소프트웨어 검증 및 실습

201111353 박수민
201111371 원정일
201111386 조경래

목차

- CI
- Eclipse
- JUnit
- ANT

CI

지속적 통합

http://www.slideshare.net/sunnykwak90/ss-59267532

CI
지속적 통합이란

- 소프트웨어공학(SE, Software Engineering) 관점에서, 지속적 통합(CI, Continuous Integration)은 지속적으로 품질제어(Quality Control)를 실행하는 것이다.

- 지속적 통합은 모든 개발을 완료한 뒤에 품질제어를 적용하는 고전적인 방법을 대체하는 방법으로서 소프트웨어의 질적 향상과 소프트웨어를 배포하는데 걸리는 시간을 줄이는데 초점이 맞추어져 있다.

CI
지속적 통합의 등장

- 소프트웨어 공학(SE)은 소프트웨어를 '어떻게 하면 잘 만들 수 있나?' 를 고민하는 학문 분야이다. 좋은 소프트웨어는 너무나 당연한 얘기지만 필요한 기능이 모두 제대로 동작하고, 실행 중 비정상적으로 중단되지 않으며 자료가 훼손되지 않아야 한다.
- 따라서, SE는 소프트웨어의 품질을 검사하기 위해 소프트웨어 완성 단계에 개발자가 아닌 'QA, 품질담당자'가 객관적으로 소프트웨어의 품질을 검사해야한다고 권고한다.
- 소프트웨어는 완성하기 전에 기능이나 UI를 확인할 수 없다는 특성 때문에 개발과정에서는 품질을 측정할 수 없다는 의견이 지배적이었다.
- 모든 개발을 마친 후에 품질 검사를 수행하면, 검사하는데 드는 시간과 노력이 많이 드는데다, 심각한 결함이 발견되면, 전체 일정에 차질을 빚고 만다.
- 발상의 전환을 통해 개발하는 도중에도 자주 반복적으로 빌드(혹은 통합)을 하고, 기능을 테스트하고, 품질을 검증하자는 것이 '지속적인 통합'이다.

CI

Q. 지속적 통합은 프로그램을 구현하는 방법인가?
A. 아닙니다. 좋은 소프트웨어를 만들기 위한 행동지침입니다.

- 지속적 통합은 익스트림 프로그래밍(eXtreme Programing, XP) 커뮤니티에서 나왔으며, XP 방법론의 지지자인 마틴 파울러와 켄트 벡 이 1999년 경 처음으로 지속적 통합에 대해 글을 썼다

CI

마틴 파울러 Martin Fowler

- 지속적 통합은 여러명으로 구성된 팀이 작업한 것을 자주 통합하는 것을 가리키는 소프트웨어 개발 활동으로서 여기서 자주는 각 팀원의 작업을 적어도 하루에 한번 이상, 매일 여러번의 통합을 하는 것을 의미한다. 매번 이루어지는 통합은 자동화된 빌드에 의해 통합 에러가 없는지 가능한 빨리 확인하고 검증된다.

- http://martinfowler.com/articles/continuousIntegration.html

CI

왜 지속적인 통합을 해야 하는가?

- 통합(빌드)은 어렵고, 시간과 비용(노력)은 점차 기하급수적으로 증가하는데, 그 원인은 다음과 같다.
  - 컴포넌트 혹은 기능의 지속적인 증가
  - 코드의 증가에 비례하는 버그의 폭발
  - 프로그램이 복잡해질수록 빌드 주기가 길어짐

CI
지속적 통합의 이점

- 프로젝트 관리
  - 시스템 개발의 문제점을 가급적 빨리 발견
  - 비용, 일정, 예산 상의 위험(risk) 감소
- 코드 품질
  - 측정 가능하고, 시각적으로 확인 할 수 있는 코드 품질 확보
  - 지속적이고 자동화된 회귀 단위 테스트 수행 가능.

Eclipse

통합 개발 환경

소개

- 다양한 플랫폼에서 쓸 수 있으며, 자바를 비롯한 다양한 언어를 지원하는 프로그래밍 통합 개발 환경을 목적으로 시작하였으나, 현재는 OSGi를 도입하여, 범용 응용 소프트웨어플랫폼으로 진화하였다.
- 주요기능 : Code Edit / Compile / Build / Unit Test, Debug
- 라이센스 : EPL(ECLIPSE PUBLIC LICENSE) / 무료
- 특징
  - 하나의 도구 안에서 JAVA Code를 작성하고, 이에 대한 debugging 기능을 제공하여 사용자가 쉽고 빠르게 SW개발을 할 수 있는 통합 개발환경(IDE : Integrated Development Environment)
- http://www.eclipse.org/

목차

- JDK 설치 하기
- Eclipse 설치하기
- Eclipse 사용하기

JDK 설치하기

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

사용자의 운영체제에 맞게 선택하여 다운 받으면 된다.

Java SE Development Kit 8u73

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.73 MB	jdk-8u73-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.68 MB	jdk-8u73-linux-arm64-vfp-hflt.tar.gz
Linux x86	154.75 MB	jdk-8u73-linux-i586.rpm
Linux x86	174.91 MB	jdk-8u73-linux-i586.tar.gz
Linux x64	152.73 MB	jdk-8u73-linux-x64.rpm
Linux x64	172.91 MB	jdk-8u73-linux-x64.tar.gz
Mac OS X x64	227.25 MB	jdk-8u73-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.7 MB	jdk-8u73-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.08 MB	jdk-8u73-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.36 MB	jdk-8u73-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u73-solaris-x64.tar.gz
Windows x86	181.5 MB	jdk-8u73-windows-i586.exe
Windows x64	186.84 MB	jdk-8u73-windows-x64.exe

Java SE Development Kit 8u74

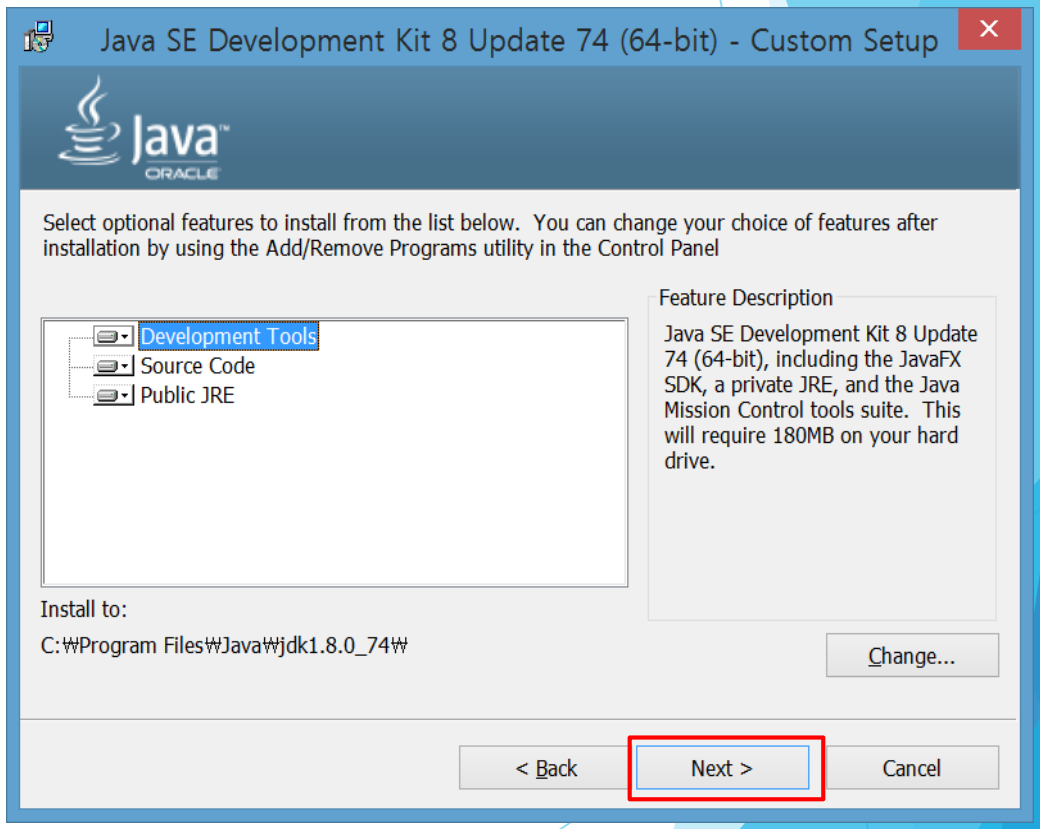
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux x86	154.74 MB	jdk-8u74-linux-i586.rpm
Linux x86	174.92 MB	jdk-8u74-linux-i586.tar.gz
Linux x64	152.74 MB	jdk-8u74-linux-x64.rpm
Linux x64	172.9 MB	jdk-8u74-linux-x64.tar.gz
Mac OS X x64	227.27 MB	jdk-8u74-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.72 MB	jdk-8u74-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.09 MB	jdk-8u74-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140.02 MB	jdk-8u74-solaris-x64.tar.Z
Solaris x64	96.19 MB	jdk-8u74-solaris-x64.tar.gz
Windows x86	182.01 MB	jdk-8u74-windows-i586.exe
Windows x64	187.31 MB	jdk-8u74-windows-x64.exe

JDK 설치하기


Java SE Development Kit 8 Update 74 (64-bit) - Setup
Java
ORACLE
Welcome to the Installation Wizard for Java SE Development Kit 8 Update 74
This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 74.
The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.
Next >
Cancel


Java SE Development Kit 8 Update 74 (64-bit) - Custom Setup
Java
ORACLE
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel
Development Tools
Source Code
Public JRE
Feature Description
Java SE Development Kit 8 Update 74 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.
Install to:
C:₩Program Files₩Java₩jdk1.8.0_74₩
Change...
< Back
Next >
Cancel

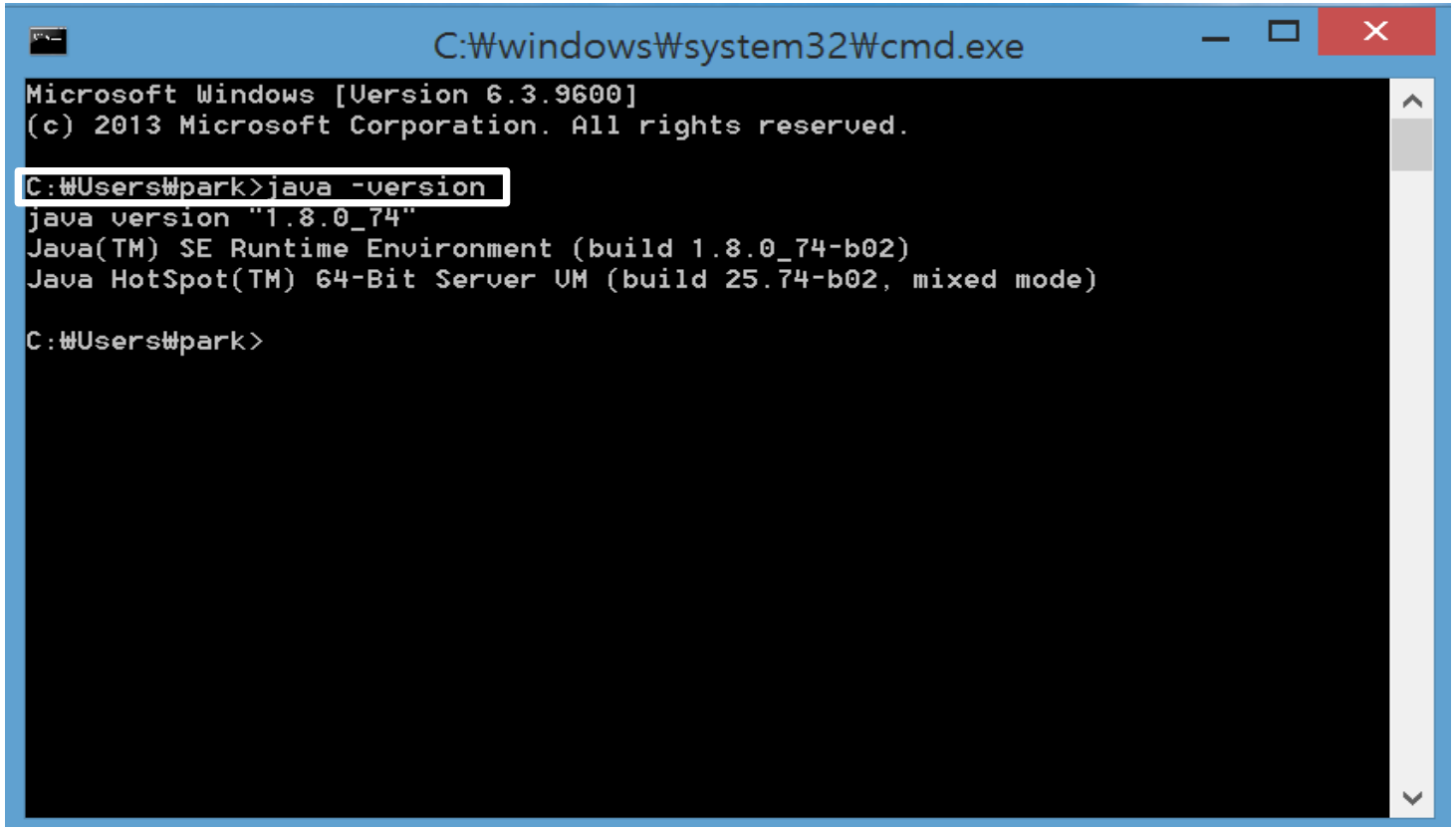
JDK 설치하기


Java 설치 - 대상 폴더
Java
ORACLE
대상 폴더
Java를 다른 폴더에 설치하려면 "변경"을 누르십시오.
설치 위치:
C:\Program Files\Java\jre1.8.0_74
변경(C)...
< 뒤로(B)
다음(N) >


Java SE Development Kit 8 Update 74 (64-bit) - Complete
Java
ORACLE
Java SE Development Kit 8 Update 74 (64-bit) Successfully Installed
Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.
Next Steps
Close

JDK 설치하기

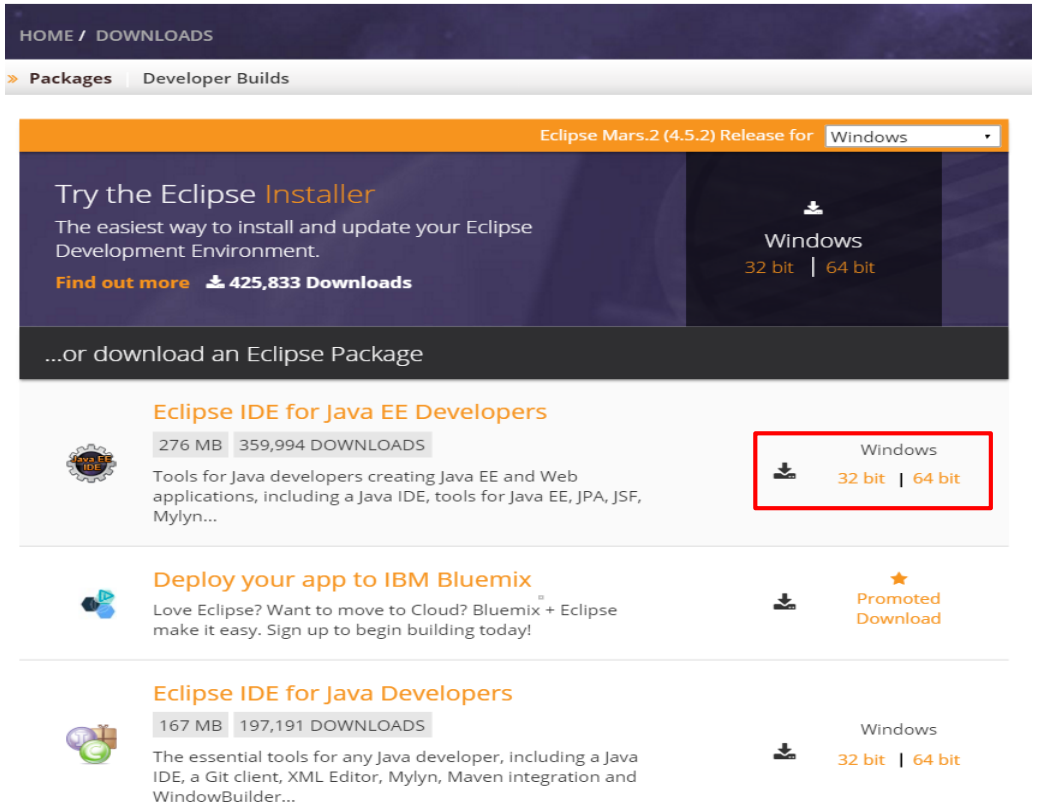
cmd 창에서 java –version 명령어를 쳤을 때 다음과 같이 나오면 정상적으로 설치된 것이다.

