스프링 부트 웹 프로젝트

chapter02

스프링 프로젝트 만들기

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

01

part

애노테이션

애노테이션이란?

애노테이션이란?

컴파일러에게 정보를 제공하여 코드의 처리 방식을 지시하는 것이다.

어노테이션은 @기호로 시작하며, 특정 기능을 실행한다.

@Annotation

애노테이션의 역할

1. 컴파일러에게 문법 에러를 체크하도록 지시
2. 프로그램을 빌드할 때 코드를 자동으로 생성할 수 있도록 지시
3. 특정 기능을 실행하도록 지시

Lesson: Annotations

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

- **Information for the compiler** — Annotations can be used by the compiler to detect errors or suppress warnings.
- **Compile-time and deployment-time processing** — Software tools can process annotation information to generate code, XML files, and so forth.
- **Runtime processing** — Some annotations are available to be examined at runtime.

애노테이션

애노테이션 예시

예를 들어 @Override는 메소드가 부모 클래스의 메소드를 정확하게 재정의 했는지 확인하고,
만약 메소드 선언부가 다르면 컴파일 에러를 발생시킨다.
이러한 방식으로 애노테이션을 사용해 코드의 문법 오류를 미리 확인할 수 있다.

부모 클래스

```
class Animal {
    void sound() {
        print(“동물이 소리를 낸다”)
    }
}
```

자식 클래스

```
class Cat extends Animal {
    @Override
    void sound() {
        print(“고양이가 야옹하고 운다”)
    }
}
```

애노테이션

애노테이션 종류

어노테이션	설명
@Override	해당 메소드가 부모클래스의 메소드를 오버라이드한다는 것을 컴파일러에게 알린다.
@Deprecated	앞으로 사용하지 않을 메소드나 클래스를 표시한다.
@FunctionalInterface	함수형 인터페이스인 것을 나타낸다.
@SuppressWarning	컴파일러가 특정 경고 메시지를 무시하도록 지시한다.
@Target	어노테이션을 정의할 때 적용 대상을 지정한다.
@Documented	어노테이션 정보를 javadoc으로 작성된 문서에 포함시킨다.
@Inherited	어노테이션이 하위 클래스에 상속되도록 한다.
@Retention	어노테이션이 유지되는 기간을 설정한다. (컴파일 단계, 런타임 단계)

애노테이션

롬복에서 제공하는 애노테이션

롬복(Lombok) 라이브러리는 코드에서 반복적으로 작성해야 하는 부분을 자동으로 완성해주는 기능을 제공한다.

```java
public class Person {

    String name;
    int age;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public Person(String name, int age) {
        super();
        this.name = name;
```

직접 메소드를 만들어서 사용한 경우

```java
@Getter
@Setter
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Person {

    String name;
    int age;

}
```

롬복을 사용한 경우

롬복에서 제공하는 애노테이션

아래 어노테이션은 모두 클래스에 설정한다.

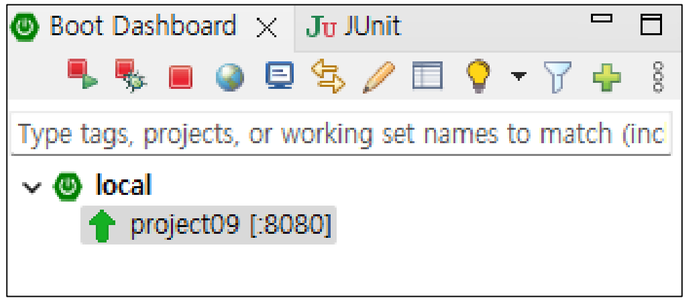
어노테이션	설명
@Getter	Getter 메소드를 생성한다
@Setter	Setter 메소드를 생성한다
@NoArgsConstructor	기본 생성자 메소드를 생성한다
@AllArgsConstructor	모든 멤버변수를 초기화하는 생성자 메소드를 생성한다
@Builder	빌더 패턴을 사용하여 체인 방식으로 객체를 생성한다
@ToString	객체의 정보를 반환하는 toString 메소드를 생성한다
@Data	Getter, Setter, ToString 메소드를 한번에 생성한다

