

1. JDK 설치 파일 다운로드

먼저 **Oracle** 로 가서 **Java** 를 설치파일을 다운로드 합니다.

<https://www.oracle.com/kr/java/technologies/javase/javase-jdk8-downloads.html>

Java SE Development Kit 8u301

This software is licensed under the [Oracle Technology Network License Agreement for Oracle Java SE](#)

Product / File Description	File Size
Linux ARM 64 RPM Package	59.15 MB
Linux ARM 64 Compressed Archive	70.84 MB
Linux ARM 32 Hard Float ABI	73.55 MB
Linux x86 RPM Package	109.49 MB
Linux x86 Compressed Archive	138.48 MB
Linux x64 RPM Package	109.24 MB
Linux x64 Compressed Archive	138.78 MB
macOS x64	197.26 MB
Solaris SPARC 64-bit (SVR4 package)	133.66 MB
Solaris SPARC 64-bit	94.8 MB
Solaris x64 (SVR4 package)	134.42 MB
Solaris x64	92.66 MB

Oracle 의 JDK 설치 목록 중에서 macOS 의 dmg 파일을 선택합니다.

Java SE Development Kit 8u301

This software is licensed under the [Oracle Technology Network License Agreement for Oracle Java SE](#)

Product / File Description	File Size
Linux ARM 64 RPM Package	59.15 MB
Linux ARM 64 Compressed Archive	70.84 MB
Linux ARM 32 Hard Float ABI	73.55 MB
Linux x86 RPM Package	
Linux x86 Compressed Archi	
Linux x64 RPM Package	
Linux x64 Compressed Archi	
macOS x64	197.26 MB
Solaris SPARC 64-bit (SVR4 package)	133.66 MB
Solaris SPARC 64-bit	94.8 MB
Solaris x64 (SVR4 package)	134.42 MB
Solaris x64	92.66 MB

You must accept the [Oracle Technology Network License Agreement for](#)

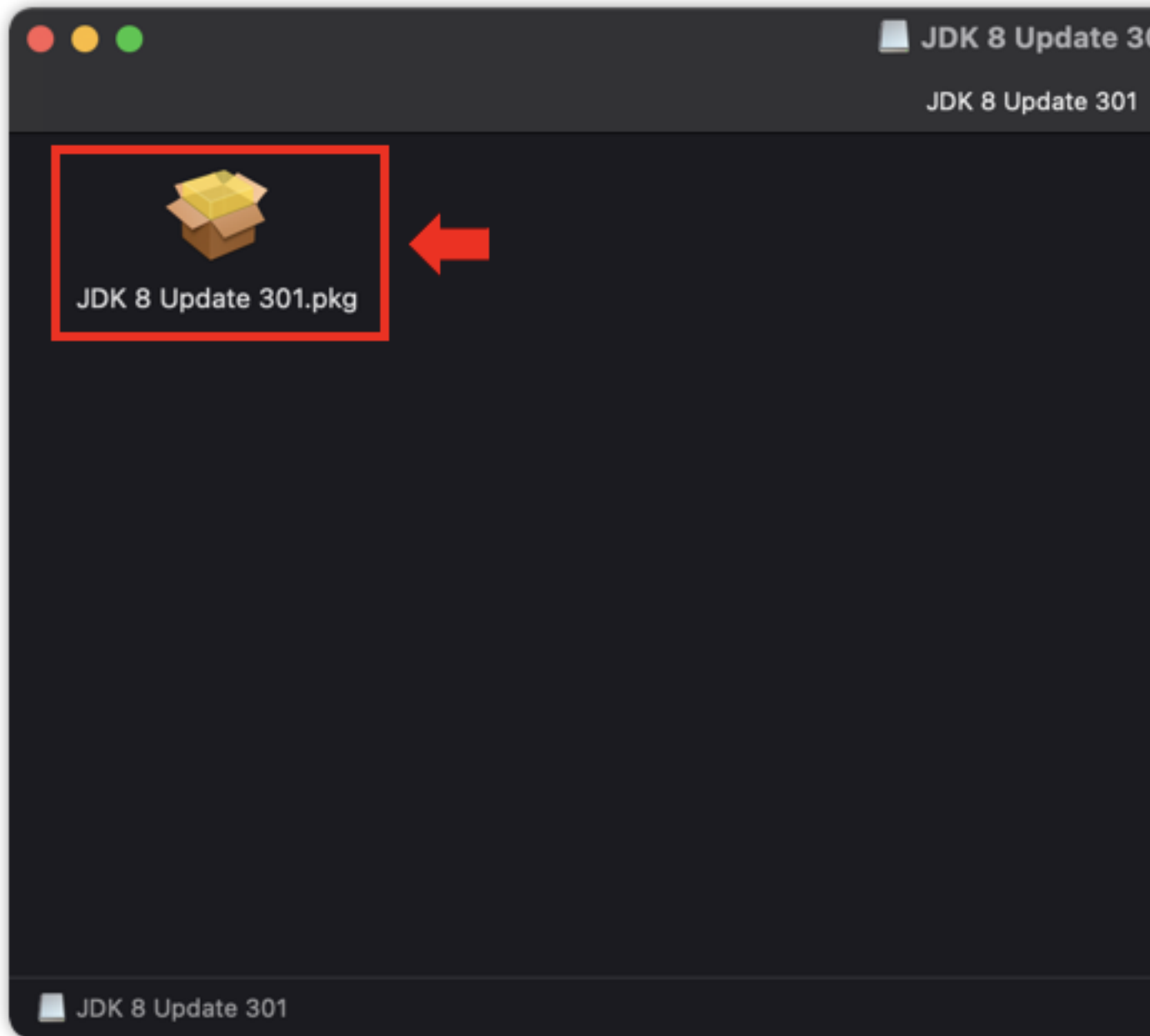
☒ I reviewed and accept the Oracle Technology Network License Agre

[Download jdk-8u301-macosx-x64.dmg](#)

체크하여 동의하여 주시고 아래 파일 다운로드 버튼을
클릭하여 다운로드 합니다.

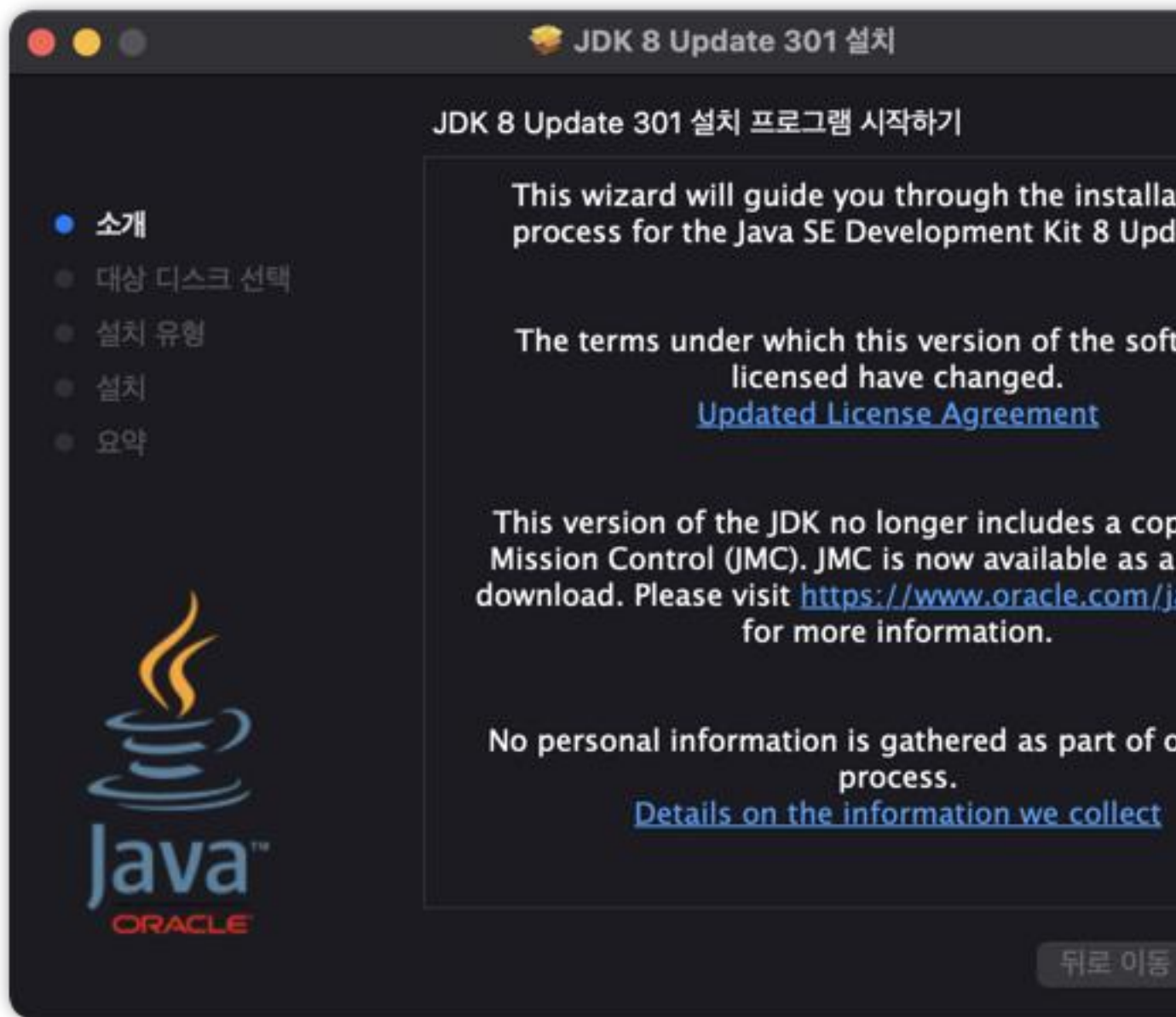
Oracle 계정 로그인이 되어있지 않다면 로그인 창으로
넘어가고 로그인을 하면 바로 다운로드가 진행됩니다.