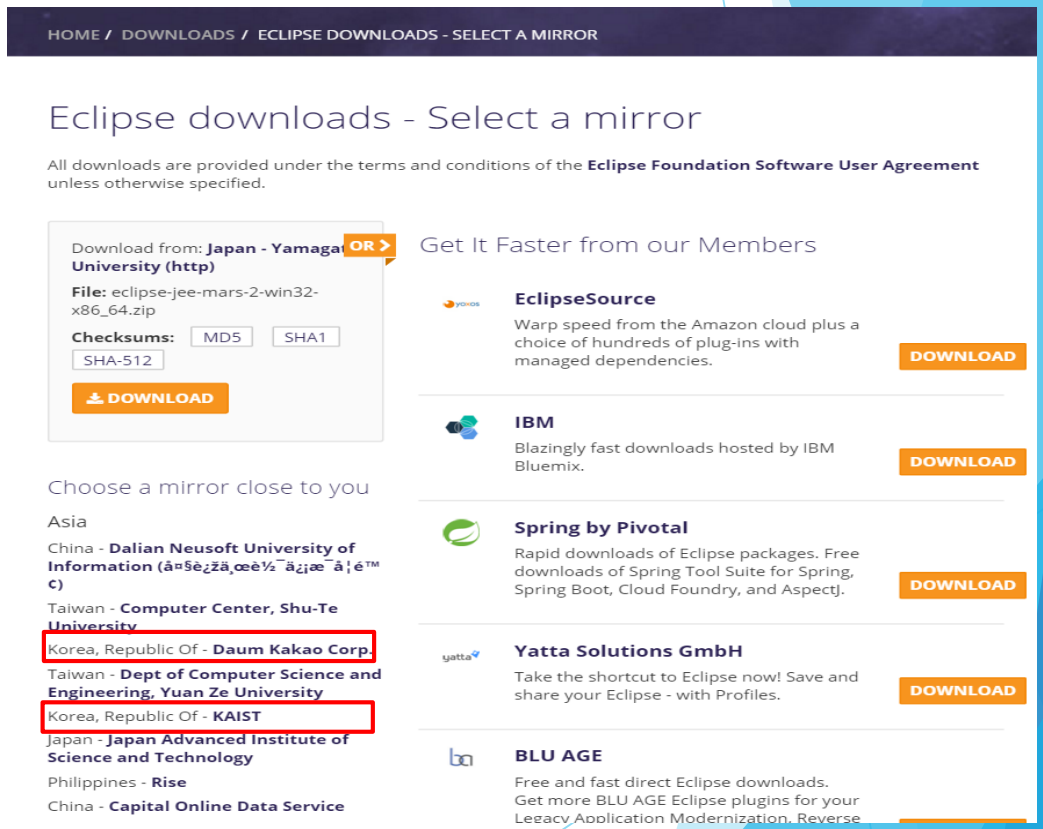


Eclipse설치하기

https://www.eclipse.org/downloads/index-developer.php

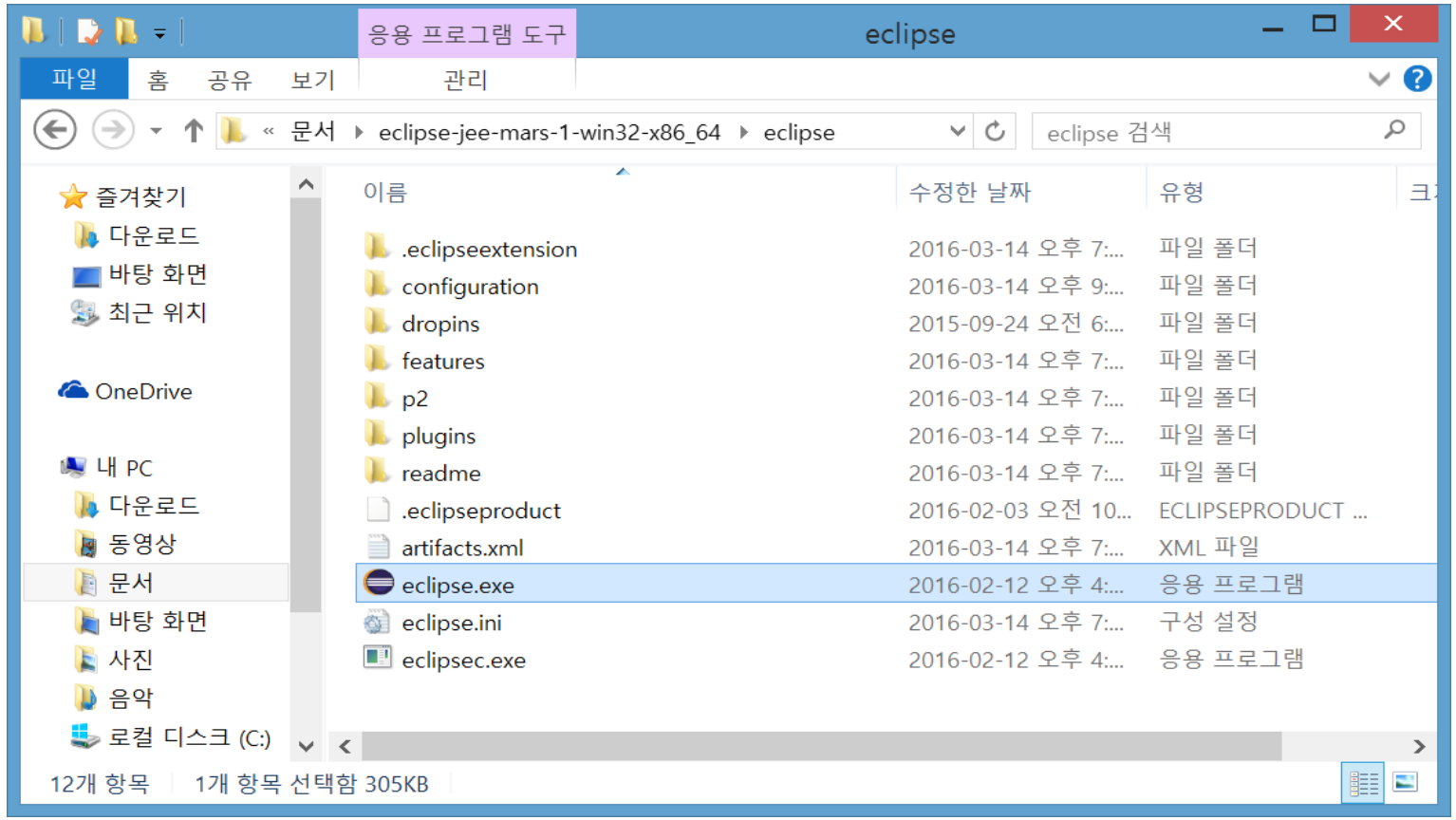
사용자의 운영체제에 맞게 선택하여 다운 받으면 된다.





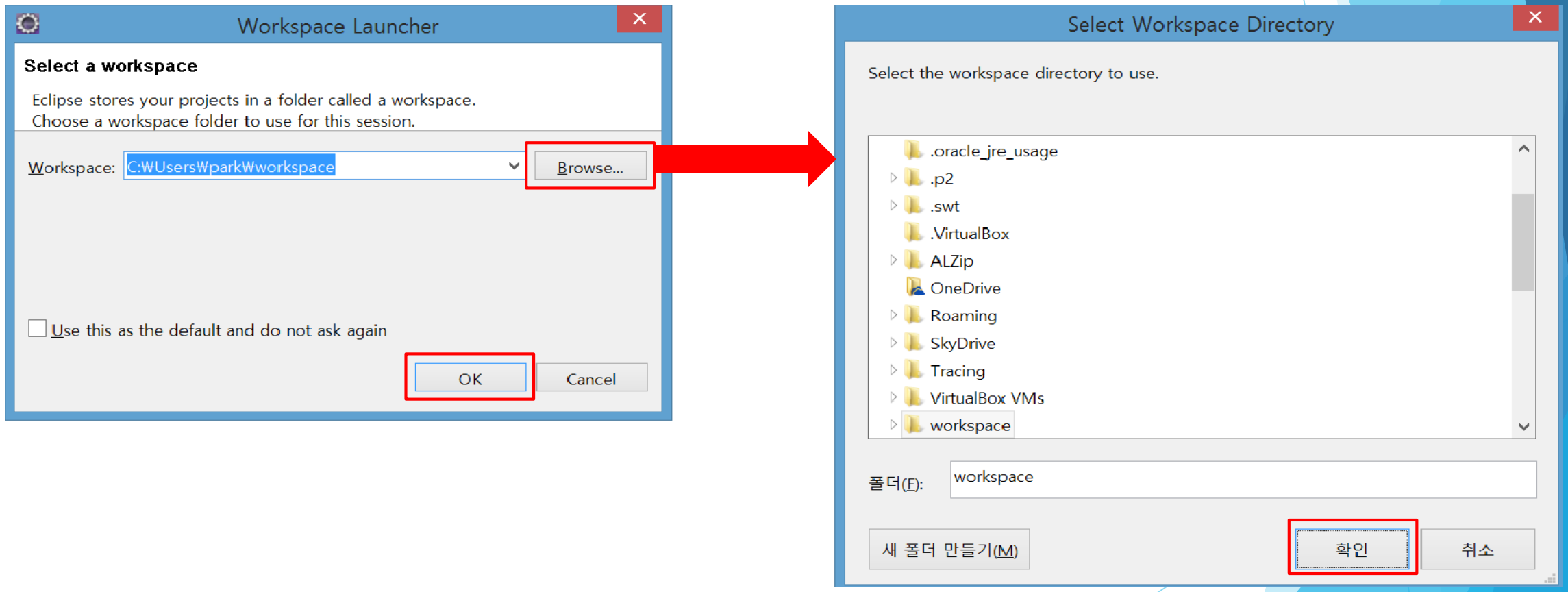
Eclipse 사용하기

Eclipse가 설치된 directory를 찾아 exe파일을 실행 시킨다.



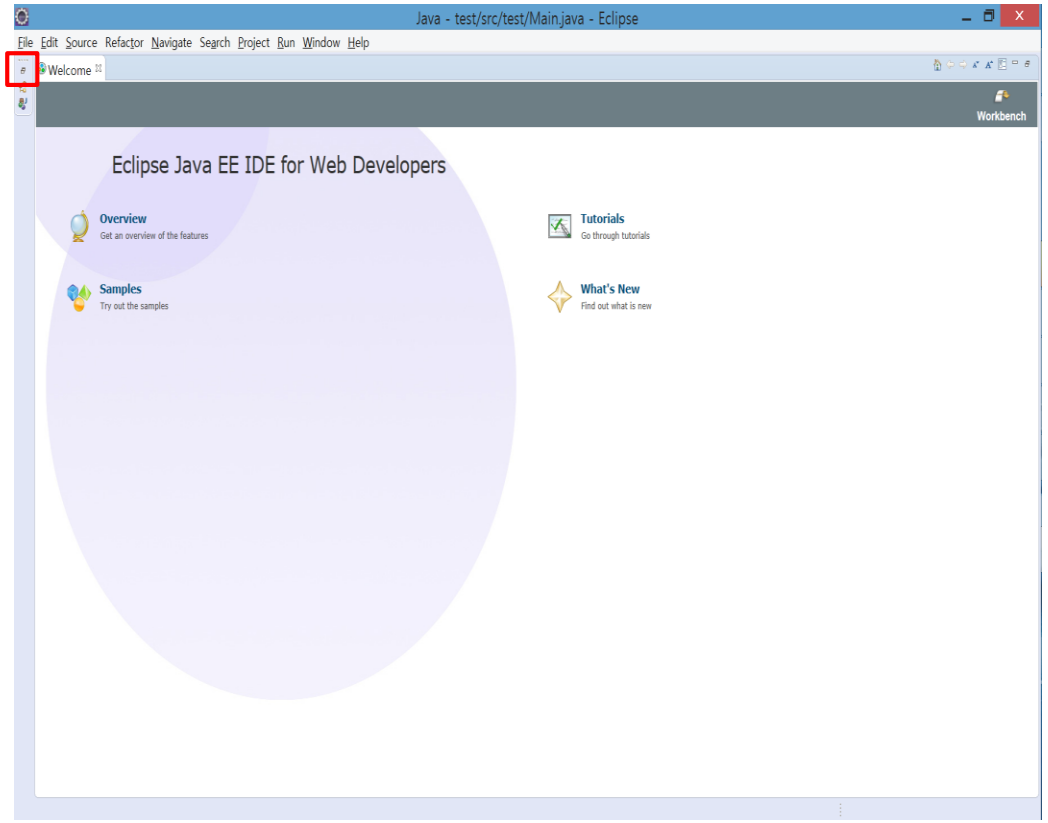
Eclipse 사용하기

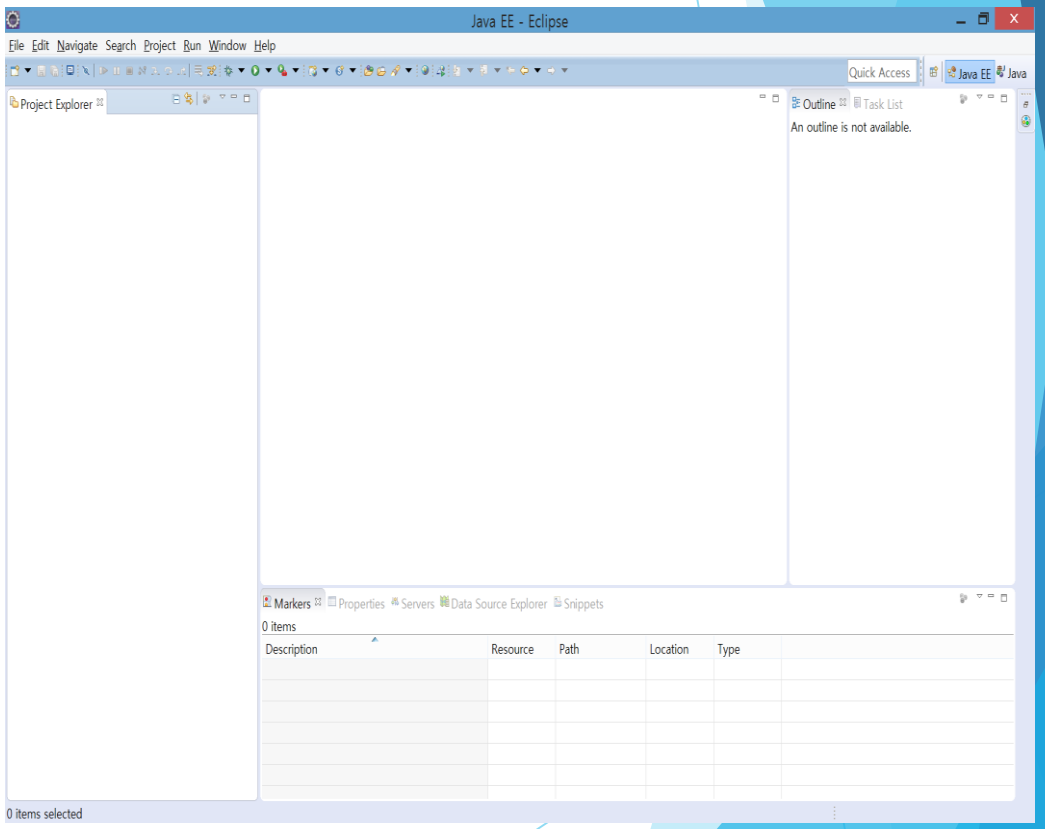
작성한 프로젝트가 저장될 workspace를 설정한다.