테스트에서 제공하는 애노테이션

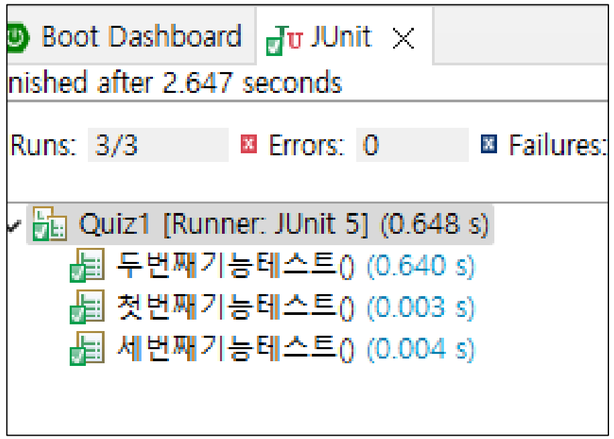
Person PersonTest

스프링 프로젝트에서는 일반적으로 메인 클래스를 하나만 사용한다.
특정 기능이 정상적으로 동작하는지 확인하려면 테스트 기능을 사용하는 것이 좋다.
스프링에는 JUnit과 Spring Test를 사용하여 단위 테스트를 만들 수 있으며,
이를 통해 어플리케이션의 각 기능을 개별적으로 검증할 수 있다.

프로젝트 실행 화면



단위 테스트 화면



애노테이션

테스트에서 제공하는 애노테이션

애노테이션	설명	위치
@SpringBootTest	테스트를 위해 임시로 스프링 테스트 환경을 실행한다.	클래스
@Test	해당메소드를 단위테스트로 설정한다. 단독으로 코드를 실행할 수 있다.	메소드
@BeforeEach	각 테스트 메서드가 실행되기 전에 실행할 코드를 설정한다.	메서드
@AfterEach	테스트 메서드가 실행된 후에 실행할 코드를 설정한다	메서드

연습문제

1번.

1) 다음과 같이 도서(Book) 클래스을 설계하세요.
  - 속성: 제목, 가격, 출판사, 페이지수
  - 기능: 모든 멤버변수의 getter/setter, 디폴트생성자, 모든 멤버변수를 초기화하는 생성자, 도서 정보를 반환하는 메소드
2) 도서 인스턴스를 3개 생성하세요.
  - 디폴트 생성자, 전체 생성자, 빌더를 각각 사용하여 생성
3) 모든 도서의 정보를 출력하세요.

2번.

1) 다음과 같이 자동차(Car) 클래스을 설계하세요.
  - 속성: 모델명, 제조사, 색
  - 기능: 모든 멤버변수의 getter/setter, 디폴트생성자, 모든 멤버변수를 초기화하는 생성자, 자동차 정보를 반환하는 메소드
2) 자동차 인스턴스를 3개 생성하세요.
3) 모든 자동차의 정보를 출력하세요.

3번.

1) 다음과 같이 학생(Student) 클래스을 설계하세요
  - 속성: 학번, 이름, 나이
  - 기능: 모든 멤버변수의 getter/setter, 디폴트생성자, 모든 멤버변수를 초기화하는 생성자, 학생 정보를 반환하는 메소드
2) 학생 인스턴스를 3개 생성하세요
3) 모든 학생의 정보를 출력하세요

4번.

1) 다음과 같이 영화(Movie) 클래스을 설계하세요
  - 속성: 제목, 감독, 개봉일
  - 기능: 모든 멤버변수의 getter/setter, 디폴트생성자, 모든 멤버변수를 초기화하는 생성자, 영화 정보를 반환하는 메소드
2) 영화 인스턴스를 3개 생성하세요
3) 모든 영화의 정보를 출력하세요

의존성 주입 (Dependency Injection, DI) 이란?

의존성 주입이란 클래스가 필요한 객체를 직접 생성하지 않고 외부에서 주입받는 방식이다.

아래 첫번째 그림은 클래스 안에서 필요한 객체를 직접 생성하는 방식이다.

두번째 그림은 의존성 주입을 사용한 방식으로, 스프링 컨테이너가 Chef 객체를 관리하고
Restaurant 클래스는 Chef 객체를 스프링으로부터 주입받아 사용한다.

