

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPTO

1 차 세 미 나

SHOUT OUR PASSION TOGETHER
SOPT

모든 설명은 Window 10 설명 기준입니다.

01

JAVA 기초

1. Java 프로젝트 생성
2. 객체와 클래스
3. Builder 패턴
4. Interface
5. 상속
6. Generic
7. Exception
8. Annotation

02

Spring Boot 시작하기

1. Spring 소개
2. Spring Boot 프로젝트 생성
3. Spring Boot 구성요소
4. Web Application 실행
5. Controller 생성 맛보기

○

01

○

Java 기초

객체 지향 프로그래밍(Object-Oriented Programming)

1. 프로그램을 수 많은 객체라는 기본 단위로 나누고, 이 객체들의 상호 작용으로 프로그램을 구성하는 방식
2. 로직을 상태(state)와 행위(behavior)로 이루어진 객체로 만드는 것이다.
3. 레고 블록처럼 객체를 조립해서 하나의 프로그램을 만드는 것
4. 코드의 재사용성을 증가시키는 목적
5. 유지 보수를 감소시키는 장점
6. 강한 응집력(Strong Cohesion), 약한 결합력(Weak Coupling)을 지향

객체 지향 프로그래밍(Object-Oriented Programming)의 특징

1. 캡슐화(Encapsulation)

프로그램의 세부 구현을 외부로 드러나지 않도록 특정 클래스 내부로 감추는 것
접근 제어 지시자를 통해 외부에서 접근을 제어한다.

2. 상속(Inheritance)

자식 클래스가 부모 클래스의 특성과 기능을 그대로 물려받는 것을 말한다.
자식 클래스에서 재 정의를 통해 부모 클래스의 메소드를 사용할 수 있다.

3. 다형성(Polymorphism)

하나의 메소드, 클래스가 상황에 따라 다른 의미로 해석될 수 있다.
Overloading은 다형성의 가장 대표적인 예

다형성(Polymorphism)

1. 다형성의 조건 (Ex. List)

1. 공통의 부모 : 클래스 계층 구조(상속 관계)

List(부모), ArrayList(자식), LinkedList(자식)

1. 공통의 메소드 재정의 : 메소드 재정의, 동적 바인딩

add 메소드

2. 부모 타입의 변수로 호출 : 업 캐스팅 후 재정의 된 메소드 호출

List list1 = new ArrayList();

List list2 = new ArrayList();

```
List list1 = new ArrayList();  
List list2 = new LinkedList();  
list1.add(1);  
list2.add(1);
```

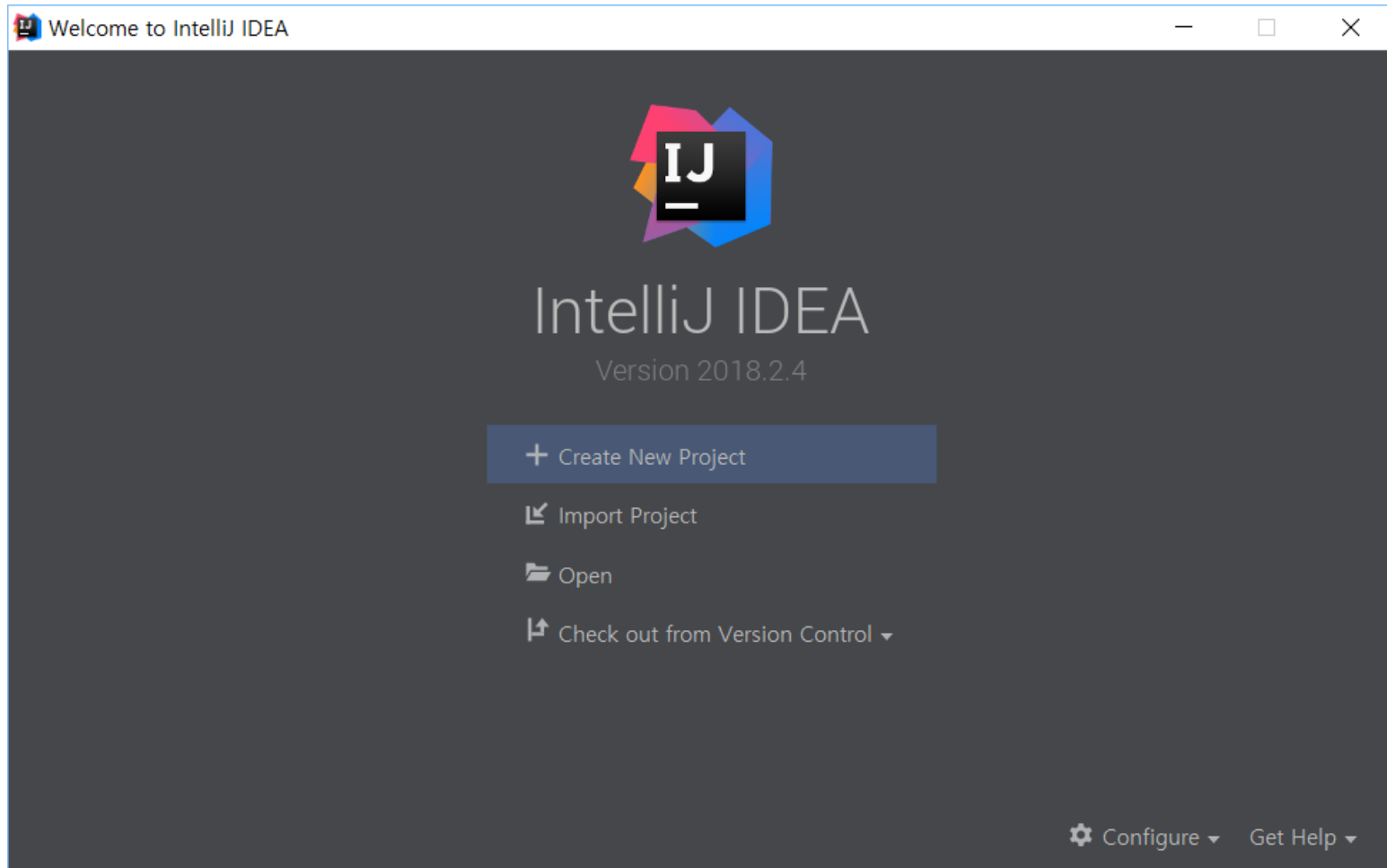
Upcasting(업 캐스팅) : 하위 클래스에서 상위 클래스의 형으로 캐스팅(형 변환) 되는 것

○

01

○

Java 프로젝트 생성

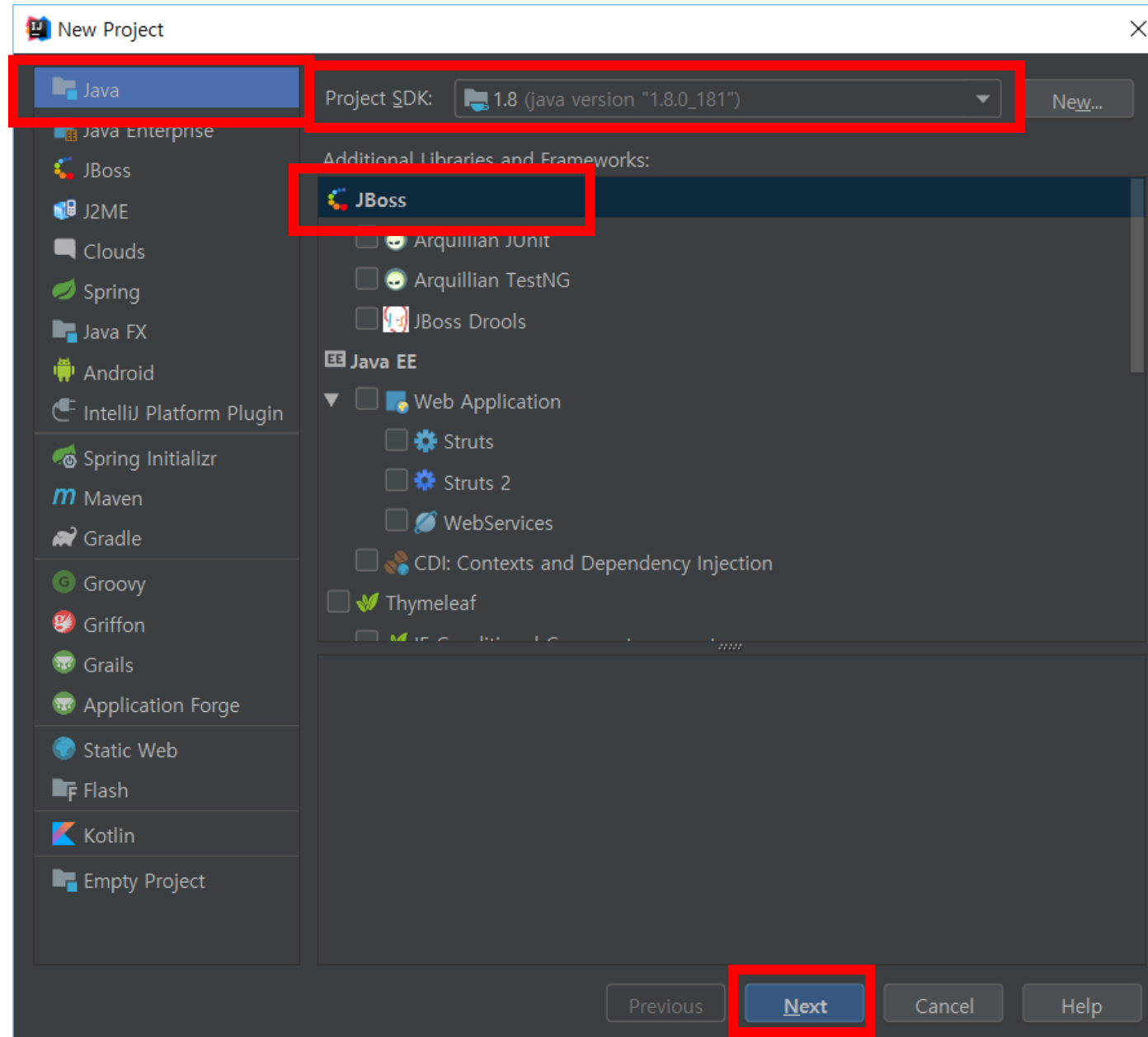


1. Create New Project

01 Java 기초

Java 프로젝트 생성

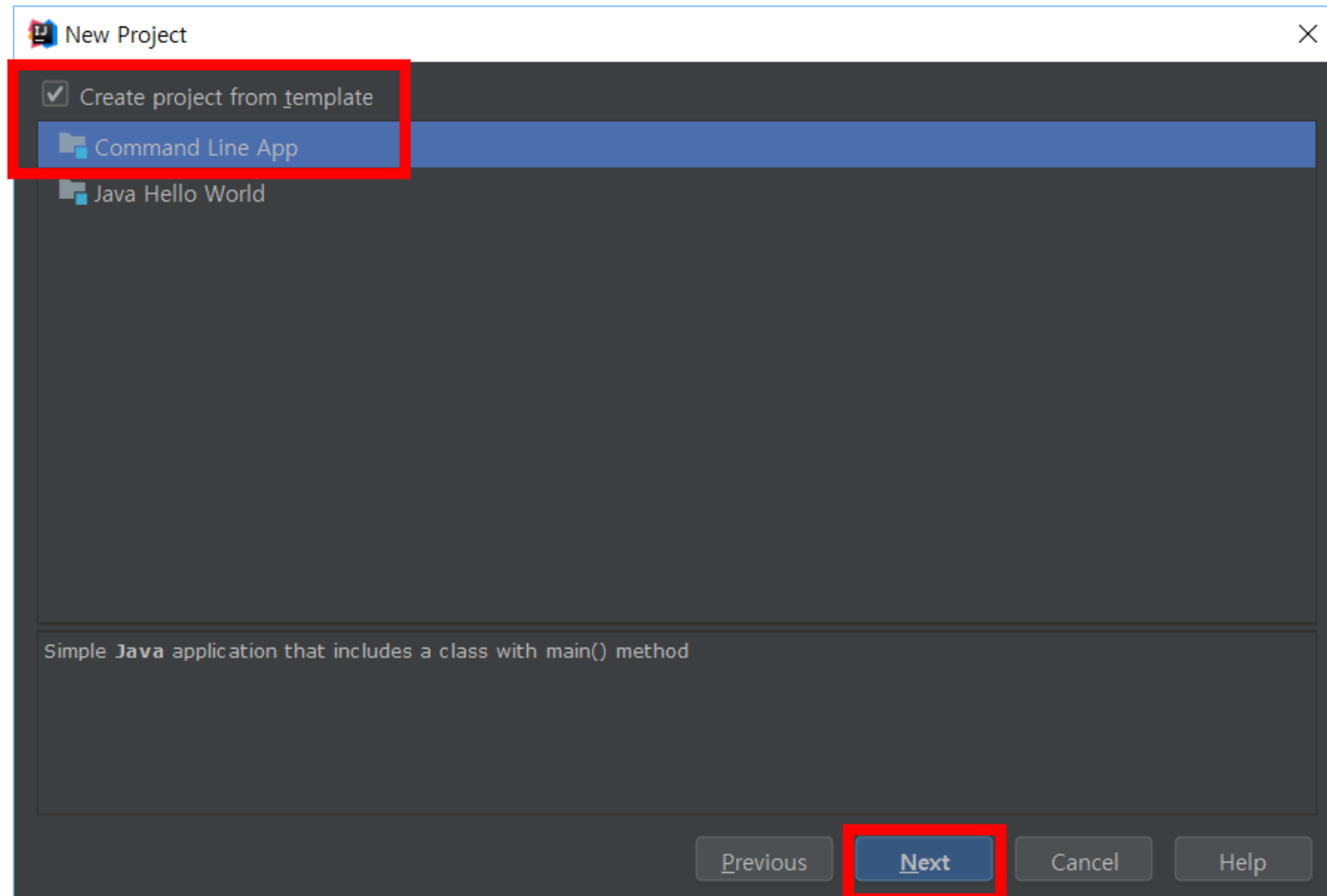
1. Java 선택
2. Java 버전 확인
(Project SDK : 1.8)
3. Java 선택 후 자동으로
선택되는 JBoss
4. Next



