Software Verification

#1 Junit, Eclipse 및 빌드 환경

Software Verification Team 4

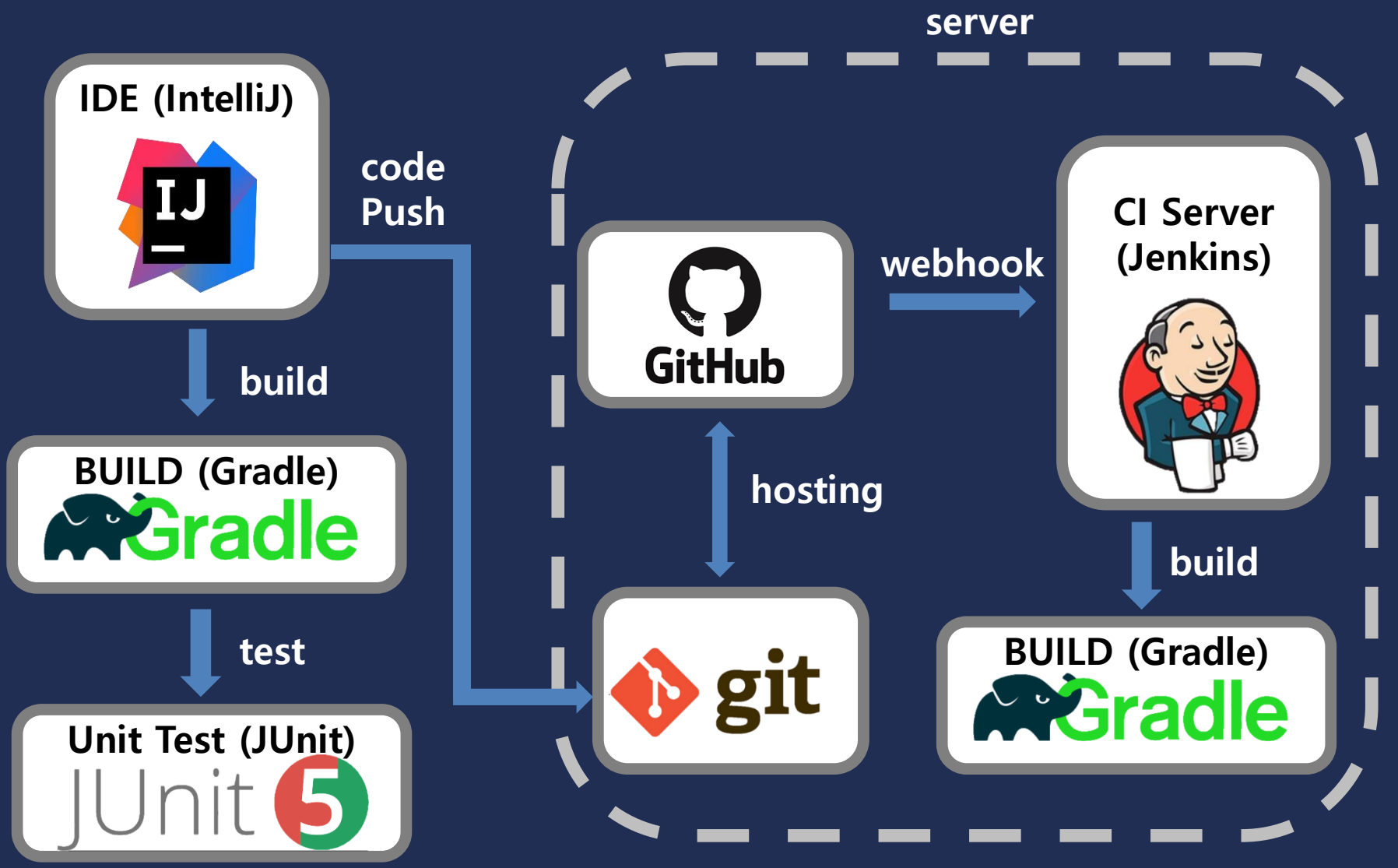
강 정 모
송 상 연
신 승 화

Software Verification

#1 Junit, Edipse 및 빌드 환경

CONTENTS

01 Overall Structure



02 IDE – IntelliJ (IntelliJ vs Eclipse)

IntelliJ VS Eclipse

장점

- 구문분석을 통한 코드 자동 완성기능
- Eclipse와 비교해 상대적으로 가벼움
- 안정적 Project 단위 관리

단점

- 유료(Ultimate 버전)

단점

- IntelliJ 보다 무겁다
- Workspace 단위 관리

장점

- 일반적인 자동완성
- 무료

02 IDE – IntelliJ (why IntelliJ?)

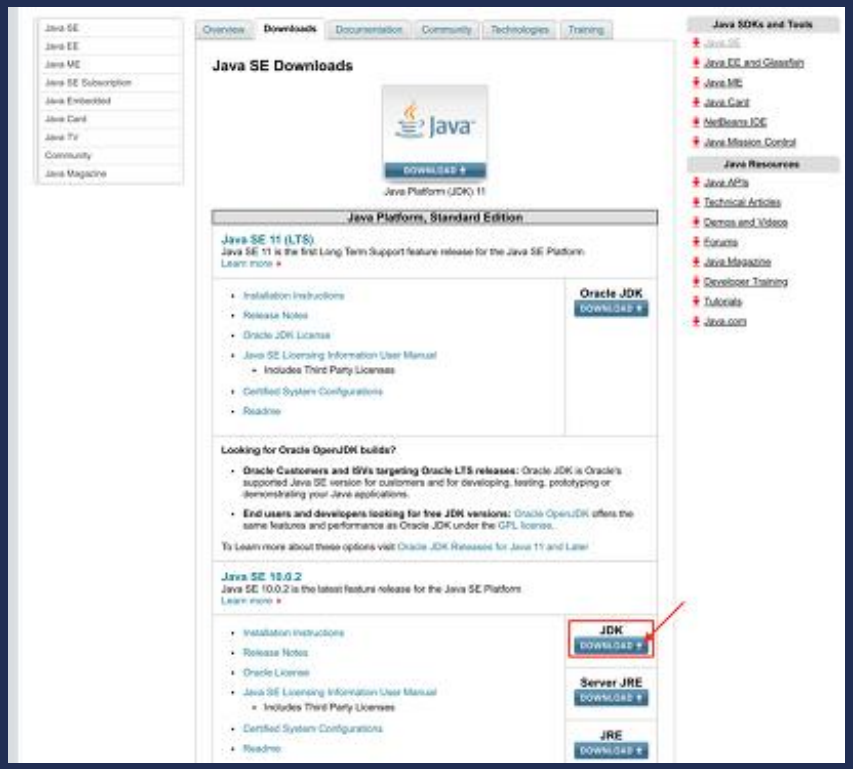
- 상당한 IDE의 안정성
- 다양한 프로그래밍 언어를 지원하는 많은 플러그인 제공
- 디자인 & 개발이 쉽게 진행 가능
- 프로그램이 가벼움
- 디버깅을 위한 설정이 간단함
- 코드의 자동 완성 기능이 뛰어남 (지능적임)

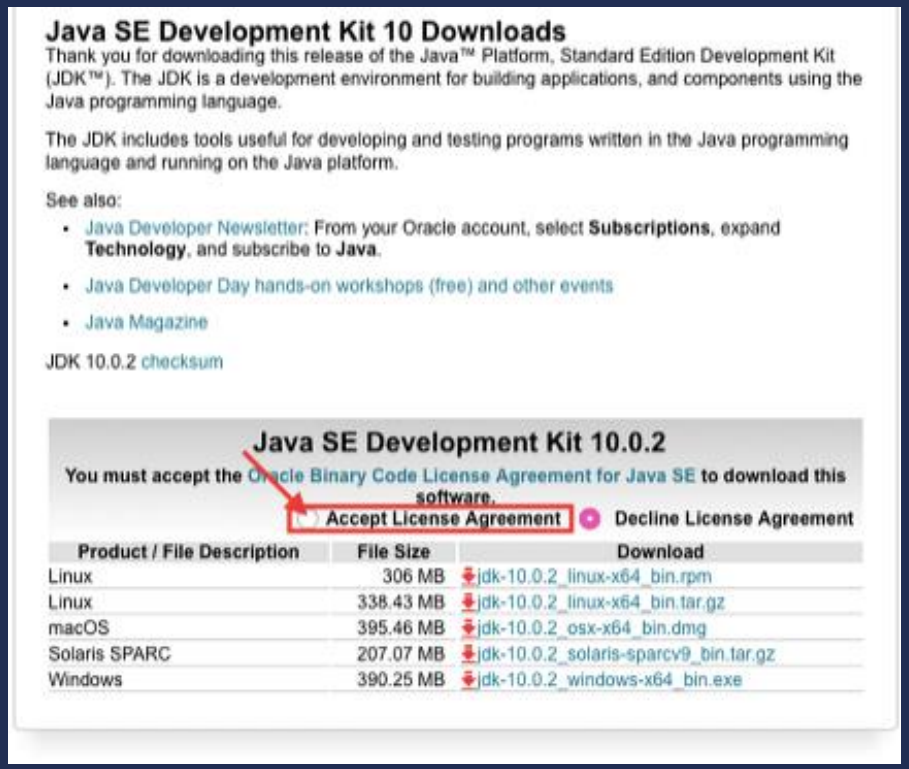
02 IDE – IntelliJ (JDK - Installation)

JDK 설치 방법

02 IDE – IntelliJ (JDK - Installation)

ORACLE 접속 → JDK 다운로드 → DOWNLOAD → Accept License Agreement 선택





02 IDE – IntelliJ (JDK - Installation)

ORACLE 접속 → JDK 다운로드 → DOWNLOAD → Accept License Agreement 선택
→ 운영체제에 맞게 설치 패키지 클릭

※ Gradle & JUnit 5를 위해서는 JDK 8버전 이상 사용해야함 (8버전 사용)

02 IDE – IntelliJ (JDK - Installation)

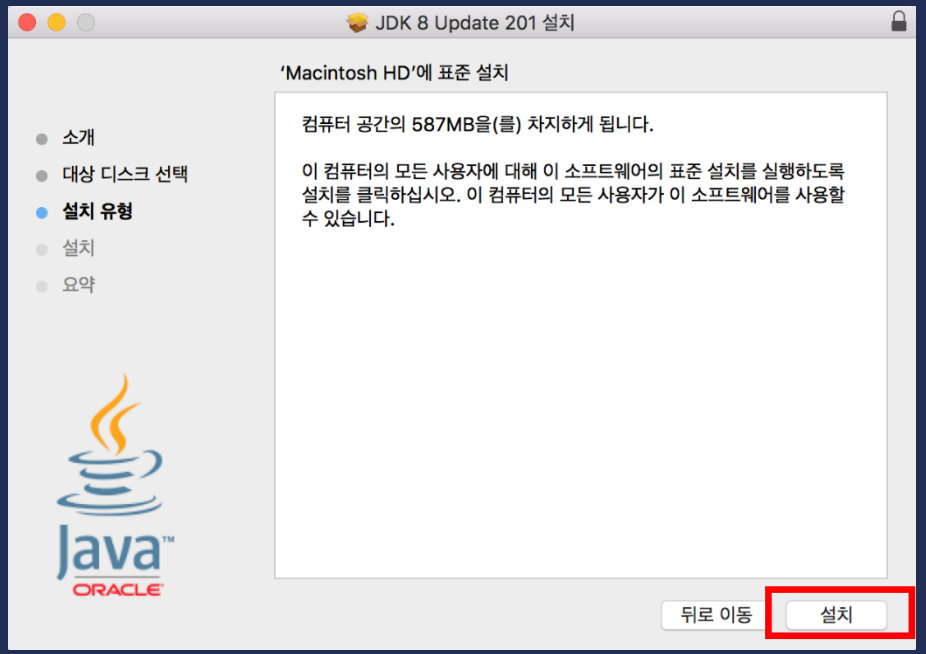
ORACLE 접속 → JDK 다운로드 → DOWNLOAD → Accept License Agreement 선택 → 운영체제에 맞게 설치 패키지 클릭 → 다운로드 된 설치 패키지 클릭

02 IDE – IntelliJ (JDK - Installation)

→ 상자 클릭

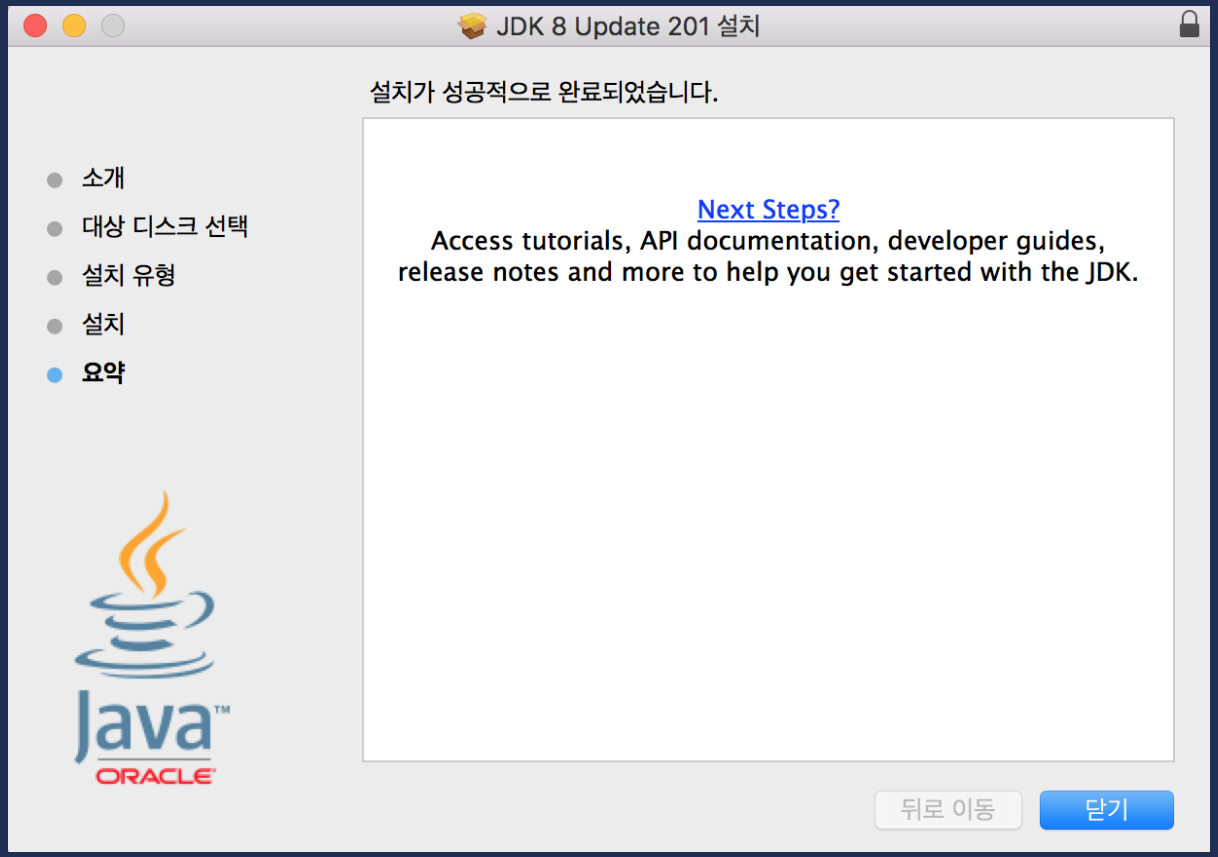
02 IDE – IntelliJ (JDK - Installation)

→ 상자 클릭 → 계속 클릭 → 설치 클릭



02 IDE – IntelliJ (JDK - Installation)

→ 상자 클릭 → 계속 클릭 → 설치 클릭 → 설치 완료

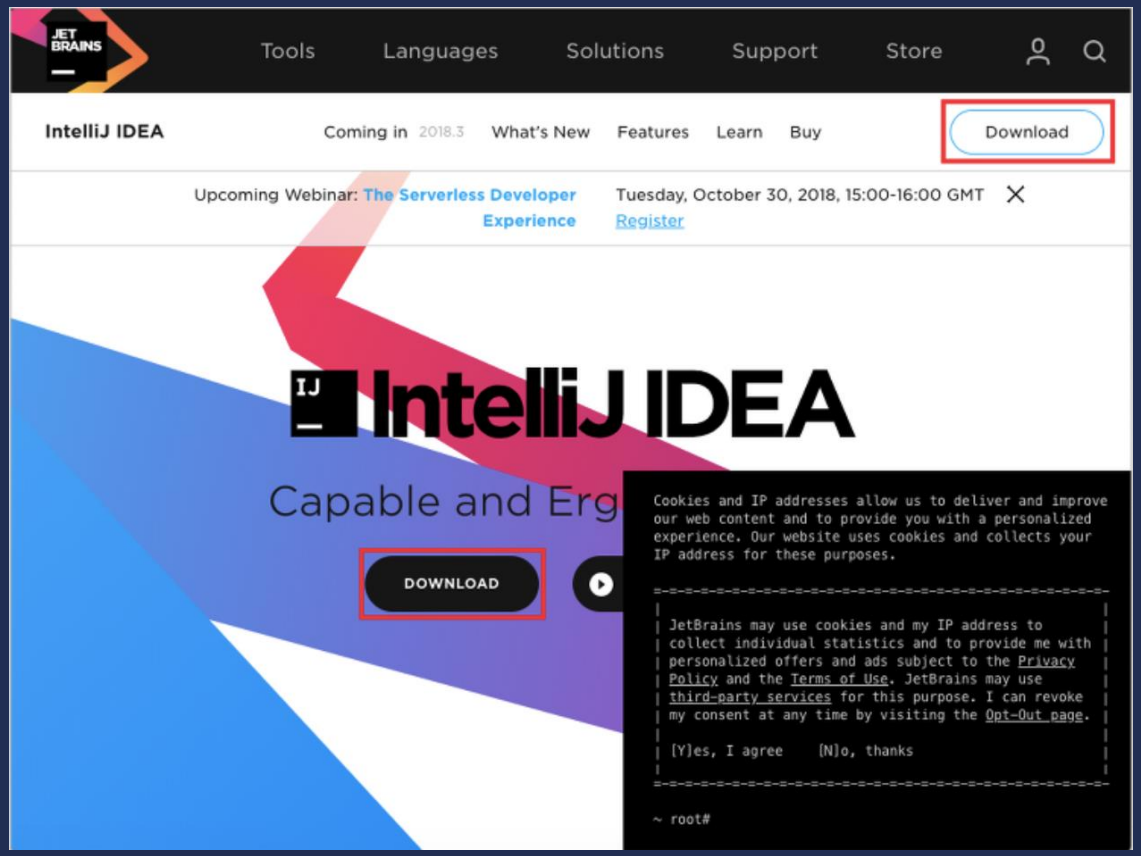


02 IDE – IntelliJ (IntelliJ - Installation)

IntelliJ 설치 방법

02 IDE – IntelliJ (IntelliJ - Installation)

Download 클릭

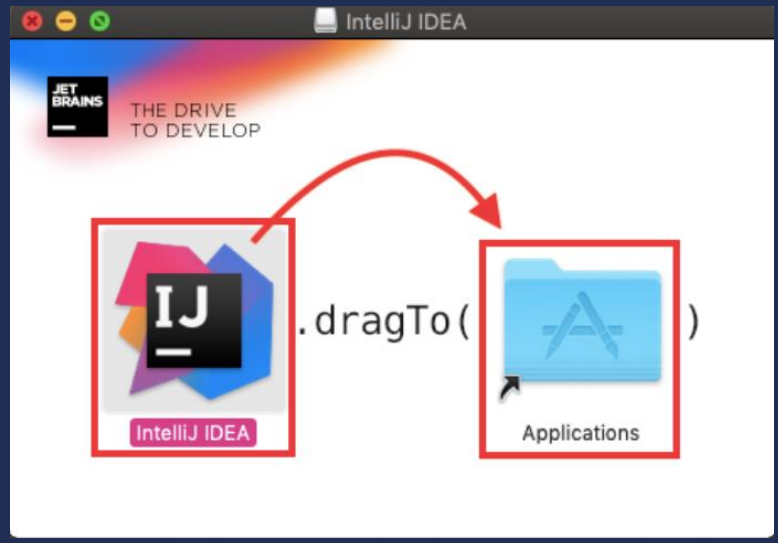


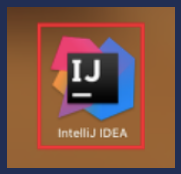
02 IDE – IntelliJ (IntelliJ - Installation)

Download 클릭 → 운영체제 선택 & Download 클릭

02 IDE – IntelliJ (IntelliJ - Installation)

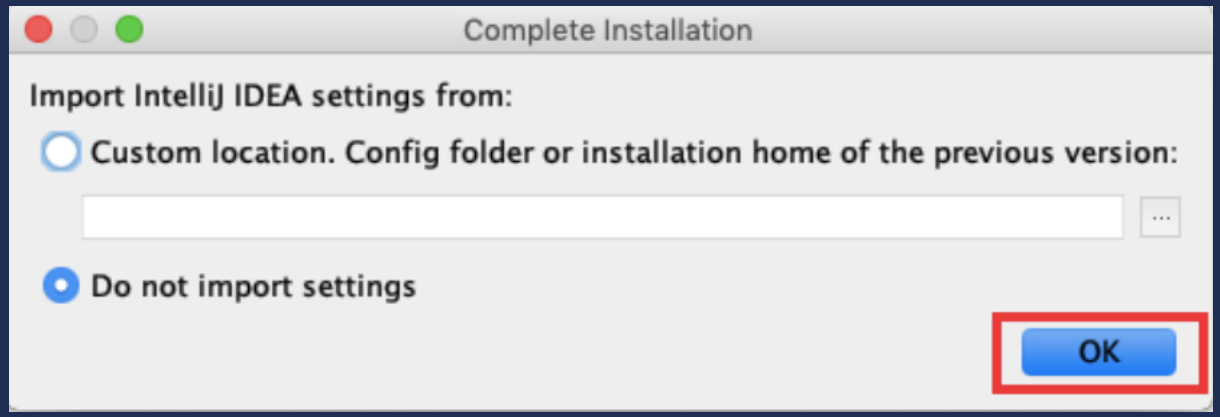
Download 클릭 → 운영체제 선택 & Download 클릭 → 설치 파일 클릭
→ 아이콘 "Applications" 로 옮김





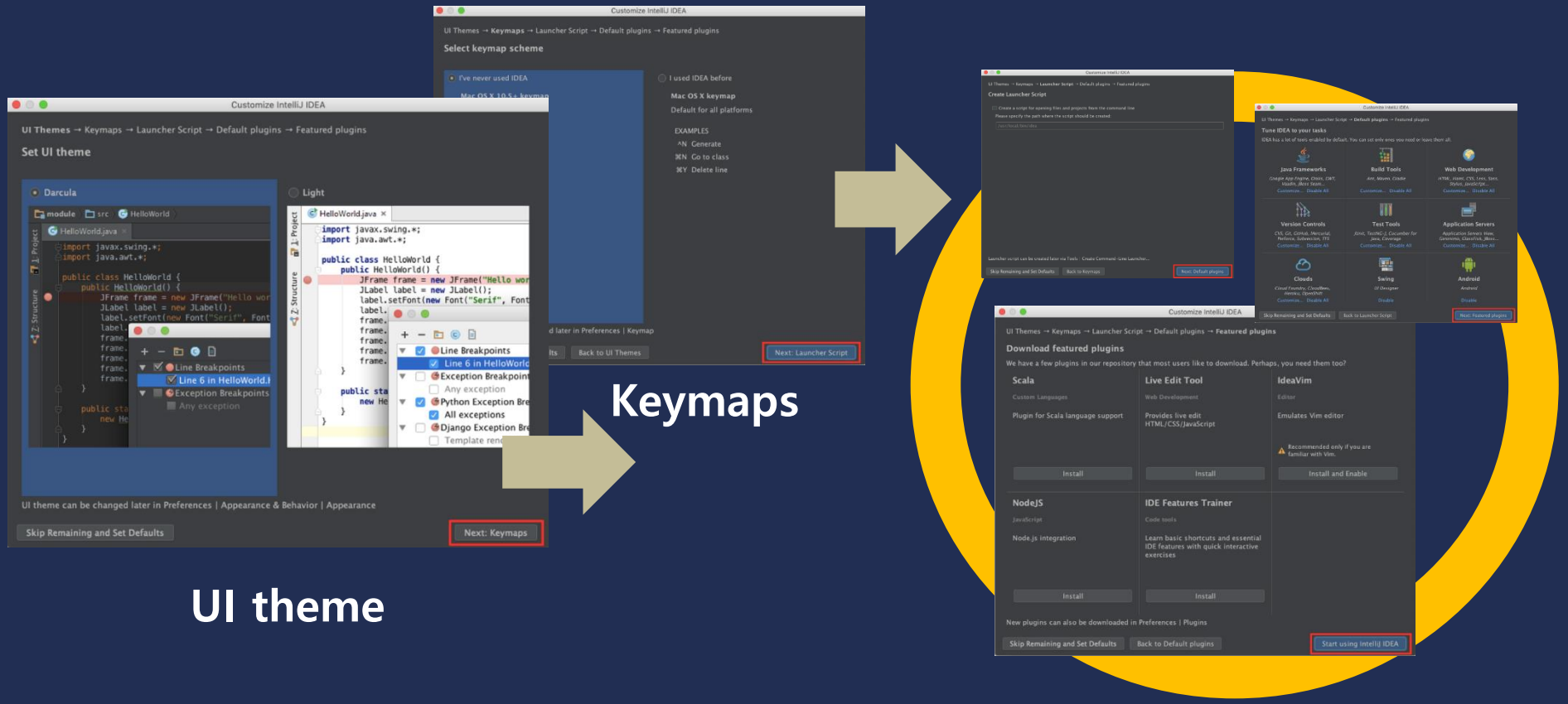
02 IDE – IntelliJ (IntelliJ - Installation)

"OK" 클릭



02 IDE – IntelliJ (IntelliJ - Installation)

"OK" 클릭 → UI theme → Keymaps → Launcher Script, Default & Featured plugins



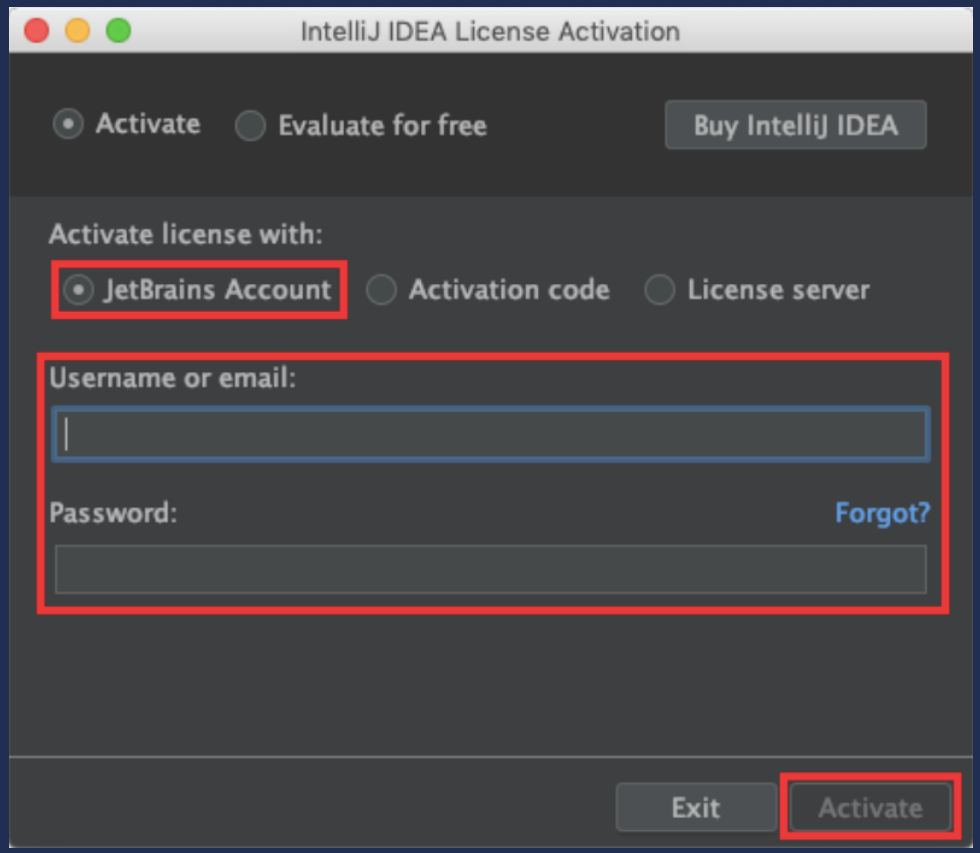
UI theme

Keymaps

Launcher Script, Default plugins, Featured plugins
(나중에도 설정가능하므로 Next)

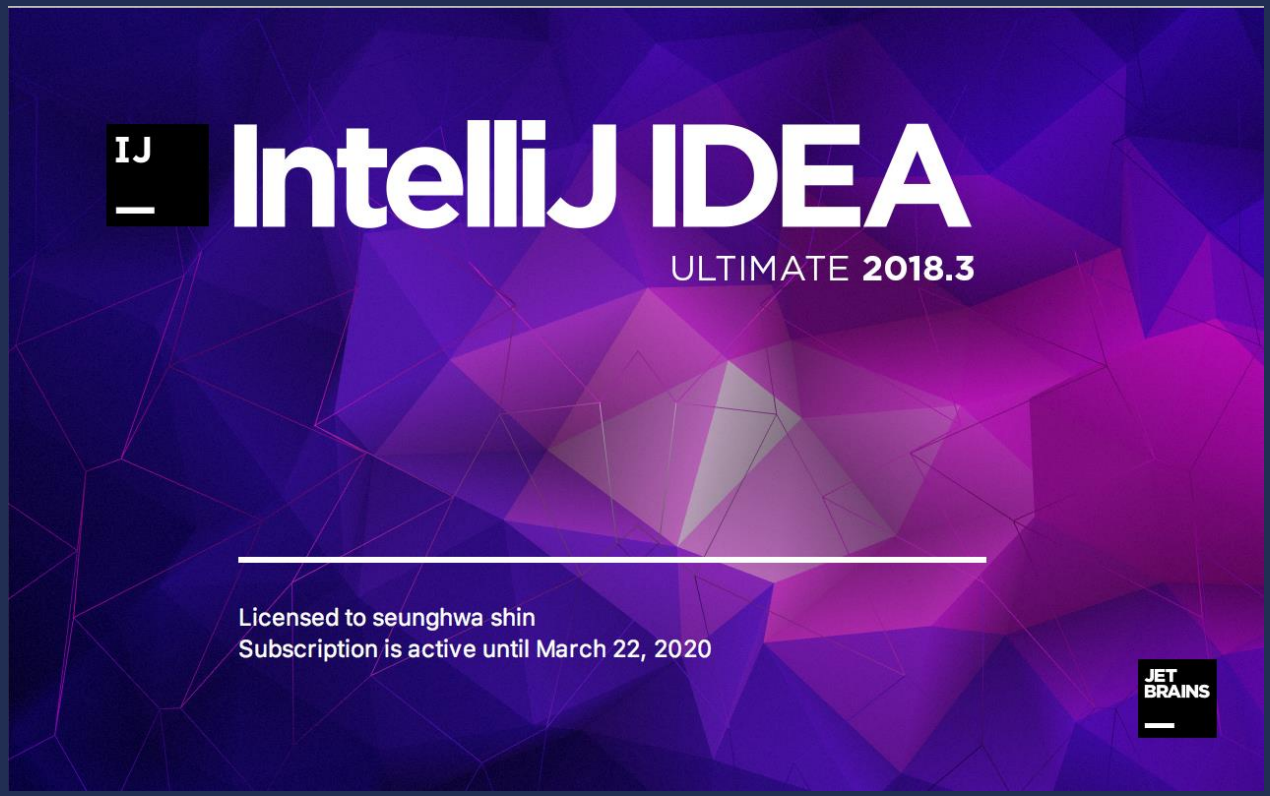
02 IDE – IntelliJ (IntelliJ - Installation)

"OK" 클릭 → UI theme → Keymaps → Launcher Script, Default & Featured plugins → 학생 계정으로 로그인



02 IDE – IntelliJ (IntelliJ - Installation)

"OK" 클릭 → UI theme → Keymaps → Launcher Script, Default & Featured plugins → 학생 계정으로 로그인 → 설치 완료



03 Build Environment – Gradle

1) Groovy 언어 기반의 빌드 스크립트를 사용하는 오픈소스 빌드 자동화 도구

2) Ant의 task 구조와 Maven의 repository 사용 가능.
→ Ant와 Maven의 이점을 합침.