직접 필요한 객체를 생성하는 방식

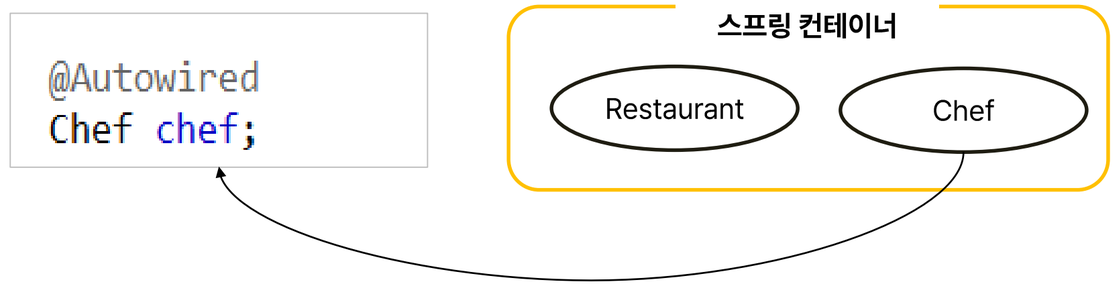
```
public class Restaurant {

    Chef chef = new Chef();

}
```

필요한 객체를 컨테이너에서 꺼내서 사용하는 방식

```
@Autowired
Chef chef;
```

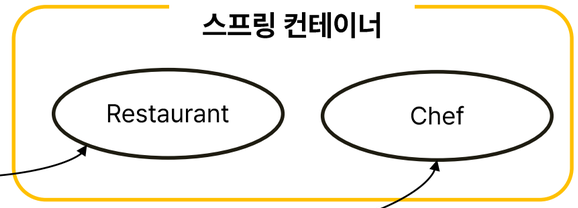


의존성 주입의 과정

1. 프로젝트가 실행될 때, 스프링 컨테이너는 @Component가 붙은 클래스를 찾아 빈(Bean)으로 등록한다.
2. 스프링 컨테이너는 해당 클래스를 기반으로 객체(빈)를 생성하고 이를 저장한다.

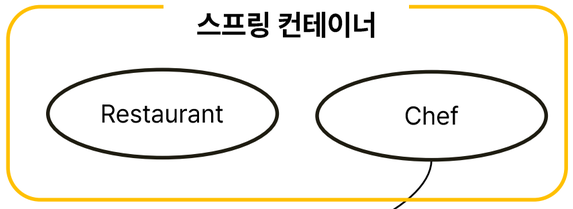
```
@Component
public class Restaurant {

@Component
public class Chef {
```



3. 스프링 컨테이너는 @Autowired가 붙은 필드를 찾아 주입할 빈을 확인한다.
4. 컨테이너에서 해당 빈을 찾아, 필드에 할당한다.

```
@Autowired
Chef chef;
```



의존성 주입은 왜 필요할까?

Chef Restaurant RestaurantTest

의존성 주입은 객체 간의 의존성을 줄이고, 유연성을 높이기 위해 필요하다.
스프링은 프로젝트가 시작될 때 필요한 객체를 미리 만들고, 이를 필요한 곳에 사용한다.
이 과정에서 클래스 간의 직접적인 의존관계가 사라지므로, 객체의 역할과 구현을 분리할 수 있다.
이렇게 하면 특정 객체를 쉽게 교체할 수 있어, 코드의 유지보수성이 높아진다.

아래 그림은 의존성 주입을 통해 chef를 선택하는 과정이다.
chef 인터페이스를 구현한 여러 클래스 중 하나(백종원)을 선택해 주입하며, 이를 통해 쉐프를 쉽게 교체할 수 있다.

레스토랑의 쉐프

백종원

이연복

백종원을 쉐프로 선택

```
@Component
public class 백종원 implements Chef {

}
```

```
public class 이연복 implements Chef{

}
```

쉐프에 백종원이 주입

```
@Autowired
Chef chef;
```

chef= 백종원 (id=103)

DI에서 제공하는 애노테이션

DI는 스프링 컨테이너에서 객체(빈)을 생성하고, 이를 관리하는 기능이다.

어노테이션	설명	위치
@Component	스프링이 해당 클래스의 인스턴스를 생성하고, 이를 빈으로 등록한다.	클래스
@Autowired	스프링 컨테이너가 해당 필드의 타입에 맞는 빈을 찾아서 필드에 주입한다.	필드

연습문제

1번.

1) 다음과 같이 강아지(Dog) 클래스를 설계하세요.
   - sound 메소드: "왕왕 짖는다" 라는 메세지를 출력
2) 스프링 컨테이너에 강아지 객체를 저장하세요.
3) 단위 테스트 클래스를 작성하세요.
   - Dog 타입의 변수를 선언하고, 스프링 컨테이너에서 해당 객체를 주입 받으세요.
   - 주입 받은 변수를 사용하여 sound() 메소드를 호출하세요.

2번.

1) 다음과 같이 고양이(Cat) 클래스를 설계하세요.
   - eat 메소드: "쥐를 먹는다" 라는 메세지를 출력
2) 스프링 컨테이너에 고양이 객체를 저장하세요.
3) 단위 테스트 클래스를 작성하세요.
   - Cat 타입의 변수를 선언하고, 스프링 컨테이너에서 해당 객체를 주입 받으세요.
   - 주입 받은 변수를 사용하여 eat() 메소드를 호출하세요.

