외부 설정

애플리케이션(프로젝트) 외부에서 주입할 수 있는 설정.

똑같은 애플리케이션도 구동하는 서버 환경이 다를 수 있다.

이를 대비하기 위해 스프링부트는 애플리케이션 수정 없이 설정만 변경하여 서비스 할 수 있도록 외부 `프로퍼티` 설정을 제공하고 있다.

외부 설정 주요 방법

- `.properties` 혹은 `.yml` 에 주입할 프로퍼티 기재 (파일명은 `application` 으로 설정하는 것이 스프링부트의 컨벤션이다.)
- 환경 변수 형태로 주입
- 커맨드라인 (cmd/terminal) 의 args 로 주입

> You can use properties files, YAML files, environment variables, and command-line arguments to externalize configuration. Property values can be injected directly into your beans by using the `@Value` annotation, accessed through Spring's `Environment` abstraction, or be bound to structured objects through `@ConfigurationProperties`.
>
> 출처 : https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-external-config

외부 설정 우선순위

우선 순위 (오름차순) - 로딩하는 순서가 17 ~ 1번이라 중복되는 속성은 오버라이드 되는 것이다.
1. (Devtools 를 사용한다면) `$HOME/.config/spring-boot` 디렉토리에 위치하는 Devtools global settings properties
2. Test 프로젝트의 `@TestPropertySource`
3. Test 프로젝트에 위치한 `@SpringBootTest`
4. cmd(Terminal)의 args
5. `SPRING_APPLICATION_JSON` 의 프로퍼티
6. `ServletConfig` init 파라미터
7. `ServletContext` init 파라미터
8. `java:comp/env` 에 위치한 JNDI 속성
9. Java System properties (`System.getProperties()`).
10. 운영체제의 환경 변수
11. `random.*` 에 위치한 `RandomValuePropertySource`
12. jar 밖에 있는 Profile-specific application properties (`application-{profile}.properties` 혹은 `application-{profile}.yml`)
13. jar 안에 있는 Profile-specific application properties (`application-{profile}.properties` 혹은 `application-{profile}.yml`)**
14. jar 밖에 있는 application properties 파일. (`applicaton.properties` 혹은 `application.yml`)
15. jar 안에 있는 application properties 파일. (`applicaton.properties` 혹은 `application.yml`)
16. @Configuration 클래스에 선언된 `@PropertySource`
17. `SpringApplication.setDefaultProperties` 에 선언된 기본 프로퍼티

출처 : https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-external-config

15위 : application.properties 혹은 application.yml 에 프로퍼티 주입

application.properties

```
my.name.first = sera
my.name.last = lee
my.age = @{random.int}
my.hobby = reading, eating, coding
message = Hi, Hello, \
Bye, See you
```

`.properties`의 특징

- key-value 쌍의 매핑 형태로 표현한다.
- 상속관계는 `.`으로 표기한다.
- 리스트는 `,`로 표기한다.
- 주석은 `#`으로 표기한다.

application.yml

```yaml
my:
  name:
    first: sera
    last: lee
  age: 10
  hobby:
    - reading
    - eating
    - coding
message:
  - Hi
  - Hello
  - Bye
  - See you
```

`yml/yaml`의 특징

- 상속관계는 `:`과 `줄바꿈`으로 한다.
- 자식 프로퍼티는 부모 들여쓰기 + 2칸으로 한다.
- `값`표기는 `:`에 한 칸을 띄운다.
- 리스트는 `-`로 표기한다.
- 주석은 `#`으로 표기한다.

> yml 참고 : https://lejewk.github.io/yaml-syntax/

둘 중 아무거나 사용하면 된다.

이렇게 만들어진 applicaton 프로퍼티 파일을 애플리케이션에서 불러와보자.

