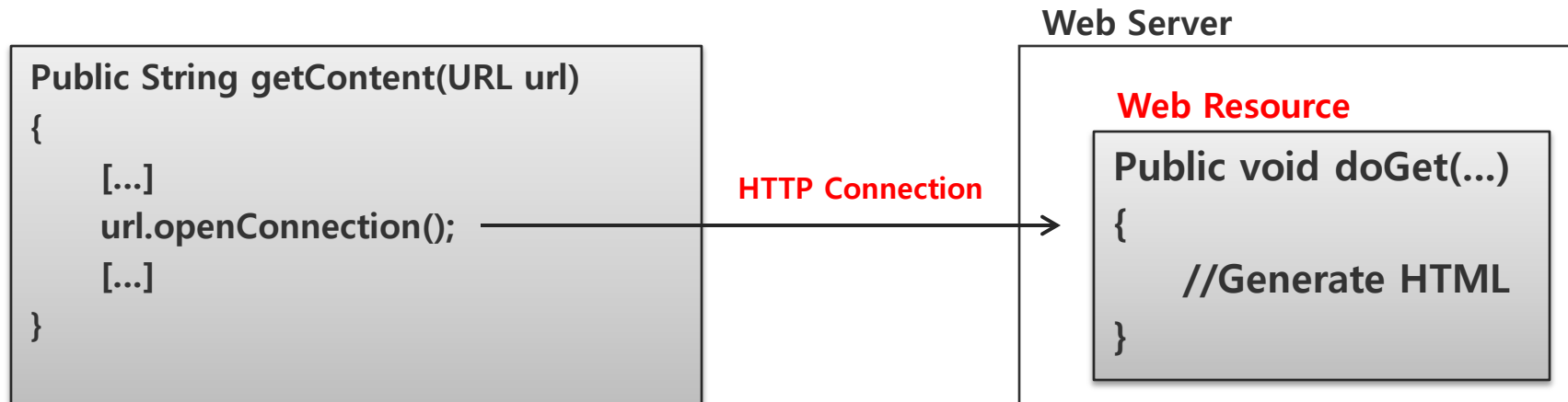
- 웹 리소스 구현과 독립적으로 **getContent** 메서드를 테스트 해야 함
- **getContent** 메서드에 변경을 가하면 안됨
- 테스트 코드는 외부 환경에 독립적으로 실행되어 함



HTTP Connection 예제

□ 해결책 (Solution)

- 웹 리소스 스터빙(WebServer Resource Stubbing)
- HTTP 연결 스터빙(HTTP Connection Stubbing)



웹 리소스 스템빙(Stubbing)

□ Web Resource Stubbing

- 테스트를 위해 웹 서버를 설치 VS 임베디드(Embedded) 웹 서버 사용

□ 실제 웹 서버 설치

- 테스트를 실행하기 전에 테스트 환경을 완벽하게 구축해야 함
- 테스트 실패 시, 테스트 실패의 원인을 찾기가 어려움
- 테스트가 반복 가능하기 위해서는 내부적으로 환경을 제어할 수 있어야 함
- **결국, 테스트 원칙에 위배됨**

□ 임베디드 웹 서버 사용

- **Jetty** 같은 경량 임베디드 웹 서버 사용
- 테스트 케이스 내에서 자바 코드로 웹 서버를 제어할 수 있음
- 따라서, 원하는 응답 설정이 가능

□ Jetty Sample

```
public class JettySample
{
    public static void main(String[] args) throws Exception
    {
        HttpServer server = new HttpServer();
        SocketListener listener = new SocketListener();
        listener.setPort(8080);
        server.addListener(listener);

        HttpContext context = new HttpContext();
        context.setContextPath("/");
        context.setResourceBase("./");
        context.addHandler(new TestGetContentOkHandler());
        server.addContext(context);
        server.start();
    }
}
```

HTTP Connection 예제

□ Jetty 응답 처리 Handler 작성

```
private class TestGetContentOkHandler extends AbstractHttpHandler
{
    public void handle(String pathInContext, String pathParams,
        HttpRequest request, HttpResponse response) throws IOException
    {
        OutputStream out = response.getOutputStream();
        ByteArrayISO8859Writer writer = new ByteArrayISO8859Writer();
        writer.write("It works");
        writer.flush();
        response.setIntField(HttpFields.__ContentLength,
            writer.size());
        writer.writeTo(out);
        out.flush();
        request.setHandled(true);
    }
}
```

□ Jetty 를 이용하여 작성할 Junit Test Case의 구조

```
public class TestWebClientSkeleton extends TestCase
{
    protected void setUp()    {
        //여기에서 Jetty 서버 시작
    }

    protected void tearDown()    {
        //Jetty 종료
    }

    public void testGetContentOk() throws Exception {
        WebClient client = new WebClient();
        String result = client.getContent(new URL( "http://localhost:8080/testGetContentOk"));
        assertEquals ("It works", result);
    }
}
```


웹 리소스 스템킹(Stubbing) - 고려사항

□ 테스트 독립성

- 테스트는 깔끔함과 단정함을 유지해야 함
- 환경과 다른 개발자들에게서 독립적인 상태를 유지해야 함
- 테스트 메서드 단위에서도 독립성을 유지해야 함

□ 성능

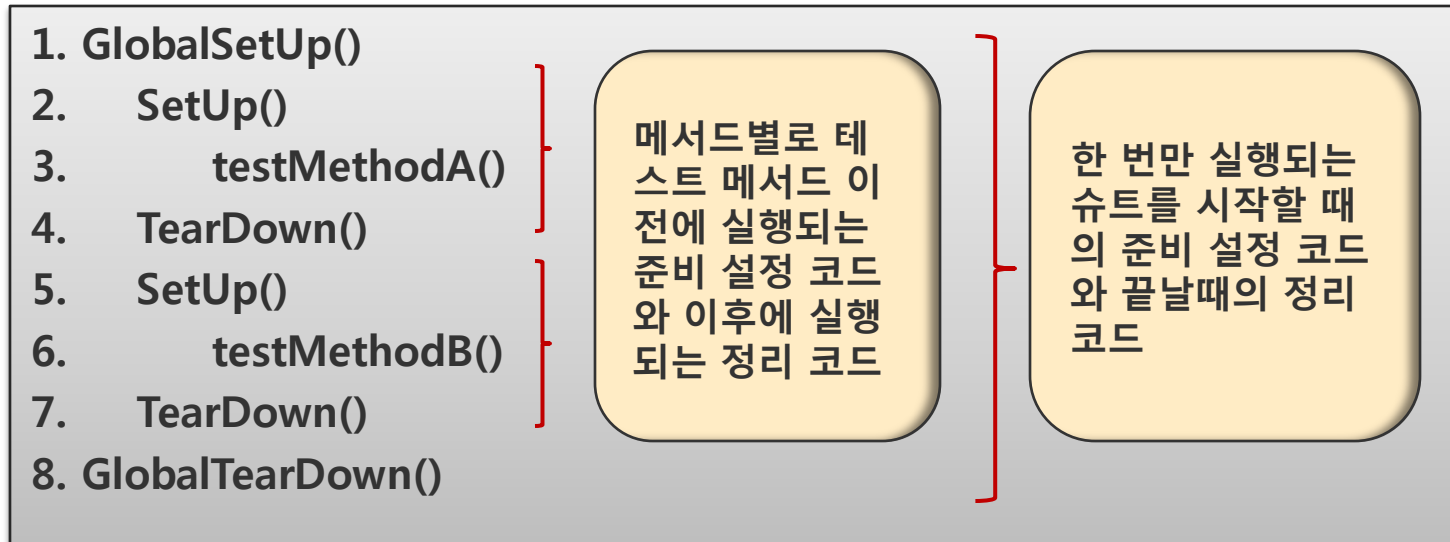
- 모든 테스트를 실행하는 데 너무 오랜 시간이 걸려서는 안됨
- 너무 오래 걸리는 테스트는 개발에 심각한 부담을 줄 수 있음

□ 테스트 독립성 VS 성능

- 깨끗한 환경의 오래 걸리는 테스트 vs 일부 환경을 재사용하는 빠른 테스트
- 주어진 상황에 맞는 적절한 해결책을 선택하는 것이 중요

□ TestCase에서 전역 환경 설정하기

- junit.extensions.TestSetup 클래스 이용하기
- TestSetup 클래스를 확장하여 전역 Setup, Teardown 메서드를 정의
- TestCase의 Suite 메서드에서 TestSetup 인스턴스를 반환하도록 함



□ **TestSetup**을 이용한 전역 환경 설정 클래스 만들기

```
public class TestWebClientSetup1 extends TestSetup
{
    protected static HttpServer server;
    public TestWebClientSetup1(Test suite) {
        super(suite);
    }
    protected void setUp() throws Exception {
        [...]
        server.start();
    }
    protected void tearDown() throws Exception e {
        server.stop();
    }
    private class TestGetContentOkHandler extends AbstractHttpHandler {
        [...]
    }
}
```

□ TestCase에서 전역 TestSetup 사용하기

```
public class TestWebClient1 extends TestCase {  
  
    public static Test suite() {  
        TestSuite suite = new TestSuite();  
        suite.addTestSuite(TestWebClient1.class);  
        return new TestWebClientSetup1(suite);  
    }  
  
    public void testGetContentOk() throws Exception {  
        WebClient client = new WebClient();  
        String result = client.getContent(new URL( "http://localhost:8080/testGetContentOk"));  
        assertEquals("It works", result);  
    }  
}
```

웹 리소스 스템(Stubbing) 회고

□ 웹 리소스 스템 회고

- 웹 리소스 스템으로 getContent 메서드의 독립적 유닛 테스트 보장
- getContent, 더하여 HttpURLConnection까지 테스트
- 결국은 통합테스트까지 실행한 것이 됨

□ 문제 점

- 유닛 테스트 단계에서 통합테스트까지 할 필요가 없음
- 스템을 작성하는 데 오랜 시간이 걸림
- 테스트케이스마다 서버에 다른 값을 기대할 수 있고, 이럴 경우 전체 테스트가 너무 무거워짐



스템은 되도록 간단해야 하며,
스템을 유지보수하는데 많은 노력이 들어가면 안됨

HTTP Connection 스템킹(Stubbing)

□ HTTP Connection 스템킹

- 웹 리소스 자체를 스템킹하지 않고 HTTP Connection을 스템킹 함
- 웹 서버가 필요 없음
- getContent 메서드 테스트에만 집중할 수 있음
- 작성하기 쉽고, 실패 시 오류 발견의 쉬움

□ 작성방법

- URL 객체의 인터페이스 이용
- URL 객체가 커스텀 HttpURLConnection객체를 반환하도록 설정
- 커스텀 HttpURLConnection 객체가 우리가 원하는 결과를 반환하도록 함

우리의 관심 대상이 되는 메서드(인터페이스)만을 구현해야 함!!
스템의 복잡도 관리가 관건!

HTTP Connection 스템빙(Stubbing)

□ Custom URL Protocol Handler 작성

필요한 메서드만
구현

```
public class TestWebClient1 extends TestCase {
    protected void setUp() {
        URL.setURLStreamHandlerFactory(new StubStreamHandlerFactory());
    }
    private class StubStreamHandlerFactory implements URLStreamHandlerFactory {
        public URLStreamHandler createURLStreamHandler(String protocol) {
            return new StubHttpURLStreamHandler();
        }
    }
    private class StubHttpURLStreamHandler extends URLStreamHandler {
        protected URLConnection openConnection(URL url) throws IOException {
            return new StubHttpURLConnection(url);
        }
    }
    public void testGetContentOk() throws Exception {
        WebClient client = new WebClient();
        String result = client.getContent(new URL("http://localhost:8080/testGetContentOk"));
        assertEquals("It works", result);
    }
}
```

HTTP Connection 스템빙(Stubbing)

□ HttpURLConnection Stub 만들기

필요한 메서드만
구현

```
public class StubHttpURLConnection extends HttpURLConnection
{
    private boolean isInput = true;
    protected StubHttpURLConnection(URL url) {
        super(url);
    }
    public InputStream getInputStream() throws IOException {
        if (!isInput) {
            throw new ProtocolException("Cannot read from URLConnection" +
                " if doInput=false (call setDoInput(true))");
        }
        ByteArrayInputStream bais = new ByteArrayInputStream(new String("It works").getBytes());
        return bais;
    }
    public void disconnect() {}
    public void connect() throws IOException {}
    public boolean usingProxy() {
        return false;
    }
}
```


모의 객체(Mock Object)의 활용

- 모의 객체(Mock Object) 개요
- **Unit test and Mock Objects**
- 모의 객체 사용방법
- 간단한 모의 객체 활용 예제
- **Mock Objects Frameworks** 소개

모의 객체(Mock Object) 개요

□ 모의 객체의 정의

- 단위 테스트를 위한 보조 객체
- 진짜 객체(real objects)를 흉내내는 가짜 객체(dummy implementations)
- 단위 테스트가 시스템 환경으로부터 독립적이며, 안정적이고, 재사용 가능하도록 돕는 객체

□ 모의 객체의 적용시점

- 진짜 객체가 **정해지지 않은 결과를 제공**하는 경우
- 진짜 객체를 **준비 설정하기 어려운** 경우
- 진짜 객체가 네트워크 에러와 같은 **직접 유발시키기 어려운 동작**을 하는 경우
- 진짜 객체가 **느린** 경우
- 진짜 객체가 **사용자 인터페이스를 가지거나, 사용자 인터페이스인** 경우
- **테스트가 진짜 객체가 수행한 작업을 알아야 하는** 경우
- 진짜 객체가 **아직 존재하지 않는** 경우

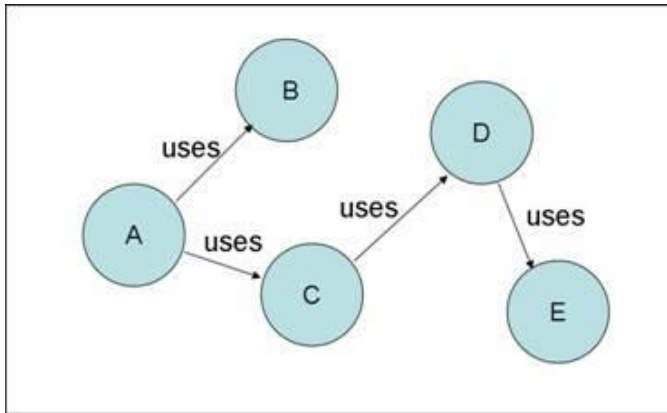
모의 객체(Mock Object) 개요 - 계속

□ 모의 객체 사용 시 주의사항

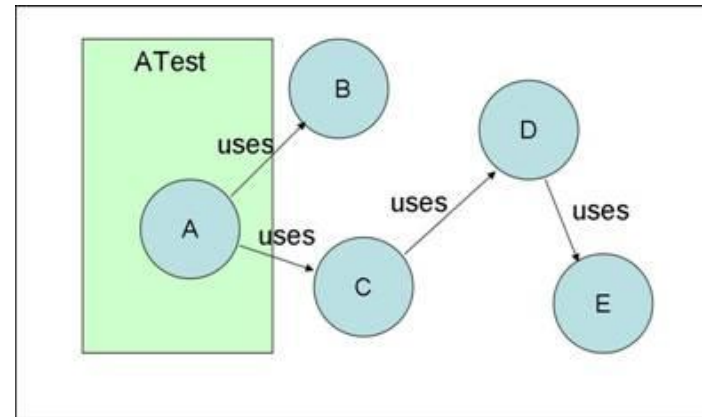
- 꼭 써야 할 경우에 최소한으로 사용하도록 한다.
- Mock을 과도하게 사용할 경우, 테스트의 가치는 낮아지며 오히려 테스트에 버그가 생길 수 있다.

Unit test and Mock Objects

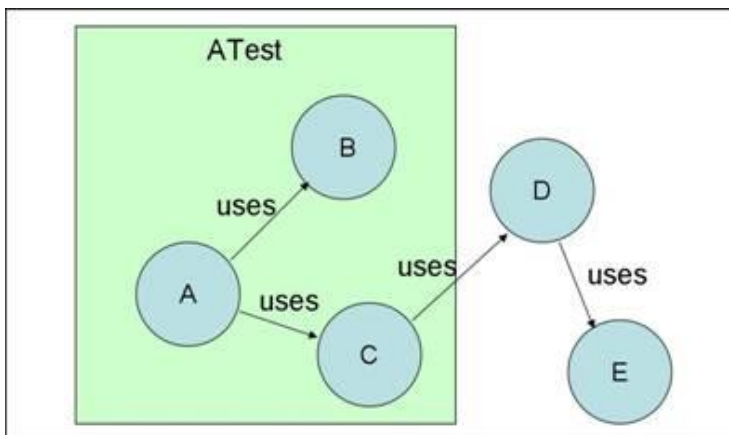
□ Object Oriented(OO) application에서의 모의 객체를 활용한 단위 테스트



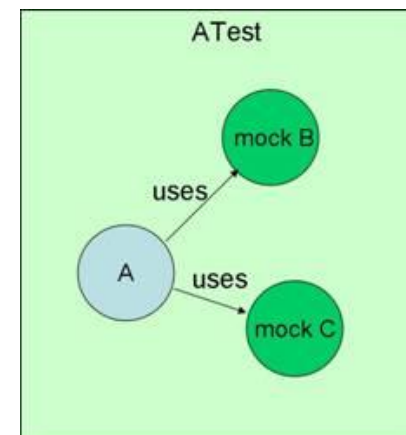
a network of
objects



ATest unit
test



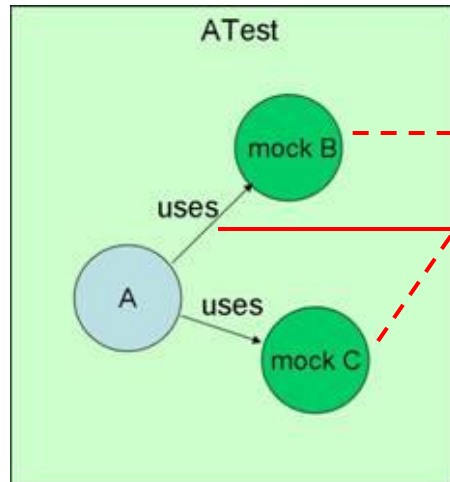
ATest adding B
and C



ATest with mock B and
mock C

□ 테스트에서 모의 객체를 사용하기 위한 핵심 세 단계

- 객체를 설명하기 위해 인터페이스를 사용
- 제품 코드에 맞게 그 인터페이스를 구현
- 테스트에 쓸 모의 객체의 인터페이스를 구현



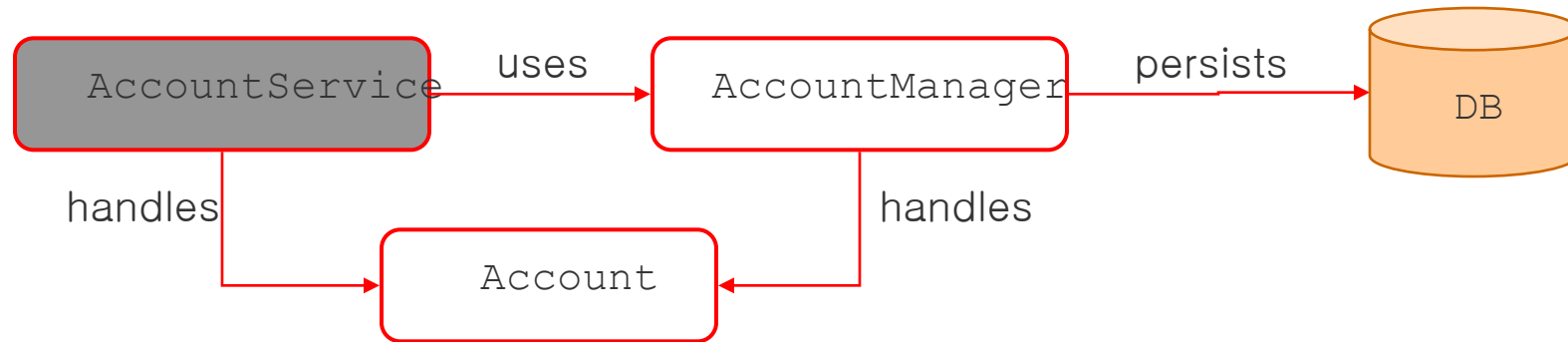
Mock B,C : 제품 코드 B,C Interface의 구현체

ATest는 mock B,C의 인스턴스를 생성하여 사용함

➡ A(테스트 대상이 되는 코드)는 B, C의 진짜 객체(real object)를 사용하는지 모의 객체(Mock Object)를 사용하는지 알지 못한다.

간단한 모의 객체 활용 예제 – 계좌이체 테스트 (1/5)

□ 은행 계좌 이체 테스트



□ 각 클래스의 역할

- AccountService : 계좌이체 기능 제공
- Account : 계좌정보(고객ID, 잔액) 객체로써, 기본적인 입금,출금 기능 제공
- AccountManager : 고객ID로 계좌정보 조회, 계좌정보 수정 기능 제공

□ 가정

- 계좌이체 단위테스트를 수행하고자 함
- 계좌 정보를 저장할 수 있는 DB가 제공되지 않음
- AccountManager는 아직 구현되지 않은 인터페이스를 사용

간단한 모의 객체 활용 예제 – 계좌이체 테스트 (2/5)

□ Account.java

```
public class Account {  
    private String accountId; // 계좌아이디  
    private long balance; // 잔액  
    public Account(String accountId, long initialBalance) {  
        this.accountId = accountId; this.balance = initialBalance;  
    }  
    public void debit(long amount) {  
        this.balance -= amount;  
    }  
    public void credit(long amount) {  
        this.balance += amount;  
    }  
    public long getBalance() {  
        return this.balance;  
    }  
}
```

□ AccountManager.java

```
public interface AccountManager {  
    Account findAccountForUser(String userId);  
    void updateAccount(Account account);  
}
```

간단한 모의 객체 활용 예제 – 계좌이체 테스트 (3/5)

□ AccountService.java

```
public class AccountService {
    private AccountManager accountManager;
    public void setAccountManager(AccountManager manager) {
        this.accountManager = manager;
    }
    public void transfer(String senderId, String beneficiaryId, long amount) {
        Account sender = this.accountManager.findAccountForUser(senderId);
        Account beneficiary = this.accountManager.findAccountForUser(beneficiaryId);
        sender.debit(amount);
        beneficiary.credit(amount);
        this.accountManager.updateAccount(sender);
        this.accountManager.updateAccount(beneficiary);
    }
}
```

□ 상황

- transfer() 메서드의 단위 테스트 수행하고자 함
- AccountManager 인터페이스가 미구현 상태이며, 수행할 기능을 정의한 인터페이스가 제공되어 있음
- AccountManager의 기능을 대신할 Mock Object를 생성하여 이체 테스트를 수행하기로 함

간단한 모의 객체 활용 예제 – 계좌이체 테스트 (4/5)

□ MockAccountManager.java

```
public class MockAccountManager implements AccountManager {
    private Hashtable accounts = new Hashtable();
    public void addAccount(String userId, Account account) {
        this.accounts.put(userId, account);
    }
    public Account findAccountForUser(String userId) {
        return (Account) this.accounts.get(userId);
    }
    public void updateAccount(Account account) {
        // do nothing
    }
}
```

- AccountManager를 implements하는 MockAccountManager를 작성
- MockAccountManager에는 어떠한 비즈니스 로직(**business logic**)도 포함하고 있지 않으며, transfer method 단위 테스트를 수행하기 위한 최소한의 기능을 제공함

간단한 모의 객체 활용 예제 – 계좌이체 테스트 (5/5)

□ TestAccountService.java

```
public class TestAccountService extends TestCase {
    public void testTransferOk() {
        // Mock Object 인스턴스 생성
        MockAccountManager mockAccountManager = new MockAccountManager();

        Account senderAccount = new Account("1", 200);
        Account beneficiaryAccount = new Account("2", 100);

        mockAccountManager.addAccount("1", senderAccount);
        mockAccountManager.addAccount("2", beneficiaryAccount);

        AccountService accountService = new AccountService();
        accountService.setAccountManager(mockAccountManager);
        accountService.transfer("1", "2", 50);

        assertEquals(150, senderAccount.getBalance());
        assertEquals(150, beneficiaryAccount.getBalance());
    }
}
```

1) Mock Object의 인스턴스를 생성한다.