3) 빌드 성능 향상을 위해 다양한 기능 제공

4) Plugin 을 통한 확장성 좋음.
→ PMD, Sonar 등의 static analysis tool을 plugin의 형태로 적용할 수 있음.

03 Build Environment – Gradle (Maven vs Gradle)

- Pom.xml을 이용한 정형화된 빌드 시스템
  → 동적인 요소인 빌드를
  정적인 xml로 정의하기 어려움.

  → 여러 dependency를 추가할 경우
  code가 길어지고 가독성이 떨어짐.

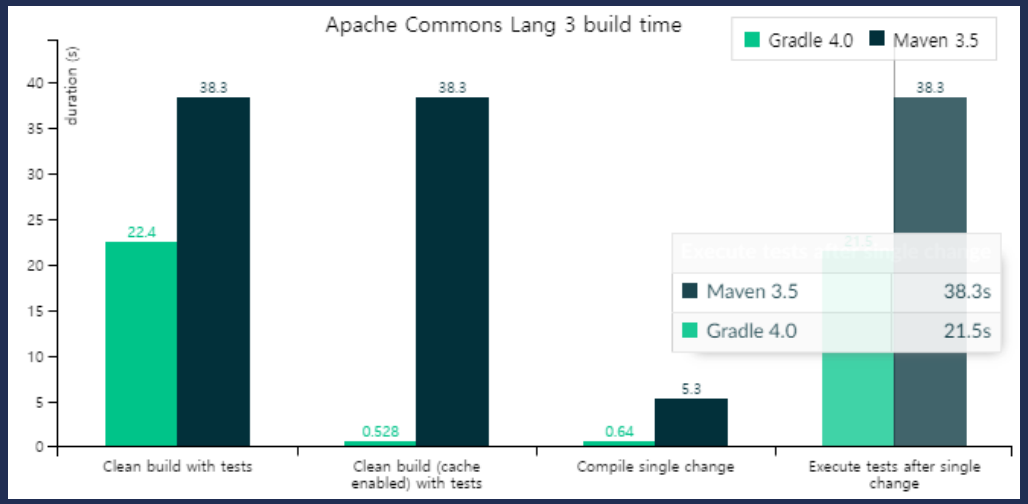
  → project 간에 종속성이 생기면
  이를 표현해주기가 복잡함.

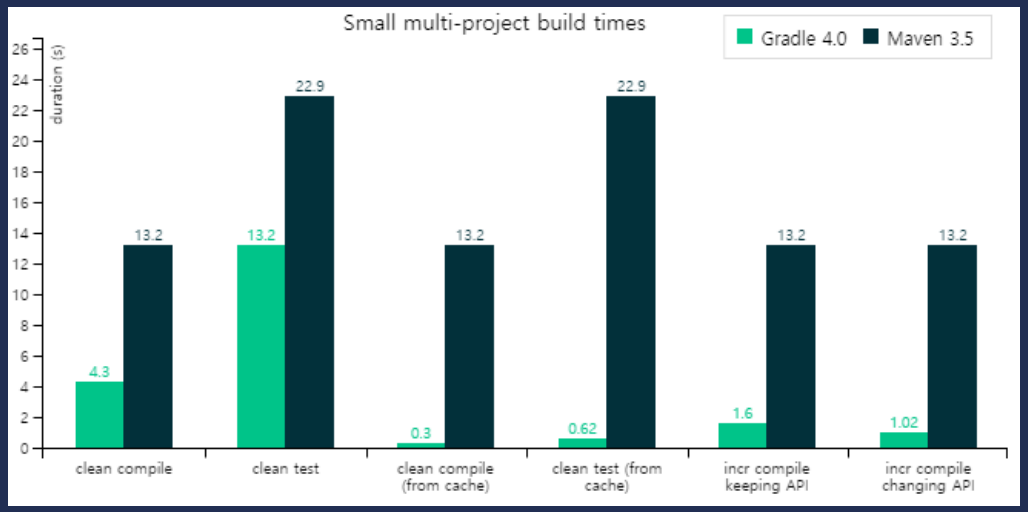
- JVM에서 돌아가는 groovy 언어 기반
  → 빌드의 동적인 요소를
  프로그래밍 언어처럼 표현 가능.

  → xml에 비해 code가 간결함.

  → project 간에 종속성이 생기면
  상속을 통해 해결 가능.

03 Build Environment – Gradle (Maven vs Gradle)

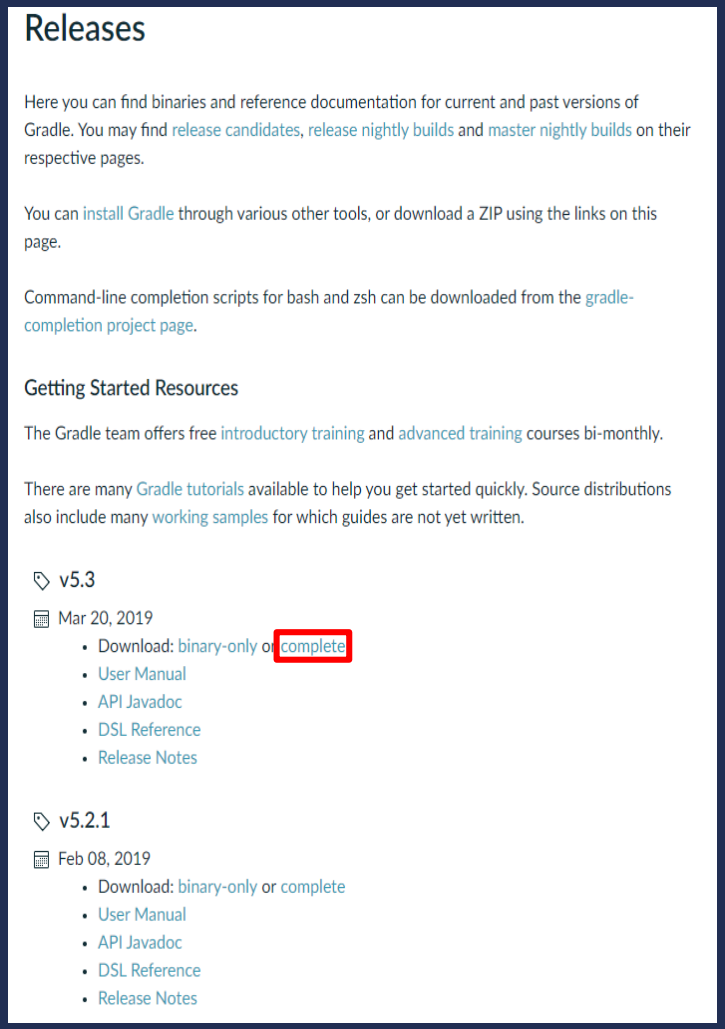




Build cache, Gradle daemon 같은 기술을 사용하여 maven에 비해 Build performance를 개선시킴.

→ 하나의 program을 개발하기 위해 다수의 인원이 여러 번 수행하는 총 build 횟수를 생각하면 performance의 개선은 상당한 이익이 됨.

03 Build Environment – Gradle (Installation)



Releases

Here you can find binaries and reference documentation for current and past versions of Gradle. You may find release candidates, release nightly builds and master nightly builds on their respective pages.

You can install Gradle through various other tools, or download a ZIP using the links on this page.

Command-line completion scripts for bash and zsh can be downloaded from the gradle-completion project page.

Getting Started Resources

The Gradle team offers free introductory training and advanced training courses bi-monthly.

There are many Gradle tutorials available to help you get started quickly. Source distributions also include many working samples for which guides are not yet written.

v5.3

Mar 20, 2019

- Download: binary-only or complete
- User Manual
- API Javadoc
- DSL Reference
- Release Notes

v5.2.1

Feb 08, 2019

- Download: binary-only or complete
- User Manual
- API Javadoc
- DSL Reference
- Release Notes

IntelliJ IDE 에서는 Gradle Project를 바로 생성할 수 있음.

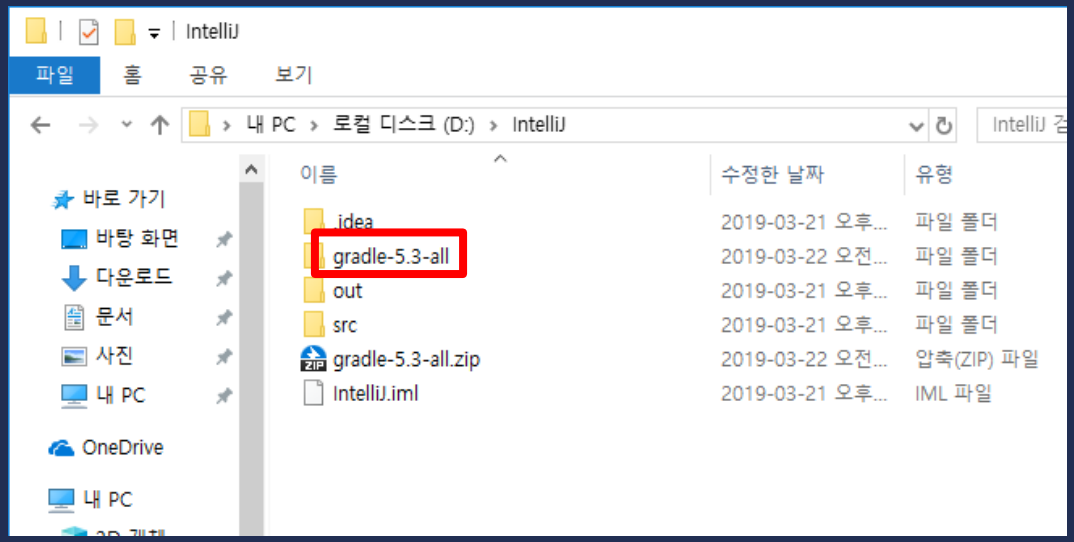
하지만 필요에 따라 다른 version의 Gradle을 사용할 수 있음.

https://gradle.org/releases/

JDK 7 이상을 사용하는 대부분의 OS에 설치 가능하며 Groovy Library를 포함하여 배포.

→ Groovy Library를 따로 설치할 필요 없음.

03 Build Environment – Gradle (Installation)



다운로드 받은 Gradle file은

원하는 경로에 저장 후

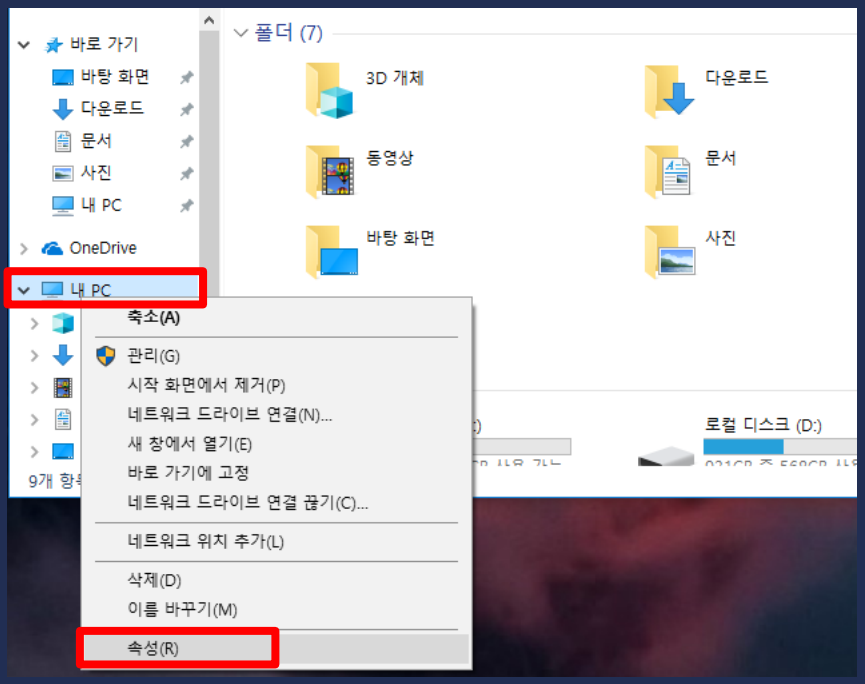
압축을 해제한다.

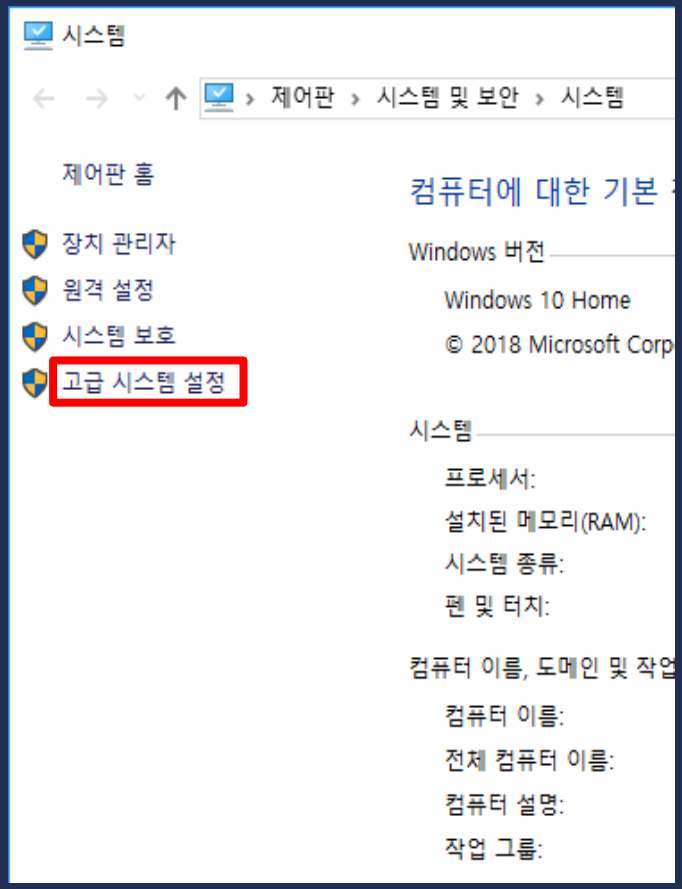
이후 모든 경로에서

바로 접근해 사용이 가능하도록

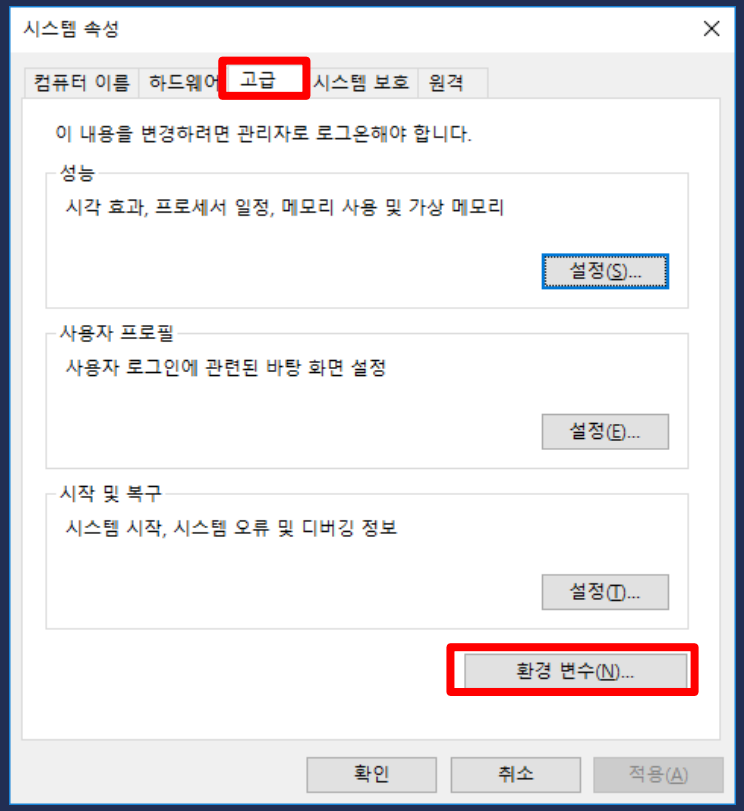
환경 변수 등록을 해준다.

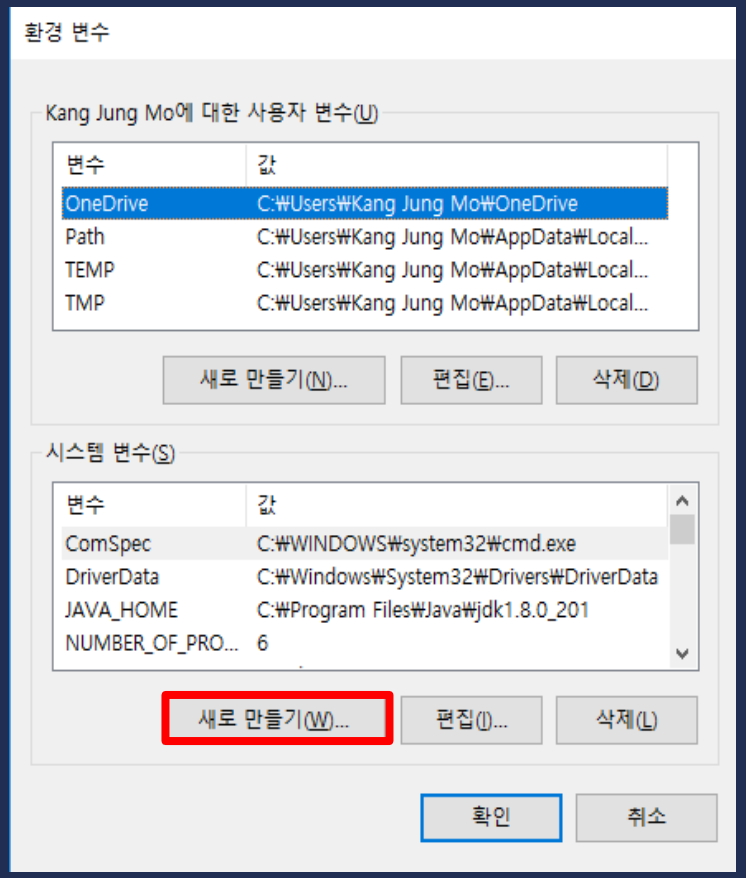
03 Build Environment – Gradle (Installation)



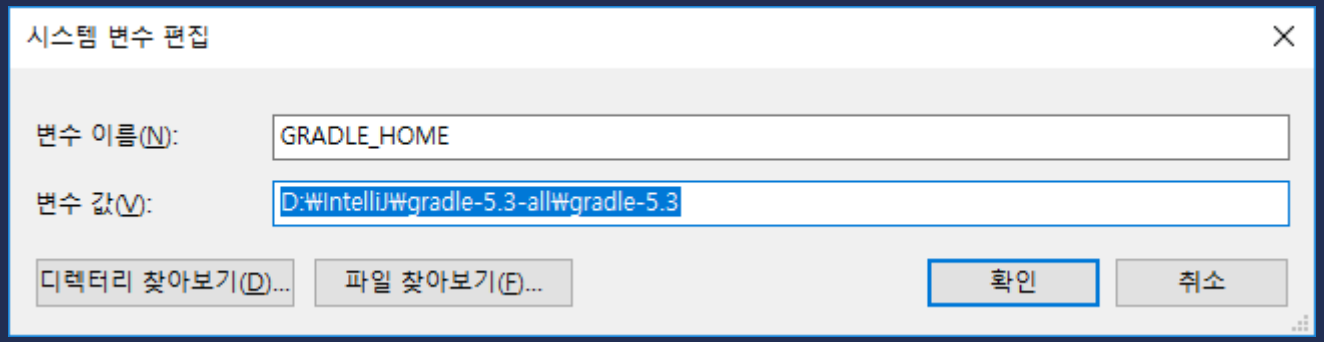


03 Build Environment – Gradle (Installation)





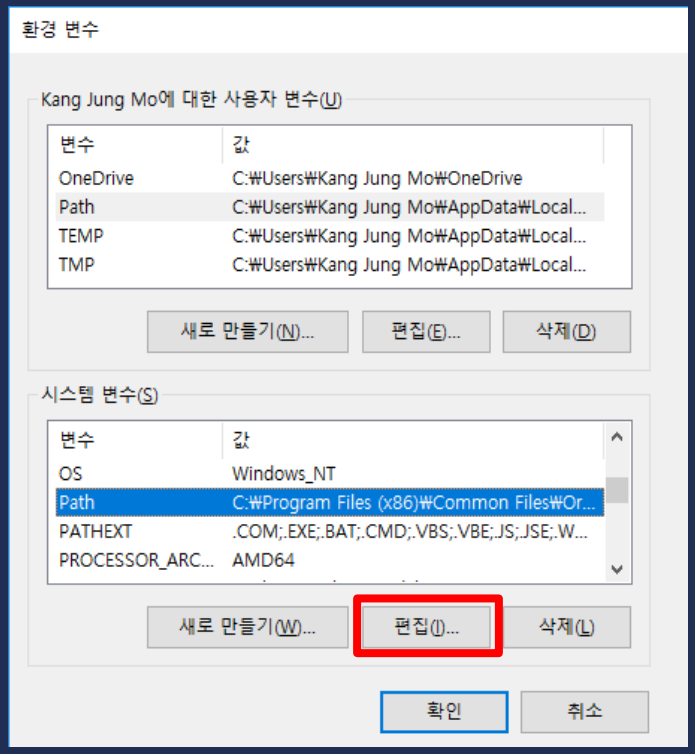
03 Build Environment – Gradle (Installation)

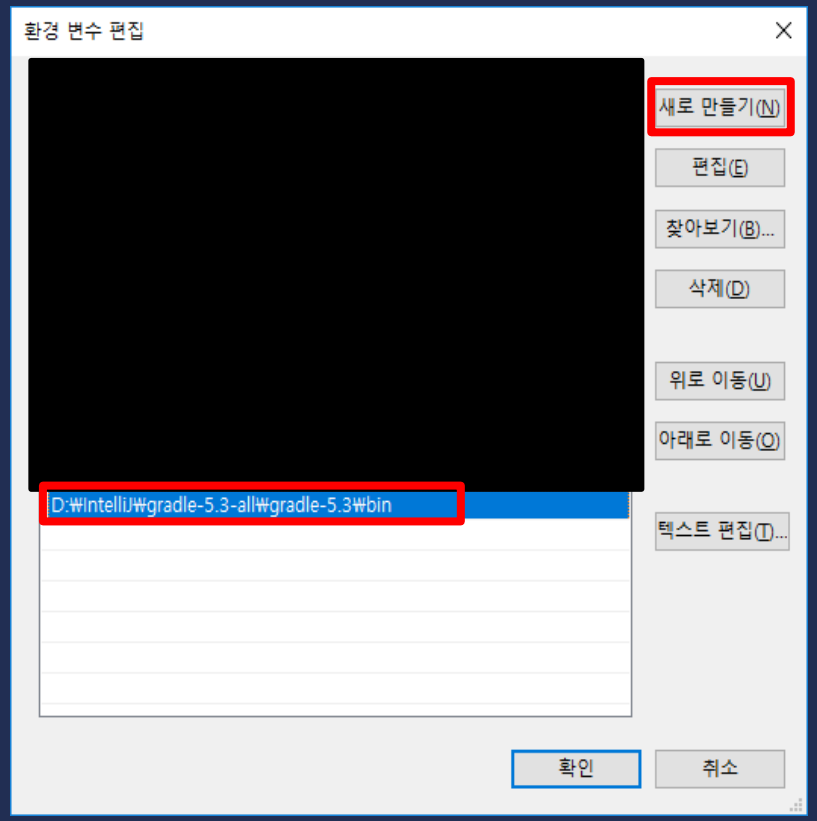


변수 이름 및 변수 값을 설정해준다.

변수 값에는 Gradle zip 파일을 받아 압축을 해제한 경로를 설정해준다.

03 Build Environment – Gradle (Installation)





이후 Path 변수에도 경로를 추가해준다.

03 Build Environment – Gradle (Installation)

```
C:₩Users₩Kang Jung Mo>gradle -version

Welcome to Gradle 5.3!

Here are the highlights of this release:
 - Feature variants AKA "optional dependencies"
 - Type-safe accessors in Kotlin precompiled script plugins
 - Gradle Module Metadata 1.0

For more details see https://docs.gradle.org/5.3/release-notes.html


------------------------------------------------------------
Gradle 5.3
------------------------------------------------------------

Build time:   2019-03-20 11:03:29 UTC
Revision:     f5c64796748a98efdbf6f99f44b6afe08492c2a0

Kotlin:       1.3.21
Groovy:       2.5.4
Ant:          Apache Ant(TM) version 1.9.13 compiled on July 10 2018
JVM:          1.8.0_201 (Oracle Corporation 25.201-b09)
OS:           Windows 10 10.0 amd64
```

환경 변수 설정이 끝났으면,

Win + R 을 눌러 실행 창을 띄우고

cmd를 입력해 커맨드 창을 띄운다.

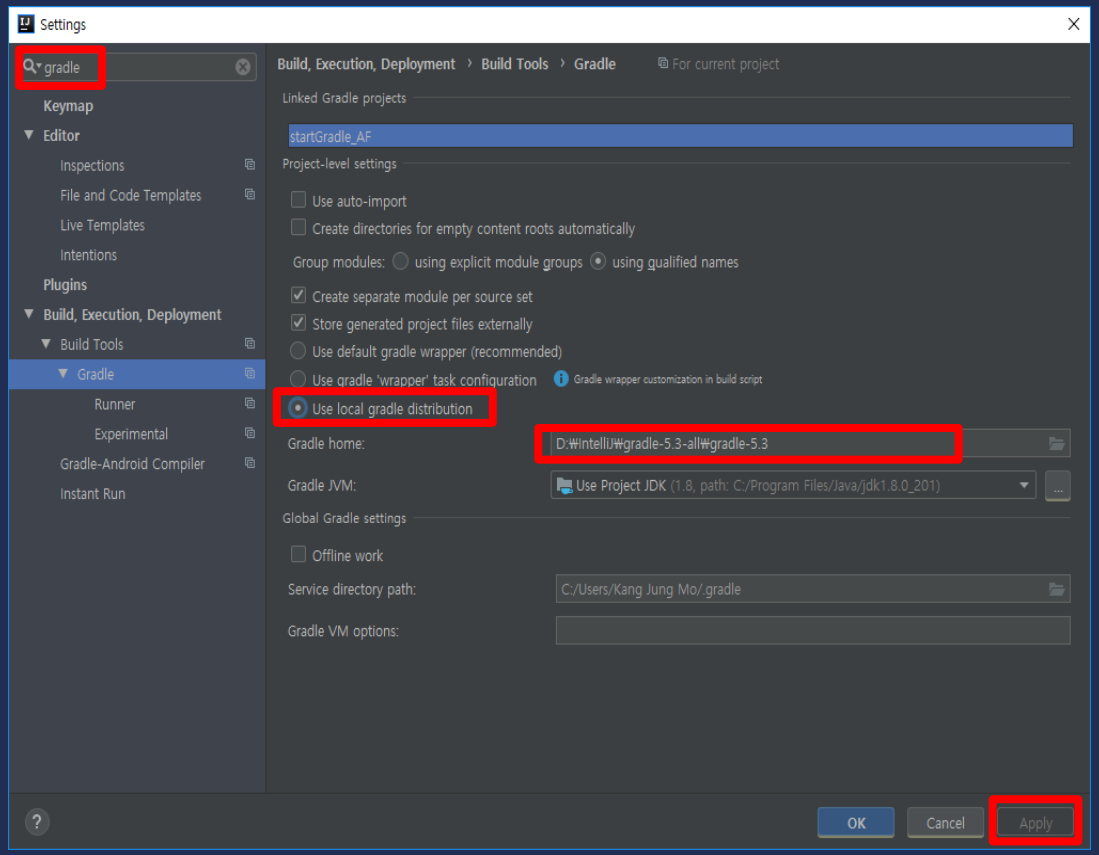
gradle –version 커맨드를 입력해

자신이 다운로드 받은

Gradle의 version이 정상적으로

나오는지 확인한다.

03 Build Environment – Gradle (Installation)



IntelliJ IDE를 실행시켜

좌측 상단 메뉴바에서

File – Setting 창을 띄우고

검색창에 Gradle을 검색한다.