2. JDK 설치



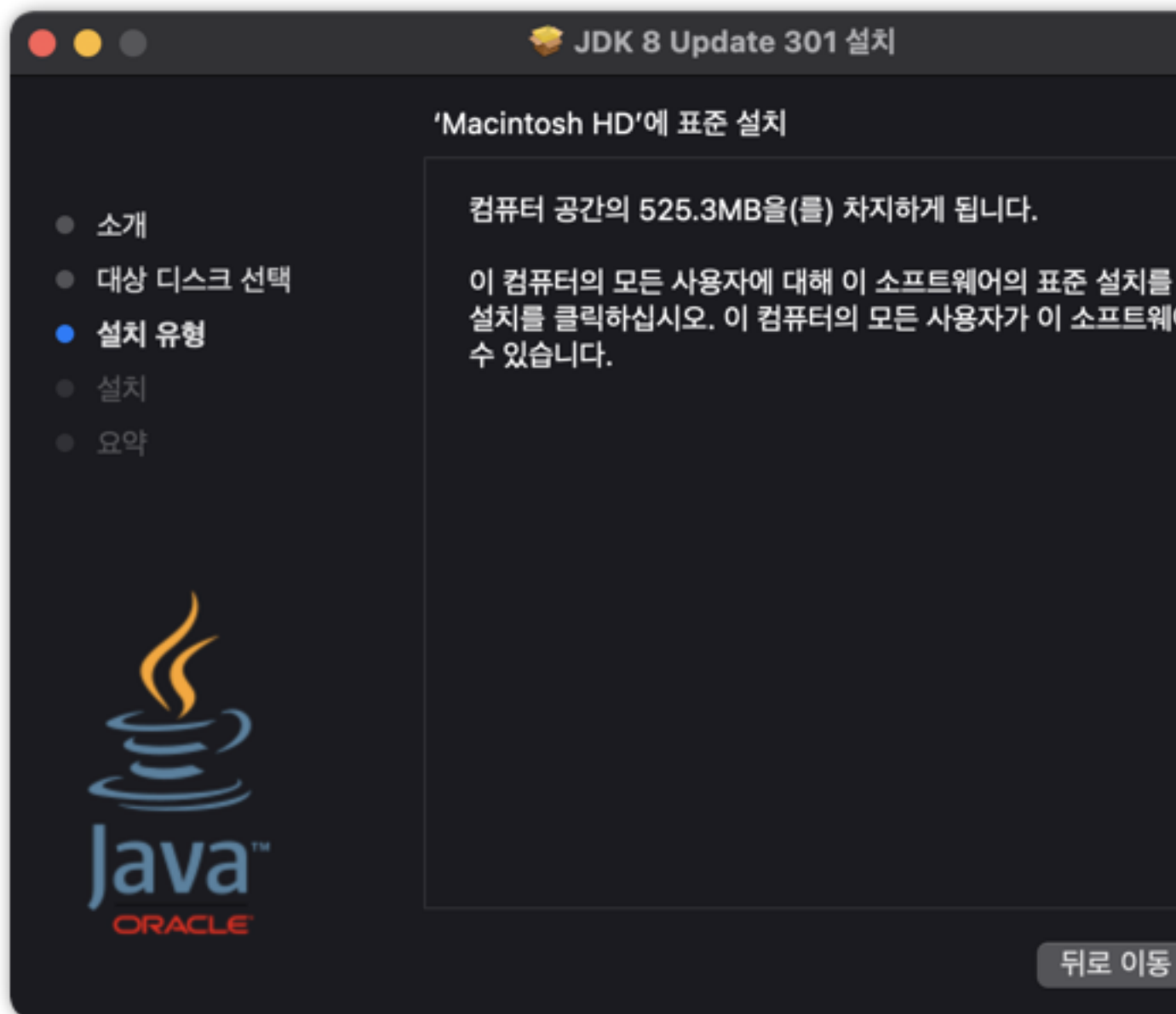
다운로드한 **dmg** 파일을 열면 위와같이 **pkg** 파일이 보입니다.

pkg 파일을 실행합니다.

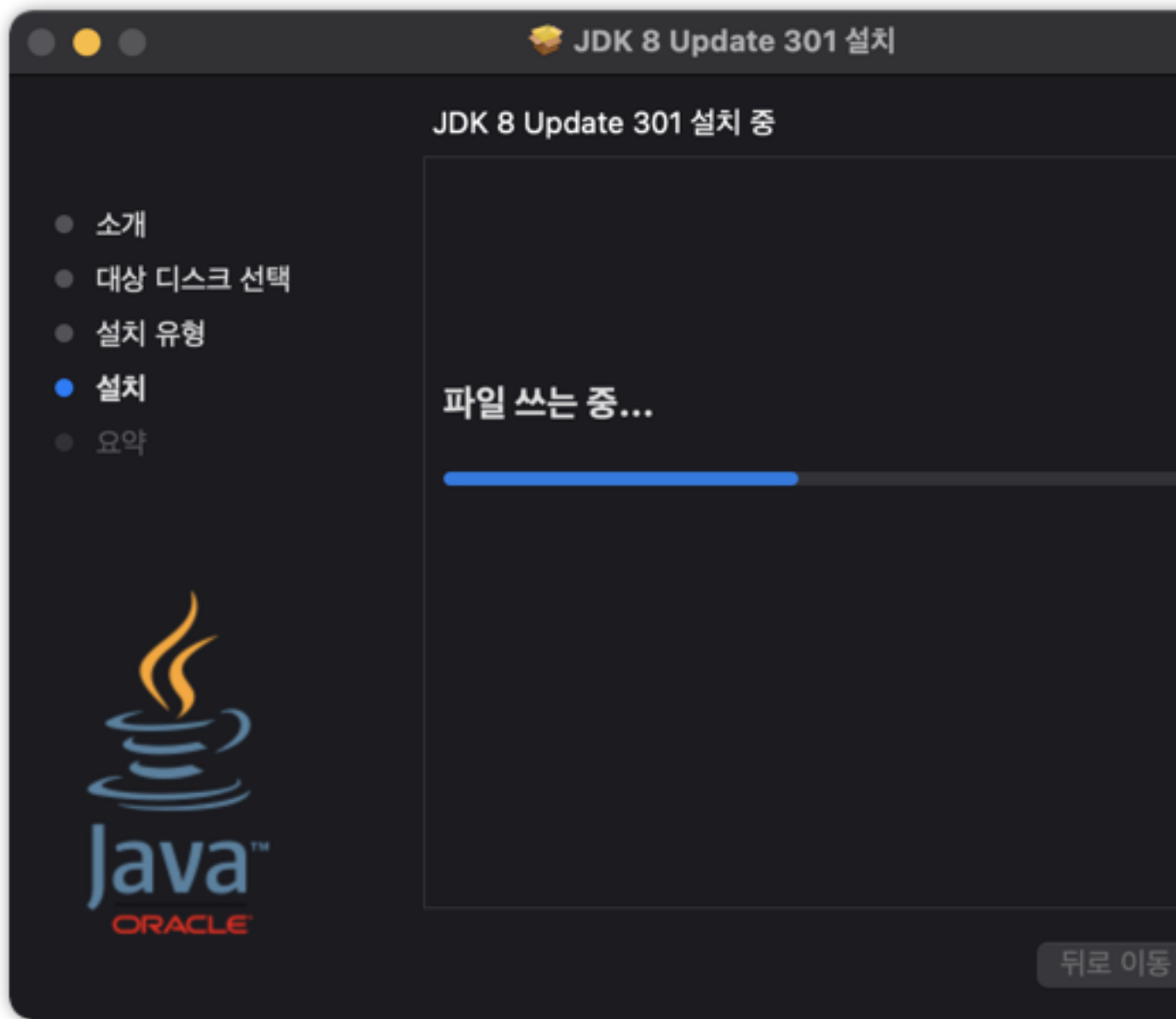


pkg 파일을 실행시키면 JDK 설치 프로그램이 시작됩니다.

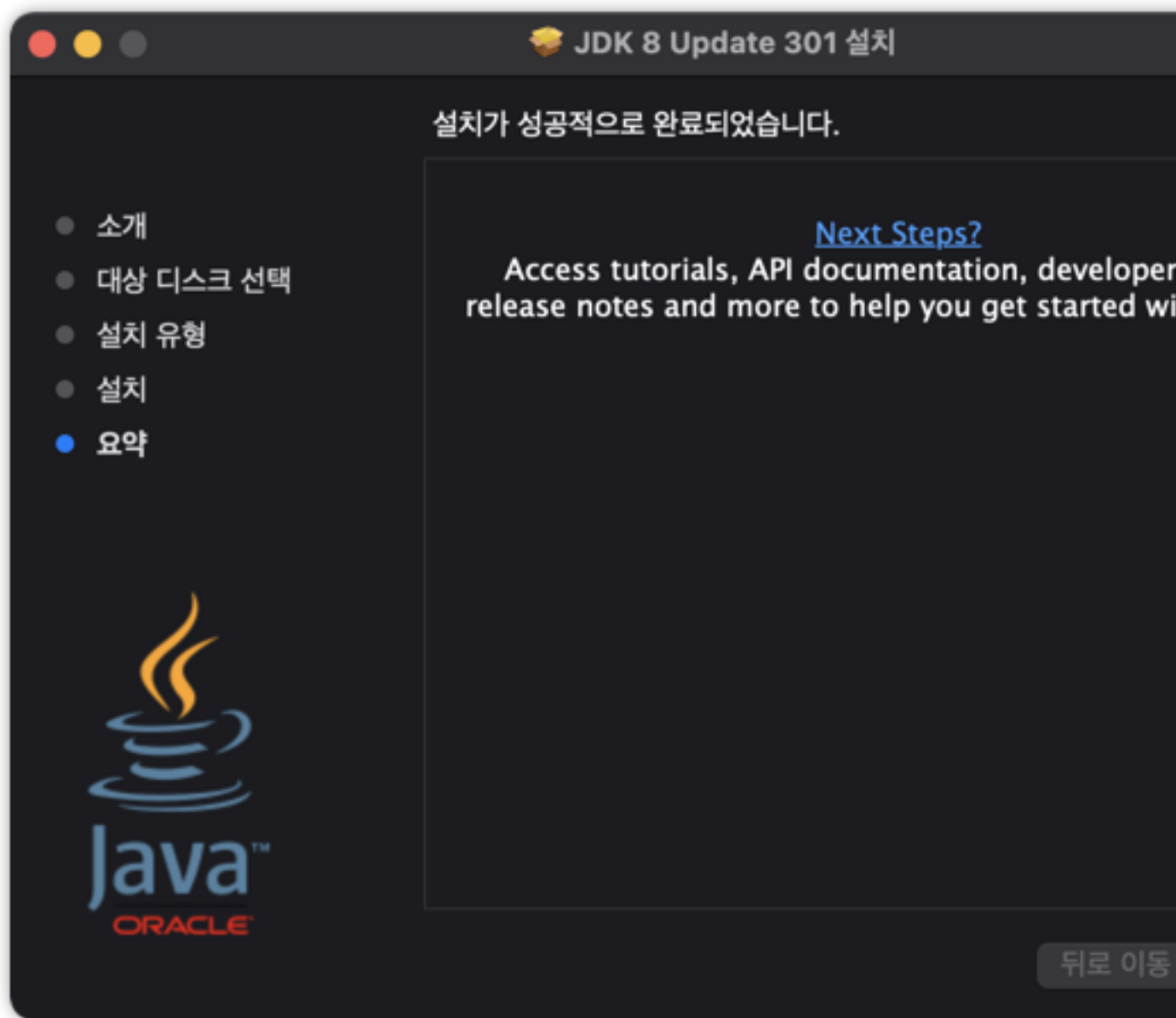
계속 버튼을 클릭하여 진행합니다.