2) 예상되는 객체의 행위(behavior)를 설정한다. (모의 객체가 stub()과는 다른 점이다)

3) 단위 테스트가 예상대로 수행되는지 검증한다. (assertion을 이용한 검증)

Mock Objects Frameworks 소개

□ Mock Objects Frameworks

- 애플리케이션에서 custom mock 객체를 구현하는 번거로움을 없애기 위해 다양한 Mock 프레임워크가 존재
- 대표적인 Mock Frameworks : JMock, EasyMock, RMock 등

□ JMock

- 장점
 - 잘 정리된 API를 제공하여 편의 제공
 - 클래스와 인터페이스의 구체적인 모방이 가능
- 단점
 - 리팩토링이 어려움

□ EasyMock

- 장점
 - 리팩토링이 쉬우며, stub의 형태로 사용 가능
- 단점
 - 구체적인 클래스의 모방이 어려움

□ RMock

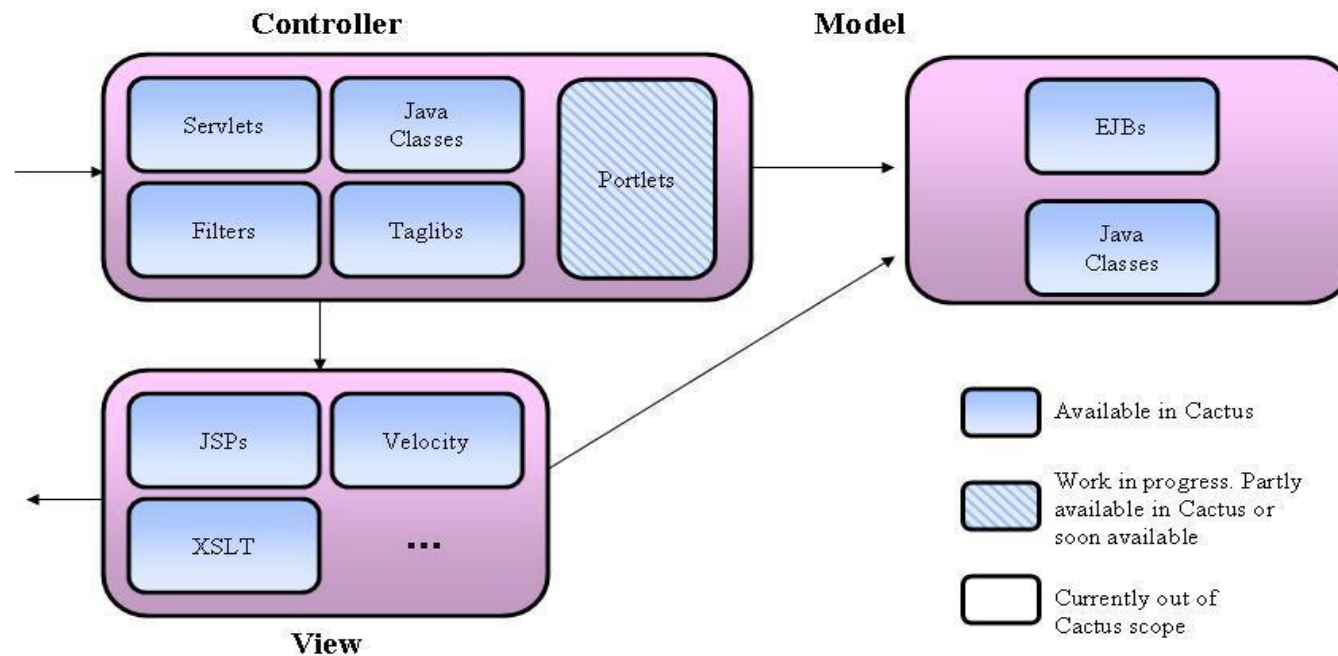
- 장점
 - 특정 모호한 케이스들에 대한 테스트가 가능하여, JMock 등과 함께 사용 가능

Cactus를 활용한 컨테이너 기반 테스트

- ❑ Cactus 소개
- ❑ Cactus를 이용한 컴포넌트 테스트
- ❑ Cactus 테스트 실행하기
- ❑ Cactus 작동원리

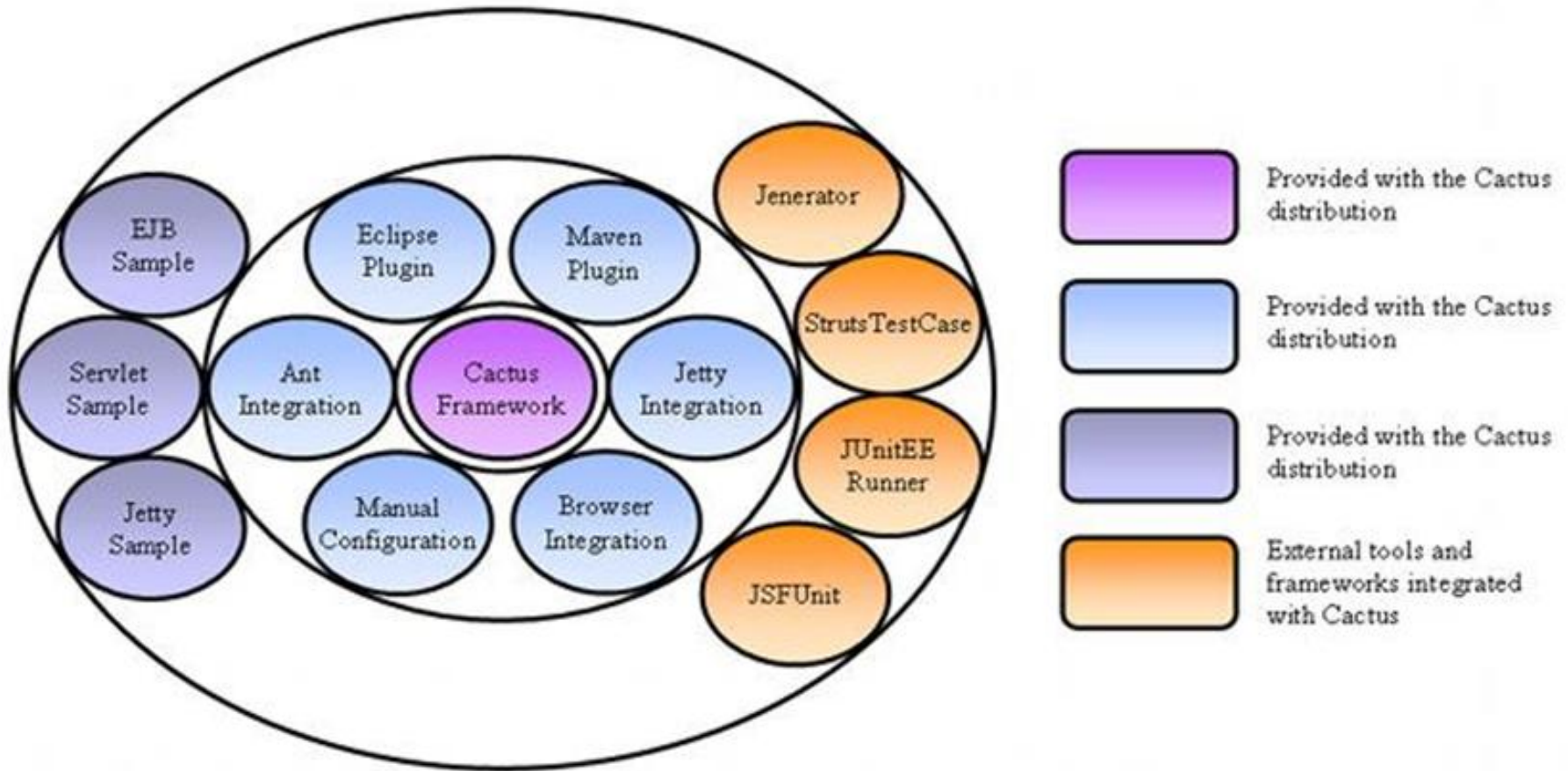
□ Cactus 소개

- 서버 사이드 자바 코드를 테스트하기 위한 테스트 프레임워크
- 서버 사이드 코드에 대한 테스트 작성을 쉽게 할 수 있도록 함
- Junit을 기반으로 함
- 컨테이너 내에서 실행되도록 설계되어 있음
- <http://jakarta.apache.org/cactus>



Cactus소개 - 생태계

□ Cactus 생태계



Cactus 소개 – 생태계 구성

□ Cactus 프레임워크(Cactus Framework)

- Cactus의 핵심 부분
- Cactus 테스트를 작성하기 위한 API를 제공

□ Cactus 통합 모듈 (Cactus Integration Module)

- Cactus Front Ends
- Cactus 프레임워크를 쉽게 사용할 수 있는 방법 제공
- Ant scripts, Eclipse 플러그인, Maven 플러그인...

□ Cactus 예제(Cactus sample)

- Cactus 테스트를 만드는 방법을 소개
- 통합 모듈 사용 방법에 대한 소개

Cactus가 바라보는 3가지 관점의 단위 테스트

□ 코드 단위 테스트

- 스텝이나 모의 객체를 사용하여 테스트 함

□ 통합 단위 테스트

- Cactus 를 사용할 수 있는 테스트 유형
- 컨테이너에서의 상호작용을 테스트할 수 있음

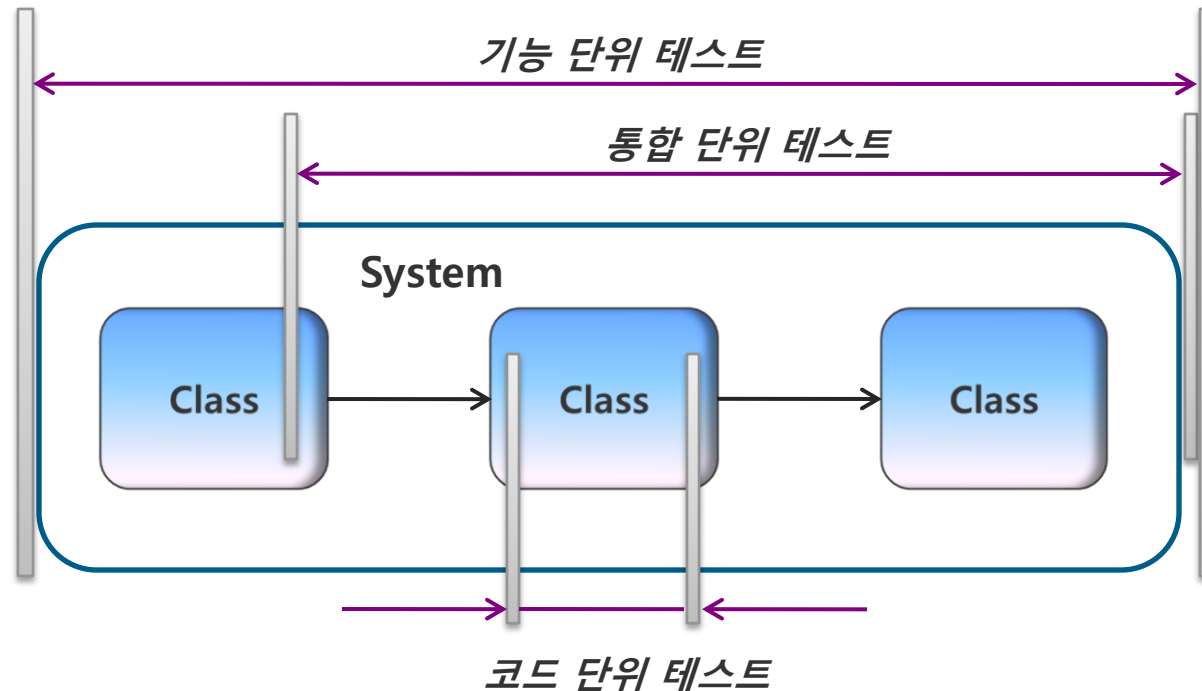
□ 기능 단위 테스트

- 서버로부터 기대된 응답이 오는지 테스트 – 서버 응답 값 테스트
- HTTPUnit이 이런 종류의 테스터에 해당
- 유스케이스 단위의 테스트 (ex, 로그인 유스케이스)

Cactus 통합 단위 테스트

□ 통합 단위테스트(Integration Unit Test)의 장점

- 단위 테스트의 장점을 모두 가짐
- 코드가 대상 컨테이너에서도 올바르게 작동하는지를 확인가능
- 기능 단위 테스트로도 확장 가능



Cactus를 이용한 테스트 작성

- **Cactus**를 이용하여 테스트할 수있는 서버 모듈
 - EJB
 - HTTPUnit
 - JSP
 - Security
 - JSP
 - Servlet
 - Servlet-Filter

Cactus를 이용한 컴포넌트 테스트

□ 예제 서블릿

```
public class SampleServlet extends HttpServlet
{
    public boolean isAuthenticated(HttpServletRequest request)
    {
        HttpSession session = request.getSession(false);
        if (session == null) {
            return false;
        }
        String authenticationAttribute = (String) session.getAttribute("authenticated");
        return Boolean.valueOf(authenticationAttribute).booleanValue();
    }
}
```

Cactus를 이용한 컴포넌트 테스트

□ Cactus를 사용하여 Test Case 작성

```
public class TestSampleServletIntegration extends ServletTestCase {  
  
    private SampleServlet servlet;  
    protected void setUp() {  
        servlet = new SampleServlet();  
    }  
    public void testIsAuthenticatedAuthenticated() {  
        session.setAttribute("authenticated", "true");  
        assertTrue(servlet.isAuthenticated(request));  
    }  
    public void testIsAuthenticatedNotAuthenticated() {  
        assertFalse(servlet.isAuthenticated(request));  
    }  
    public void beginIsAuthenticatedNoSession(WebRequest request) {  
        request.setAutomaticSession(false);  
    }  
    public void testIsAuthenticatedNoSession() {  
        assertFalse(servlet.isAuthenticated(request));  
    }  
}
```

Cactus 테스트 실행하기

□ Cactus 테스트 실행 방법

- 컨테이너에서 `test`가 일어나기 때문에 순수 Junit 테스트 보다 실행이 복잡함

□ 실행 순서

1. 코드 패키징 (packaging codes)
2. 코드 배포 (deploy codes)
3. 컨테이너 시작
4. Junit 테스트 실행

□ Cactus/Jetty Integration 사용

- 복잡한 컨테이너 테스트를 간소화해주는 여러 통합 라이브러리 제공
- Ant Integration, Browser Integration, Eclipse Plugin, Jetty Integration...

Cactus 테스트 실행하기

□ Cactus/Jetty Integration 사용

- 대부분의 IDE에서 모두 실행 가능
- 테스트 실행 시간이 매우 빠름
- Breakpoint를 이용한 디버깅 가능

□ Cactus/Jetty Integration 사용 방법

- Cactus/Jetty Integration을 Eclipse에 설치
- Cactus/Jetty 통합을 위한 Junit 테스트 슈트를 생성 해야 함
- 테스트 슈트에서 JettyTestSetup을 사용하도록 지정

□ TIP : Cactus 테스트와 순수 Junit 테스트 분리

- 컨테이너 테스트와 순수 Junit 테스트는 물리적으로 분리하는 것을 권장함
- 두 테스트는 생명 주기가 다름
- 순수 Junit 테스트가 Cactus보다 더 많이 실행되며, 더 빨리 실행되어야 함
- Cactus테스트는 순수 Junit테스트와는 다른 실행 환경을 갖음

Cactus 테스트 실행하기

□ JettyTestSetup으로 실행하는 Cactus 테스트

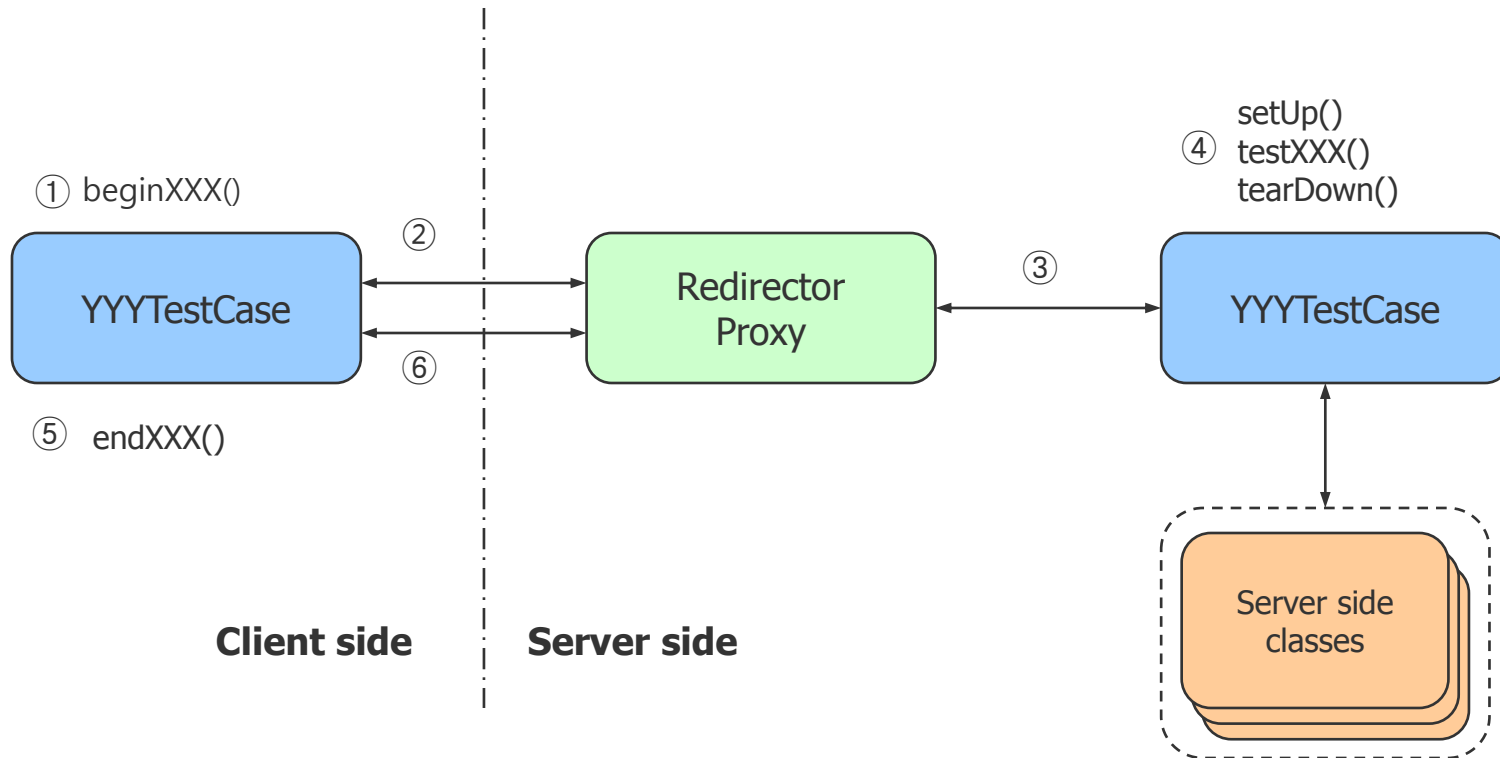
```
public class TestAllWithJetty
{
    public static Test suite() {
        System.setProperty("cactus.contextURL", "http://localhost:8080/test");
        TestSuite suite = new TestSuite("All tests with Jetty");
        suite.addTestSuite(TestSampleServletIntegration.class);
        return new JettyTestSetup(suite);
    }
}
```

Cactus 테스트 실행하기

□ 컨테이너 테스트의 단점

- 테스트하려고 하는 대상 컴포넌트에 특화된 툴이 필요함
 - Cactus를 사용하여 J2EE 애플리케이션을 테스트
 - 다른 컴포넌트 모델에는 그에 맞는 테스트 프레임워크가 필요
 - 이에 반해, 모의객체는 거의 모든 컴포넌트 모델에서 사용 가능
- 테스트 실행 시간이 길어짐
 - 컨테이너 설정과 실행에 많은 시간이 소요됨
 - Jetty : 1초, Tomcat : 5초, Weblogic : 30초
- 설정이 복잡함
 - 테스트를 실행하기 전에 배포가 필요
 - EJB 같은 경우 복잡한 DD를 설정해야 함
 - 개발환경 내에서 데이터베이스 접근이 용이하지 않을 수도 있음

□ Cactus 작동 원리



단위 테스트 품질 향상을 위한 툴 소개

- ❑ 코드 커버리지 분석기 – **Coverlipse**
- ❑ 패키지 의존성 분석기 – **JDepend**
- ❑ 코딩 표준 분석기 - **CheckStyle**

Coverlipse – 코드 커버리지 분석기

□ Coveripse

- 코드 커버리지 분석을 위한 이클립스 플러그-인
- junit 테스트를 사용하여 테스트된 소스코드의 비율을 평가
- Junit과 연동하여 갖고 있는 사용하여 Junit 테스트의 코드 커버리지를 표시

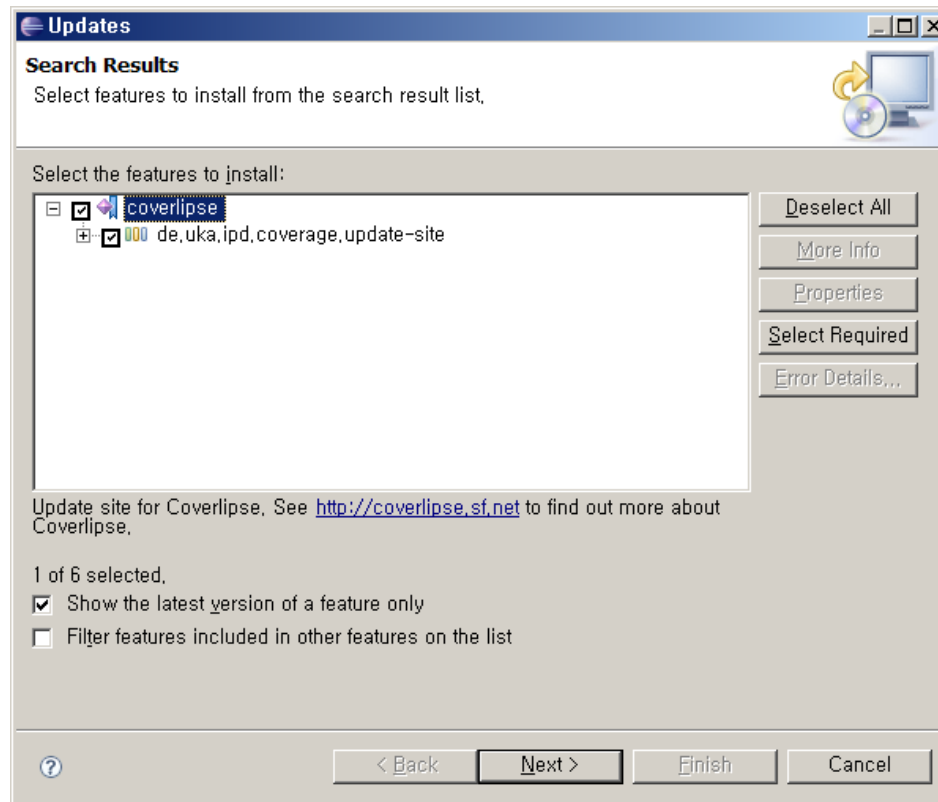
□ Coverlipse의 기능

- 한번의 실행만으로 모든 테스트 커버리지 분석 가능
- 블록 커버리지
- 특별한 설정이 필요 없음
- 테스트에 포함하거나 배제해야할 패키지를 쉽게 선택할 수 있음
- 이클립스 자바 에디터에 결과가 바로 반영