Eclipse 사용하기

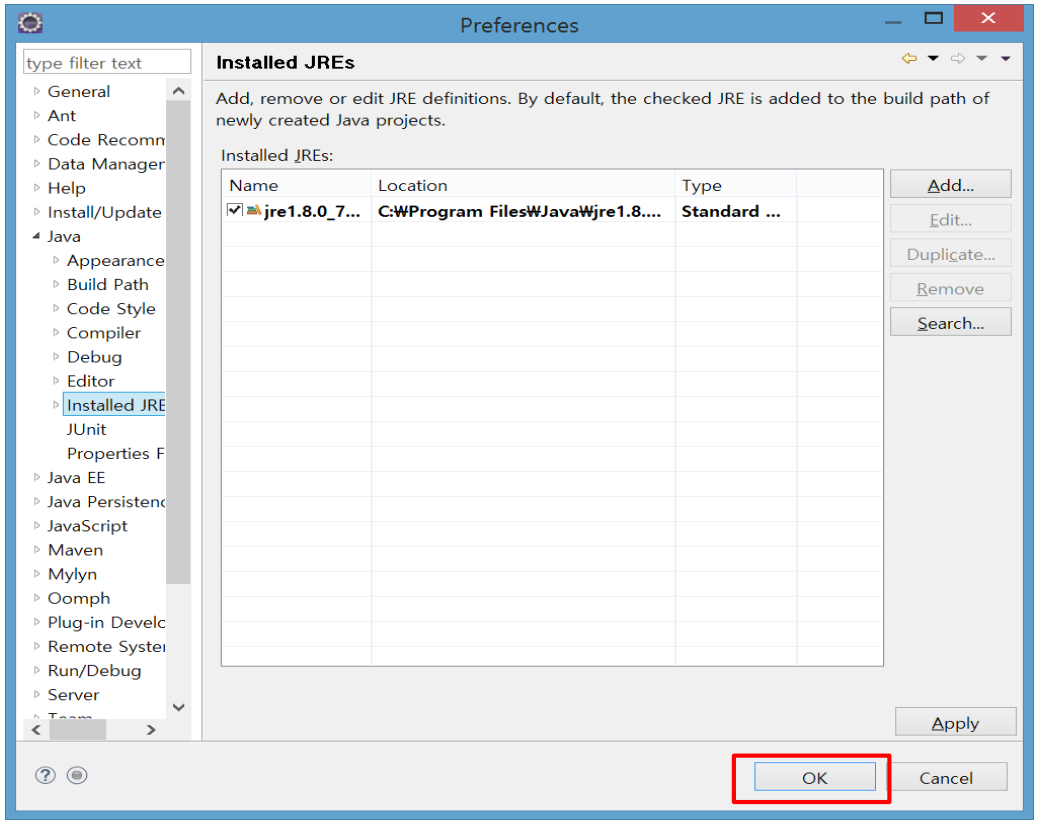
Welcome 초기 화면과 기본 화면



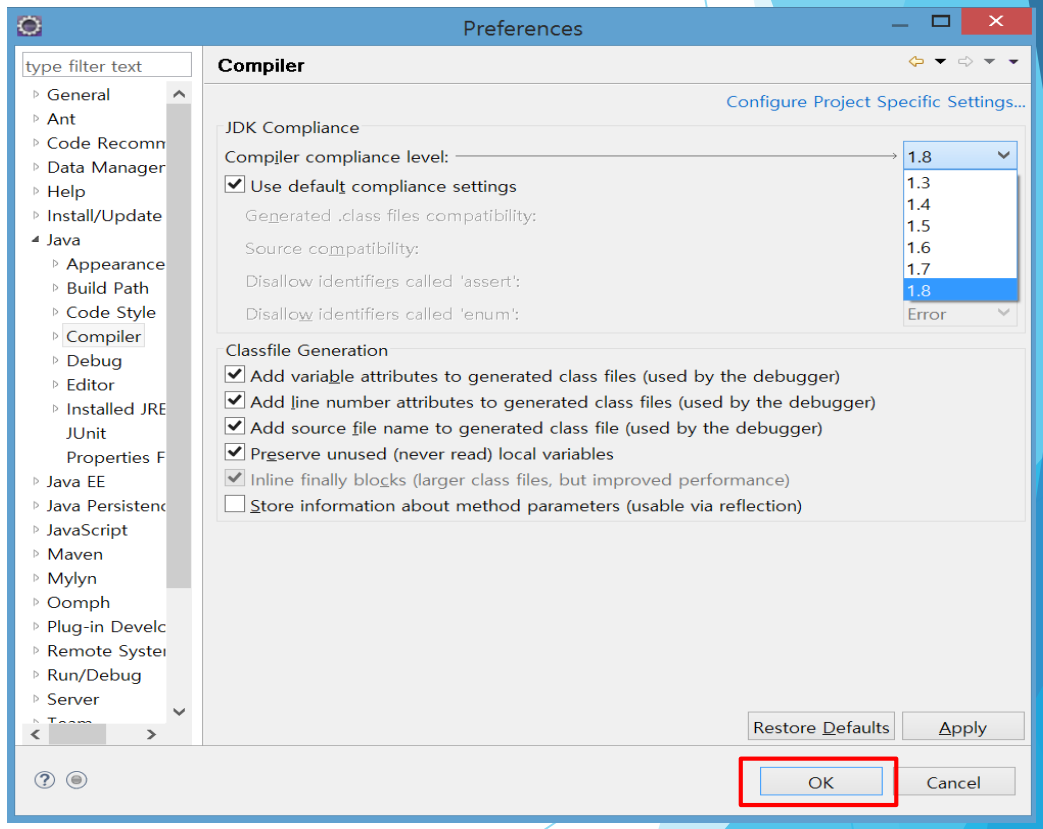


Eclipse 사용하기

Eclipse에서 jre가 설치되어 있는지 확인하는 방법

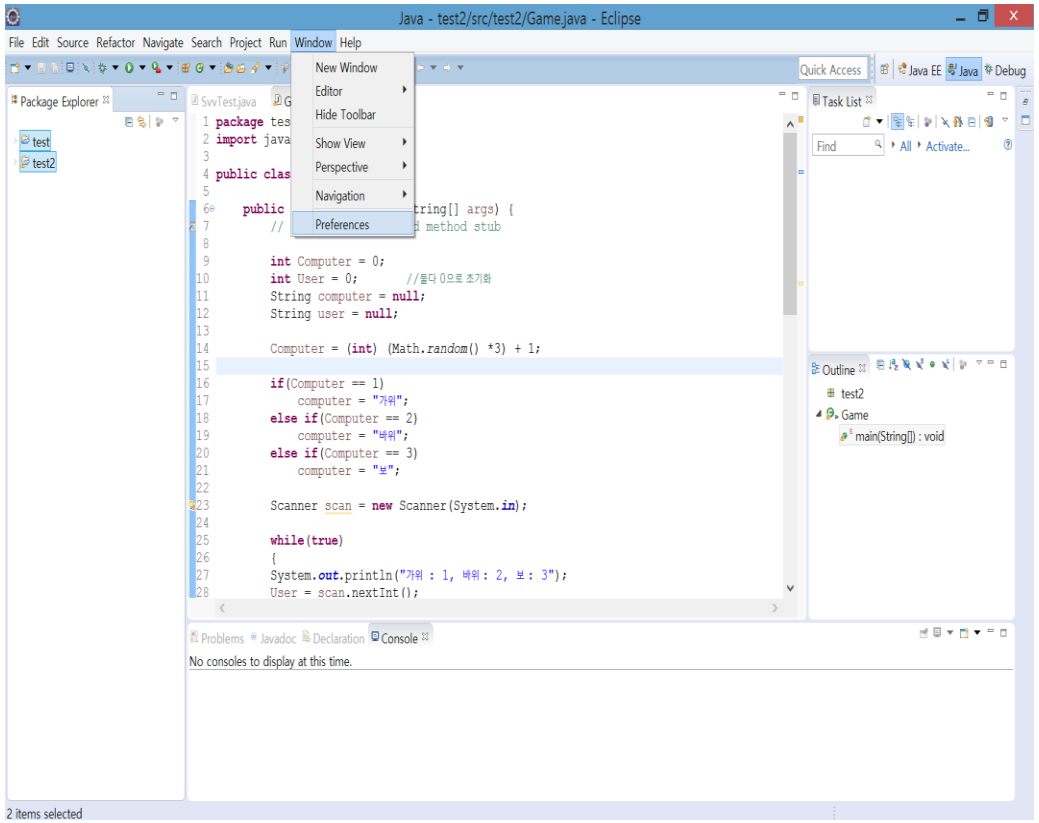


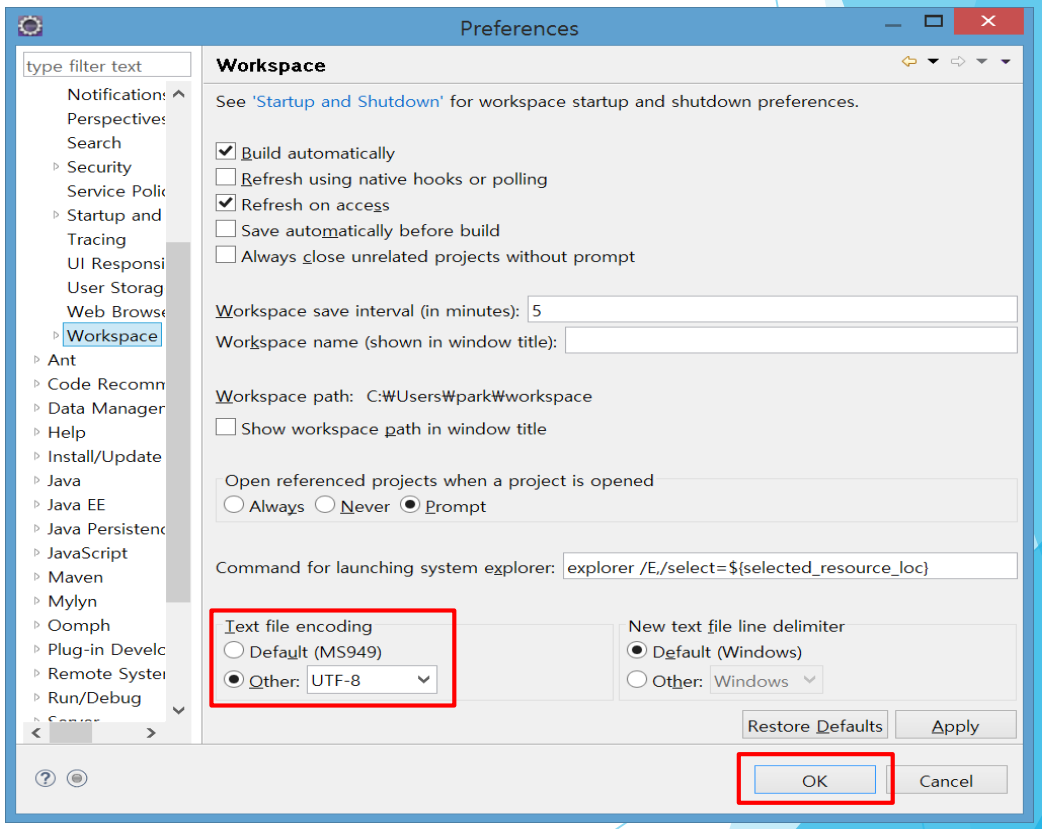
Compiler는 jdk 1.8로 통일한다



Eclipse 사용하기

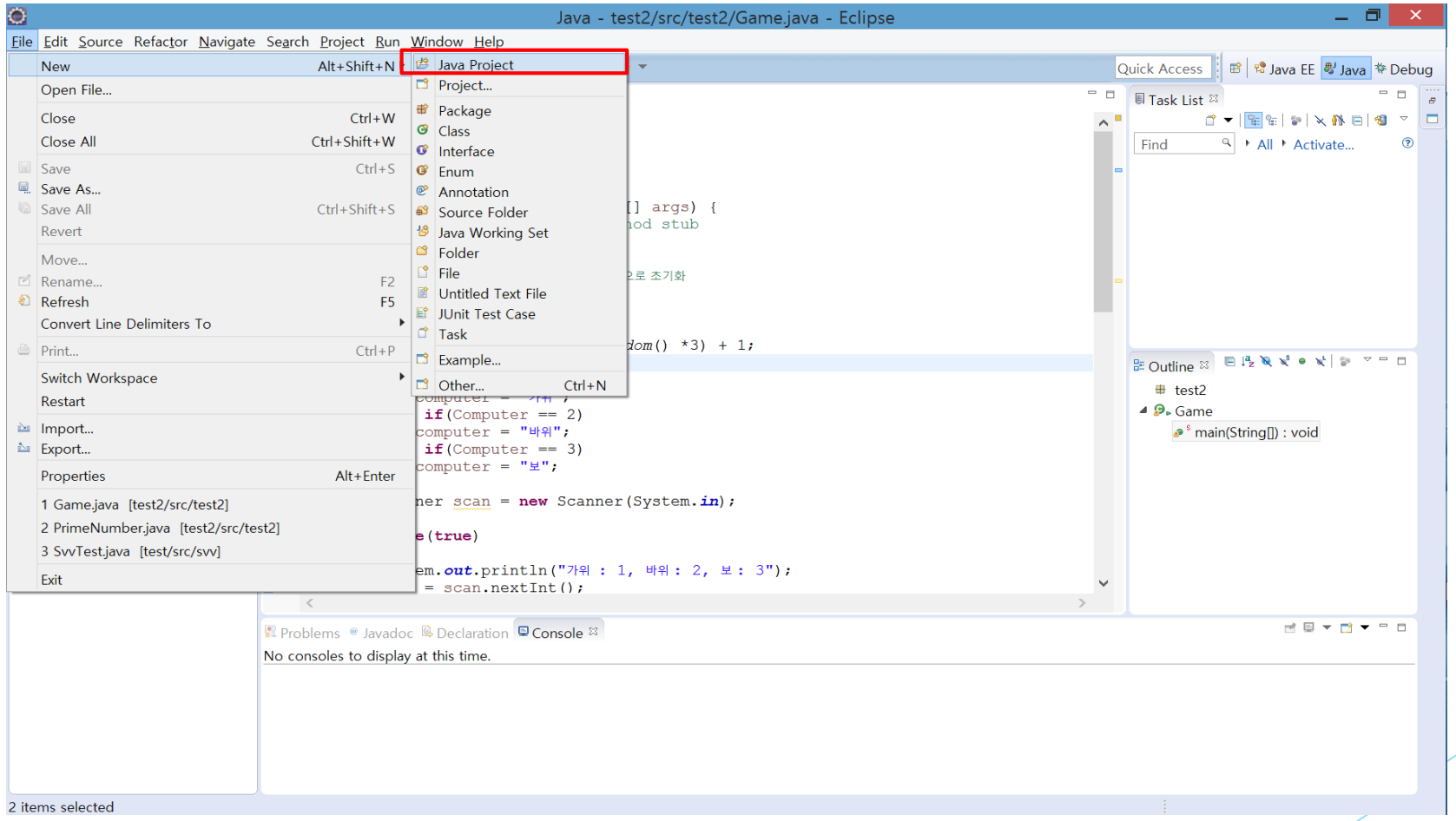
Team Project를 위해 Encoding Type을 UTF-8로 설정한다.





Eclipse 사용하기

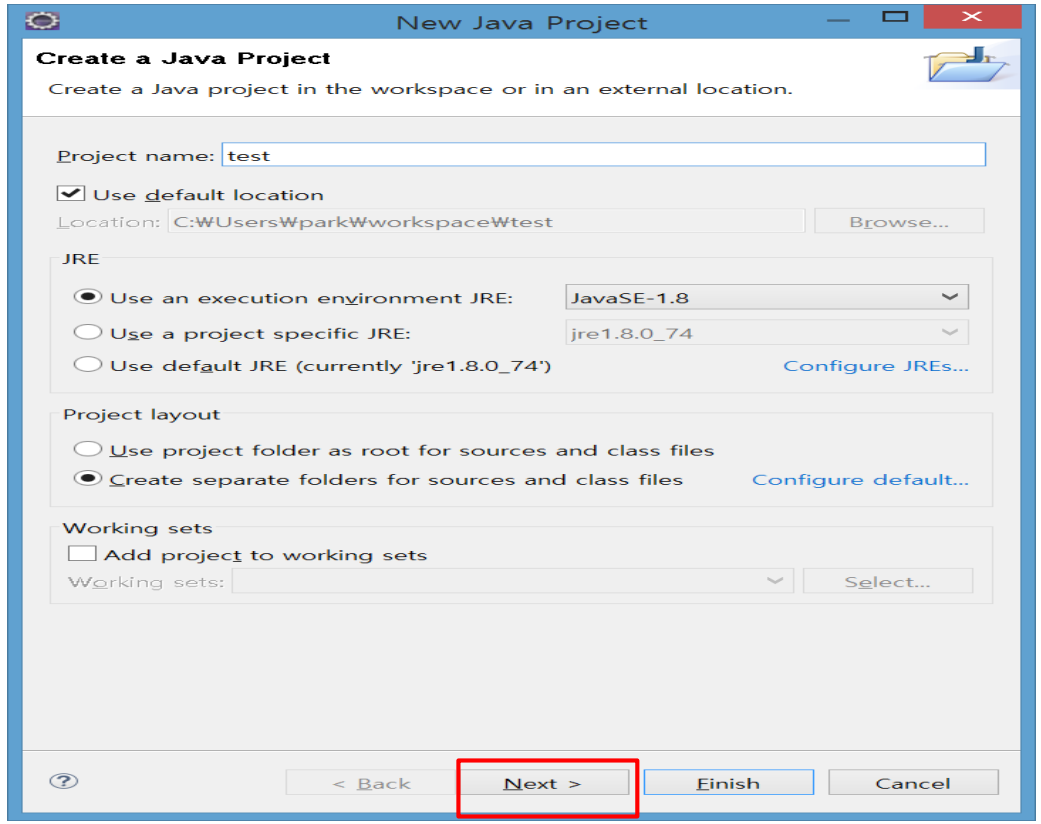
New Project



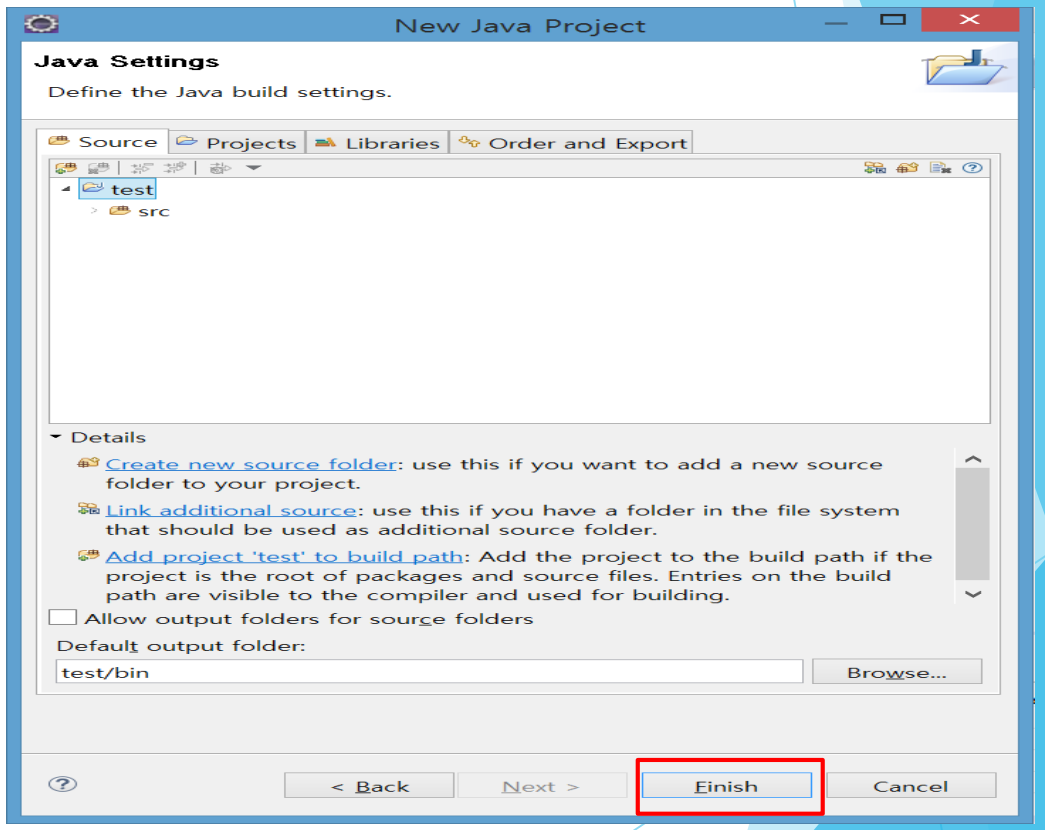
Eclipse 사용하기

New Project

프로젝트 이름을 입력한다.