01 Java 기초

Java 프로젝트 생성

1. Create project from template 선택
2. Command Line App 선택
3. Next



01 Java 기초

Java 프로젝트 생성

1. 프로젝트 이름 설정
2. 프로젝트 저장 경로
3. 기본 패키지 이름 설정
4. Finish

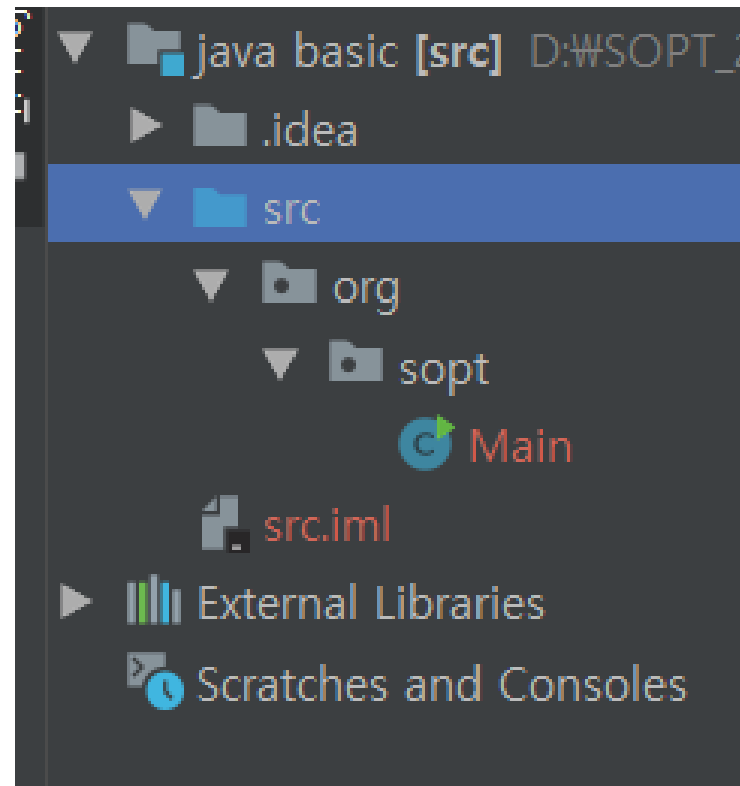
New Project

Project name: seminar

Project location: D:\WSOPT_23\repo\srd

Base package: org.sopt

Previous Finish Cancel Help



1. 프로젝트 기본 디렉토리 구조



01



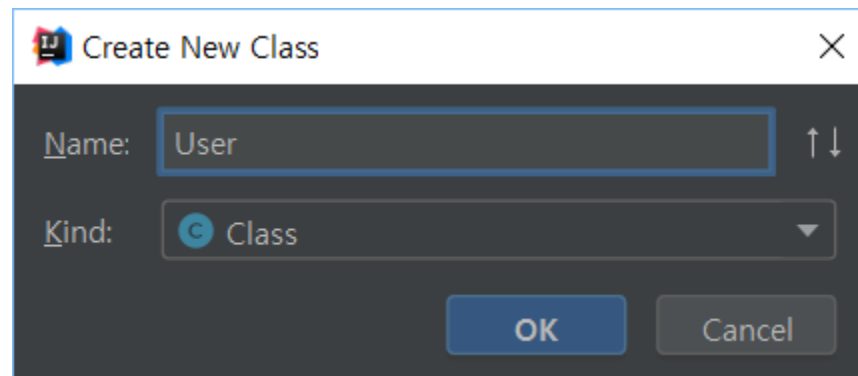
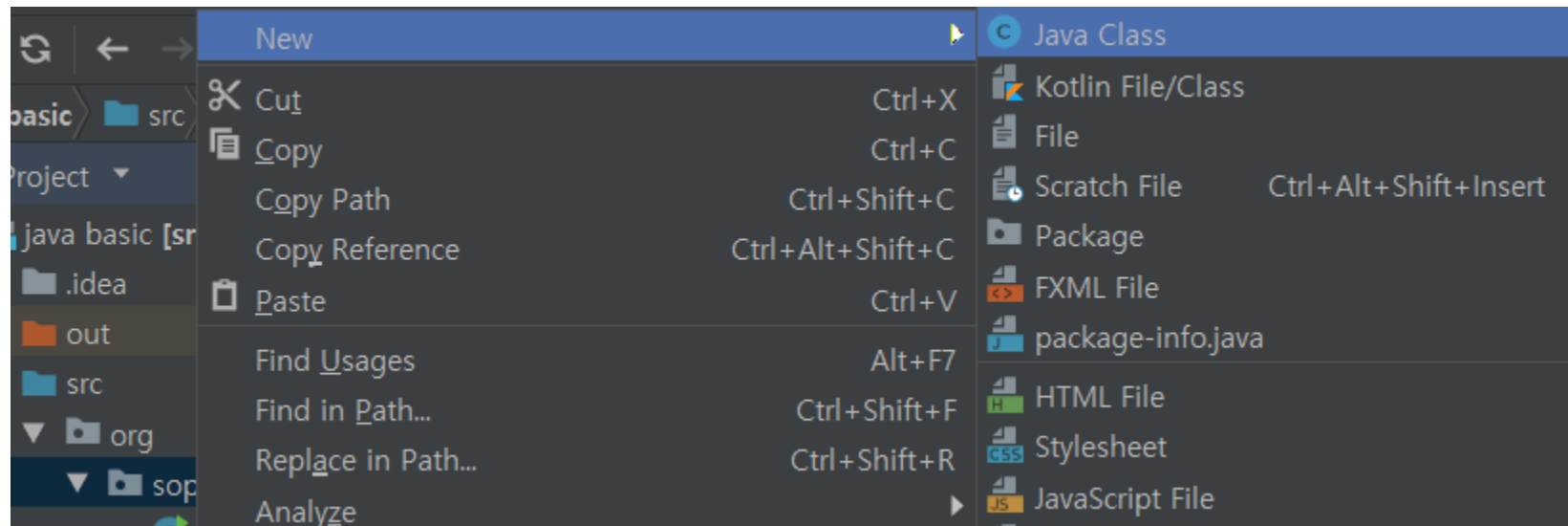
객체와 클래스

클래스(Class)

1. 객체(Object)를 만드는 기능을 한다.
2. 붕어빵 틀(클래스)과 붕어빵(객체)
3. 여러가지 필드(Field)와 메소드(Method)를 가진다.
4. JVM 시작 시 RunTime Data Area의 Method(Static) Area에 할당된다.

```
public class User {  
}
```

간단한 형태의 클래스



1. 클래스를 생성할 패키지 선택
2. 마우스 우 클릭 -> New -> Java Class
3. 클래스 생성

```
public class User {  
  
    //Field = 속성 = 상태 = 멤버 변수  
  
    //회원 고유 번호  
    private int idx;  
  
    //이름  
    private String name;  
  
    //파트  
    private String part;  
}
```

1. Field는 객체의 속성, 상태를 표현한다.
2. 접근 제어 지시자는 private
3. Field(멤버 변수)는 JVM 내의 Method(Static) Area에 할당된다.


```
public class User {  
  
    //Field = 속성 = 상태 = 멤버 변수  
  
    //회원 고유 번호  
    private int idx;  
  
    //이름  
    private String name;  
  
    //파트  
    private String part;  
  
    //인사 메소드  
    public void hello() {  
        System.out.println("안녕하세요. 저는 " + part + "파트 " + name + "입니다.");  
    }  
}
```

1. Method
2. 객체의 동작, 행위, 기능
3. Method(메소드)는 JVM 내의 Method(Static) Area에 할당된다

```
/**
 * 회원 객체 생성자
 * Default 생성자
 * 매개변수가 없는 생성자
 */
public User() {
    this.idx = 9;
    this.name = "배다슬";
    this.part = "Server";
}

/**
 * 회원 객체 생성자
 * 매개변수가 있는 생성자
 *
 * @param idx      회원 고유 번호
 * @param name     이름
 * @param part     파트
 */
public User(final int idx, final String name, final String part) {
    this.idx = idx;
    this.name = name;
    this.part = part;
}
```

1. Constructor(생성자)
2. Instance 생성해주는 특수한 메소드
3. 값을 반환하지 않는다.
4. 생성자의 이름은 클래스의 이름과 동일하다.
5. 접근 제어 지시자는 반드시 public

```
public class Main {  
    public static void main(String[] args) {  
        //Default 생성자로 객체 생성  
        final User user1 = new User();  
        //매개변수를 통해 객체 생성  
        final User user2 = new User(2, "류지훈", "기획");  
    }  
}
```

Instance 생성

1. User 타입의 user 객체 변수 생성(현재는 비어 있다.)
2. new 키워드를 통해 JVM RunTime Data Area내의 Heap 영역에 Instance를 저장할 메모리를 할당
3. User 생성자를 통해 객체를 생성(초기화)
4. 생성자가 종료되면 new 연산자는 객체의 참조 값을 반환한다.
5. user 변수에 User 객체의 참조 값 할당

```
public class Main {  
    public static void main(String[] args) {  
        //Default 생성자로 객체 생성  
        final User user1 = new User();  
        //매개변수를 통해 객체 생성  
        final User user2 = new User(2, "류지훈", "기획");  
        user1.hello();  
        user2.hello();  
    }  
}
```

안녕하세요. 저는 Server파트 배다슬입니다.

안녕하세요. 저는 기획파트 류지훈입니다.

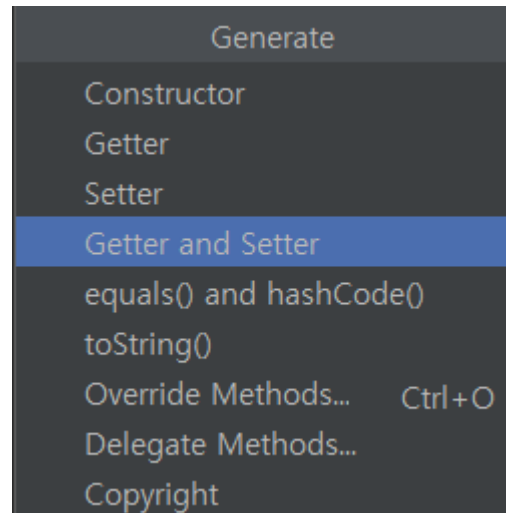
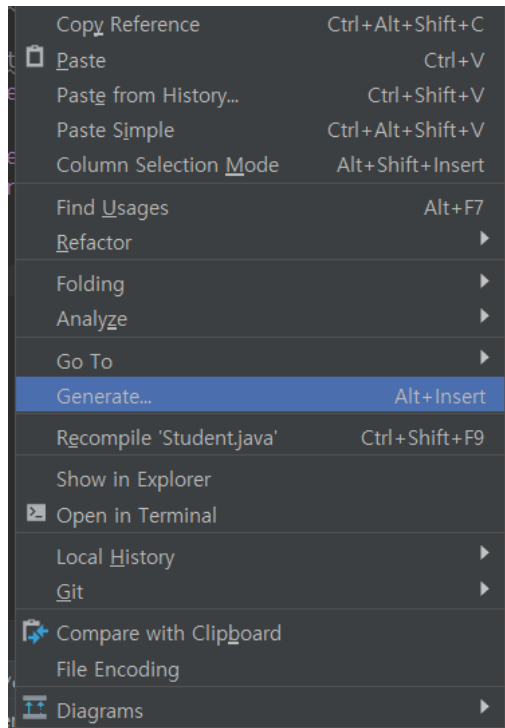
Process finished with exit code 0

객체의 메소드 실행

은닉화

클래스의 Field는 Private로 접근 제어지시자로 지정한 뒤
Getter, Setter를 통해 Filed의 값을 변경, 호출 한다.