Coverlipse – 코드 커버리지 분석기

□ 설치

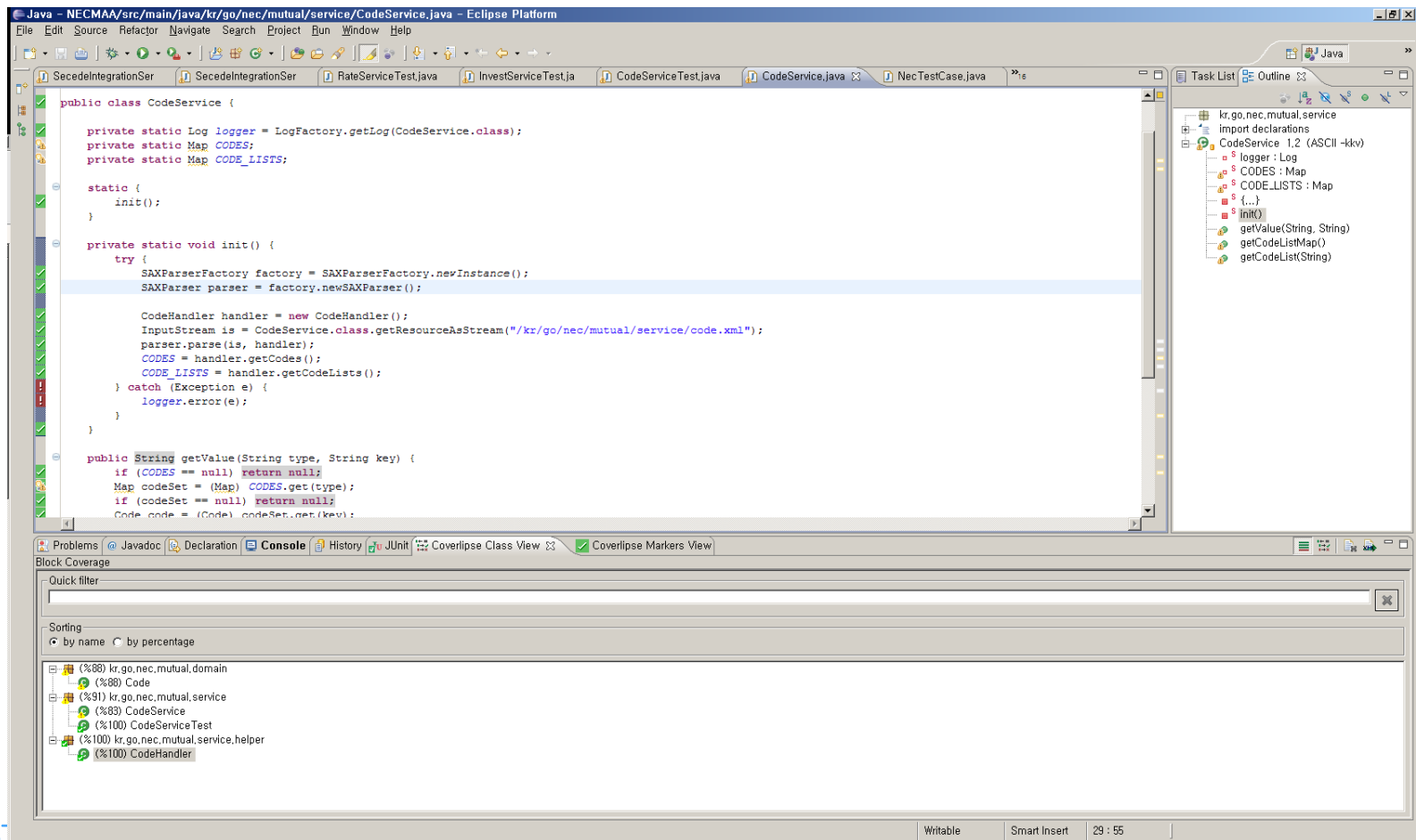
- Eclipse의 Software Update... 기능을 통해 설치
- <http://coverlipse.sf.net/update>
- Eclipse JDT Plug-in Developer Resources도 같이 업데이트



Coverlipse – 코드 커버리지 분석기

□ 실행

- Junit 테스트 케이스를 선택한 후 Run As...에서 Junit w/coverLipse 선택
- 테스트 케이스 실행 후 다양한 형태로 코드 커버리지 결과 확인 가능



Cobertura – 코드 커버리지 분석기

□ Cobertura

- 정적 코드 커버리지 분석 도구
- junit 테스트를 사용하여 테스트된 소스코드의 비율을 평가
- Junit과 연동하여 사용하며 Junit 테스트의 코드 커버리지를 표시

□ Coverlipse의 기능

- 테스트 실행결과로 코드 커버리지 리포트를 생성
- 그 외에도 현재 프로젝트에 대한 다양한 정보를 제공
 - Dependency 목록
 - CI 설정 정보
 - Issue 트래킹 서버 설정 정보
 - 형상관리 서버 정보

Coverlipse – 코드 커버리지 분석기

□ 설치

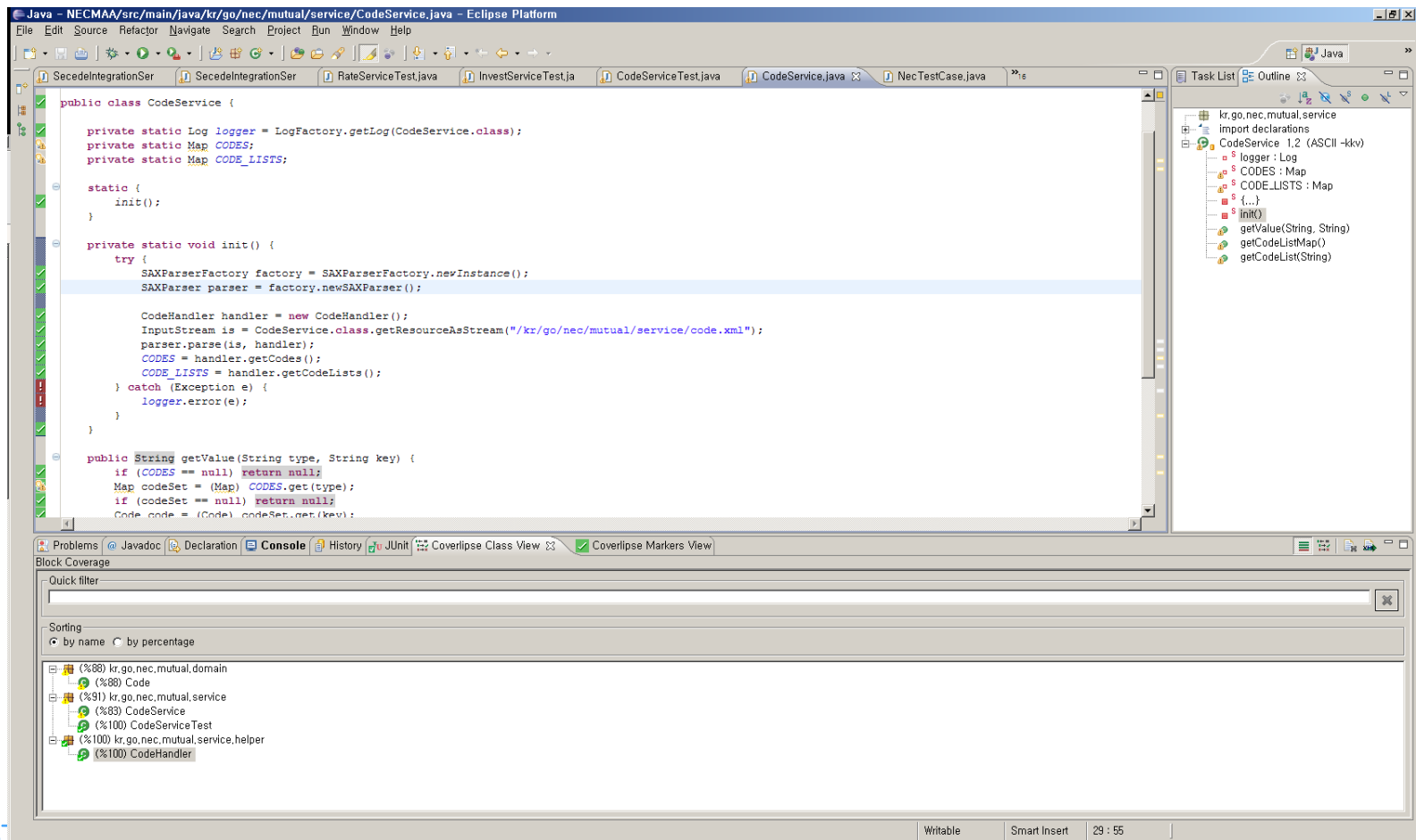
- Pom.xml에 Plugins에 다음과 같은 구문 추가

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>cobertura-maven-plugin</artifactId>
  <executions>
    <phase>pre-site</phase>
    <execution>
      <goals>
        <goal>clean</goal>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Coverlipse – 코드 커버리지 분석기

□ 실행

- Junit 테스트 케이스를 선택한 후 Run As...에서 Junit w/coverLipse 선택
- 테스트 케이스 실행 후 다양한 형태로 코드 커버리지 결과 확인 가능



JDepend – 패키지 의존성 분석기

□ Jdepend

- Java 클래스 파일 디렉토리를 검색
- 각 자바 패키지의 설계 품질 매트릭을 생성
- 확장성, 재사용성, 유지보성의 관점에 설계 품질을 측정
- 아키텍처 정합성 측정

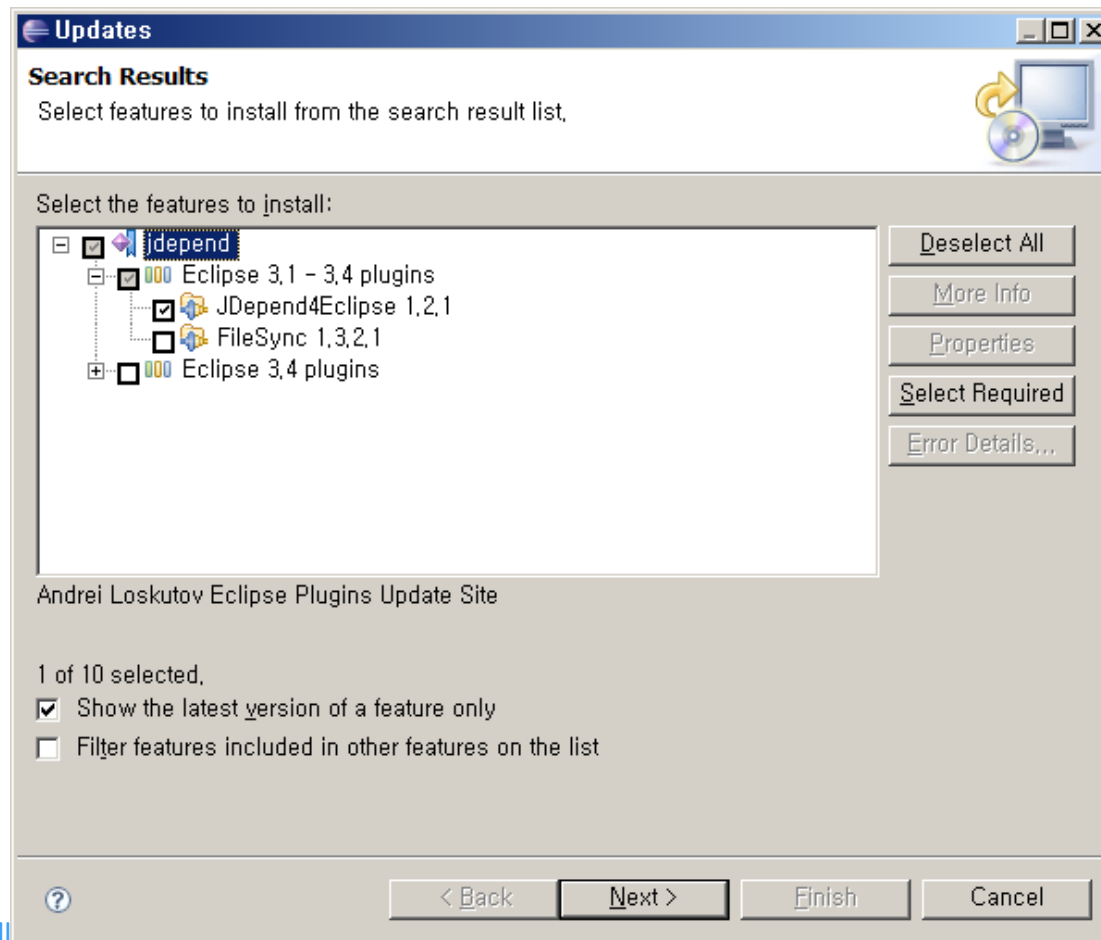
□ 주의사항

- 매트릭에 나오는 결과가 절대적인 좋고/나쁨을 의미하지 않음
- Jdepend에 의해 생성된 설계 품질 매트릭을 설계 측정의 기준으로 삼지 말것
- 설계에 대한 측정과 이해의 용도가 적합함
- 다음 리팩토링 작업에서의 참고 자료 및 결과 확인
- Jdepend는 툴 이상도 이하도 아님

JDepend - 패키지 의존성 분석기

설치

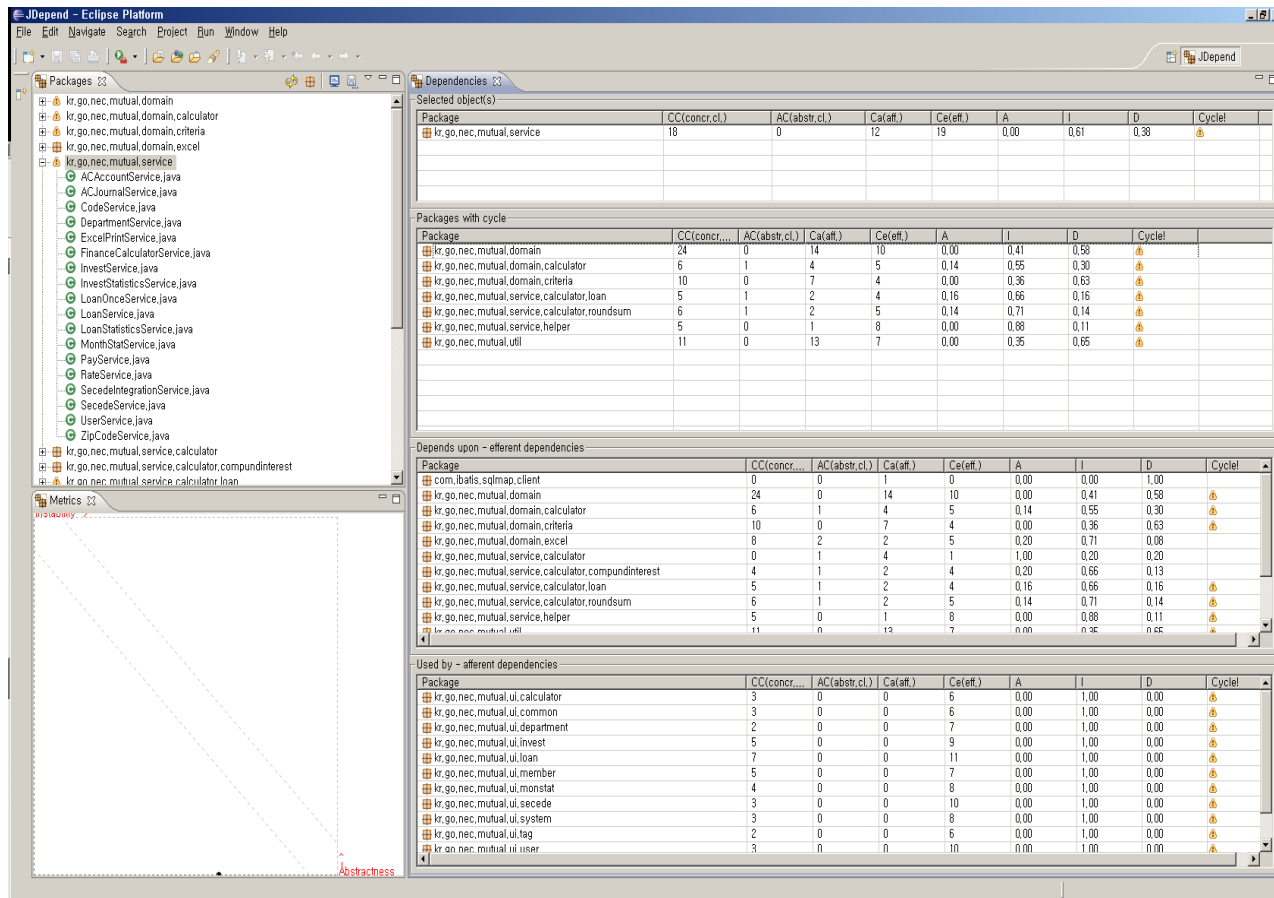
- Eclipse의 Software Update... 기능을 통해 설치
- <http://andrei.gmxhome.de/eclipse/>



JDepend - 패키지 의존성 분석기

□ 실행

- 소스 폴더를 오른쪽 클릭 후 Run Jdepend Analysis 수행
- 분석을 실행하여 결과 리포트가 표시 됨



The screenshot displays the JDepend Eclipse Platform interface. The left pane shows a project tree with various packages under 'kr.go.nec.mutual'. The right pane shows the 'Dependencies' view, which includes several tables:

- Selected object(s):** A table showing the selected package 'kr.go.nec.mutual.service' and its metrics.
- Packages with cycle:** A table listing packages involved in dependency cycles, such as 'kr.go.nec.mutual.domain', 'kr.go.nec.mutual.domain.calculator', and 'kr.go.nec.mutual.domain.criteria'.
- Depends upon - efferent dependencies:** A table listing packages that the selected package depends on, including 'com.ibatis.sqlmap.client', 'kr.go.nec.mutual.domain', and 'kr.go.nec.mutual.domain.calculator'.
- Used by - afferent dependencies:** A table listing packages that use the selected package, including 'kr.go.nec.mutual.ui.calculator', 'kr.go.nec.mutual.ui.department', and 'kr.go.nec.mutual.ui.invest'.

Each table contains columns for Package, CC(concr.cl.), AC(abstr.cl.), Ca(aff.), Ce(eff.), A, I, D, and Cyclist. The Cyclist column contains small icons indicating the type of dependency or cycle.

CheckStyle - 코딩 표준 분석기

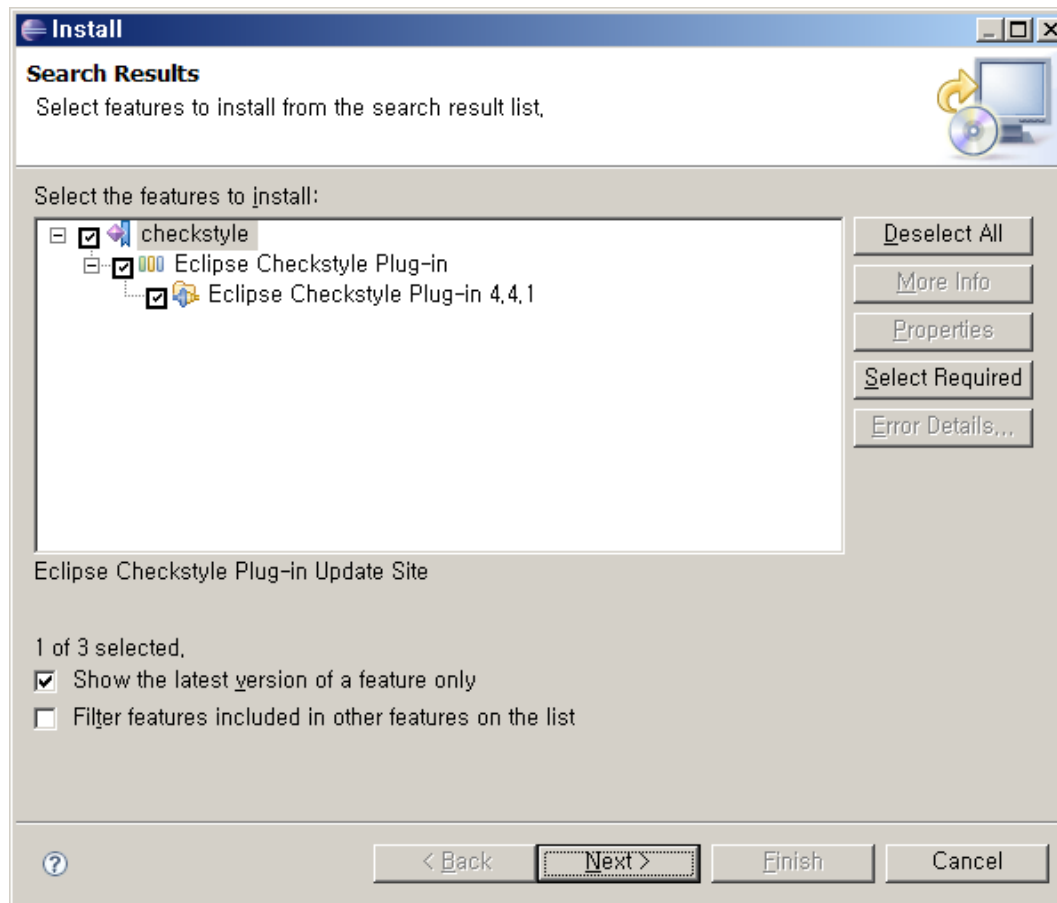
□ CheckStyle

- 코드 베이스에 대한 관리 능력은 소프트웨어 총 비용에 큰 영향을 미침
- 관리 능력은 개발자의 욕구 불만에도 기영
- 코드 변경이 쉬울수록, 새로운 제품 기능을 추가하기도 쉬움
- 코딩 표준 위반 사항을 찾는데 도움을 줌

CheckStyle - 코딩 표준 분석기

설치

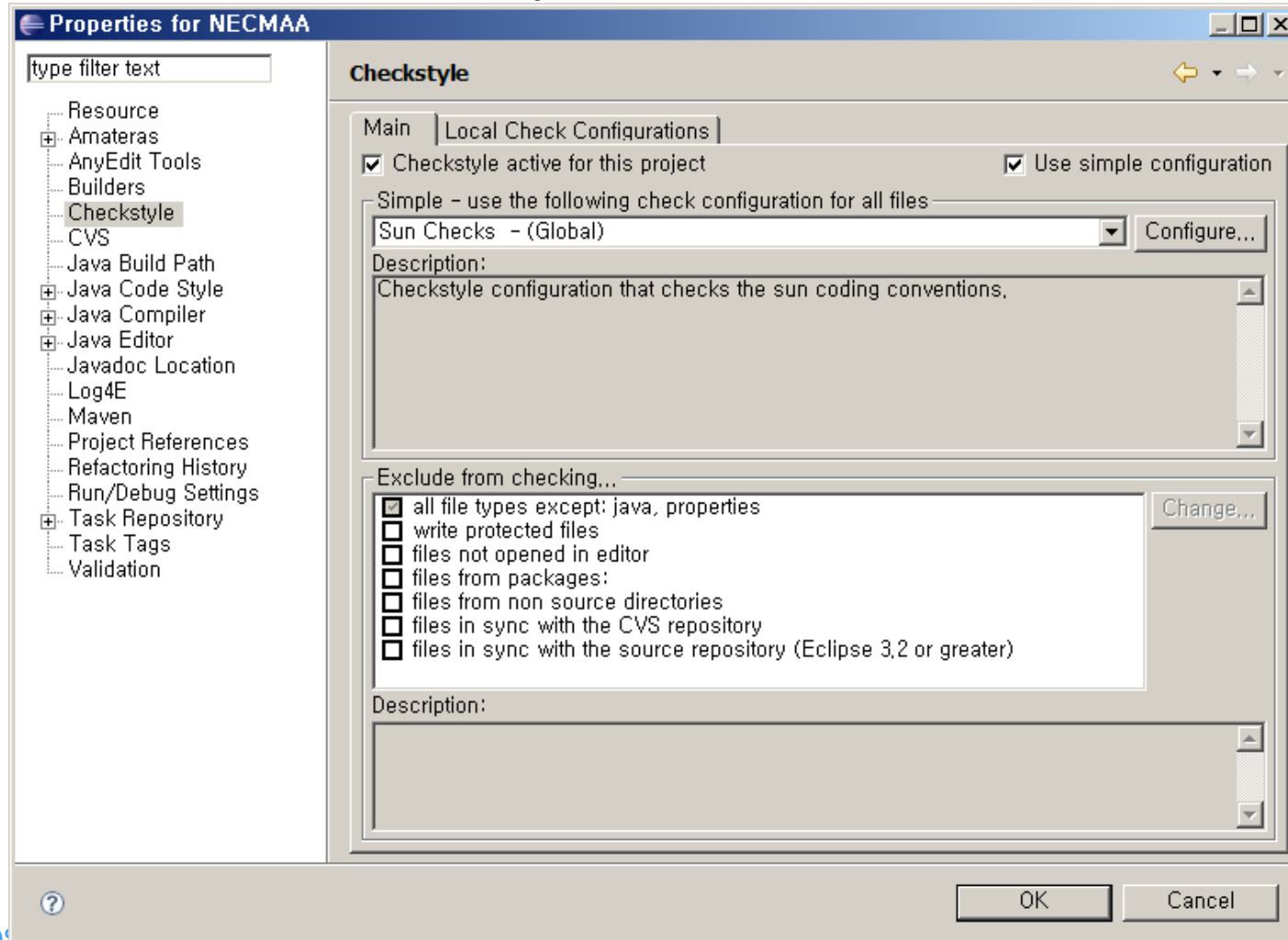
- Eclipse의 Software Update... 기능을 통해 설치
- <http://eclipse-cs.sourceforge.net/update>