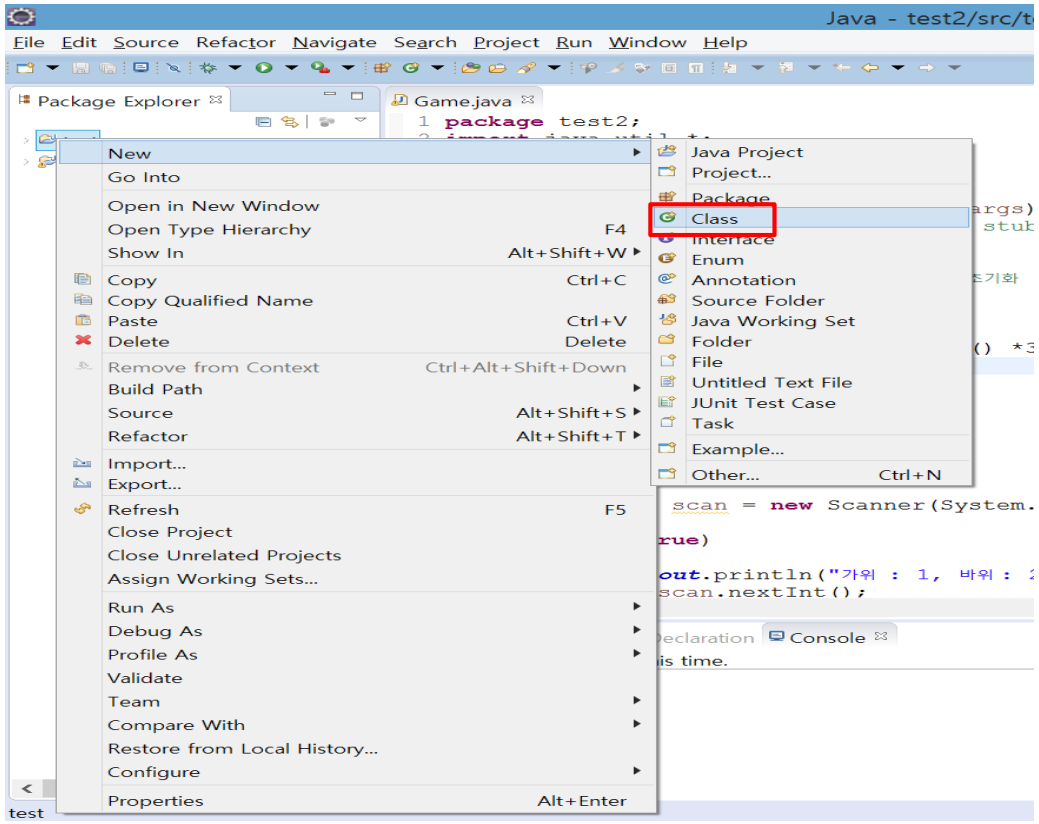


Library 추가 등 세부적인 Setting

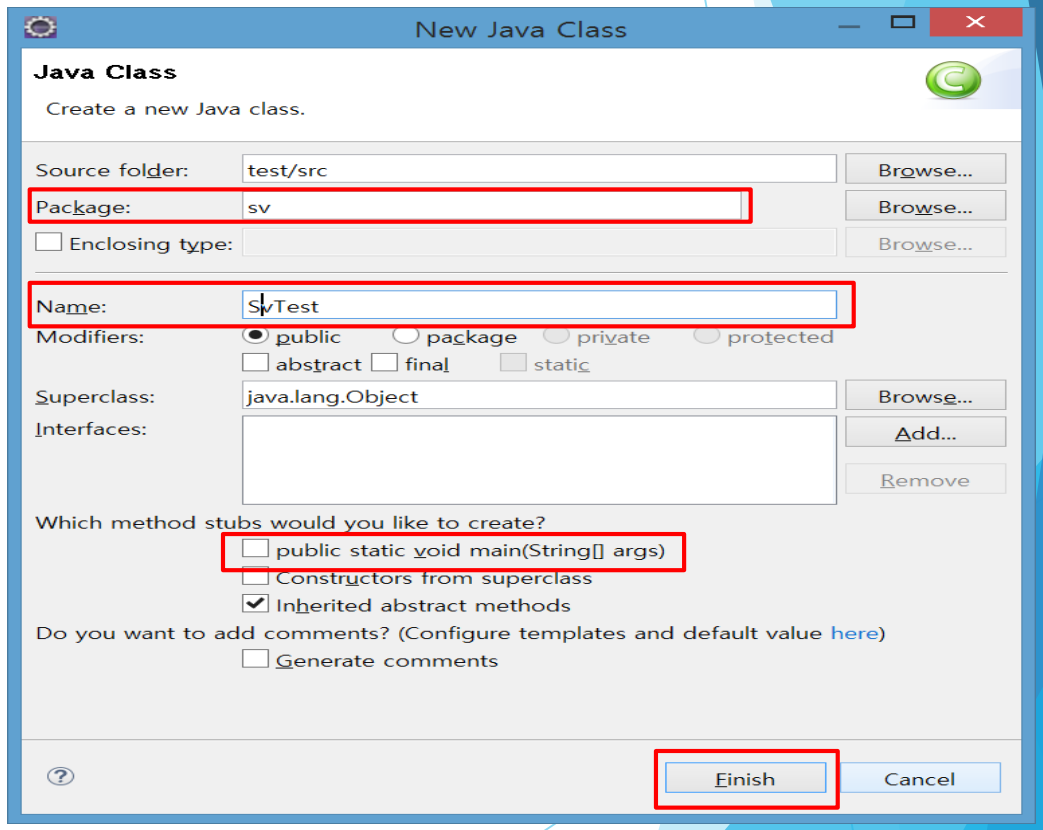


Eclipse 사용하기

클래스를 생성한다

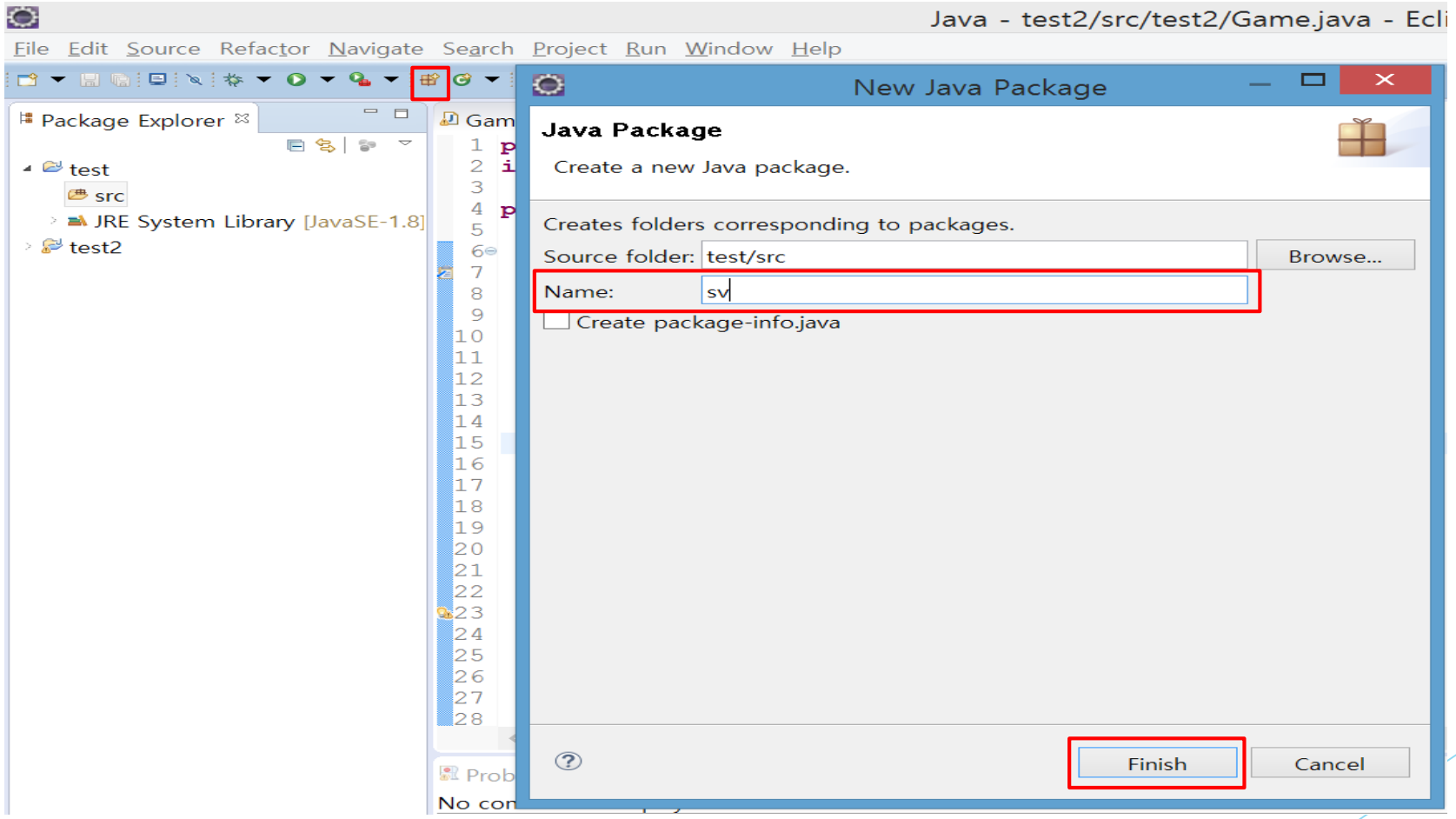


Package 이름, 클래스 이름을 정의한다



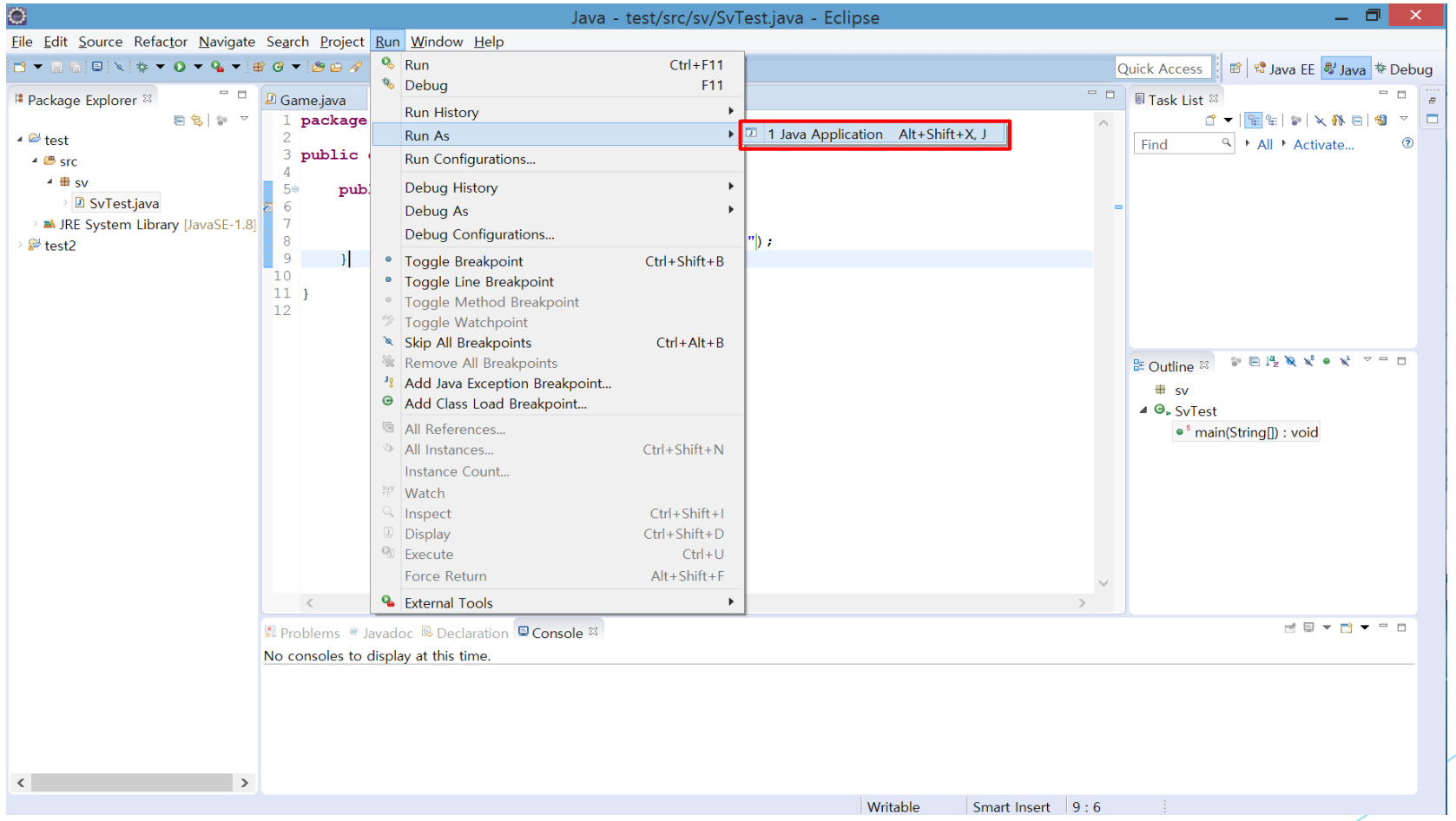
Eclipse 사용하기

Package를 생성하는 다른 방법



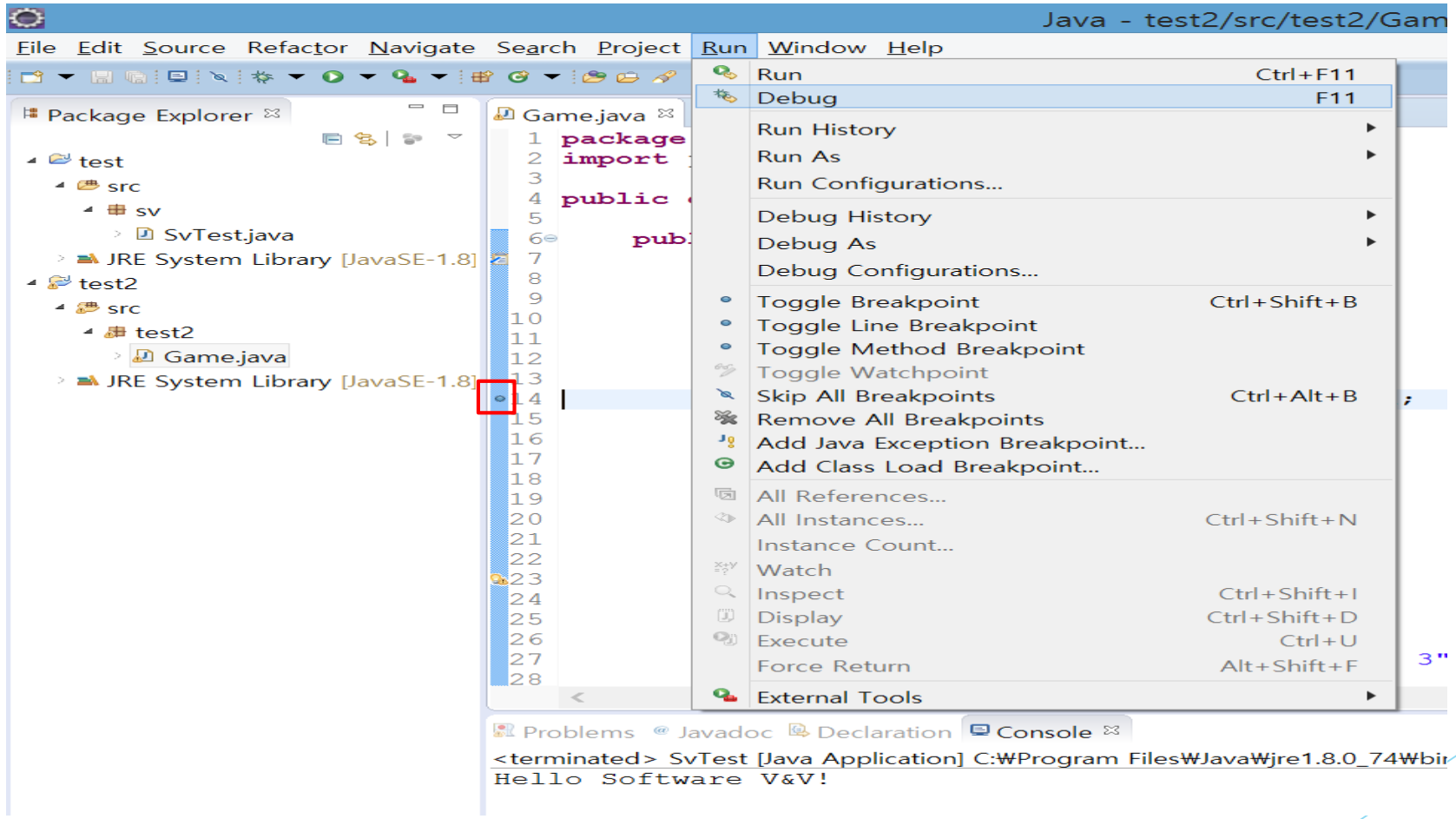
Eclipse 사용하기

Project Run

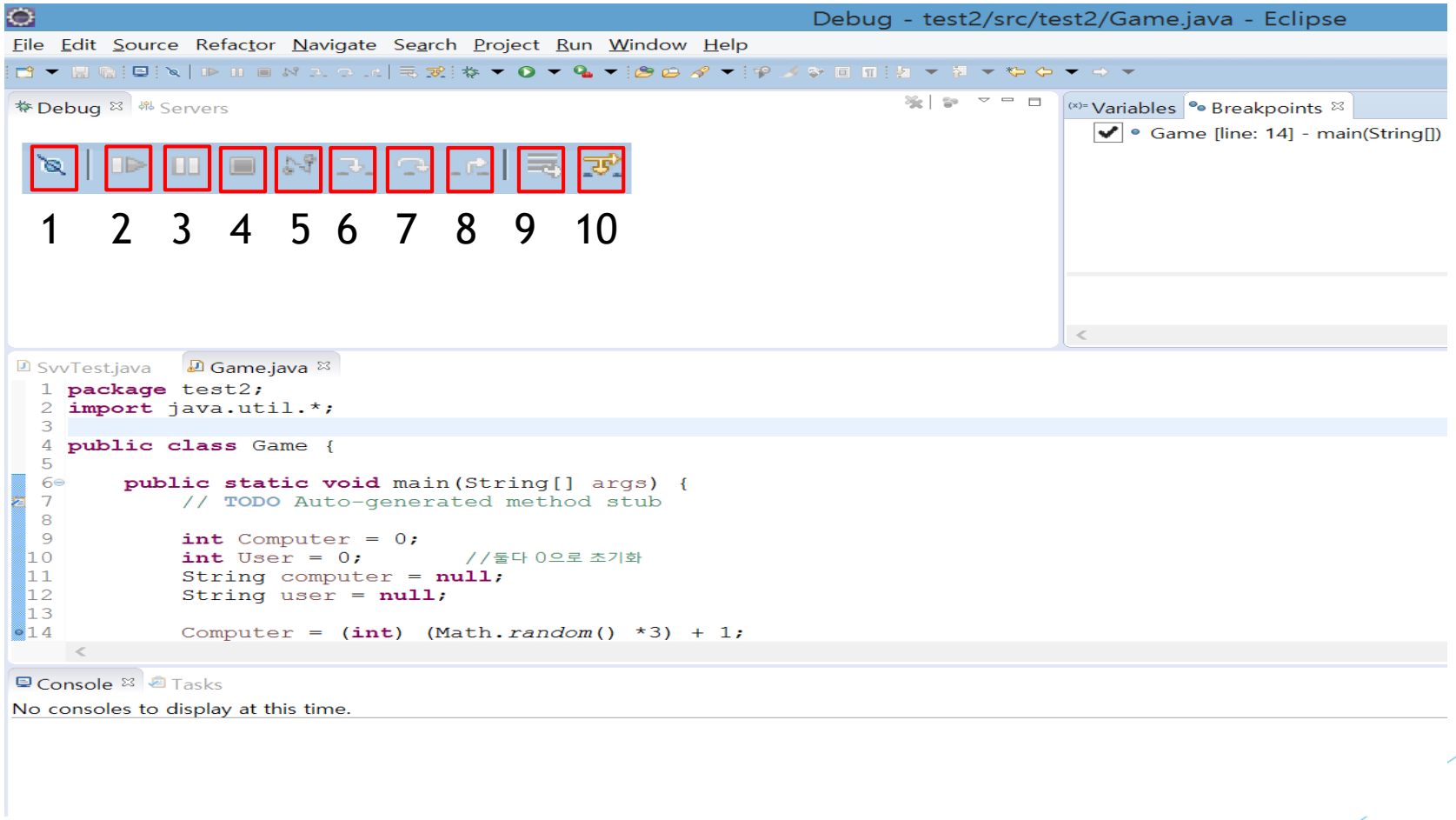


Eclipse 사용하기

Breakpoint & Debug



Eclipse 사용하기
Breakpoint & Debug

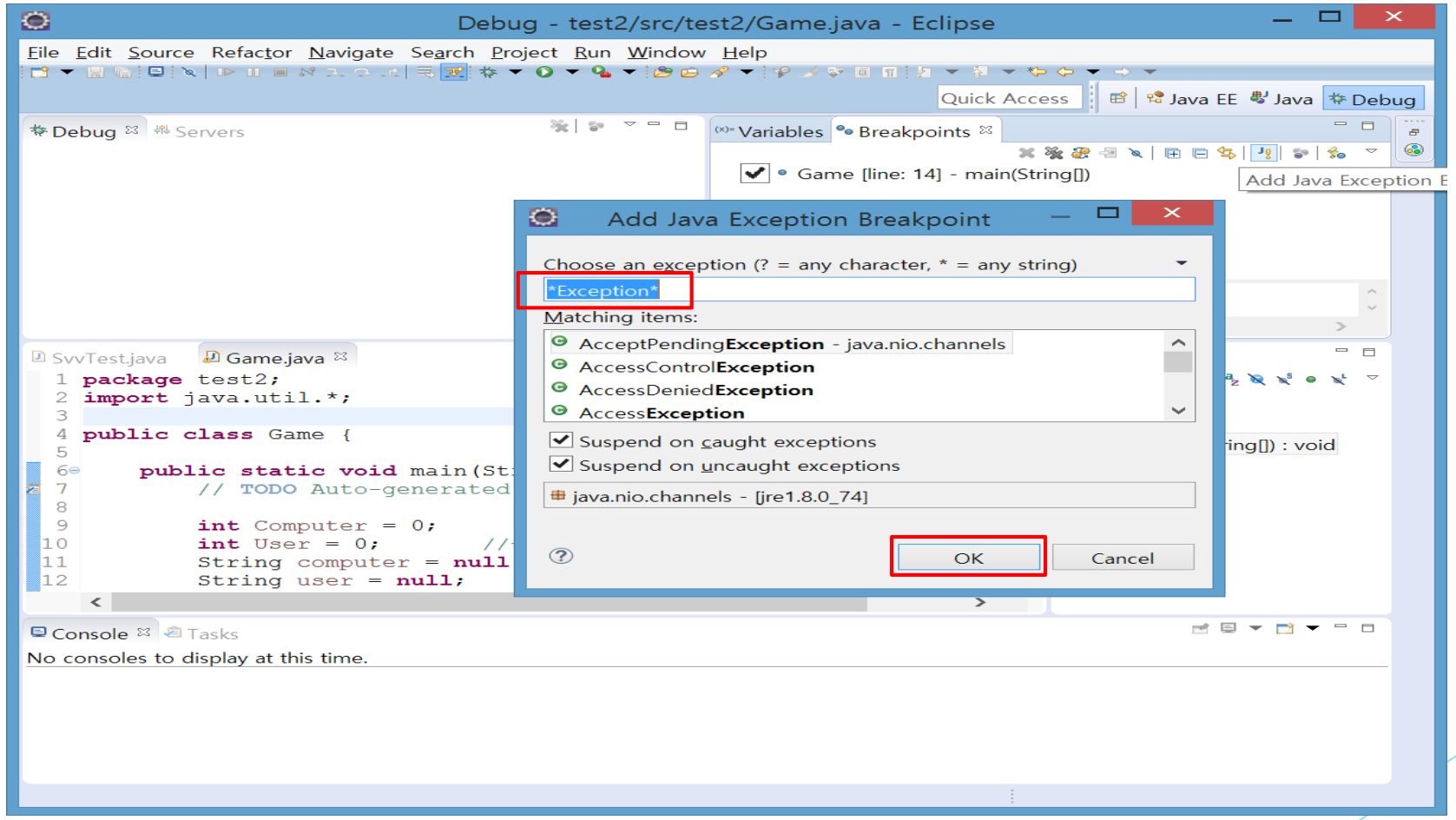


1. Skip All Breakpoints
모든 브레이크 포인트 건너뜀
2. Resume(F8)
다음 브레이크 포인트까지 진행함
3. Suspend
쓰레드를 일시 정지하며 현재 수행문에 지정한 것과 같음
4. Terminate
쓰레드 종료
5. Disconnet

6. Step Into(F5)
한 단계 진행하는데 다음 라인이 함수 안이면 함수 안으로 들어감
7. Step Over(F6)
함수 호출을 지나치고 현재 위치에서 한 단계 진행
8. Step Return(F7)
현재 함수 끝까지 바로 가서 리턴한 후 함수 호출부로 되돌아감
9. Drop to Frame
선택한 스택 프레임의 첫 행으로 이동. 처음부터 다시 하고자 할때
10. Use Step Filters(Shift+F5)
스템 필터링

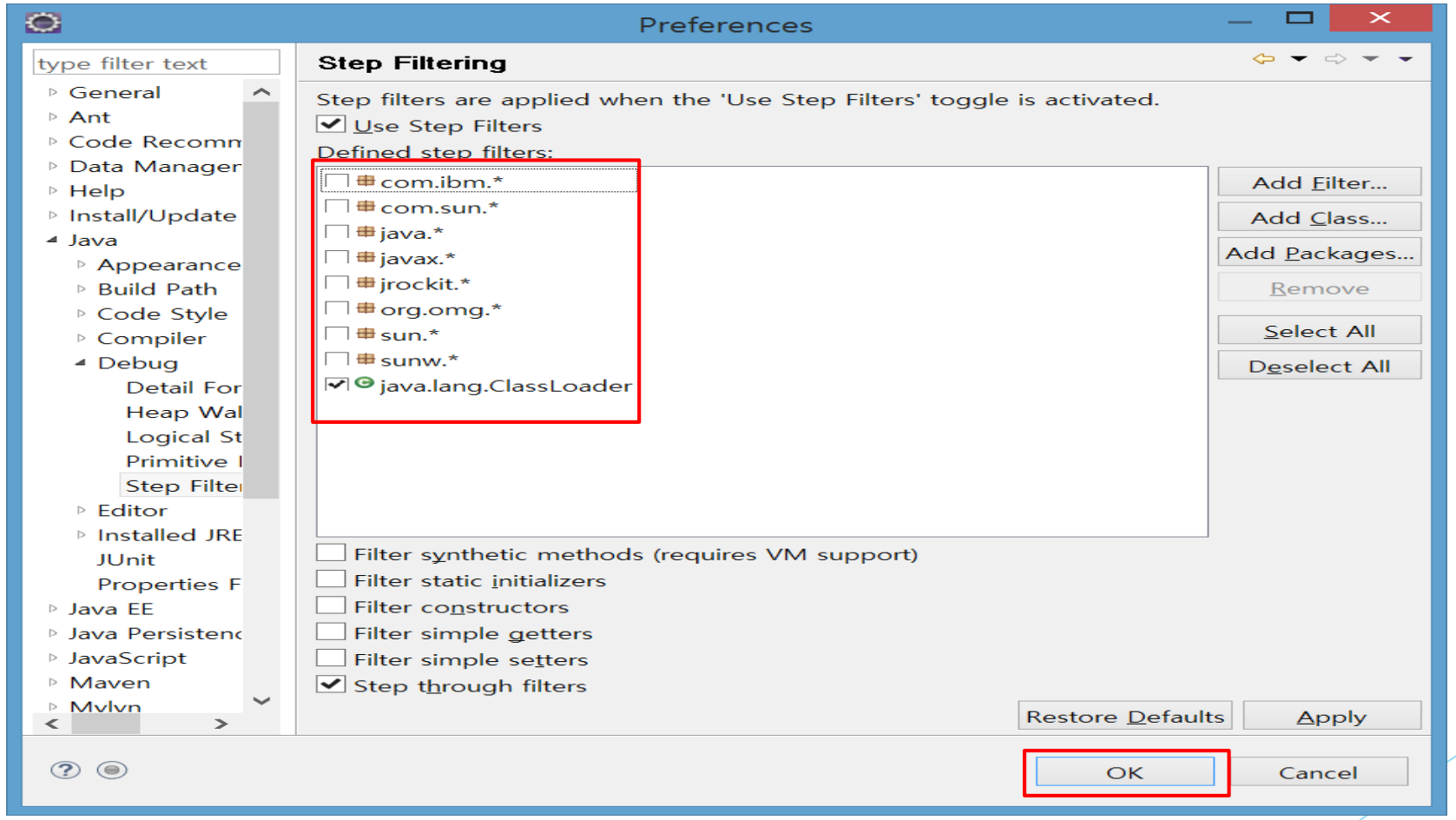
Eclipse 사용하기
Breakpoint & Debug

예외 Breakpoint를 지정해서 특정 예외가 발생할 때만 Breakpoint가 동작하게 할 수 있다.