1. Getter : 값을 호출
2. Setter : 값을 변경



```
public int getIdx() {  
    return idx;  
}  
  
public void setIdx(final int idx) {  
    this.idx = idx;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(final String name) {  
    this.name = name;  
}  
  
public String getPart() {  
    return part;  
}  
  
public void setPart(final String part) {  
    this.part = part;  
}
```

접근 제어 지시자

지시자	클래스 내부	동일 패키지	상속	이외
Private	O	X	X	X
Default	O	O	X	X
Protected	O	O	O	X
public	O	O	O	O



01



Builder 패턴

01 Java 기초

Builder 패턴

USER						
사용자						
PK	AI	FK	Null	Logical Name	Name	Type
✓	✓	+		사용자 고유 id	user_idx	INT
		+	✓	이메일	email	VARCHAR(45)
		+	✓	비밀번호	password	VARCHAR(45)
		+	✓	이름	name	VARCHAR(45)
		+	✓	포인트	point	INT
		+	✓	가입 날짜	signup_at	DATETIME
		+	✓	프로필 사진	profile_url	TEXT
		+	✓	배경 사진	background_url	TEXT
		+	✓	역할	position	VARCHAR(100)
		+	✓	자기소개	introduce	TEXT
		+	✓	포트폴리오 url	portfolio_url	TEXT
		+	✓	목적	aim	VARCHAR(100)
		+	✓	분야	department	VARCHAR(100)
		+	✓	지역	area	VARCHAR(100)
		+	✓	fcm토큰	fcm_token	TEXT

실제 App-Jam 시 DB 모델링

```
public class User2 {
    private Integer userIdx;
    private String email;
    private String password;
    private String name;
    private Integer point;
    private Timestamp signupAt;
    private String profileUrl;
    private String backgroundUrl;
    private String position;
    private String introduce;
    private String portfolioUrl;
    private String aim;
    private String department;
    private String area;
    private String fcmToken;
    public User2(
        final Integer userIdx,
        final String email,
        final String password,
        final String name,
        final Integer point,
        final Timestamp signupAt,
        final String profileUrl,
        final String backgroundUrl,
        final String position,
        final String introduce,
        final String portfolioUrl,
        final String aim,
        final String department,
        final String area,
        final String fcmToken) {
        this.userIdx = userIdx;
        this.email = email;
        this.password = password;
        this.name = name;
        this.point = point;
        this.signupAt = signupAt;
        this.profileUrl = profileUrl;
        this.backgroundUrl = backgroundUrl;
        this.position = position;
        this.introduce = introduce;
        this.portfolioUrl = portfolioUrl;
        this.aim = aim;
        this.department = department;
        this.area = area;
        this.fcmToken = fcmToken;
    }
}
```



```
final User2 cowalkerPM = new User2(1, "pm@cowalker.com", "1234", "이충엽", 0, null,  
    "ImgPath", "개설자", "안녕하세요.", "ImgPath", "ImgPath", "메인기획자", null, "의정부", "tokenValue");  
  
final User2 cowalkerSubPM = new User2(2, "subPm@cowalekr.com", "5678", "길태환", 0, null,  
    "ImgPath", "참여자", "안녕하세요.", "ImgPath", "ImgPath", "서브기획자", null, "의정부", "tokenValue");  
  
final User2 designer1 = new User2(3, "designer1@cowalekr.com", "5678", "홍지연", 0, null,  
    null, "참여자", null, null, null, "디자이너", null, null, "tokenValue");
```

끔찍한 결과

15개의 파라미터를 정확한 위치에 위치시켜야 한다.

Builder Pattern

1. 생성자에 들어갈 매개변수를 차례대로 받아들이고 모든 매개변수를 받은 뒤에 이 변수들을 통합하여 한번에 사용한다.
2. 데이터의 순서에 상관 없이 객체를 만든다.
3. 사용자가 봤을 때 명시적이고 명확하게 이해할 수 있어야 한다.
4. 불필요한 생성자를 만들지 않고 객체를 만든다.

```
public class UserBuilder {  
  
    //회원 고유 번호  
    private int idx;  
  
    //이름  
    private String name;  
  
    //파트  
    private String part;  
  
    public UserBuilder setIdx(final int idx) {  
        this.idx = idx;  
        return this;  
    }  
  
    public UserBuilder setName(final String name) {  
        this.name = name;  
        return this;  
    }  
  
    public UserBuilder setPart(final String part) {  
        this.part = part;  
        return this;  
    }  
  
    /**  
     * User Builder  
     * @return User객체  
     */  
    public User build() {  
        return new User(this.idx, this.name, this.part);  
    }  
}
```

```
final UserBuilder userBuilder = new UserBuilder();  
  
final User user3 = userBuilder  
    .setIdx(3)  
    .setName("남윤환")  
    .setPart("Android")  
    //build 메소드를 통해 객체가 생성되서 반환된다.  
    .build();
```

안녕하세요. 저는 Server파트 배다슬입니다.
안녕하세요. 저는 기획파트 류지훈입니다.
안녕하세요. 저는 Android파트 남윤환입니다.

간결하게 가능

조금 더 명시적이고 명확하게 객체를 만들 수 있다.

```
final User2Builder user2Builder = new User2Builder();

final User2 designer2 = user2Builder
    .setUserIdx(4)
    .setEmail("designer2@cowalekr.com")
    .setPassword("0000")
    .setName("박채원")
    .setPoint(0)
    .setSignupAt(null)
    .setProfileUrl("ImgPath")
    .setPosition("디자이너")
    .setIntroduce("웹 디자이너입니다.")
    .setPortfolioUrl("ImgPath")
    .setAim(null)
    .setArea("수원")
    .setBackgroundUrl(null)
    .setFcmToken(null)
    .setDepartment(null)
    .build();
}
```

SHOUT · OUR · PASSION · TOGETHER



01



Interface

Interface(인터페이스)

1. 클래스들이 구현해야 하는 동작을 지정하는데 사용되는 추상형
2. 규약, 규제
3. 추상 클래스의 극단적인 경우
4. 인터페이스도 상속이 가능하고, 클래스와 달리 다중 상속이 가능하다.
5. 객체의 내부 구조를 알 필요 없이 인터페이스의 메소드만 알고 있으면 된다.
6. 다형성을 이용한 인터페이스의 구현 객체를 손쉽게 교체할 수 있다.

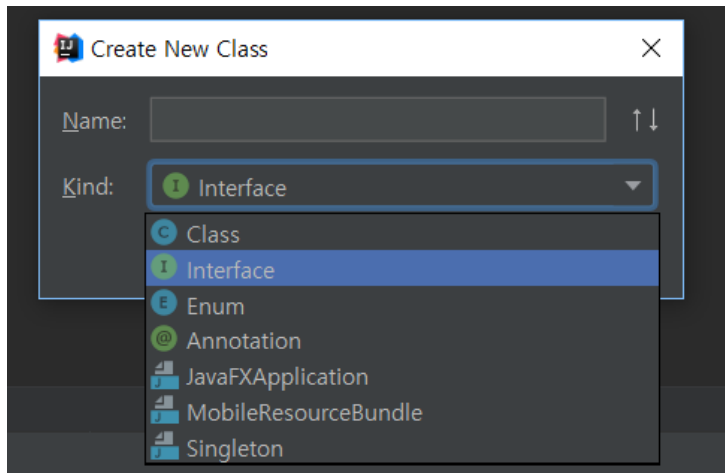
```
public interface UserInterface {  
}
```

간단한 형태의 인터페이스

Java 8에서의 Interface

1. 상수 필드(public static final) : 기존
2. 추상 메소드(public abstract) : 기존
3. 정적 메소드(public static) : 기존
4. 디폴트 메소드(public default) : 추가

이전에 개발한 구현 클래스를 그대로 사용하고, 변경하지 않으면서 새롭게 개발하는 클래스는 디폴트 메소드를 이용해 새로운 기능을 개발할 수 있다.



```
public interface UserInfo {  
    //상수 필드  
    //public static final 생략이 가능하다.  
    public static final String DEPT = "소프트웨어공학과";  
    int DEPTNO = 320;  
  
    //추상 메소드  
    //public abstract 생략이 가능하다.  
    public abstract void getInfo();  
  
    String getDept();  
  
    //디폴트 메소드  
    //메소드 내용까지 구현이 가능하다.  
    //해당 인터페이스 구현시 디폴트 메소드를 override해서 사용할 수 있다.  
    //public 생략이 가능하다.  
    public default void setState(final boolean state) {  
        if (state) {  
            System.out.println("휴학중");  
        } else {  
            System.out.println("재학중");  
        }  
    }  
  
    //정적 메소드  
    //public 생략이 가능하다.  
    public static void world() {  
        System.out.println("World!");  
    }  
}
```



```
public class UserServiceImpl implements UserInterface {  
  
    //필수 구현 메소드  
    @Override  
    public void getInfo() {  
  
    }  
  
    //필수 구현 메소드  
    @Override  
    public String getDept() {  
        return null;  
    }  
  
    //선택 구현 메소드  
    @Override  
    public void setState(boolean state) {  
        //재정의해서 사용할 수 있다.  
    }  
}
```

Interface 구현체

```
public class UserServiceImpl implements UserInterface, Intreface1, Interface2 {  
    // 필수 구현 메서드
```

다중 Interface 구현체도 가능하다.

S H O U T · O U R · P A S S I O N · T O G E T H E R



01



상

속

상속(Inheritance)

1. 객체지향의 재활용성을 극대화 시킨 프로그래밍 기법, 동시에 객체 지향을 복잡하게 하는 주요 원인
2. 부모 클래스의 내용을 자식 클래스가 물려 받는 것
3. 하나의 부모 클래스는 여러 개의 자식 클래스를 가질 수 있지만, 하나의 자식 클래스는 오직 하나의 부모 클래스를 가져야 한다.
4. 자식 클래스는 부모 클래스로부터 받은 메소드를 재정의 해서 사용할 수 있다.

```
public class Student extends Person {  
}
```

간단한 형태의 상속

```
public class Person {  
    private String name;  
  
    public Person(final String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

부모 클래스

```
public class Student extends Person {  
    private String deaprtment;  
  
    public Student(final String name, final String deaprtment) {  
        //부모 클래스의 생성자 사용  
        super(name);  
        this.deaprtment = deaprtment;  
    }  
  
    public String getDeaprtment() {  
        return deaprtment;  
    }  
  
    public void setDeaprtment(String deaprtment) {  
        this.deaprtment = deaprtment;  
    }  
}
```

자식 클래스

```
final Student student = new Student("배다슬", "소프트웨어공학과");  
//부모 클래스의 메소드  
System.out.println(student.getName());  
//자식 클래스의 메소드  
System.out.println(student.getDeaprtment());
```

배다슬
소프트웨어공학과

SHOUT · OUR · PASSION · TOGETHER

○

01

○

Generic

Generic(제네릭)

1. 클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법
2. 다양한 타입의 객체들을 다루는 메소드나, 컬렉션 클래스에서 컴파일 시 타입 체크를 해주는 기능이다.
3. 타입 안정성을 제공한다.
4. 타입체크와 형변환을 생략할 수 있으므로 코드가 간결해 진다.
5. 컴파일 단계에서 오류를 찾을 수 있다.
6. 중복 제거가 가능하다.

```
public class DefaultRes<T> {  
}
```

간단한 형태의 제네릭


```
public class DefaultRes<T> {  
    private T responseData;  
  
    public DefaultRes(final T responseData) {  
        this.responseData = responseData;  
    }  
}
```

```
DefaultRes<Student> defaultRes1 = new DefaultRes<>(student);  
DefaultRes<User> defaultRes2 = new DefaultRes<>(user1);  
DefaultRes<User2> defaultRes3 = new DefaultRes<>(cowalwerPM);
```

1. 각각의 인스턴스를 생성할 때 <> 사이에 어떤 데이터 타입을 사용했느냐에 따라 다르다.

```
public class DefaultRes<T, S> {  
    private T responseData;  
    private S statusCode;  
    |  
}
```

1. 복수의 제네릭을 사용 가능 (어떠한 문자도 사용할 수 있다.)

```
public class DefaultRes<T extends Person> {  
    private T responseData;
```

1. 제네릭으로 올 수 있는 데이터 타입을 특정 클래스의 자식으로 한정할 수도 있다.