CheckStyle - 코딩 표준 분석기

□ 실행

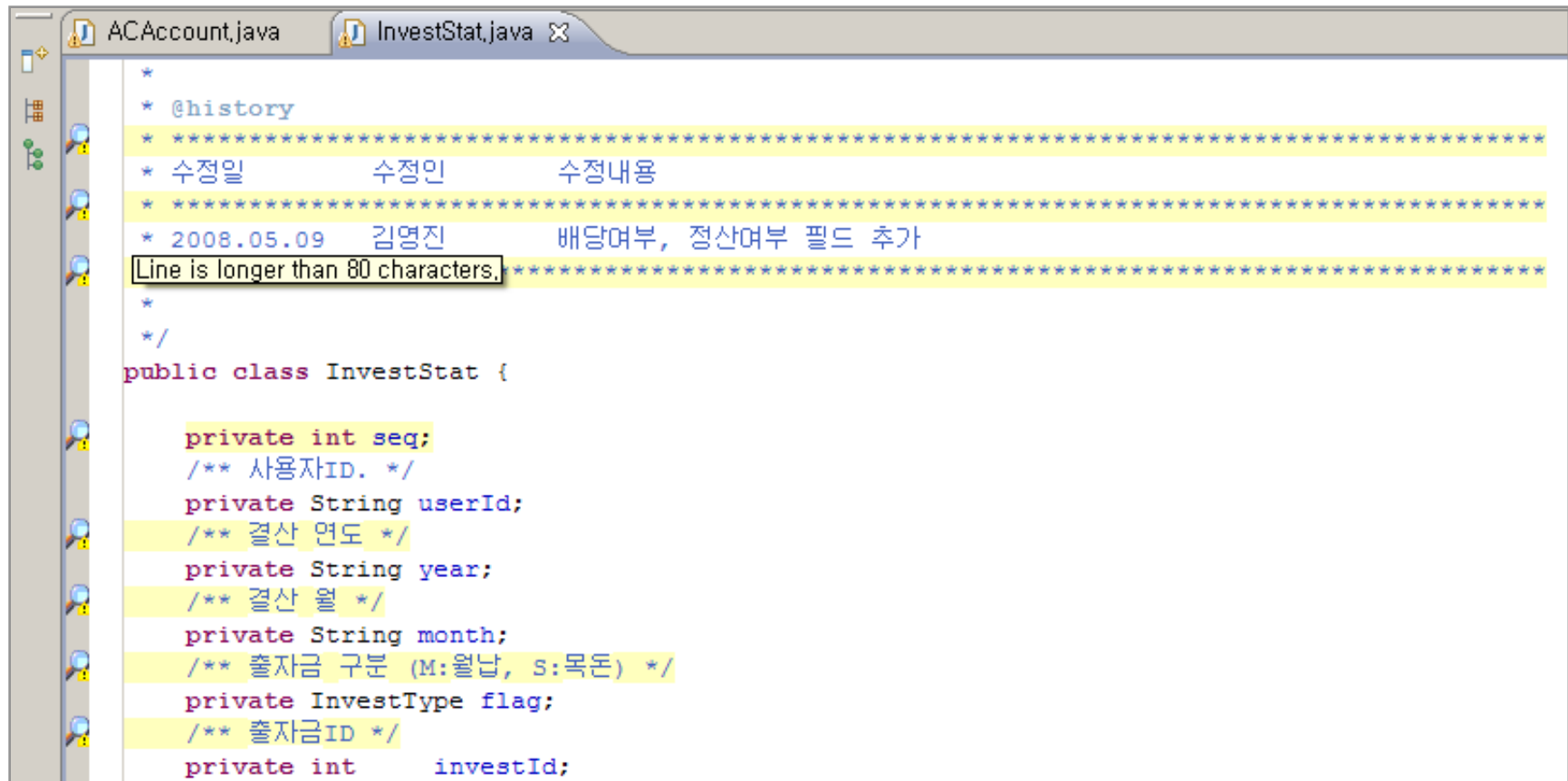
- 각 프로젝트에서 CheckStyle 사용을 활성화



CheckStyle - 코딩 표준 분석기

□ 실행

- 각 에디터에 검사 결과가 표시됨



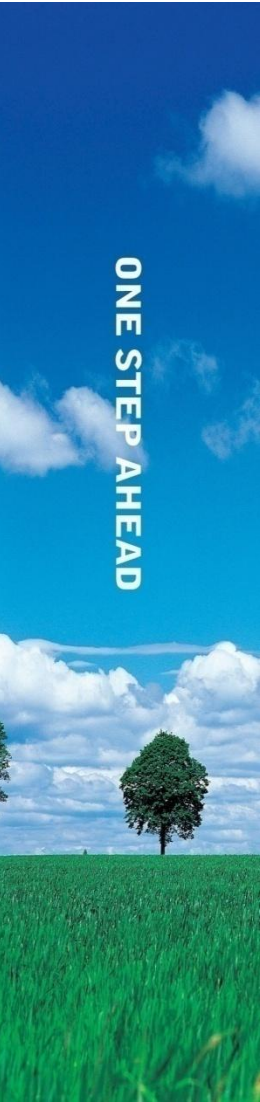
```

ACAccount.java  InvestStat.java ✕
*
* @history
* ****
* 수정일      수정인      수정내용
* ****
* 2008.05.09  김영진      배당여부, 정산여부 필드 추가
Line is longer than 80 characters.
*
*/
public class InvestStat {

    private int seq;
    /** 사용자ID. */
    private String userId;
    /** 결산 연도 */
    private String year;
    /** 결산 월 */
    private String month;
    /** 출자금 구분 (M:월납, S:목돈) */
    private InvestType flag;
    /** 출자금 ID */
    private int      investId;
    
```

4. JUnit 테스트 심화 I

- **POJO & 경량 프레임워크를 이용한 아웃-컨테이너 테스트**



POJO & 경량 프레임워크를 이용한 아웃-컨테이너 테스트

- ❑ J2EE 개발의 한계
- ❑ J2EE 개발 대안 – **POJO** + 경량 프레임워크의 사용
- ❑ **iBatis Basic**
- ❑ **POJO + iBatis**를 이용한 애플리케이션 테스트

J2EE 애플리케이션의 개발의 한계

□ EJB (Enterprise Java Bean)

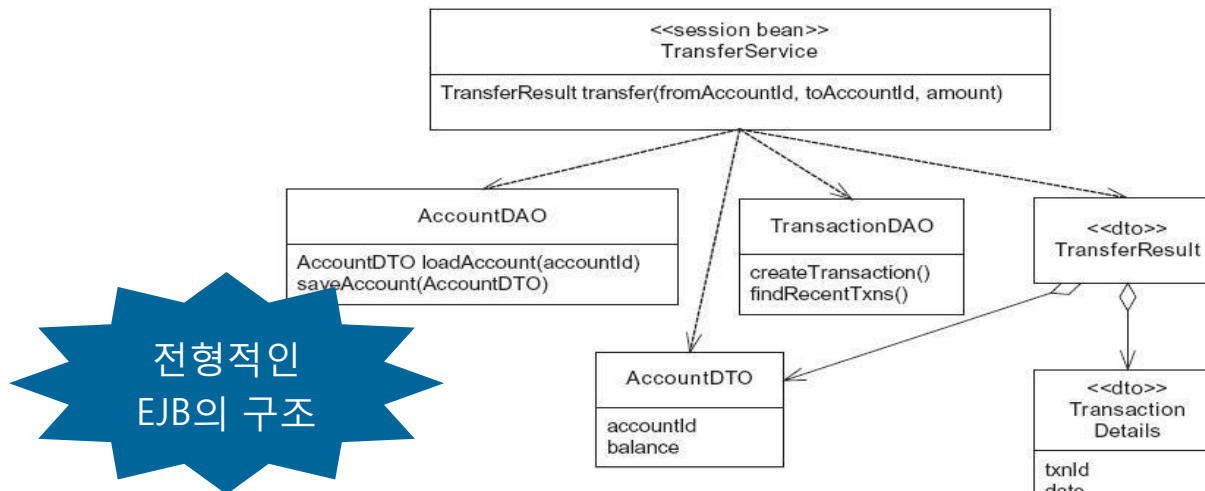
- 분산 비즈니스 애플리케이션을 위한 자바 표준 아키텍처
- 비즈니스 애플리케이션 개발 편의를 위한 많은 인컨테이너 서비스 제공
 - 트랜잭션 처리, 메시지 처리, 보안 매커니즘...
- 세션빈, 엔티티빈, 메시지 드리븐 빈으로 구성
- EJB3까지 릴리즈된 상태

□ EJB2 에 대한 환멸

- 적절한 환경에서 J2EE 아키텍처와 설계방식을 따르는 것이 좋을 수도 있음
- J2EE 개발 방식은 사고의 자유를 제한 함
- “애플리케이션을 쉽게 작성할 수 있도록 한다”는 목표 달성 실패

□ EJB2의 제약 사항

- 하나의 EJB를 만들기 위해 여러 개의 XML 배포 설명자를 생성해야 함
- 여러 종류의 소스파일을 만들어야 함
 - Remote Interface, Home Interface, Bean 클래스
- 개발자가 알 수 없는 여러 개의 콜백 메소드를 생성
- EJB 컨테이너 안에서만 실행이 가능
- 비즈니스 로직의 테스트가 용이하지 않음
- 객체 지향 설계가 용이하지 않음



J2EE 애플리케이션의 개발의 한계

□ 새로운 대안 **EJB3**

- EJB를 POJO로 작성 가능, 서버환경과 개발환경 분리
- 배치 기술자(Deployment Descriptor)를 어노테이션으로 대체가능
- 엔티티빈의 상속 가능
- 엔티티빈을 프리젠테이션 티어에서 반환 데이터 클래스로 사용 가능
 - DTO의 필요성 감소
- 엔티티 빈에 표준 자바 **Persistence** 매커니즘 사용

□ **EJB3**의 단점

- EJB 컴포넌트는 세션빈, 엔티티빈, 메시지-드리븐 빈에 속해야 함
- 이 범주를 벗어나면 **EJB** 컨테이너가 제공하는 서비스를 사용할 수 없음
- 쉽고 빠른 배포가 여전히 어려움
- Collection 류의 클래스 지원이 미비함

J2EE 개발 대안 – POJO + 경량 프레임워크의 사용

□ J2EE 개발 대안

- POJO와 경량 프레임워크를 결합
- EJB의 트랜잭션 처리, Persistence 관리같은 기능을 지원할 수 있음
- 객체지향 설계 및 개발 가능하게 됨

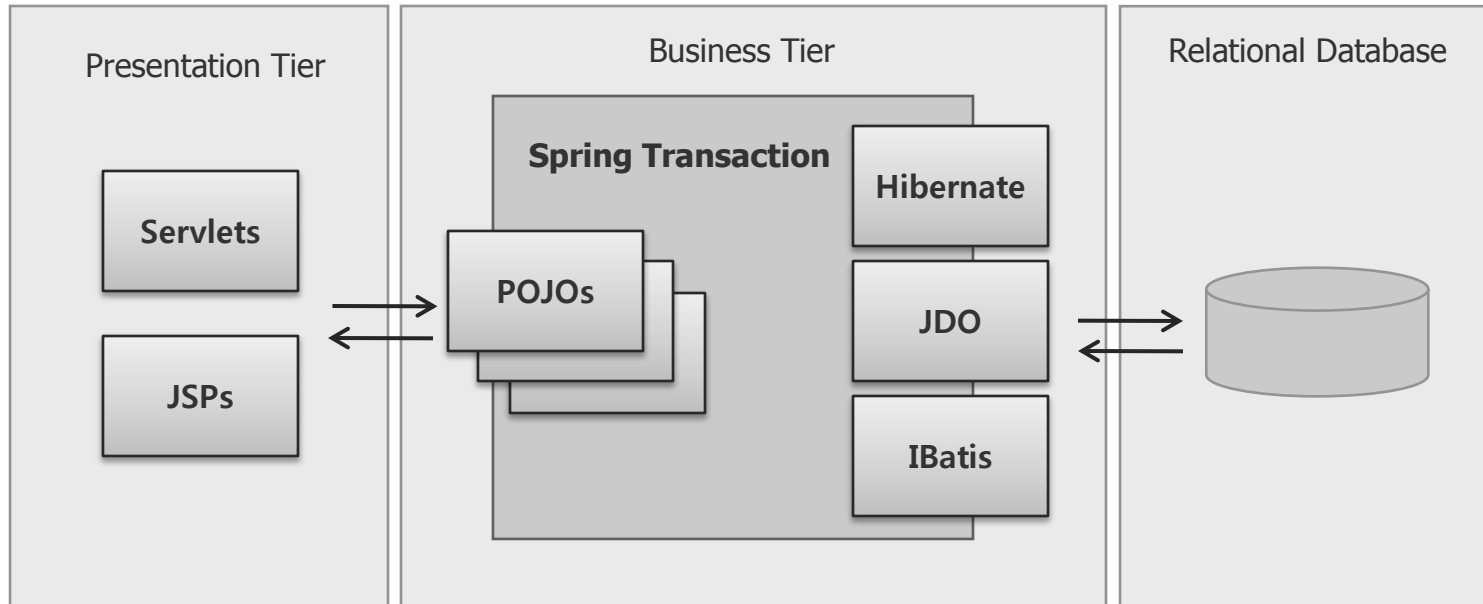
□ POJO(Plain Old Java Object)

- 그냥 자바 객체
- EJB 프레임워크에서 정의한 특별한 인터페이스를 포함하지 않음
- EJB 컨테이너나 데이터베이스 없이 개발 가능 - 개발이 쉽고 빨라짐
- 특정 구현 기술이 없매이지 않기 때문에 이식성 향상

□ 경량 프레임워크(Lightweight Framework)

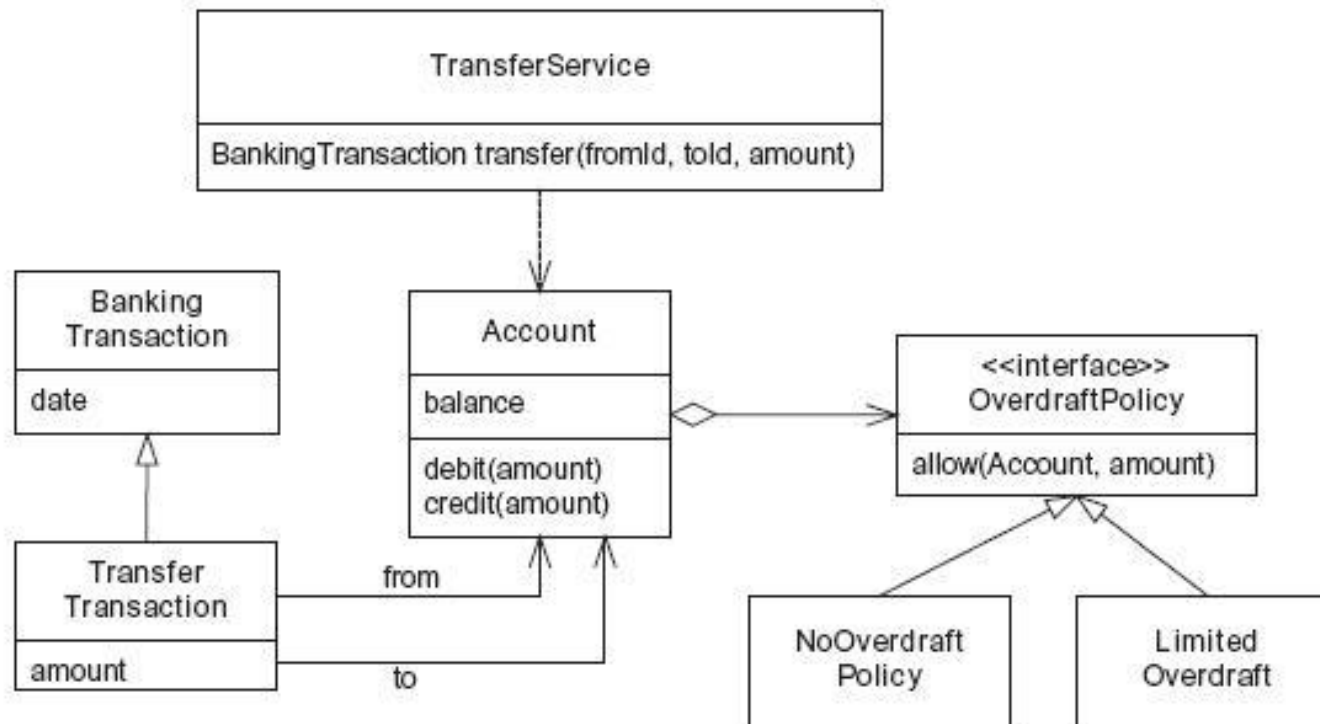
- EJB가 제공하는 서비스를 지원해 줄 수 있는 가벼운 프레임워크들
- Hibernate, JDO, **iBatis**, Spring

□ POJO + 경량 프레임워크



J2EE 개발 대안 - POJO + 경량 프레임워크

□ 객체 지향 프로그램이 가능해짐



Spring 이란? (1/4)

□ 오픈 소스 프레임워크

- Rod Johnson 창시
 - Expert one-on-one J2EE Design and Development, 2002, Wrox
 - Expert one-on-one J2EE Development without EJB, 2004, Wrox
- 엔터프라이즈 어플리케이션 개발의 복잡성을 줄여주기 위한 목적
- EJB 사용으로 수행되었던 모든 기능을 일반 POJO(Plain Old Java Object)를 사용해서 가능하게 함.
- www.springframework.org

□ 한 줄 정의

- **Spring**은 가벼운 의존 역행(*lightweight dependency injection*)과 관점 지향 컨테이너(*aspect-oriented container*) 및 프레임워크이다.



※ POJO의 다른 용어는 *plain-vanilla JavaBeans*

Spring 이란? (2/4)

□ Spring 이름의 기원

- Rod Johnson이 쓴 “Expert One-on-One J2EE Design and Development”라는 책에 Spring의 개념들이 모두 포함되어 있음.
- Wrox 출판사 포럼에서 활기찬 논의가 있는 와중에 Juergen Hoeller와 Yann Caroff와 함께 오픈 소스 프로젝트로 출발
- 전통적인 J2EE를 “겨울” (winter) 에 빗대어 “겨울”후의 “봄”으로 새로운 시작을 의미
- 2003년 6월 Spring 1.0 출시



※ 참조 : <http://blog.springsource.com/main/author/rodj/>

Spring 이란? (3/4)

□ 가벼움 (Lightweight)

- 크기와 처리 능력 측면에서의 가벼움
- 전체 2.5MB 분량의 단일 jar 파일
- Spring이 필요로 하는 처리 능력(성능)은 무시할 정도
- 비침투적(nonintrusive)
 - Spring 을 사용한 어플리케이션 내의 객체들은 Spring 과 아무런 의존관계 (dependency)가 없음.

□ 의존성 주입 (Dependency Injection)

- 의존성 주입 기술을 통해서 느슨한 결합(loose coupling) 유지
- 객체 스스로가 의존 객체를 생성하고 검색하는 대신에 수동적으로 의존관계가 주어짐.
- JNDI (Java Naming & Directory Interface)의 반대 개념
 - 컨테이너로부터 객체가 의존관계를 검색하는 대신에 컨테이너가 바로 초기화시에 객체에게 의존관계를 주입

Spring 이란? (4/4)

□ 관점 지향 (Aspect-oriented)

- 어플리케이션 비즈니스 로직과 시스템 서비스(감사 및 트랜잭션 관리 등)를 분리하는 개발 기법
- 어플리케이션 객체는 수행하고자 하는 비즈니스 로직만을 구현하며, 로깅이나 트랜잭션 관리와 같은 시스템 관심사는 전혀 책임이 없음.

□ 컨테이너 (Container)

- 어플리케이션 객체의 생명주기(lifecycle)와 설정을 담고 관리한다는 측면
- 어플리케이션 객체가 생성되는 방법, 설정되는 방법, 연관을 맺는 방법을 지정

□ 프레임워크

- 단순한 컴포넌트로부터 복잡한 어플리케이션을 설정하고 구성하는 것이 가능
- 어플리케이션 객체는 XML 파일 형태로 선언적으로(declaratively) 구성
- 인프라 기능 (트랜잭션 관리, 저장 프레임워크 통합 등)의 많은 부분을 제공

□ Hibernate

- 대표적인 자반기반 **OR 매핑** 도구
- 오픈소스, 상업적 이용가능 – **LGPL**
- 오라클, DB2, MySQL를 비롯한 대부분의 데이터베이스 지원
- 다양한 프레임워크와 쉽게 연동이 가능함
 - 스텔라츠, 스프링, 웹워크
- 문서화가 잘 되어있고, 커뮤니티가 활성화 되 있음
- 노가다성 구현이 없음 – **SQL** 코딩이 필요 없음
- 개발속도가 빠르며, 객체지향 접근방식

□ 단점

- OR 매핑 개념을 배우는데 시간이 걸림
- **Hibernate**가 생성하는 쿼리에 대한 튜닝이 어려움
- **JPA**방식으로 사용할 경우 데이터베이스 관리가 어려움

□ iBatis

- SQL이 등장하고 수십년이 지났지만 여전히 압도적인 사용
- 관계형 데이터베이스와 SQL를 가치를 인정
- 자바기반 프로젝트에서 효과적인 SQL의 사용방법에 대한 고민으로 탄생
- 자바기반 SQL 매퍼(SQL Mapper)
- SQL을 캡슐화하고 애플리케이션으로부터 분리
 - 비즈니스 로직과 데이터베이스 로직 분리
- ORM이 제공하는 대부분의 기능을 제공
 - 지연로딩, 조인해서 값 가져오기, 실시간 코드 생성, 상속
 - 어떠한 데이터 모델과 객체모델의 조합도 사용 가능