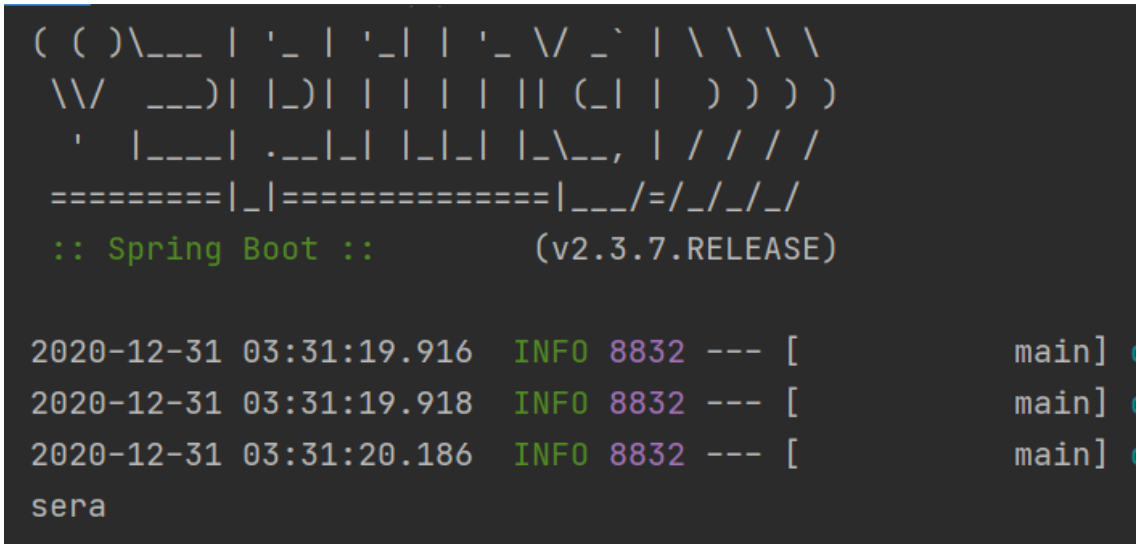
MyRunner.class (@ComponentScan 이 되는 아무 곳에나 클래스를 만든다.)

```java
package com.megait.externalized_configuration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class MyRunner implements ApplicationRunner {

    @Value("${my.name.first}")
    private String firstName;

    @Override
```

```
    public void run(ApplicationArguments args) throws Exception {
        System.out.println(firstName);
    }
}
```

main()은 건드리지 않아도 된다. main()을 바로 실행해보자.

결과



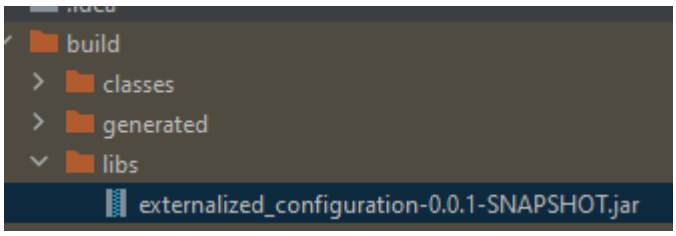
4위 : cmd 명령에 args 형태로 주입

일단 jar/war 로 패키징부터 해야 한다.

Terminal 에서

```
gradlew build
```

그러면 이렇게 build/libs 에 jar/war 가 생성된다.



이것을 실행해보자.

```
java -jar build/libs/externalized_configuration-0.0.1-SNAPSHOT.jar
```

그러면 아까 main()을 실행한 것과 같은 결과가 나온다.

그럼 이번엔 args 를 넣어보자.

sera를 pikachu로 바꿔보자.

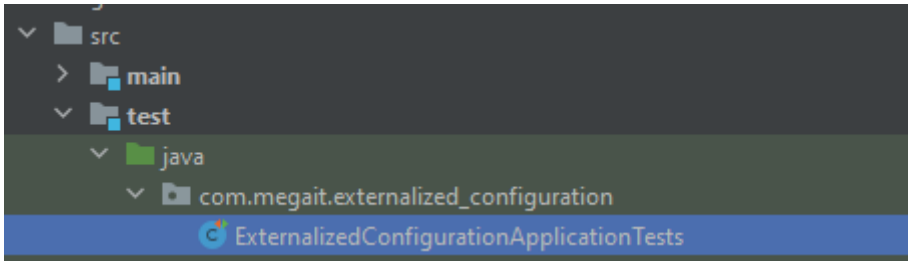
```
java -jar build/libs/externalized_configuration-0.0.1-SNAPSHOT.jar --
my.name.first=pikachu
```

결과

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.7.RELEASE)

2020-12-31 03:41:20.858  INFO 13552 --- [           main]
 세라몬 with PID 13552 (D:\spring-boot\externalized_config
OT.jar started by Sera in D:\spring-boot\externalized_conf
2020-12-31 03:41:20.862  INFO 13552 --- [           main]
2020-12-31 03:41:21.321  INFO 13552 --- [           main]
pikachu
```

3위 Test 프로젝트에 위치한 `@SpringBootTest`

이제 Test 클래스로 가보자.



ExternalizedConfigurationApplicationTests.java