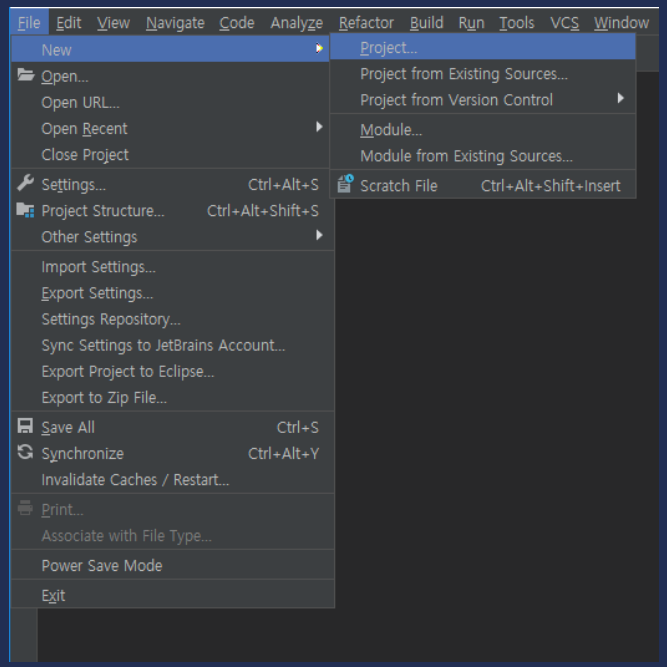
User local Gradle distribution 옵션에

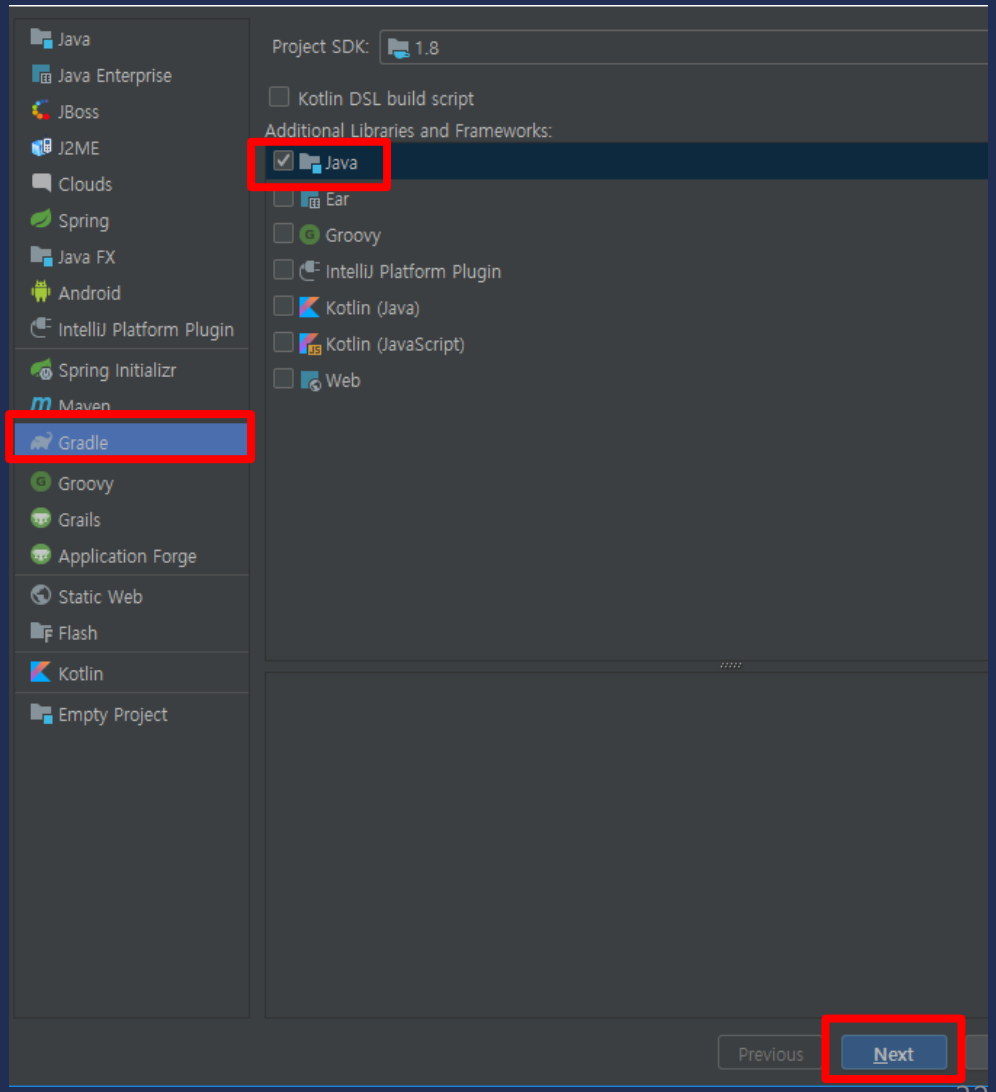
체크 후 Gradle home 경로를

설정해주고 Gradle JVM Version을

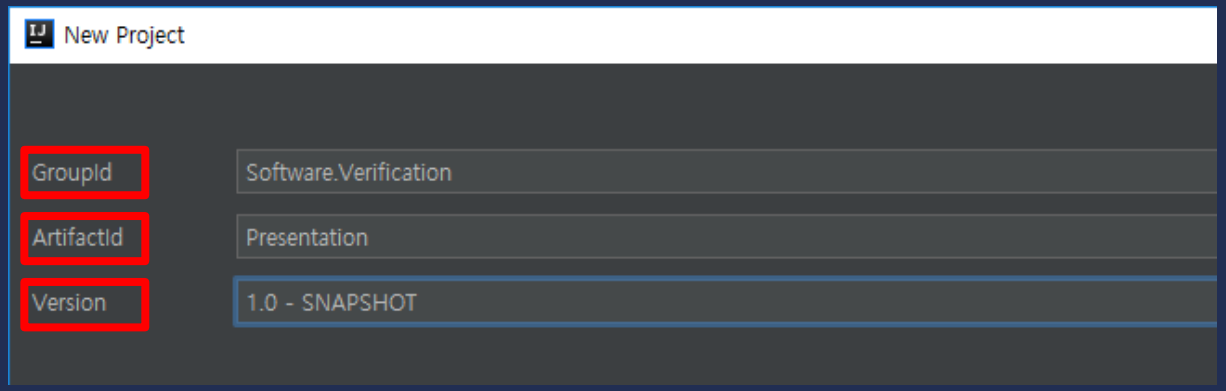
확인하고 apply를 누른다.

03 Build Environment – Gradle (Gradle project)





03 Build Environment – Gradle (Gradle project)

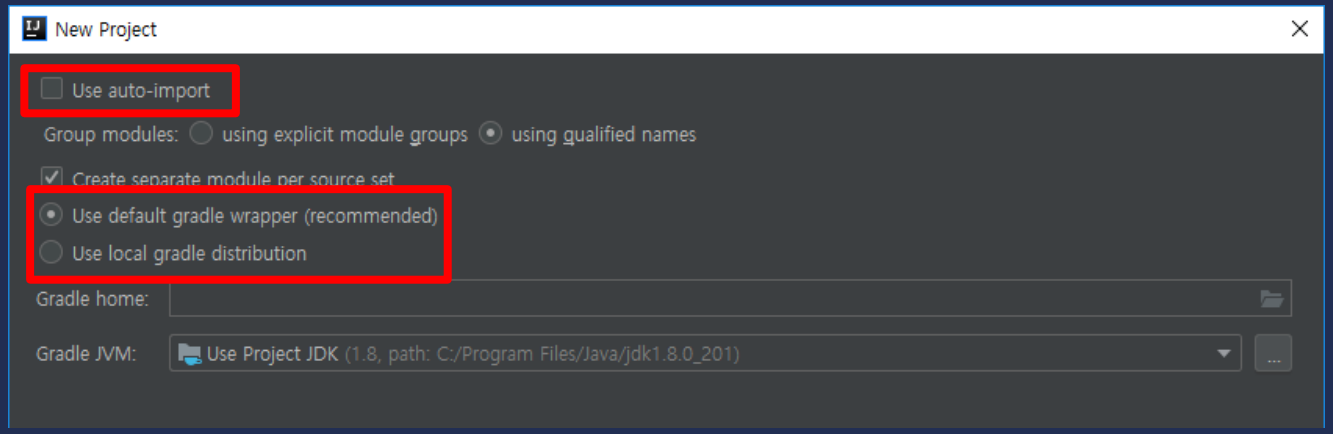


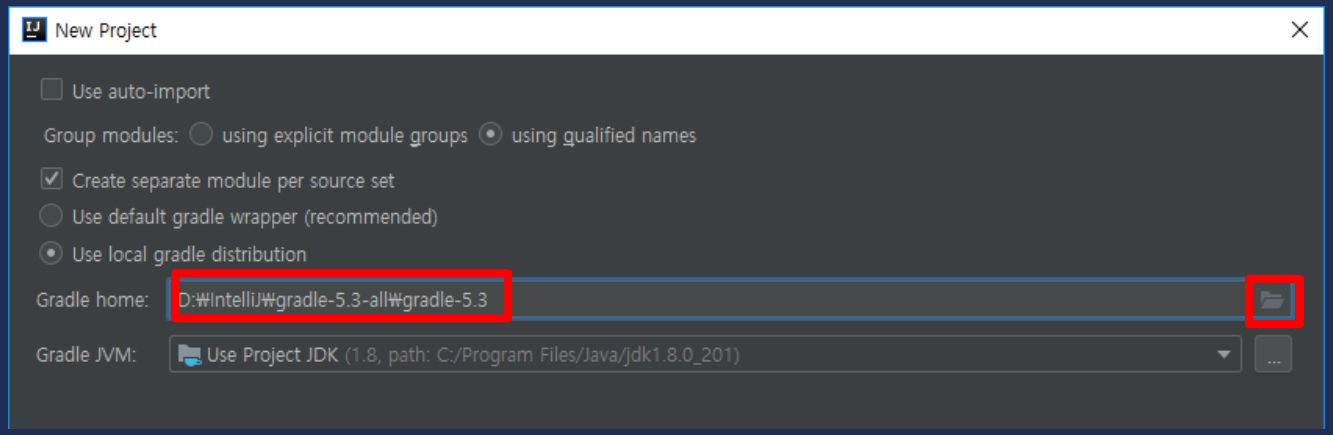
GroupId : 내 project를 모든 project 중에서 고유하게 식별하게 해 주는 Id.
package 명명 규칙을 따르며
최소한 내가 control 할 수 있는 domain 이름을 가져야 함.
ex) kr.ac.konkuk.dslab, org.apache.maven

ArtifactId : 버전 정보를 생략한 jar 파일의 이름을 적는다.
단, 소문자로만 작성하며 특수문자는 사용하지 않는다.

Version : 숫자와 . 으로 이루어진 일반적인 version 형태를 사용한다.

03 Build Environment – Gradle (Gradle project)





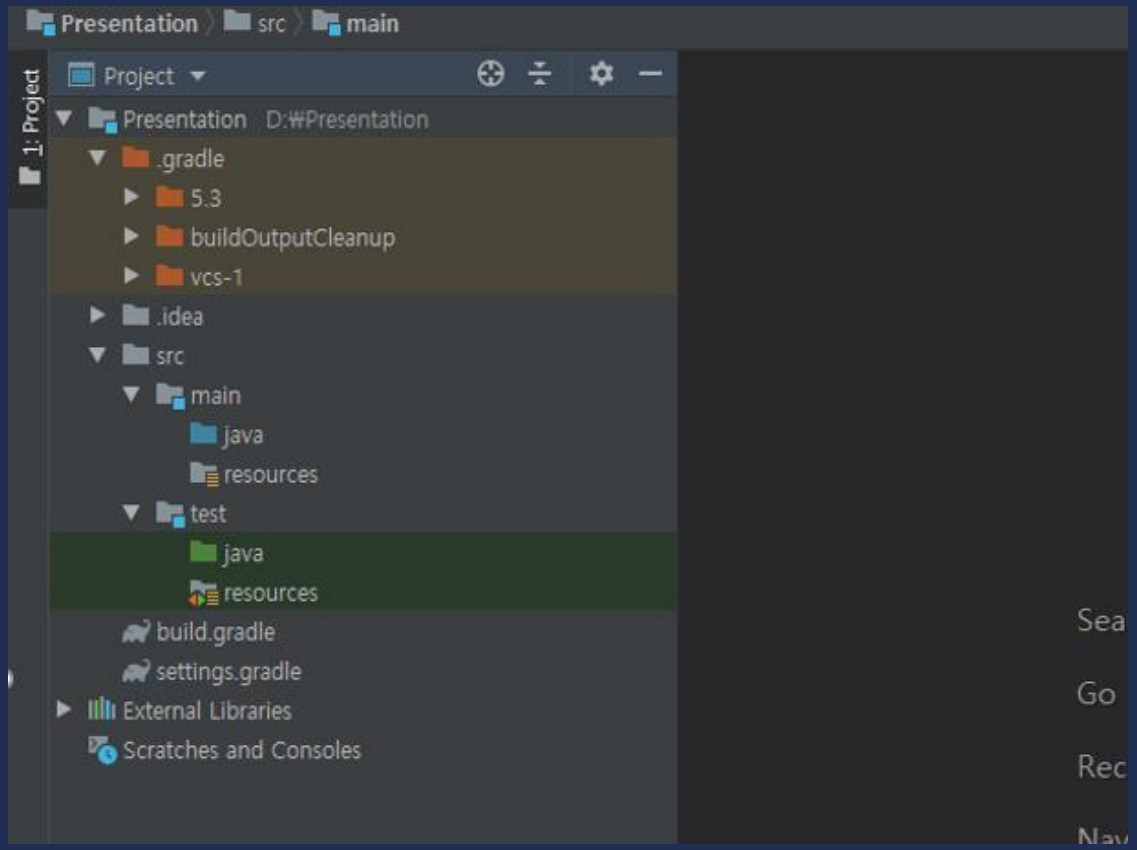
Use auto-import

- 의존성 library를 자동으로 추가해줌.

default / local

- IntelliJ IDE 내부의 Gradle을 사용할 것인지, 직접 다운 받은 version의 Gradle을 사용할 것인지 선택 가능.

03 Build Environment – Gradle (Gradle project)



이후 넘어가는 창에서

좌측 하단의

Finish를 누르면 옆과 같은

Gradle project가 생성됨.

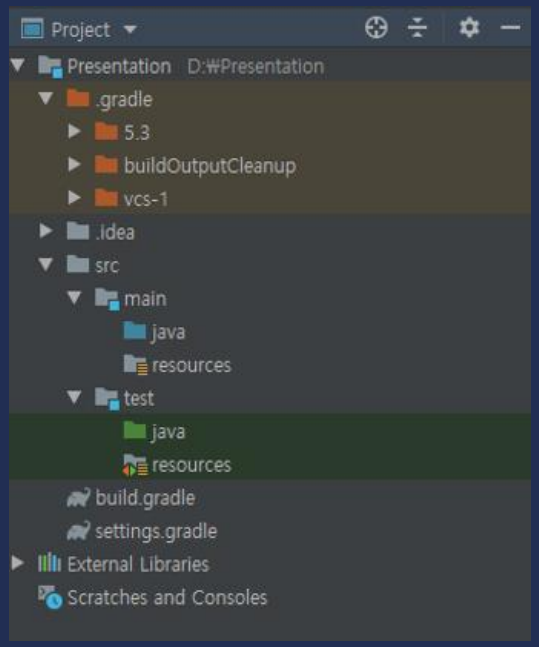
참고)

만약 default Gradle 이 아니라

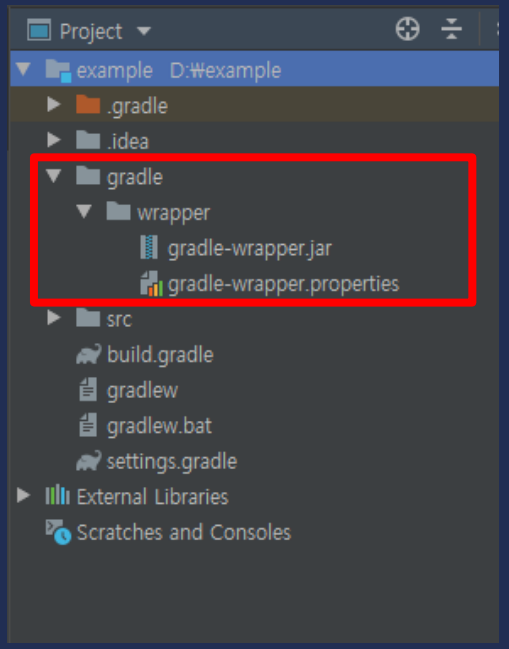
Local Gradle 로 선택했을 경우

gradle 폴더가 없음.

03 Build Environment – Gradle (Gradle project)

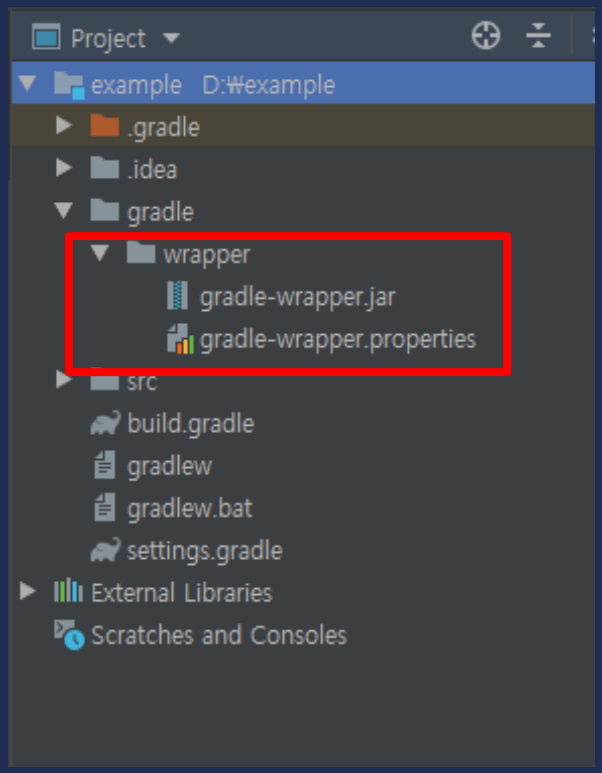


Default 가 아닐 때
→ gradle 폴더가 없음.



Default 일 때
→ gradle 폴더가 있음.

03 Build Environment – Gradle (Gradle project)



※ Gradle wrapper?

Gradle wrapper는 project를 build할 때

gradle/wrapper 디렉토리를 생성하고

해당 디렉토리에 jar을 복사함.

이 상태로 VCS(Version Control System)에 올리면

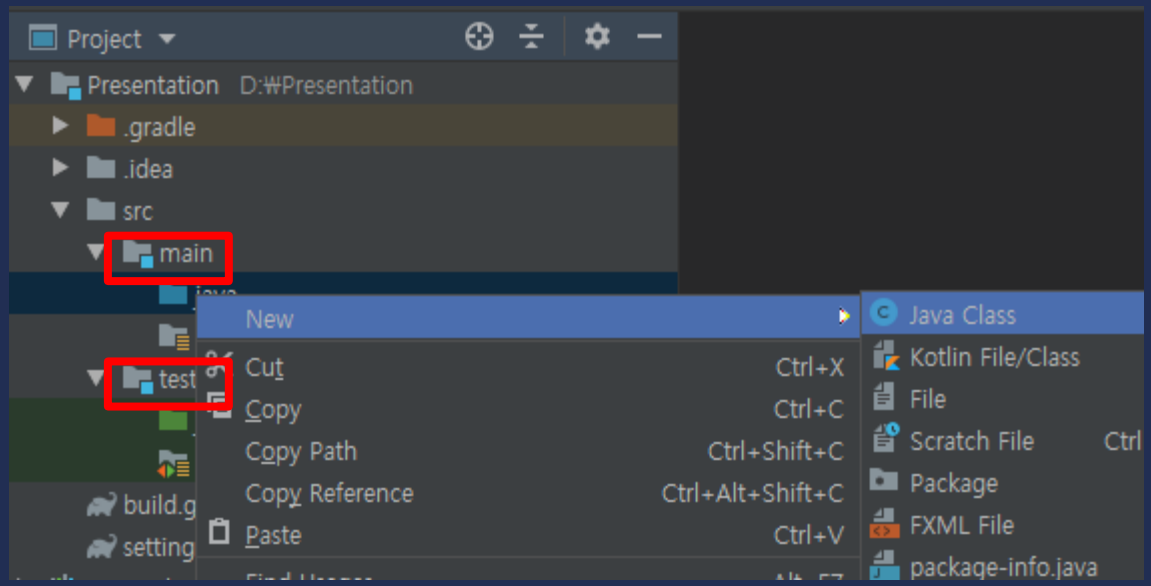
다른 사람들은 java나 Gradle을

추가로 설치할 필요가 없어지며,

빌드를 하는 유저들이 모두 동일한 버전의 Gradle을

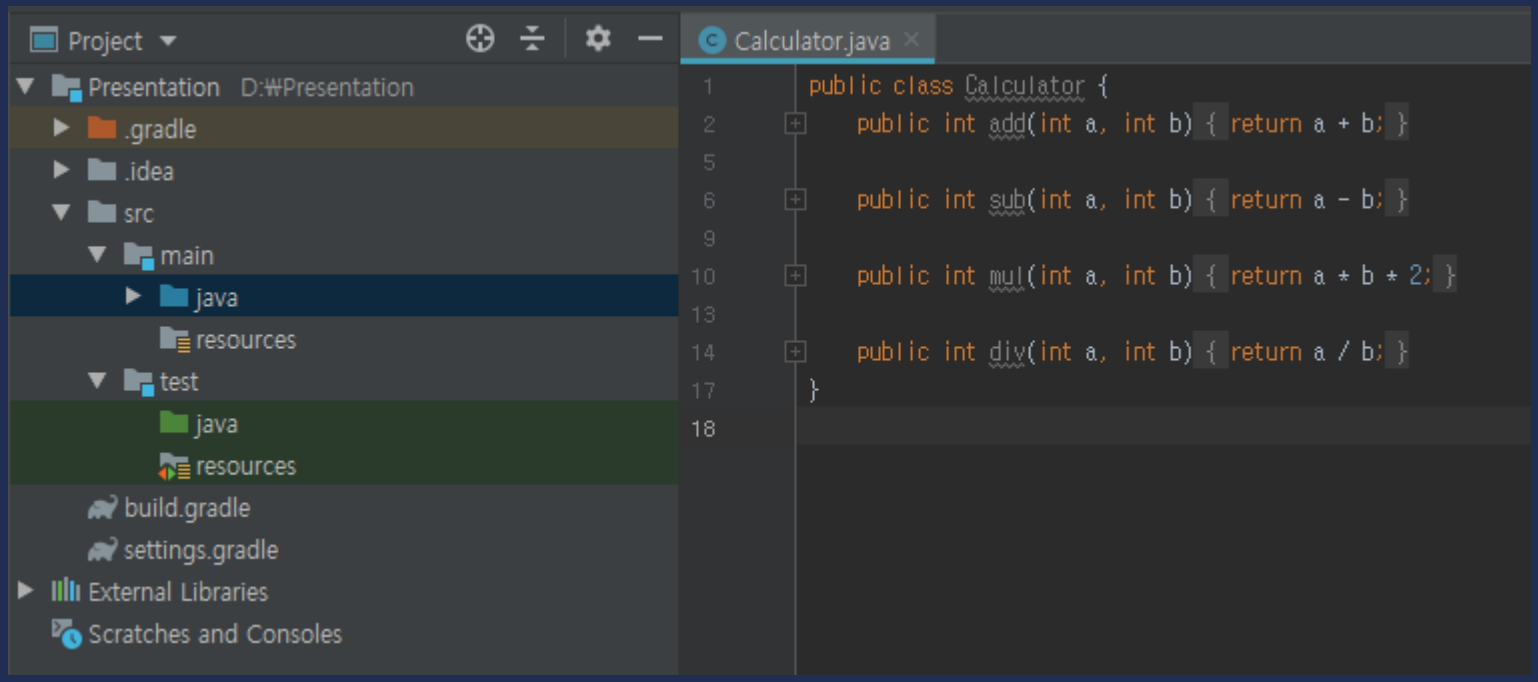
사용하는 것을 보장받게 됨. → 사용 권장.

03 Build Environment – Gradle (Gradle project)



src 폴더 내에 source code를 저장하는 main 폴더와 해당 source code에 대한 test source code를 저장하는 test 폴더로 나누어져 있다.

03 Build Environment – Gradle (Gradle project)



Build test 및 JUnit test를 위한 test code 작성

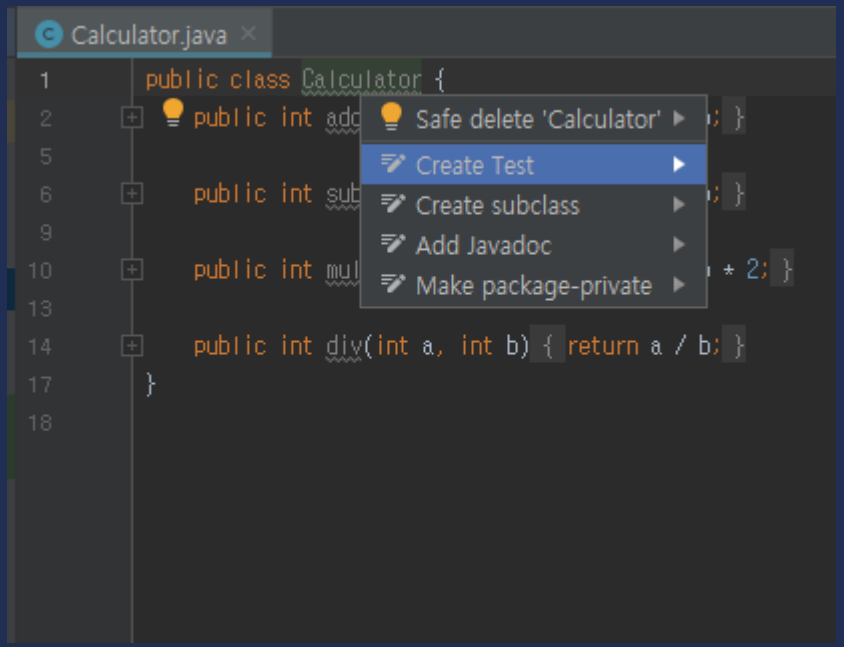
03 Build Environment – Gradle (Gradle project)

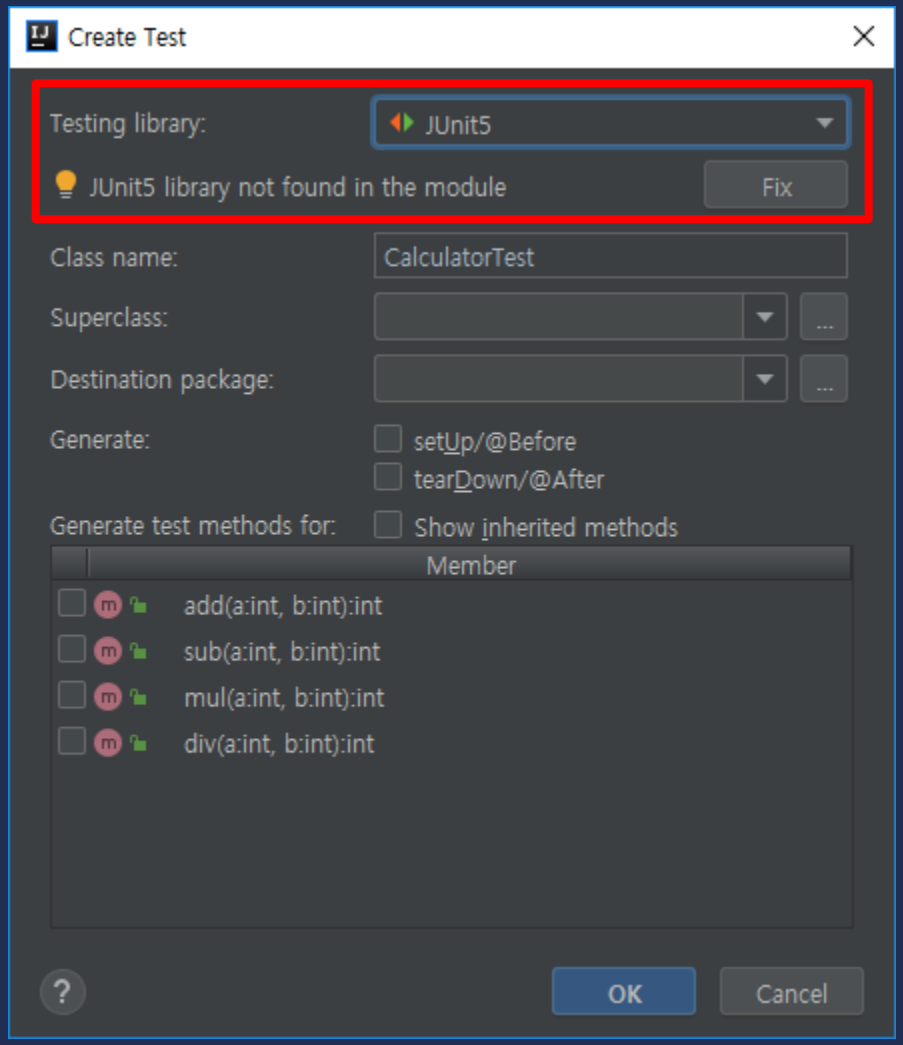
Test를 원하는 class에 커서를 두고

Alt + Enter → Create Test 클릭.

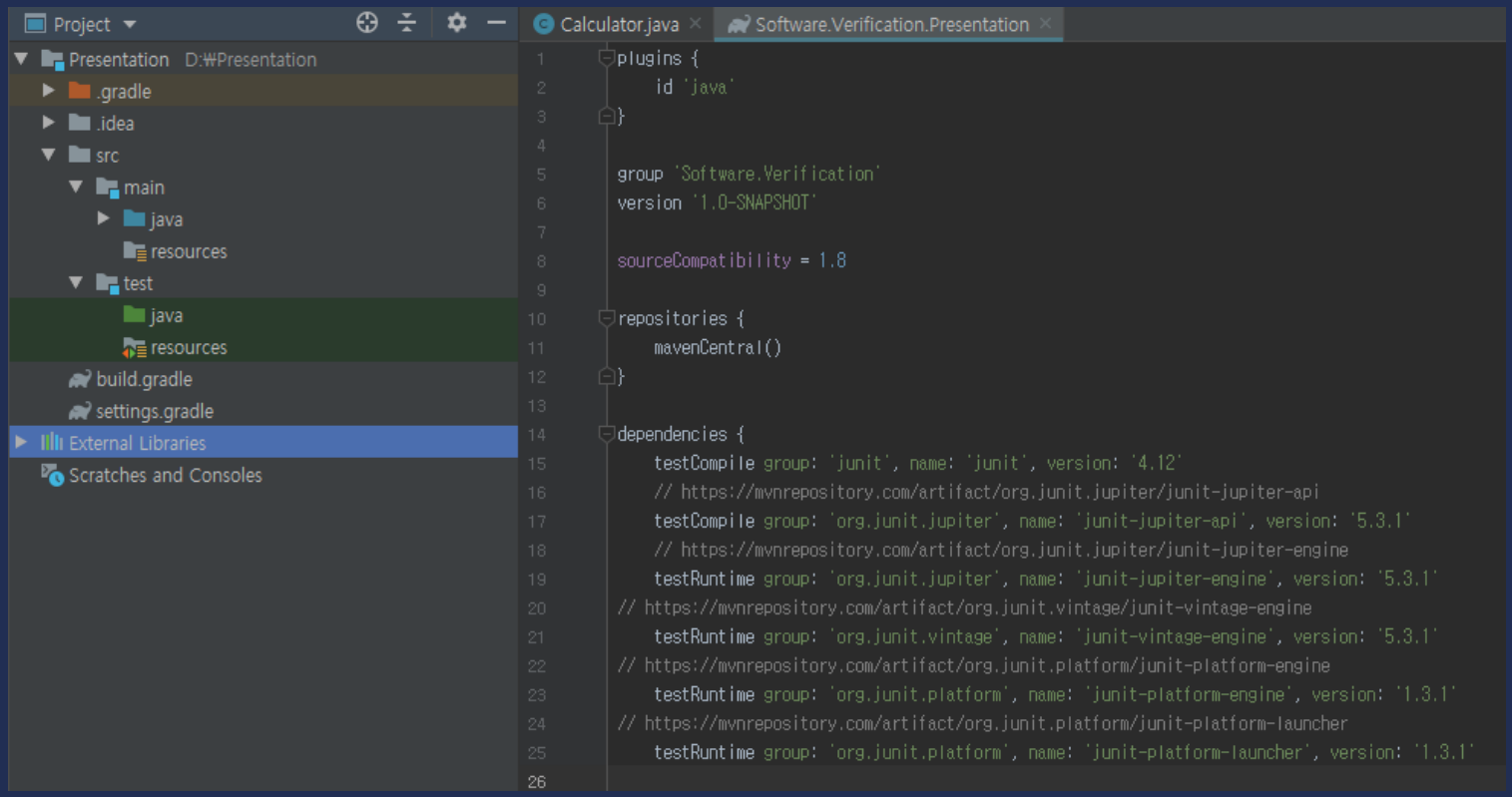
이 때 testing 을 위한 JUnit 5 library가

없음을 알 수 있다.