□ 단점

- OR Mapper에 비해서 이식성이 떨어짐
- SQL 작성하는 일 자체가 어렵고 지루한 작업임

iBatis 설치 및 구성

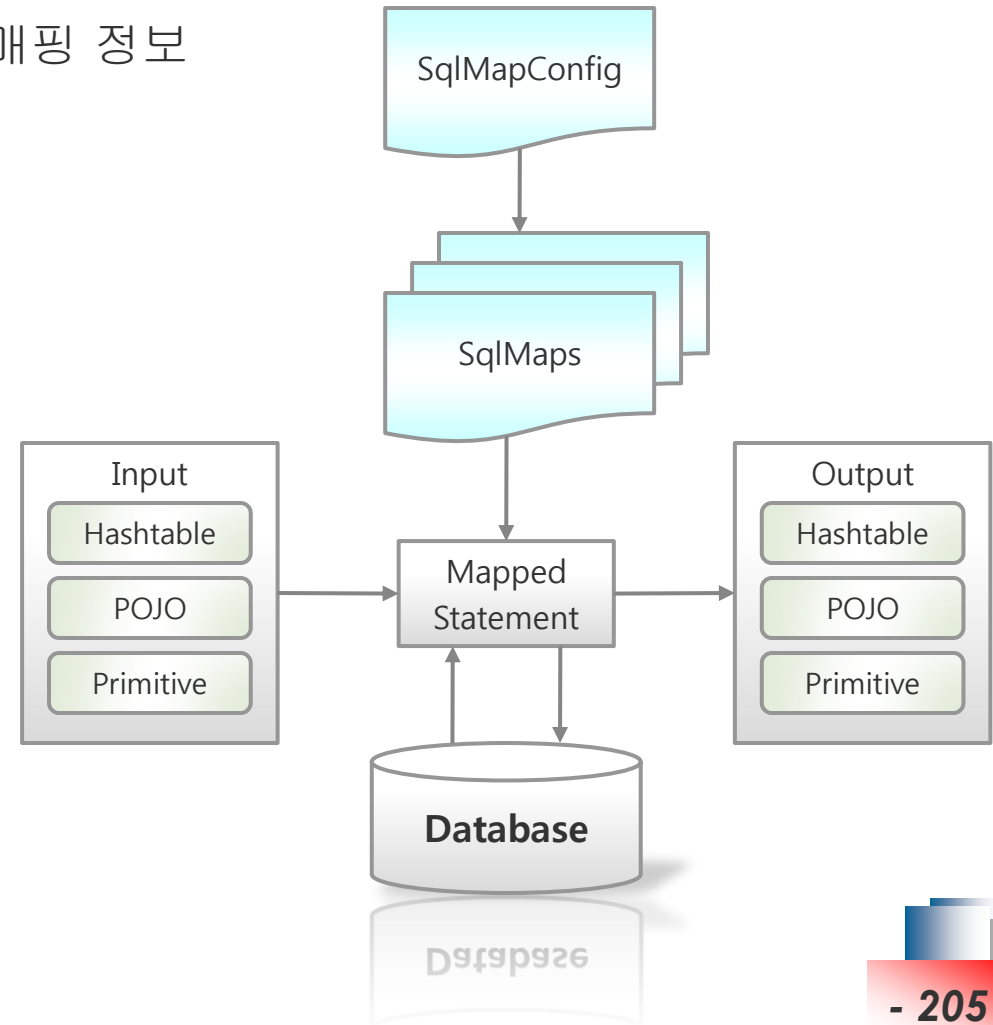
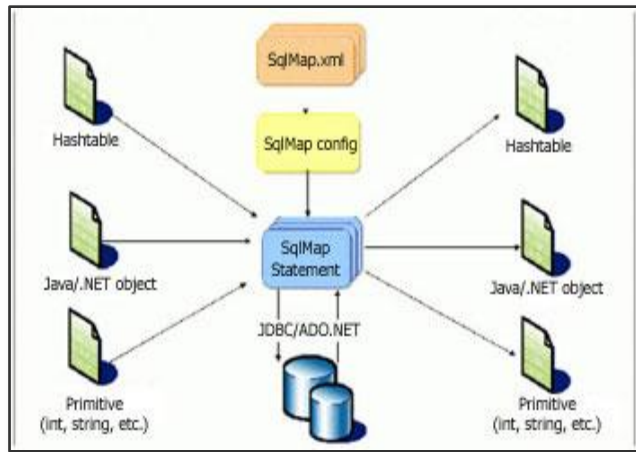
- ☐ iBatis 획득
- ☐ 배포버전 내용
- ☐ iBatis 설치
- ☐ iBatis configuration

□ 배포 버전 내용

File	목적
ibatis-2.3.0.677.jar	SQL Map 과 DAO Framework에서 공통으로 쓰이는 컴포넌트 SQL Map Framework 컴포넌트
user-javadoc.zip	Framework으로 작업하는 사용자들을 위한 제한된 버전의 문서
dev-javadoc.zip	iBatis 프로젝트 전체 문서 포함
ibatis-src.zip	Framework jar로 빌드하기위한 전체 소스코드 포함

❑ iBatis configuration 개념도

- SqlMapConfig : 전역 설정 및 트랜잭션 관리 정보
- SqlMap : 입출력값과 DB간의 매핑 정보



□ SQL Map 설정 파일(SqlMapConfig.xml)

- 중앙 허브의 역할
- DB connection 정보 제공

프로퍼티
(optional)

전역 설정 옵션

트랜잭션 관리
형태

SqlMap 파일 참조

```
<?xml version="1.0" encoding="EUC-KR"?>
<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
  <properties resource="sqlmap-config.properties"/>
  <settings useStatementNamespaces="false" cacheModelsEnabled="true"
    enhancementEnabled="true" lazyLoadingEnabled="true"
    maxRequests="${maxRequests}" maxSessions="${maxSessions}"
    maxTransactions="${maxTransactions}" />

  <transactionManager type="${transactionManager}">
    <dataSource type="${dataSource}">
      <property name="DataSource" value="${dsName}" />
      <property name="JDBC.Driver" value="${driver}" />
      <property name="JDBC.ConnectionURL" value="${url}" />
      <property name="JDBC.Username" value="${user}" />
      <property name="JDBC.Password" value="${pword}" />
    </dataSource>
  </transactionManager>

  <sqlMap resource="kr/nextree/nexbay/da/ibatis/maps/User.xml" />
</sqlMapConfig>
```

iBatis configuration

□ properties 요소

- resource 속성 : classpath상의 리소스
- url 속성 : url 상의 리소스

sqlmap-config.properties

```
maxRequests=32
maxSessions=10
maxTransactions=5
transactionManager=JDBC
dataSource=SIMPLE
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/nexbay
user=nexbay
pword=nexbay
```

```
...
<sqlMapConfig>
  <properties resource="sqlmap-config.properties"/>
  <settings useStatementNamespaces="false" cacheModelsEnabled="true"
    enhancementEnabled="true" lazyLoadingEnabled="true"
    maxRequests="${maxRequests}" maxSessions="${maxSessions}"
    maxTransactions="${maxTransactions}" />

  <transactionManager type="${transactionManager}">
    <dataSource type="${dataSource}">
      <property name="DataSource" value="${dsName}" />
      <property name="JDBC.Driver" value="${driver}" />
      <property name="JDBC.ConnectionURL" value="${url}" />
      <property name="JDBC.Username" value="${user}" />
      <property name="JDBC.Password" value="${pword}" />
    </dataSource>
  </transactionManager>

  <sqlMap resource="kr/nextree/nexbay/da/ibatis/maps/User.xml" />
</sqlMapConfig>
```

□ settings 요소

- 설정되는 속성들은 SQL Map에 전역적으로 영향을 줌
- 속성 내용

범위	내용
lazyLoadingEnabled	데이터를 꼭 필요할 때에만 로딩함
cacheModelsEnabled	iBatis가 캐싱을 할지 결정함. 캐싱에 대한 설정이 SQL Map에 되어 있는 경우 영향을 받는다.
useStatementNamespaces	true일 경우 각 SQL Map마다 고유의 네임스페이스를 가지게 되어 mapping문의 중복을 방지한다.
maxRequests	insert,update,delete 또는 프로시저 call과 같은 SQL 요청
maxSessions	관련된 트랜잭션 또는 요청의 그룹과 같은 스레드 레벨의 세션
maxTransactions	최대 트랜잭션 수

□ typeAlias 요소

- alias로 type명을 대체
- iBatis는 정의된 alias로 언제든지 type에 접근 가능

```
<typeAlias alias="Category"  
            type="kr.nexttree.nexbay.domain.object.Category"/>
```

- 미리 정의된 typeAlias
 - Transaction manager : JDBC, JTA, EXTERNAL
 - Data types : string, long, int, date 등
 - Data source factory : SIMPLE, DBCP, JNDI
 - Cache controller : FIFO, LRU, MEMORY, OSCACHE
 - XML result : Dom, domCollection, Xml, XmlCollection

□ typeAlias 요소(계속)

▪ 주요 typeAlias

Alias	Type
JDBC	com.ibatis.sqlmap.engine.transaction.jdbc.JdbcTransactionConfig
JTA	com.ibatis.sqlmap.engine.transaction.jta.JtaTransactionConfig
EXTERNAL	com.ibatis.sqlmap.engine.transaction.external.ExternalTransactionConfig
string 외	java.lang.String 외 다수
SIMPLE	com.ibatis.sqlmap.engine.datasource.SimpleDataSourceFactory
DBCP	com.ibatis.sqlmap.engine.datasource.DbcpDataSourceFactory
JNDI	com.ibatis.sqlmap.engine.datasource.JndiDataSourceFactory
FIFO	com.ibatis.sqlmap.engine.cache.fifo.FifoCacheController
LRU	com.ibatis.sqlmap.engine.cache.lru.LruCacheController
MEMORY	com.ibatis.sqlmap.engine.cache.memory.MemoryCacheController
OSCACHE	com.ibatis.sqlmap.engine.cache.oscache.OSCacheController

□ transactionManager 요소

- 미리 정의된 Transaction Manager 타입
 - JDBC : 단순 JDBC 기반의 Transaction Manager를 제공함
 - JTA : application이 동작하는 컨테이너 기반의 Transaction Manager를 제공함
 - EXTERNAL : 트랜잭션 관리를 iBatis에서 하지 않음
- Transaction Manager가 사용하는 Data Source Factory 지정
 - SIMPLE : 단순 connection pool 생성
 - DBCP : Jakarta Commons Database Connection Pool 구현함
 - JNDI : 컨테이너 기반의 data source를 iBatis에서 사용함

□ typeHandler 요소

- JDBC 데이터 타입과 application 데이터 타입 변환을 위하여 사용함



- iBatis에서 미리 정의된 typeHandler로 대부분 적용 가능
- 사용자정의 typeHandler를 만들 경우 어떻게 그리고 언제 사용될 것인가 명

어떻게
언제

```

<typeHandler
  callback="kr.nexttree.nexbay.da.ibatis.UserTypeHandlerCallBack"
  javaType="kr.nexttree.nexbay.domain.object.UserType" jdbcType="VARCHAR"/>
    
```


❑ sqlMap 요소

- SQL Map 파일의 위치 명시
- resource 속성 : class path 상의 SQL Map 파일
- url 속성 : java.net.URL 클래스로 파일 위치 결정

Working with mapped statements

- ❑ **iBatis의 JavaBean**
- ❑ **SqlMap API**
- ❑ **SqlMap의 Mapping문**
- ❑ **<select> Mapping문**
- ❑ **Parameter 매핑**
- ❑ **Result 매핑**

□ iBatis의 Java property 접근

- JavaBean 방식의 property 접근
- 1쌍의 접근 메소드가 필요함

```
public String getName();  
public void setName(String name);
```

- property 타입이 boolean일 경우

```
public boolean isStudent();  
public void setStudent(boolean isStudent);
```

- Bean 네비게이션

Java code	Dot 표기법
anOrder.getAccount().getUsername()	anOrder.account.username
anOrder.getOrderItem().get(0).getProductId()	anOrder.orderItem[0].productId
anObject.getID()	anObject.ID
anObject.getxCoordinate()	anObject.xCoordinate

❑ SqlMapClient 인터페이스

- 가장 많이 사용하는 인터페이스
- 30여개의 메소드 존재

❑ queryForObject() 메소드

- DB로 부터 1개의 row를 가져올 때 사용

```
Object queryForObject(String id, Object parameter) throws SQLException;  
Object queryForObject(String id, Object parameter, Object result) throws SQLException;
```

- 첫번째 메소드는 SQL Map에 정의된 select문 id와 해당 파라미터로 사용
- 두번째 메소드는 결과 객체 생성이 어려운 경우 사용
- 1개 이상의 row가 리턴 된 경우 Exception 발생

□ queryForList() 메소드

- DB로 부터 1개 또는 그 이상의 row가 리턴될 경우 사용
- List 형태로 리턴함

```
List queryForList(String id, Object parameter) throws SQLException;  
List queryForList(String id, Object parameter, int skip, int max) throws SQLException;
```

- 첫번째 메소드는 사용법이 앞과 동일
- 두번째 메소드는 DB에서 가져온 전체 row중 일부만 사용하고 싶은 경우 사용
 - 예) DB에서 100건을 가져올 때 이 중 앞의 10건만 취하고 싶으면 skip=0, max=10

□ queryForMap() 메소드

- 여러 건을 Map 형태로 리턴함

```
Map queryForMap(String id, Object parameter, String key) throws SQLException;  
Map queryForMap(String id, Object parameter, String key, String value) throws SQLException;
```

- 첫번째 메소드의 경우
 - key는 유일함을 결정할 수 있는 property 명을 입력
 - 결과 Map의 value는 완전한 객체로 이루어짐
- 두번째 메소드의 경우
 - Map의 value로 들어갈 property 명을 입력
 - 결과 Map의 value는 지정한 property로 이루어짐

queryForMap() 사용예

```
Map accountMap = sqlMap.queryForMap("Account.getAll", null, "accountId");  
Map accountMap = sqlMap.queryForMap("Account.getAll", null, "accountId", "username");
```

□ SqlMap Mapping문이란?

- 각 SqlMap파일(xml)에 정의함
- select, update, insert, delete 등을 수행함
- SqlMap API에서 id로 접근 가능

□ Mapping문의 종류

Statement Type	속성	하위 요소	용도
<select>	id, parameterClass, resultClass, parameterMap, cacheModel	모든 dynamic 요소	조회
<insert>	id, parameterClass, parameterMap	모든 dynamic 요소 selectKey	입력
<update>	id, parameterClass, parameterMap	모든 dynamic 요소	수정
<delete>	id, parameterClass, parameterMap	모든 dynamic 요소	삭제

SqlMap의 Mapping문

□ Mapping을 수행하지 않는 Mapping문

Statement Type	속성	하위 요소	용도
<sql>	id	모든 dynamic 요소	매핑을 수행하지 않지만 매핑 가능한 매핑문에서 사용된다.
<include>	refid	모든 dynamic 요소	매핑을 수행하지 않지만 <sql>에서 정의된 것을 매핑문에 추가시켜준다.

```

<sql id="select-order">
  select * from order
</sql>
<sql id="select-count">
  select count(*) as value from order
</sql>
<sql id="where-shipped-after-value">
  <![CDATA[where shipDate > #value:DATE#]]>
</sql>
<select id="getOrderShippedAfter" resultClass="mp">
  <include refid="select-order" />
  <include refid="where-shipped-after-value" />
</select>
<select id="getOrderCountShippedAfter" resultClass="int">
  <include refid="select-count" />
  <include refid="where-shipped-after-value" />
</select>

```

모든 주문 column

주문 횟수

특정일 이후 주문

특정일 이후의 모든 주문 column

특정일 이후의 주문 횟수

〈select〉 Mapping문

□ ‘#’ 지시자로 인라인 **parameter** 사용

- SqlMap API로 입력된 파라미터의 매핑 가능
- 간접적 매핑방식으로 String일 경우 ‘ ’ 생성(LIKE Query 문에서 적용 어려움)

```
<select id="selectUser" resultMap="userMap">
  select
    user.USER_ID as userId,
    user.PASSWORD as password,
    user.NAME as name,
    user.EMPLOYEE_NO as employeeNo,
    user.EMAIL as email,
    user.TYPE as userType,
    user.CREDIT as credit,
    user.PHONE_NUM as phoneNumber,
    user.ZIP_CODE as zipCode,
    user.ADDR1 as address1,
    user.ADDR2 as address2,
    user.GRADE as grade
  from USER_TB user
  where user.USER_ID = #value#
</select>
```

```
...
where user.USER_ID = 'tsong';
```

```
User user = (User) client.queryForObject("selectUser", userId);
```

〈select〉 Mapping문

□ '\$' 지시자로 인라인 parameter 사용

- SqlMap API로 입력된 파라미터를 직접적으로 매핑
- LIKE Query 문에서 적용 가능

```
<select id="selectUser" resultMap="userMap">
  select
    user.USER_ID as userId,
    user.PASSWORD as password,
    user.NAME as name,
    user.EMPLOYEE_NO as employeeNo,
    user.EMAIL as email,
    user.TYPE as userType,
    user.CREDIT as credit,
    user.PHONE_NUM as phoneNumber,
    user.ZIP_CODE as zipCode,
    user.ADDR1 as address1,
    user.ADDR2 as address2,
    user.GRADE as grade
  from USER_TB user
  where user.NAME like '%$value$%'
</select>
```

```
...
where user.NAME like '%송%';
```

```
List<User> allUsers = client.queryForList("selectAllUsers", nameKey);
```

〈select〉 Mapping문

□ 자동 Result Map

- select문 실행 결과를 자동으로 객체에 매핑 가능함
- 단일 column 매핑
 - accountId 값이 Integer 타입으로 자동 매핑된다.

```
<select id="getAllAccountIdValues" resultClass="int">
    select accountId as value
    from Account
</select>
```

```
List<Integer> allAccountIds =
    client.queryForList("getAllAccountIdValues", null);
```

- 다중 column 매핑
 - select 문의 column명과 결과 객체의 Bean property 명이 일치해야 함

```
<select id="selectAllCategories" resultClass="Category">
    select
        cat.CAT_ID as id,
        cat.CAT_NAME as name,
        cat.CAT_DESC as description
    from CATEGORY_TB cat
</select>
```

Parameter 매핑

□ Parameter 매핑의 두가지 방안

- 인라인 매핑
 - SqlMap Mapping문 내부에서 바로 기술함
 - 매핑이 복잡한 경우 명시적이지 못함
- 외부 매핑
 - 외부에서 정의됨
 - 보다 명시적임

인라인 매핑의 예

```
<insert id="createCategory" parameterClass="Category">
    insert into CATEGORY_TB (
        CAT_ID,
        CAT_NAME,
        CAT_DESC
    )
    values (
        #id#,
        #name#,
        #description#
    )
</insert>
```

외부 매핑의 예

```
<parameterMap id="userAdminParam" class="Admin">
    <parameter property="userId"/>
    <parameter property="password"/>
    <parameter property="name"/>
    <parameter property="employeeNo"/>
    <parameter property="email"/>
    <parameter property="userType"
        javaType="UserType" jdbcType="VARCHAR"/>
</parameterMap>
```

Parameter 매핑

□ 외부 Parameter 매핑

속성	설명
property	매핑문에 전달할 JavaBean property 명칭 또는 Map value의 key. 들어갈 이름을 매핑문에 필요한 만큼 반복해서 정의할 수 있음. 예) update 문에서 set 절과 where절에 동일한 이름이 반복적으로 들어갈 수 있음.
javaType	세팅될 parameter의 Java property 타입을 명시적으로 정의
jdbcType	세팅될 parameter의 데이터베이스 타입을 명시적으로 정의 예) Java 타입이 Date(java.util.Date)인 경우 jdbcType을 DATE인지 DATETIME인지 명시할 필요가 있음
nullValue	nullValue에 명시된 값이 JavaBean property에서 넘어오면 데이터베이스에 null을 기록함 예) 매직 null number(-9999)가 넘어온 경우에 데이터베이스에 null을 기록함
typeHandler	javaType 및 jdbcType이 일치되는 경우 typeHandler가 적용되나 그것 없이 직접 typeHandler를 지정할 수 있음

- 외부 Parameter 매핑이 유용한 경우
 - 인라인 Parameter 매핑이 잘 동작하지 않을 경우
 - 성능을 향상시킬 경우
 - 명시적 매핑을 할 경우

Parameter 매핑

□ 인라인 Parameter 매핑

- 인라인 Parameter에 데이터베이스 타입 명시
 - #value:[jdbcType]#
 - #value,jdbcType=[jdbcType]#

□ Primitive Parameter

- int 또는 long 같은 Primitive 타입은 iBatis에서 자동으로 Integer, Long으로 래핑됨

□ JavaBean과 Map Parameter의 차이점

- JavaBean : property가 정해져 있으므로 load time에 매핑 오류 발견 가능
- Map : property가 Runtime에 정해지므로 load time에 매핑 오류 발견 불가

□ 명시적 **Result** 매핑

속성	설명
property	결과 객체의 JavaBean property 또는 Map 내용
column	JDBC ResultSet의 column
columnIndex	ResultSet의 column명 대신 index를 주어 결과 값 매핑이 가능. 약간의 성능 향상이 있으며 필수 사항은 아님.
jdbcType	ResultSet column의 타입을 명시함. java.util.Date의 경우 해당되는 jdbcType이 여러가지(DATE, DATETIME)이므로 정확한 매핑을 위해 명시하는 것이 좋음. JDBC driver에 따라 정의할 필요가 없는 것도 있음.
javaType	세팅될 Java 타입을 명시함.
nullValue	데이터베이스에서 null이 넘어온 경우 세팅할 값 지정
select	객체 관계를 표현하며 복잡한 property 타입을 로드 할 수 있음 들어갈 값은 반드시 다른 매핑문의 이름이어야 함.

□ Primitive result

- iBatis는 primitive 형태의 결과를 허용하지 않음(int, long, double, ..)
- 래핑된 형태의 타입을 리턴(Integer, Long, Double, ..)

```
<select id="getAllOrderCount" resultClass="int">
    select count(*) as value
    from order
</select>
```

```
Integer allCount =
    client.queryForObject("getAllOrderCount", null);
```

- JavaBean 타입으로 래핑 할 경우 primitive 형태의 결과를 얻을 수 있음

```
public class PrimitiveResult {
    private int orderCount;
    public int getOrderCount() {
        return orderCount;
    }
    public void setOrderCount(int orderCount) {
        this.orderCount = orderCount;
    }
}
```

```
<resultMap id="primitiveResultMapExample" class="PrimitiveResult">
    <result property="orderCount" column="orderCount" />
</resultMap>
```


□ JavaBean과 Map Result의 장단점

방식	장점	단점
Bean	성능 향상 컴파일 타임의 이름 체크 IDE에서 Refactoring 지원 가능 Type casting이 적음	get, set메소드 필요
Map	코드량이 적음	느리다 컴파일 타임 체크 불가 런타임 오류가 많음 Refactoring 지원이 없음

- ❑ 데이터 업데이트 구축하기
- ❑ 입력 데이터(**insert**)

□ Nonquery SQL문의 SqlMap API

- insert, update, delete 메소드 제공
- insert 메소드

```
Object insert(String id, Object parameterObject) throws SQLException;
```

입력 후 key를 리턴하는 용도로 드물게 사용된다. 특별한 설정이 필요

- update 메소드

```
int update(String id, Object parameterObject) throws SQLException;
```

영향을 받은 row 수를 리턴 (JDBC driver가 지원하는 경우)

- delete 메소드

```
int delete(String id, Object parameterObject) throws SQLException;
```

삭제된 row 수를 리턴

입력 데이터(insert)

□ 인라인 Parameter 사용

- 빠르게 insert 매핑문 작성 가능
- 가독성이 떨어짐
- Parameter 매핑정보가 비슷한 query에서 중복됨

```
<insert id="insertWithInlineInfo">
  insert into account (
    accountId,
    username, password,
    memberSince,
    firstName, lastName,
    address1, address2,
    city, state, postalCode,
    country, version
  ) values (
    #accountId:NUMBER#,
    #username:VARCHAR#, #password:VARCHAR#,
    #memberSince:TIMESTAMP#,
    #firstName:VARCHAR#, #lastName:VARCHAR#,
    #address1:VARCHAR#, #address2:VARCHAR#,
    #city:VARCHAR#, #state:VARCHAR#, #postalCode:VARCHAR#,
    #country:VARCHAR#, #version:NUMBER#
  )
</insert>
```

```
Account account = new Account();
account.setAccountId(new Integer(9999));
account.setUsername("inlineins");
account.setPassword("poohbear");
account.setFirstName("Inline");
account.setLastName("Example");
sqlMapClient.insert("Account.insertWithInlineInfo", account);
```

입력 데이터(insert)

□ 외부 Parameter Map 사용

- 성능 향상
- Load 타임에 이름 체크 가능
- 성능이나 유지보수면에서 인라인 Parameter 방식보다 나음.

```
<parameterMap id="fullParameterMapExample" class="Account">
  <parameter property="accountId" jdbcType="NUMBER" />
  <parameter property="username" jdbcType="VARCHAR" />
  <parameter property="password" jdbcType="VARCHAR" />
  <parameter property="memberSince" jdbcType="TIMESTAMP" />
  <parameter property="firstName" jdbcType="VARCHAR" />
  <parameter property="lastName" jdbcType="VARCHAR" />
  <parameter property="address1" jdbcType="VARCHAR" />
  <parameter property="address2" jdbcType="VARCHAR" />
  <parameter property="city" jdbcType="VARCHAR" />
  <parameter property="state" jdbcType="VARCHAR" />
  <parameter property="postalCode" jdbcType="VARCHAR" />
  <parameter property="country" jdbcType="VARCHAR" />
  <parameter property="version" jdbcType="NUMBER" />
</parameterMap>
```

```
<insert id="insertWithInlineInfo"
  parameterMap="fullParameterMapExample">
  insert into account (
    accountId,
    username, password,
    memberSince,
    firstName, lastName,
    address1, address2,
    city, state, postalCode,
    country, version
  ) values (
    ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
  )
</insert>
```

입력 데이터(insert)

□ 자동생성키

- 거의 대부분의 데이터베이스는 키의 자동생성 방안을 제공
- iBatis는 자동생성키를 Object로 리턴하는 <selectKey> 요소를 제공함
- <selectKey>는 <insert>의 하위요소임.