연습문제

3번.

1) 다음과 같이 회사원(Employee) 클래스를 설계하세요.
  - work 메소드: "회사원이 일을 한다" 라는 메세지를 출력

2) 스프링 컨테이너에 회사원 객체를 저장하세요.

3) 단위 테스트 클래스를 작성하세요.
  - Employee 타입의 변수를 선언하고, 스프링 컨테이너에서 해당 객체를 주입 받으세요.
  - 주입 받은 변수를 사용하여 work() 메소드를 호출하세요.

4번.

1.다음과 같이 카페 매니저(Manager)클래스를 설계하세요.
  - 아무것도 없음

2.다음과 같이 카페(Cafe)클래스를 설계하세요.
  - 속성: 카페 매니저

3.스프링 컨테이너에 카페와 매니저 객체를 저장하세요.

4.단위 테스트 클래스를 작성하세요
  - Café 타입의 변수를 선언하고, 스프링 컨테이너에서 객체를 해당 주입 받으세요.
  - Manager 타입의 변수를 선언하고, 스프링 컨테이너에서 해당 객체를 주입 받으세요.
  - 카페와 매니저 객체를 사용하여 매니저의 주소를 확인하세요.

연습문제

5번.

1) 다음과 같이 의사(Doctor) 클래스을 설계하세요.
   - 아무것도 없음
2) 다음과 같이 병원(Hospital) 클래스을 설계하세요.
   - 속성: 담당의사
3) 스프링 컨테이너에 병원과 의사 객체를 저장하세요.
4) 단위 테스트 클래스를 작성하세요
   - Hospital 타입의 변수를 선언하고, 스프링 컨테이너에서 해당 객체를 주입 받으세요.
   - Doctor 타입의 변수를 선언하고, 스프링 컨테이너에서 해당 객체를 주입 받으세요.
   - 병원과 의사 객체를 사용하여 의사의 주소를 확인하세요

연습문제

6번.

1) 다음과 같이 선생님(Teacher) 인터페이스를 설계하세요.
   - 추상 메소드: teach()
2) Teacher 인터페이스를 구현하는 두 개의 클래스를 설계하세요.
   - MathTeacher 클래스: teach 메소드에서 "수학 선생님이 수업을 가르칩니다" 라는 메세지를 출력
   - ScienceTeacher 클래스: teach 메소드에서 "과학 선생님이 수업을 가르칩니다" 라는 메세지를 출력
3) 스프링 컨테이너에 MathTeacher와 ScienceTeacher 객체를 저장하세요.
4) 단위테스트 클래스를 작성하세요.
   - Teacher 타입의 변수를 선언하고, MathTeacher 객체 또는 ScienceTeacher 객체를 주입 받으세요.
   - 주입 받은 변수를 사용하여 teach 메소드를 호출하세요.

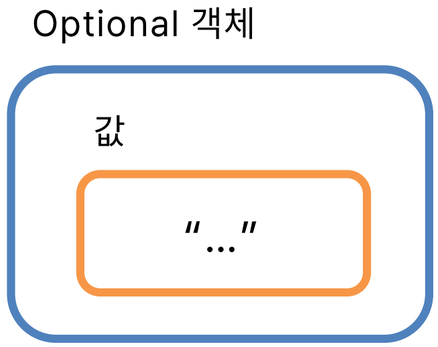
Optional 클래스

Optional 클래스란?

OptionalTest

Optional 클래스란?

Optional 클래스는 null 값을 안전하게 처리하기 위한 클래스이다.
값이 있을 수도 있고 없을 수도 있는 상황에서 NullPointerException을 방지하고,
값의 존재 여부에 따라 적절하게 처리할 수 있게 도와준다.



Optional 클래스의 주요 메소드

of(): 값을 포함하는 Optional 객체를 생성한다. 값이 null이면 예외가 발생한다.
ofNullable(): 값이 null일 수 있는 경우, 안전하게 Optional 객체를 생성한다.
isPresent(): Optional 객체에 값이 존재하면 true, 없으면 false를 반환한다.
ifPresent(): 값이 있을 때만 특정 작업을 수행한다.
orElse(): 값이 없을 경우 기본 값을 반환한다.
get(): Optional 객체에 있는 값을 반환한다.
    값이 없으면 NoSuchElementException이 발생한다.