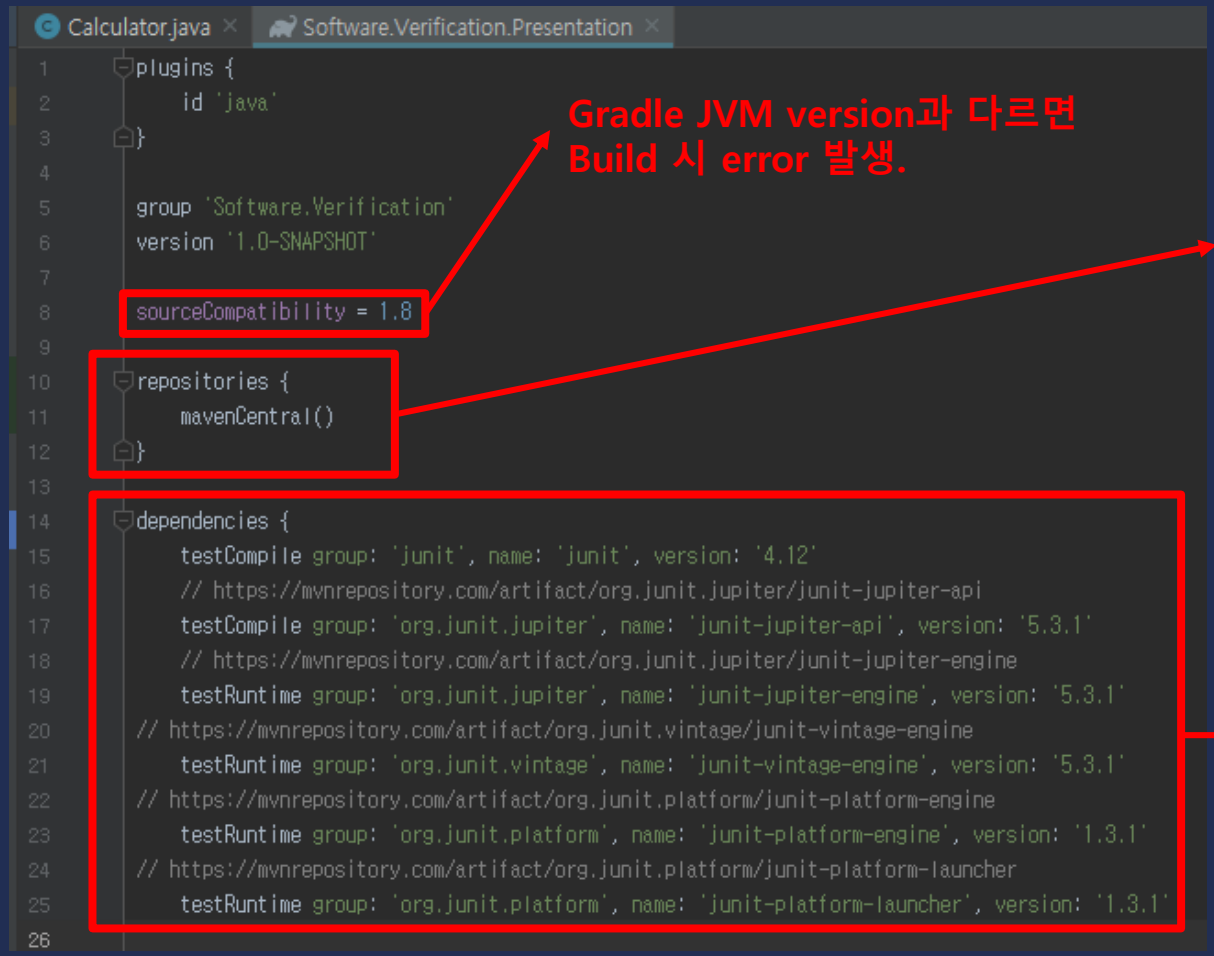
03 Build Environment – Gradle (Gradle project)



Fix를 통해 download 받을 수도 있지만

Gradle을 통해 build 하면서 의존성 library에 추가해 사용 가능하다.

03 Build Environment – Gradle (Gradle project)



[Repositories]

의존성 라이브러리가

저장되어 있는

저장소를 말함.

ex) https://mvnrepository.com/

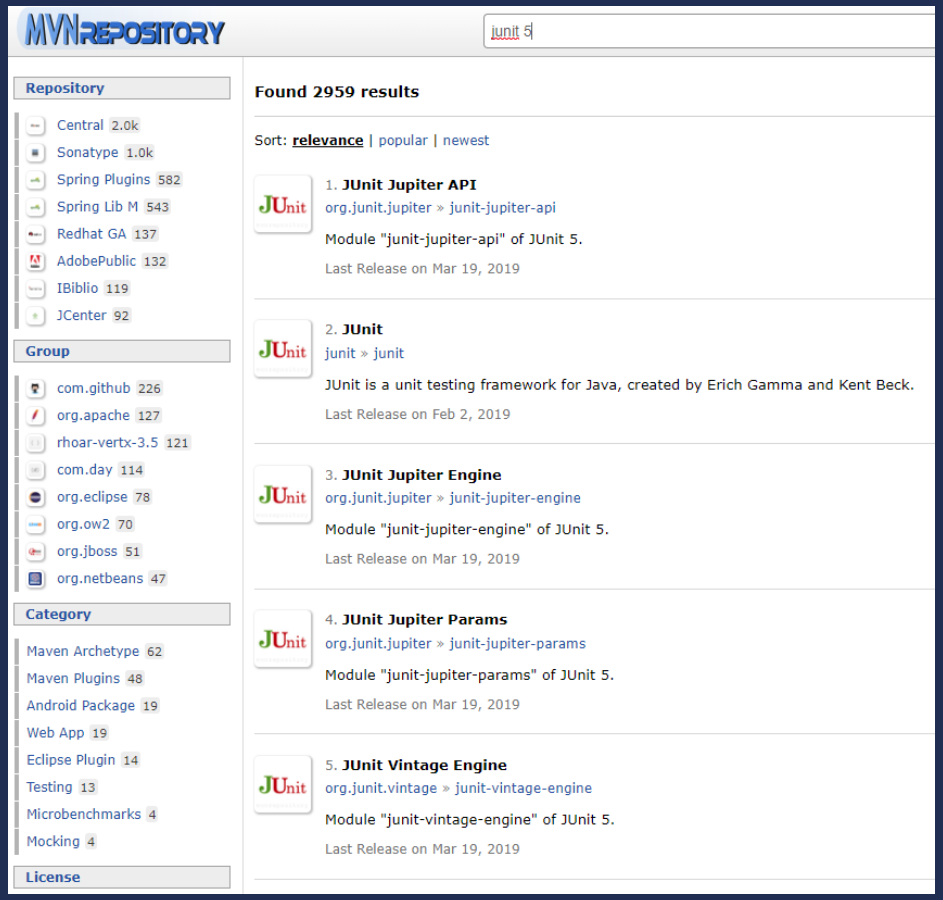
[dependencies]

해당 project 를 build 할 때

필요한 (dependent한)

외부 라이브러리들을 서술함.

03 Build Environment – Gradle (Gradle project)

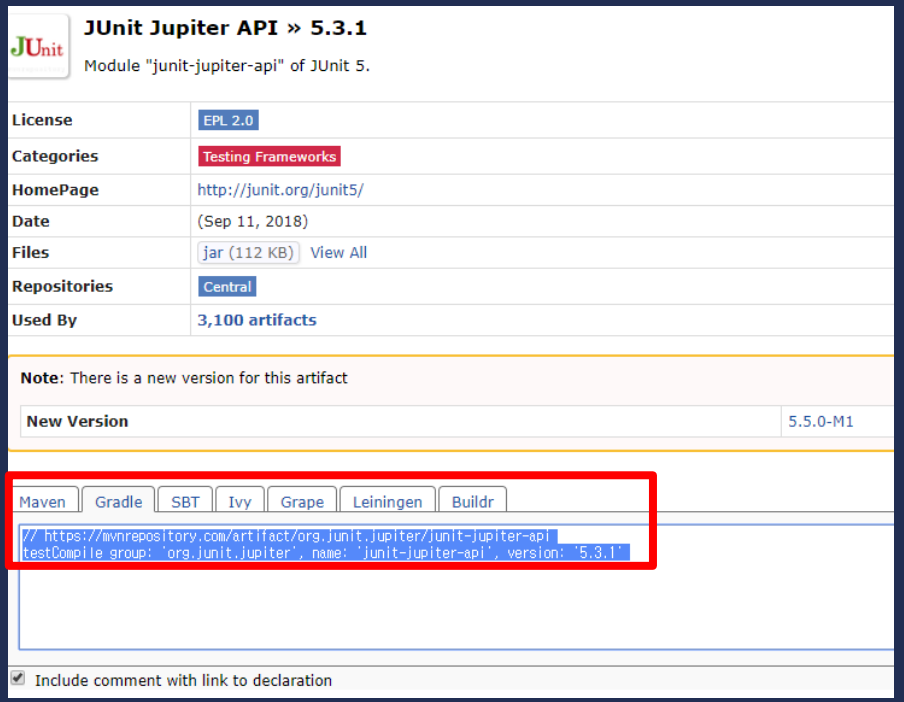


EPL : Eclipse Public License

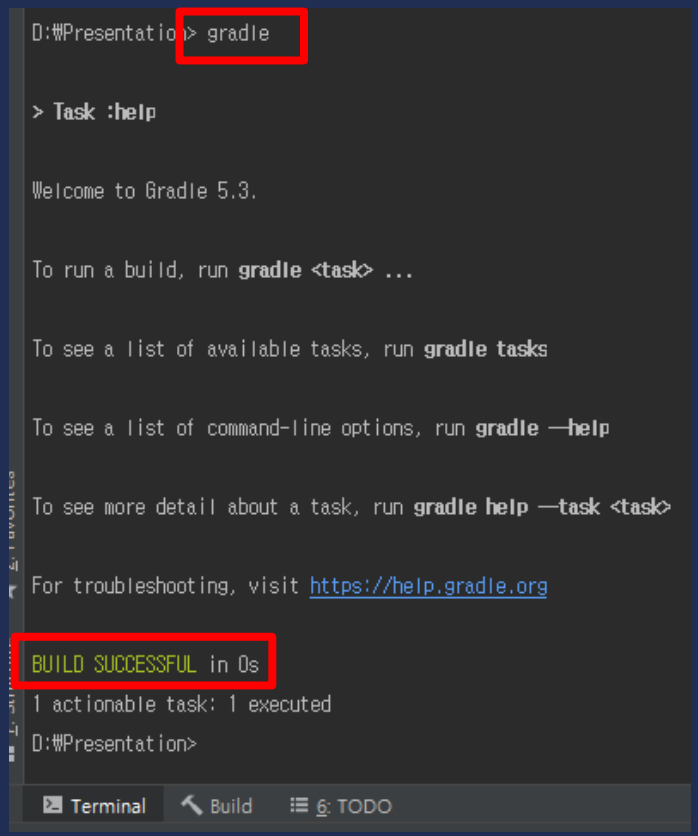
→ 특허 관련 소송 중심의 용어를 제거하여

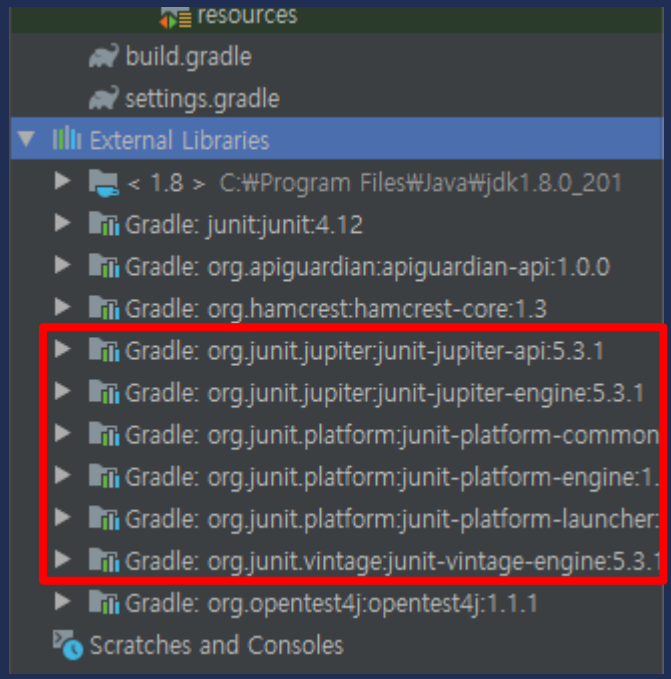
CPL을 대체한 오픈소스 소프트웨어 라이센스.

CPL : Common Public License



03 Build Environment – Gradle (Gradle project)

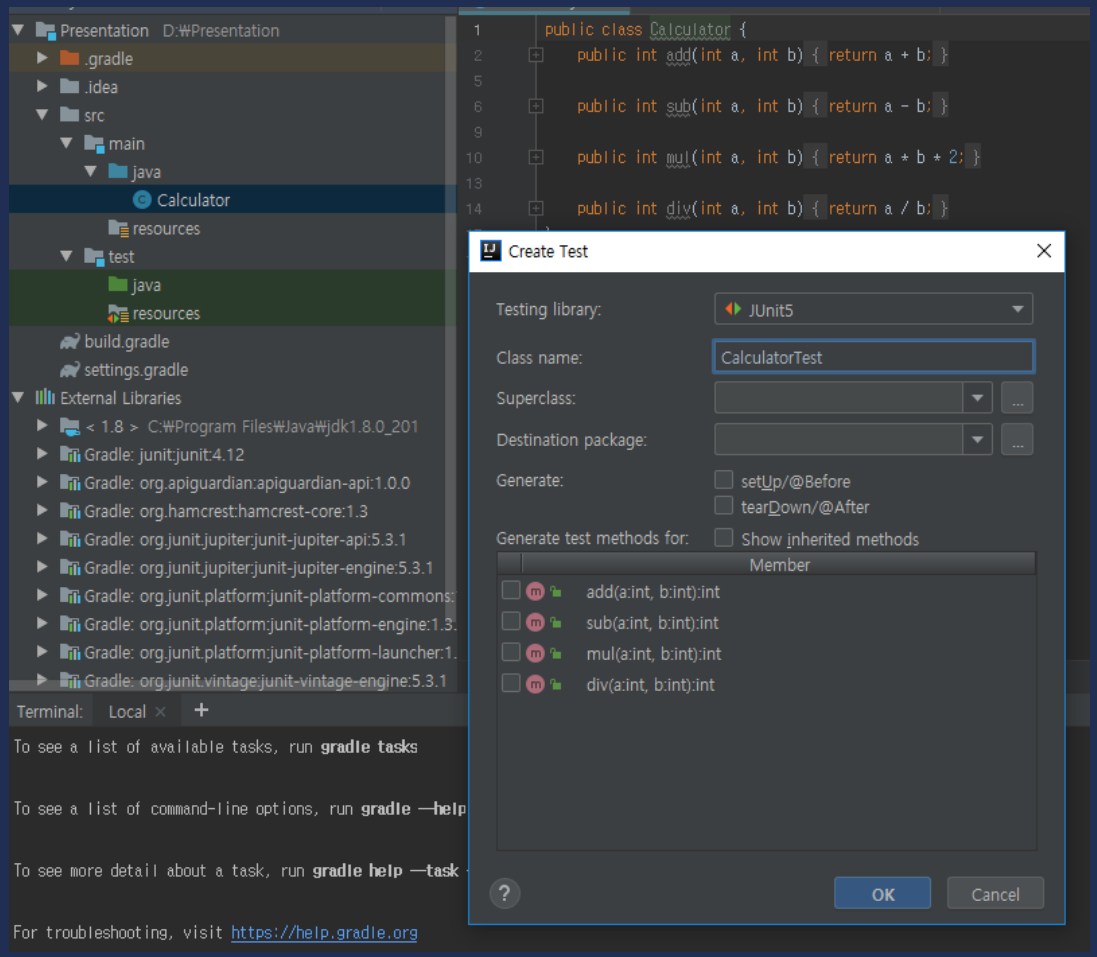




아래의 terminal 창에서 gradle 명령어로 build를 함.

External Libraries 에 JUnit 5 libraries 가 추가됨.

03 Build Environment – Gradle (Gradle project)



이전과 같이 Create test 창을 띄웠을 때 JUnit 5를 다운로드 받을 필요 없이, 바로 사용할 수 있게 된 것을 볼 수 있다.

04 Unit Test – JUnit 5 (What is JUnit 5?)

➢ **소개**

자바 언어(Java)의 단위 테스트를 위한 프레임워크

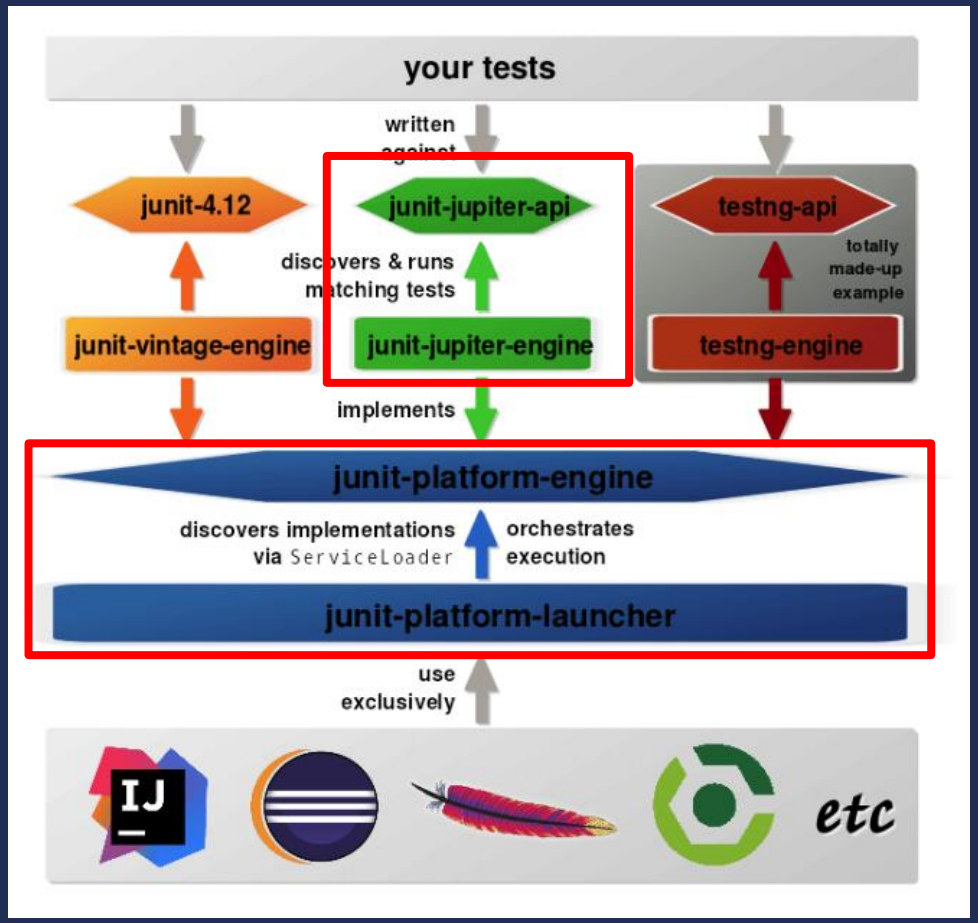
➢ **기능**

- Assertion 으로 테스트케이스의 수행 결과를 판별하여 passed or failed 로 표시
- Annotation(@) 지원

➢ **효과**

- System.out.print로 번거롭게 디버깅하지 않아도 된다.
- 프로그램 테스트 시 걸리는 시간을 관리할 수 있다.

04 Unit Test – JUnit 5 (What is JUnit 5?)



JUnit 5의 경우,

다양한 module로 구성되어 있음.

1) Junit – Jupiter – api

→ test code 작성 module

2) Junit – Jupiter – engine

→ test code 실행 module

3) Junit – platform – engine

Junit – platform – launcher

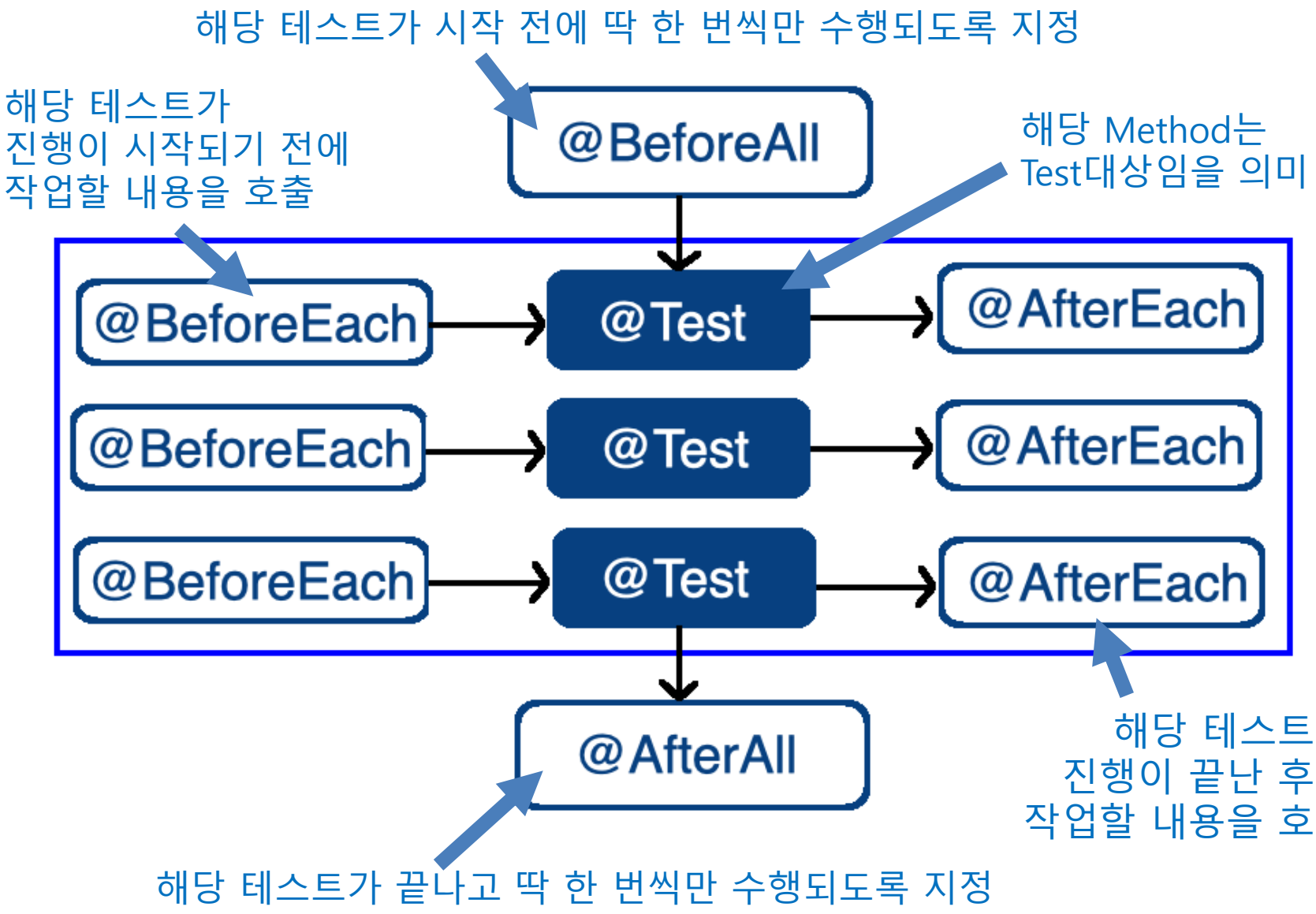
→ test engine을 통해 test를 발견,

실행 및 결과 보고 module

04 Unit Test – JUnit 5 (Assertions)

Method	내용
assertArrayEquals(a,b)	배열 a와b가 일치함을 확인한다.
assertEquals(a,b)	객체 a와b의 값이 같은지 확인한다.
assertSame(a,b)	객체 a와b가 같은 객체임을 확인한다.
assertTrue(a)	a가 참인지 확인한다.
assertFalse(a)	a가 거짓인지 확인한다.
assertNotNull(a)	a객체가 Null이 아님을 확인한다.

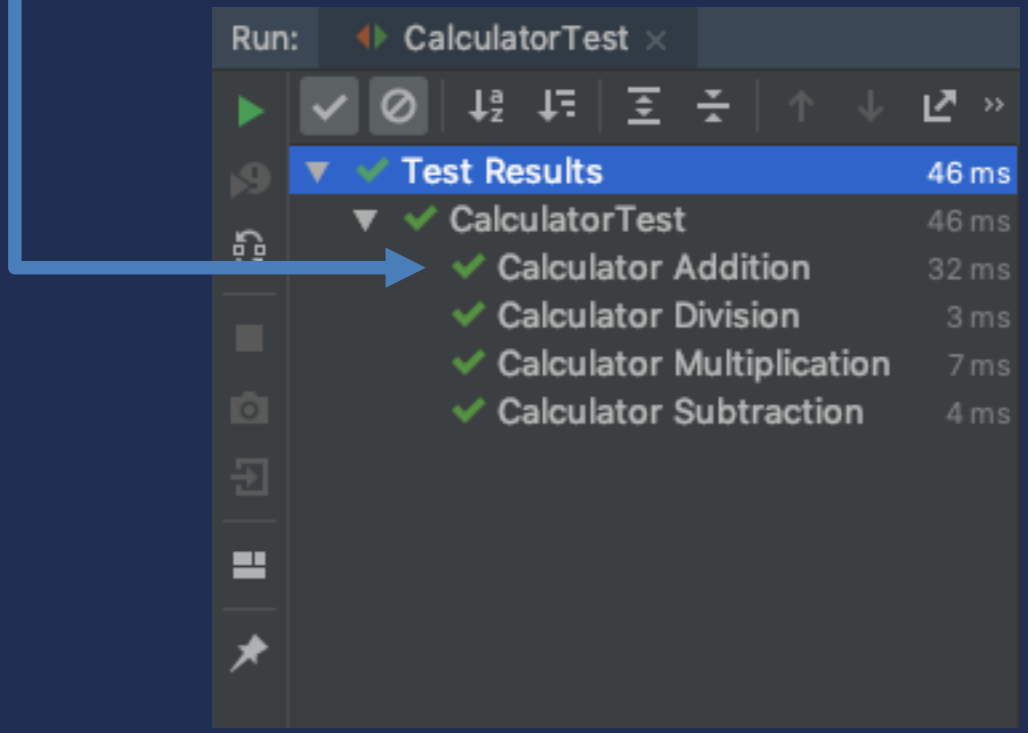
더 많은 Assert클래스의 method들 참고 :
http://junit.sourceforge.net/javadoc/org/junit/Assert.html


해당 테스트가 시작 전에 딱 한 번씩만 수행되도록 지정
해당 테스트가
진행이 시작되기 전에
작업할 내용을 호출
@BeforeAll
해당 Method는
Test대상임을 의미
@BeforeEach
@Test
@AfterEach
@BeforeEach
@Test
@AfterEach
@BeforeEach
@Test
@AfterEach
@AfterAll
해당 테스트가
진행이 끝난 후에
작업할 내용을 호출
해당 테스트가 끝나고 딱 한 번씩만 수행되도록 지정

04 Unit Test – JUnit 5 (Annotation ②)

@DisplayName : 테스트 클래스나 메서드의 표시 이름 지정

ex) @DisplayName("Calculator Addition")

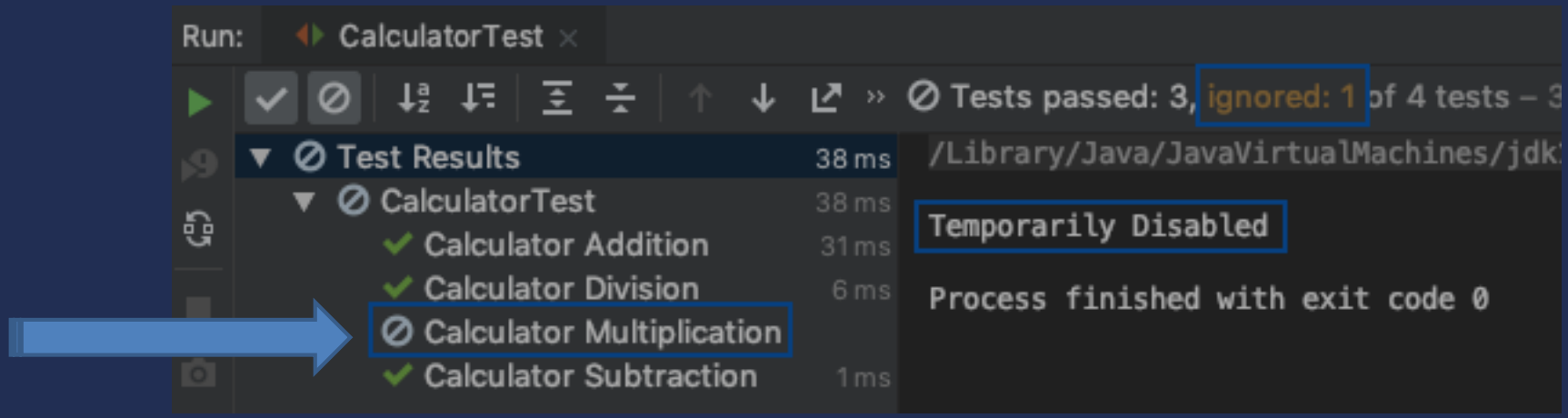


04 Unit Test – JUnit 5 (Annotation ③)

@Disabled: 테스트를 수행하지 않고 건너뜀

ex)

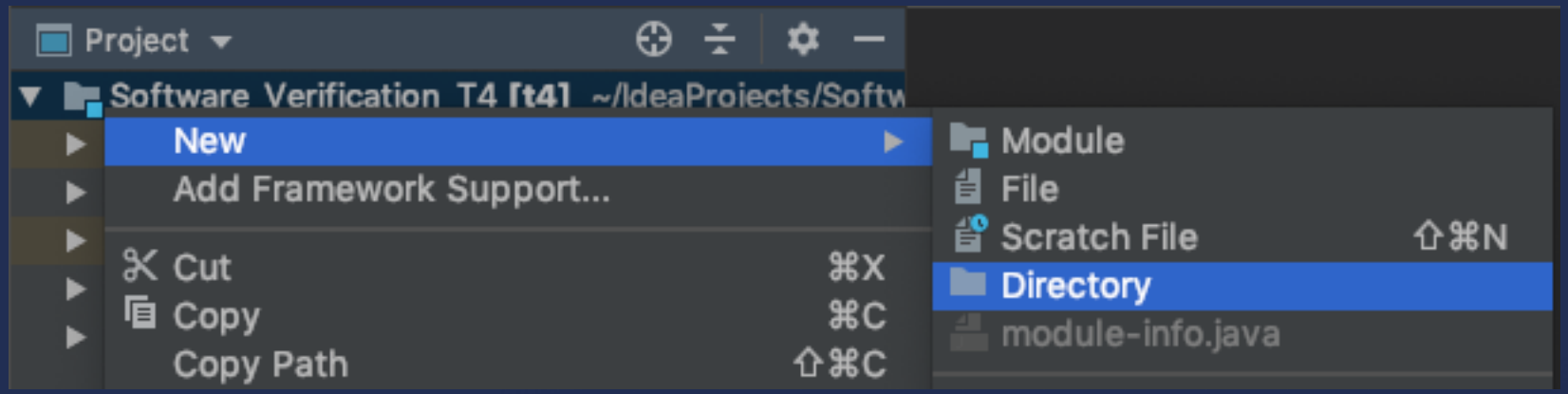
```
@Test
@DisplayName("Calculator Multiplication")
@Disabled("Temporarily Disabled")
void mul() {
    int a = 7;
    int b = 3;
    int expected = a * b;
    Assertions.assertEquals(expected, calculator.mul(a, b));
}
```

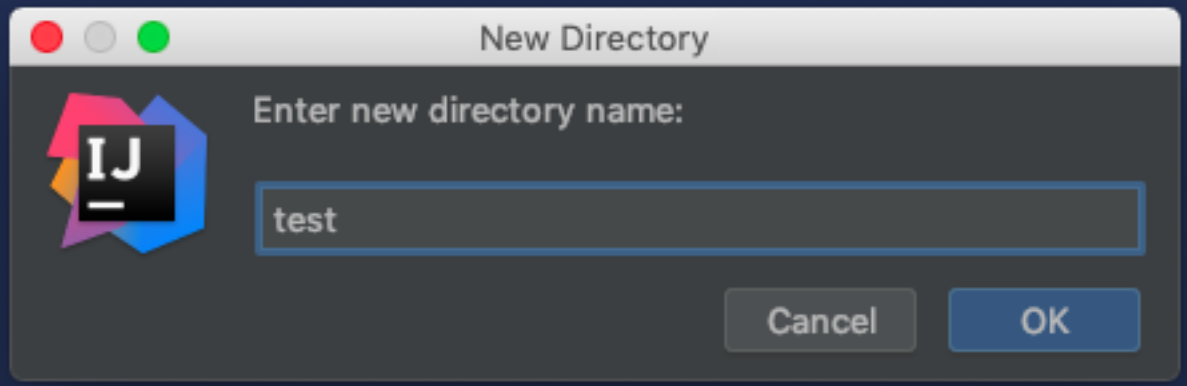


04 Unit Test – JUnit 5 (Test Environment ①)

1) IntelliJ 에서 테스트 환경을 구축할 Project 오른쪽 클릭

→ New → Directory 로 테스트 소스 폴더를 만든다.