Eclipse 사용하기
Breakpoint & Debug

지정한 외부 Library나 자바 Core 클래스에 대해서는 Step Into로 해도 그 내부로 들어가지 않아 디버깅에 편리하다



JUnit

Unit Test

소개
유닛 테스트

- 소스 코드의 테스트 대상 모듈이 의도된 대로 정확히 작동하는지를 검사
  - 모든 모듈에 대한 테스트 케이스를 작성한다.
- 장점
  - 단위 유닛이 정확하게 동작하는지 검증 가능
    - 작은 단위로 나누어 테스트하여 어느 부분에서 에러가 발생하였는지를 확인 가능하다.
  - 변경의 용이함
    - 코드 변경 후에도 모듈이 의도대로 작동하고 있는 것을 증명 가능하다.
  - 모듈 통합의 용이함
  - 문서화
  - ...
- 그러나 모든 상황에 대한 테스트 케이스를 작성할 수는 없기 때문에, 유닛 테스트가 완벽함을 보장하지는 않는다!

목차

- Junit 소개
- JUnit 사용
- JUnit 활용

JUnit
소개

- 자바 기반 테스트를 위한 프레임워크로, 단위 모듈의 구현도를 테스트한다.
- 주요기능 : Test case 생성 및 실행, 오류추적
- 라이선스 : Common Public License 1.0 / 무료
- 특징
  - 메서드와 같은 단위 모듈 별 테스팅을 가능케 함으로써 코드 품질을 보장한다.
  - 정확한 단위 테스팅을 가능케 함으로써 통합 테스팅 시 회귀결함을 줄인다.
  - 다른 모듈에 의존하지 않고, 원하는 모듈만 임의의 순서대로 수행할 수 있다.
- http://www.junit.org

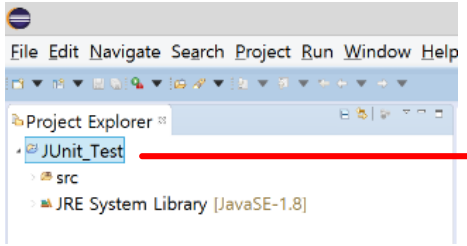
JUnit
소개

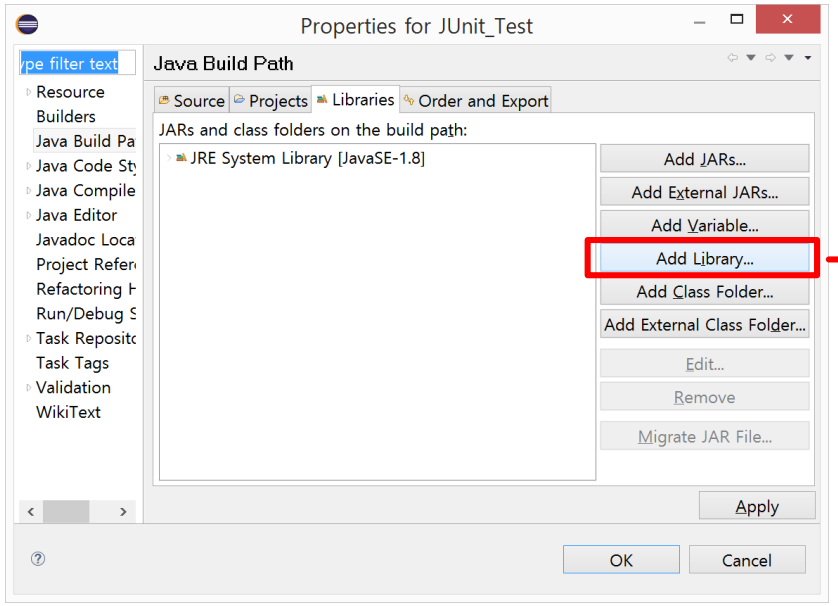
- 장점
  - 테스트 케이스를 추가적인 클래스에서 작성하여 관리에 용이하며, 추가적으로 문서화의 효과를 얻을 수 있다.
  - 테스트 결과로 메서드의 수행 시간을 표시하여 성능의 확인이 용이하다.
- 사용
  - 이클립스에 통합되어 있기 때문에 추가적인 설치가 필요 없다.
  - Java 프로젝트에 Junit 라이브러리를 추가하는 것으로 사용 가능하다.

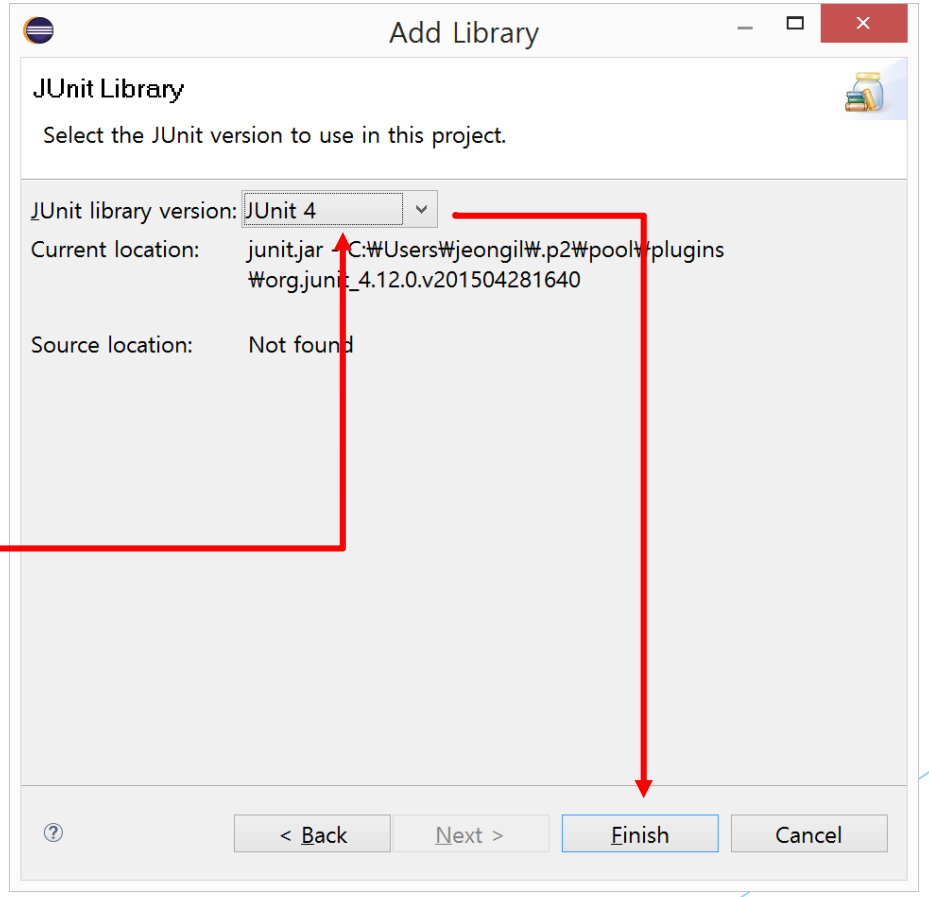
JUnit 사용

프로젝트 생성과 설정

1) 프로젝트 생성 후 설정 → Junit 라이브러리 추가







JUnit 사용

테스트 대상 클래스 작성

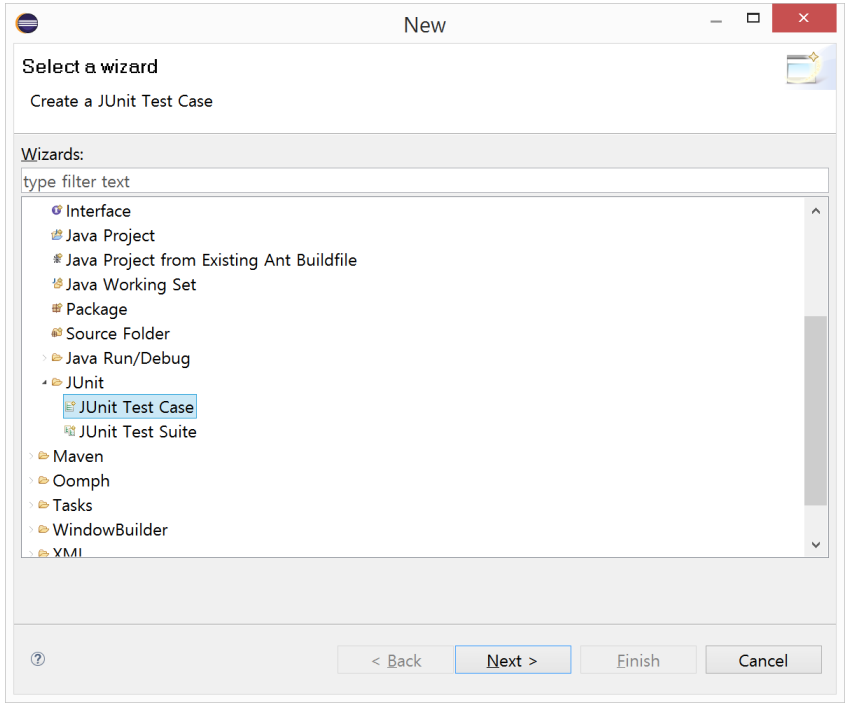
Cal 클래스의 sum 메서드는 두 개의 정수를 입력 받아 두 정수의 합을 출력한다.

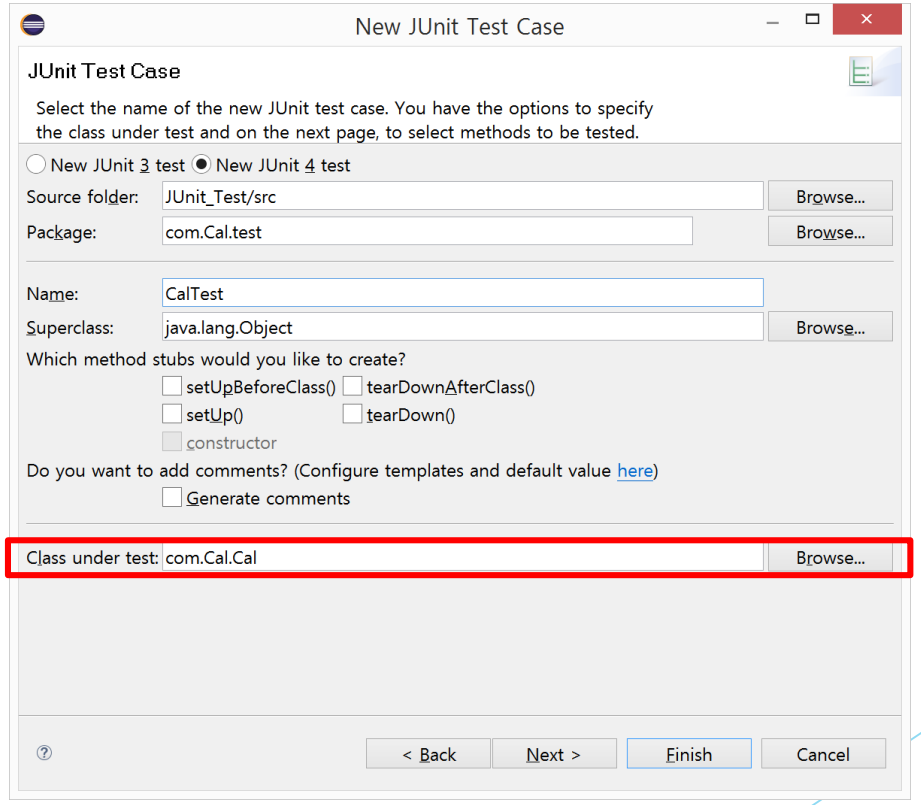
```java
package com.Cal;

public class Cal {
    public int sum(int in1, int in2)
    {
        return in1 + in2;
    }
}
```

JUnit 사용

테스트 클래스 생성

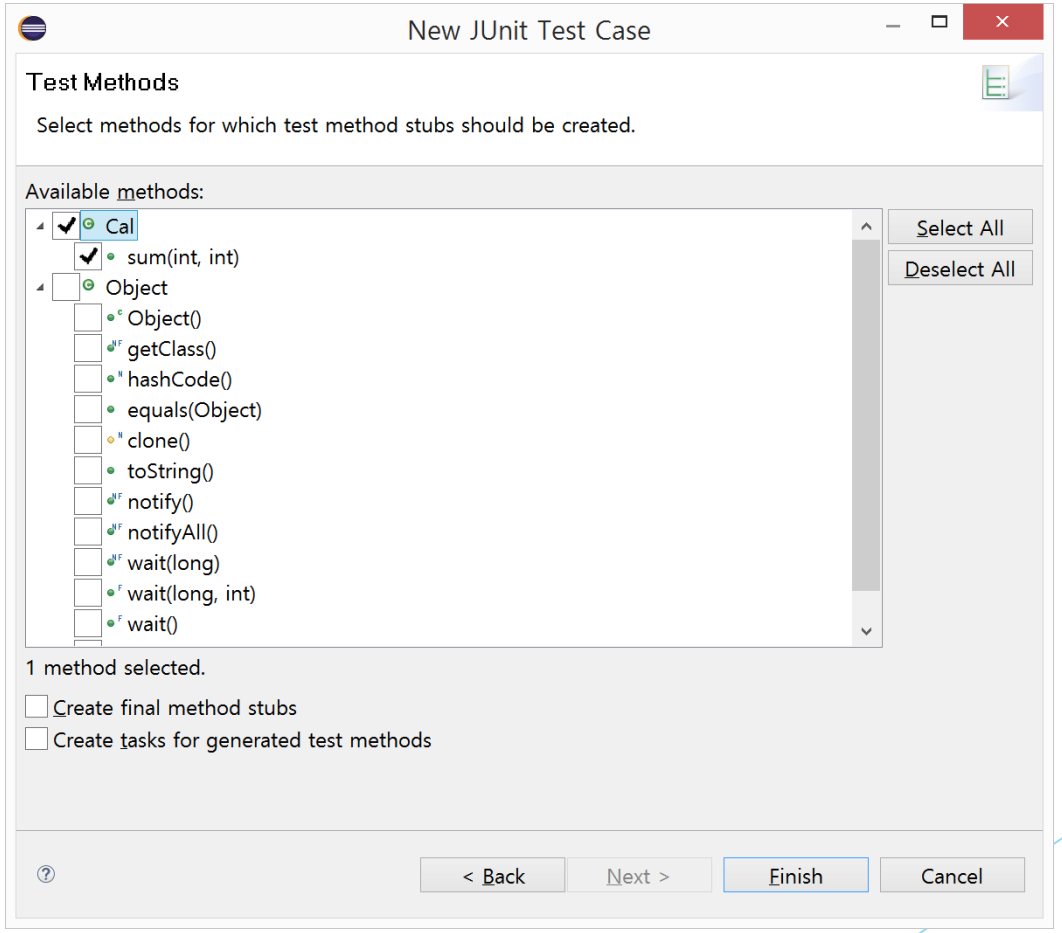
2) New → Junit Test Case
3) 테스트 할 대상 클래스를 선택한다.





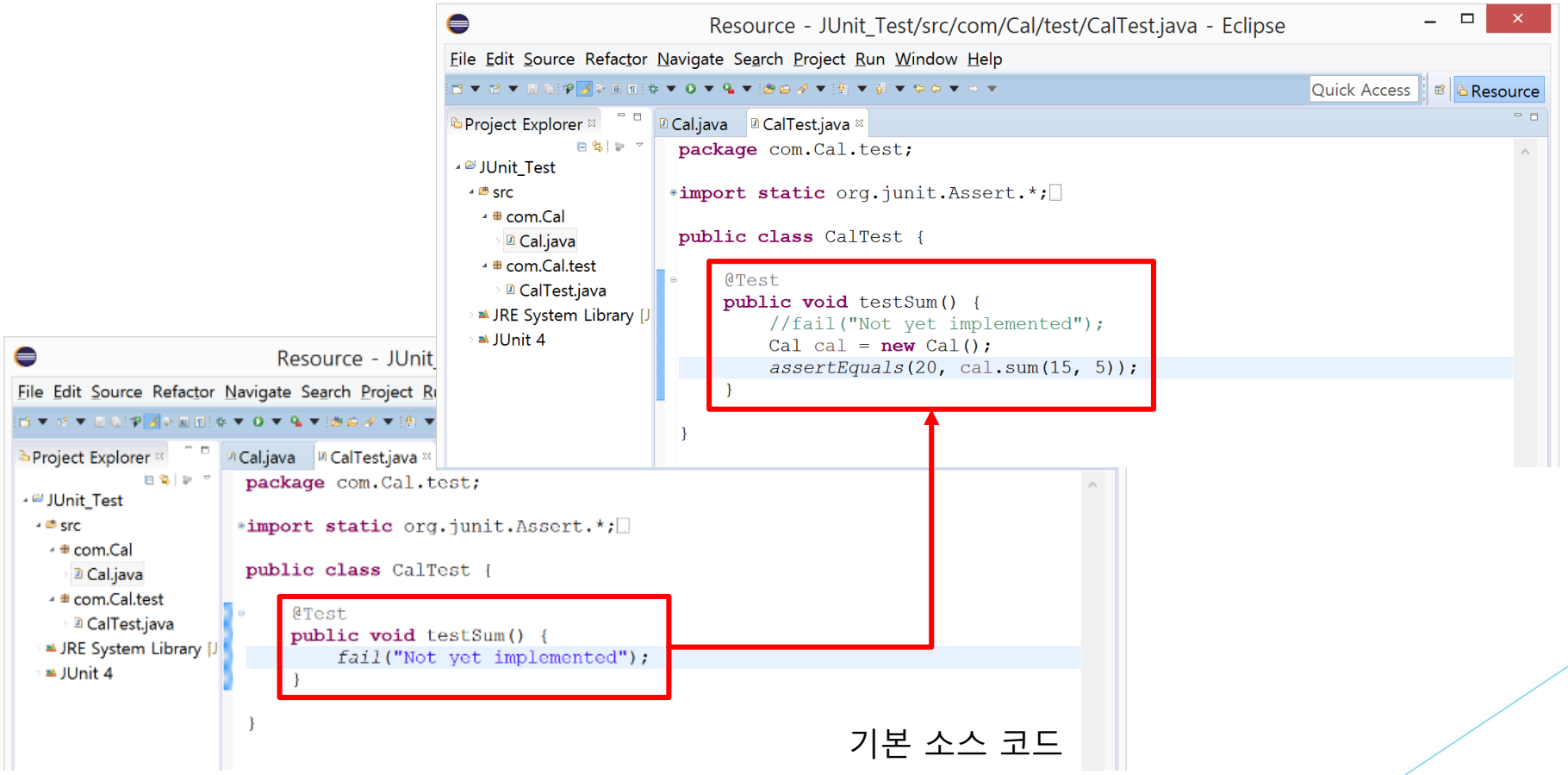
JUnit 사용
테스트 대상 설정

4) 테스트 대상 클래스의 요소를 선택한다.