설치 버튼을 클릭합니다.

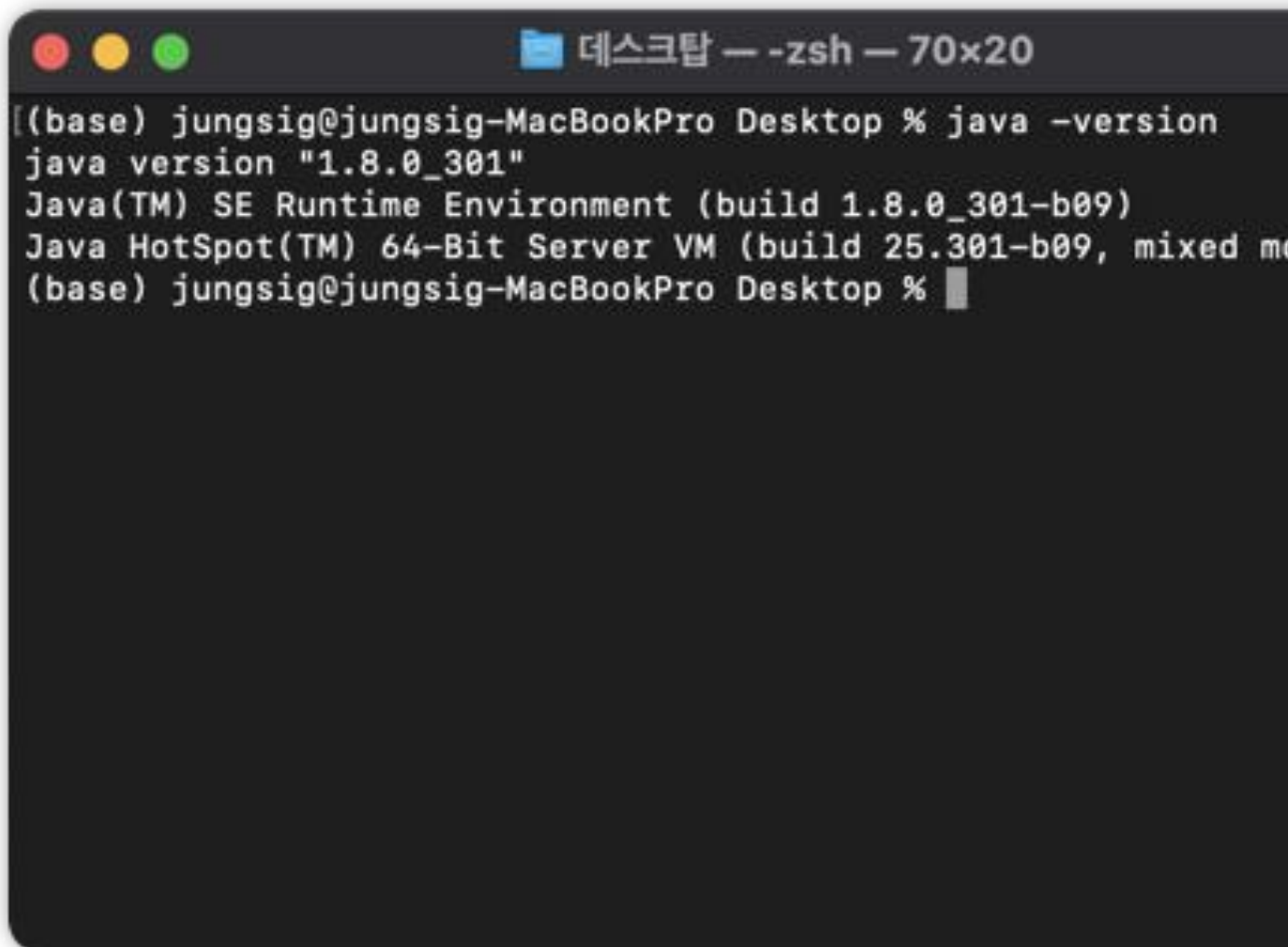


설치 버튼을 클릭하면 위 사진과 같이 설치가 진행됩니다.



설치가 완료되었습니다.

닫기 버튼을 클릭하여 설치를 마칩니다.

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, and a folder icon followed by the text '데스크탑 — -zsh — 70x20' on the right. The terminal content shows a command prompt where the user has entered 'java -version'. The output of the command is displayed in four lines: 'java version "1.8.0_301"', 'Java(TM) SE Runtime Environment (build 1.8.0_301-b09)', 'Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)', and a final prompt line '(base) jungsig@jungsig-MacBookPro Desktop %' with a cursor.

```

(base) jungsig@jungsig-MacBookPro Desktop % java -version
java version "1.8.0_301"
Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)
(base) jungsig@jungsig-MacBookPro Desktop %

```

터미널 창을 열어서,

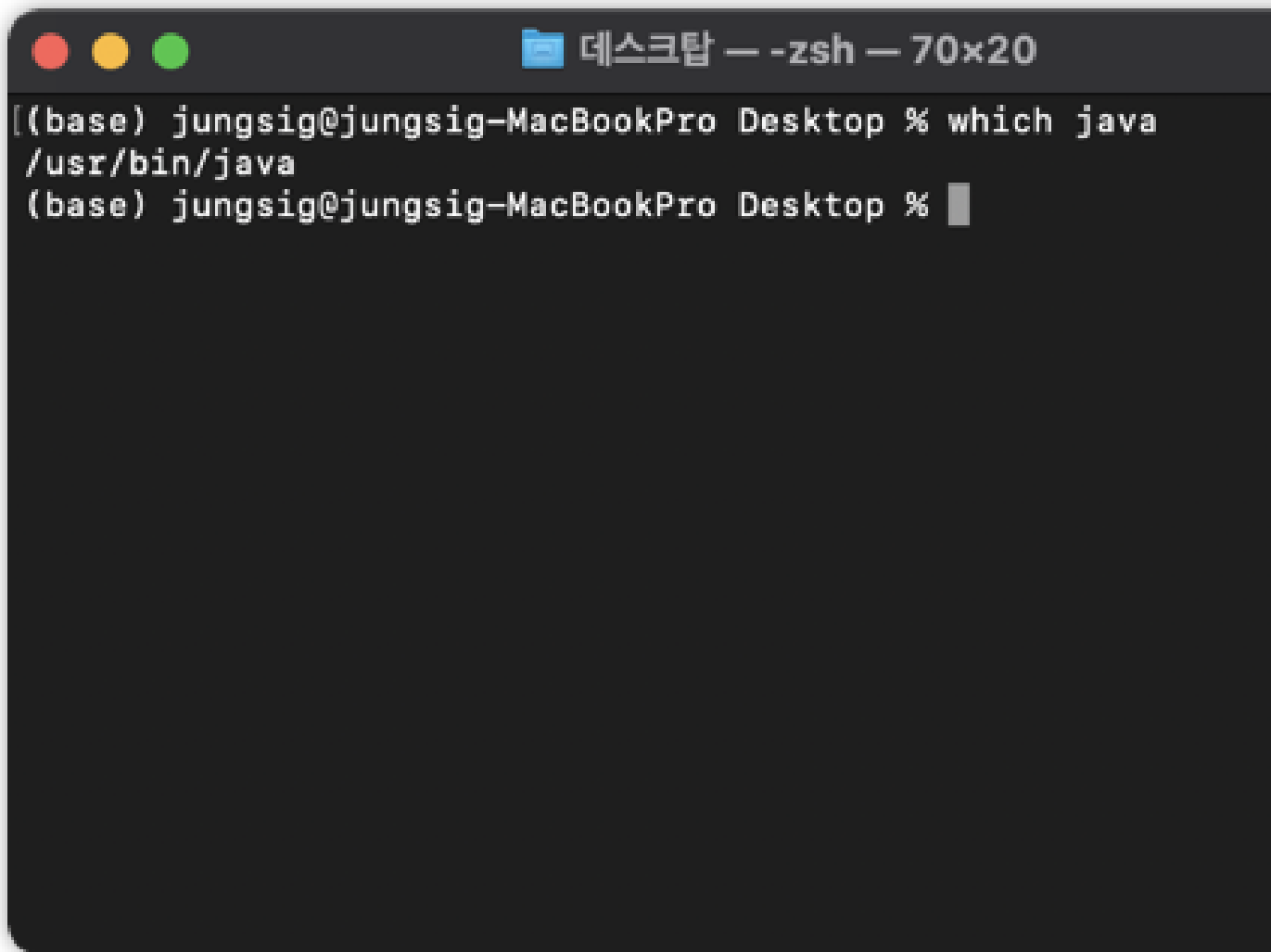
```
java -version
```

명령어를 입력하면, 설치한 Java 의 버전이 출력되면서

Java 설치가 완료된 것을 확인할 수 있습니다.

예전에는 환경변수를 직접 지정해줬었던 것으로 기억
하는데 이제는 그런 과정 없이 되는가봅니다.