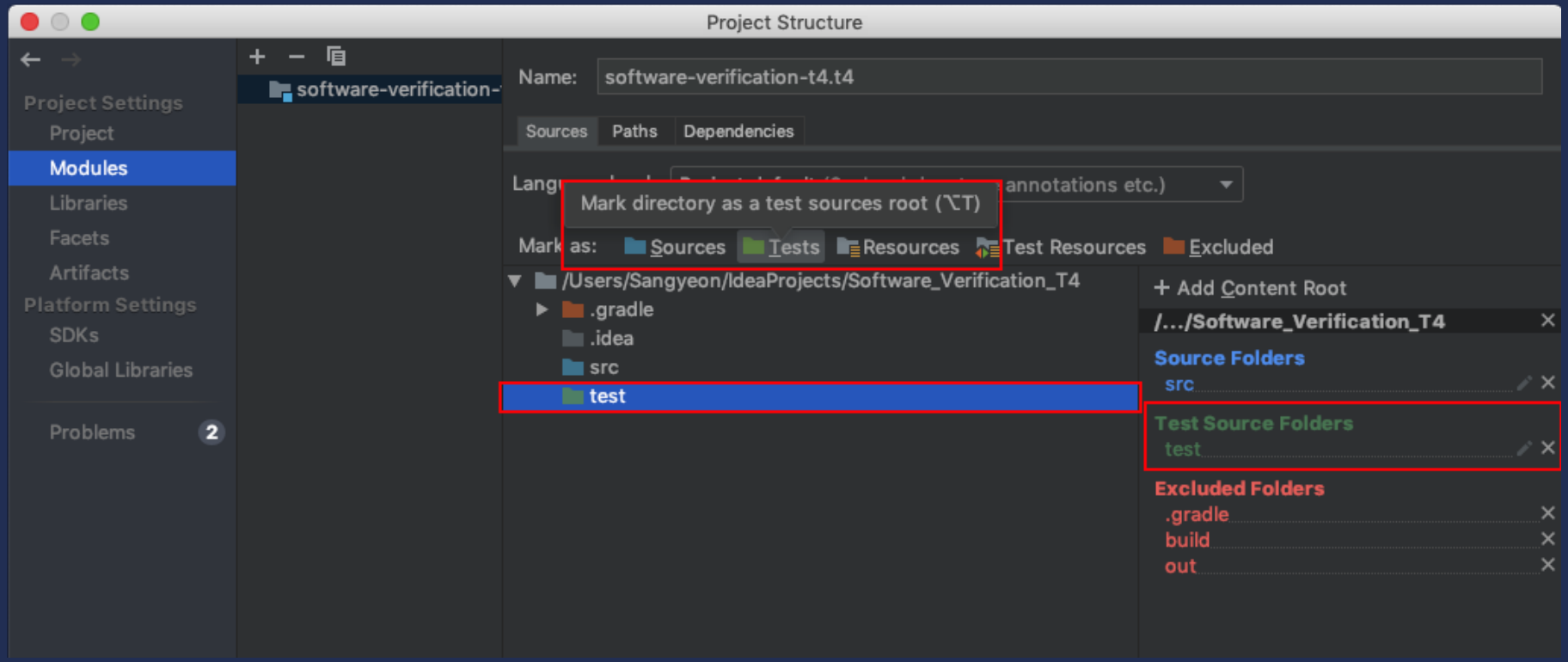


04 Unit Test – JUnit 5 (Test Environment ②)

2) Project Structure → Modules → Sources 에서

1)에서 만든 테스트 소스 폴더를 Test Sources Root로 지정한다.

제대로 되었으면 다음과 같이 초록색 폴더로 설정된다.

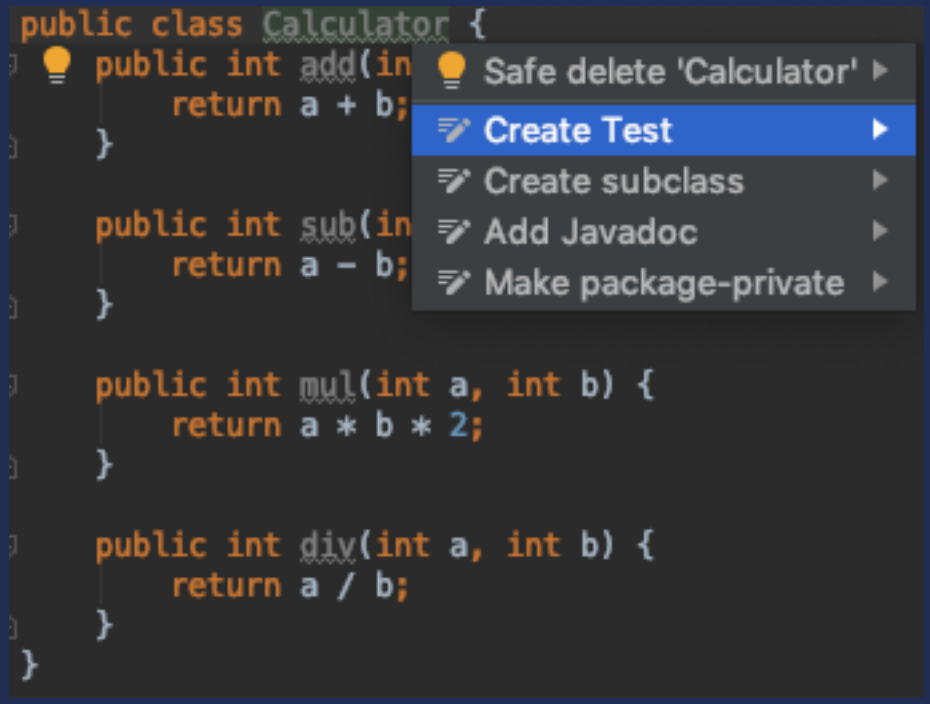


04 Unit Test – JUnit 5 (Test Code Creation ①)

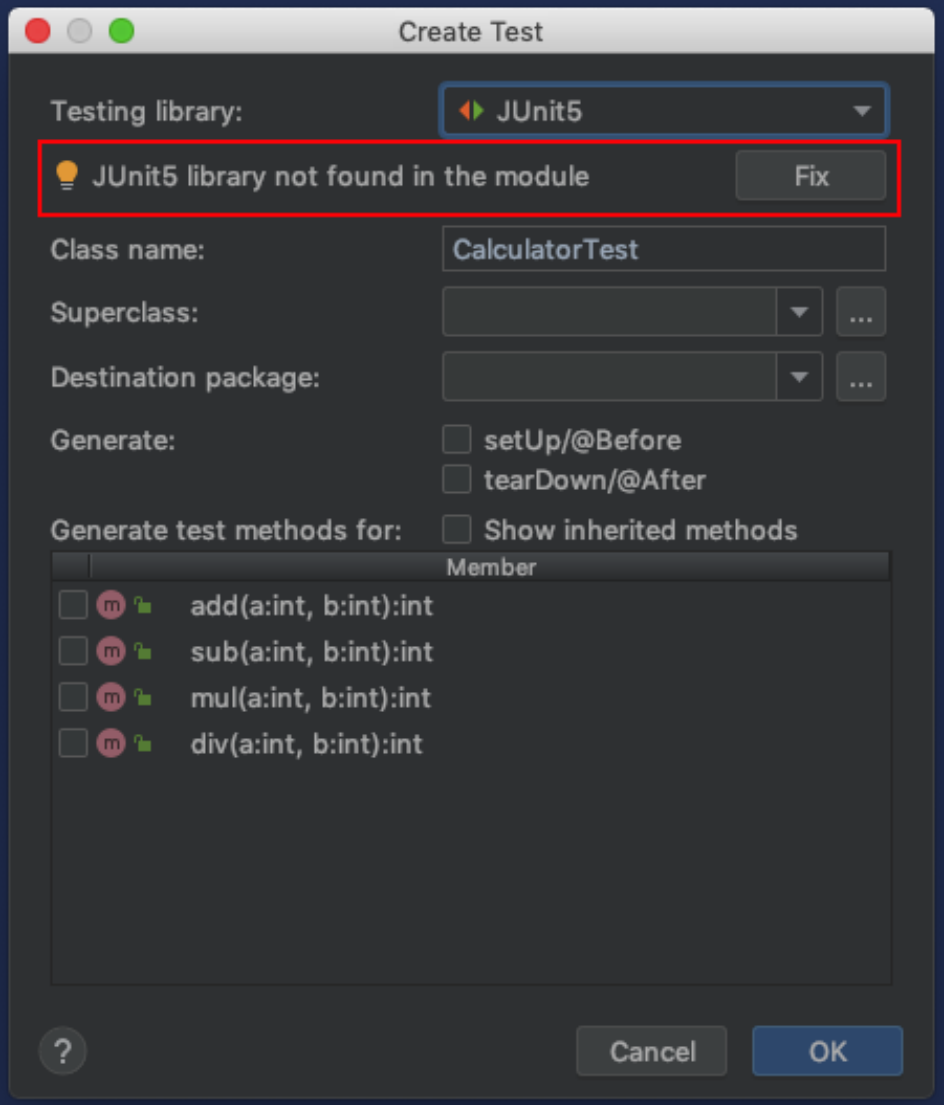
Calculator.java

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int sub(int a, int b) {
        return a - b;
    }

    public int mul(int a, int b) {
        return a * b * 2;
    }

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

3) 테스트를 실행할 Class Name에 커서를 올린 후 Alt + Enter를 누르면 그림과 같이 메뉴가 등장하는데, 거기서 Create Test를 누른다.

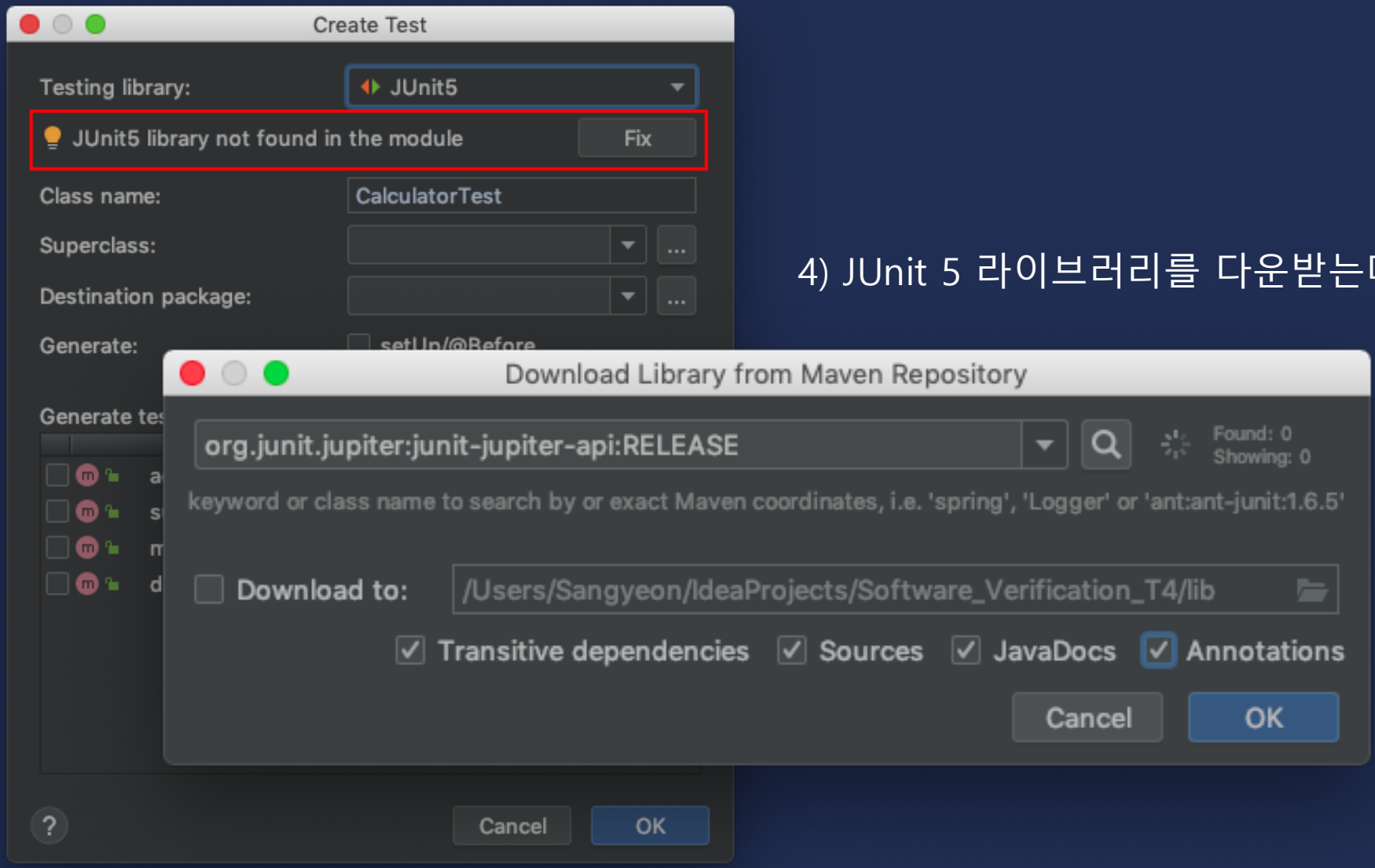


04 Unit Test – JUnit 5 (Test Code Creation ②)



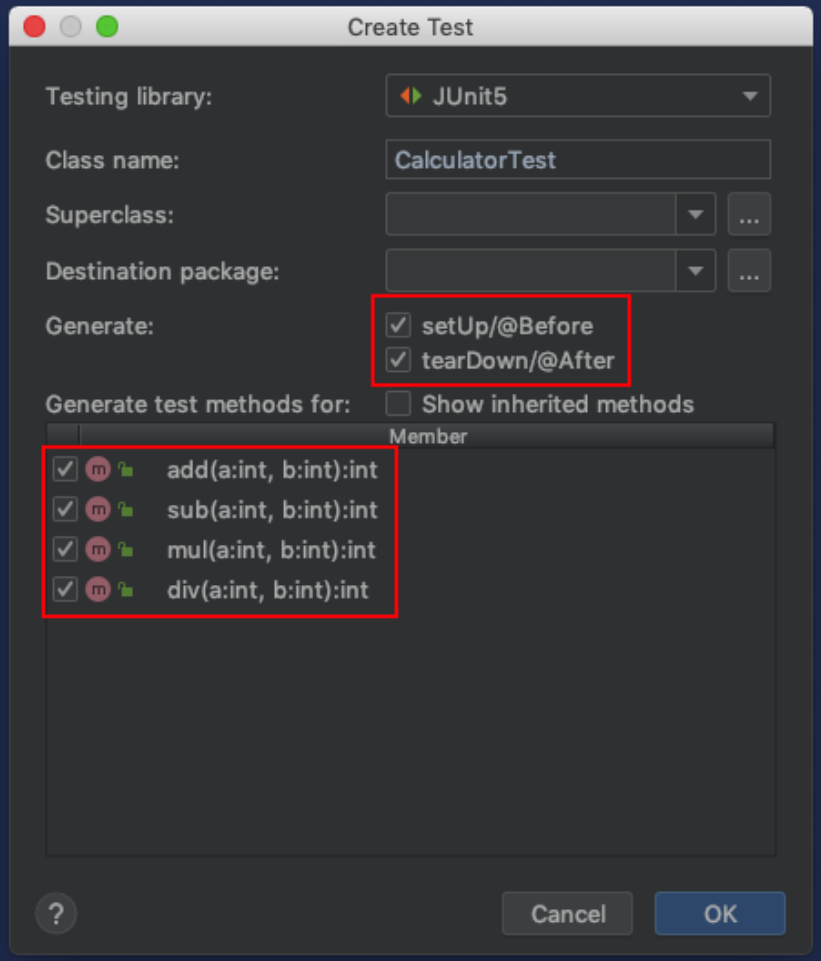
4) JUnit 5 라이브러리를 다운받는다.

04 Unit Test – JUnit 5 (Test Code Creation ②)



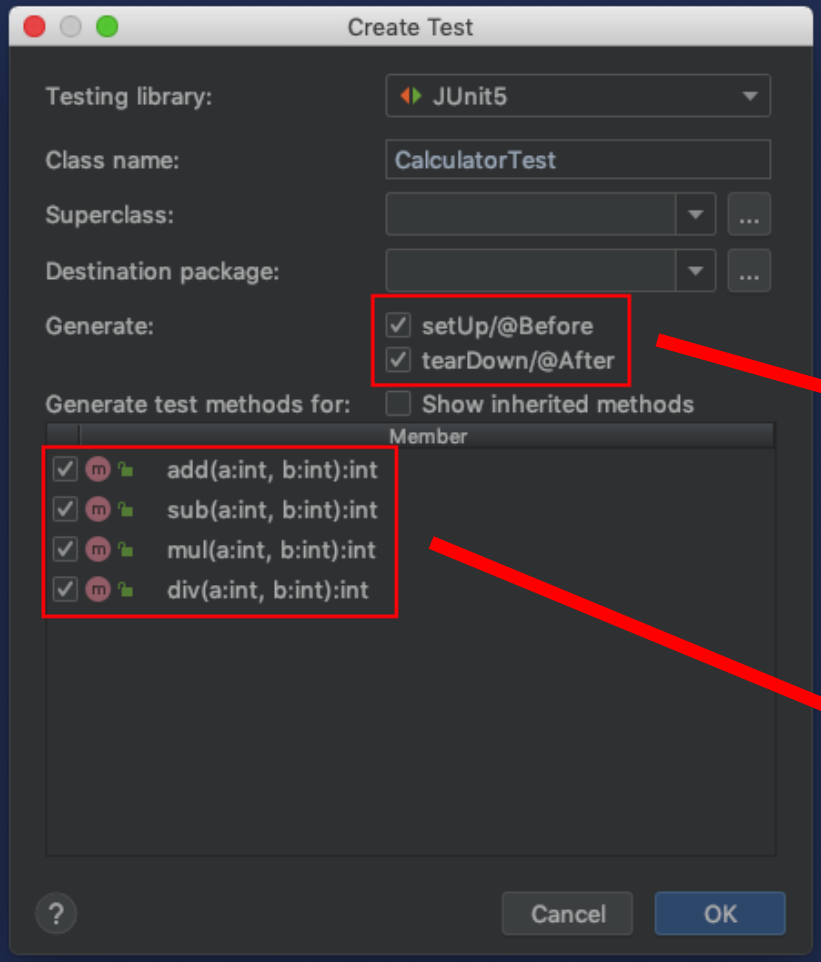
4) JUnit 5 라이브러리를 다운받는다.

04 Unit Test – JUnit 5 (Test Code Creation ③)



5) 테스트를 실행할 메소드를 선택하고 OK버튼을 누른다.

04 Unit Test – JUnit 5 (Test Code Creation ③)



Calculator.java | CalculatorTest.java

```
1  import org.junit.jupiter.api.AfterEach;
2  import org.junit.jupiter.api.BeforeEach;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.*;
6
7  class CalculatorTest {
8
9      @BeforeEach
10     void setUp() {
11     }
12
13     @AfterEach
14     void tearDown() {
15     }
16
17     @Test
18     void add() {
19     }
20
22     @Test
23     void sub() {
24
25     @Test
26     void mul() {
27     }
28
29     @Test
30     void div() {
31     }
32 }
```

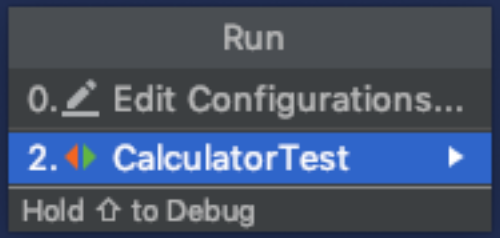
04 Unit Test – JUnit 5 (Test Code Writing)

6) Assert구문을 이용하여
테스트 메소드를 작성한다.

-> 해당 예제에서는
assertEquals(expected, actual)로
테스트 대상 메소드의
실행 결과(actual)가
예상 값(expected)과 같은지 확인

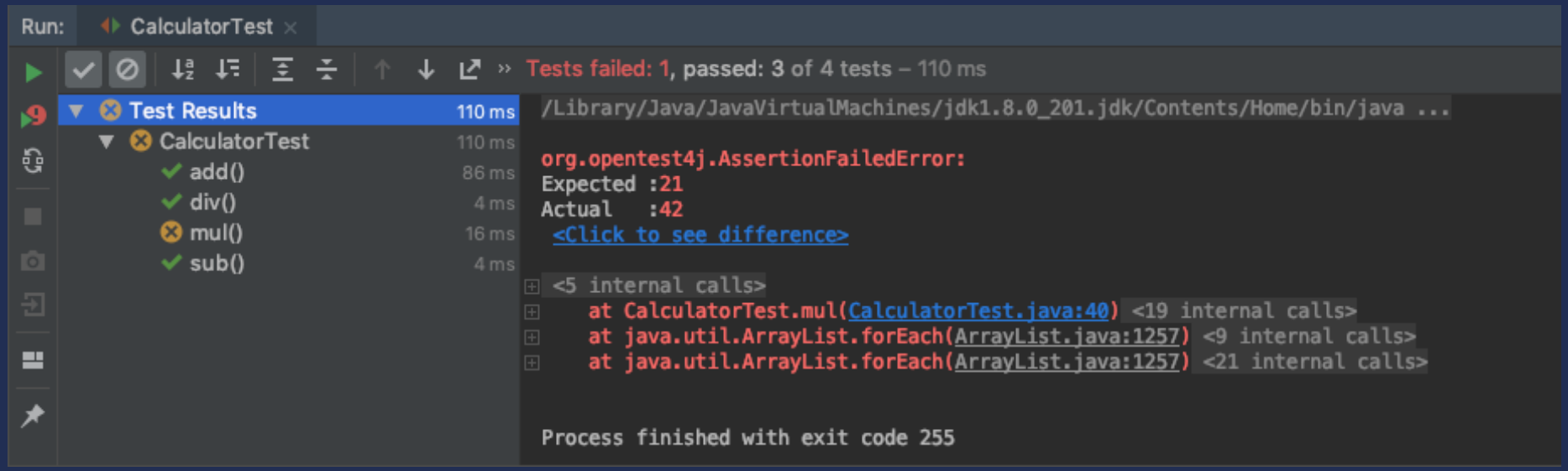
```java
class CalculatorTest {

    private Calculator calculator;

    @BeforeEach
    void setUp() {
        this.calculator = new Calculator();
    }

    @AfterEach
    void tearDown() {
    }

    @Test
    void add() {
        int a = 7;
        int b = 3;
        int expected = a + b;
        Assertions.assertEquals(expected, calculator.add(a, b));
    }

    @Test
    void sub() {
        int a = 7;
        int b = 3;
        int expected = a - b;
        Assertions.assertEquals(expected, calculator.sub(a, b));
    }

    @Test
    void mul() {
        int a = 7;
        int b = 3;
        int expected = a * b;
        Assertions.assertEquals(expected, calculator.mul(a, b));
    }

    @Test
    void div() {
        int a = 7;
        int b = 3;
        int expected = a / b;
        Assertions.assertEquals(expected, calculator.div(a, b));
    }
}
```

04 Unit Test – JUnit 5 (Test Code Execution)

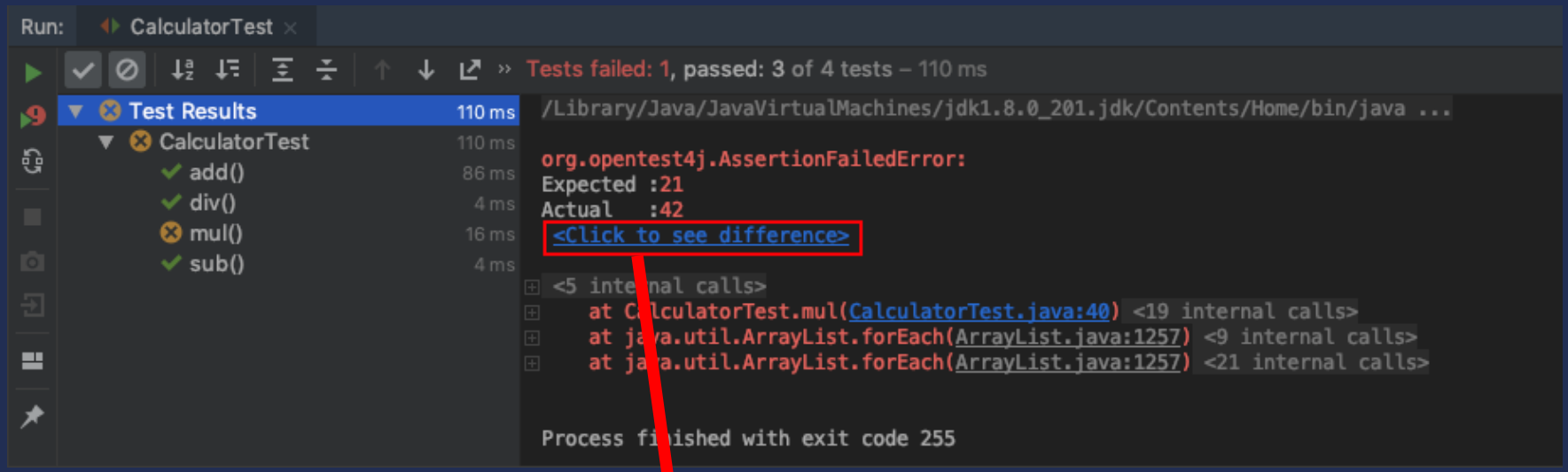


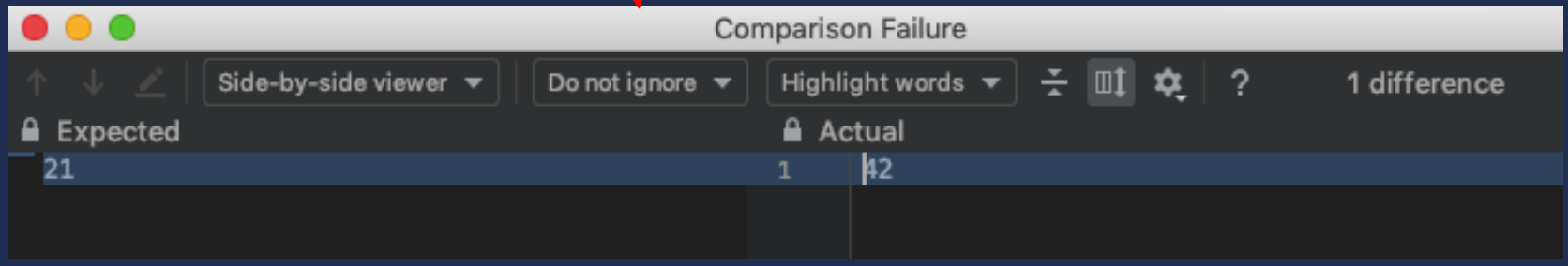
7) 테스트 소스를 실행한다.

8) 테스트 결과를 확인한다.