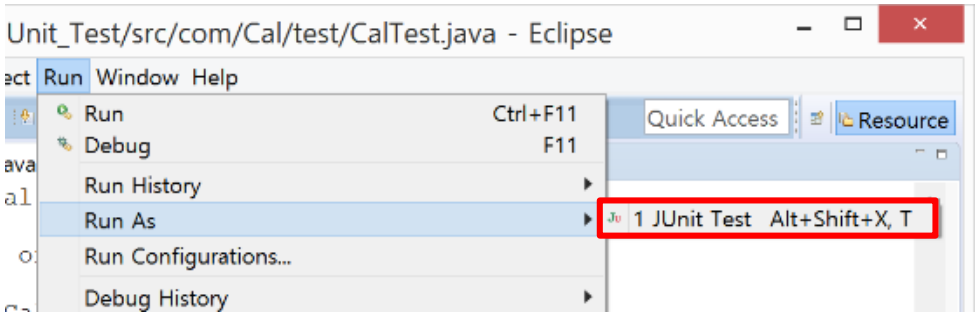
JUnit 사용
테스트 구문 작성

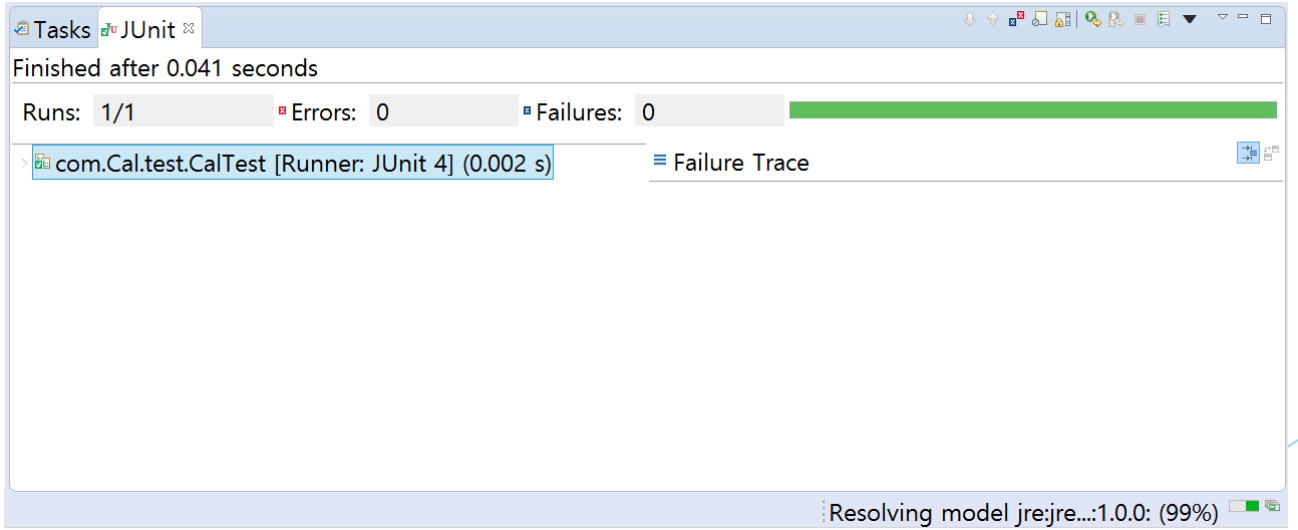
5) 기본 테스트 코드를 적절한 형태로 변형시킨다.



JUnit 사용
테스트 실행과 결과

6) JUnit Test를 선택하여 테스트를 수행





JUnit 활용
무엇을 검사하는 것이 좋은가?

- 값 분석
  - 결과값이 존재하는가?
  - 결과값이 기대한 값과 일치하는가?
  - 적절한 범위의 값이 도출되는가?
  - 충분한 수의 값이 나오는가?
- 에러발생
  - 예상되는 에러는 무엇이며, 실제로 발생할 수 있는가?
- 성능
  - 예정된 시간 내에 수행 가능한가?

JUnit 활용
테스트 메서드

- 가장 많이 사용하는 메서드는 org.junit.Assert 클래스가 지원하는 assert 메서드들이다.
  - import static org.junit.Assert.*;

- assert 메서드들은 객체의 값과 상태를 비교하여 테스트의 성공/실패 여부를 결정한다.

JUnit 활용
테스트 메서드

http://junit.sourceforge.net/javadoc/org/junit/Assert.html

메서드	입력	성공 조건
assertEquals	double, long, String, Object…	입력한 두 값이 동일함.
assertArrayEquals	byte[], char[], int[], long[], Object[]	입력한 두 배열이 동일함.
assertNull	Object	입력한 객체가 Null임.
assertNotNull	Object	입력한 객체가 Null이 아님.
assertSame	Object	두 객체가 동일한 객체임.
assertNotSame	Object	두 객체가 같지 않음.
assertTrue	boolean	입력한 값이 참임.
assertFalse	boolean	입력한 값이 거짓임.
fail	(String)	무조건 실패함.

JUnit 활용
어노테이션

- JDK 5.0 부터 지원하기 시작한 문법으로, @ 기호를 사용하여 메타 데이터를 기술한다.
- JUnit에서는 @Test를 비롯한 다양한 어노테이션을 지원하여 테스트를 효율적으로 수행할 수 있도록 지원한다.

```
public class CalTest {

    @Test
    public void testSum() {
        //fail("Not yet implemented");
        Cal cal = new Cal();
        assertEquals(20, cal.sum(15, 5));
    }

}
```

JUnit 활용
어노테이션

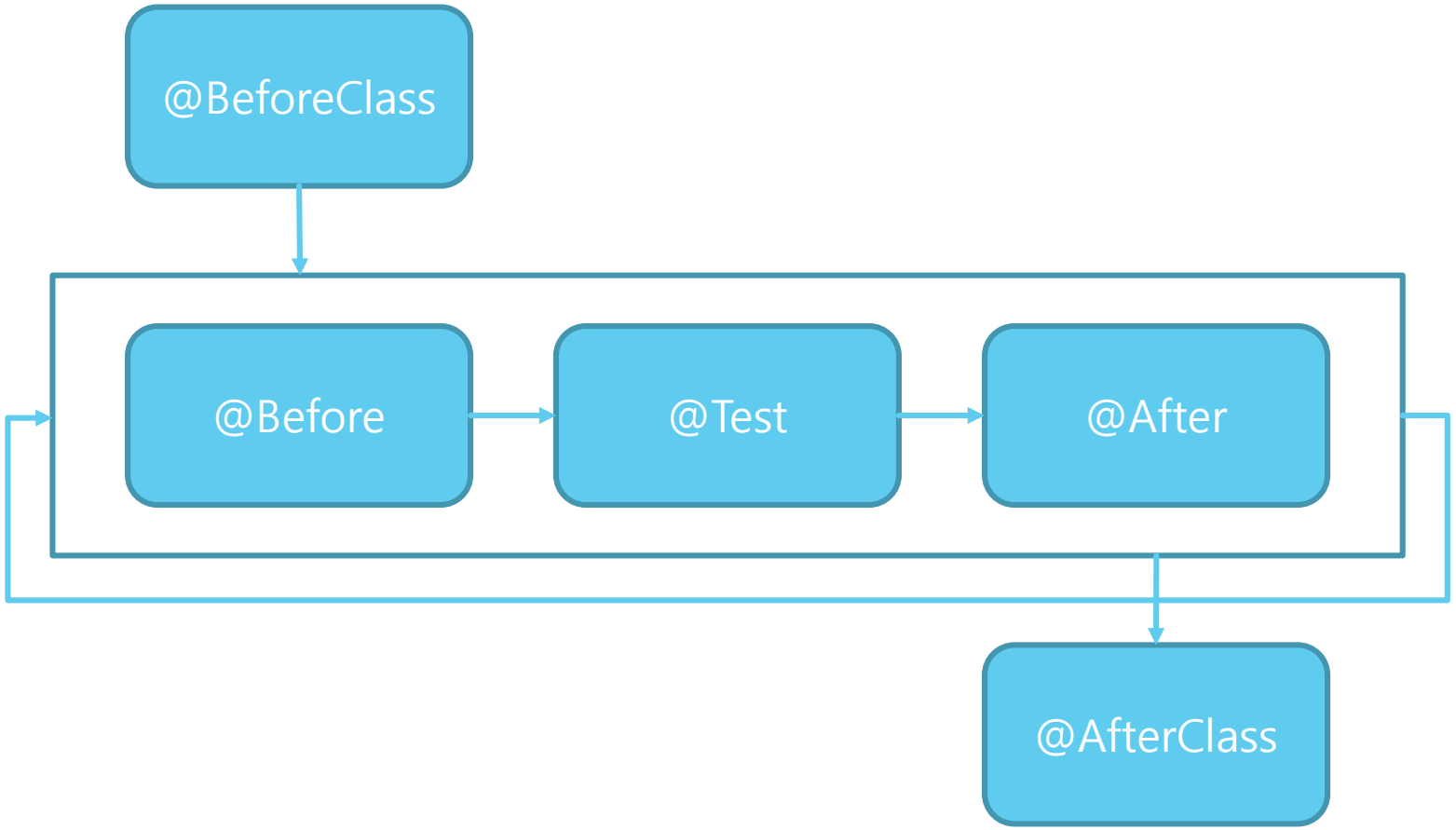
이름	의미
@Test	테스트를 수행하는 메서드임을 표시
@Test(timeout=n)	테스트 대상 메서드의 수행 시간이 n ms 이상일 경우 fail
@Test(expected=Exception.class)	테스트 대상 메서드가 주어진 예외를 Throw 하지 않을 경우 fail

JUnit 활용
어노테이션

이름	의미
@BeforeClass	모든 테스트의 시작 부분에 단 한 번 시작되는 메서드
@AfterClass	모든 테스트의 종료 이후에 단 한 번 실행되는 메서드
@Before	각 테스트의 시작마다 실행되는 메서드
@After	각 테스트의 종료마다 실행되는 메서드
@Ignore	해당 테스트 메서드를 무시함

JUnit 활용

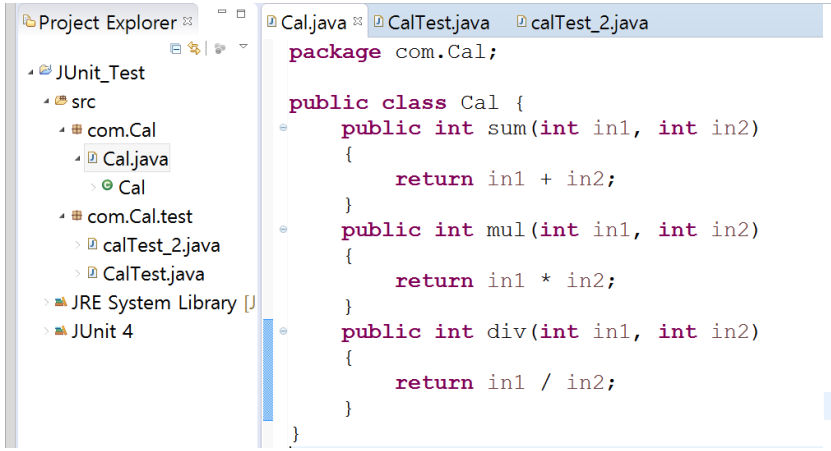
어노테이션에 의한 테스트 수행 순서



JUnit 활용

코드 확장

1) 두 수의 곱을 반환하는 mul, 나눈 값을 반환하는 div 메서드를 생성한다.
2) 변경된 대상 클래스를 반영한 새로운 테스트 코드를 생성한다.



Cal.java

```java
package com.Cal;

public class Cal {
    public int sum(int in1, int in2)
    {
        return in1 + in2;
    }
    public int mul(int in1, int in2)
    {
        return in1 * in2;
    }
    public int div(int in1, int in2)
    {
        return in1 / in2;
    }
}
```

CalTest.java / calTest_2.java

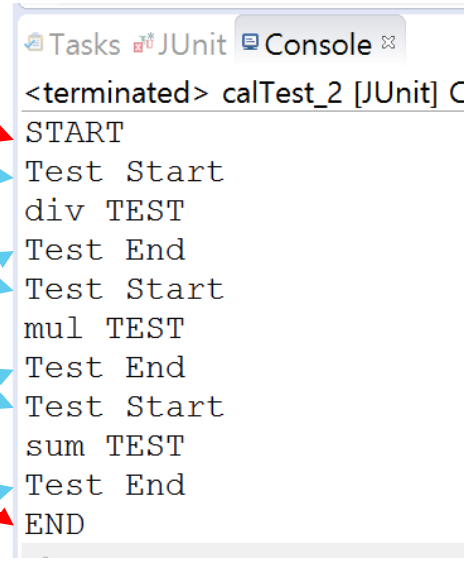
```java
@BeforeClass
public static void startClass() {
    System.out.println("START");
}
@AfterClass
public static void endClass() {
    System.out.println("END");
}
@Before
public void start(){
    System.out.println("Test Start");
}
@Test
public void testSum() {
    Cal cal = new Cal();
    System.out.println("sum TEST");
    assertEquals(15+5, cal.sum(15, 5));
}
@Test
public void testMul() {
    System.out.println("mul TEST");
    Cal cal = new Cal();
    assertEquals(10*15, cal.mul(10, 15));
}
@Test
public void testDiv() {
    System.out.println("div TEST");
    Cal cal = new Cal();
    assertEquals(5.0/2.0, cal.div(5, 2), 0);
}
@After
public void end(){
    System.out.println("Test End");
```

JUnit 활용

테스트 코드 수행 순서

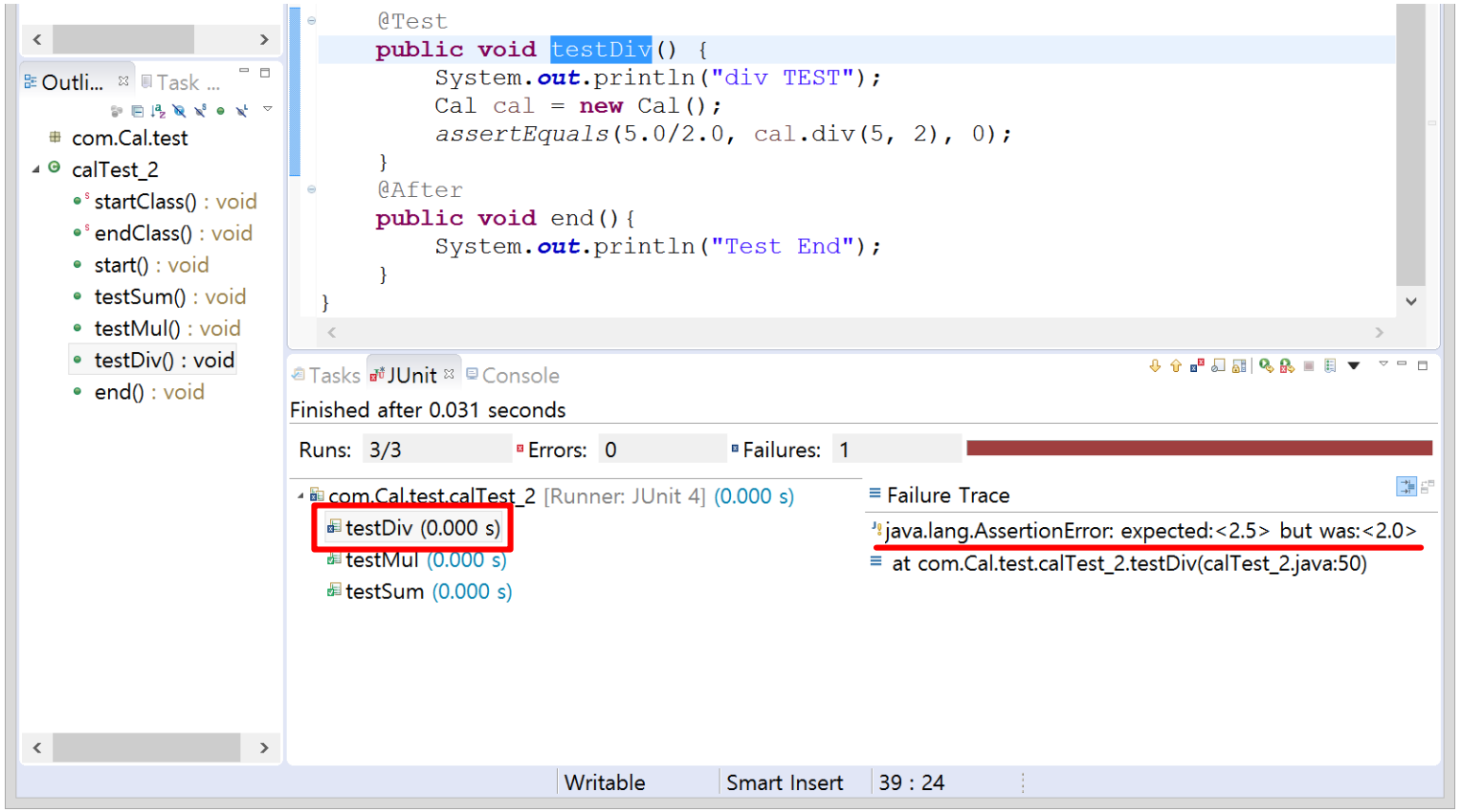
3) 테스트 수행 시 콘솔 출력을 통해 메서드의 수행 순서를 확인할 수 있다.

```
@BeforeClass
public static void startClass() {
    System.out.println("START");
}
@AfterClass
public static void endClass() {
    System.out.println("END");
}
@Before
public void start(){
    System.out.println("Test Start");
}
@Test
public void testSum() {
    Cal cal = new Cal();
    System.out.println("sum TEST");
    assertEquals(15+5, cal.sum(15, 5));
}
@Test
public void testMul() {
    System.out.println("mul TEST");
    Cal cal = new Cal();
    assertEquals(10*15, cal.mul(10, 15));
}
@Test
public void testDiv() {
    System.out.println("div TEST");
    Cal cal = new Cal();
    assertEquals(5.0/2.0, cal.div(5, 2), 0);
}
@After
public void end(){
    System.out.println("Test End");
```



JUnit 활용
테스트 결과와 분석

4) 테스트가 실패한 경우, 결과 화면에 실패한 코드와 원인이 출력된다.