```
public interface CRUDService<T> {  
    DefaultRes<T> findOne(final int id) throws InternalServerError ;  
    DefaultRes<T> save (final T t) throws InternalServerError ;  
    DefaultRes<T> update (final int id, T t) throws InternalServerError ;  
    DefaultRes<T> delete (final int id) throws InternalServerError ;  
}
```

제네릭과 인터페이스 활용 예

○

01

○

Exception

Exception(예외)

1. 오류가 발생하지 않는 프로그램은 없다.
2. 기능이 많아질수록 오류가 발생할 확률도 높아진다.
3. 프로그램을 만든 개발자가 상정한 정상적인 처리에서 벗어나는 경우에 이를 처리하기 위한 방법

```
try {  
  
} catch (Exception e) {  
  
}
```

간단한 형태의 예외

```
Connection connection = null;
PreparedStatement statement = null;
ResultSet resultSet = null;
try {
    connection = DB.getConnection("student1");
    statement = connection.prepareStatement(sql);
    resultSet = statement.executeQuery();
    while (resultSet.next()) {
        list.add(makeUserObj(resultSet));
    }
    return list;
}
//해당 에러 발생시 실행
catch (SQLException e) {
    printError(e);
}
//무조건 실행
finally {
    connection.close();
    statement.close();
    resultSet.close();
}
```

```
//java 7에서 추가된 try-with-resources  
//자원이 자동으로 release 된다.  
try (Connection connection = DB.getConnection("student1");  
    PreparedStatement statement = connection.prepareStatement(sql);  
    ResultSet resultSet = statement.executeQuery()) {  
    while (resultSet.next()) {  
        list.add(makeUserObj(resultSet));  
    }  
    return list;  
}  
//해당 예외가 발생시 실행  
catch (SQLException e) {  
    printError(e);  
}
```

1. Java 7에서 추가된 Try-With-Resources 문법
2. 자원을 자동으로 Release 해준다.


```
public class InternalServerError extends Exception {  
  
    private static final long serialVersionUID = 1L;  
  
    public InternalServerError(final String msg) {  
        super(msg);  
    }  
}
```

사용자 정의 예외

```
if (res.getStatusCode() == StatusCode.SERVICE_UNAVAILABLE) {  
    //circuit breaker 강제 발생  
    throw new InternalServerError("Remote Server Error");  
}
```

예외의 사용



01



Annotation

Annotation(어노테이션)

1. Java 5에서 추가된 문법
2. Java 코드에 마치 주석 처럼 달아 특수한 의미를 부여한다.
3. Compile Time, Run Time시에 해석될 수 있다.
4. 자바, Spring이 제공해 주는 것도 있고, 사용자가 직접 만들 수도 있다.

```
@Override  
public String toString() {  
    return "";  
}
```

@Override Annotation의 예, toString()

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Auth {
}
```

사용자 정의 Annotation (@Auth)

1. @Target : Annotation이 적용할 위치를 선택
 1. PACKAGE, TYPE, FIELD, METHOD..등
2. @Retention : Java Compile가 Annotation을 다루는 방법을 설정
 1. 특정 시점까지 영향을 미치는 지를 결정
 2. SOURCE(컴파일 전까지), CLASS(클래스 참조할 때 까지),
RUNTIME(JVM에 의해 계속 참조 가능, Reflection사용)

```
@GetMapping("/{user_idx}")  
@Auth  
public ResponseEntity<DefaultRes<UserRes>> ge
```

사용자 정의 Annotation (@Auth) 사용

```
@Autowired  
public UserController(final UserService userService) { this.userService = userService; }
```

```
@RestController  
public class FirstController {  
  
    @GetMapping("/hello")  
    public String Hello() { return "Hello world!"; }  
}
```

Spring의 Annotation 사용



02



Spring Boot 시작하기

○

02

○

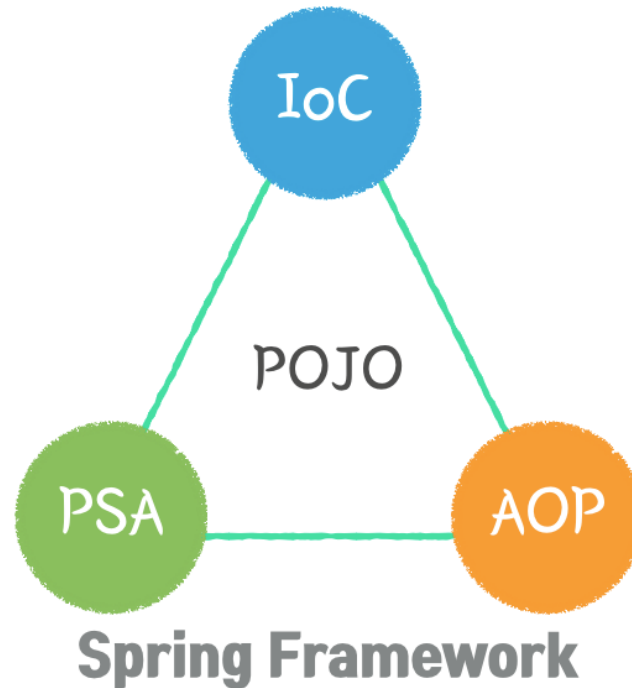
Spring 소개



spring

by Pivotal™

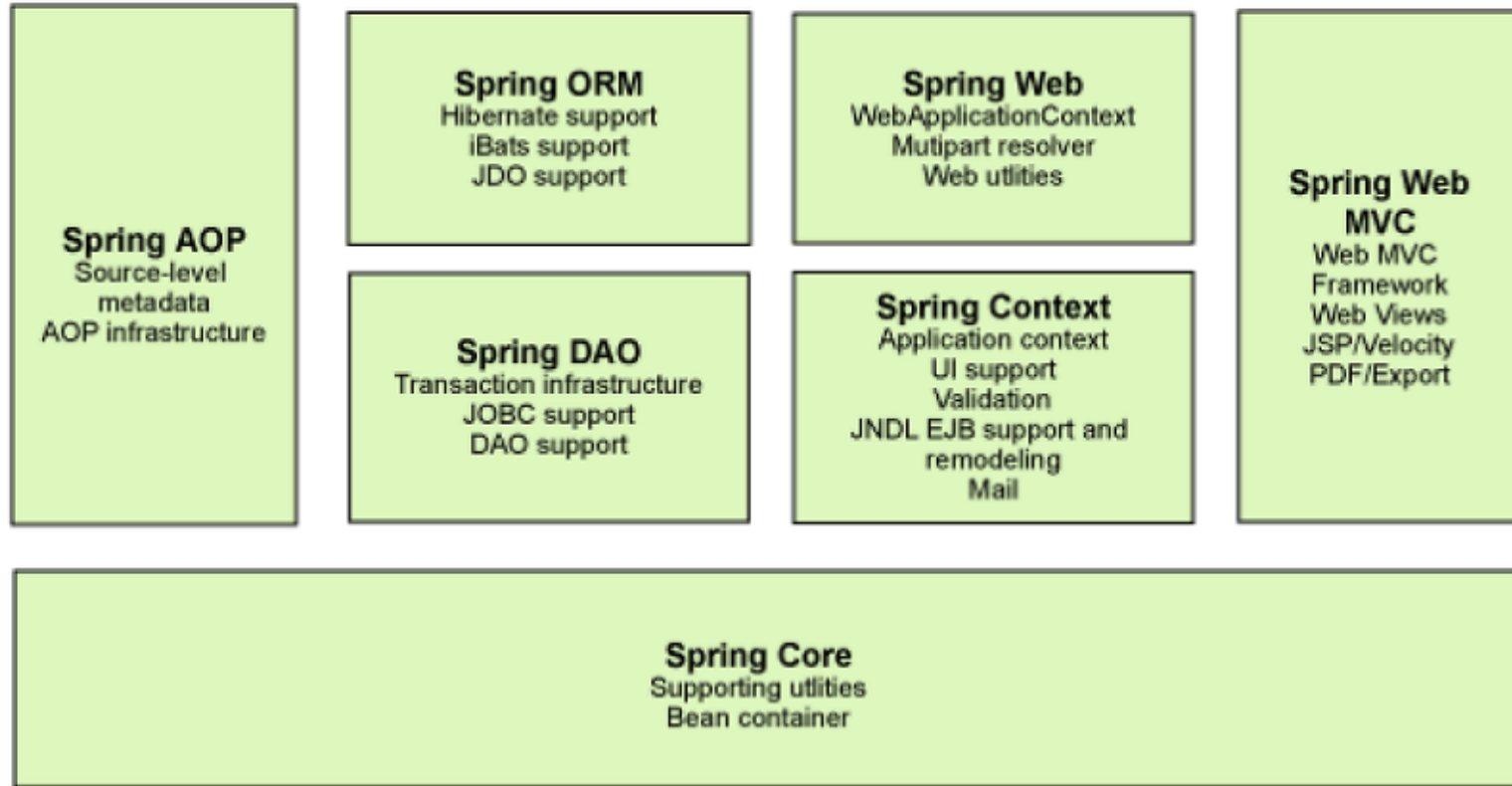
1. Spring Framework
2. 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크
3. 동적인 웹 사이트를 개발하기 위해 여러가지 서비스를 제공하고 있다.
4. 전자정부 표준 프레임워크의 기반 기술
5. 엔터 프라이즈급 애플리케이션을 만들기 위한 **경량솔루션**(가벼운 프레임워크라는 뜻이 아니다)
6. 필요한 부분만 가져다 사용할 수 있도록 모듈화 되어 있다.
7. 각 모듈은 독립적으로 분리되어 있고, 재사용이 가능하다.
8. **DI/IOC와 AOP를 지원한다.**



1. POJO(Plain Old Java Object) : 평범한 옛날 자바 객체, 자바 객체
2. IoC(Inversion Of Control) : 제어의 역전, 컨테이너는 개발자 대신 객체의 생성부터 소멸까지 책임진다.
3. DI(Dependency Injection) : 의존성 주입
4. AOP(Aspect Oriented Programming) :관점 지향 프로그래밍
5. PSA(Portable Service Abstraction) : 일관성 있는 서비스 추상화

02 Spring Boot 시작하기

Spring 소개



1. Core : 스프링의 핵심, IoC 기능이 구현된 BeanFactory 제공
2. Context : 스프링의 기본, 스프링 기반에서 구현된 기능 객체들에 대한 접근 방법을 제공
3. DAO : JDBC에 대한 추상화 계층, 반복적인 코딩을 줄여주고, 트랜잭션 관리 기능도 제공
4. ORM : 객체와 관계 매핑, ORM 제품들을 스프링 기능과 조합해서 사용 가능
5. AOP : 관점 지향 프로그래밍 기능 제공
6. Web, Web MVC : MVC Web Application 개발에 필요한 기본 기능 제공

Spring VS Node

Spring	Node.js
Java	JavaScript
멀티 스레드	싱글 스레드 이벤트 루프





1. Spring Project의 하나
2. 수작업으로 초기 셋팅 하는 과정 없이 간단히 Spring Project를 생성할 수 있다.
3. 프로젝트마다 기본적으로 설정하게 되는 부분들을 이미 내부적으로 가지고 있다.
4. Servlet Container를 기본 내장하고 있다.(Tomcat, Jetty)
5. pom.xml에서 의존 라이브러리의 버전을 자동으로 관리해준다.
6. 자주 사용하는 프로젝트 조합을 미리 만들어 놓아 스프링을 더욱 쉽고 간단하게 사용하기 위해 만들어졌다.
7. 설정을 자동으로 해준다.