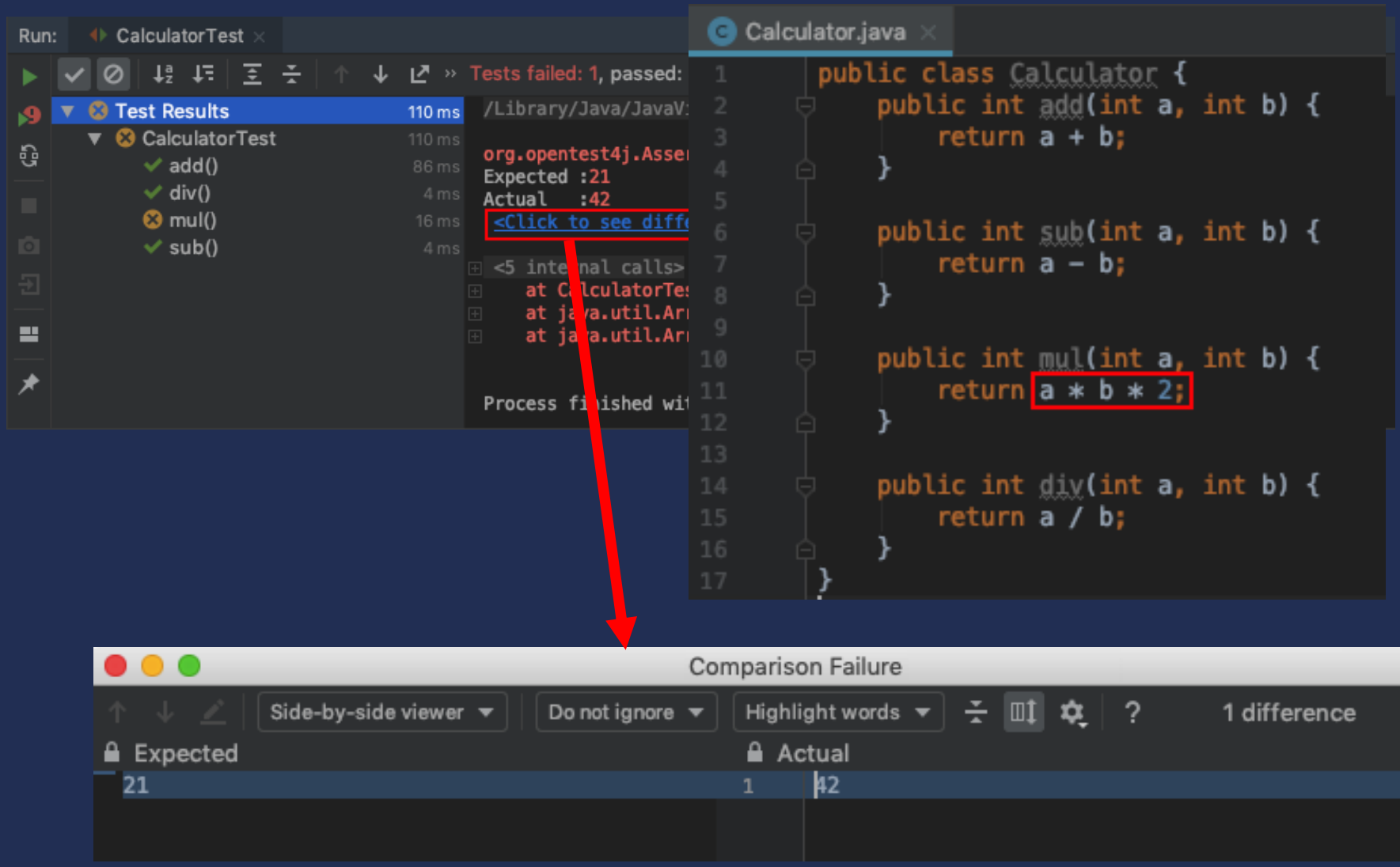


04 Unit Test – JUnit 5 (Checking the Test result ①)





04 Unit Test – JUnit 5 (Checking the Test result ②)



04 Unit Test – JUnit 5 (Checking the Test result ③)

Calculator.java

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int sub(int a, int b) {
        return a - b;
    }

    public int mul(int a, int b) {
        return a * b * 2;
    }

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

Calculator.java | CalculatorTest.java

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int sub(int a, int b) {
        return a - b;
    }

    public int mul(int a, int b) {
        return a * b;
    }

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

04 Unit Test – JUnit 5 (Checking the Test result ④)

Calculator.java

```
1   public class Calculator {
2       public int add(int a, int b) {
3           return a + b;
4       }
5
6       public int sub(int a, int b) {
7           return a - b;
8       }
9
10      public int mul(int a, int b) {
11          return a * b * 2;
12      }
13
14      public int div(int a, int b) {
15          return a / b;
16      }
17  }
```

Calculator.java | CalculatorTest.java

```
1   public class Calculator {
2       public int add(int a, int b) {
3           return a + b;
4       }
5
6       public int sub(int a, int b) {
7           return a - b;
8       }
9
10      public int mul(int a, int b) {
11          return a * b;
12      }
13
14      public int div(int a, int b) {
15          return a / b;
16      }
17  }
```

Run: CalculatorTest

✔ Tests passed: 4 of 4 tests – 78 ms

Test Results	78 ms
CalculatorTest	78 ms
add()	71 ms
div()	3 ms
mul()	3 ms
sub()	1 ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

05 CI (Continuous integration) - Jenkins

Jenkins

1) CI(Continuous Integration, 지속적 통합)를 지원해주는 도구

2) Nightly-build → Auto-testing build
(When webhooks are occurred…)

3) Plugin을 활용하여 다른 tool과 호환성 높음
→ project 관리 : Redmine
→ 형상 관리 : GitHub
→ test 결과 통보 : slack
→ static analysis : PMD, FindBugs, SonarQube

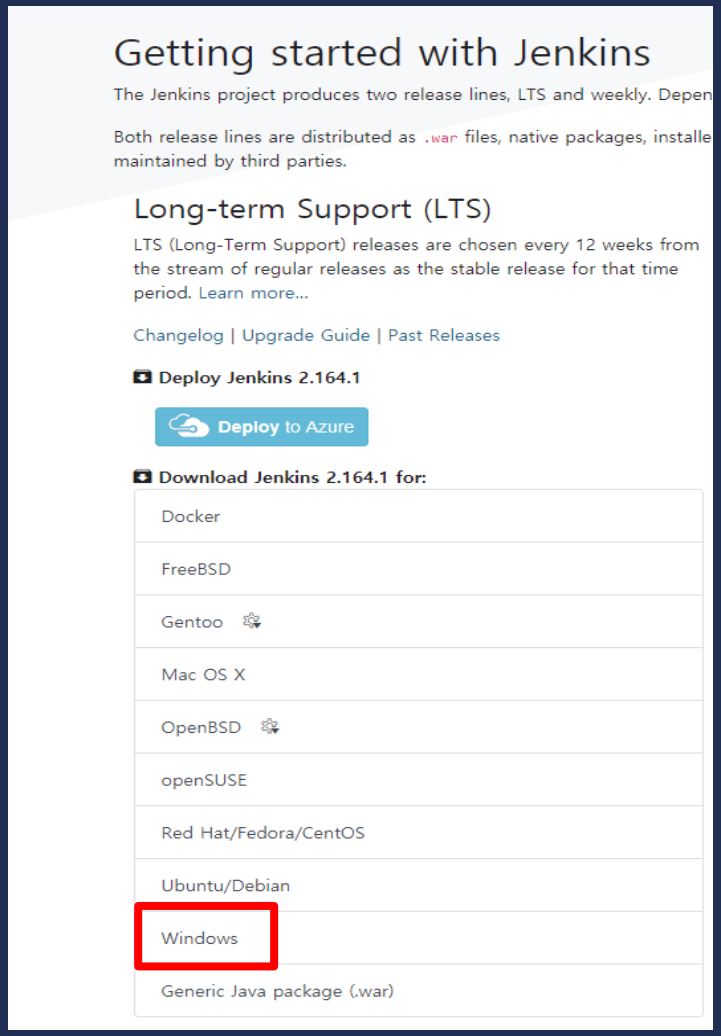
⇒ CTIP 환경 구축에 용이함.

05 CI (Continuous integration) – Jenkins(Installation)

Jenkins 다운로드 link

https://jenkins.io/

05 CI (Continuous integration) – Jenkins(Installation)



설치 방법은 2가지가 존재함.

1) War 확장자 파일로 받아서

tomcat server 위에서 구동 시키는 방법

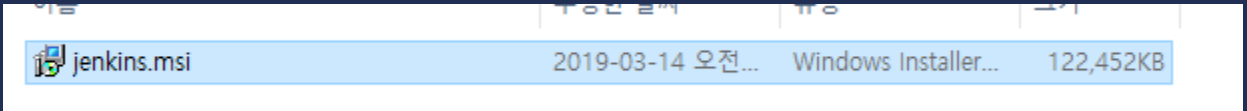
2) Window에 설치 파일 형식으로 받아서

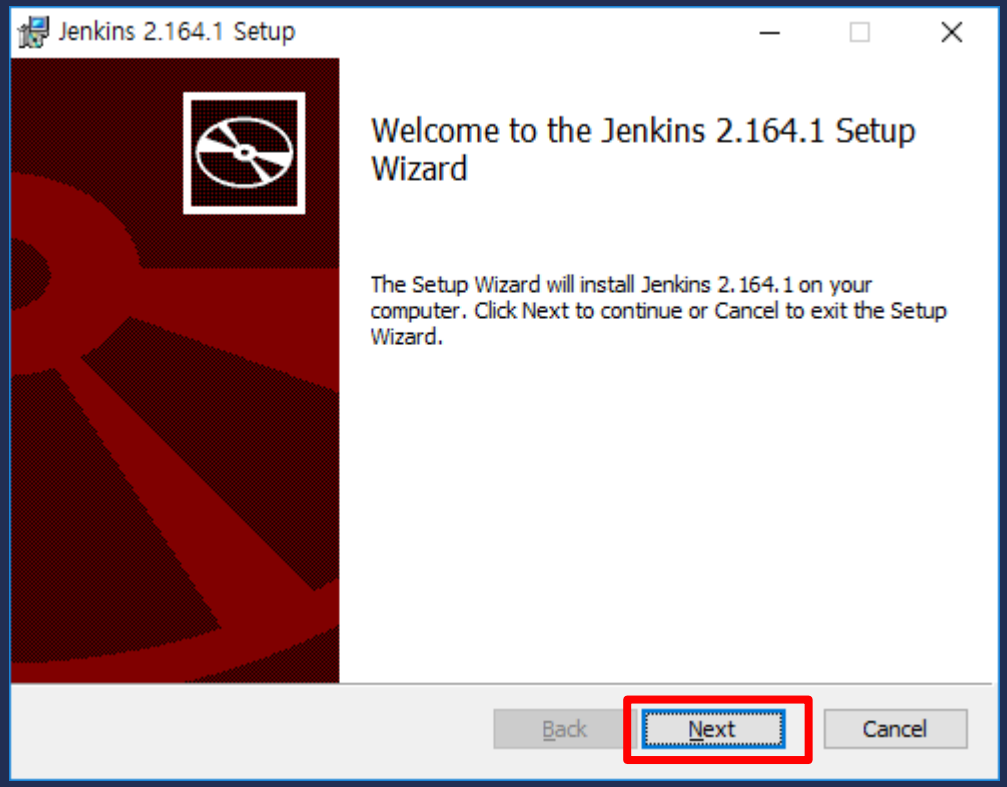
service 형태로 내장된 server를 가지고

구동 시키는 방법

여기서는 2) 방법으로 실행할 것.

05 CI (Continuous integration) – Jenkins(Installation)





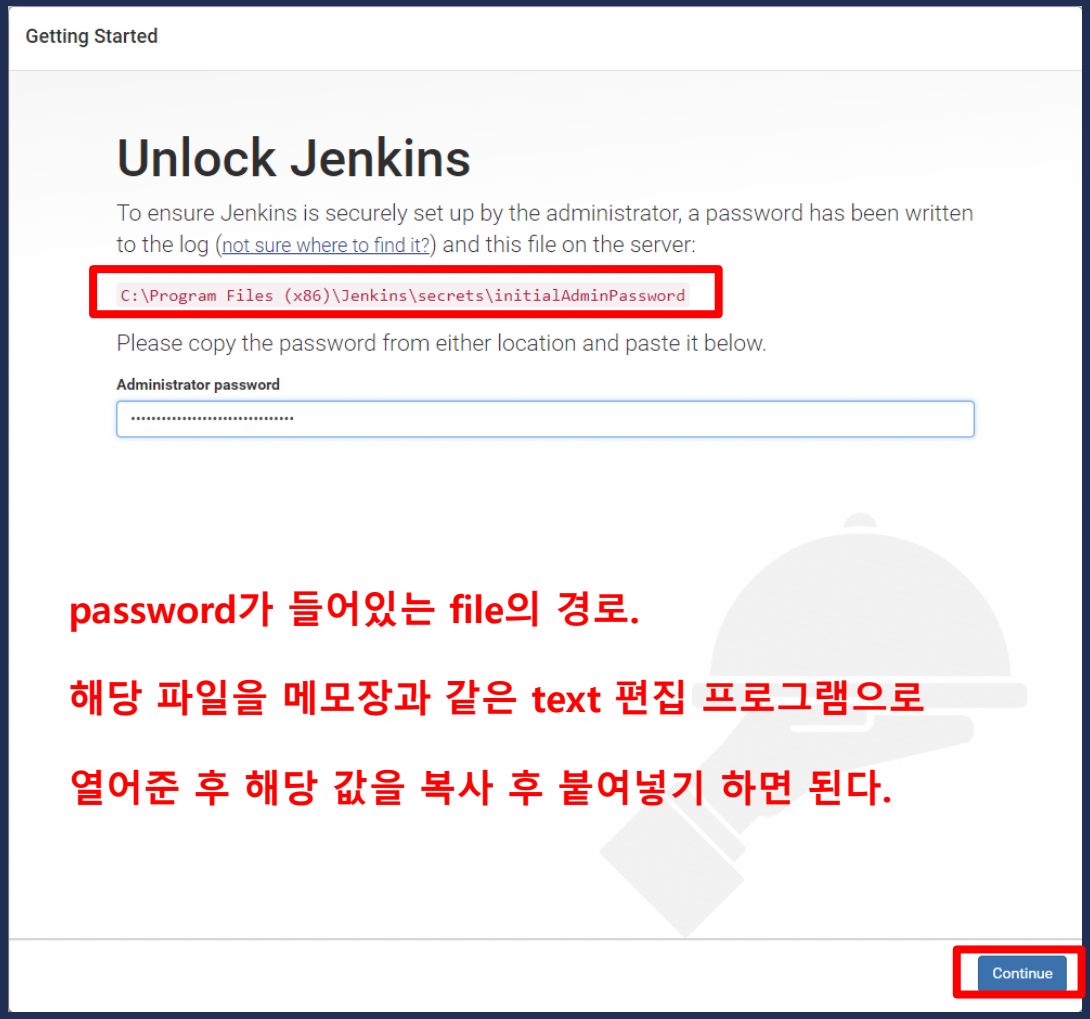
다운로드 받은 후 압축을 해제하면

msi 확장자 파일이 등장함.

실행하여 설치창으로 진입.

이후 계속 next를 눌러서 진행함.

05 CI (Continuous integration) – Jenkins(Installation)



password가 들어있는 file의 경로.

해당 파일을 메모장과 같은 text 편집 프로그램으로

열어준 후 해당 값을 복사 후 붙여넣기 하면 된다.

설치가 완료된 이후

잠시 기다리면 localhost:8080

Internet 창이 뜨면서

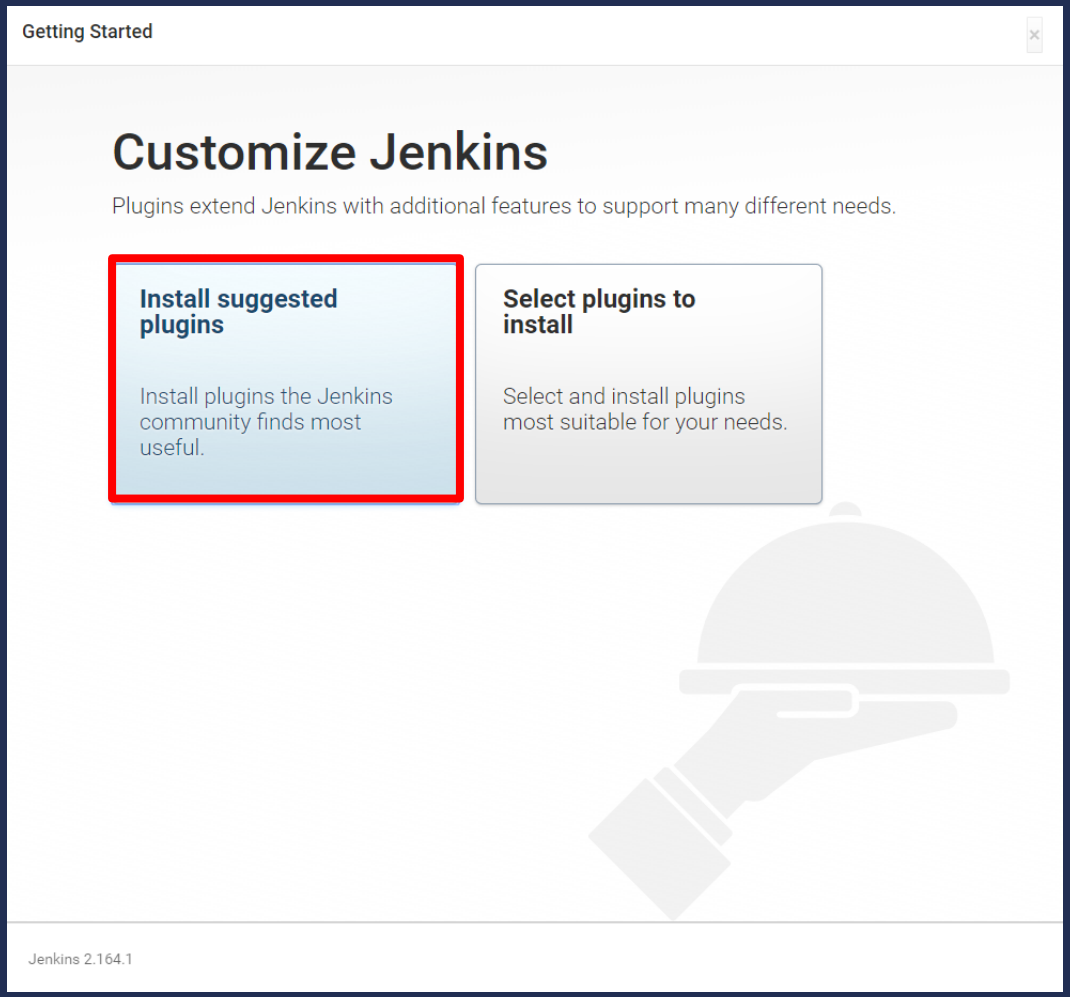
Jenkins 준비 중 상태가 된다.

모든 준비가 완료되면 Getting

Started 창으로 진입한다.

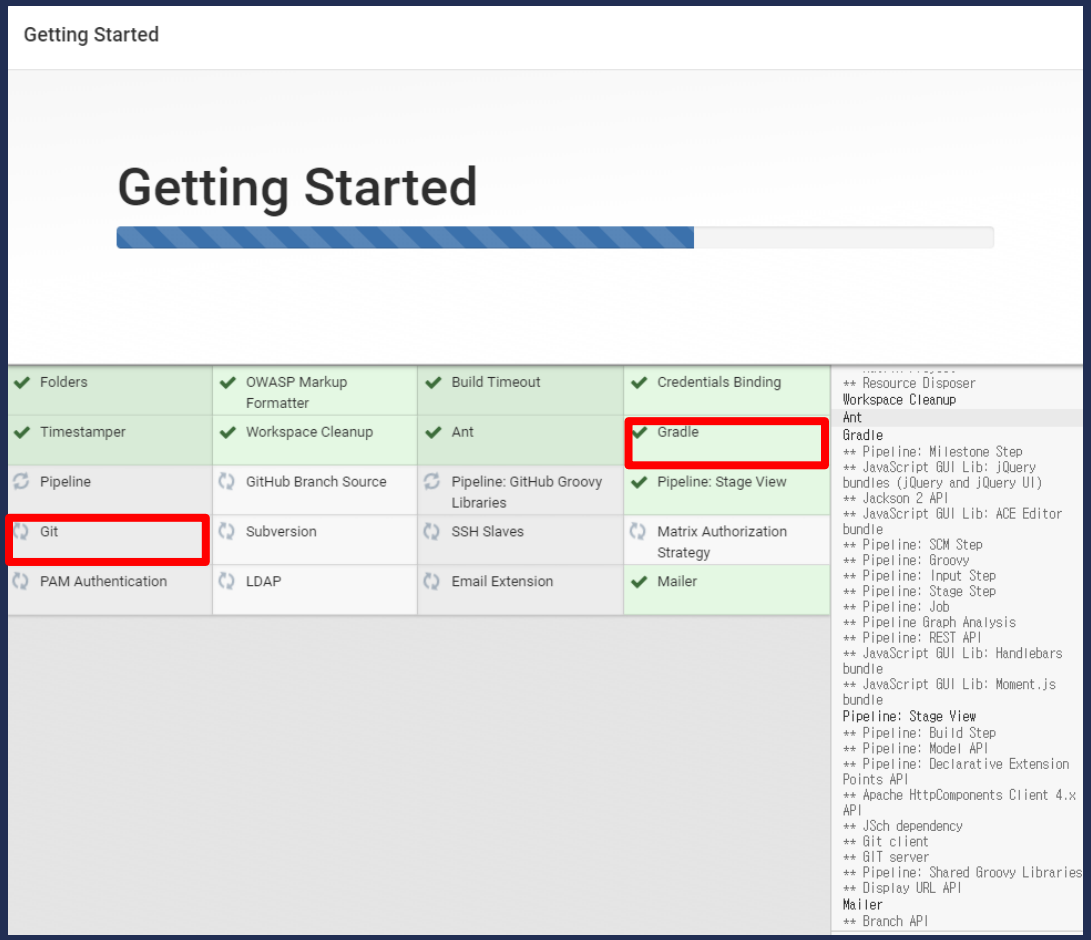
Password를 입력하고 continue.

05 CI (Continuous integration) – Jenkins(Installation)



Suggested plugins들을 설치한다.

05 CI (Continuous integration) – Jenkins(Installation)

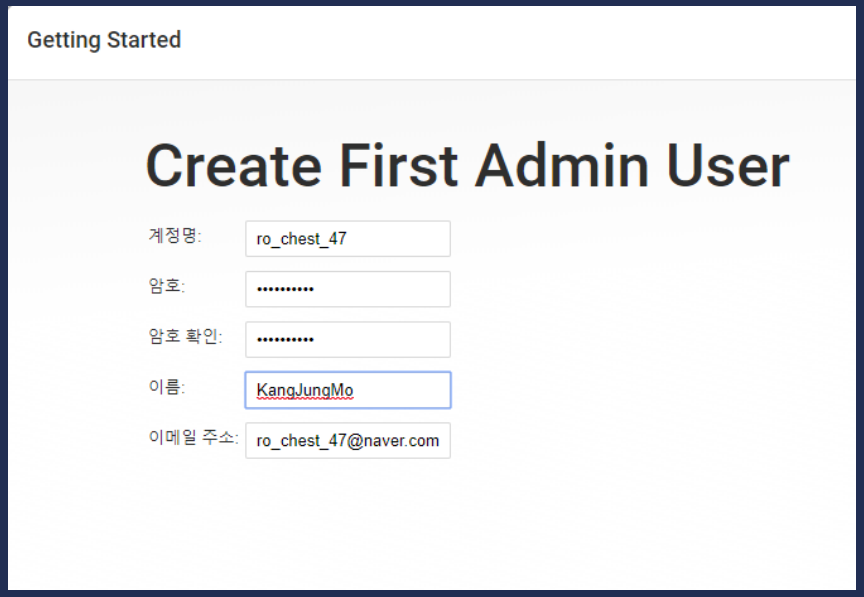


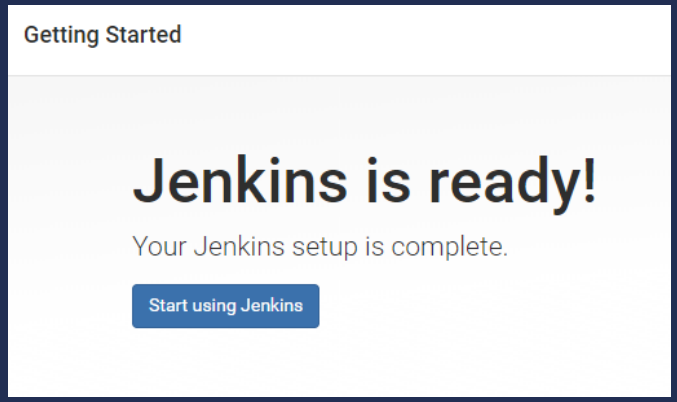
추후 연동하게 될 Git과

Gradle plugin이 설치되는 것을

확인할 수 있다.

05 CI (Continuous integration) – Jenkins(Installation)



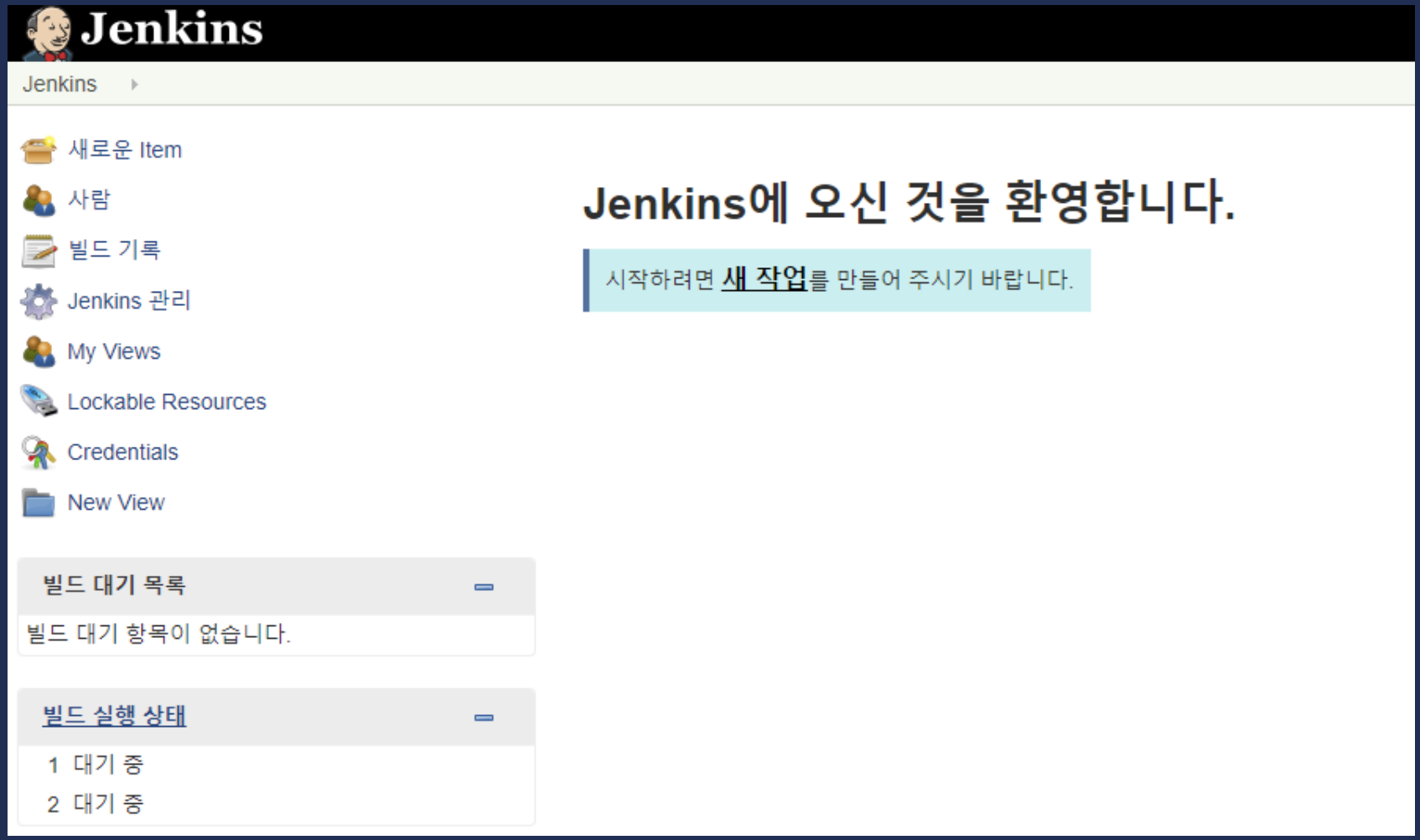


설치 완료 후 관리자 계정을

등록하고 Jenkins URL을 설정하면

Jenkins를 사용할 준비를 마친다.

05 CI (Continuous integration) – Jenkins(Installation)



05 CI (Continuous integration) – Jenkins

그러나.....

1. Localhost 이므로 git과 연동시킬 때 webhook 발생이 까다로움.
2. 나는 바로 접속할 수 있으나 외부에서 접속이 까다로움.

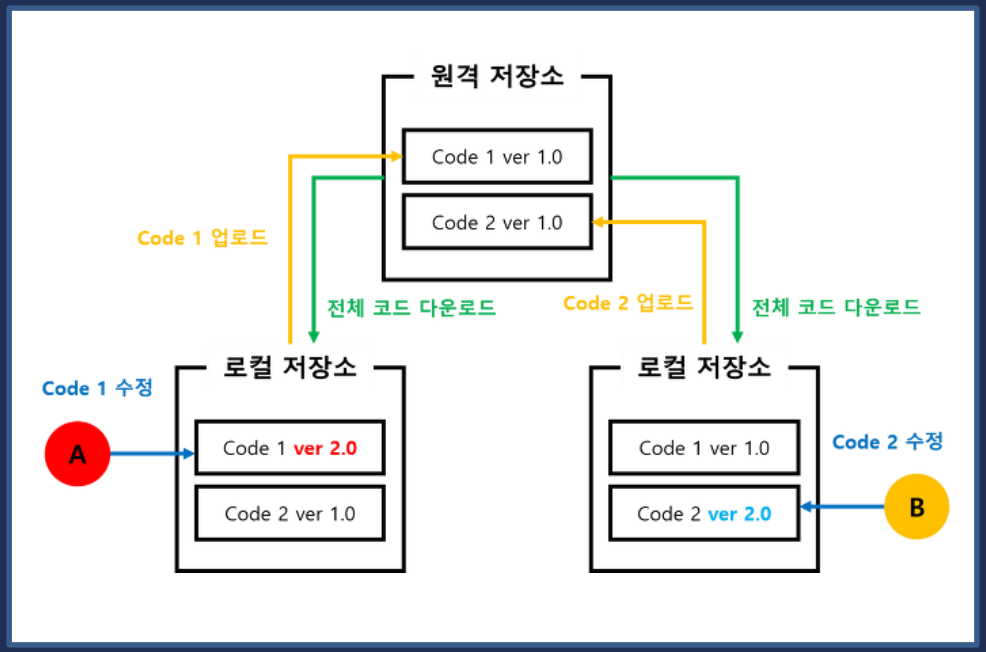
   다수와 협업하며 CI를 요구하는 project 상황에 적합한가?
3. 내 컴퓨터를 일종의 server 로 사용하는 것 ⇒ 내 컴퓨터가 동작하지 않을 때는?
4. 설정 변경 시 service 로 동작되므로 제어판 – 관리 도구 – 서비스에서

   계속 중지 – 재실행을 해야 함.

→ ngrok 을 이용하여 외부 접속이 가능하게 만들 수 있지만 근본적인 해결책은

되지 않는다고 판단함. ⇒ AWS EC2로 Jenkins server를 구축함.

06 CI (Continuous integration) – Git & GitHub



- 분산 버전 관리 시스템 (형상 관리)

→ workspace

실제 source code를 저장하는 폴더

→ local repository

commit 내역을 저장하고 있는

workspace 하위 .git 폴더

→ remote repository

네트워크상에 존재하는 repository

- SHA-1 Hash를 사용하여 data를 Hash object의 형태로 만들어 관리함

- Local에서도 편하게 작업이 가능함.

06 CI (Continuous integration) – Git & GitHub

- Git을 지원해주는 web hosting service.

- Git을 통해 다른 사람들과 협업할 때 필요한 remote repository를 제공 및 유지, 관리해줌.

- GUI를 통해 시각적으로 접근 가능

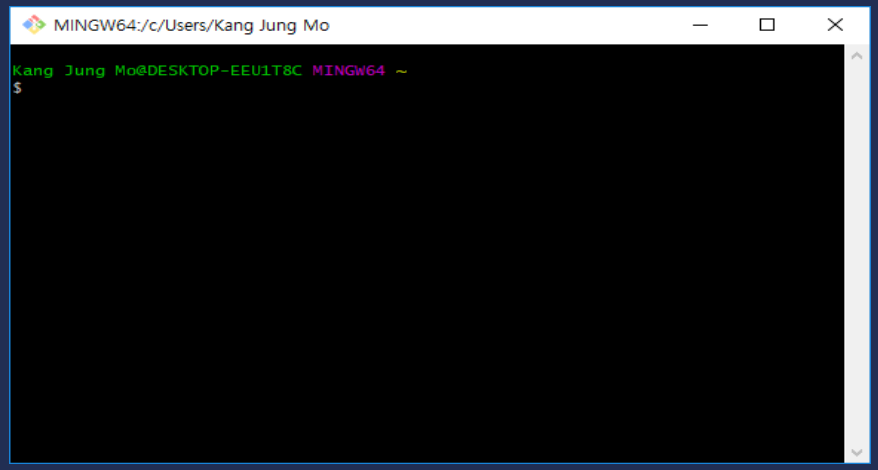
- 형상 관리 및 빌드 자동화를 위해 다양한 tool과 연동 가능
  ex) Jenkins, Travis CI

06 CI (Continuous integration) – Git & GitHub

https://git-scm.com/

Git download URL.