JUnit 활용

코드 수정

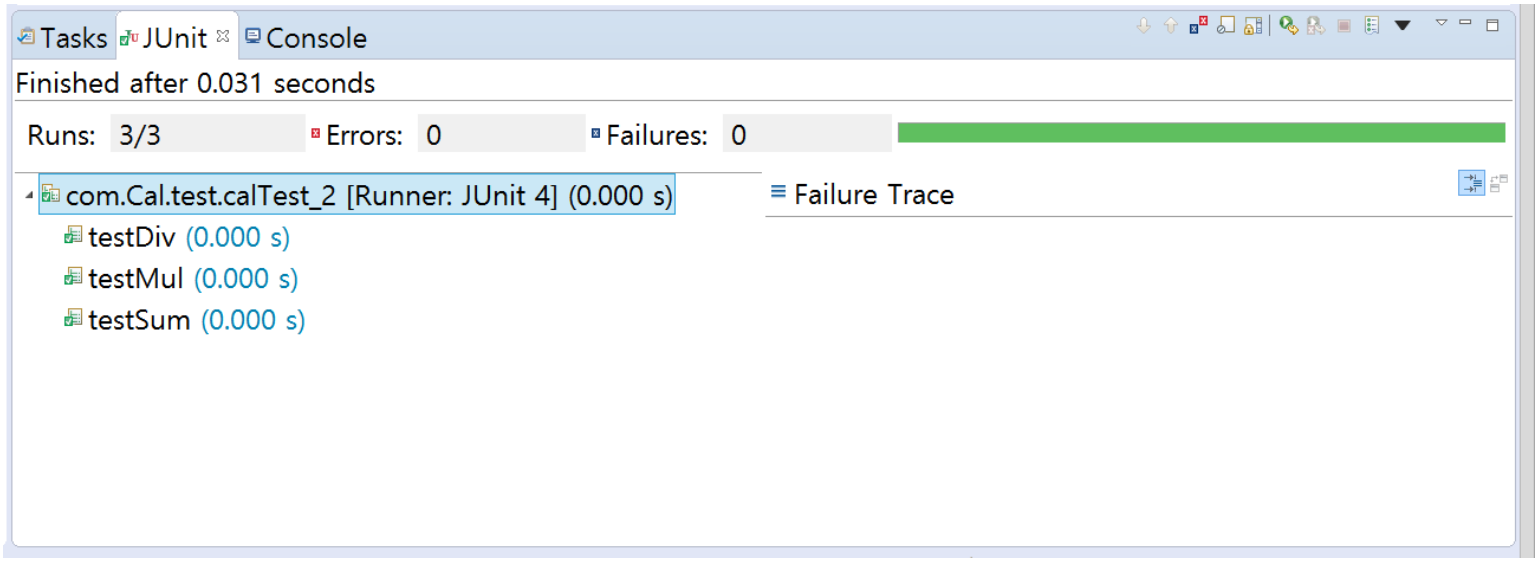
5) 실패 원인을 분석하여 코드를 수정한다.

```
public int div(int in1, int in2)
{
    return in1 / in2;
}
```

```
package com.Cal;

public class Cal {
    public int sum(int in1, int in2)
    {
        return in1 + in2;
    }
    public int mul(int in1, int in2)
    {
        return in1 * in2;
    }
    public float div(float in1, float in2)
    {
        return in1 / in2;
    }
}
```

JUnit 활용
수정 후 결과

6) 테스트가 정상적으로 수행되었다.



ANT

Build

소개

- 자바 프로그래밍 언어에서 사용하는 자동화된 소프트웨어 빌드 도구
  유닉스나 리눅스에서 사용되는 make와 비슷하나 자바언어로 구현되어 있어 자바 실행환경이 필요하며 자바 프로젝 트들을 빌드하는데 표준으로 사용
- 주요기능 : 패키지 빌드 자동화
- 라이센스 : BSD License (Berkeley Software Distribution) / 무료
- 특징
  - Build 자동화(컴파일, Javadoc생성, 실행, FTP SCP, SFTP연결, CVS연동, 다른 공학도구와의 연동)
  - 배포
  - 유닛 테스트 (JUnit 활용, HTML등 테스트 결과 보고서 작성)
- http://ant.apache.org/

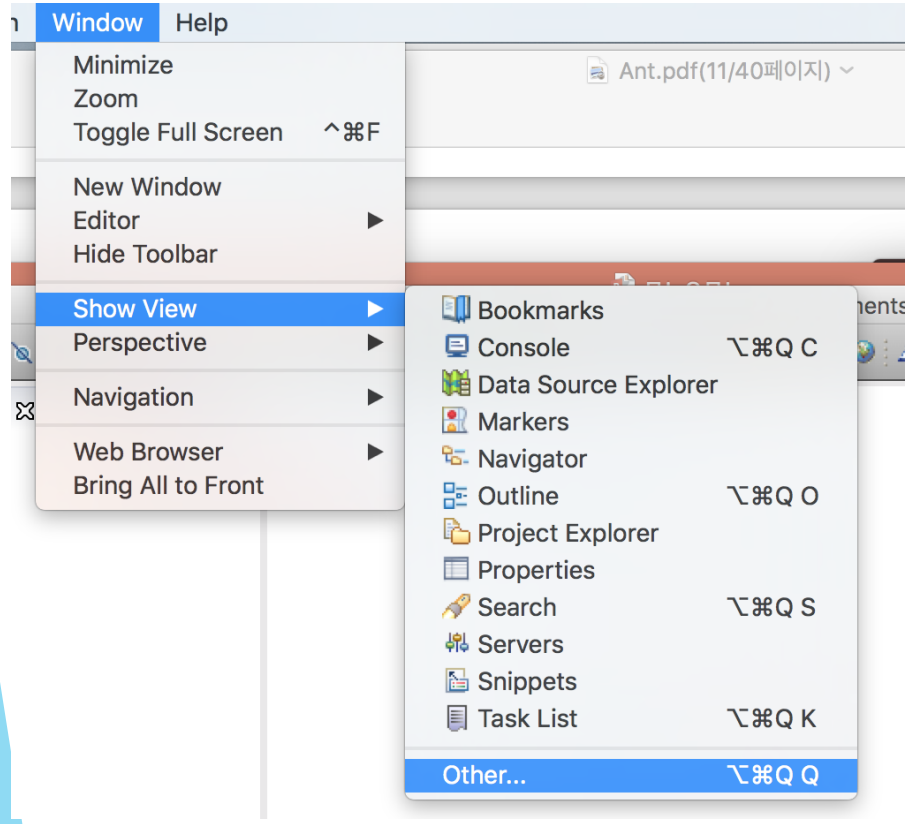
목차

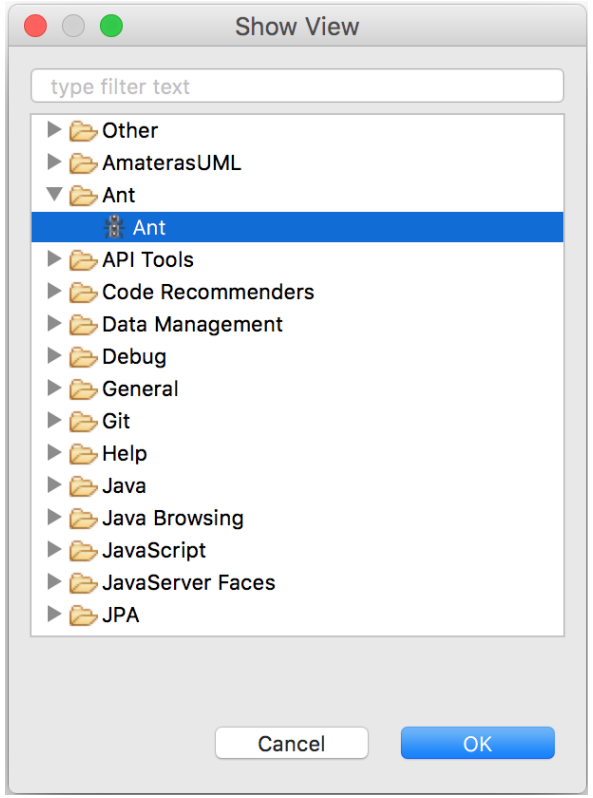
- ANT 설치 확인
- ANT 설치
- ANT 사용

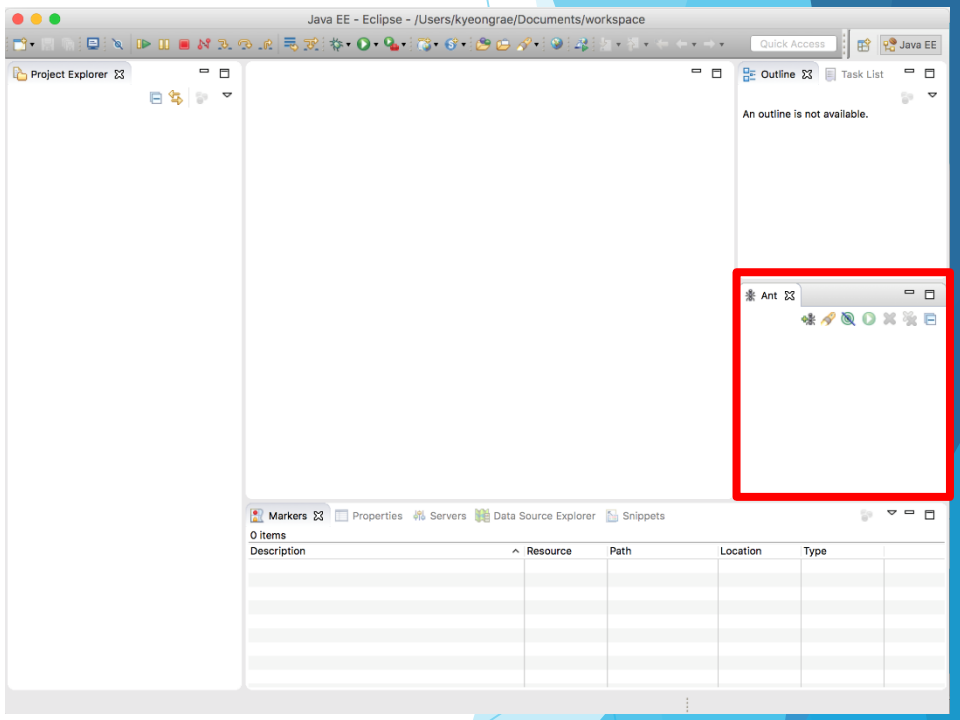
ANT 설치 확인

Eclipse Mars.2 (4.5.2) 에는 기본 설치되어 있습니다.

만약 아래 화면에서 ANT를 찾을 수 없다면, 이클립스를 재설치 하거나, ANT를 설치해주세요.



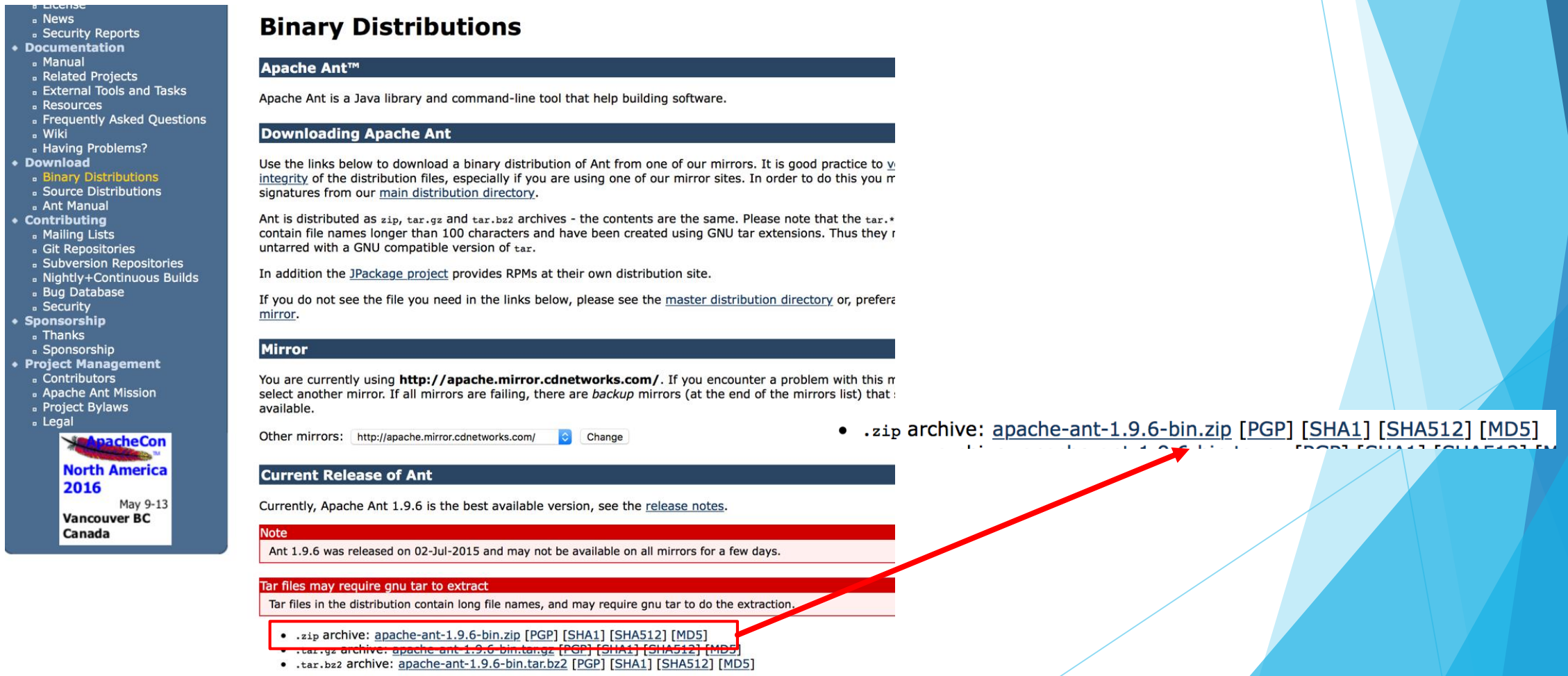




ANT 설치

수동 설치

http://ant.apache.org/bindownload.cgi

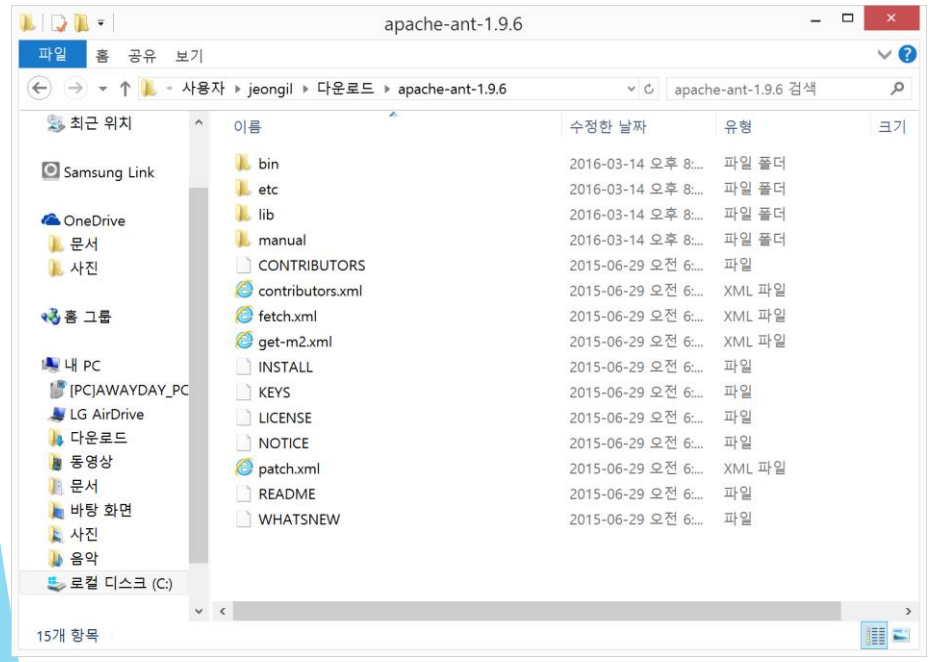


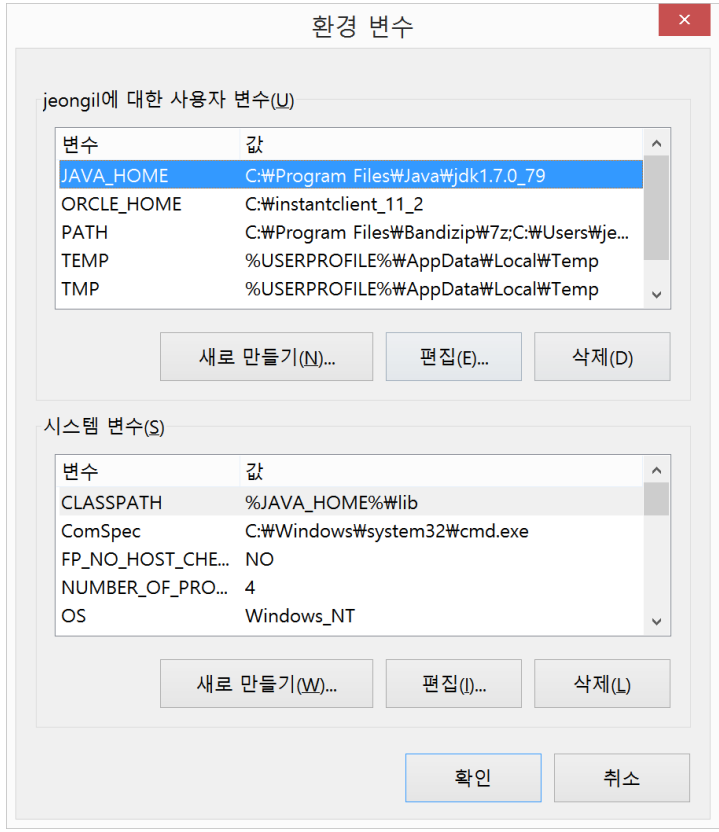
- .zip archive: apache-ant-1.9.6-bin.zip [PGP] [SHA1] [SHA512] [MD5]

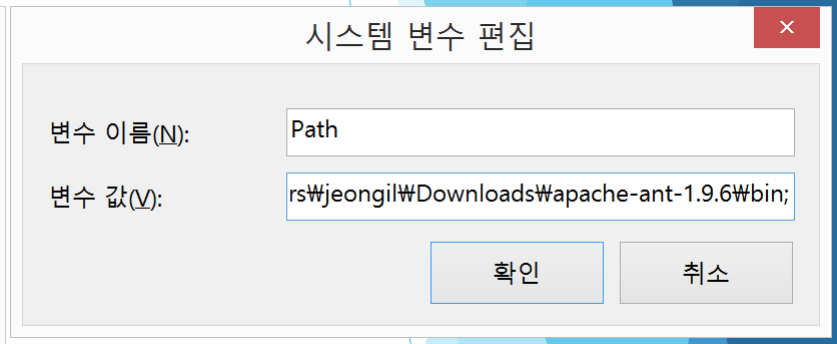
ANT 설치

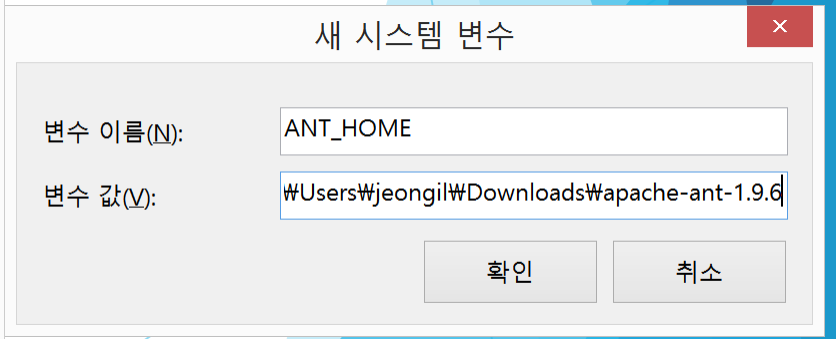
환경 변수 설정

1) 다운받은 ANT의 압축을 해제하고 환경변수를 등록해줍니다.
2) 이클립스를 실행 후 앞에서 설명한 ANT 설치확인을 진행하면 됩니다.