참고로 **java** 가 어디에 설치되었는지 확인하시려면

A terminal window titled '데스크탑 — -zsh — 70x20' with standard macOS window controls (red, yellow, green buttons). The terminal shows the command `which java` being executed, resulting in the output `/usr/bin/java`. The prompt is `(base) jungsig@jungsig-MacBookPro Desktop %`.

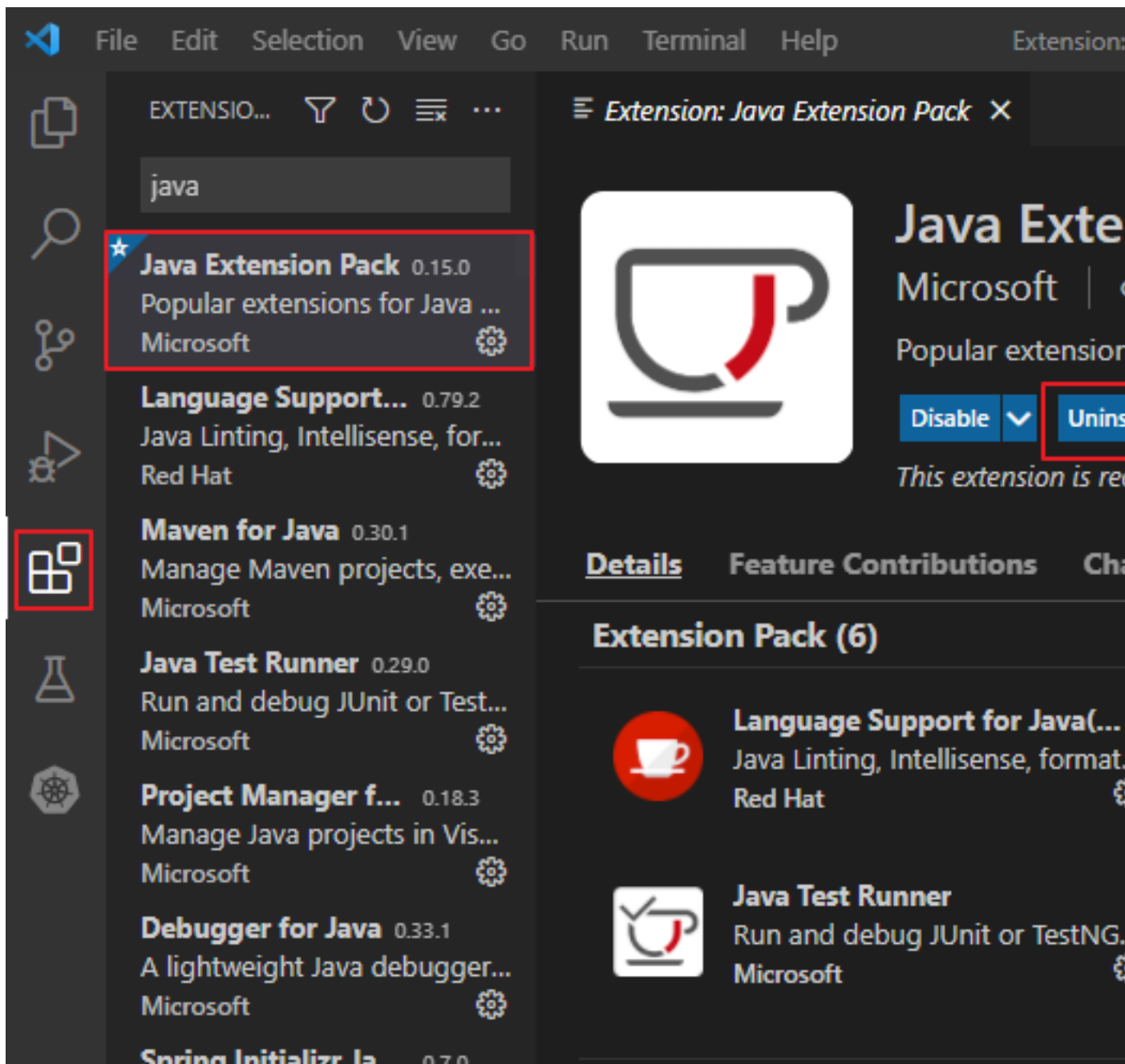
```
(base) jungsig@jungsig-MacBookPro Desktop % which java
/usr/bin/java
(base) jungsig@jungsig-MacBookPro Desktop %
```

`which java`

위 명령어를 통해 확인하실 수 있습니다.

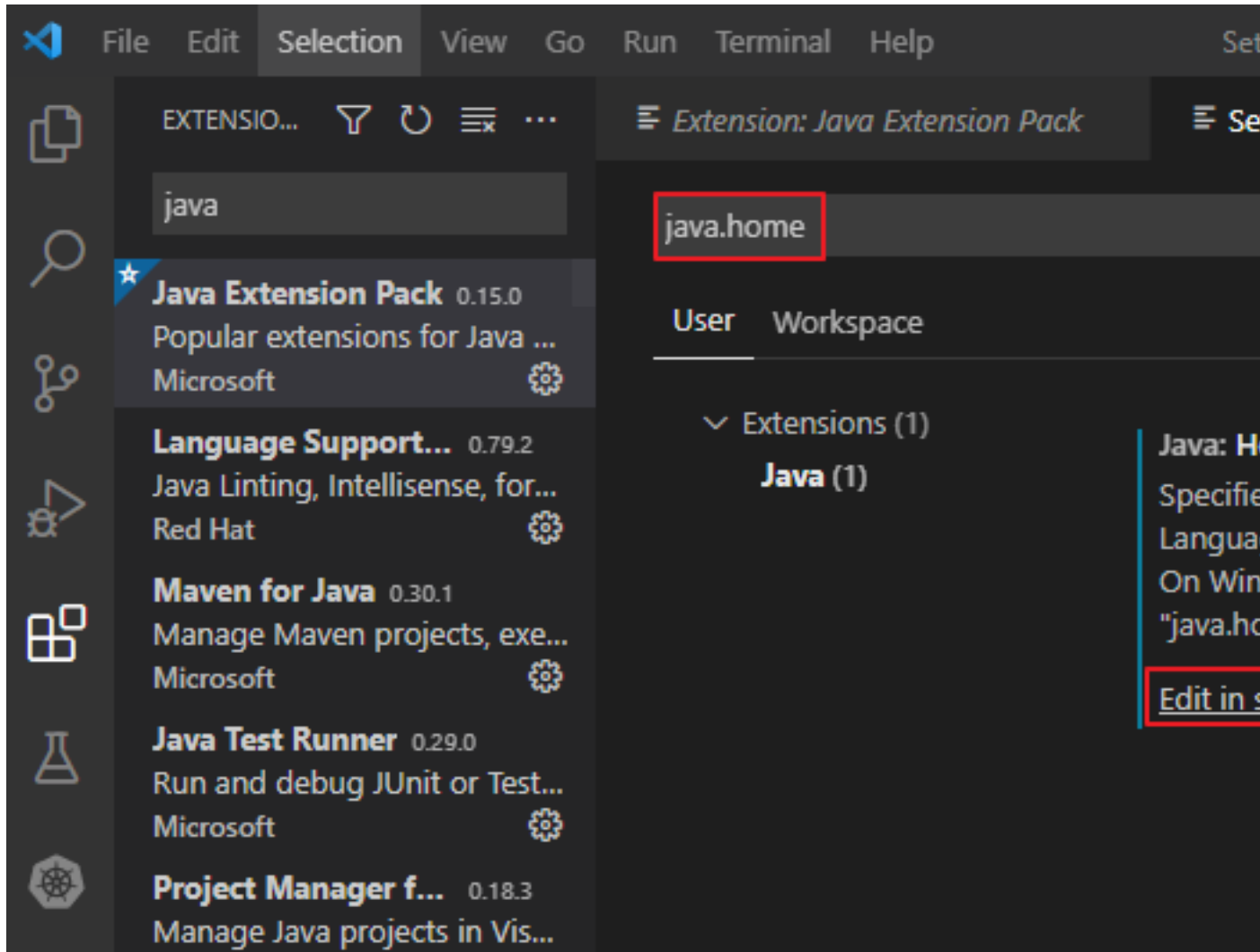
VS Code 세팅하기

이제 VS Code 를 실행하여 자바 개발 환경을 적용해 보자.