설치 파일을 받고 next를 계속 눌러 진행한다.

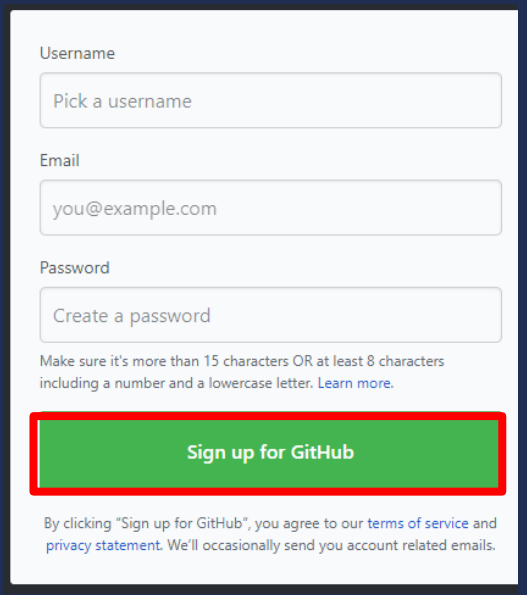


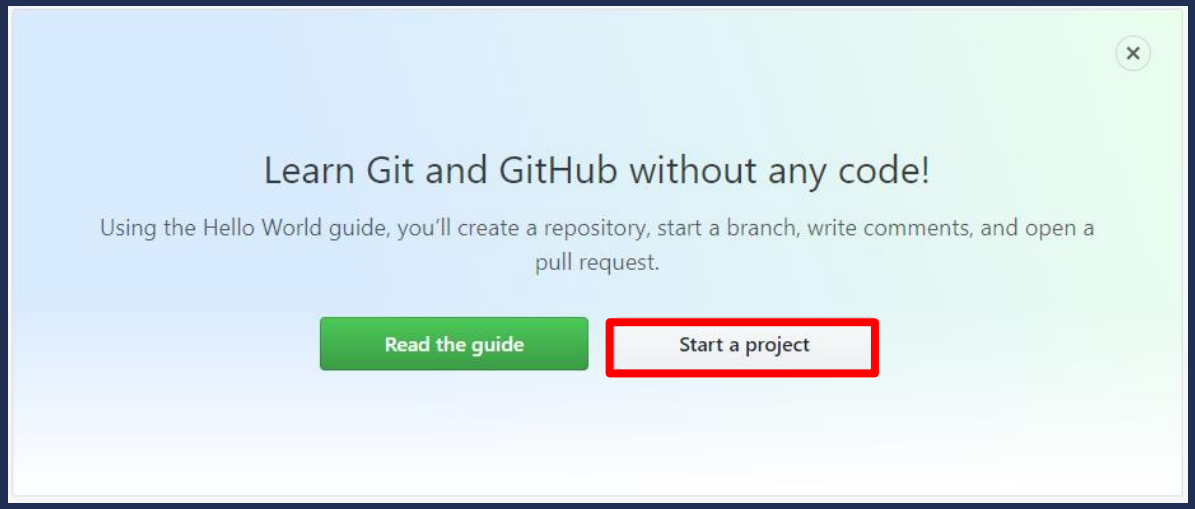
설치가 완료되면 window에서 리눅스처럼

각종 git 명령어를 사용할 수 있는

Git bash를 실행할 수 있다.

06 CI (Continuous integration) – Git & GitHub



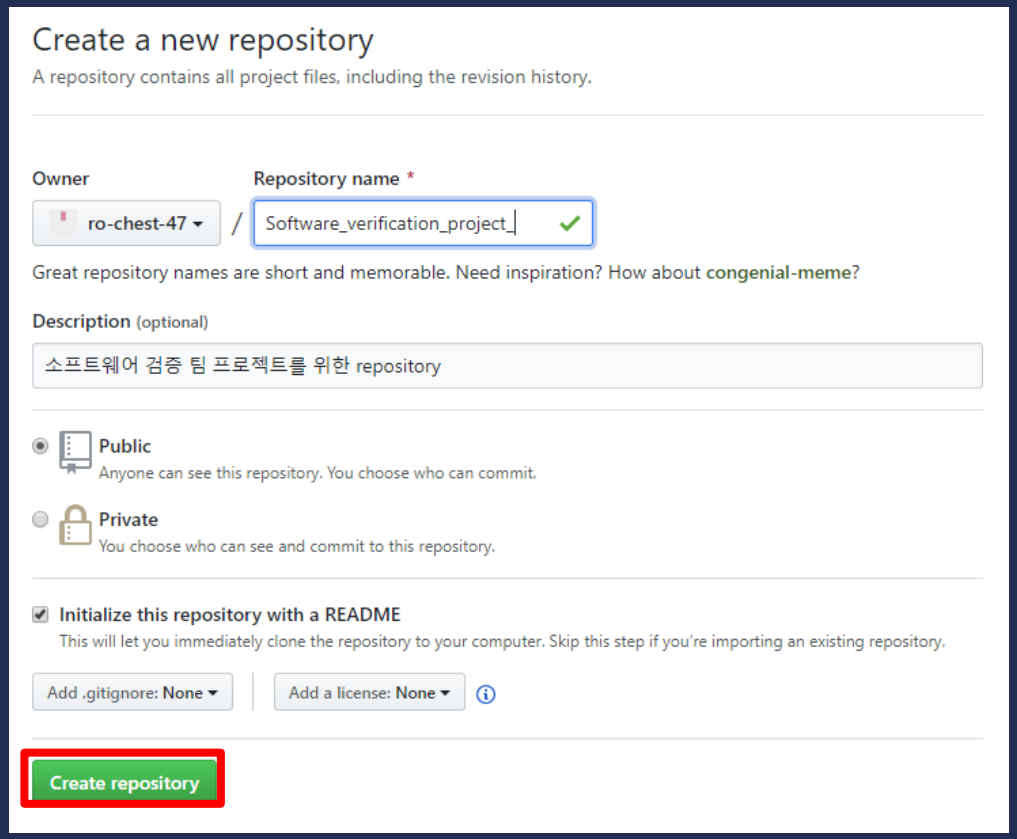


Git을 설치한 후에는 원격 저장소를 만들기 위해

GitHub에 가입한다. 가입하고 나서 project(repository)를 생성한다.

https://github.com/

06 CI (Continuous integration) – Git & GitHub



저장소 이름과,

이 저장소가 무엇을 위한 것인지

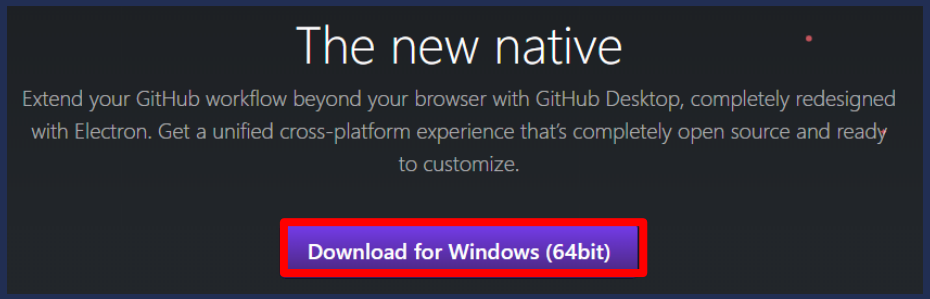
Description 등을 서술하고,

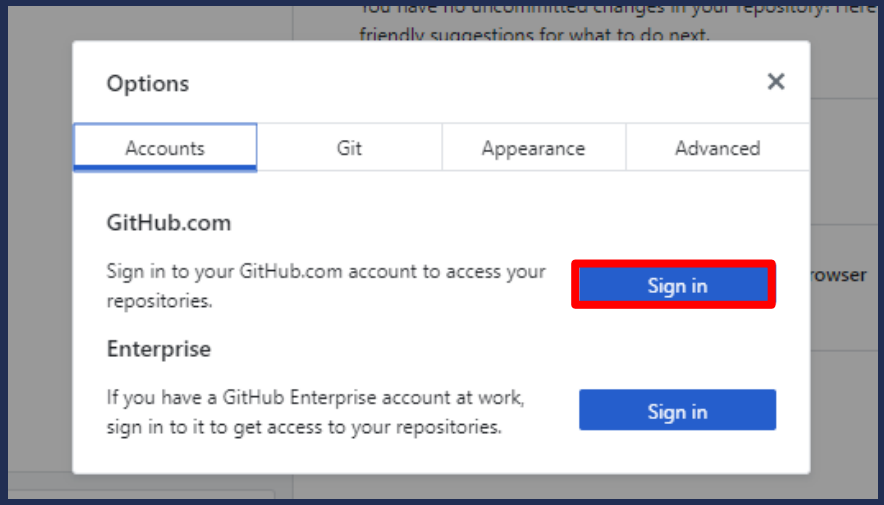
private/public 옵션을 선택한다.

Create repository로 원격 저장소를

생성한다.

06 CI (Continuous integration) – Git & GitHub



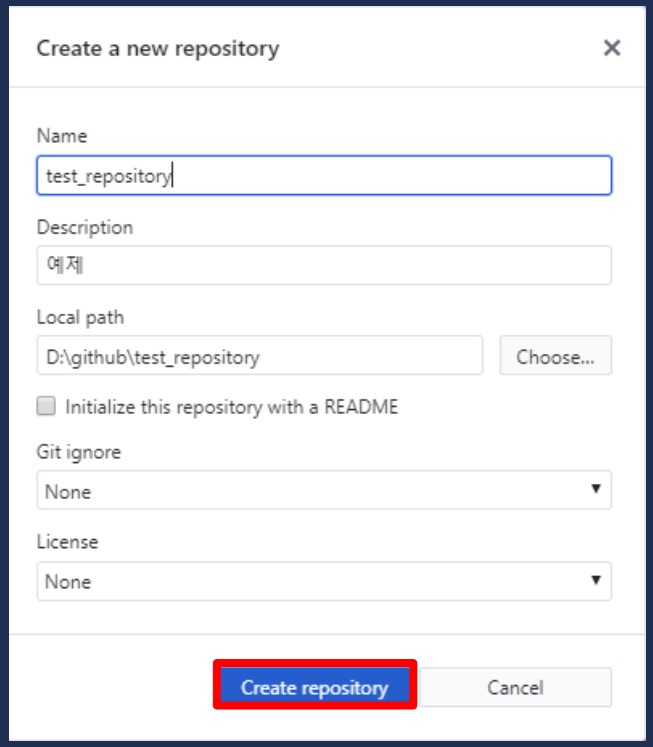


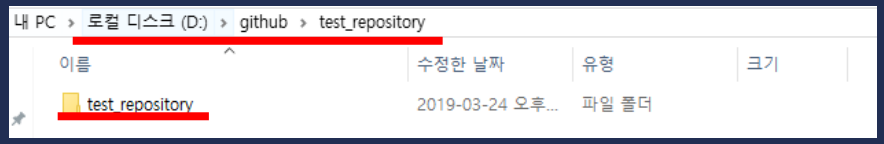
Git bash로도
commit, push 등의 작업을
진행할 수 있으나
GitHub Desktop 을 통해
더 편하게 진행할 수 있음.
https://desktop.github.com/

설치 파일을 다운로드 후 설치 시
GitHub Desktop이 실행됨.

File – option 으로 들어가서
GitHub id 로 Sign in을 진행한다.

06 CI (Continuous integration) – Git & GitHub





File – new repository 으로 들어간다.

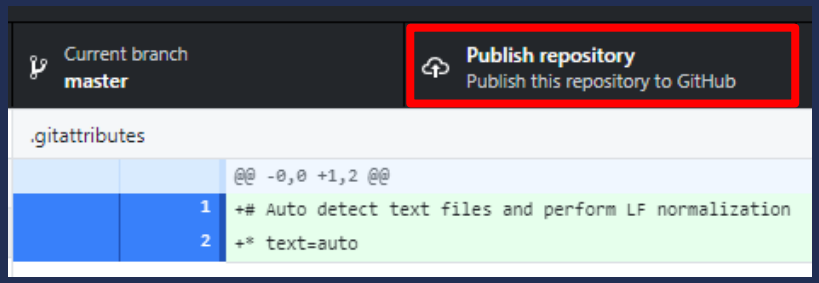
생성하고 싶은 Local repository 이름과

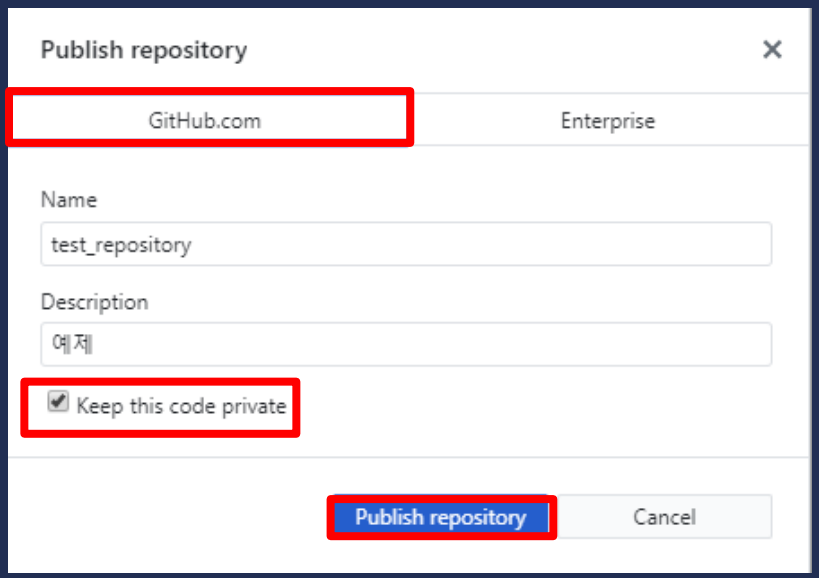
Description, Local repository 경로를 설정한다.

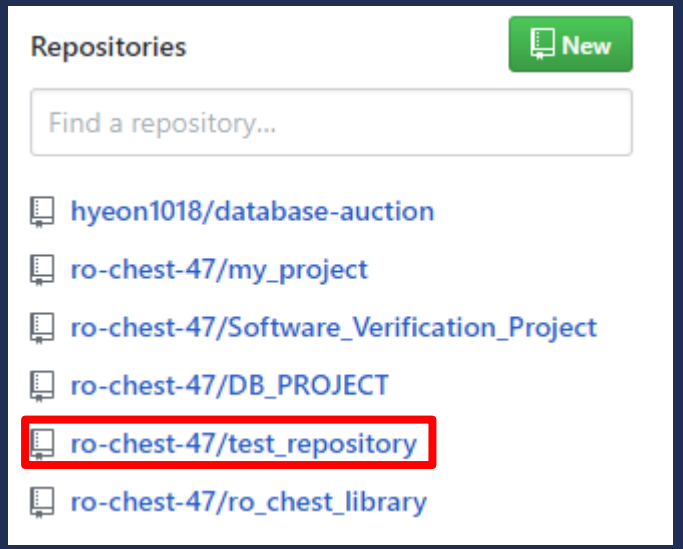
이후 해당 경로에 Local repository 가

생성된 것을 확인할 수 있다.

06 CI (Continuous integration) – Git & GitHub





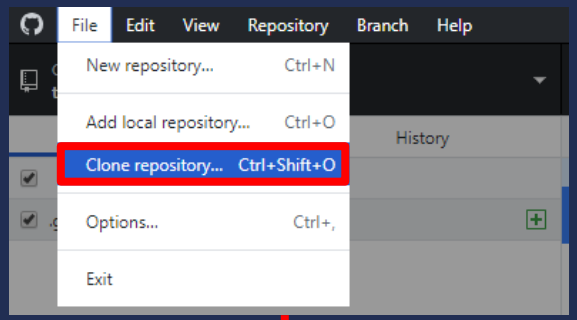


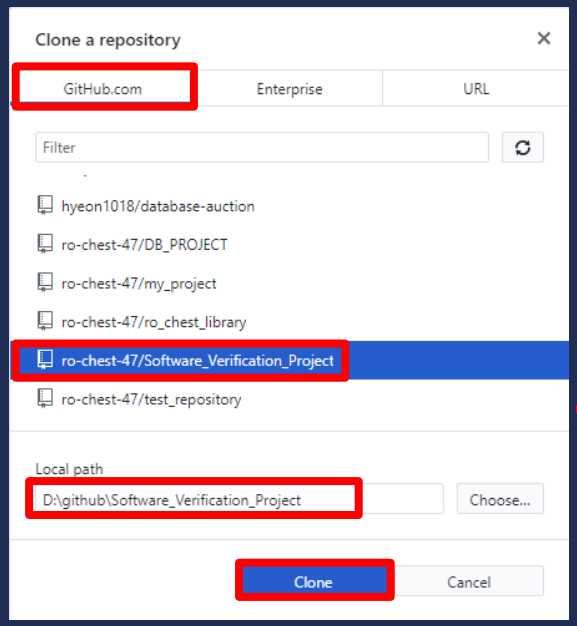
해당 Local repository를 GitHub의 Remote repository로 등록하기 위해 Publish repository를 누른다.

참고)

Keep this code private 체크 시 Private repository로 생성된다.

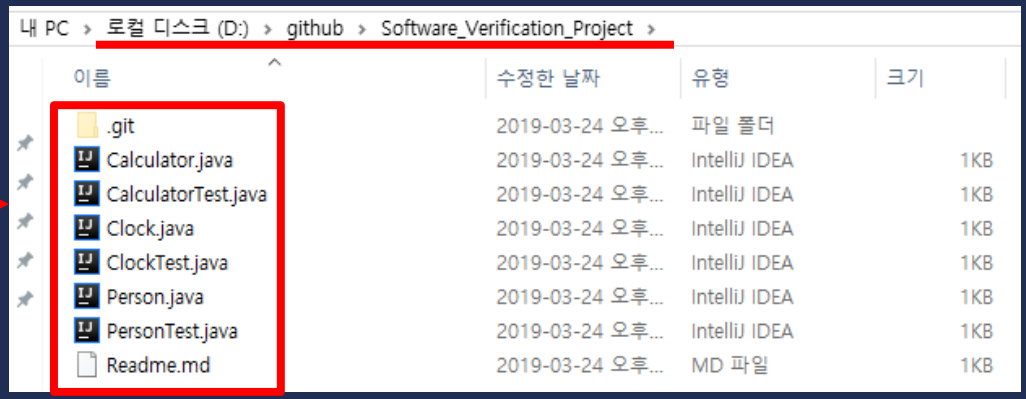
06 CI (Continuous integration) – Git & GitHub



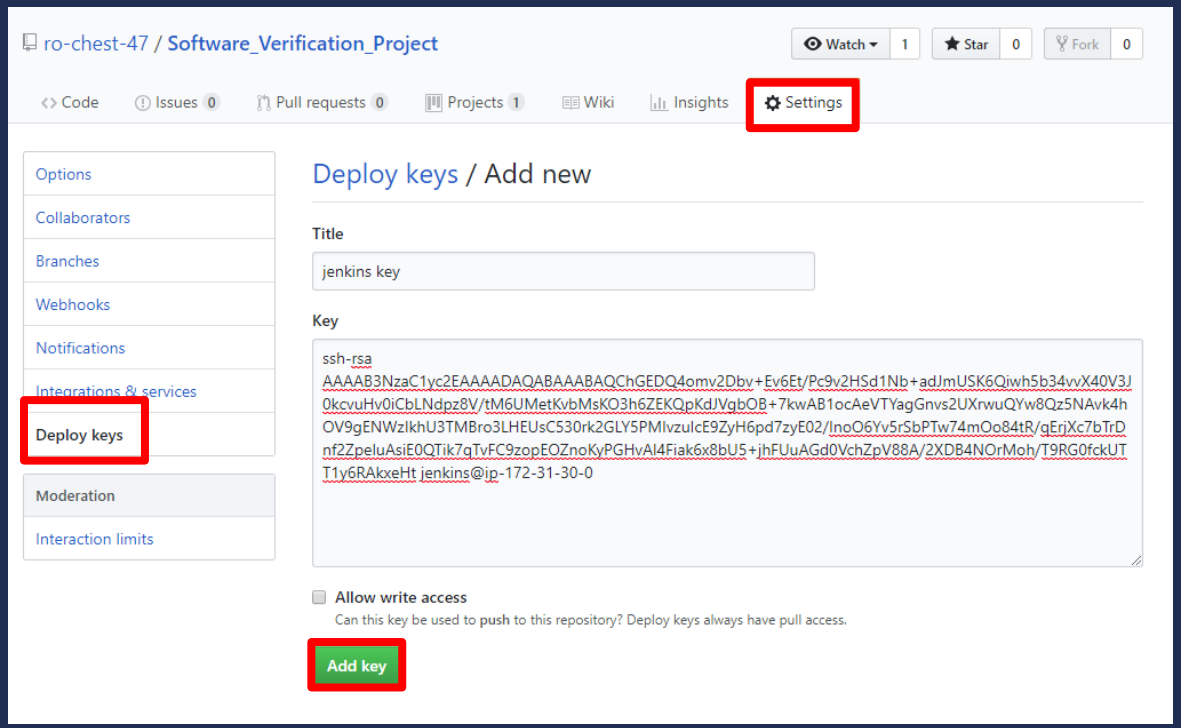


이번에는 Remote repository를 Local repository로 만들어서 작업하기 위해 File – Clone repository로 들어간다.

Local로 내려 받을 remote repository를 선택 후 Path를 설정하고 clone을 누른다.



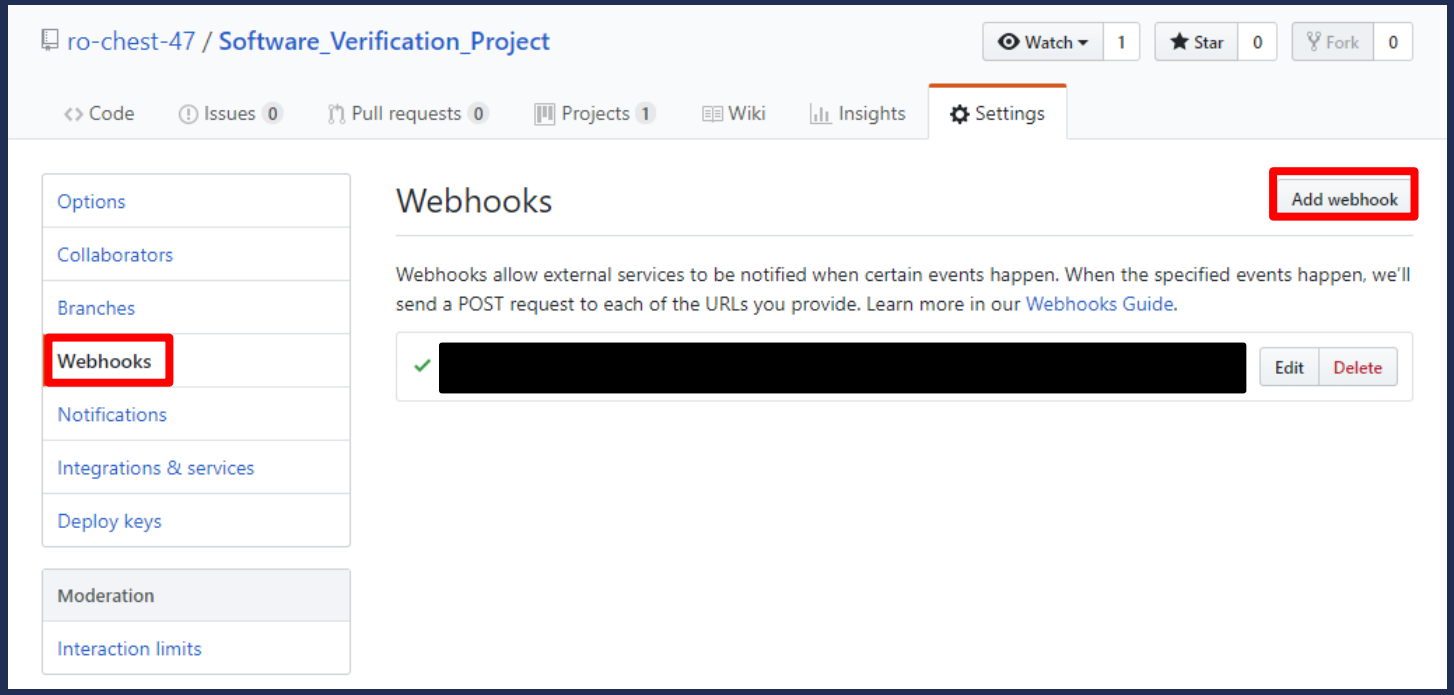
06 CI (Continuous integration) – GitHub + Jenkins



구축된 Jenkins 서버와 git의 Repository를 연동시키기 전에 ssh-keygen 프로그램으로 RSA 암호 기반의 Public key와 Private key를 만든다.

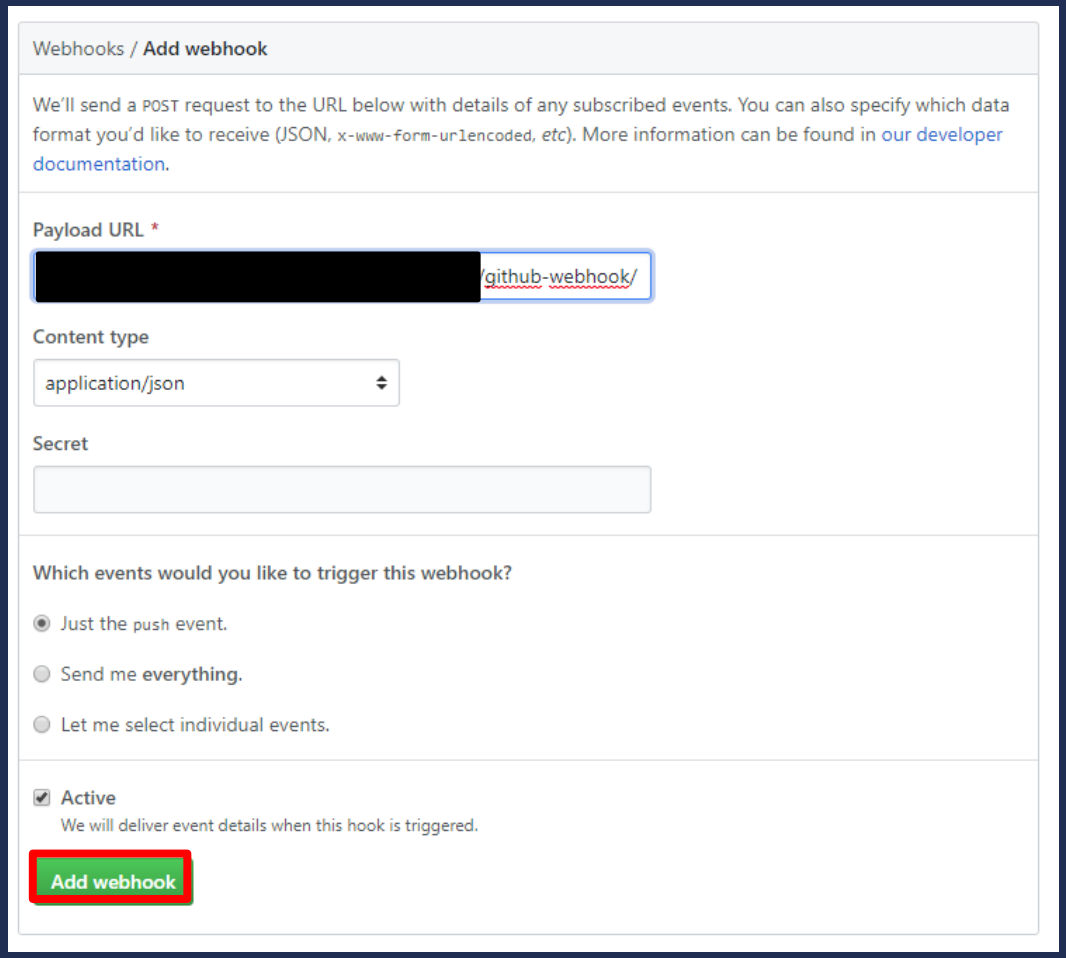
이후 연동시킬 git Repository의 Setting tab으로 들어가 Deploy keys – Add new 에 Public key를 등록한다.

06 CI (Continuous integration) – GitHub + Jenkins



등록 후 Webhook tab으로 가서 Add webhook을 선택한다.

06 CI (Continuous integration) – GitHub + Jenkins



Jenkins server URL 뒤에

/github-webhook/

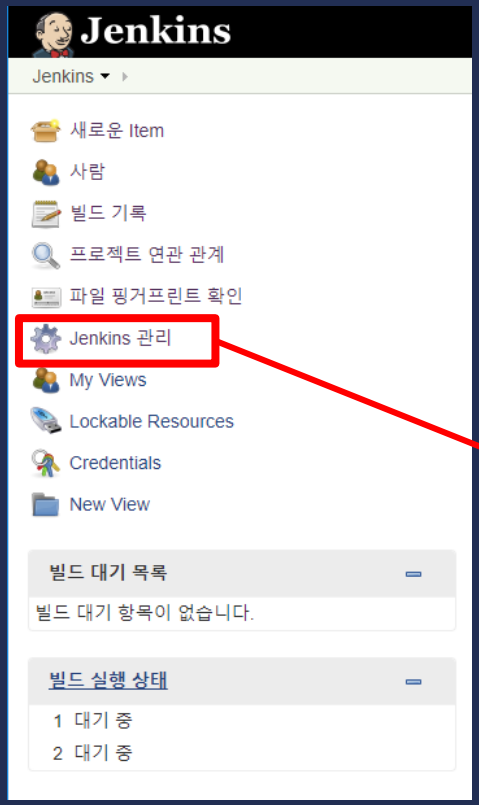
을 붙여 Payload URL 을 기입하고,

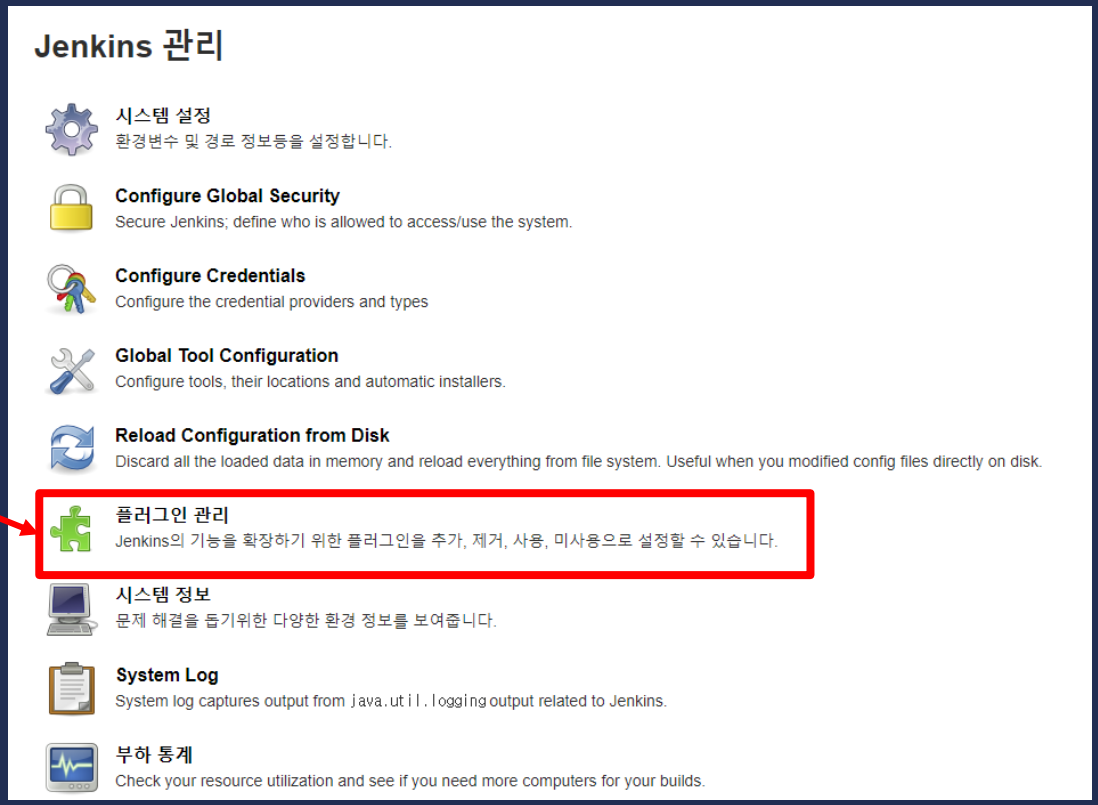
Content type 을 설정한 후

Just the push event, Active 옵션에

체크 후 Add webhook을 한다.