```
package com.megait.externalized_configuration;


import org.assertj.core.api.Assertions;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.core.env.Environment;
```

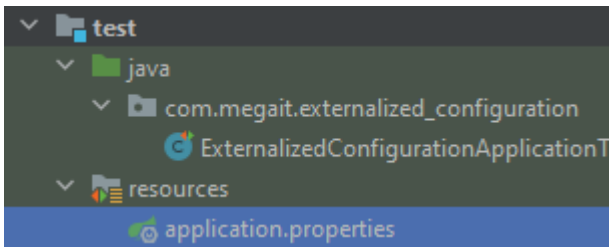
```java
@SpringBootTest
class ExternalizedConfigurationApplicationTests {

    @Autowired
    Environment environment;

    @Test
    void contextLoads() {
        String name = environment.getProperty("my.name.first");
        System.out.println("Test : " + name);
        Assertions.assertThat(name).isEqualTo("sera");
    }

}
```

이 테스트에서는 name이 sera가 아니면 Exception을 일으킨다.

Exception 이 안나는 것을 보면 `application.properties`(15위) 에서 지정해둔 name 속성이 그대로 있다는 것을 확인해 볼 수 있다.

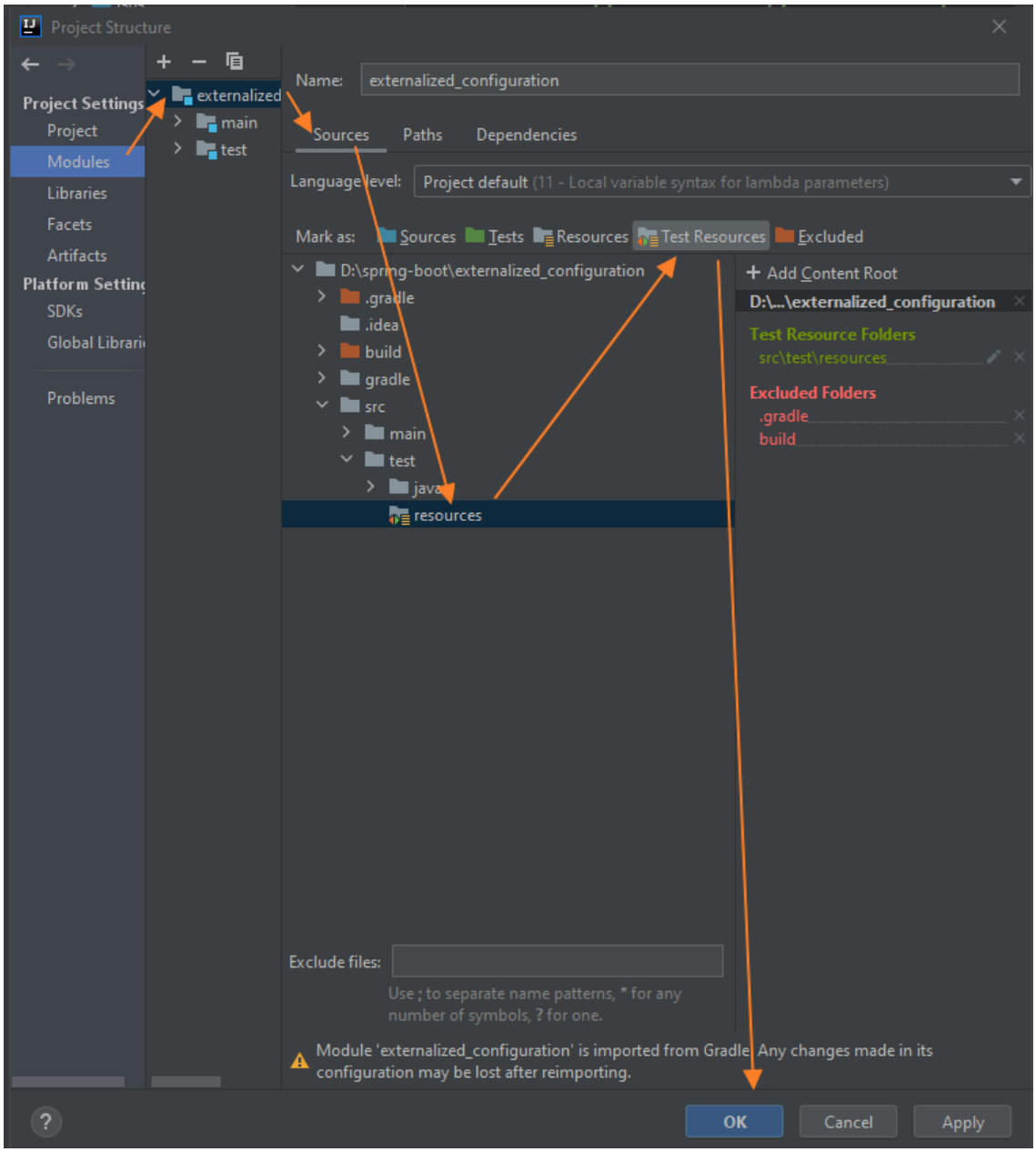
그렇다면 그 보다 높은 우선순위인 Test의 `@SpringBootTest`(3위)에서 이를 변경했을 때 정말 변경이 되는지 확인해보자.

test에 resources 폴더를 만들고 application.properties (yml) 을 복붙한다.

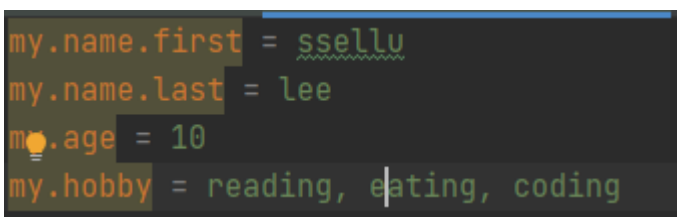


참고) resources 폴더의 아이콘이 저런 모양이 아니라면 (일반 폴더의 모양이라면) Test Resources로 적용되지 않은 것이니 다음과 같이 설정한다.

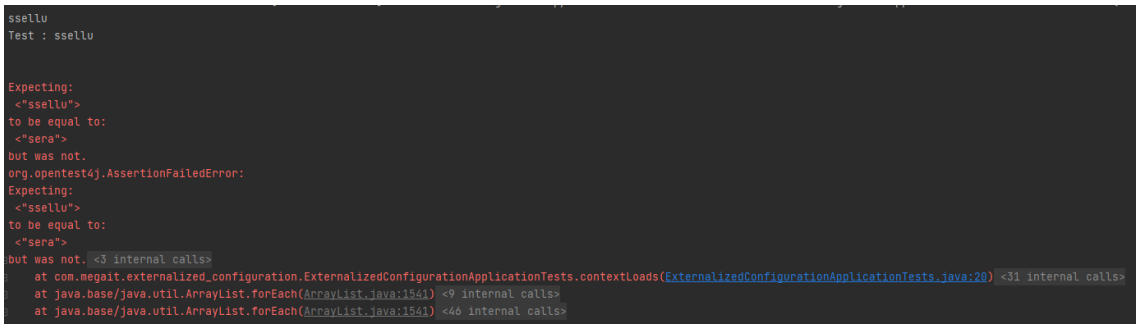
Project Structure



그러면 복붙한 Test의 application.properties 에 다음과 같이 이름을 바꿔보자.



Test 실행 결과



2위 @TestPropertySource

Test 파일에 `@TestPropertySource`을 사용하여 프로퍼티를 바꿔줄 수 있다.