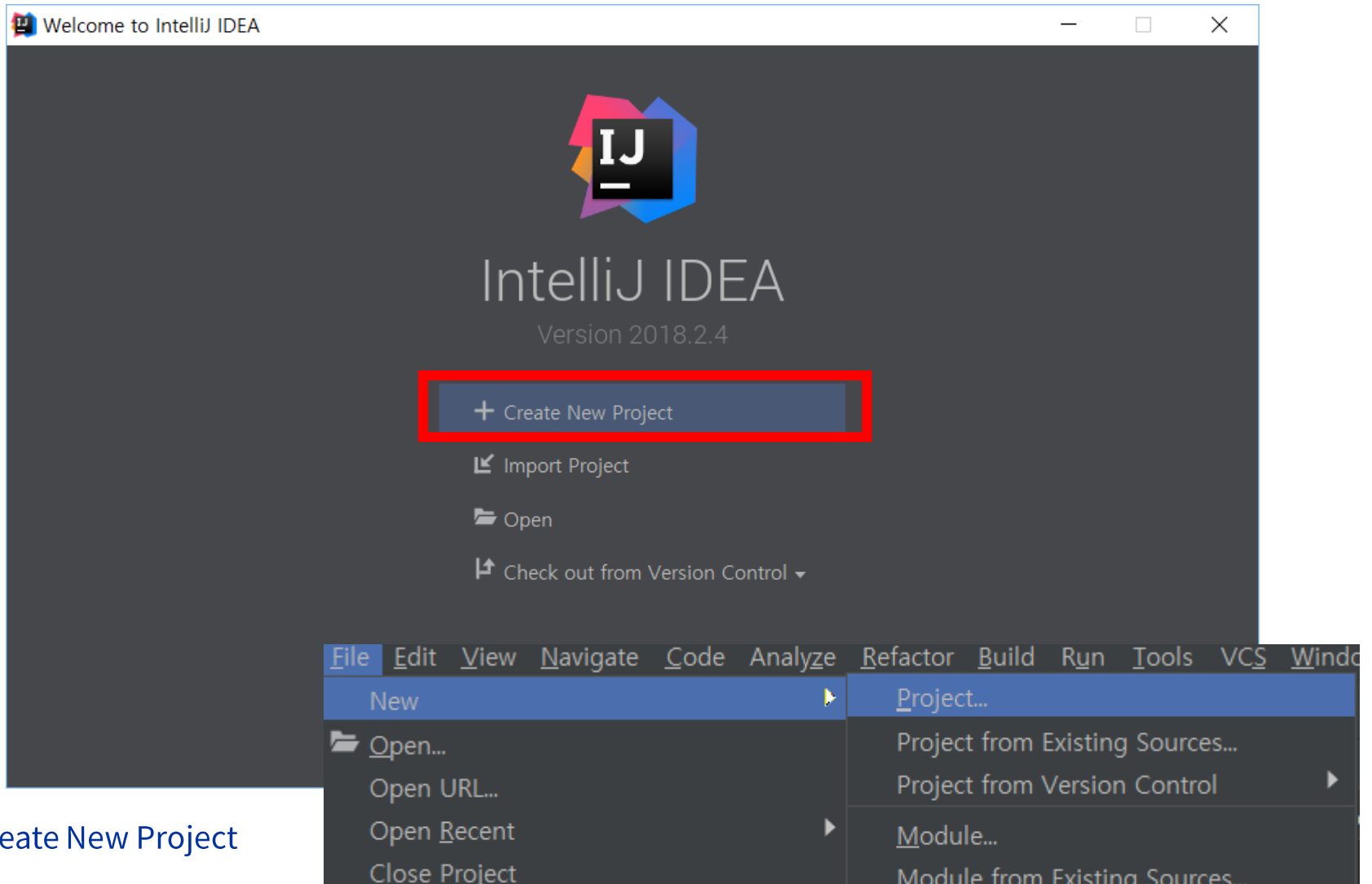
○

02

○

Spring Boot 프로젝트 생성

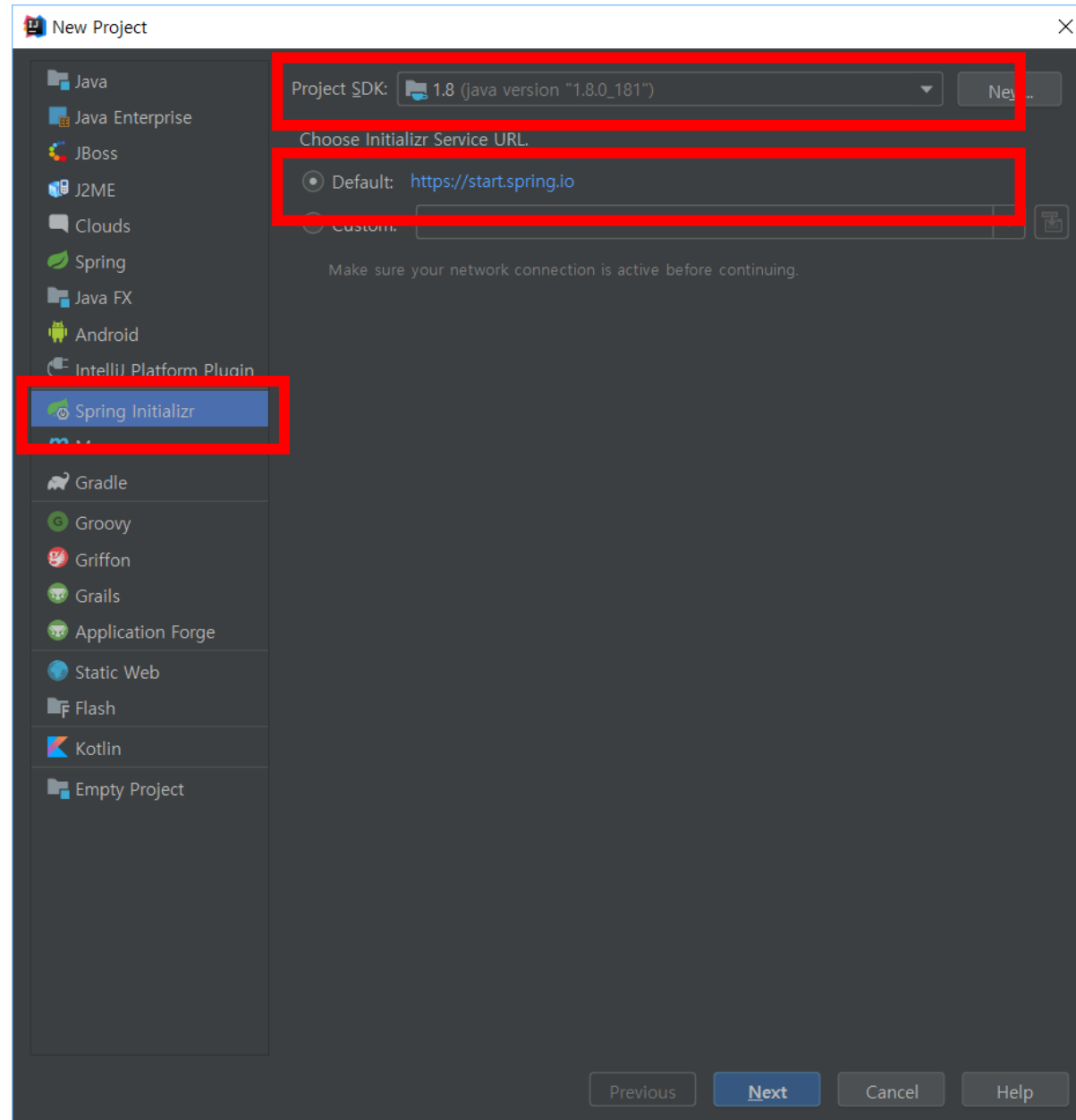
02 Spring Boot 시작하기 프로젝트 생성



1. Create New Project

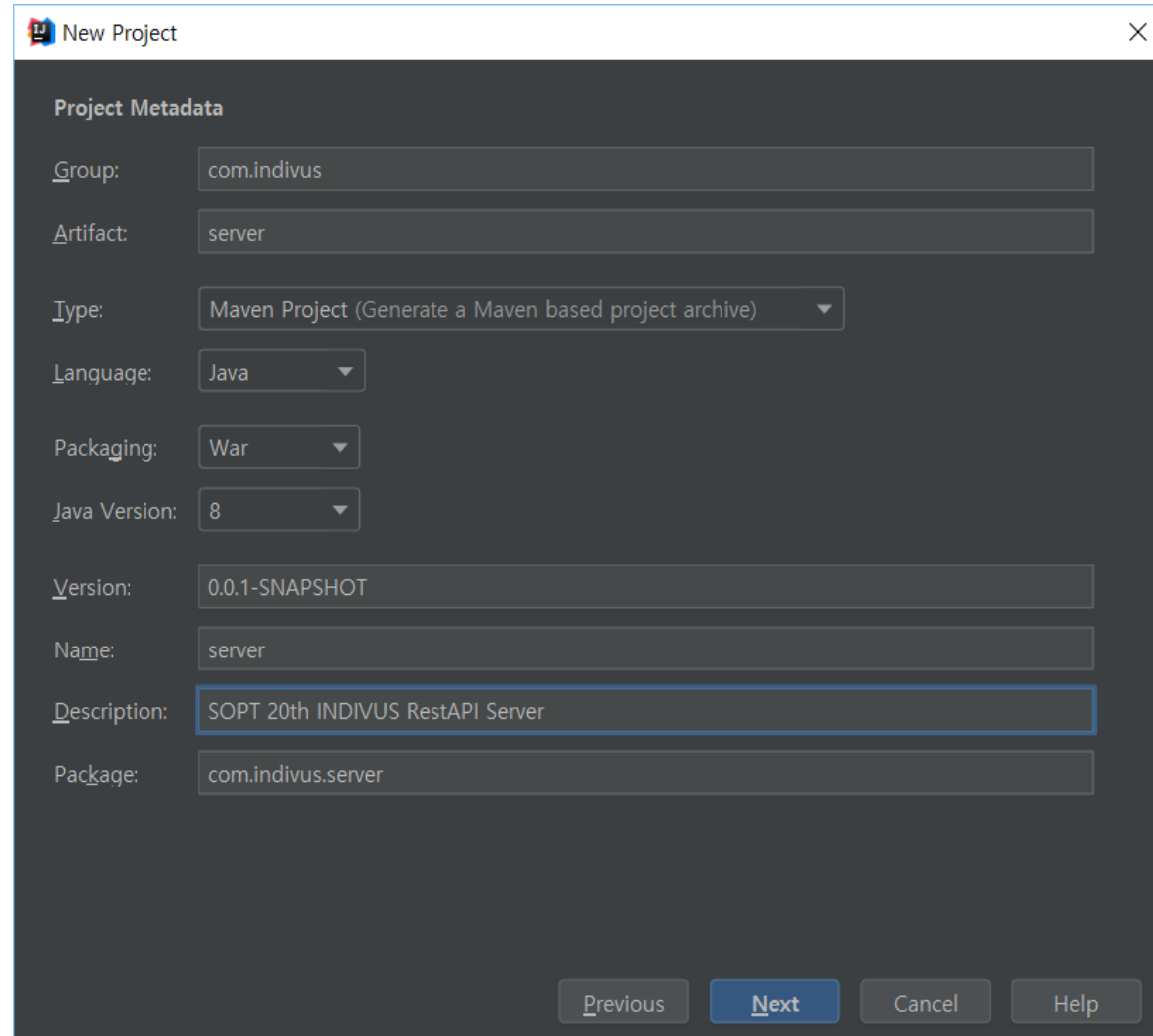
02 Spring Boot 시작하기 프로젝트 생성

1. Spring Initializr
2. SDK : Java 8
3. Initializr Service URL : Default



02 Spring Boot 시작하기 프로젝트 생성

1. Group : Group id
2. Artifact : Maven Build의 결과로 얻을 수 있는 일반적인 jar / war 와 같은 실행 파일의 이름
3. Type : Build 도구
4. Language : 사용 언어
5. Packaging : Build 결과물 타입
6. Java Version : 8
7. Version : 프로젝트 버전
8. Name : Artifact와 같음(기본값)
9. Description : 프로젝트 설명
10. Package : Group + Artifact와 같음(기본값)



The screenshot shows the 'New Project' dialog box with the following fields and values:

- Group:** com.indivus
- Artifact:** server
- Type:** Maven Project (Generate a Maven based project archive)
- Language:** Java
- Packaging:** War
- Java Version:** 8
- Version:** 0.0.1-SNAPSHOT
- Name:** server
- Description:** SOPT 20th INDIVUS RestAPI Server
- Package:** com.indivus.server

At the bottom, there are four buttons: Previous, Next, Cancel, and Help. The 'Next' button is highlighted in blue.