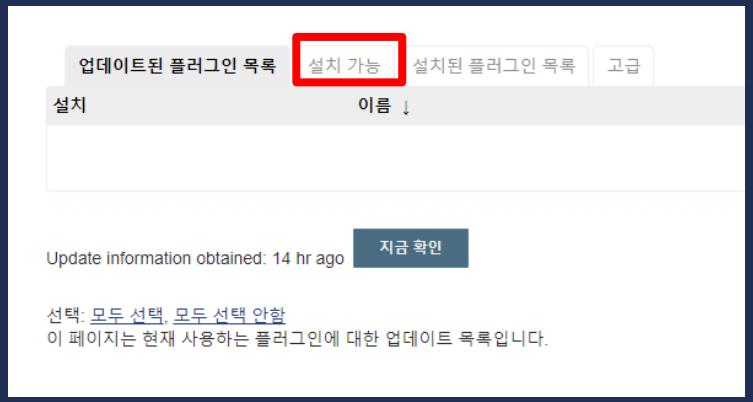
06 CI (Continuous integration) – GitHub + Jenkins





Push가 올 때 마다 build를 할 수 있도록 Github integration plugin을 설치한다.

06 CI (Continuous integration) – GitHub + Jenkins



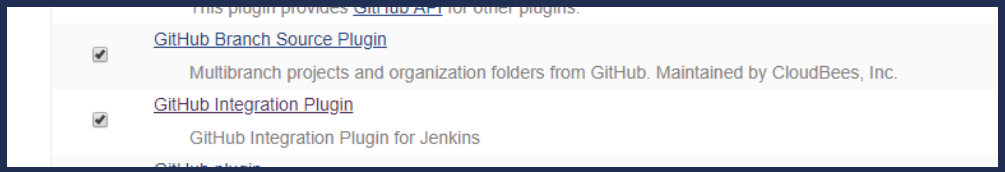
설치 가능 탭으로 간 후 좌측 상단 filter에

GitHub integration을 검색한다.

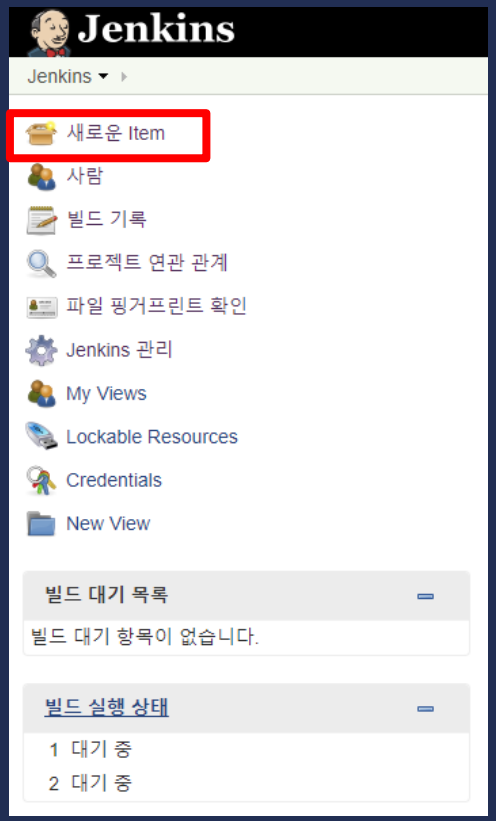
해당 plugin 설치 후

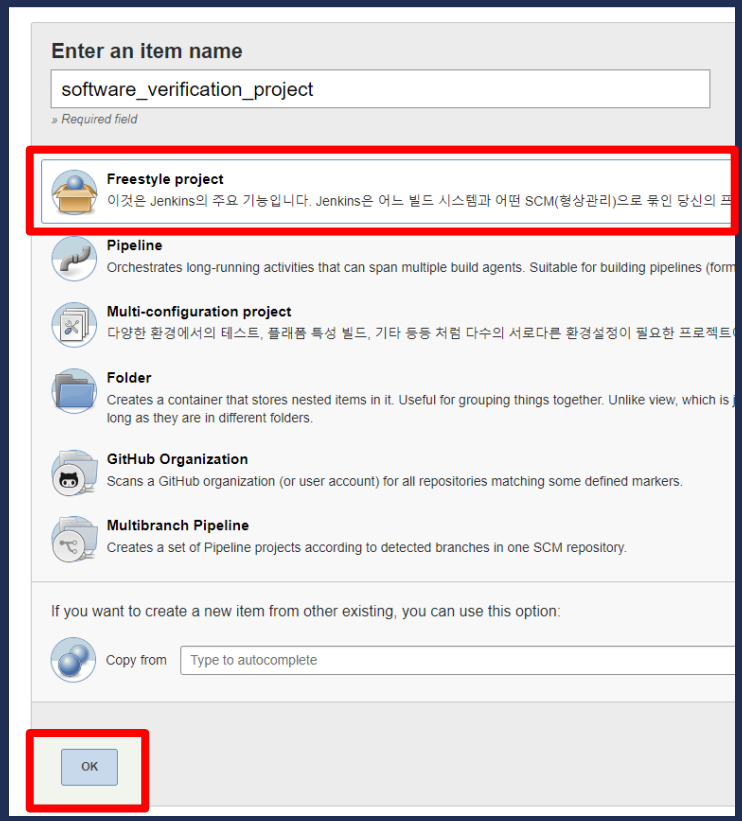
Jenkins 재시작 옵션에 체크하여

Jenkins를 다시 시작한다.



06 CI (Continuous integration) – GitHub + Jenkins

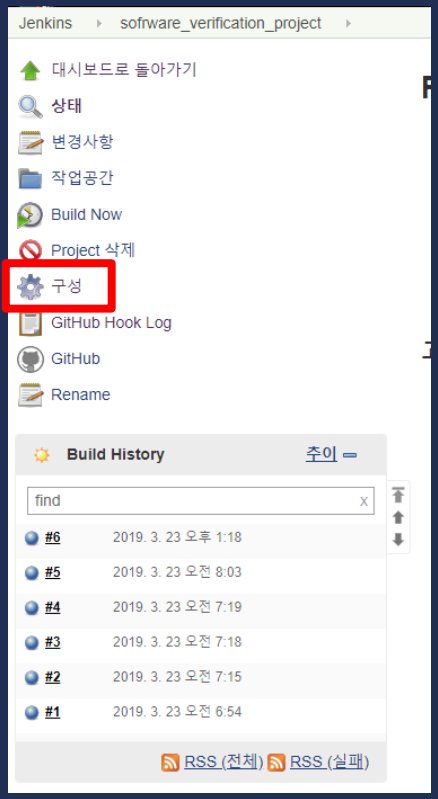


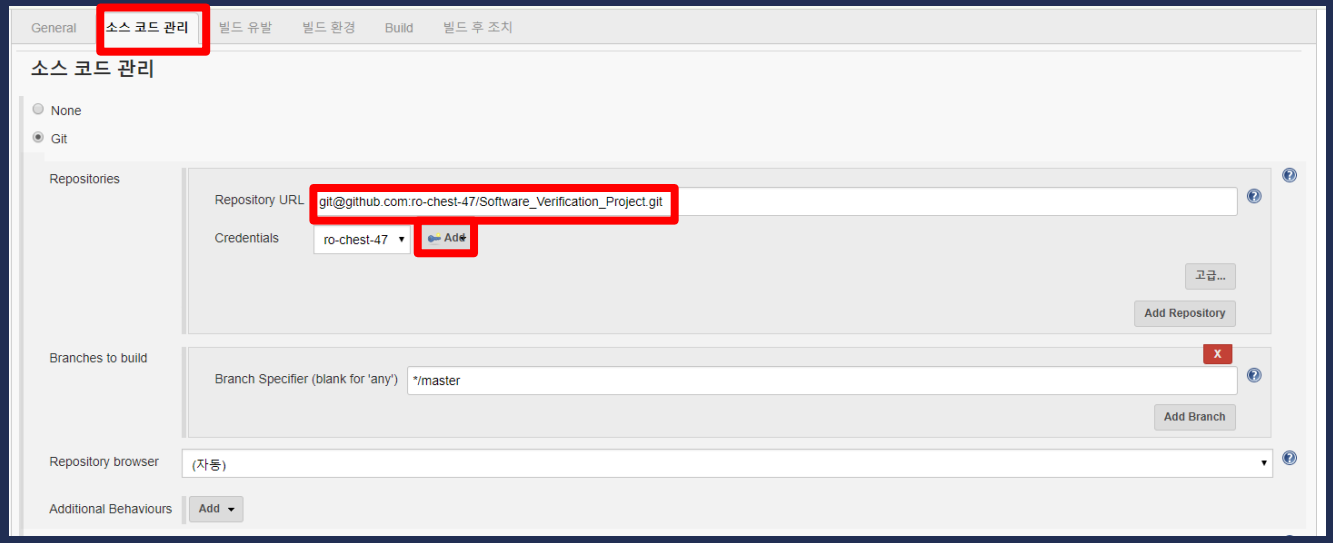


Freestyle project로

Project를 하나 생성한다.

06 CI (Continuous integration) – GitHub + Jenkins



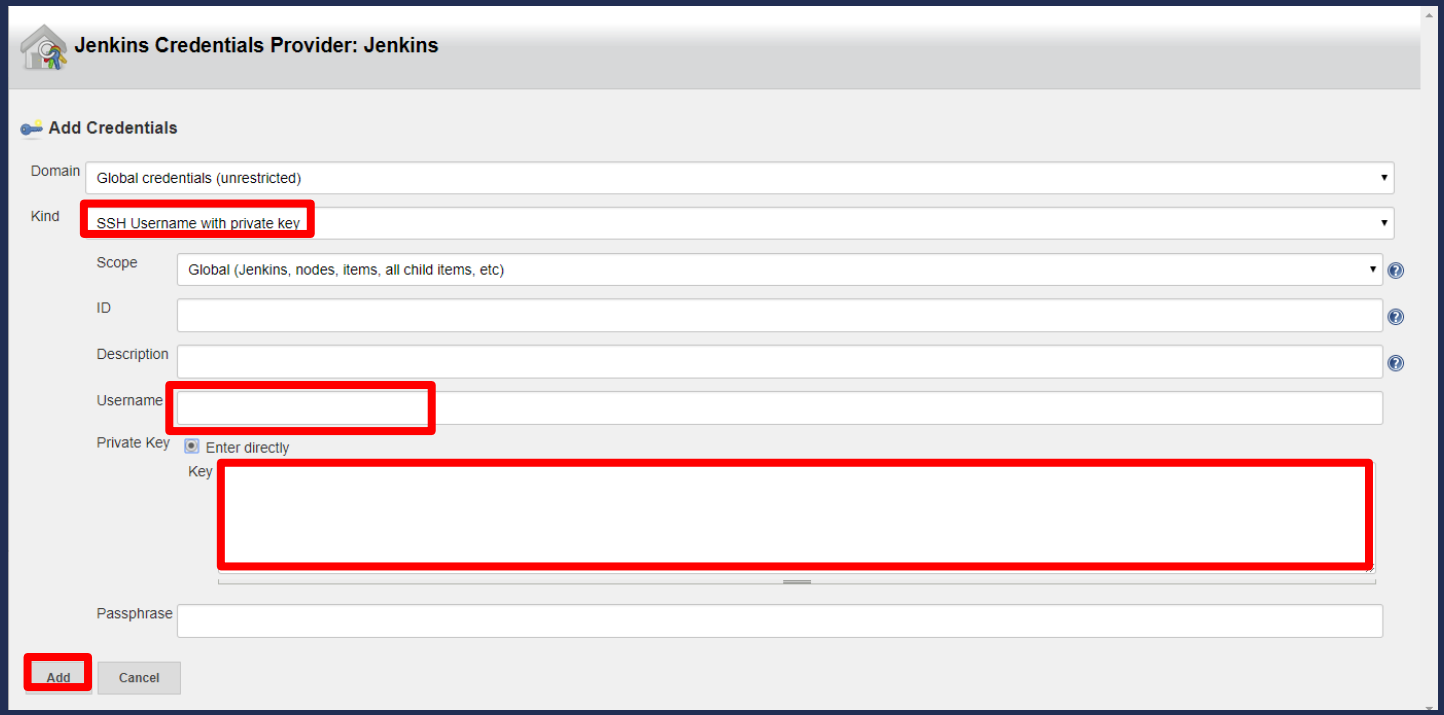


생성된 project의 구성-소스 코드 관리로 들어가

Git을 선택하고 repository URL을 기입한다.

이후 Add credentials 을 해준다.

06 CI (Continuous integration) – GitHub + Jenkins

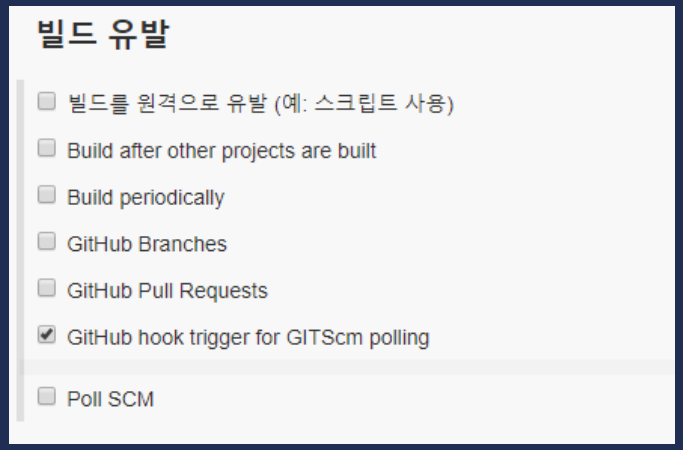


Username에 GitHub 계정 이름을 기입하고,

Private key에 아까 생성한 Private key 값 전체를 넣어준다.

이후 Add를 눌러 등록하고 선택한다.

06 CI (Continuous integration) – GitHub + Jenkins



빌드 유발에서

GitHub hook trigger for GITScm polling

옵션을 체크하고 apply – save 한다.

없으면 GitHub integration plugin 설치를 확인한다.

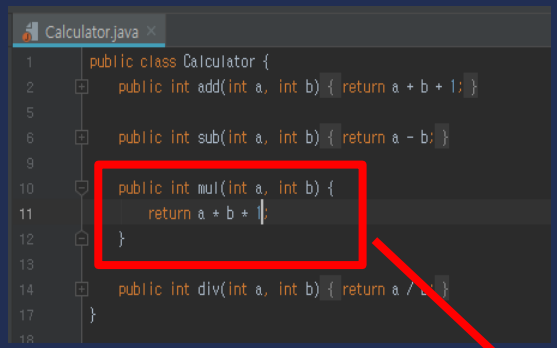
추가로, private key와 repository 주소를 옳게

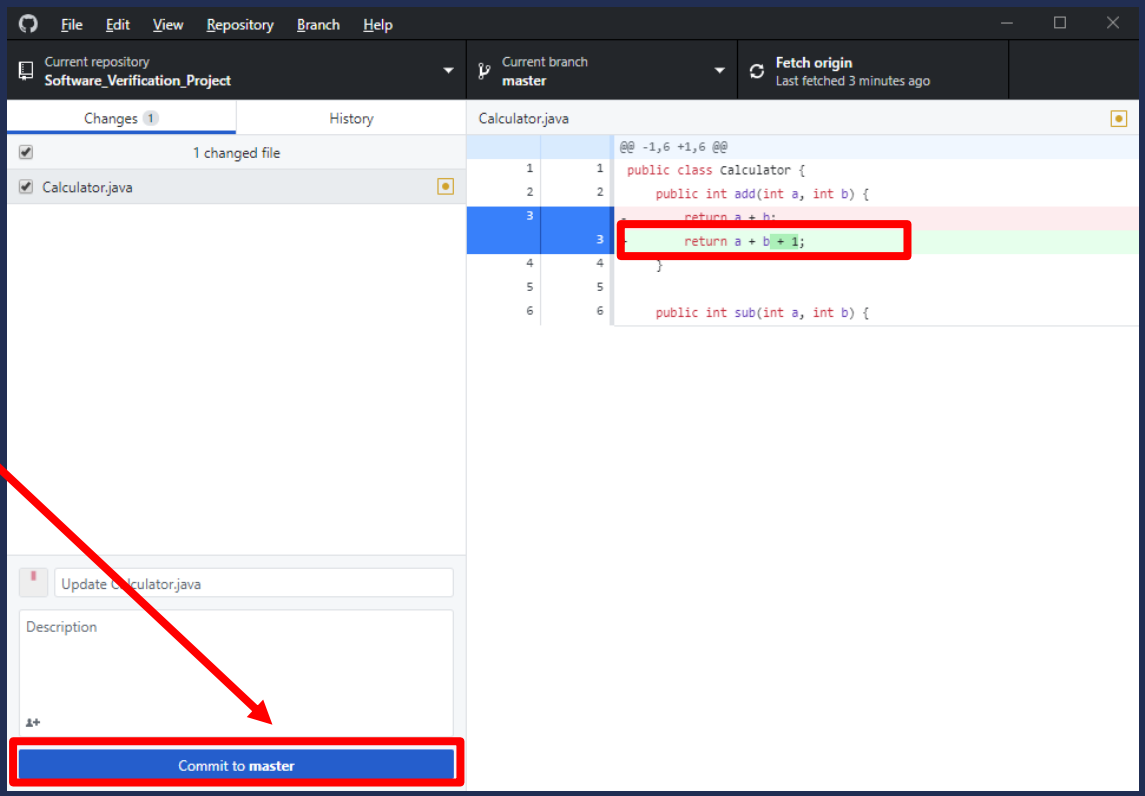
기입했음에도 불구하고 소스코드 관리 탭에서

Error가 발생하면 해당 서버에

Git이 설치되어 있는지 확인한다.

06 CI (Continuous integration) – GitHub + Jenkins





GitHub desktop 을 통해

연동한 remote repository의

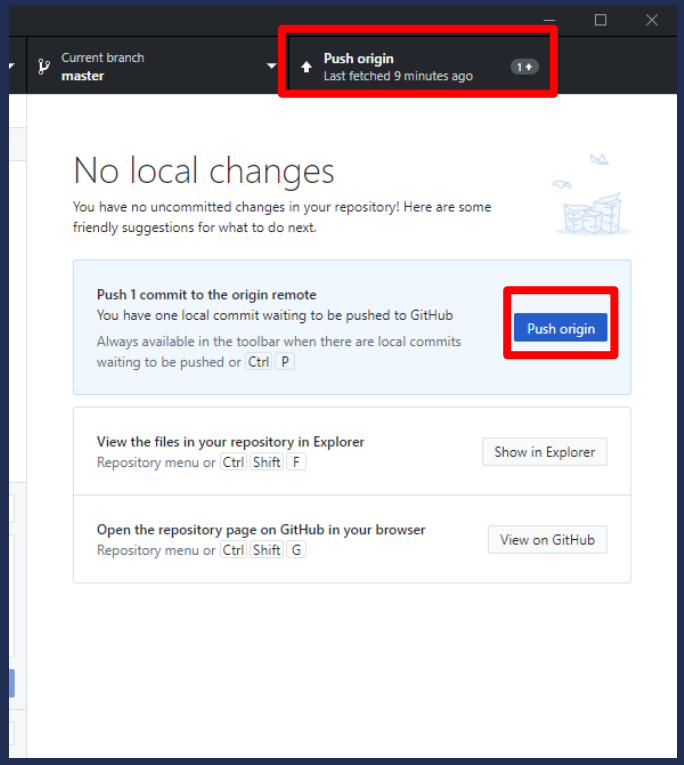
일부 source code를

Local repository에서

수정 후 commit → push 한다.

06 CI (Continuous integration) – GitHub + Jenkins

```
$ git config --global user.email "ro_chest_47@naver.com"

$ git config --global user.name "ro-chest-47"

Kang Jung Mo@DESKTOP-EEU1T8C MINGW64 ~
$
```



Commit 시 error가 발생하면

Git 설치 후 최초 사용자 등록을

안 해주었기 때문에 발생한 것이므로

Git bash 실행 후

git config --global user.email
"git 가입 시 작성한 email"

git config --global user.name "git username"

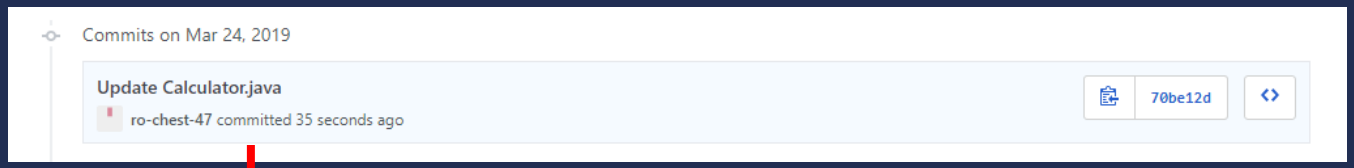
을 입력한다.

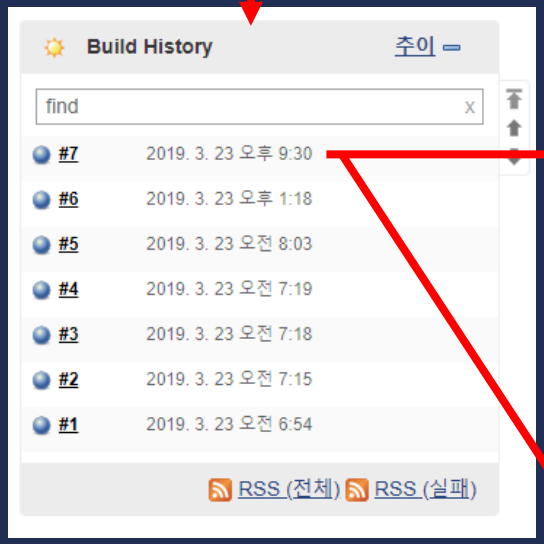
이후 commit → push 하여

Remote repository에 Local repository의

변경 사항을 반영시킨다.

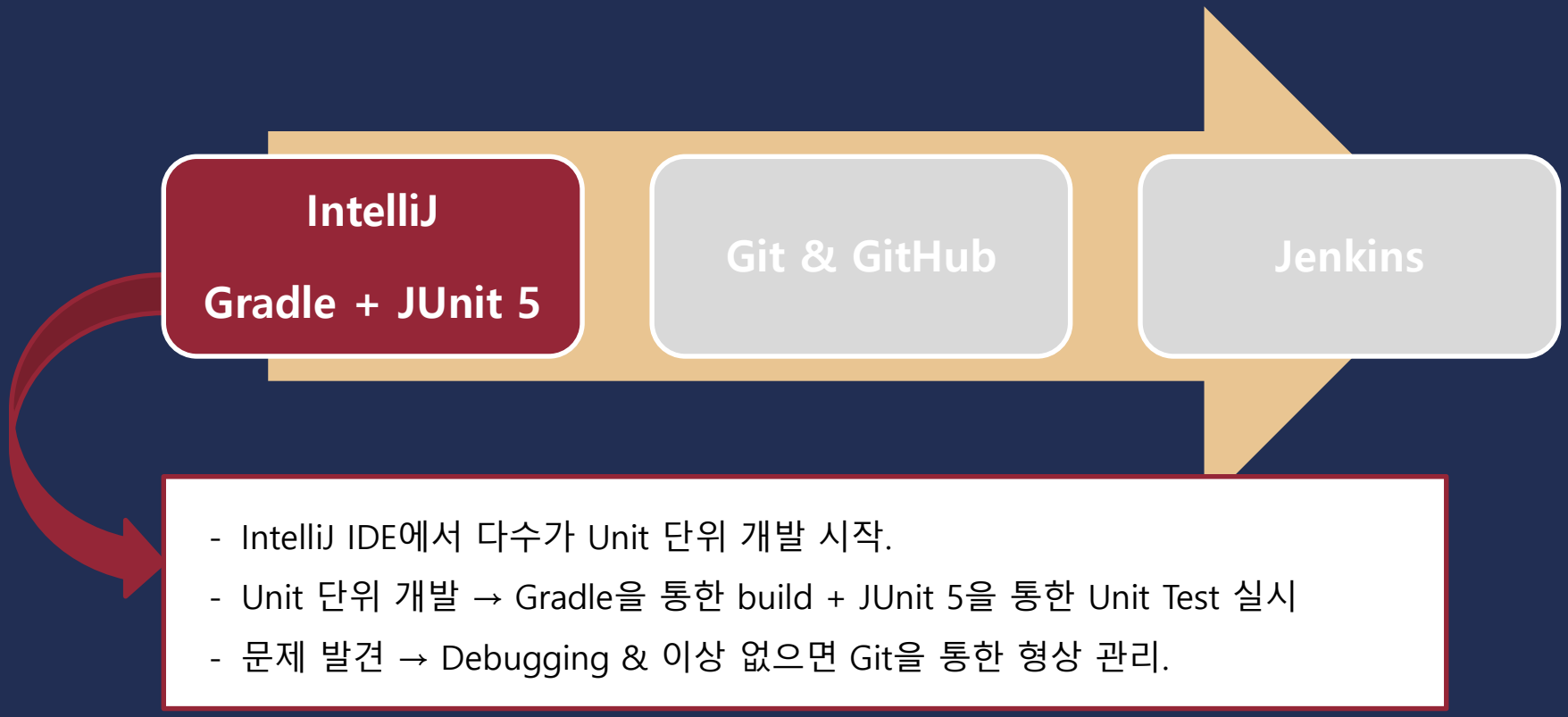
06 CI (Continuous integration) – GitHub + Jenkins





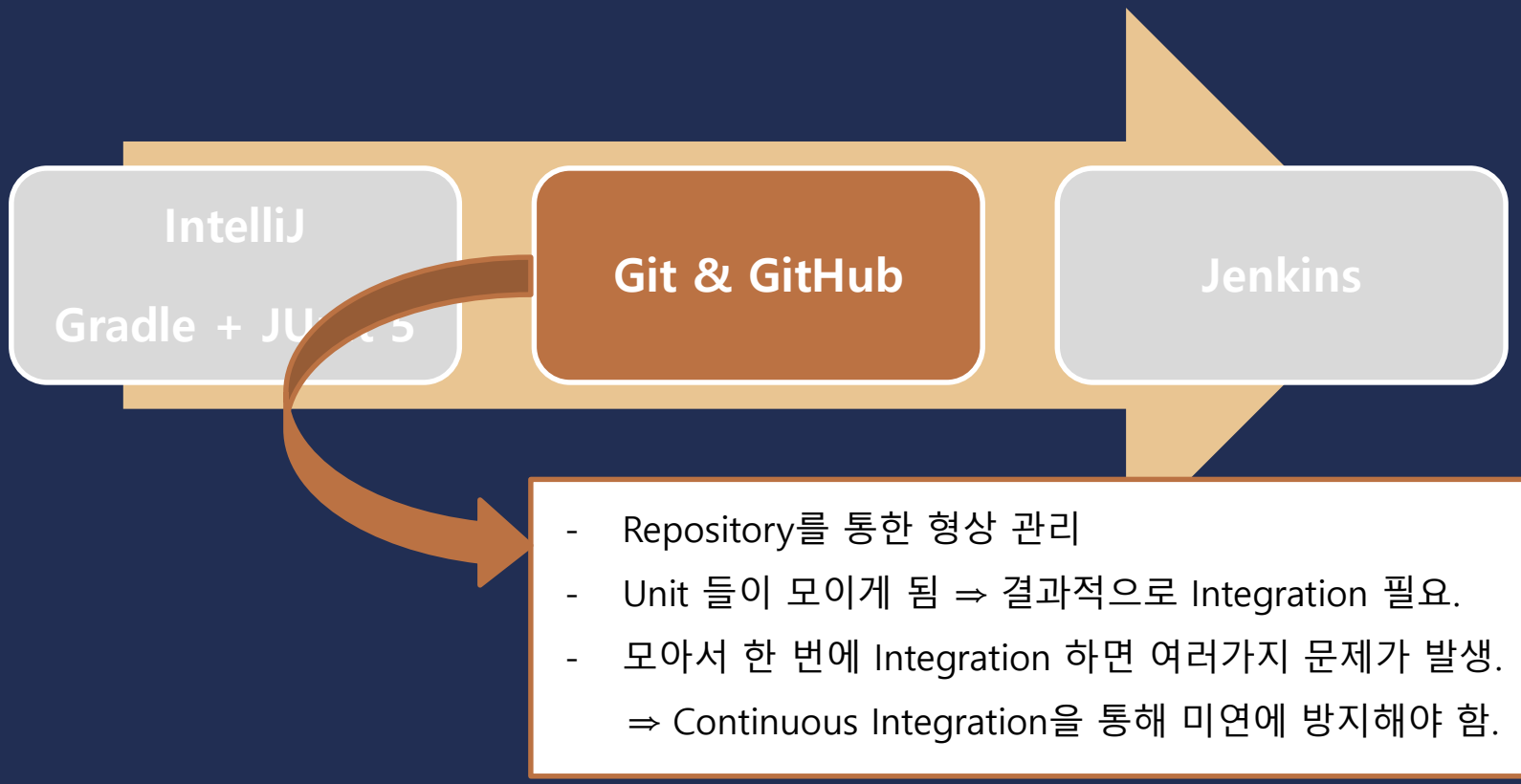
07 Summary

07 Summary



- IntelliJ IDE에서 다수가 Unit 단위 개발 시작.
- Unit 단위 개발 → Gradle을 통한 build + JUnit 5을 통한 Unit Test 실시
- 문제 발견 → Debugging & 이상 없으면 Git을 통한 형상 관리.

07 Summary


IntelliJ
Gradle + JU
Git & GitHub
Jenkins
- Repository를 통한 형상 관리
- Unit 들이 모이게 됨 ⇒ 결과적으로 Integration 필요.
- 모아서 한 번에 Integration 하면 여러가지 문제가 발생.
⇒ Continuous Integration을 통해 미연에 방지해야 함.

07 Summary

- Continuous Integration 환경을 제공
- Unit들을 Integration 한 후 build 실행
- 추가적인 test 및 analysis 실행
- 계속 Integration 하면서 문제점들을 제 때에 해결
  ⇒ 결과적으로 Software Quality 향상

감사합니다

THANK YOU

Software Verification Team 4

강 정 모
송 상 연
신 승 화