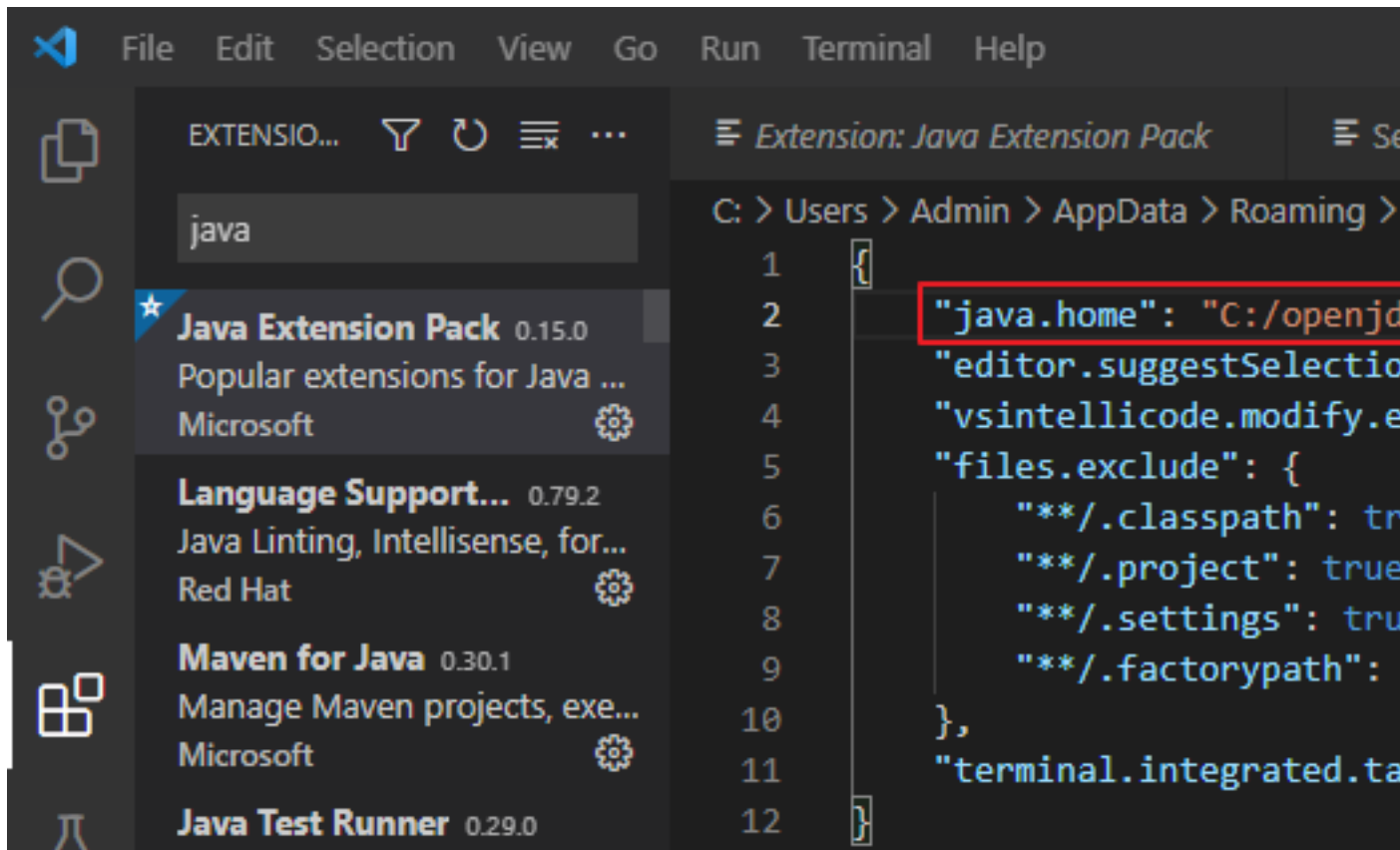


VS Code 를 실행 후, 왼쪽의 extension 을 클릭한 다음
java extension pack 을 다운받는다.

그 다음, 상단의 File -> Preferences -> Setting 로 들어간다.

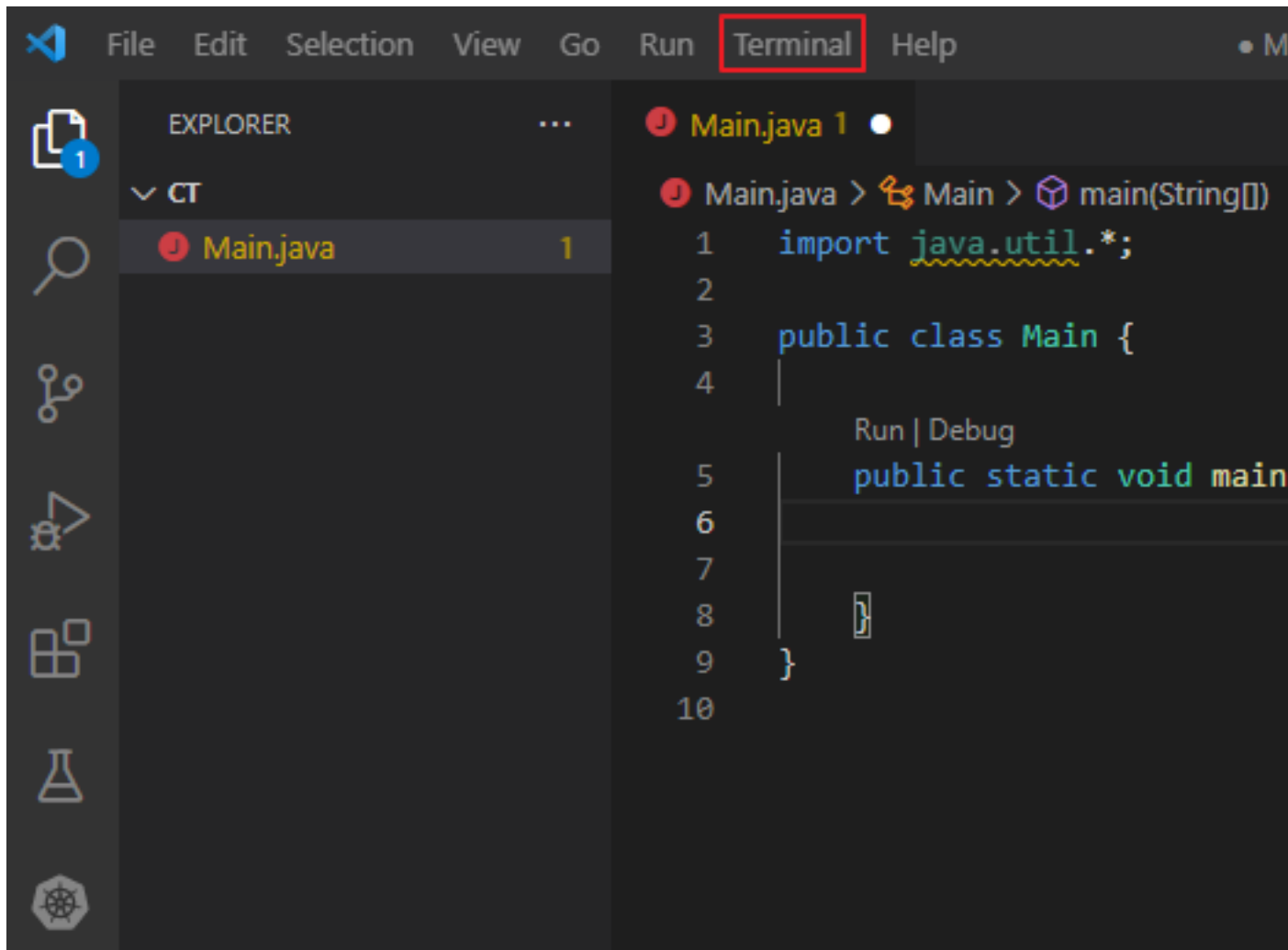


검색 창에 java.home 을 검색하면 다음과 같은 결과가 나온다. Edit in settings.json 을 클릭한다.



그 다음, JDK 를 다운받은 경로를 java.home 경로에 작성해 주면 된다. settings.json 의 내용은 다를 수 있다.

이제 세팅을 완료하였다. VS Code 에서 .java 파일을 작성해 보자.



자바 파일 형식이 잘 작성되는 것을 알 수 있다.
작성한 자바 프로그래밍 코드를 실행하려면 오른쪽
상단의 ▶를 클릭한 후, run java 를 누르면 된다.
디버깅을 할려면 debug java 를 누르면 된다. 실행
터미널을 보고 싶다면 상단의 Terminal -> new
Terminal 을 클릭하면 된다.

이제 VS Code 에서도 자바 프로그래밍을 할 수 있도록 자바 개발 환경을 설정해 보았다. 앞으로는 간단한 코드 작성은 VS Code 에서 하도록 해야겠다-

객체(object)

객체라는 단어부터 잘 와닿지가 않습니다. 객체의 뜻은 어떤 대상, 사람이나 사물을 말합니다. 눈에 보이는 모든 것들과 보이지 않은 것들도 모두 객체입니다.

굉장히 범위가 넓습니다. 모든 것이 객체다. 라는 말도 자주 씁니다. 이럴 때는 한정적인 케이스를 알아보는게 좋습니다.

예를 들어 TV 는 객체입니다. TV 객체에는 고유의 속성이 있을테고 행동이 있을 것 입니다. 여기서 TV 의 속성을 클래스의 속성이라고 하고 행동을 메소드라고 합니다.

TV 에 채널이 12 개 있다. 그러면 1 부터 12 까지의 채널이 TV 의 속성이라고 볼 수 있습니다. 메소드는 이 속성들에 적용할 수 있는 행동을 말합니다.

리모콘을 눌러 파워를 키고 채널을 5 번으로 돌렸습니다. 채널을 돌리는 행동은 TV 에 할 수 있는 행동입니다. 전자레인지의 채널은 돌리거나 하지 않습니다.

객체의 속성과 메소드는 상당히 긴밀하게 연결되어 있습니다 이런 식으로 객체를 만들면 실생활의 문제에 대하여 직관적으로 설계하여 프로그래밍에 구현할 수 있습니다. 특히 자바 언어는 객체지향 프로그래밍의 전통을 이어가고 있습니다.

클래스(class)

그래서 클래스는 객체를 컴퓨터 프로그램으로 설계해 놓은 것 입니다. 객체라는 사물은 광범위하고 관념적이므로 그 모든 내용을 컴퓨터 코드로 만들 수는 없습니다. 중요한 속성과 메소드만 구현하는 것입니다.

사실 자바를 시작해온 순간부터 클래스를 사용하고 있었습니다.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
    }  
}
```

```
}  
}
```

public class Main 이 그동안 사용해오던 클래스입니다.
public static void main()함수는 좀 특별한 메소드긴
하지만 이것도 메소드입니다.

계산기 클래스 예제

자바는 100% 객체지향 프로그래밍을 지향하기
때문에 시작점도 이미 클래스입니다.

별도의 클래스를 만들어 보겠습니다. 쉬운 예제로
접근하면 좋은데요.

사칙연산이 가능한 계산기를 만들어 보겠습니다.

```
class Calculator {  
    int result;  
    public void add(int a, int b){  
        result = a + b;  
    }  
    public void sub(int a, int b){  
        result = a - b;  
    }  
    public void mul(int a, int b){  
        result = a * b;  
    }  
    public void div(int a, int b){  
        if(b != 0) {  
            result = a / b;  
        }  
    }  
}
```

```
        } else {  
            result = 0;  
        }  
    }  
}
```

실생활에 있는 계산기를 객체로 만든다고 생각하면 됩니다. 단 자바의 경우 정적 타이핑 언어기 때문에 형변환이라던가 조금 더 신경쓸 부분이 많습니다. 그런 부분은 차후에 알아가기로 하고 지금은 클래스가 무엇인가 집중합니다.

계산기의 결과값을 result 라는 변수에 저장하게 해봤습니다. result 가 곧 내부 속성입니다. 함수형 프로그래밍을 할 때는 return 으로 호출한 곳으로 돌려주지만 계산기는 자신의 데이터를 가지고 있으므로 당장 돌려주지 않아도 됩니다. 멤버변수에 가지고 있으면 됩니다.

이를 사용하는 방식도 다릅니다.

클래스 사용의 기본은 인스턴스를 만들어서 사용합니다. 인스턴스(instance)란 또 뭐냐 용어가 많이 나옵니다. 쉽게 말해 클래스가 설계도라면 인스턴스는 제품입니다.

실제 세계의 객체 아이디어를 가져와서 클래스라는 설계도를 만들고 그 설계도를 가지고 메모리에서 실제 작동하는 제품이 인스턴스(instance)입니다. 좀 한번에 이해가 안가도 정상입니다. 좀 시간이 걸립니다.

그래서 인스턴스를 만들면 하나의 메모리 공간을 할당 받습니다. 멤버변수를 보유하는 메모리 공간 뿐 아니라 메소드(특정 규칙을 가진 함수) 역시 사용할 수 있습니다.