□ 입력시 생성된 **primary key**를 받는 두가지 방안

- 입력 후 key를 가지고 오는 방법
 - 동시 입력 문제 발생 가능
- key를 먼저 가지고 온 후 입력하는 방법
 - 동시 입력 문제는 없지만 입력시 많은 작업이 수반됨

```
<insert id="insert">
  INSERT INTO Account (
    username, password
  ) VALUES (
    #username#, #password#
    <selectKey keyProperty="accountId" resultClass="int">
      SELECT SCOPE_IDENTITY()
    </selectKey>
  </insert>
```

```
<insert id="insertSequence">
  <selectKey keyProperty="accountId" resultClass="int">
    SELECT nextVal(#sequence#)
  </selectKey>
  INSERT INTO Account (
    accountId, username, password
  ) VALUES (
    #accountId#, #username#, #password#
  </insert>
```

□ Stand-alone Application

- 어플리케이션 start 스크립트에 클래스 패스 정의
- `java -cp ibatis-2.3.0.677.jar:... MyMainClass`

□ Web Application

- WEB-INF/lib에 추가
- Web 컨테이너의 다른 경로에 추가하는 경우 오작동 발생 가능
 - 예) shared/lib, common/lib
 - 클래스 로더가 로드시 ibatis config 파일을 못찾는 문제 발생

❑ MySQL DB 설정

- MYSQL Command Line Client 실행
- root 계정으로 접속

```
mysql> create database ibatis_start;  
Query OK, 1 row affected (0.03 sec)
```

```
mysql> grant all privileges on *.* to ibatisuser@"%" identified by 'ibatisuser' with grant option;  
Query OK, 0 rows affected (0.03 sec)
```

```
mysql>
```

→ 'ibatis_start' 데이터베이스 생성

→ 'ibatisuser' 계정 생성

□ 테이블 생성 및 초기 데이터 입력

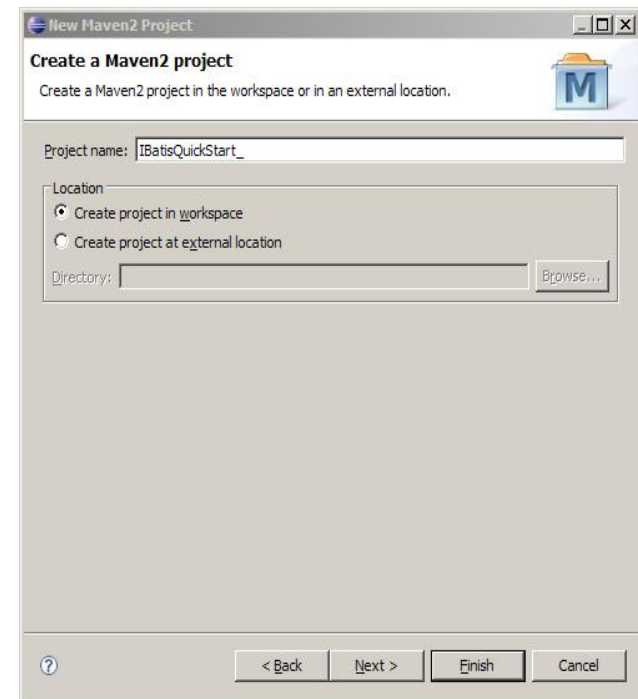
```
CREATE TABLE `user_tb` (  
  `USER_ID` varchar(10) NOT NULL,  
  `PASSWORD` varchar(128) NOT NULL,  
  `NAME` varchar(128) NOT NULL,  
  `EMAIL` varchar(128) NOT NULL,  
  `TYPE` varchar(10) NOT NULL,  
  PRIMARY KEY (`USER_ID`),  
  UNIQUE KEY `USER_UN` (`USER_ID`)  
) ENGINE=InnoDB DEFAULT CHARSET=euckr;
```

```
INSERT INTO USER_TB VALUES (  
  'tsong', '1234', '송태국',  
  'tsong@nextree.co.kr', 'Admin'  
);  
  
INSERT INTO USER_TB VALUES (  
  'syhan', '1234', '한성영',  
  'syhan@nextree.co.kr', 'Member'  
);
```

```
mysql> use ibatis_start;  
Database changed  
mysql> CREATE TABLE `user_tb` (  
  -> `USER_ID` varchar(10) NOT NULL,  
  -> `PASSWORD` varchar(128) NOT NULL,  
  -> `NAME` varchar(128) NOT NULL,  
  -> `EMAIL` varchar(128) NOT NULL,  
  -> `TYPE` varchar(10) NOT NULL,  
  -> PRIMARY KEY (`USER_ID`),  
  -> UNIQUE KEY `USER_UN` (`USER_ID`)  
  -> ) ENGINE=InnoDB DEFAULT CHARSET=euckr;  
Query OK, 0 rows affected (0.06 sec)  
  
mysql>  
mysql> INSERT INTO USER_TB VALUES (  
  -> 'tsong', '1234', '송태국', 'tsong@nextree.co.kr', 'Admin'  
  -> );  
Query OK, 1 row affected (0.00 sec)  
  
mysql> INSERT INTO USER_TB VALUES (  
  -> 'syhan', '1234', '한성영', 'syhan@nextree.co.kr', 'Member'  
  -> );  
Query OK, 1 row affected (0.00 sec)
```

□ Eclipse 프로젝트 생성

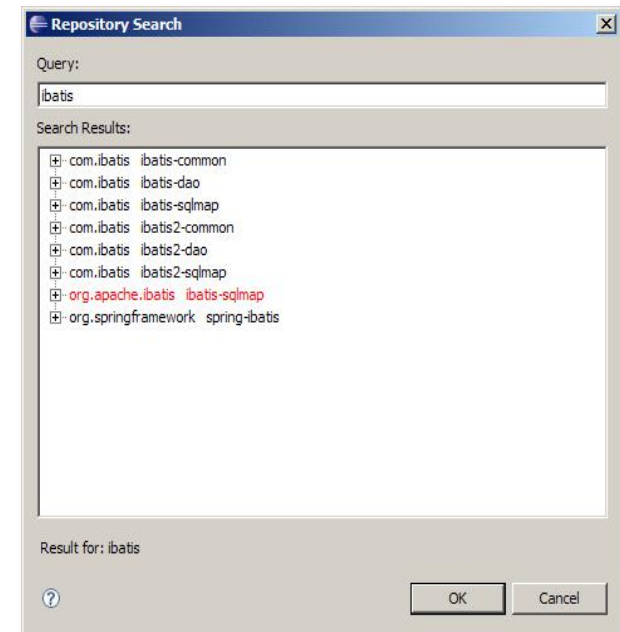
- New > Other... > Maven Project
- 프로젝트명 'IBatisQuickStart' 입력 후 Finish



□ Eclipse 프로젝트 생성(계속)

■ 클래스 패스 설정

- IBatisQuickStart 프로젝트 선택 후 오른쪽 마우스 클릭
- Maven > Add Dependency 선택
- iBatis 2.3 추가
- MySql JDBC Driver 5.X 추가
- Junit 추가



□ 도메인 클래스 작성

User.java

```
package kr.nexttree.ibatisstart.domain;

public class User {
    private String userId;
    private String password;
    private String name;
    private String email;
    private String userType;
    // getter, setter(계속)
```

```
// getter, setter
public String getUserId() {
    return userId;
}

public void setUserId(String userId) {
    this.userId = userId;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getUserType() {
    return userType;
}

public void setUserType(String userType) {
    this.userType = userType;
}
}
```

□ iBatis SqlMap Config(sqlmap-config.xml) 작성

```
<?xml version="1.0" encoding="EUC-KR"?>
<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
    <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
            <property name="JDBC.Driver"
                value="com.mysql.jdbc.Driver" />
            <property name="JDBC.ConnectionURL"
                value="jdbc:mysql://localhost:3306/ibatis_start" />
            <property name="JDBC.Username"
                value="ibatisuser" />
            <property name="JDBC.Password"
                value="ibatisuser" />
        </dataSource>
    </transactionManager>

    <sqlMap resource="User.xml" />
</sqlMapConfig>
```

iBatis Quick Start

□ SqlMap 파일 작성(User.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMap
  PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
  "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="User">
  <typeAlias alias="User" type="kr.nexttree.ibatisstart.domain.User"/>

  <resultMap id="userMap" class="User">
    <result property="userId" column="userId"/>
    <result property="password" column="password"/>
    <result property="name" column="name"/>
    <result property="email" column="email"/>
    <result property="userType" column="userType"/>
  </resultMap>

  <select id="selectAllUsers" resultMap="userMap">
    select
      user.USER_ID as userId,
      user.PASSWORD as password,
      user.NAME as name,
      user.EMAIL as email,
      user.TYPE as userType
    from USER_TB user
    order by user.USER_ID ASC
  </select>
</sqlMap>
```

iBatis Quick Start

□ iBatis Data Access Object 작성

- UserIBatisDao.java

```
package kr.nexttree.ibatisstart.da.ibatis;

import java.io.IOException;
import java.io.Reader;
import java.sql.SQLException;
import java.util.List;

import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;

import kr.nexttree.ibatisstart.domain.User;

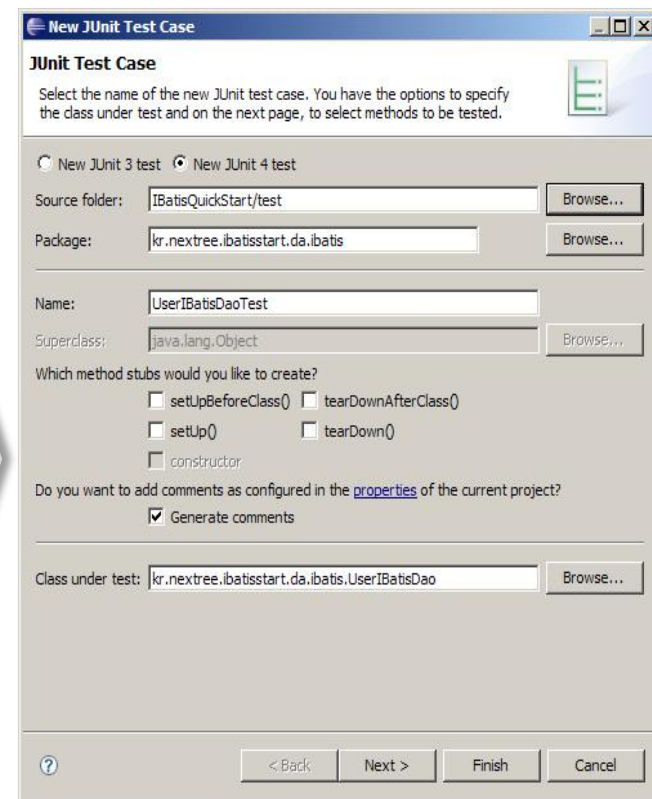
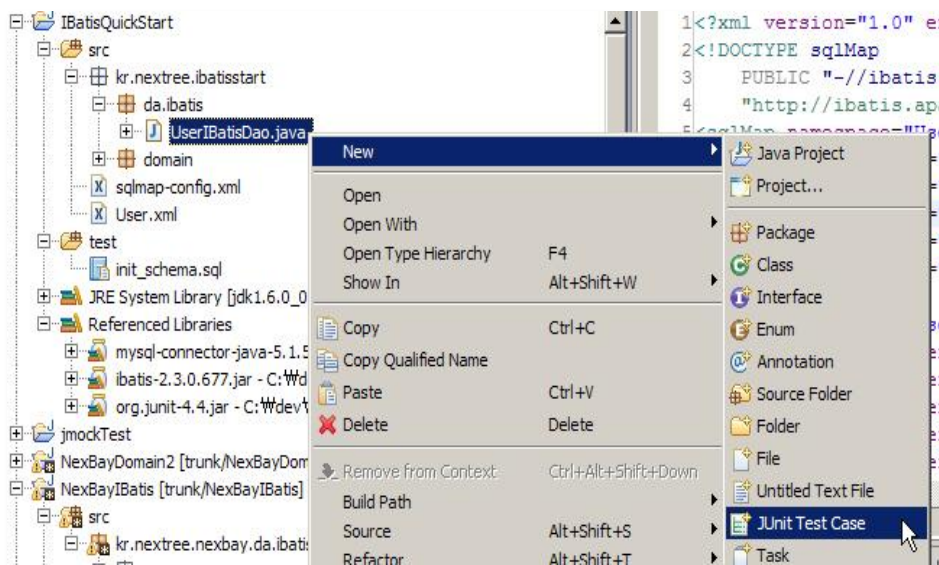
public class UserIBatisDao {
    private static final String resource = "sqlmap-config.xml";
    private SqlMapClient client;

    public UserIBatisDao() {
        try {
            Reader reader = Resources.getResourceAsReader(resource);
            this.client = SqlMapClientBuilder.buildSqlMapClient(reader);
        } catch (IOException e) {
            throw new RuntimeException("SqlMapClient 생성중 오류발생", e);
        }
    }

    @SuppressWarnings("unchecked")
    public List<User> findAllUsers() {
        try {
            List<User> users = client.queryForList("selectAllUsers", null);
            return users;
        } catch (SQLException e) {
            throw new RuntimeException("사용자 조회중 오류 발생", e);
        }
    }
}
```

□ Test Case 작성

- UserIBatisDao.java 선택 상태에서 오른쪽 마우스 버튼
- New > JUnit Test Case
- Source folder를 test 폴더로 변경한다.
- Finish 선택



□ Test Case 작성(UserIBatisDaoTest.java)

- 사용자 목록 조회 테스트임

```
package kr.nexttree.ibatisstart.da.ibatis;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

import java.util.List;

import kr.nexttree.ibatisstart.domain.User;

import org.junit.Test;

public class UserIBatisDaoTest {
    @Test
    public void testFindAllUsers() {
        UserIBatisDao dao = new UserIBatisDao();
        List<User> users = dao.findAllUsers();
        assertNotNull(users);
        assertEquals(2, users.size());
        assertEquals("syhan", users.get(0).getUserId());
        assertEquals("tsong", users.get(1).getUserId());
    }
}
```

조회된 사용자 목록은 2
건

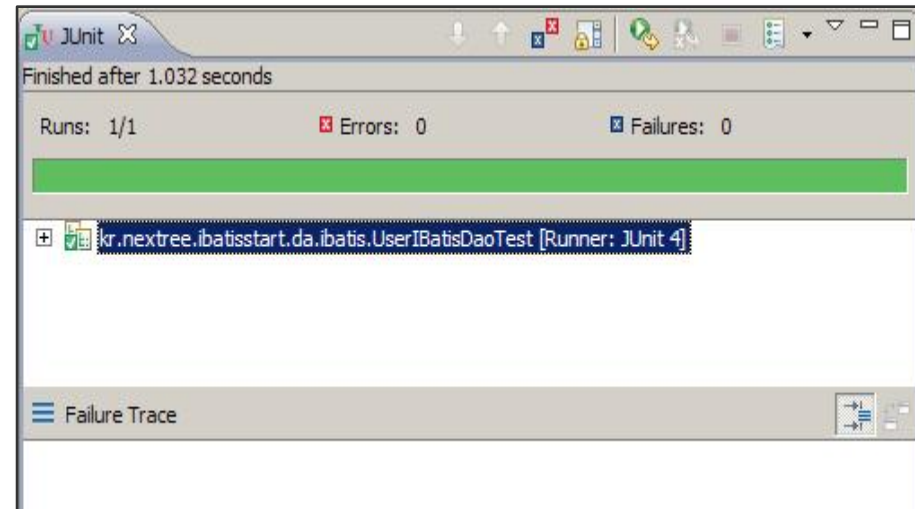
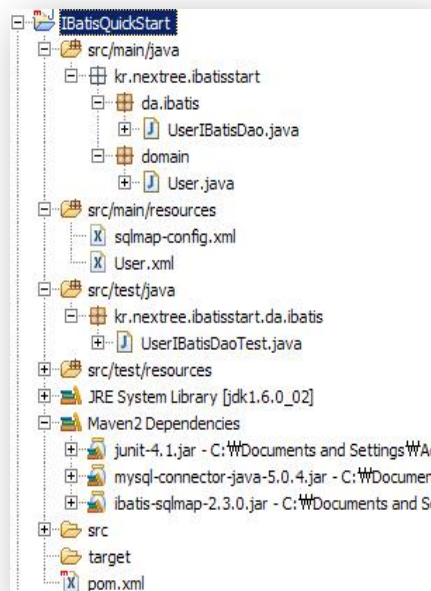
첫번째 사용자 ID는
'syhan'

두번째 사용자 ID는
'tsong'

□ TestCase 실행

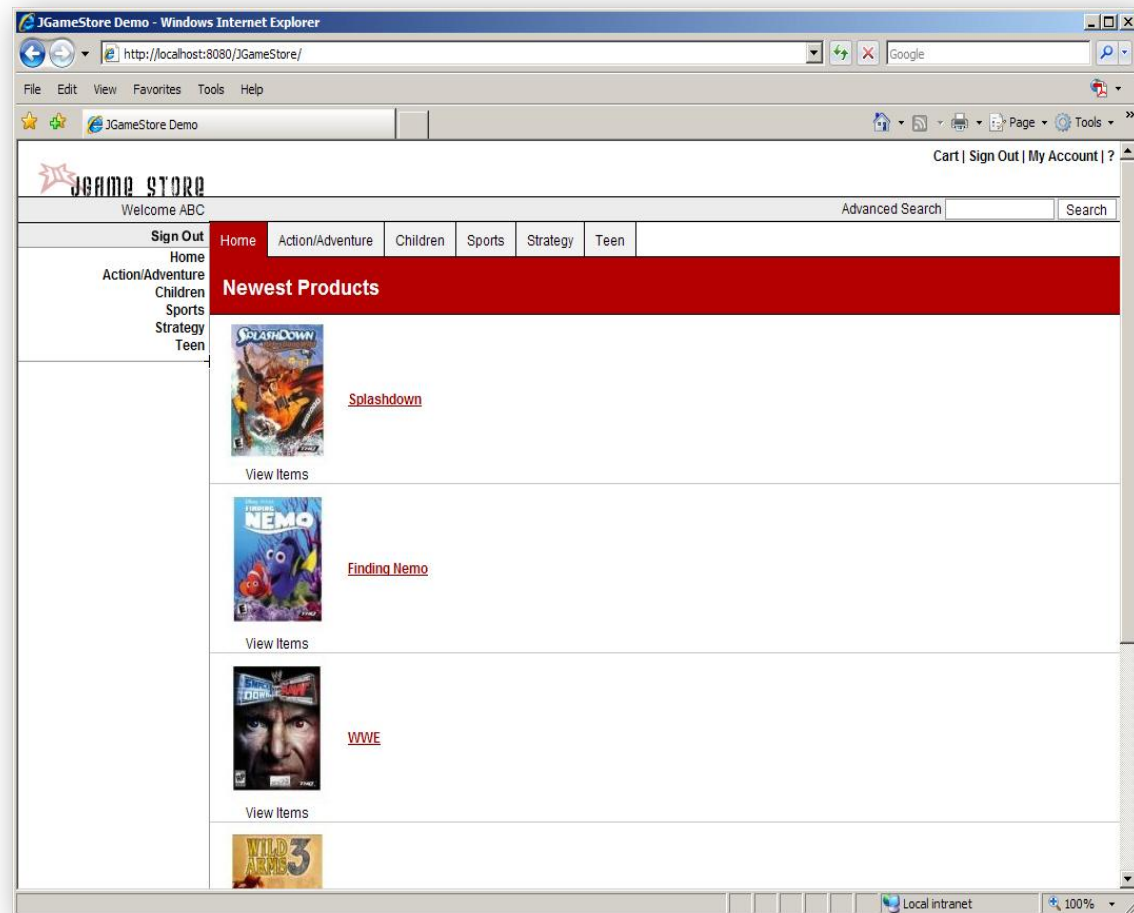
- UserIBatisDaoTest.java 선택후 오른쪽 마우스 버튼
- Run As > JUnit Test
- 지금까지의 설정이 잘 되었으면 테스트 성공함.

완성된 프로젝트 전체 구조



❑ Manning에서 제공하는 예제 프로젝트

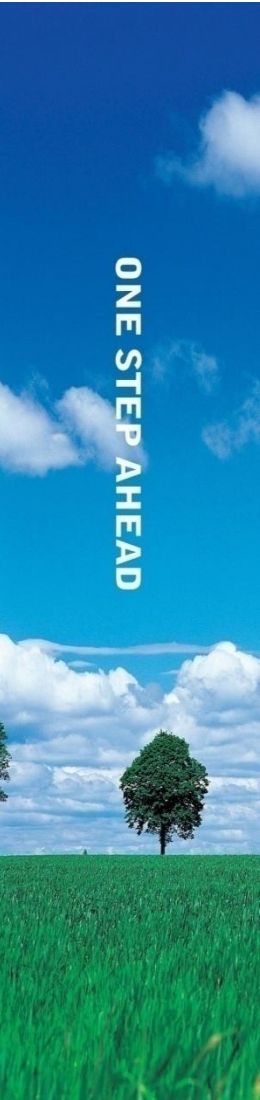
- <http://www.manning.com/begin/>



5. Junit 테스트 심화 II

- 데이터베이스 테스트 전략
- **DBUnit**을 이용한 단위 테스트
- 지속적인 통합(**Continuous Integration**)

ONE STEP AHEAD



데이터베이스 테스트 전략

- ❑ 데이터베이스 테스트가 중요한 이유
- ❑ 단위 테스트 전략
- ❑ 비즈니스 로직 단위 테스트
- ❑ 데이터베이스 연결 코드 단위 테스트
- ❑ 데이터베이스 통합 단위 테스트

데이터베이스 테스트가 중요한 이유

- 비즈니스 애플리케이션에서 데이터베이스 사용은 필수
- 하지만 대부분의 경우 데이터베이스 테스트에 어려움을 겪음



dependency는 모든 범위의 소프트웨어 개발에서 가장 중요한 문제이다.
프로그램에서 중복을 없애는 것은 dependency를 없애는 것이다

□ 비즈니스 로직 단위 테스트

- 비즈니스 로직 테스트에 초점을 맞춤
- 데이터베이스 접근 코드로부터 독립적으로 테스트가 가능해야 함
- 데이터베이스 접근 코드를 모의 객체로 대체해서 테스트 가능

□ 데이터베이스 접근 코드 단위 테스트

- 데이터베이스 접근을 위해 **Persistence API**를 사용하는 코드에 대한 테스트
- **Persistence API**를 제대로 사용하고 있는지에 초점을 맞추는 테스트
- 실제 데이터베이스에 연결하지 않고 모의 객체로 테스트 가능

□ 데이터베이스 통합 단위 테스트

- 데이터베이스 제공하는 기능을 체크
 - 데이터베이스 연결, 쿼리 실행, 스토어드 프로시저, 트리거, 제약조건...
- 컨테이너 테스트 or 경량프레임워크사용 or DB Unit

□ 목표

- 데이터베이스 접근 코드를 포함하지 않고 비즈니스 로직을 테스트 함
- 데이터베이스와 분리하여 테스트 함으로써 문제에 집중할 수 있음
- 데이터베이스 접근 레이어와 비즈니스 접근 레이어를 분리하는 것이 필수

```
public interface DataAccessManager  
{  
    Collection execute(String sql) throws Exception;  
}
```

```
private DataAccessManager dataManager;  
  
public void init() throws ServletException {  
    super.init();  
  
    try {  
        this.dataManager = new JdbcDataAccessManager();  
    }  
    catch (NamingException e) {  
        throw new ServletException(e);  
    }  
}  
  
public Collection executeCommand(String command)  
    throws Exception {  
    return this.dataManager.execute(command);  
}
```


□ 모의 객체를 이용한 비즈니스 로직 단위 테스트

- 데이터베이스 코드에 대한 인터페이스 구현(확장)을 통한 모의객체 생성

```
private DataAccessManager dataManager;

public void setDataAccessManager(DataAccessManager manager) {
    this.dataManager = manager;
}

public void init() throws ServletException {
    super.init();
    try {
        setDataAccessManager(new JdbcDataAccessManager());
    }
    catch (NamingException e) {
        throw new ServletException(e);
    }
}

public Collection executeCommand(String command) throws Exception {
    return this.dataManager.execute(command);
}
```

□ 모의 객체를 이용한 비즈니스 로직 단위 테스트

```
public class TestAdminServletDynaMock extends TestCase
{
    public void testSomething() throws Exception {
        Mock mockManager = new Mock(DataAccessManager.class);
        DataAccessManager manager =(DataAccessManager) mockManager.proxy();
        mockManager.expectAndReturn("execute", C.ANY_ARGS, new ArrayList());

        AdminServlet servlet = new AdminServlet();
        servlet.setDataAccessManager(manager);

        Servlet.executeCommand("SELECT * FROM ....");
    }
}
```

데이터베이스 접근 코드 단위 테스트

□ 데이터베이스 접근 코드 테스트

- 비니지스 로직을 데이터베이스 접근 코드로부터 분리하여 테스트 작성과 실행이 간결해짐
- 하지만 실제 데이터베이스 접근 코드에 대한 테스트도 필수적임
- 모의 객체를 통한 접근 코드 테스트 작성을 통해 테스트 할 수 있음
- JDBC API는 모의 객체 사용이 쉽도록 잘 설계되어 있음

□ 접근코드 예제 - **JDBCDataAccessManager**

```
public class JdbcDataAccessManager implements DataAccessManager{
    private DataSource dataSource;
    public JdbcDataAccessManager() throws NamingException {
        this.dataSource = getDataSource();
    }
    protected DataSource getDataSource() throws NamingException {
        InitialContext context = new InitialContext();
        DataSource dataSource = (DataSource) context.lookup("java:/DefaultDS");
        return dataSource;
    }
    protected Connection getConnection() throws SQLException {
        return this.dataSource.getConnection();
    }
    public Collection execute(String sql) throws Exception {
        Connection connection = getConnection();

        ResultSet resultSet = connection.createStatement().executeQuery(sql);
        RowSetDynaClass rsdc = new RowSetDynaClass(resultSet);
        resultSet.close();
        connection.close();
        return rsdc.getRows();
    }
}
```

데이터베이스 접근 코드 단위 테스트

□ 테스트를 위한 **JDBCDataAccessManager** 확장

```
public class TestableJdbcDataAccessManager extends JdbcDataAccessManager
{
    private Connection connection;
    public TestableJdbcDataAccessManager() throws NamingException {
        super();
    }
    public void setConnection(Connection connection) {
        this.connection = connection;
    }
    protected Connection getConnection() throws SQLException {
        return this.connection;
    }
    protected DataSource getDataSource() throws NamingException {
        return null;
    }
}
```

데이터베이스 접근 코드 단위 테스트

□ JDBCDataAccessManager 테스트 작성

```
public class TestJdbcDataAccessManagerMO1 extends TestCase {  
    private MockStatement statement;  
    private MockConnection2 connection;  
    private MockSingleRowResultSet resultSet;  
    private MockResultSetMetaData resultSetMetaData;  
    private TestableJdbcDataAccessManager manager;  
  
    protected void setUp() throws Exception {  
        resultSetMetaData = new MockResultSetMetaData();  
        resultSet = new MockSingleRowResultSet();  
        resultSet.setupMetaData(resultSetMetaData);  
  
        statement = new MockStatement();  
        connection = new MockConnection2();  
        connection.setupStatement(statement);  
        manager = new TestableJdbcDataAccessManager();  
        manager.setConnection(connection);  
    }  
    [...]
```

□ JDBCDataAccessManager 테스트 작성

//앞 페이지에서 계속...