02 Spring Boot 시작하기 프로젝트 생성

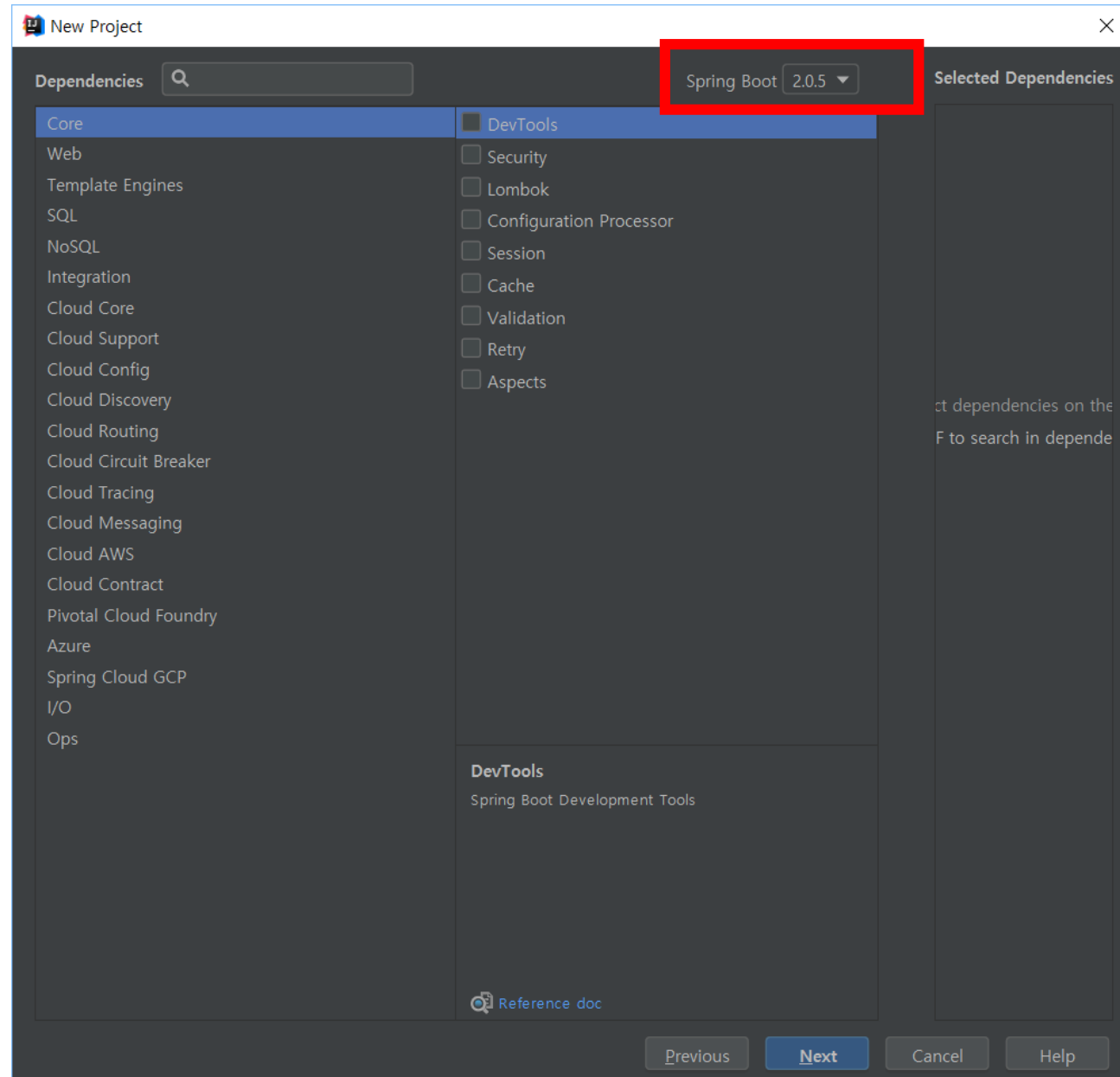
스프링의 장점

다양한 Dependencies Library

Maven에 등록되어 있는 다양한

라이브러리를 사용할 수 있다.

Node의 NPM과 비슷한 역할



02 Spring Boot 시작하기 프로젝트 생성

New Project

Dependencies

Core

Web

Template Engines

SQL

NoSQL

Integration

Cloud Core

Cloud Support

Cloud Config

Cloud Discovery

Cloud Routing

Cloud Circuit Breaker

Cloud Tracing

Cloud Messaging

Cloud AWS

Cloud Contract

Pivotal Cloud Foundry

Azure

Spring Cloud GCP

I/O

Ops

☒ Web

☐ Reactive Web

☐ Rest Repositories

☐ Rest Repositories HAL Browser

☐ HATEOAS

☐ Web Services

☐ Jersey (JAX-RS)

☐ Websocket

☐ REST Docs

☐ Vaadin

☐ Apache CXF (JAX-RS)

☐ Ratpack

☐ Mobile

☐ Keycloak

Web

Full-stack web development with Tomcat and Spring MVC

Building a RESTful Web Service

Serving Web Content with Spring MVC

Building REST services with Spring

Reference doc

Spring Boot 2.0.5

Selected Dependencies

Web

Web

선택해야 할 것

1. Web – Web

Previous

Next

Cancel

Help

02 Spring Boot 시작하기 프로젝트 생성

1. Core – Security

2. Core – Lombok

3. Core – Session

4. Web – Web

5. Web – Reactive Web

6. SQL – JPA

7. SQL – MySQL

8. SQL – JDBC

9. SQL – MyBatis

10. NoSQL – Redis

11. NoSQL – MongoDB

12. NoSQL – Elasticsearch

13. Integration – Spring Integration

14. Integration – RabbitMQ

15. Integration – Kafka

16. Integration – JMS

17. Cloud Core – Cloud Security

18. Cloud Core – Cloud OAuth2

19. Cloud Config – Config Client

20. Cloud Config – Config Server

21. Cloud Config – Zookeeper Configuration

22. Cloud Config – Consul Configuration

23. Cloud Discovery – Eureka Discovery

24. Cloud Discovery – Eureka Server

25. Cloud Discovery – Zookeeper Discovery

26. Cloud Discovery – Consul Discovery

27. Cloud Routing – Zuul

28. Cloud Routing – Ribbon

29. Cloud Circuit Breaker – Hystrix

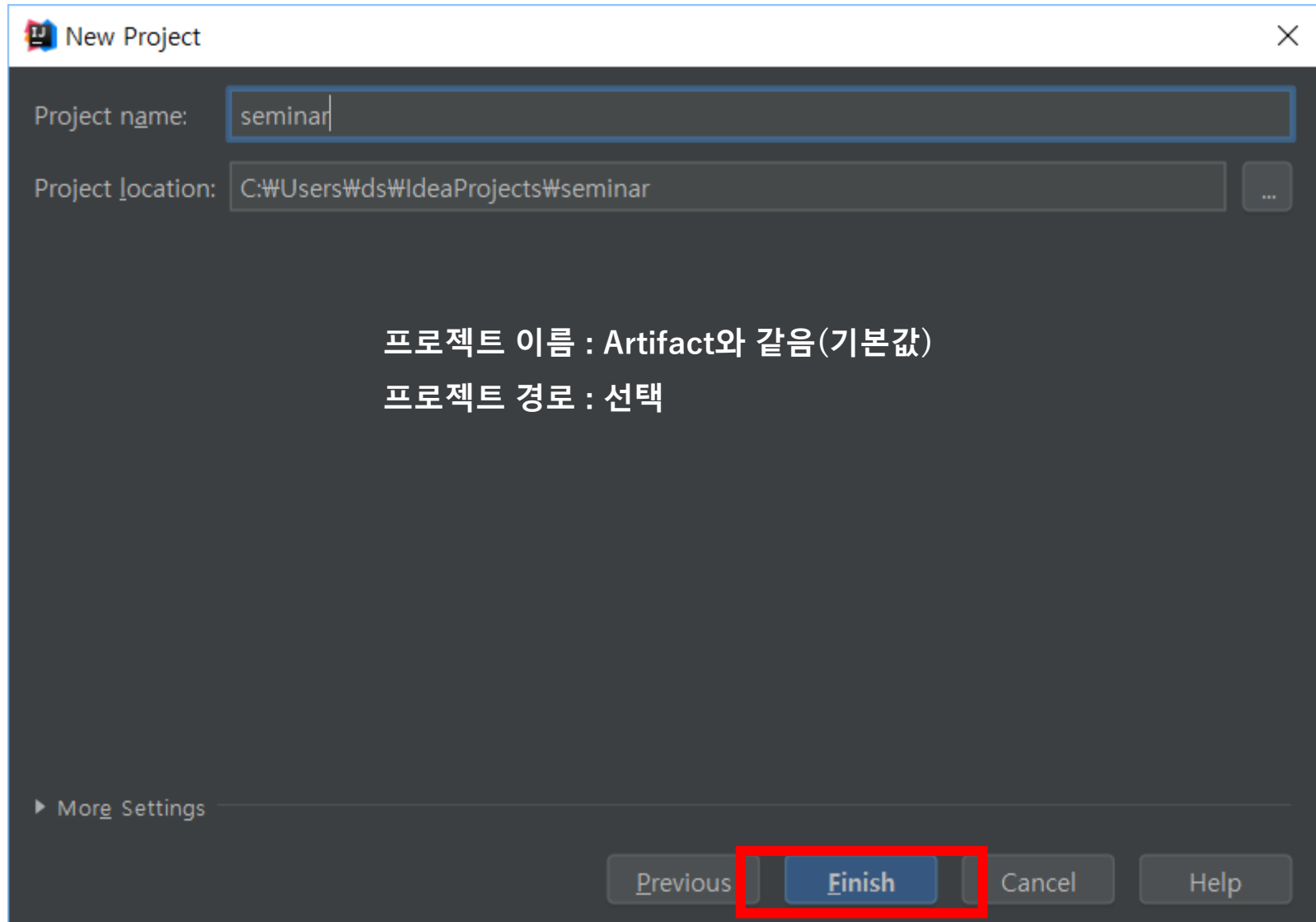
30. Cloud Circuit Breaker – Hystrix Dashboard

31. Cloud AWS – AWS Core

32. I/O – Batch

33. Ops – Actuator

02 Spring Boot 시작하기 프로젝트 생성



The image shows the 'New Project' dialog box in IntelliJ IDEA. The 'Project name' field contains 'seminar' and the 'Project location' field contains 'C:\Users\Wds\IdeaProjects\seminar'. The 'Finish' button is highlighted with a red rectangle.

New Project

Project name: seminar

Project location: C:\Users\Wds\IdeaProjects\seminar

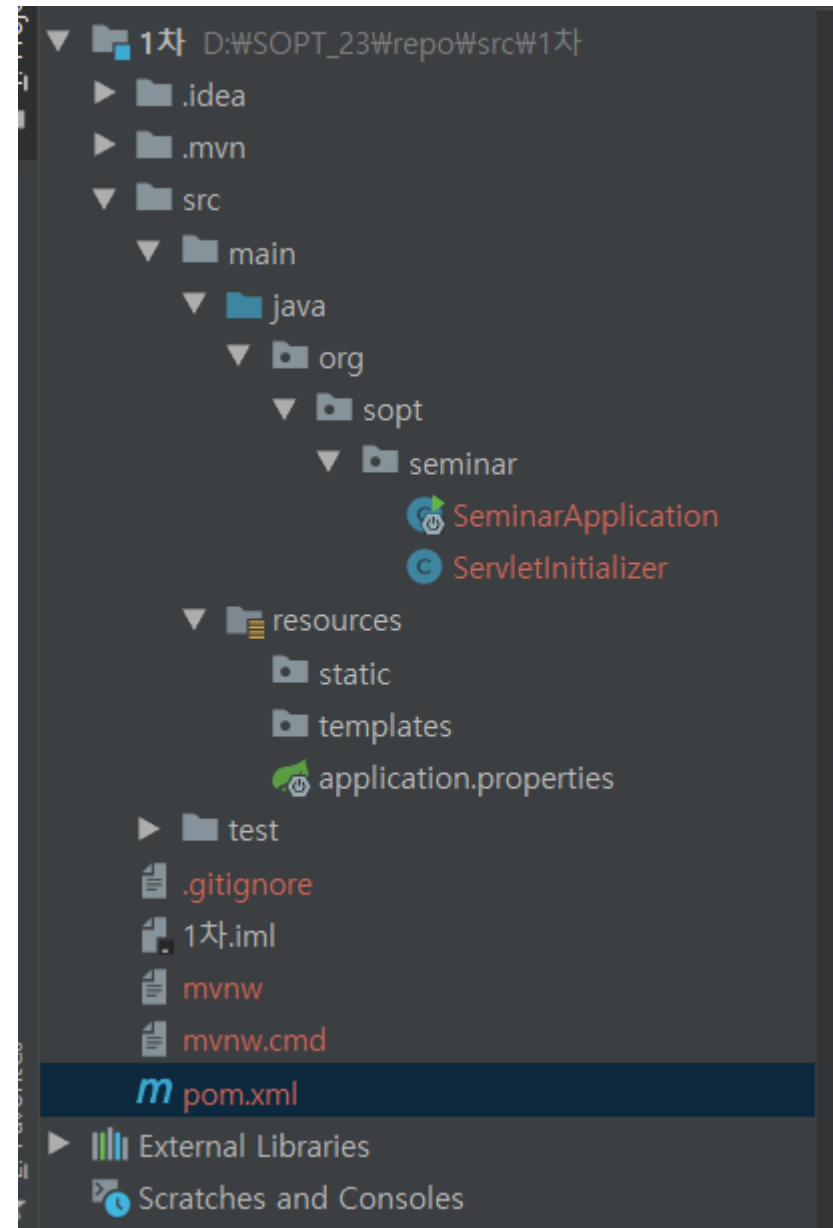
프로젝트 이름 : Artifact와 같음(기본값)
프로젝트 경로 : 선택

► More Settings

Previous Finish Cancel Help

02 Spring Boot 시작하기 프로젝트 생성

1. 처음 프로젝트 생성시 기본 디렉터리 구조





02

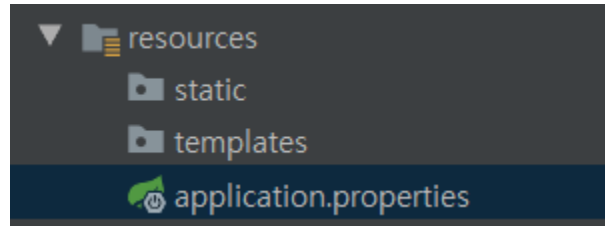


Spring Boot 구성요소

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

1. pom.xml : Project Object Model
2. Maven이 해당 파일을 토대로 의존성 관리 및 배포까지 모든 것을 관리한다.
3. Spring boot는 pom.xml을 토대로 필요한 라이브러리나 프레임워크의 의존 관계를 설정을 자동으로 해준다.