```
public class Main {  
    public static void main(String[] args) {  
        Calculator myCal = new Calculator();  
        myCal.add(10, 20);  
        System.out.println("add: " + myCal.result);  
        myCal.sub(30, 14);  
        System.out.println("sub: " + myCal.result);  
        myCal.mul(3, 9);  
        System.out.println("mul: " + myCal.result);  
        myCal.div(10, 0);  
        System.out.println("div: " + myCal.result);  
    }  
}  
add: 30  
sub: 16  
mul: 27  
div: 0
```

위 코드에서는 result 라는 공간에 사칙연산의 결과값을 돌려가면서 저장한다는 것을 볼 수 있습니다. 보통의 계산기가 결과값을 하나 저장하기 때문에(+M 을 제외하면) 프로그램에도 하나만 넣었습니다.

클래스를 만들기 위해서 class 키워드를 사용합니다.

```
접근제어자 class 클래스이름 {  
    멤버변수;  
    메소드;  
}
```

접근제어자는 public 같은 키워드를 말합니다.
클래스도 사용가능한 범위를 조절할 수 있습니다.

클래스 안에서 멤버변수나 메소드에도 각각 접근제어자를 붙일 수 있는데요. 이런 외부와의 인터페이스를 설계하는 것도 객체 지향 프로그래밍의 방식입니다.

함수를 중심으로 하는 절차지향형 프로그래밍과는 많이 다르죠. 프로그램이 작동되는 것 뿐만 아니라 객체간에 어떠한 관계를 맺고 있는지도 생각하면서 프로그래밍해야 합니다.

패키지(Package)

패키지는 클래스 파일의 묶음입니다. 프로그래밍을 하다보면 클래스를 한개만 사용하는게 아니라 여러개의 클래스가 계층구조를 가지고 사용하게 됩니다.

라이브러리도 마찬가지고요. 인텔리제이나 이클립스에서 새로운 프로젝트를 만들때 패키지도 만드는데 패키지 자체가 디렉토리 계층을 이루고 있습니다.

패키지가 복잡해 보이지만 이게 없다면 같은 이름의 클래스들간에 구분이 어려워집니다. 외부라이브러리를 사용할 때도 겹칠 수 있는데요.

라이브러리도 import 문으로 사용합니다.. 도트가 바로 디렉터리 계층구조입니다.

```
import java.time.*;  
System.out.println(LocalDate.now());
```

패키지의 사용법도 비슷합니다. 여기서는 Project 폴더가 Hello 인데 폴더로 따지면 Hello >> com 입니다.

```
package com;
```

```
com.Calculator newCal = new com.Calculator();
```

패키지는 클래스들이 계층구조를 이루고 있다. 정도는 알아둘 필요가 있습니다.

요약

이제 객체 지향 프로그래밍의 먼길을 떠날 시간입니다. 객체 지향 프로그래밍은 개념이라서 완전한 마스터가 없습니다.

한가지 팁은 너무 성급하게 생각하지 말라는 부분입니다. 객체, 클래스, 인스턴스 등 추상적 개념들이 많아서 어떤면에서는 C 언어보다 어려움이 있습니다.

Spring

스프링 프레임워크는 자바 생태계에서 가장 대중적인 응용프로그램 개발 프레임워크입니다.

의존성 주입(DI, Dependency Injection)과 제어의 역전(IOC, Inversion Of Control)은 스프링에서 가장 중요한 특징중 하나입니다. 이들로 인해서 좀더 결합도를 낮추는 방식으로 어플리케이션을 개발할 수 있습니다. 이러한 개발방식으로 개발한 응용프로그램은 단위테스트가 용이하기 때문에 보다 퀄리티 높은 프로그램을 개발할 수 있습니다.

DI 없는 예제

```
@RestController

public class MyController {

    private MyService service = new MyService();

    @RequestMapping("/welcome")

    public String welcome() {

        return service.retrieveWelcomeMessage();

    }

}
```

위의 예제를 생각해보면, MyController 는 MyService 라는 클래스에 의존합니다. 그래서 인스턴스를 얻기 위해 다음과 같이 코딩하곤 했을 겁니다.

```
MyService service = new MyService();
```

그렇게 되면 MyService 객체의 인스턴스를 얻었습니다. 이것은 객체간에 결합도가 올라간 모습입니다. 그런데 MyController 에 대한 단위테스트를 위해 Mock 객체를 생성했을 때, 어떻게 MyController 가 Mock 객체를 사용할 수 있을까요? 생각이 안나실 겁니다.

DI 있는 예제

```
@Component
```

```
public class MyService {
```

```
    public String retrieveWelcomeMessage() {  
        return "Welcome to InnovationM";  
    }  
}  
  
@RestController  
public class MyController {  
    @Autowired  
    private MyService service;  
  
    @RequestMapping("/welcome")  
    public String welcome() {  
        return service.retrieveWelcomeMessage();  
    }  
}
```

두개의 어노테이션으로 MyService 객체의 인스턴스를 쉽게 얻을 수 있습니다. 이는 결합도를 낮춘 것입니다. 스프링 프레임워크는 이렇게 일을 단순하게 만들기 위한 방법을 제공합니다.

@Component : 스프링의 BeanFactory라는 팩토리 패턴의 구현체에서 bean이라는 스프링프레임워크가 관리하는 객체가 있는데 해당 클래스를 그러한 bean 객체로 두어 스프링 관리하에 두겠다는 어노테이션.

@Autowired : 스프링 프레임워크에서 관리하는 Bean 객체와 같은 타입의 객체를 찾아서 자동으로 주입해주는 것. 해당 객체를 Bean으로 등록하지 않으면 주입해줄 수 없다.

그래서 스프링 프레임워크는 MyService 에 대한 bean 을 생성하

고 MyController 에 있는 service 변수에 주입해줍니다.

단위 테스트에서는 스프링프레임워크가 Mock 객체를 통해

MyService 를 MyController 에 있는 service 변수에 넣어주게 됩니다.

[2021.06.19 - \[이론\] - ORM 이란](#)

관점 지향 프로그래밍(AOP, Aspect Oriented Programming)은 스

프링 프레임워크에서 아주 강력한 기능입니다. 객체지향 프로그

래밍에서 중요 키포인트는 Class 입니다. 반면 AOP 에서는 관점

(Aspect)입니다. 예를 들어, 기존 프로젝트에 security 나 logging 등을 추가하고 싶을 때.. 기존 비즈니스 로직에는 손을 대지 않고 AOP 를 활용하여 추가할 수 있습니다. 특정 메소드가 끝날 때 호출할 수도 있고, 호출 직전, 메소드가 리턴한 직후 혹은 예외가 발생했을 때 등 여러가지 상황에 활용할 수 있습니다.

스프링은 스프링만의 ORM 을 가지고 있지 않습니다. 그러나 Hibernate, Apache Ibatis 등과 같은 ORM 과의 매우 우수한 통합 환경을 제공합니다.

간단히 말해, 스프링 프레임워크는 웹 어플리케이션을 개발하는데 결합도를 낮추는 방향의 개발방법을 제공한다고 말할 수 있습니다. Web 어플리케이션 개발은 스프링의 이러한 컨셉

(Dispatcher Servlet, ModelAndView, and View Resolver) 덕분에

쉬운 개발을 할 수 있게 되었습니다.

왜 스프링 부트의 필요해졌을까??

현재 당신이 스프링으로 이미 개발하고 있다면, 모든 기능을 다

갖춘 스프링 어플리케이션을 개발하면서 직면한 문제들에 대해서

되새겨 생각해보세요. 만일 생각을 못하는 분들을 위해 몇가지

꼬집어보겠습니다. Transaction Manager, Hibernate Datasource,

Entity Manager, Session Factory 와 같은 설정을 하는데에 어려움

이 많이 있었습니다. 최소한의 기능으로 Spring MVC 를 사용하여

기본 프로젝트를 셋팅하는데 개발자에게 너무 많은 시간이 걸렸습니다.

```
<bean
class="org.springframework.web.servlet.view.InternalR
esourceViewResolver">
  <property name="prefix">
    <value>/WEB-INF/views/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>

<mvc:resources mapping="/webjars/**"
location="/webjars/">

<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-
name>
    <param-value>/WEB-INF/my-servlet.xml</param-
value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Hibernate 를 사용할 때는 datasource, EntityManager 등과 같은 것을 설정해야 합니다. 이는 아래와 같습니다.

```
<bean id="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass" value="${db.driver}"
/>
    <property name="jdbcUrl" value="${db.url}" />
    <property name="user" value="${db.username}" />
    <property name="password" value="${db.password}"
/>
</bean>

<jdbc:initialize-database data-source="dataSource">
    <jdbc:script
location="classpath:config/schema.sql" />
    <jdbc:script location="classpath:config/data.sql"
/>
</jdbc:initialize-database>

<bean
class="org.springframework.orm.jpa.LocalContainerEnti
tyManagerFactoryBean"
    id="entityManagerFactory">
    <property name="persistenceUnitName"
value="hsqldb" />
    <property name="dataSource" ref="dataSource" />
</bean>

<bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionMana
ger">
    <property name="entityManagerFactory"
ref="entityManagerFactory" />
    <property name="dataSource" ref="dataSource" />
</bean>
```



```
<tx:annotation-driven transaction-  
manager="transactionManager"/>
```

스프링부트는 스프링의 이러한 설정에 관한 이슈를 어떻게 풀었을까??