```
public void testExecuteOk() throws Exception {
    String sql = "SELECT * FROM CUSTOMER";
    statement.addExpectedExecuteQuery(sql, resultSet);

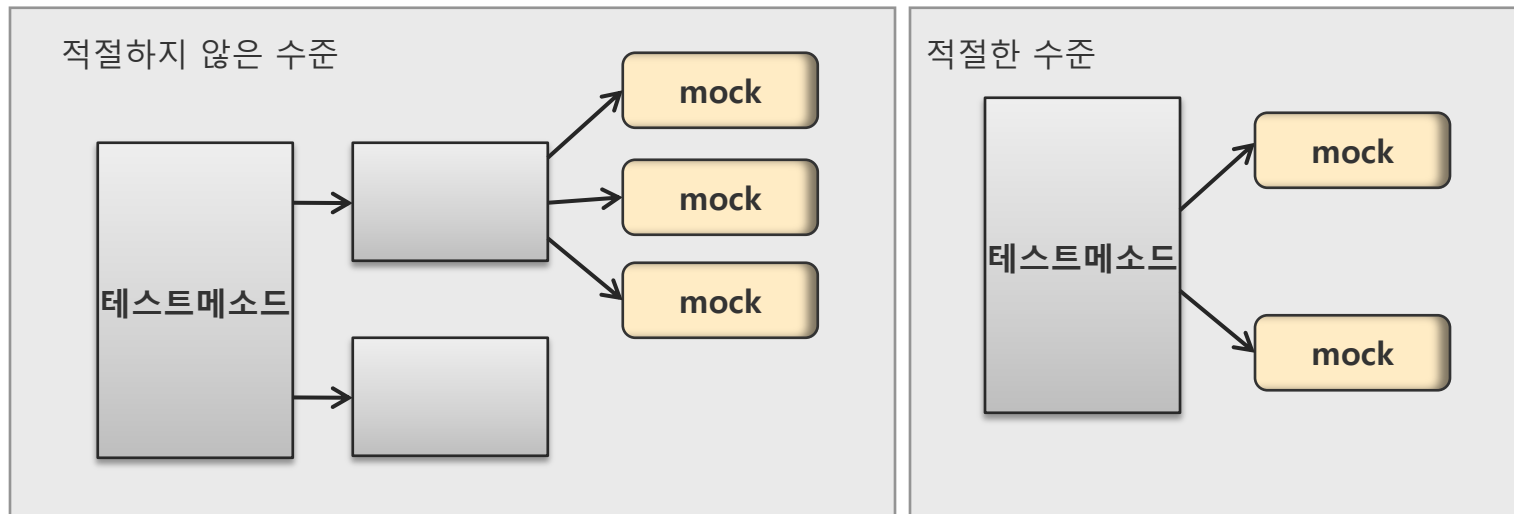
    String[] columnsLowercase = new String[] {"firstname", "lastname"};
    String[] columnsUppercase = new String[] {"FIRSTNAME", "LASTNAME"};
    String[] columnClassNames = new String[] {String.class.getName(), String.class.getName()};
    resultSetMetaData.setupAddColumnNames(columnsUppercase);
    resultSetMetaData.setupAddColumnClassNames(columnClassNames);
    resultSetMetaData.setupGetColumnCount(2);

    resultSet.addExpectedNamedValues(columnsLowercase, new Object[] {"John", "Doe"});
    Collection result = manager.execute(sql);
    Iterator beans = result.iterator();
    assertTrue(beans.hasNext());
    DynaBean bean1 = (DynaBean) beans.next();
    assertEquals("John", bean1.get("firstname"));
    assertEquals("Doe", bean1.get("lastname"));
    assertTrue(!beans.hasNext());
}
}
```

- <http://mockobjects.com> 에서 제공하는 mockobj.jar
- JDBC API에 대한 Mock Object 제공

□ 모의 객체 사용시 주의 사항

- Mocking 해야 하는 객체가 많아질 수록 더 많은 **setup**이 필요
- Setup 코드가 많을 수록 작성이 어렵고, 유지보수 비용이 커짐
- 테스트하려는 코드보다 모의 객체 설정이 더 어렵다면 **refactoring**이 필요
- 왜 **setup** 코드가 많아지나?
 - 테스트하려는 메서드가 너무 많을 경우
 - 환경을 설정하기가 어려운 경우
 - 적절하지 않은 수준에서 모의 객체를 만들 경우



데이터베이스 접근 코드 단위 테스트

□ 실패 상황 테스트

//앞 소스에서 계속...

```
public void testExecuteCloseConnectionOnException() throws Exception {
```

```
    String sql = "SELECT * FROM CUSTOMER";
```

```
    statement.setupThrowExceptionOnExecute(new SQLException("sql error"));
```

```
    connection.setExpectedCloseCalls(1);
```

```
    Try {
```

```
        manager.execute(sql);
```

```
        fail("Should have thrown a SQLException");
```

```
    }
```

```
    catch (SQLException expected) {
```

```
        assertEquals("sql error", expected.getMessage());
```

```
    }
```

```
}
```

```
protected void tearDown() {
```

```
    connection.verify();
```

```
    statement.verify();
```

```
    resultSet.verify();
```

```
}
```

```
}
```

데이터베이스 접근 코드 단위 테스트

□ 리팩토링(Refactoring)

```
public class JdbcDataManager implements DataManager{

    [...]

    public Collection execute(String sql) throws Exception {
        Try {
            Connection connection = getConnection();
            ResultSet resultSet = connection.createStatement().executeQuery(sql);
            RowSetDynaClass rsdc = new RowSetDynaClass(resultSet);
            result = rsdc.getRows();
        }
        Finally {
            if(resultSet != null) resultSet.close();
            if(connection != null) connection.close();
        }
        return result;
    }
}
```

데이터베이스 통합 단위 테스트

□ 데이터베이스 통합 단위 테스트

- 실제 데이터베이스에 연결하여 단위 테스트를 진행하는 과정

□ 테스트 목표

- 데이터베이스 연결이 적절한가?
- 데이터베이스에 위치하는 비즈니스 로직(Stored Procedure)의 동작유무
- 트리거가 적절히 동작하는가?
- 제약조건이 의도된 데로 동작하는가?
- 참조 무결성이 적절히 설정되었는가?
- 자바 객체와 데이터베이스간의 데이터 변환이 적절한가?

데이터베이스 통합 단위 테스트

□ 정밀한 테스트 전략 수립 필요

- 실제 데이터베이스를 어떤 식으로 접근할 것인가에 대한 전략 필요
 - 인-컨테이너, 아웃-컨테이너 테스트 전략
 - 인-컨테이너 통합 단위 테스트
 - Cactus
 - 아웃-컨테이너 통합 단위 테스트
 - Hibernate, iBatis
- 테스트 데이터 생성/관리 전략 필요
 - 메모리 데이터베이스 **vs** 실제 데이터베이스 사용
 - 개인 데이터베이스 생성 **vs** 공용 테스트용 데이터베이스 생성
 - 데이터 베이스에 테스트 데이터 생성 **vs** 테스트 마다 데이터 생성

DBUnit을 이용한 단위 테스트

- ❑ **DBUnit** 개요
- ❑ **Dataset** 생성
- ❑ **DBTestCase** 클래스 확장하여 테스트 만들기
- ❑ **TestCase** 클래스 확장하여 테스트케이스 만들기

□ DBUnit 소개

- 데이터베이스에 대한 **Testing** 프로세스를 훨씬 단순하게 만드는 프레임워크
- 데이터베이스에서 데이터를 선택, 업데이트, 삽입, 제거하는데 사용될 수 있는 테스트 데이터를 정의하는 표준 **XML** 포맷을 제공
- 데이터베이스를 대체하는 것이 아닌 테스트 데이터를 핸들링 하는 보다 효율적인 메커니즘을 제공
- 공식 사이트 : <http://www.dbunit.org/>
 - dbunit.jar를 다운받아 classpath 내에 위치하도록 설치

□ 단위 테스트를 돕는 DBUnit

- 단위 테스트를 수행에서 데이터베이스를 사용할 경우 로컬 데이터베이스를 이용할 경우에 **DBUnit**을 이용하여 쉽게 제어 가능
- 개발 환경에서 테스트용으로 사용하는 는 독립된 서버로 동작

메이븐2에서 DBUnit 설치

□ DBUnit 설치

- POM.xml 파일에 dbunit dependency 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>

...
<version>1.0</version>
<build>
...
</build>
<dependencies>
...
  <dependency>
    <groupId>dbunit</groupId>
    <artifactId>dbunit</artifactId>
    <version>2.2</version>
  </dependency>
</dependencies>
</project>
```

DBUnit을 이용한 테스트 전략

□ DBUnit을 이용한 테스트 전략

- 개발자가 고유의 데이터베이스 인스턴스를 사용할 수 있도록 해라
 - 데이터가 꼬여서 테스트가 실패하는 상황을 방지할 수 있음
 - 데이터베이스 데이터 클린업을 간소화 할 수 있음
 - 초기상태로 되돌아갈 필요가 없음
- 여러 개의 작은 **dataset**을 사용하라
 - 대부분의 테스트케이스의 데이터베이스의 일부분만을 바라 봄
 - 테스트마다 전체 데이터를 생성하지 말고 필요한 데이터만을 생성
 - 데이터베이스 데이터를 초기화하는 오버헤드를 피할 수 있음
 - 필요한 **Dataset**만을 관리함으로써 을 팀 개발을 용이하게 함
 - 통합테스트 과정에서는 **CompositeDataSet**을 사용하여 데이터를 통합할 수 있음
- 같은 데이터를 공유하는 테스트케이스는 한번만 초기화 해라
- 연결관리 전략을 사용하라
 - Remote Client, In-Container 테스트에 따라 다른 연결 전략을 사용
 - Remote Client 테스트의 경우 연결이 한번만 이루어지도록 함
 - In-Container 테스트의 경우 **Datasource**를 사용해서 연결을 관리하도록 함

□ DBUnit의 구성

- DataSet
 - 데이터베이스에 입력할 데이터를 다루는 양식 제공
- DatabaseOperation
 - 데이터베이스에 데이터를 입력/삭제/수정하기 위한 오퍼레이션 제공
- Verify
 - Dataset간의 데이터를 비교하기 위한 도구 제공
 - assertEquals(ITable expected, ITable actual)
 - assertEquals(IDataSet expected, IDataSet actual)

□ DataSet

- DBUnit을 통해 데이터베이스에 입력할 데이터를 다루는 객체
- 다양한 DataSet을 통한 데이터를 조작 기능 제공
- 현재 총 10개의 데이터셋을 제공

□ 다양한 종류의 DataSet을 지원

구현	설명
FlatXMLDataSet	<p>Flat XML 데이터셋 문서를 읽고 씀. 각 XML 엘리먼트는 테이블 레코드에 매핑 각 XML 엘리먼트는 이름은 테이블 이름과 매핑</p> <pre> <!DOCTYPE dataset SYSTEM "my-dataset.dtd"> <dataset> <TEST_TABLE COL0="row 0 col 0" COL1="row 0 col 1" COL2="row 0 col 2"/> <TEST_TABLE COL1="row 1 col 1"/> <SECOND_TABLE COL0="row 0 col 0" COL1="row 0 col 1" /> <EMPTY_TABLE/> </dataset> </pre>

□ 다양한 종류의 DataSet을 지원

구현	설명
FlatXMLDataSet	Flat XML 데이터셋 문서를 읽고 씀. 각 XML 엘리먼트는 테이블 레코드에 매핑 각 XML 엘리먼트는 이름은 테이블 이름과 매핑

```
<!DOCTYPE dataset SYSTEM "dataset.dtd">
<dataset>
  <table name="TEST_TABLE">
    <column>COL0</column><column>COL1</column><column>COL2</column>
    <row><value>row 0 col 0</value><value>row 0 col 1</value><value>row 0 col 2</value></row>
    <row><null/><value>row 1 col 1</value><null/></row>
  </table>
  <table name="SECOND_TABLE">
    <column>COLUMN0</column><column>COLUMN1</column>
    <row><value>row 0 col 0</value><value>row 0 col 1</value></row>
  </table>
  <table name='EMPTY_TABLE'>
    <column>COLUMN0</column><column>COLUMN1</column>
  </table>
</dataset>
```

□ 다양한 종류의 DataSet을 지원

구현	설명
StreamingDataSet	<p>Produce 클래스를 입력으로 받아 이를 처리함. 단방향 데이터 접근을 제공하며, FlatXmlProducer, XmlProvider 클래스와 함께 사용될 수 있음</p> <pre> IDataSetProducer producer = new FlatXmlProducer(new InputSource("dataset.xml")); IDataSet dataSet = new StreamingDataSet(producer); </pre>
DatabaseDataSet	<p>데이터셋으로 데이터베이스에 접근할 수 있도록 해주는 어댑터. 직접 인스턴스화되지 않고 팩토리 메소드를 통해서 생성한다.</p> <pre> IDatabaseConnection.createDataSet() </pre>
QueryDataSet	<p>데이터베이스 쿼리 결과를 데이터셋으로 보관</p> <pre> QueryDataSet dataSet = new QueryDataSet(connection); dataSet.addTable("FOO", "SELECT * FROM TABLE WHERE COL='VALUE'"); dataSet.addTable("BAR"); </pre>

□ 다양한 종류의 DataSet을 지원

구현	설명
DefaultDataSet	데이터셋을 프로그램으로 생성할 수 있음
CompositeDataSet	여러 개의 데이터셋을 조합하여 하나의 데이터셋으로 사용 가능
FilteredDataSet	데이터셋으로부터 필터 조건에 따라 일부 테이블을 포함/배제 시킬 수 있음
XslDataset	MS 엑셀 데이터셋 문서를 처리함 각 시트는 테이블을 표현, 시트의 첫 행은 컬럼 이름으로 나머지 행은 데이터로 처리됨

□ 다양한 종류의 DataSet을 지원

구현	설명
ReplacementDataSet	<p>데이터셋 내의 특정 문자열을 판독하여 실행시간에 실제 값을 대체함</p> <pre><?xml version="1.0"?> <dataset> <TEST_TABLE COL0="row 0 col 0" COL1="[null]"/> <TEST_TABLE COL1="row 1 col 0" COL2="row 1 col 1"/> </dataset></pre> <pre>ReplacementDataSet dataSet = new ReplacementDataSet(new FlatXmlDataSet(...)); dataSet.addReplacementObject("[NULL]", null);</pre>

□ 기존 데이터베이스에서 **DTD** 추출하기

```
public class DatabaseExportSample {  
  
    public static void main(String[] args) throws Exception {  
  
        //iBatis 컨넥션 가져오기  
        //Connection jdbcConnection = sqlMapClient.getDataSource().getConnection();  
  
        IDatabaseConnection connection = new DatabaseConnection(jdbcConnection);  
  
        // write DTD file  
        FlatDtdDataSet.write(connection.createDataSet(), new FileOutputStream("test.dtd"));  
    }  
}
```


□ 기존 데이터베이스에서 **flat XML dataset** 추출하기

```
public class DatabaseExportSample {
    public static void main(String[] args) throws Exception {

        //iBatis 컨넥션 가져오기
        //Connection jdbcConnection = sqlMapClient.getDataSource().getConnection();

        IDatabaseConnection connection = new DatabaseConnection(jdbcConnection);

        QueryDataSet partialDataSet = new QueryDataSet(connection);
        partialDataSet.addTable("FOO", "SELECT * FROM TABLE WHERE COL='VALUE'");
        partialDataSet.addTable("BAR");
        FlatXmlDataSet.write(partialDataSet, new FileOutputStream("partial.xml"));

        IDataset fullDataSet = connection.createDataSet();
        FlatXmlDataSet.write(fullDataSet, new FileOutputStream("full.xml"));

        String[] depTableNames = TablesDependencyHelper.getAllDependentTables( connection, "X" );
        IDataset depDataSet = connection.createDataSet( depTableNames );
        FlatXmlDataSet.write(depDataSet, new FileOutputStream("dependents.xml"));

    }
}
```

DBTestCase 클래스 확장하여 테스트 만들기

□ DBTestCase에서 확장 가능한 클래스들

구현	설명
JdbcBasedDBTestCase	연결 생성을 위해 DriverManager사용
DataSourceBasedDBTestCase	Javax.sql.DataSource를 사용하여 연결 설정
JndiBasedDBTestCase	JNDI에 설정된 Javax.sql.DataSource를 사용하여 연결 설정

□ 상세 설정은 **unitils.properties**를 통해 설정

```

database.dialect=mysql
database.driverClassName=com.mysql.jdbc.Driver
database.url=jdbc:mysql://localhost/spring
database.userName=javajigi
database.password=password
database.schemaName=spring
    
```

DBTestCase 클래스 확장하여 테스트 만들기

□ DBTestCase 클래스 확장하여 만든 테스트 케이스

```
public class RateServiceDBUnitTest extends DBTestCase {

    public RateServiceDBUnitDBTestCaseTest(String name) {
        super(name);

        System.setProperty( PropertiesBasedJdbcDatabaseTester.DBUNIT_DRIVER_CLASS,
                               "oracle.jdbc.OracleDriver" );
        System.setProperty( PropertiesBasedJdbcDatabaseTester.DBUNIT_CONNECTION_URL,
                               "jdbc:oracle:thin:@192.168.XXX.XXX:1521:XE" );
        System.setProperty( PropertiesBasedJdbcDatabaseTester.DBUNIT_USERNAME, "nec_XXX" );
        System.setProperty( PropertiesBasedJdbcDatabaseTester.DBUNIT_PASSWORD, "XXX" );
    }
    protected IDataset getDataSet() throws Exception {
        return new FlatXmlDataSet(getClass().getClassLoader().getResourceAsStream("tb_rate.xml"));
    }
    @Before
    protected void setUp() throws Exception {
        IDatabaseConnection connection = getConnection();
        try { DatabaseOperation.CLEAN_INSERT.execute(connection, getDataSet()); }
        finally { connection.close(); }
    }
    ...
}
```

DBTestCase 클래스 확장하여 테스트 만들기

□ DBTestCase 클래스 확장하여 만든 테스트 케이스

```
public class RateServiceDBUnitTest extends DBTestCase {  
    ...  
    @After  
    protected void tearDown() throws Exception {  
        IDatabaseConnection connection = getConnection();  
  
        try {  
            DatabaseOperation.DELETE_ALL.execute(connection, getDataSet());  
        } finally {  
            connection.close();  
        }  
    }  
  
    @Test  
    public void testListAllDividendRates() {  
        List result1 = this.rateService.listAllDividendRates();  
        assertEquals(2, result1.size());  
    }  
}
```

IDataBaseTester 인터페이스 사용

□ IDatabaseTester 인터페이스 사용

- DB 컨넥션을 쉽게 처리할 수 있는 기능을 제공
- 2.2버전 이후 추가
- DBTestCase도 내부적으로 사용함

구현	설명
JdbcDatabaseTester	단순 JDBC 연결용 연결 생성을 위해 DriverManager사용
PropertiesBasedJdbcDatabaseTester	단순 JDBC 연결용 연결 생성을 위해 DriverManager사용 세부 설정은 시스템 프로퍼티로부터 읽음
DataSourceDatabaseTester	Javax.sql.DataSource를 사용하여 연결 설정
JndiDatabaseTester	JNDI에 설정된 Javax.sql.DataSource를 사용하여 연결 설정

TestCase 클래스 확장하여 테스트케이스 만들기

□ TestCase를 확장하여 만든 dbunit 테스트케이스

```
public class RateServiceDBUnitTest extends TestCase {
    private DataSource dataSource;
    private IDatabaseTester databaseTester;

    protected IDataset getDataSet() throws Exception {
        return new FlatXmlDataSet(getClass().getClassLoader().getResourceAsStream("tb_rate.xml"));
    }

    @Before
    protected void setUp() throws Exception {
        databaseTester = new JdbcDatabaseTester("oracle.jdbc.OracleDriver",
                                                "jdbc:oracle:thin:@192.168.XXX.XXX:1521:XE",
                                                "nec_XXX", "XXX");

        databaseTester.setDataSet( getDataSet() );
        databaseTester.setSetUpOperation(DatabaseOperation.CLEAN_INSERT);
        databaseTester.onSetup();
    }
    ...
}
```

TestCase 클래스 확장하여 테스트케이스 만들기

□ TestCase를 확장하여 만든 dbunit 테스트케이스

```
public class RateServiceDBUnitTest extends TestCase {  
    ...  
    @After  
    protected void tearDown() throws Exception {  
        databaseTester.setTearDownOperation(DatabaseOperation.DELETE_ALL);  
        databaseTester.onTearDown();  
    }  
  
    @Test  
    public void testListAllDividendRates() {  
        List result1 = this.rateService.listAllDividendRates();  
        assertEquals(2, result1.size());  
    }  
}
```

Spring + dbunit 테스트케이스 작성

□ Spring의 DI(Dependency Injection)을 이용

```
public class RateServiceDBUnitTest extends AbstractDependencyInjectionSpringContextTests {
    private DataSource dataSource;

    //di로 dependency 주입
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    protected IDataset getDataSet() throws Exception {
        return new FlatXmlDataSet(getClass().getClassLoader().getResourceAsStream("tb_rate.xml"));
    }

    protected void onSetUp() throws Exception {
        super.onSetUp();
        Connection jdbcConnection = dataSource.getConnection();
        IDatabaseConnection connection = new DatabaseConnection(jdbcConnection);
        try {
            DatabaseOperation.CLEAN_INSERT.execute(connection, getDataSet());
        } finally {
            connection.close();
        }
    }
}
```