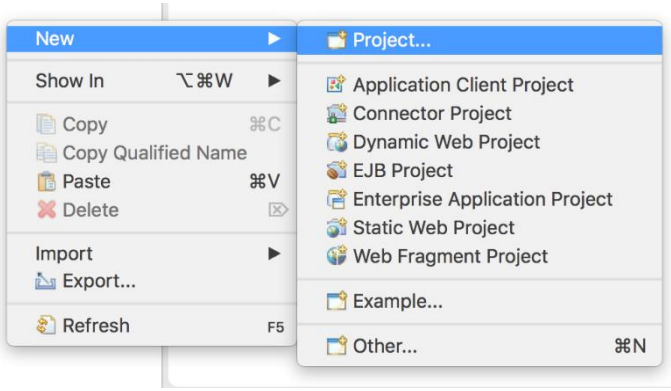


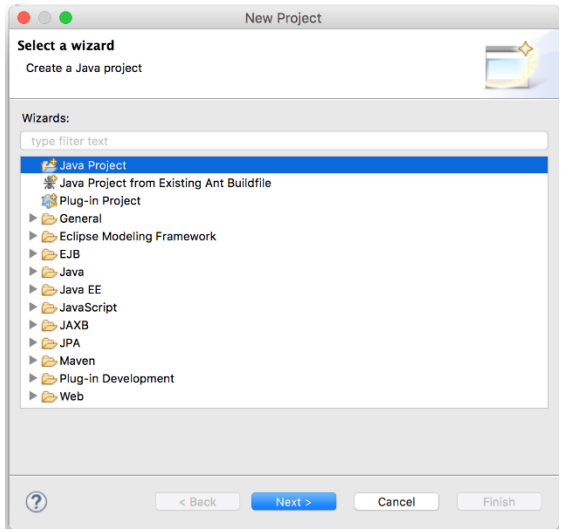


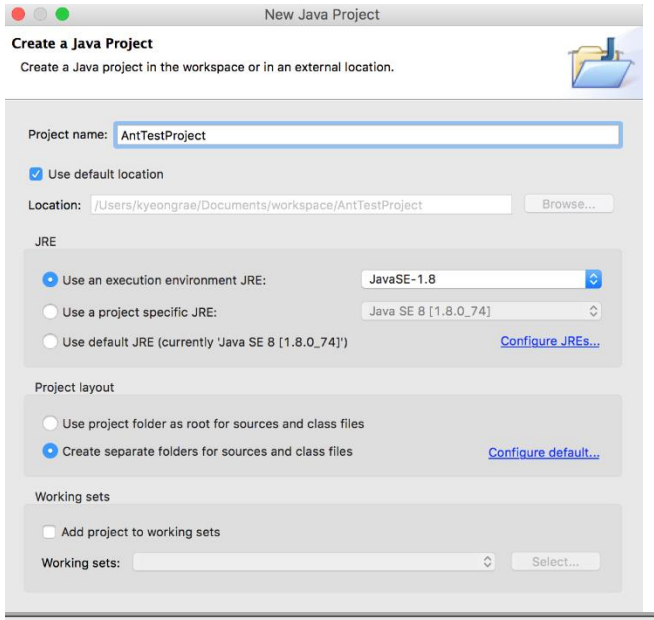


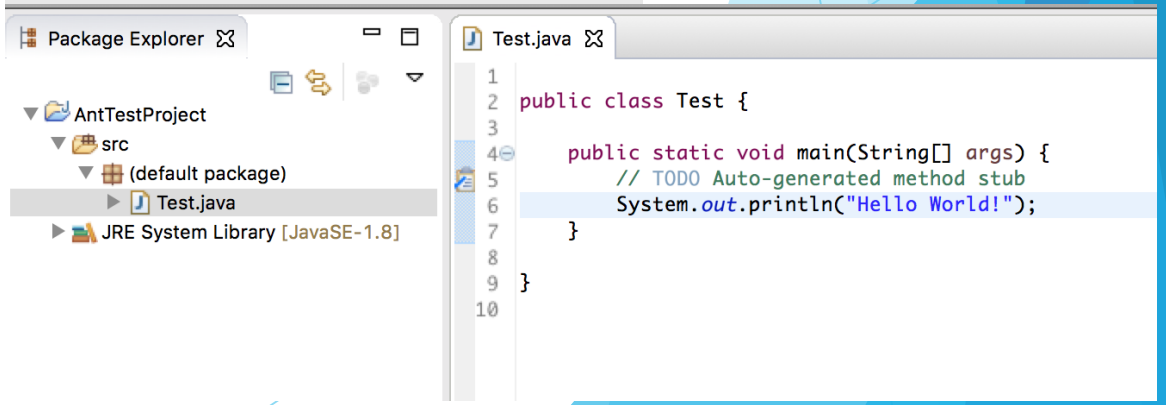
ANT 사용

테스트 프로젝트 생성



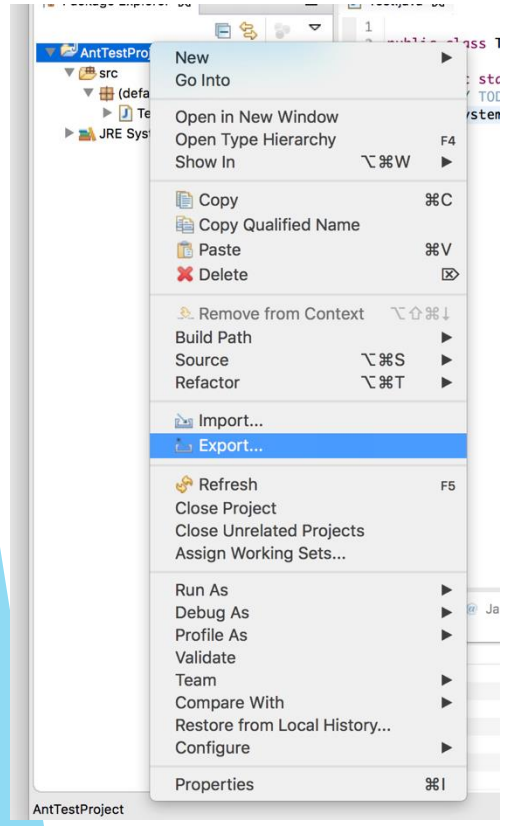


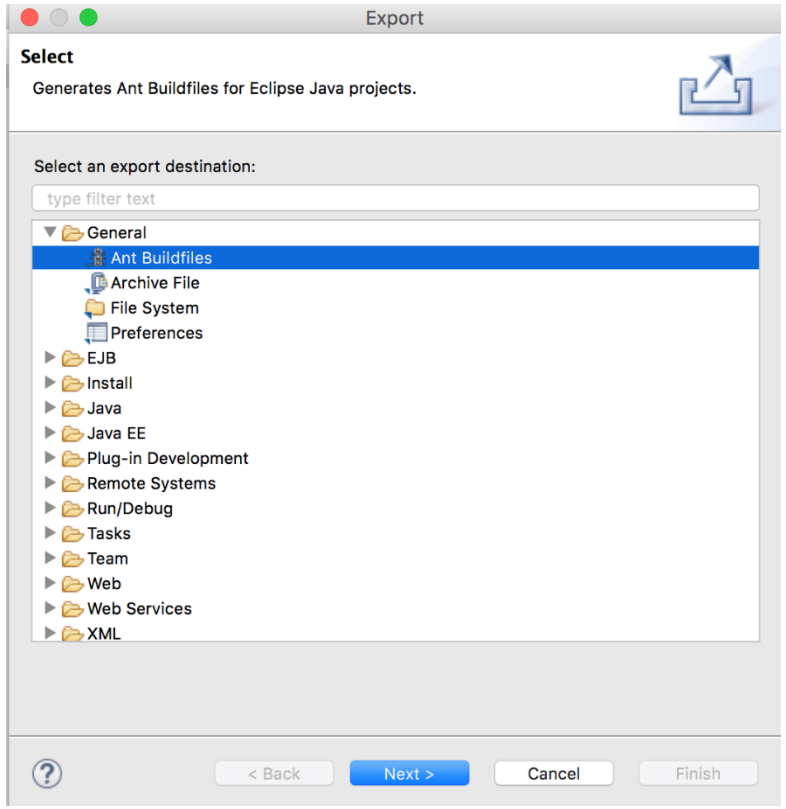


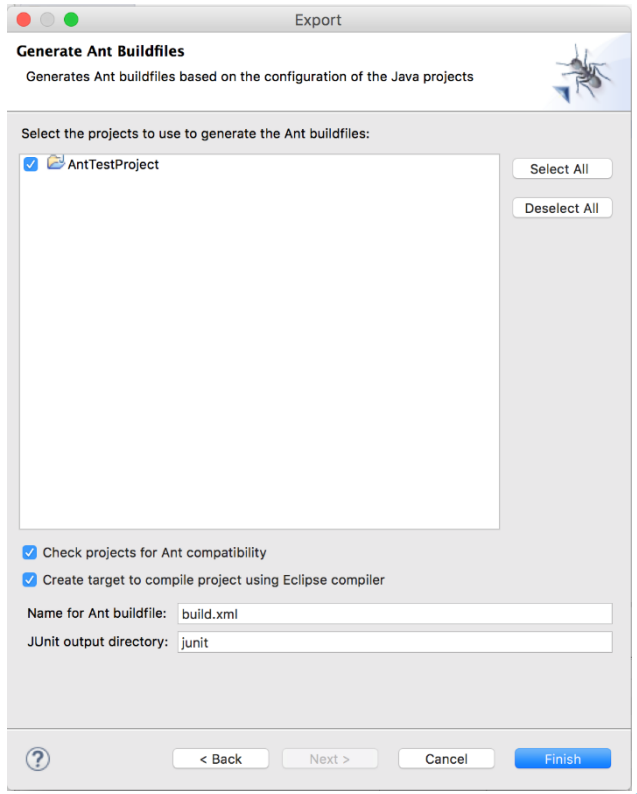


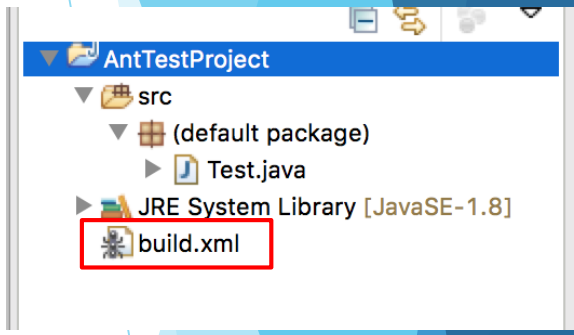
Package explorer 에서 마우스 우클릭을 하여
새로운 JAVA 프로젝트를 생성합니다.
프로젝트 명은 무관합니다.
생성한 프로젝트에서 "Hello World" 를 프린트합니다.

ANT 사용
EXPORT









Package Explorer 에서 project에 마우스 우클릭하여 export를 선택하고 Ant Buildfiles 를 선택해 줍니다. 옵션을 default 옵션입니다.
Finish 하면, 프로젝트 내에 build.xml이 생겨납니다.

ANT 사용

Ant의 Build.xml파일 : Project요소, Target요소, Task요소를 포함

- 타깃 간의 의존 관계는 target 요소의 depends 속성으로 기술
  - 그림예시 : dist 타깃은 compile 타깃에 의존
    - compile → dist → clean 순으로 동작

```
1  <project name="project_name" default="all" basedir=".">
2      <target name="all" depends="compile, dist, clean" >
3
4      </target>
5
6      <target name="compile">
7
8      </target>
9
10     <target name="dist" depends="compile">
11
12     </target>
13
14     <target name="clean">
15
16     </target>
17 </project>
```

- 일반적인 Ant의 명령어
  - property : 속성을 지정
  - mkdir : 새로운 디렉토리 생성
  - copy : 파일, 디렉토리 복사
  - javac : 컴파일
  - jar:jar파일생성
  - javadoc : javadoc 생성
  - delete : 파일, 디렉토리 삭제
  - Java : Java 프로그램을 실행
  - Junit : 테스트 프레임 워크 JUnit 을 사용하여 Java 프로그램을 테스트
  - Jnitreport : junit 작업 출력 결과 파일을 사용하여 HTML 형식 등에 대응하는 보고서를 생성 – ftp:FTP연결을시작하고파일업로드,다운로드등을가능하게함
  - zip:지정된디렉토리와파일을ZIP형식으로압축·보관
  - echo:콘솔(명령라인환경)에문자열출력
  - splash:런타임에지정된시간동안시작표시
  - buildnumber : 빌드 번호를 업데이트

ANT 사용

Ant의 Build.xml파일 : Project요소, Target요소, Task요소를 포함

- property : 속성을 지정
  - 속성에서는 대소문자를 구별
    > foo.src라는 이름의 속성으로 "src"를 지정 시

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

> 빌드 스크립트의 다른 부분에서 이 속성을 참조하려면 "${foo.src}"와 같이 기록

```
<javac srcdir="${foo.src}" destdir="${build}" />
```

> 파일을 읽어 속성을 설정

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

- mkdir : 새로운 디렉토리 생성
  - 디렉토리 생성 예시

```
<mkdir dir="${dist}" />
<mkdir dir="${dist}/lib" />
```

ANT 사용

Ant의 Build.xml파일 : Project요소, Target요소, Task요소를 포함

- copy : 파일, 디렉토리 복사
  - 파일 하나 복사 예시

```
<copy file="myfile.txt" tofile="mycopy.txt" / >
```

  - 디렉토리에서 다른 디렉토리 복사 예시

```
<copy todir="../new/dir “>
    <fileset dir="src_dir “>
</copy>
```

  - 특정 디렉토리의 원하는 파일만 지정하여 복사 : *.java 파일을 제외한 나머지 파일 복사 예시

  > exclude는 특정 내용을 제외하는 키워드이고 **의 경우 src_dir 디렉토리 아래 모든 디렉토리를 재귀적으로 탐색

```
<copy todir="../dest/dir “>
    <fileset dir="src_dir “>
        <exclude name="**/*.java" />
    </fileset>
</copy>
```

ANT 사용

Ant의 Build.xml파일 : Project요소, Target요소, Task요소를 포함

- javac : JAVA 소스파일의 컴파일
  - 재귀적으로 탐색하여, class파일이 없거나 class 파일이 java 파일보다 오래된 경우에만 컴파일
  - ${src}와 그 하위 디렉토리에 있는 모든 java 파일을 컴파일 예시
    > 결과를 ${build} 디렉토리에 저장, 클래스패스에는 xyz.jar가 포함되고, 디버깅 옵션을 켜고 컴파일

```
<javac srcdir="${src}" destdir="${build}" classpath="xyz.jar" debug="on" />
```

  - ${src}와 ${src2} 및 그 하위 디렉토리에 있는 java파일을 컴파일 : 결과를 ${build} 디렉토리에 저장
    > mypackage/p1과 mypackage/p2에 있는 파일만을 사용하고, 클래스패스에 xyz.jar가 포함
    > 여기서 include가 특정 내용을 포함하는 키워드 입니다

```
<javac srcdir="${src}:${src2}" destdir="${build}
include="mypackage/p1/**,mypackage/p2/** "
exclude="mypackage/p1/testpackage/**" classpath="xyz.jar' debug="on" />
```

ANT 사용

Ant의 Build.xml파일 : Project요소, Target요소, Task요소를 포함

- jar : 지정된 파일들을 jar로 묶습니다.
  - ${build}/classes 밑에 있는 파일을 app.jar로 묶기

```
<jar destfile="${dist}/lib/app.jar" basedir="${build}/classes" />
```

  - mypackage/text 밑에 있는 파일만을 묶고 Test.class는 제외

```
<jar destfile="${dist}/lib/app.jar" basedir="{build}/classes"
 include="mypackage/test/**" exclude="**/Test.class" />
```

  - ${build}/classes와 ${src}/resources 밑에 있는 파일을 app.jar로 묶되, Test.class는 제외

```
<jar destfile="${dist}/lib/app.jar “>
    <fileset dir="${build}/classes" exclude="**/Test.class" />
    <fileset dir="${src}/resources" />
</jar>
```

- javadoc : javadoc 문서를 생성,
  - src 디렉토리 밑에 있는 소스파일을 읽어 javadoc 문서를 생성하여 ${doc} 디렉토리에 저장

```
<javadoc destdir="${doc} “>
    <fileset dir="${src} “>
    </fileset>
</javadoc>
```

ANT 사용

Ant의 Build.xml파일 : Project요소, Target요소, Task요소를 포함

- delete : 하나의 파일 또는 디렉토리와 그 하위 디렉토리, fileset에 지정된 파일을 삭제
  - /lib/ant.jar 파일을 삭제

```
<delete file="/lib/ant.jar" />
```

  - lib 디렉토리와 그 하위 디렉토리를 삭제

```
<delete dir="/lib" />
```

  - 현재 디렉토리와 그 하위 디렉토리에서 확장자가 bak인 모든 파일을 삭제

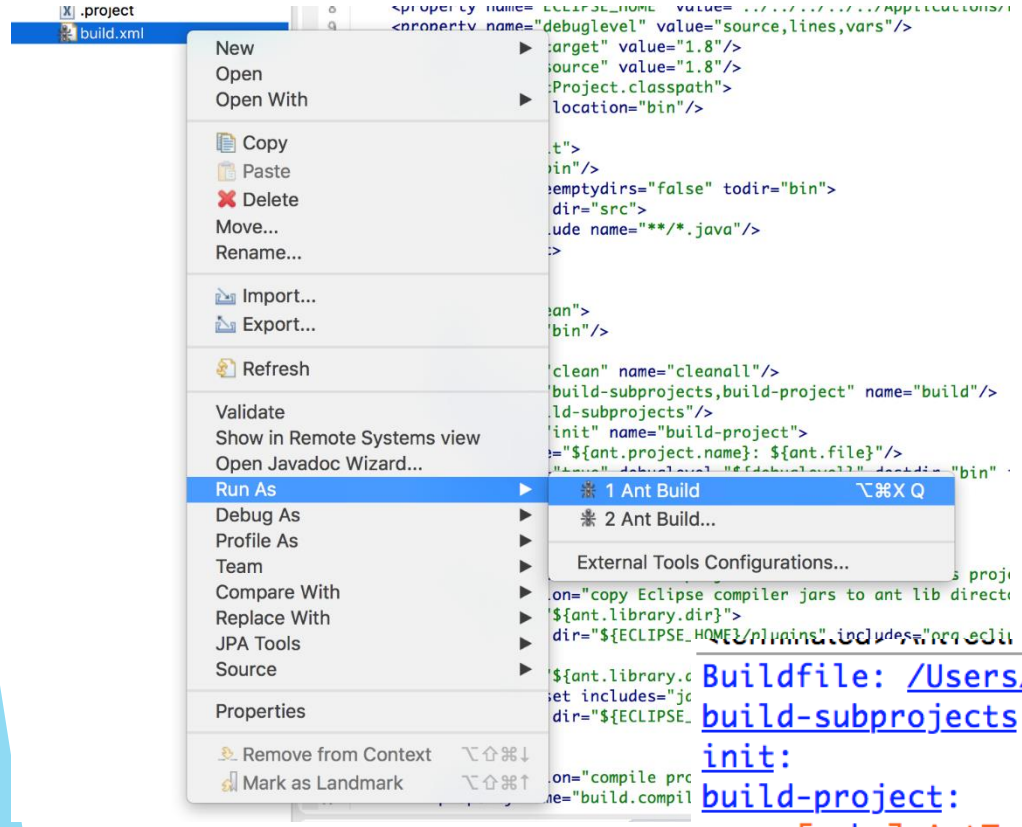
```
<delete>
    <fileset dir="." include="**/*.bak" />
</delete>
```

  - build 디렉토리와 그 하위 디렉토리를 삭제합니다. includeEmptyDirs를 "true"로 설정하면 fileset을 사용할 때 빈 디렉토리도 포함

```
<delete includeEmptyDirs="true “>
    <fileset dir="build" />
</delete>
```

ANT 사용

build.xml 에서 설정한대로 빌드해보기



Project explorer에서 build.xml에 마우스 우클릭
Run As에서 Ant Build를 선택
빌드 완료 후 Console 창에 메세지가 출력.

```
Buildfile: /Users/kyeongrae/Documents/workspace/AntTestProject/build.xml
build-subprojects:
init:
build-project:
     [echo] AntTestProject: /Users/kyeongrae/Documents/workspace/AntTestProject/build.xml
build:
BUILD SUCCESSFUL
Total time: 388 milliseconds
```