1.

스프링부트는 자동설정(AutoConfiguration)을 이용하였고 어플리케이션 개발에 필요한 모든 내부 디펜던시를 관리합니다. 개발자가 해야하는건 단지 어플리케이션을 실행할 뿐입니다. 스프링의 jar 파일이 클래스 패스에 있는 경우 Spring Boot 는 Dispatcher Servlet 으로 자동 구성합니다. 마찬가지로 만약 Hibernate 의 jar 파일이 클래스 패스내에 존재한다면 이를 datasource 로 자동설

정하게 됩니다. 스프링부트는 미리설정된 스타터 프로젝트를 제공합니다.

2.

웹어플리케이션을 개발하는 동안, 우리는 사용하려는 jar, 사용할 jar 버전, 함께 연결하는 방법이 필요합니다. 모든 웹 어플리케이션에는 Spring MVC, Jackson Databind, Hibernate 코어 및 Log4j와 같은 유사한 요구사항들이 있습니다. 그래서 우리는 이러한 jar 들의 서로 호환되는 버전들을 따로 선택을 해주어야 했습니다. 이런 복잡도를 줄이기 위해서 스프링 부트는 SpringBoot Starter 라고 불리는 것을 도입했습니다.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>4.2.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.5.3</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.0.2.Final</version>
</dependency>
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```

Starter 는 편리하게 스프링 부트 어플리케이션에 포함할 수 있는
편리한 dependency 세트입니다. Spring 과 Hibernate 를 사용하려
면 프로젝트에 spring-boot-starter-data-jpa 라는 의존성을 포함
시키면 간단하게 사용할 수 있습니다.

스프링 부트 프로젝트의 의존성 관리

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```

아래의 스크린샷은 하나의 dependency 에 포함된 여러가지 패키

지들을 나타냅니다.

📁 Maven Dependencies

📦 spring-boot-2.0.2.RELEASE.jar - C:\Users\Ankit Kumar\.m2\repository\org\springframe

- ▷ 📦 org.springframework.boot
- ▷ 📦 org.springframework.boot.admin
- ▷ 📦 org.springframework.boot.ansi
- ▷ 📦 org.springframework.boot.builder
- ▷ 📦 org.springframework.boot.cloud
- ▷ 📦 org.springframework.boot.context
- ▷ 📦 org.springframework.boot.context.annotation
- ▷ 📦 org.springframework.boot.context.config
- ▷ 📦 org.springframework.boot.context.embedded.tomcat
- ▷ 📦 org.springframework.boot.context.event
- ▷ 📦 org.springframework.boot.context.logging
- ▷ 📦 org.springframework.boot.context.properties
- ▷ 📦 org.springframework.boot.context.properties.bind
- ▷ 📦 org.springframework.boot.context.properties.bind.handler
- ▷ 📦 org.springframework.boot.context.properties.bind.validation
- ▷ 📦 org.springframework.boot.context.properties.source
- ▷ 📦 org.springframework.boot.convert
- ▷ 📦 org.springframework.boot.diagnostics
- ▷ 📦 org.springframework.boot.diagnostics.analyzer
- ▷ 📦 org.springframework.boot.env
- ▷ 📦 org.springframework.boot.info
- ▷ 📦 org.springframework.boot.jackson
- ▷ 📦 org.springframework.boot.jdbc
- ▷ 📦 org.springframework.boot.jdbc.metadata
- ▷ 📦 org.springframework.boot.jms
- ▷ 📦 org.springframework.boot.json
- ▷ 📦 org.springframework.boot.jta.atomikos
- ▷ 📦 org.springframework.boot.jta.bitronix
- ▷ 📦 org.springframework.boot.jta.narayana
- ▷ 📦 org.springframework.boot.lang

스프링 부트 패키지 jar

당신이 한번 스타터 Dependency 를 추가하면 스프링부트 스타터 웹프로젝트가 pre-packaged 된 형태로 제공됩니다. 그로인해 개발자는 Dependency 관리와 호환버전에 대하여 고려할 필요가 없습니다.

스프링 부트 스타터 프로젝트 옵션들

응용프로그램의 종류에 따라 쉽고 빠르게 개발을 도와줄 수 있는 몇가지 스프링 스타터 프로젝트를 소개하겠습니다

* spring-boot-starter-web-services : SOAP 웹 서비스

* spring-boot-starter-web : Web, RESTful 응용프로그램

* spring-boot-starter-test : Unit testing, Integration Testing

* spring-boot-starter-jdbc : 기본적인 JDBC

* spring-boot-starter-hateoas : HATEOAS 기능을 서비스에 추가

* spring-boot-starter-security : 스프링 시큐리티를 이용한 인증과 권한

* spring-boot-starter-data-jpa : Spring Data JPA with Hibernate

* spring-boot-starter-cache : 스프링 프레임워크에 캐싱 지원 가능

* spring-boot-starter-data-rest : Spring Data REST 를 사용하여 간단한 REST 서비스 노출



Spring Boot = Spring Framework + Default Configuration +
Launcher + Plugins + Sever

Spring Boot 는 기본적으로 제공되는 기능들을 통해 빠른 시간에
어플리케이션을 개발할 수 있도록 합니다.

설정 커스터마이징을 통해 Spring Framework 의 기능을 모두 사
용할수 있어서 엔터프라이즈급 어플리케이션도 개발 가능합니다.

스프링 부트 시작하기

스프링 부트 개념과 활용을 공부하고 정리하는 포스트입니다.

스프링 부트 소개

스프링 부트는 제품 수준의 스프링 기반 애플리케이션을 만들 때 쉽고 빠르게 만들 수 있게 도와준다. 스프링 부트는 일일이 직접 설정하지 않아도 된다. 기본적으로 가장 많은 유저들이 사용하는 설정을 제공해 준다.

이러한 스프링 부트의 목표는 다음과 같다.

- 모든 스프링 개발에 있어서 더 빠르고 폭 넓은 사용성을 제공한다.
- **Convention over Configuration**(설정보다 관습)¹을 제공한다. 하지만 원하는대로 쉽고 빠르게 변경할 수 있다.
- 비즈니스 로직을 구현하는데 필요한 기능 뿐만 아니라 Non-functional features(비 기능적 요구 사항)도 제공한다.
- Code generation 과 Xml 을 요구하지 않는다.

요구 사항

스프링 부트 2.0.3 버전은 Java 8 이상, Spring Framework 5.0.7 이상, Maven 3.2 또는 Gradle 4 이상을 요구한다.

서블릿은 3.1 이상이면 된다.

부트 시작하기

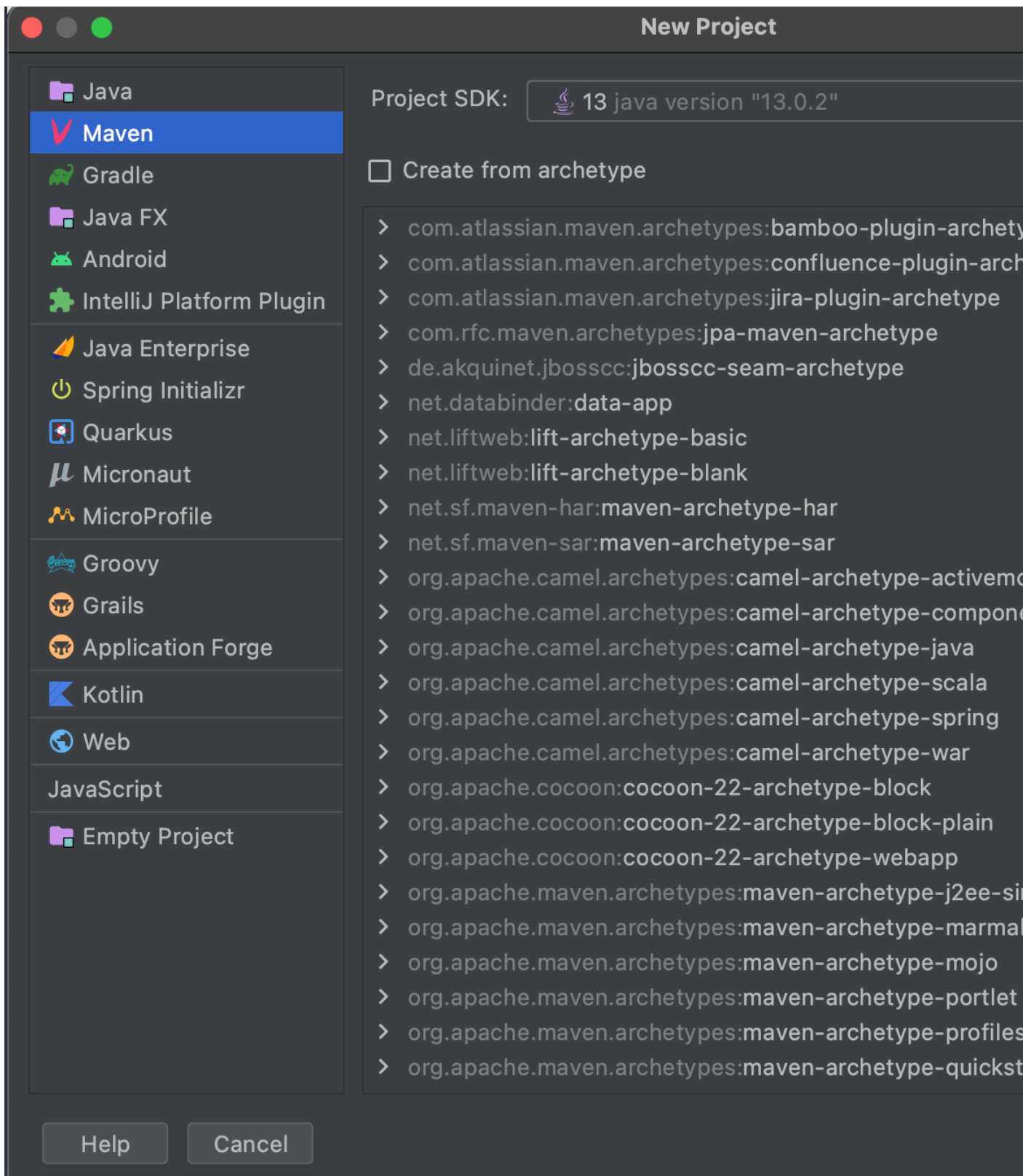
부트를 작성하는 방법은 공식 문서에서도 제공을 해준다.