```
@SpringBootTest
@TestPropertySource(properties = "my.name.first=pikachu")
class ExternalizedConfigurationApplicationTests {
    ....
}
```

참고) 만약 바꿀 프로퍼티가 많다면... File을 따로 만들어서 주입할 수 있다.

XXX.properties

```
my.name.first=pikachu
```

```
@SpringBootTest
@TestPropertySource(locations = "classpath:/XXX.properties")
class ExternalizedConfigurationApplicationTests {
    ....
}
```

하지만 이들 모두는 타입세이프하지 않다. 프로퍼티를 수정하고자 할 때 일일히 문자열을 자바 코드에 타이핑해야 하기때문에 오타의 위험에서 벗어날 수 없다.

이럴 때 사용하는 것이 프로퍼티를 객체화하여 빈으로 등록하는 `@ConfigurationProperties` 를 활용해 볼 수 있다.

https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-external-config-typesafe-configuration-properties

프로퍼티 이름과 프로퍼티 객체 필드 이름의 바인딩은 Relaxed-binding 구조다. (느슨한 바인딩)

Property	Note
`acme.my-project.person.first-name`	Kebab case, which is recommended for use in `.properties` and `.yml` files.
`acme.myProject.person.firstName`	Standard camel case syntax.
`acme.my_project.person.first_name`	Underscore notation, which is an alternative format for use in `.properties` and `.yml` files.
`ACME_MYPROJECT_PERSON_FIRSTNAME`	Upper case format, which is recommended w

@ConfigurationProperties vs. @Value

https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-external-config-validation

@ConfigurationProperties 의 검증

https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-external-config-validation

단위 변환을 제공

시간 단위와 데이터 크기 단위를 알아서 변환해준다.

https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-external-config-conversion-duration

프로퍼티 파일 암호화 방법 :

https://docs.spring.io/spring-boot/docs/2.2.4.RELEASE/reference/html/spring-boot-features.html#boot-features-encrypting-properties