1. resources : 자원 파일 저장

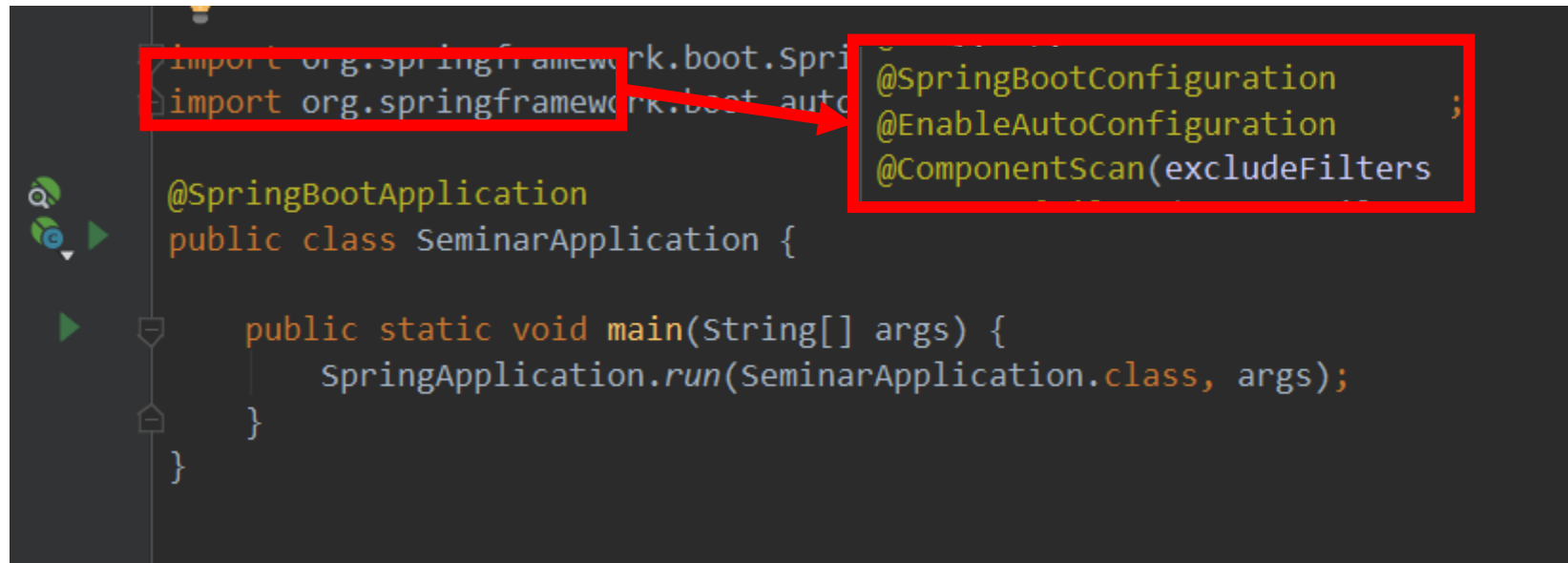
1. static : 웹 정적 리소스 저장 위치, (html, css, js, img)
2. templates : 웹 동적 리소스 저장 위치, (뷰 템플릿 저장 위치, FreeMarker, Groovy, Thymeleaf, JSP)
3. application.properties : Spring boot의 기본 설정 파일

Spring boot가 Application 시작 시 이곳의 설정들을 자동으로 반영해 준다.

이곳에 DB 정보, 서버 포트 번호 등 설정, 이럴 경우 반드시 git에 올리면 안된다.

(.gitignore에 등록 권장)

2. 이곳에 Mapper(SQL) 관련 파일이 저장될 예정



1. 프로젝트Application 파일

1. `@SpringBootApplication` = `@Configuration` + `@EnableAutoConfiguration` + `@ComponentScan`
2. `@Configuration`: 현재 클래스가 스프링의 설정 파일임을 Spring Context에게 알려주는 Annotation,
3. `@EnableAutoConfiguration`: Spring Boot가 설정을 자동으로 하도록 하는 Annotation
4. `@ComponentScan`: 다른 컴포넌트, 서비스, 설정 등을 찾을 수 있게 도와주는 Annotation

```
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(SeminarApplication.class);
    }
}
```

```
<groupId>org.sopt</groupId>
<artifactId>seminar</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
```

1. ServletInitializer

1. Packing을 War로 설정할 시 자동 생성되는 클래스 (Jar로 설정할 시엔 생성되지 않는다.)
2. WebApplicationInitializer Interface의 구현체
3. Tomcat 같은 Servlet Container 환경에서 Spring Boot Application이 동작 될 수 있도록 하는 Web Application Context를 구성한다는 의미

단위 : Class < Jar < War < Ear

Jar(Java Archive)

1. 하나의 Application 기능이 가능하도록 java 파일 등을 압축
2. 실행될 클래스를 명시해야 한다.(main)
3. 단독으로 실행이 가능하다.(JVM위에서)
4. 라이브러리, 리소스, property 파일등을 포함한다.
5. Spring boot는 WAS(Tomcat)을 내장하고 있기 때문에 jar 배포가 가능하다.

War(Web Archive)

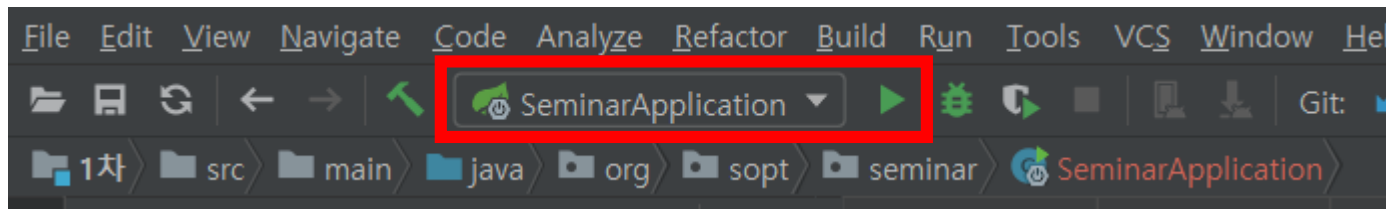
1. Web Application을 지원하기 위한 압축
2. 단독으로 실행이 불가능하다.(WAS필요)
3. Jsp, Servlet, git, html, jar 등을 압축하고 지원
4. WAS(Tomcat)이 알아서 압축을 해제해서 배포해준다.

○

02

○

Web Application 실행



1. Web Application 실행

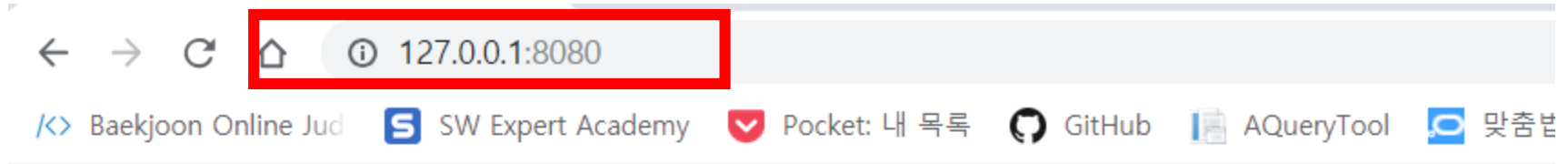
02 Spring Boot 시작하기 Web Application 실행



```
=====|_|=====|_|_|_|_|
```

```
:: Spring Boot ::      (v2.0.5.RELEASE)
```

```
2018-09-24 16:29:54.714 INFO 12872 --- [main] org.sopt.seminar.SeminarApplication : Starting SeminarApplication on DESKTOP-0109B9N with PID 12872 (D:\#SOPT_23\repo\src\seminar\target\classes started by ds i
2018-09-24 16:29:54.714 INFO 12872 --- [main] org.sopt.seminar.SeminarApplication : No active profile set, falling back to default profiles: default
2018-09-24 16:29:54.777 INFO 12872 --- [main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@19d37183: star
2018-09-24 16:29:56.495 INFO 12872 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2018-09-24 16:29:56.527 INFO 12872 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2018-09-24 16:29:56.527 INFO 12872 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.34
2018-09-24 16:29:56.527 INFO 12872 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener : The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on t
2018-09-24 16:29:56.698 INFO 12872 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2018-09-24 16:29:56.698 INFO 12872 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1921 ms
2018-09-24 16:29:56.839 INFO 12872 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Servlet dispatcherServlet mapped to [/]
2018-09-24 16:29:56.839 INFO 12872 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [/+]
2018-09-24 16:29:56.839 INFO 12872 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/+]
2018-09-24 16:29:56.839 INFO 12872 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'httpPutFormContentFilter' to: [/+]
2018-09-24 16:29:56.839 INFO 12872 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'requestContextFilter' to: [/+]
2018-09-24 16:29:56.995 INFO 12872 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequest
2018-09-24 16:29:57.167 INFO 12872 --- [main] s.w.s.m.a.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationCo
2018-09-24 16:29:57.230 INFO 12872 --- [main] s.w.s.m.a.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>
2018-09-24 16:29:57.230 INFO 12872 --- [main] s.w.s.m.a.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boc
2018-09-24 16:29:57.245 INFO 12872 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHar
2018-09-24 16:29:57.245 INFO 12872 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-09-24 16:29:57.370 INFO 12872 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
2018-09-24 16:29:57.417 INFO 12872 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2018-09-24 16:29:57.417 INFO 12872 --- [main] org.sopt.seminar.SeminarApplication : Started SeminarApplication in 3.156 seconds (JVM running for 4.698)
```



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Sep 24 16:54:18 KST 2018

There was an unexpected error (type=Not Found, status=404)

No message available

1. 127.0.0.1:8080 접속
2. White label Error Page 뜨면 성공
3. 404 Not Found

127.0.0.1 / localhost

1. localhost는 네트워크에서 사용하는 루프 백 호스트명
2. 자신의 컴퓨터를 의미
3. IPv4에서 IP 주소는 127.0.0.1 / IPv6에서는 ::1
4. 로컬 컴퓨터를 마치 원격 컴퓨터인 것 같이 통신할 수 있어 테스트 목적으로 사용한다.