1. spring.io에서 Projects - Spring Boot 선택
2. LEARN 의 Documentaion 에서 Reference Doc.을 선택
3. Getting Started - 3.Installing Spring Boot - 3.1 Installation Instructions for the Java Developer 선택

이 후 Maven 과 Gradle 중 선택하여 설치를 하면된다.

p.s. 요즘은 IDE 가 좋아서 알아서 다 설정을 해준다...

► IntelliJ



1. 새 프로젝트 생성에서 원하는 빌드 도구(Maven or Gradle)를 선택한다.
(익숙한 maven 을 선택했다.)

New Project

Name:

Location:

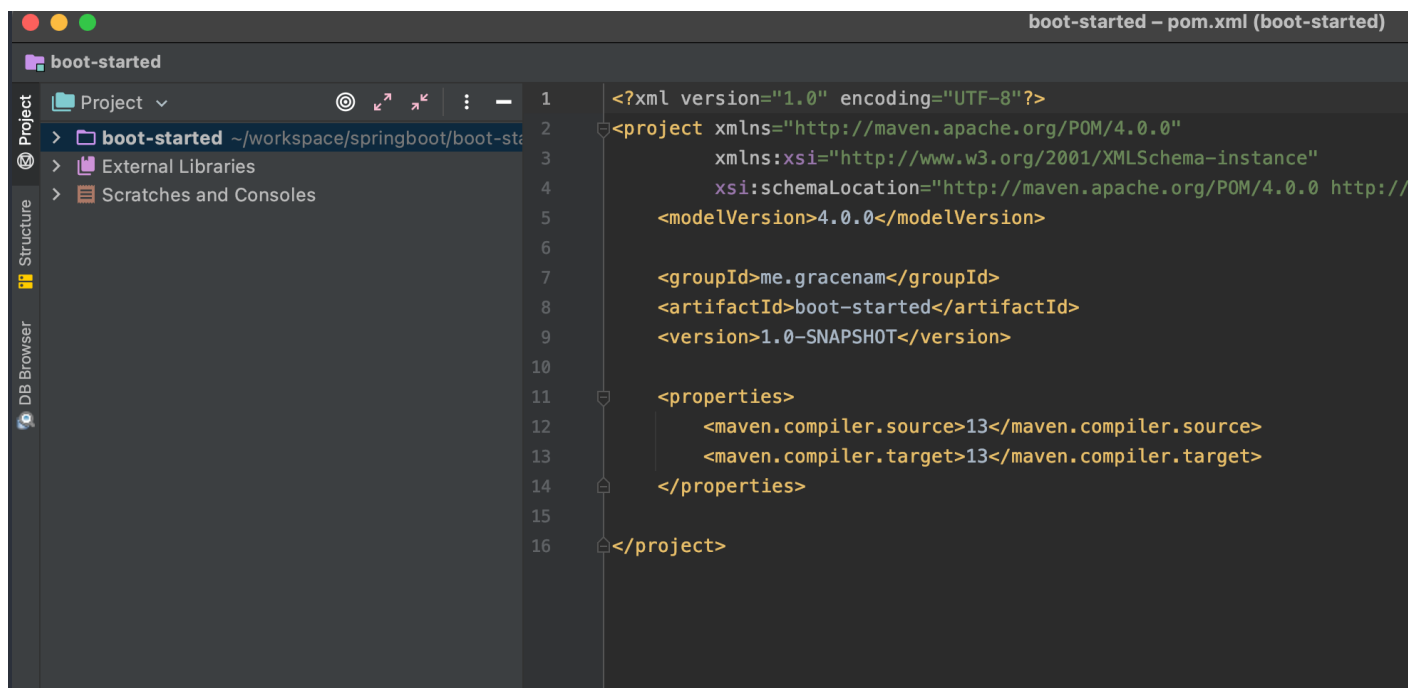
▼ Artifact Coordinates

GroupId:
The name of the artifact group, usually a company domain

ArtifactId:
The name of the artifact within the group, usually a project name

Version:

2. groupId 와 artifactId 를 정한 후 생성한다. 보통 프로젝트명은 artifactId 와 동일하다.



3. 짜잔! 위와 같이 나온다면 부트 생성이 완료된 것이다.

4. pom.xml 에 아래 코드들을 추가해준다.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.4.4</version>
</parent>

<properties>
  <maven.compiler.source>13</maven.compiler.source>
  <maven.compiler.target>13</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>2.4.4</version>
    </plugin>
  </plugins>
</build>
```

5. java 아래에 패키지를 하나 만들고(me.gracenam) 메인 클래스(Application.java)를 생성한다. 6. Application 클래스에 @SpringBootApplication 애노테이션을 추가하고 메인 메서드를 생성한 후 실행해서 아래와 같이 출력된다면 기본적인 애플리케이션 생성에 성공한 것이다.

The screenshot shows the IntelliJ IDEA IDE interface. The top toolbar has the Run button (a green play icon) highlighted. The Project view on the left shows the project structure: boot-started > src > main > java > me > gracenam > Application. The main editor displays the code for Application.java:

```
1 package me.gracenam;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Application.class, args);
11     }
12
13 }
14
```

Below the code editor, the Run console shows the output of the application. It starts with a Spring Boot logo and version (v2.4.4), followed by a series of log messages indicating the application is starting successfully:

```
2021-05-01 14:57:21.692 INFO 89199 --- [main] me.gracenam.Application : Starting Application
2021-05-01 14:57:21.704 INFO 89199 --- [main] me.gracenam.Application : No active profile set
2021-05-01 14:57:23.041 INFO 89199 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized wi
2021-05-01 14:57:23.052 INFO 89199 --- [main] o.apache.catalina.core.StandardService : Starting service [Tom
2021-05-01 14:57:23.052 INFO 89199 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engi
2021-05-01 14:57:23.136 INFO 89199 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring e
2021-05-01 14:57:23.136 INFO 89199 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationCo
2021-05-01 14:57:23.384 INFO 89199 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing Executor
2021-05-01 14:57:23.622 INFO 89199 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on por
2021-05-01 14:57:23.643 INFO 89199 --- [main] me.gracenam.Application : Started Application i
```

IntelliJ 얼티밋 버전을 사용한다면 IDE 에서 제공하는 Spring Initializr 를 사용해서 바로 생성할 수 있다. 의존성 주입도 자동으로 해준다!

스프링 부트 프로젝트 생성기

스프링 부트 프로젝트를 생성하는 또 다른 방법이 있다. 바로 스프링 부트 프로젝트 생성기(Spring Initializr)를 사용하는 것이다. 아래의 사이트에 들어가면 원하는 빌드 도구와 버전 등을 선택해서 프로젝트를 생성할 수 있다.

▶ [Spring Initializr](#)

생성한 프로젝트를 IDE 등으로 열어서 사용할 수 있다.

이 외에도 콘솔을 이용해서 생성할 수도 있지만 이 방법은 다소 귀찮으므로 생략하도록 하겠다.

Reference

- [스프링 부트 개념과 활용](#)
- [공식 문서](#)

1. 프레임워크를 사용하는 개발자가 취해야 하는 결정의 수를 줄이기 위한 소프트웨어 디자인 패러다임. 가장 많은 유저들이 사용하는 설정을 제공하는 것. [↪](#)

GitHub File

[GitHub 에서 원문보기](#)

Category

[spring boot](#)

Tags

[spring boot intro](#)

Comments

SPRING BOOT 의 다른 글

- [스프링 부트 활용 : 스프링 데이터 4 부](#) 17 Sep 2021
- [스프링 부트 활용 : 스프링 데이터 3 부](#) 16 Sep 2021
- [스프링 부트 활용 : 스프링 데이터 2 부](#) 15 Sep 2021
- [스프링 부트 활용 : 스프링 데이터 1 부](#) 14 Sep 2021
- [스프링 부트 활용 : 스프링 웹 MVC 11 부](#) 13 Sep 2021
- [스프링 부트 활용 : 스프링 웹 MVC 10 부](#) 10 Sep 2021
- [스프링 부트 활용 : 스프링 웹 MVC 9 부](#) 09 Sep 2021
- [스프링 부트 활용 : 스프링 웹 MVC 8 부](#) 08 Sep 2021
- [스프링 부트 활용 : 스프링 웹 MVC 7 부](#) 03 Sep 2021
- [스프링 부트 활용 : 스프링 웹 MVC 6 부](#) 05 Aug 2021
- [스프링 부트 활용 : 스프링 웹 MVC 5 부](#) 27 Jun 2021
- [스프링 부트 활용 : 스프링 웹 MVC 4 부](#) 13 Jun 2021

- [스프링 부트 활용 : 스프링 웹 MVC 3 부](#) 06 Jun 2021
- [스프링 부트 활용 : 스프링 웹 MVC 2 부](#) 05 Jun 2021
- [스프링 부트 활용 : 스프링 웹 MVC 1 부](#) 30 May 2021
- [스프링 부트 활용 : Spring-Boot-Devtools](#) 30 May 2021
- [스프링 부트 활용 : 테스트](#) 26 May 2021
- [스프링 부트 활용 : 로깅 2 부](#) 25 May 2021
- [스프링 부트 활용 : 로깅 1 부](#) 23 May 2021
- [스프링 부트 활용 : 프로파일](#) 23 May 2021
- [스프링 부트 활용 : 외부 설정 2 부](#) 21 May 2021
- [스프링 부트 활용 : 외부 설정 1 부](#) 15 May 2021
- [스프링 부트 활용 : SpringApplication](#) 11 May 2021
- [스프링 부트 원리 : 정리](#) 06 May 2021
- [스프링 부트 원리 : 독립적으로 실행 가능한 JAR](#) 06 May 2021
- [스프링 부트 원리 : 내장 웹 서버](#) 04 May 2021
- [스프링 부트 원리 : 자동 설정](#) 02 May 2021
- [스프링 부트 원리 : 의존성](#) 01 May 2021
- [스프링 부트 시작하기](#) 01 May 2021