Spring + dbunit 테스트케이스 작성

□ Spring의 DI(Dependency Injection)을 이용

```
public class RateServiceDBUnitTest extends AbstractDependencyInjectionSpringContextTests {
    ...

    protected void onTearDown() throws Exception {
        super.onTearDown();
        Connection jdbcConnection = dataSource.getConnection();
        IDatabaseConnection connection = new DatabaseConnection(jdbcConnection);

        try {
            DatabaseOperation.DELETE_ALL.execute(connection, getDataSet());
        } finally {
            connection.close();
        }
    }

    @Test
    public void testListAllDividendRates() {
        List result1 = this.rateService.listAllDividendRates();
        assertEquals(2, result1.size());
    }
}
```

Dbunit을 이용한 데이터 검증

□ 테스트 데이터와 실제 데이터의 비교

```
public class RateServiceDBUnitTest extends AbstractDependencyInjectionSpringContextTests {  
    ...  
  
    @Test  
    public void testListAllDividendRates() {  
        ...  
  
        IDataset databaseDataSet = getConnection().createDataSet();  
        ITable actualTable = databaseDataSet.getTable("TB_RATE");  
  
        IDataset expectedDataSet = new FlatXmlDataSet(new File("expectedDataSet.xml"));  
        ITable expectedTable = expectedDataSet.getTable("TB_RATE");  
  
        Assertion.assertEquals(expectedTable, actualTable);  
    }  
}
```

```

명령 프롬프트

[INFO] [compiler:compile]
[INFO] Compiling 148 source files to D:\work\WNECWNECMAA\target\classes
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Compiling 34 source files to D:\work\WNECWNECMAA\target\test-classes
[INFO] [surefire:test]
[INFO] Surefire report directory: D:\work\WNECWNECMAA\target\surefire-reports

-----
T E S T S
-----
Running kr.go.nec.mutual.service.RateServiceDBUnitJDBCTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1 sec
Running kr.go.nec.mutual.service.CodeServiceTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.016 sec
Running kr.go.nec.mutual.service.ZipCodeServiceTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.328 sec
Running kr.go.nec.mutual.domain.PayTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec
Running kr.go.nec.mutual.service.FinanceCalculatorServiceTest
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.031 sec
Running kr.go.nec.mutual.service.LoanOnceServiceTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.344 sec
Running kr.go.nec.mutual.service.RateServiceDBUnitTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.5 sec
Running kr.go.nec.mutual.service.calculator.roundsun.GoalSavingCalculatorTest
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathContext.DECIMAL64)) : 2239927.74135000007878988981246948242187500000000000000
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathContext.DECIMAL64)) : 2392402.74975000016856938600540161132812500000000000000
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathContext.DECIMAL64)) : 2636916.75
ERROR [main] <GoalSavingCalculatorTest.java:56> - 8146673
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathContext.DECIMAL64)) : 2273721.8057345259636640548706054687500000000000000
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathContext.DECIMAL64)) : 2428440.06094260929850861430168151855468750000000000000
ERROR [main] <GoalSavingCalculator.java:73> - interestAmount.multiply(new BigDecimal(1 - taxRate, MathContext.DECIMAL64)) : 2676536.4038515244610607624053955078125
ERROR [main] <GoalSavingCalculatorTest.java:74> - 2273722
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.015 sec
Running kr.go.nec.mutual.util.PageCriteriaTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

```

Continuous Integration

- 개요
- **Continuum** 설치
- **Continuum** 활용

□ 깨진 유리창의 이론 (Broken Window Theory)

- 사소한 곳에서 발생하며 예방이 쉽지 않다
- 문제가 확인되더라도 소홀하게 대응한다
- 문제가 커진 후 치료하려면 몇 배의 시간과 노력이 필요하다
- 투명테이프로 숨기려 해도 여전히 보인다
- 제대로 수리하면 큰 보상을 가져다 준다

Broken Window Theory

깨진 유리창 이론은 미국 범죄학에서 연구되어 정리된 법칙으로서, 도시의 슬럼화가 어떻게 발생하는가에 대한 원리를 말한다.

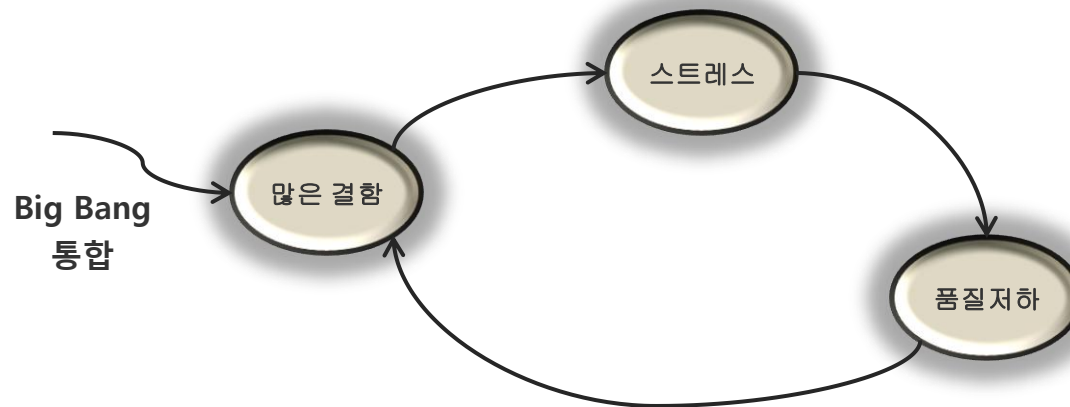
깨진 유리창을 방치해 두면, 그 지점을 중심으로 점점 슬럼화가 진행되기 시작한다는 이론이다.

참조: 위키백과



□ Big Bang 통합의 문제점

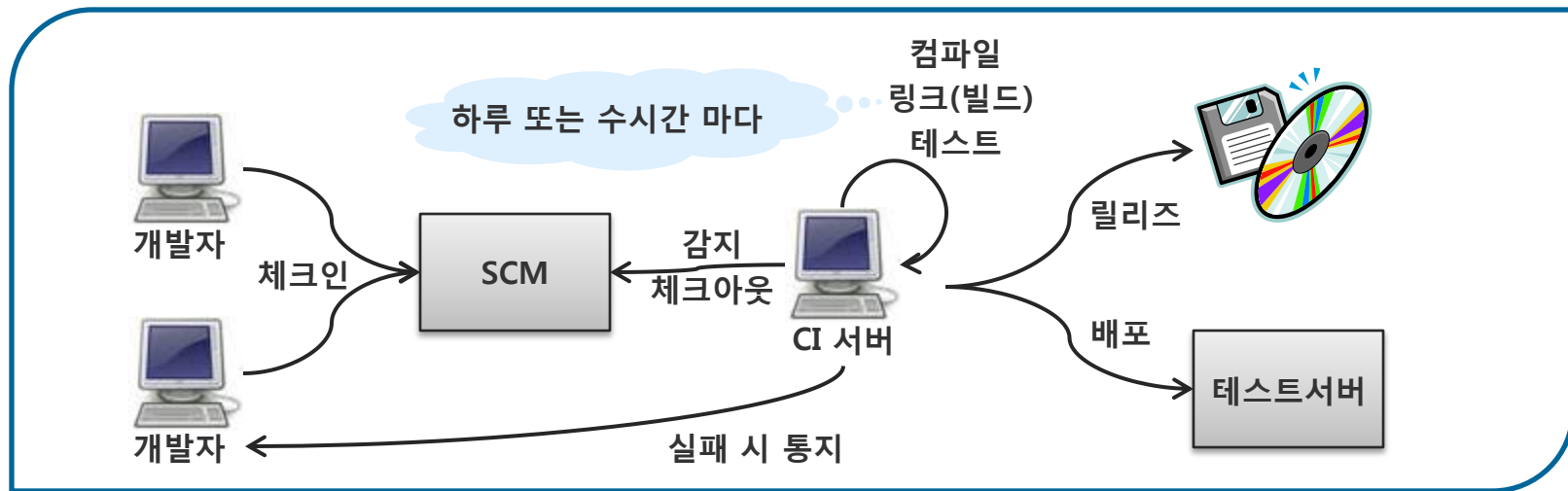
- 통합 전까지는 아무도 통합과 관련된 문제점을 모른다
- 팀 프로그램은 분할해서 정복할 수 있는 성질의 문제가 아니다
- 통합을 오래 미룰수록 비용이 더 들며 기간도 예측이 어렵다
- 오래 전에 개발했던 코드는 기억하기 힘들다
- 한 번에 쏟아지는 많은 결함은 개발자를 지치게 한다



**최대한 빨리 통합하고 지속적으로 통합하라
(eXtreme Programming)**

□ 지속적인 통합 서버

- 가능한 빠른 시기에 작은 크기의 코드라도 통합
- 변경한 것은 즉시 **SCM**에 체크인함 (단 단위 테스트 성공 후!)
- 통합서버는 변경을 인지하고 변경과 관련된 것들을 빌드
- 빌드 후 자동화된 테스트를 수행
- 테스트 성공 후 주기적으로 릴리즈
- 빌드, 테스트, 릴리즈 실패 시 즉각적으로 통지함



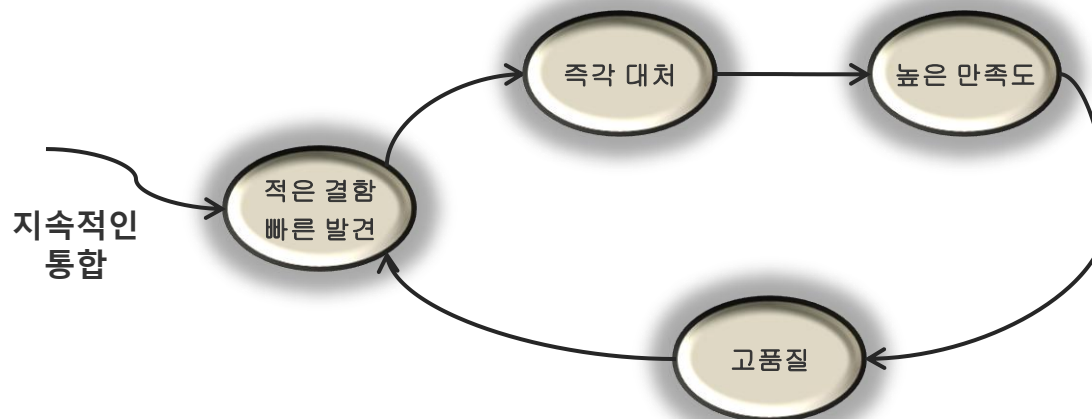
참조: <http://martinfowler.com/articles/continuousIntegration.html>

□ 지속적인 통합 서버

- 단위 소스 저장소 유지한다 (SCM: Subversion, cvs, ...)
- 빌드를 자동화한다
- **Self** 테스트가 가능한 테스트 프로그램을 작성한다 (xUnit 테스트 케이스)
- 매일 체크인 한다 (항상 변경사항이 적용되어 전달되어야 함)
- 통합 서버에는 항상 최신의 **Mainline**이 건강한 상태로 유지되어야 한다
- 빠른 빌드가 가능하도록 유지한다
- 가능한 운영환경과 동일한 환경에서 테스트 한다
- 누구라도 쉽게 최신의 실행파일을 얻을 수 있어야 한다
- 모든 사람이 무슨 일이 발생했는지 알 수 있어야 한다
- 배포(릴리즈)를 자동화한다

□ 지속적인 통합이 주는 장점

- 위험을 감소시킨다
- 통합이 언제 끝날 지 쉽게 예측할 수 있다
- 현재의 프로젝트 상황을 모두가 쉽게 알 수 있다
- 적은 결함으로 품질이 향상되고 개발자의 만족도가 높아진다
- 언제든지 제품을 릴리즈 할 수 있다



참조: <http://martinfowler.com/articles/continuousIntegration.html>

□ Continuous Integration 서버 선택 기준

- 기능
 - 버전관리 통합
 - 빌드 툴 통합
 - 피드백과 리포팅
 - 레이블링
 - 프로젝트 의존성
 - 확장성
- 신뢰성
 - 툴의 신뢰성과 성숙도, 인지도
- 내구성
 - 앞으로의 전망. 사용자 그룹
- 대상 환경
- 사용의 용이성
 - 주관적
 - 관리 UI 존재 여부 등

□ CI 서버 비교

	Continuum	CruiseControl	Luntbuild
기능	Ant, Maven1, Maven2, Shell 지원 XML-RPC와 SOAP을 사용한 원격 관리 Maven2 지원, 사용자 그룹 향후 추가 리포팅 및 피드백 메커니즘 도입 예정	버전 관리 통합, 확장성 풍부 JMX를 통한 원격 접근 RSS, X10, Jabber를 포함한 여러 피드백 메커니즘	프로젝트 의존성, 레이블링, 보안그룹 병렬개발
신뢰성	2005년 릴리즈 Apache 프로젝트	2001년 출시 대부분의 개발 단계에서 사용됨	2004년 출시 확장성 있는 사용자 지원 옵션
내구성	Apache Maven을 통한 사용자 커뮤니티 출시된 지 얼마 되지 않음	활동적인 사용자 커뮤니티	CruiseControl 만큼 사용자 커뮤니티가 활성화되어 않음
대상 머신	Linux, Mac OS X, Solaris, Win32	Windows와 Unix JVM을 실행하는 플랫폼	JVM과 서블릿 컨테이너를 실행하는 플랫폼
사용 용이성	사용과 설치가 매우 쉬움	쉬운 설치. XML 설정 파일을 수정하지 않음	설치 용이. 사용자 인터페이스/워크플로우 항상되어야 함 웹기반 설정

참조: <http://www.ibm.com/developerworks/java/library/j-ap09056/index.html>

Continuum 설치 (1/7)

□ 요구사항

- JDK 1.5 이상
- Memory 제약 없음
- Disk: Continuum 자체는 30MB 이하.
체크아웃과 소스빌드에 필요한 디스크 공간 확보 필요
- OS 제약 없음

□ 지원 **WAS**

- Standalone, Tomcat, Jboss, Jetty, Geronimo, Glassfish

□ 다운로드

- continuum 1.1 다운로드 (<http://continuum.apache.org/download.html>)
- 실습은 standalone 버전 (apache-continuum-1.1.zip)

참조: <http://continuum.apache.org/docs/1.1/installation/installation.html>

Continuum 설치 (2/7)

- ❑ **Standalone** 버전 압축을 설치 디렉토리에 푼다
- ❑ **\$CONTINUUM_HOME/conf/plexus.xml** 설정
 - Mail 서버 설정

```
...  
  
  <resource>  
    <name>mail/Session</name>  
    <type>javax.mail.Session</type>  
    <properties>  
      <property>  
        <name>mail.smtp.host</name>  
        <value>localhost</value>  
      </property>  
      <property>  
        <name>mail.smtp.port</name>  
        <value>25</value>  
      </property>  
  
      <!--  
      <property>  
        <name>mail.smtp.auth</name>  
        <value>true</value>  
      </property>  
    </properties>  
  </resource>  
...  

```

Continuum 설치 (3/7)

□ \$CONTINUUM_HOME/conf/plexus.xml 설정

- 데이터베이스 설정 (기본으로 Derby 데이터베이스를 내장)

```
...
<!--
  Datasources
-->
<resource>
  <name>jdbc/users</name>
  <type>javax.sql.DataSource</type>
  <properties>
    <property>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </property>

    <!-- Maximum number of dB connections in pool. Make sure you
         configure your mysqld max_connections large enough to handle
         all of your db connections. Set to 0 for no limit.
    -->
    <property>
      <name>maxActive</name>
      <value>100</value>
    </property>
  
```

Continuum 설치 (4/7)

□ \$CONTINUUM_HOME/conf/plexus.xml 설정

■ 포트변경

```
...  
<!-- START SNIPPET: jetty_port -->  
  <component>  
    <role>org.codehaus.plexus.contextualizer.Contextualizer</role>  
    <role-hint>jettyConfiguration</role-hint>  
    <implementation>org.codehaus.plexus.contextualizer.DefaultContextualizer</implementation>  
    <configuration>  
      <contextValues>  
        <jetty.port>9090</jetty.port>  
      </contextValues>  
    </configuration>  
  </component>  
<!-- END SNIPPET: jetty_port -->  
...
```

Continuum 설치 (5/7)

□ Windows 서비스 추가 (선택)

- \$CONTINUUM_HOME/bin/windows-x86-32/InstallService.bat 실행
- 서비스 창(services.msc)에서 Apache Continuum 편집
- 시작유형과 로그인 사용자 선택


□ 실행

- \$CONTINUUM_HOME/bin/windows-x86-32/run.bat 실행
- 실행하면서 내장한 Derby DB에 테이블 생성
(초기에 테이블 삭제 에러 발생 - 무시함)
- 다음 URL로 접속
<http://localhost:9090/continuum>

Continuum 설치 (6/7)

□ 관리자 생성

- 초기 접속 시 관리자 생성 페이지 나타남
- 관리자 정보를 넣고 “Create Admin” 버튼 클릭




Login - Register


Continuum


About


Show Project Groups

Legend

Build Now 

Build History 

Build In Progress 

Working Copy 

Create Admin User

Username: admin

Full Name*:

Email Address*:

Password*:

Confirm Password*:

Continuum 설치 (7/7)

□ 일반적인 설정 입력

- 최초 관리자 로그인을 하면 일반설정을 입력을 요구함



Current User: Dongyoul, Kim (admin) - Edit Details - Logout

Continuum | Maven | Apache

Continuum

About

Show Project Groups

Add Project

Maven 2.0.x Project

Maven 1.x Project

Ant Project

Shell Project

Administration

Schedules

Installations

Profiles

Queues

General Configuration

Working Directory*:	C:\demo\continuum-1.1\apps\continuum\webapp\WEB-INF\working-directory
Enter the working directory of the Continuum web application	
Build Output Directory*:	C:\demo\continuum-1.1\apps\continuum\webapp\WEB-INF\build-output-directory
Enter the build output directory of the Continuum web application	
Deployment Repository Directory:	
Enter the deployment repository directory of the Continuum web application	
Base URL*:	http://localhost:9090/continuum
Enter the base URL for the Continuum web application	

Save Cancel

Continuum 활용 (1/7)

□ 프로젝트 그룹 관리

- 여러 개의 프로젝트를 하나의 그룹으로 관리할 수 있음
- 프로젝트 그룹내의 프로젝트는 동일한 관리 설정들을 가질 수 있음
- 프로젝트 그룹은 프로젝트 등록과 함께 자동 생성되기도 함

- “Show Project Groups”를 선택하면 프로젝트 그룹 목록 나타남
- “Add Project Group” 버튼을 클릭하여 프로젝트 그룹 등록
 - Project Group Name: 예) Basic POM
 - Project Group Id: 예) kr.nexttree.common.pom
 - Description: 프로젝트 그룹 설명

Continuum 활용 (2/7)

□ 프로젝트 관리

- Continuum은 Maven1, Maven2, Ant, Shell 프로젝트를 관리할 수 있음
- Maven 2.0.x 프로젝트를 추가
추가하려는 프로젝트 POM에 scm 설정이 되어 있어야 함

예) 아래 그림

- POM Url: 프로젝트 POM의 SVN URL
예) <http://svn.nextree.co.kr/common/trunk/BasicPOM/pom.xml>
- Username: SCM(Subversion) 접속 아이디
- Password: SCM(Subversion) 접속
- Project Group: 디폴트로 POM의 정보로 그룹 생성. 선택 가능

```
...  
<scm>  
  <connection>scm:svn:http://svn.nextree.co.kr/common/trunk/BasicPOM</connection>  
</scm>  
...
```

참조: http://continuum.apache.org/docs/1.1/user_guides/managing_project/addProject.html

Continuum 활용 (3/7)

□ 프로젝트 그룹에 프로젝트 추가 예

Project Group Summary

Members

Build Definitions

Notifiers

Project Group Informations

Project Group Name:

Common

Project Group Id:

kr.nextree.common

Description:

Common Project

Group Actions

Default Build Definition

Build all projects

Edit

Delete Group

Release

Add New Project

Add

























Project Group Last Build Result Overview

Success : 4

Errors : 0

Failed : 0

Member Projects

	Project Name	Version	Build	Group					
<input type="checkbox"/>	 Basic Config	1.0-SNAPSHOT	5	Common					
<input type="checkbox"/>	 Basic POM	1.0-SNAPSHOT	4	Common					
<input type="checkbox"/>	 Test Config	1.0-SNAPSHOT	2	Common					
<input type="checkbox"/>	 Utilities	1.0-SNAPSHOT	17	Common					

Delete Project(s)

Default Build Definition

Build Project(s)


Cancel Build(s)

Select All


Unselect All

Continuum 활용 (4/7)

























□ 수동 빌드

- 빌드는 Build 정책(Build Definition)에 따라 자동 수행
- 그러나 필요에 딸 수동 빌드도 가능
- 프로젝트 그룹 정보에서 프로젝트의  버튼을 클릭
- 또는 빌드할 프로젝트를 선택하고 "Build Project(s)" 버튼을 클릭

□ 빌드 상황 보기

- 빌드 중인 프로젝트는  로 아이콘이 변경. 이 아이콘 클릭

☞ Member Projects

	Project Name	Version	Build	Group					
<input type="checkbox"/>	 <u>Basic Config</u>	1.0-SNAPSHOT	<u>5</u>	Common					
<input type="checkbox"/>	 <u>Basic POM</u>	1.0-SNAPSHOT	<u>4</u>	Common					
<input type="checkbox"/>	 <u>Test Config</u>	1.0-SNAPSHOT	<u>7</u>	Common					
<input type="checkbox"/>	 <u>Utilities</u>	1.0-SNAPSHOT	<u>17</u>	Common					

[Select All](#)
[Unselect All](#)

Continuum 활용 (5/7)

□ Build Definition

- 각 프로젝트나 프로젝트 그룹은 빌드하기 위한 **Build Definition**을 가져야 함
- 프로젝트에 따라 서로 다른 빌드 정책을 정의할 수 있다
- 디폴트 **build definition**을 제공한다.
- 프로젝트 수준이나 프로젝트 그룹 수준에서 **build definition**을 적용할 수 있다

□ Build Definition 관리

- 프로젝트나 프로젝트 그룹에서 **Build Definition** 메뉴를 이용
 - **POM filename**: 빌드할 **POM** 파일 이름
 - **Goals**: 빌드 시 수행할 **goal** 들
 - **Arguments**: 빌드 시 사용할 아규먼트
 - **Build Fresh**: 빌드 때 마다 **SCM update**하지 않고 **clean** 체크아웃 할 지 여부
 - **Always Build**: 항상 빌드
 - **Schedule**: 빌드에 사용할 스케줄

Continuum 활용 (6/7)

□ 스케줄 관리

- 스케줄 관리를 통해 빌드 스케줄을 지정할 수 있다
- 메인 메뉴의 "Schedules"를 선택하여 스케줄을 추가, 편집
 - Cron Express: 스케줄은 Unix Cron 스타일로 지정
예) 0 0 0/2 * * ? → 매 2시간마다
 - Max Job Time: 작업 최대 시간을 초단위로 입력
 - Enabled: 스케줄 사용 여부, 체크가 되어야만 작동한다
- Build Definition에 스케줄 설정
 - Build Definition의 편집 메뉴에서 새로 등록된 스케줄을 선택할 수 있다

☞ Add/Edit Build Definition

POM filename*:	<input type="text" value="pom.xml"/>
Goals:	<input type="text" value="clean install"/>
Arguments:	<input type="text" value="--batch-mode --non-recursive"/>
	<input type="checkbox"/> Build Fresh (Run always a clean checkout instead of an SCM update)
	<input type="checkbox"/> Always Build
Is it default?:	true
Schedule:	DEFAULT_SCHEDULE ▼
Profile:	DEFAULT_SCHEDULE My Schedule
Type:	maven2 ▼
Description:	<input type="text" value="default maven2 buildDefinition"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

Continuum 활용 (7/7)

□ 통지 관리

- 자동 빌드의 상황을 담당자에게 통지하는 기능
- 프로젝트 그룹이나 프로젝트의 **Notifies** 메뉴에서 “Add” 선택
- 통지방법 선택
Type: Mail, IRC, Jabber, MSN, Wagon
- Mail 설정 예
 - Mail Recipient Address: 통지 받을 메일 주소
- 통지 상황 선택
 - 자동 빌드 상황에 따라 어느 경우 통지할 지 결정

Mail Recipient Address:

☐ Send a mail to latest committers

☐ Send on Success

☐ Send on Failure

☐ Send on Error

☐ Send on Warning