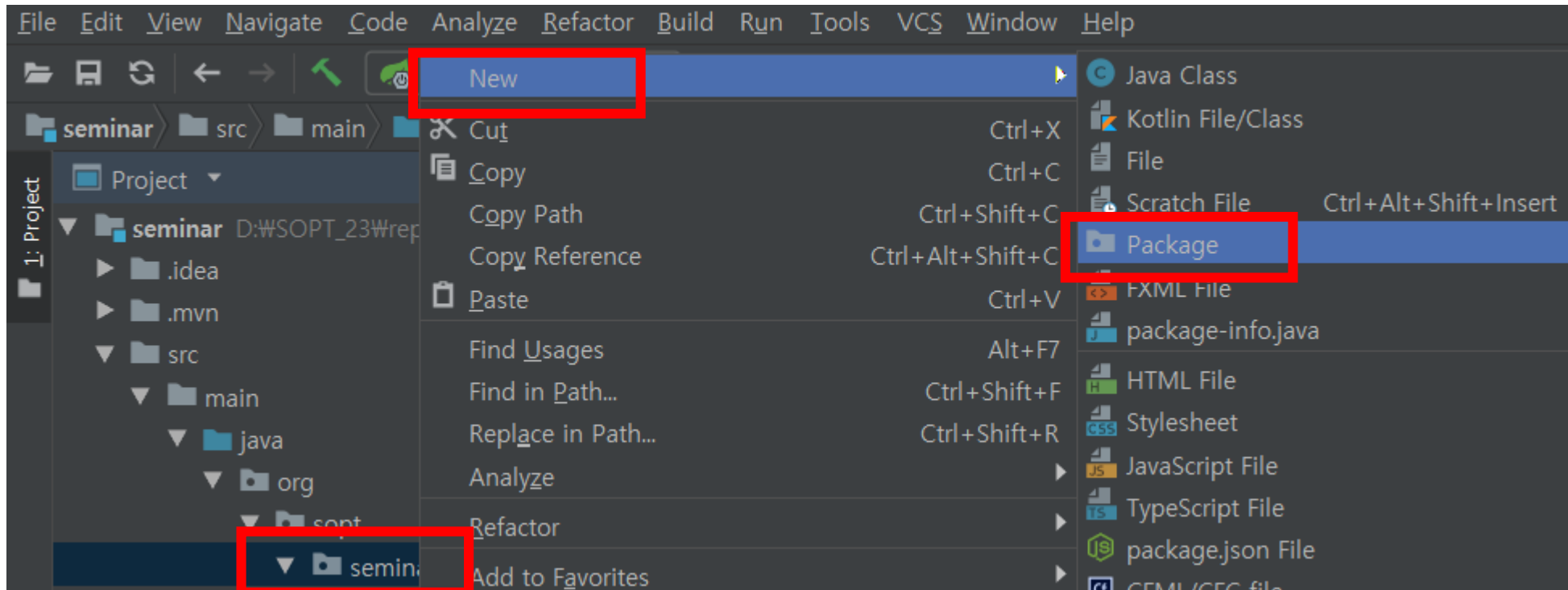
○

02

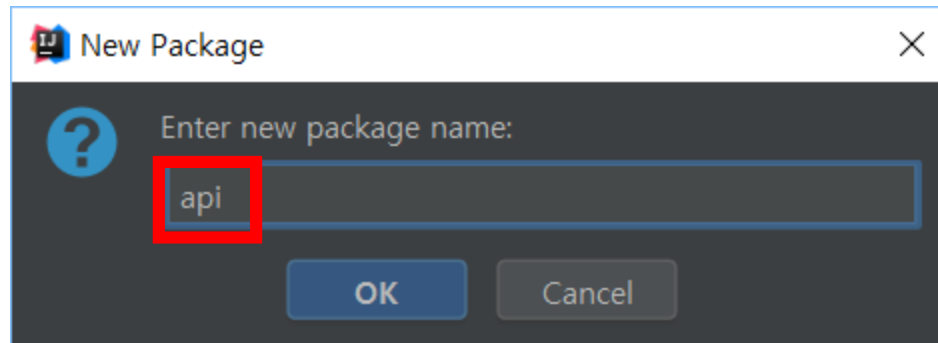
○

Controller 생성 맛보기

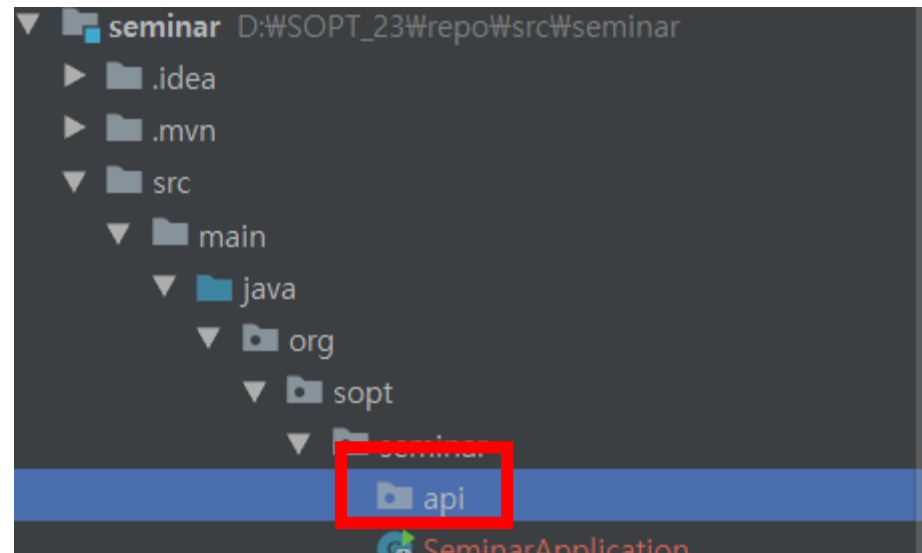
02 Spring Boot 시작하기 Controller 생성



1. seminar 선택 후 마우스 우 클릭
2. New 선택
3. Package 선택



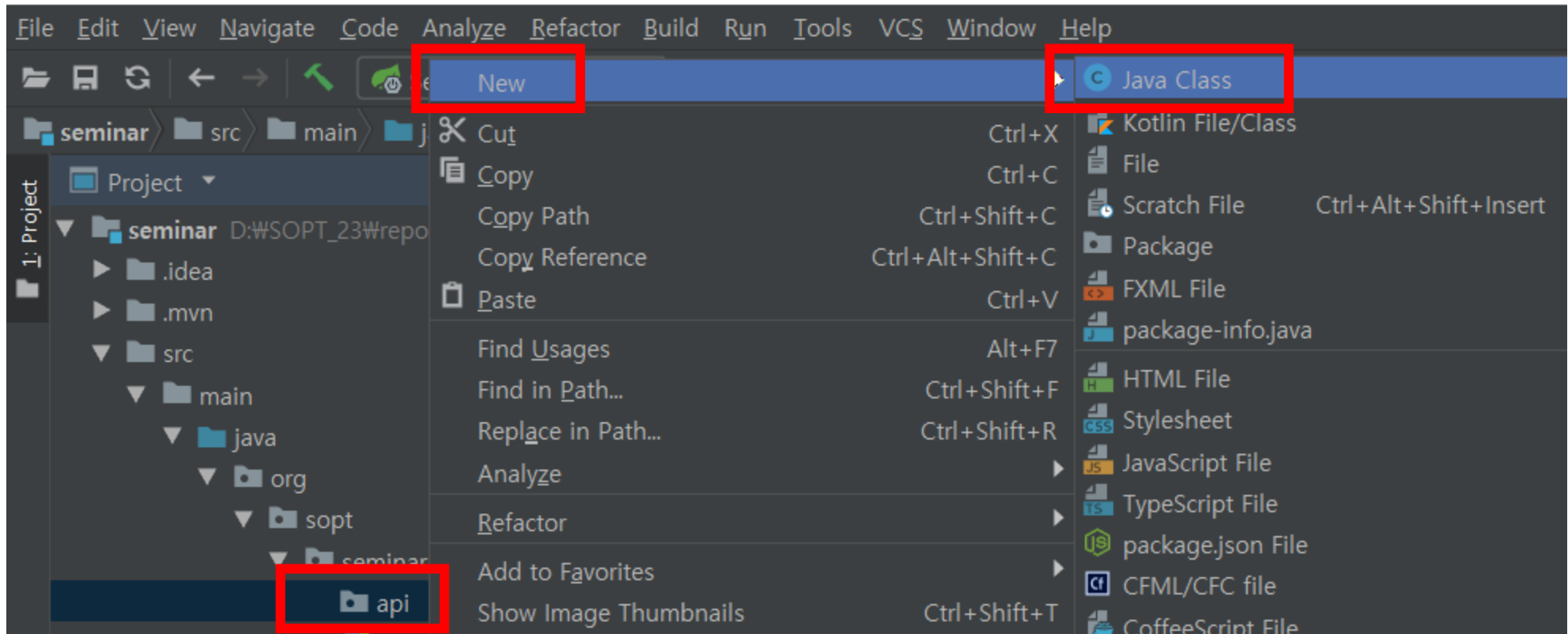
1. api package 생성



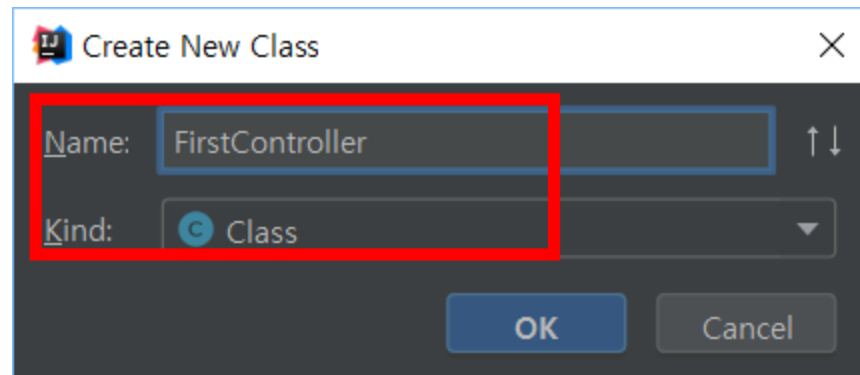
1. api package 생성

02 Spring Boot 시작하기 Controller 생성

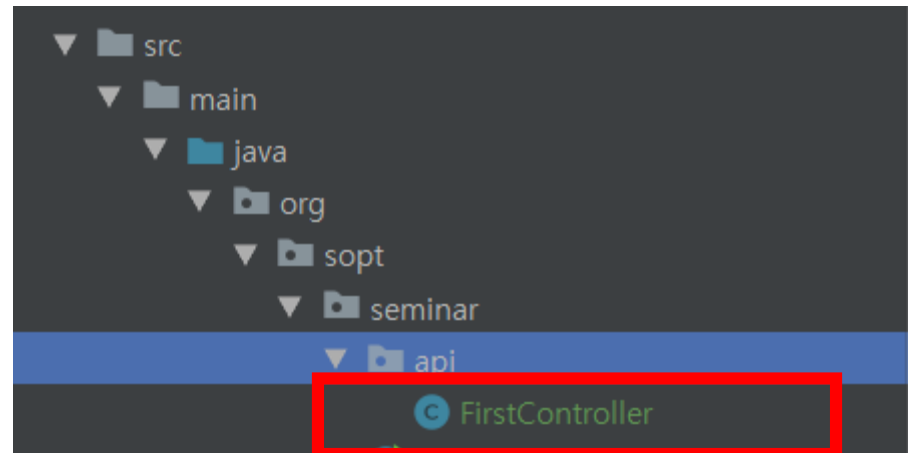
seminar [D:\WSOPT_23\repo\src\seminar] - seminar - IntelliJ IDEA



1. api 선택 후 마우스 우 클릭
2. New 선택
3. Java Class 선택



1. FirstController 클래스 파일 생성



1. api package 안에 FirstController 생성

```
package org.sopt.seminar.api;

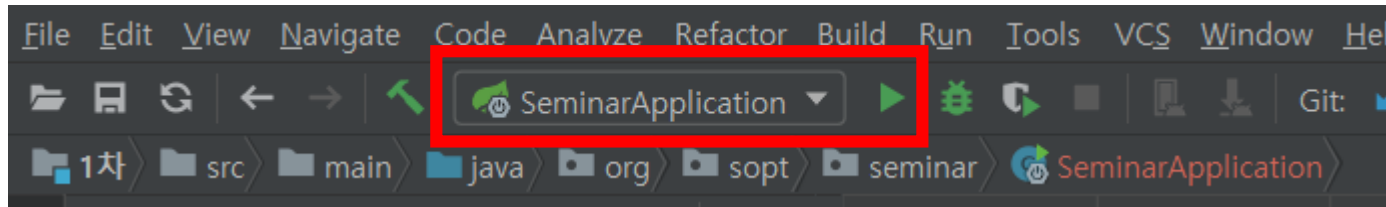
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Created by ds on 2018-09-24.
 */

@RestController
public class FirstController {

    @GetMapping("/hello")
    public String Hello() {
        return "Hello World!";
    }
}
```

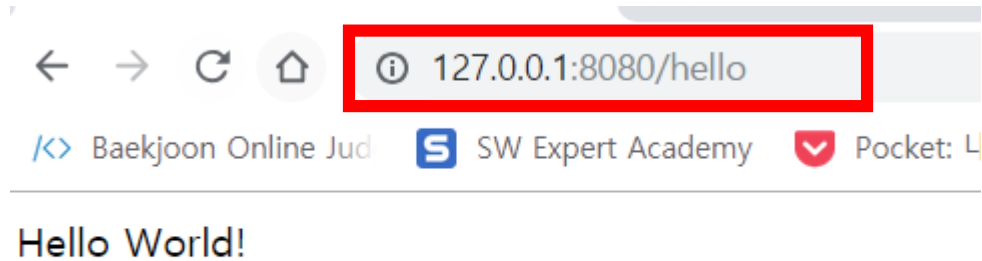
1. First Rest Controller 코드 작성



1. Web Application 실행

```
Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApp
Mapped "{[/hello],methods=[GET]}" onto public java.lang.String org.sopt.seminar.api.FirstController.Hello()
```

Controller 반영



1. 127.0.0.1:8080/hello 접속
2. Hello World! 뜨면 성공

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

THANK U

PPT 디자인 한승미 세미나 자료 배다슬

SHOUT OUR PASSION TOGETHER
SOPT