OOP

OOP (Object Oriented Programming): 객체 지향 프로그래밍

* 소프트웨어를 어떤 기능, 논리, 절차의 흐름보다는 데이터나 객체를 중심으로 사고하고 설계하는 것
* Ex): Class: Human/ Objects: Name/ Properties: Email, Address/ Methods: Verify, send mail
* 사람이란 데이터의 첫 정의는 클래스라고 부른다.
* 하지만 모두 사람이니까 이제 구분할 수 있어야 한다. 그래서 사용 것이 이름, 그리고 이 이름이 즉 클래스의 객체, 오브젝트 혹은 인스턴스라고 부른다.
* 사람은 보편적인 어떤 속성들을 가지고 있다. 그것은 필드라고도 하고 프로퍼티라고도 하고 여러 가지 멤버라고도 부른다. 이메일, 주소, 나이, 키 등 여러가지 정보들이 될 수 있다.
* 이 클래스(사람)만이 할 수 있는 동작이 있다. 그것들은 메소드라고 부른다. 이메일을 보낸다, 걷는다, 음식을 먹는다 등이 메소드이다.
* 이렇게 하나의 단단한 클래스를 정의하고 또 여러 가지 서로 다른 도메인들을 다루는 클래스들을 정의해서, 이 클래스 간의 상호작용을 만들고 거기에서부터 비즈니스 로직을 풀어나가는 것이 OOP이다.

OOP로 만든 정렬 구현체: BubbleSort \*코드 참조

설명: 입력 인자로 리스트를 받아서 그걸로 새로운 ArrayList를 만들고 result로 넣는다. 입출력을 메모리에 구분 해 놓았기 때문에 출력에 변화를 줘도 입력에 영향이 가지 않는다.

정렬은 전형적인 버블정렬 오름차순 정렬

객체 지향 설계:

* SRP: 한 클래스는 하나의 책임만 가져야 한다.
* OCP: 소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다.
* LSP: “프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다.”
* ISP: “특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다.”
* DIP: 프로그래머는 “추상화에 의존해야지, 구체화에 의존하면 안된다.”

DIP를 이용해 내부 코드를 건드리지 않고 구현체를 갈아 끼울 수 있다:

BubbleSort => JavaSort

Sort라는 인터페이스 생성으로 메인 클래스가 버블 소트에 간결합 되어있는 걸 느슨하게 바꿔준다.

Spring Framework

@Configuration: 이 annotation 인해 Config class는 spring 전용 configuration class가 되는 것이다. 이로 인해 Spring application의 context가 뜰 때 해당 config에 어떤 설정들이 잡혀 있는지 Spring이 살펴본다.

@ComponentScan(“패키지 경로”): 패키지 경로를 Root 삼아서 이 이하에 있는 모든 spring bean을 스캔한다.

@Component: bean으로 등록해주는 annotation

@Service: 기본적으로 @Component와 동일하다. 다만 spring에서 @Service가 붙은 클래스를 서비스 역할을 하는 빈으로 인식

Sort는 interface이기 때문에 bean으로 등록하지 않는다.

@Contoller/@RestController: @Controller를 사용했을 때는 API의 응답을 view-resolver가 view로 변환, 그래서 view를 찾아서 화면에 출력해준다.

@RestController: API를 호출하면 데이터를 있는 그대로 보내준다. 즉, response에 body-context를 그대로 넣어서 보내준다. 그때 쓰는 것이 response-body annotation이다. (@ResponseBody: 클래스에 붙이면 클래스 전체에, 메소드에 붙이면 메소드에만 적용) 해당 return을 그대로 응답에 넣어준다. @ResponseBody annotation을 포함하고 있는 controller가 @RestController annotation이다.

Spring Boot Properties

스프링 부트의 기본 기능 전체를 튜닝하는 부트 전용 설정 프로퍼티

Classpath:application.properties, application.yml로 제어 가능

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

* 부트의 기능 거의 대부분을 제어 가능
* 기본값이 세팅 되어 있어서 심지어 아무것도 쓰지 않아도 작동함
* 웹 설정/ 템플릿 엔진 설정/ 서버 설정/ 시큐리티 설정/ actuator 설정
* Devtools 설정/ testing 설정 등등 할 수 있음

Spring Boot Property는 이런 Spring Boot에서 제공하는 기능 전반에 관한 기초적인 세팅을 지원한다. 요구사항을 만족시키는 내에서는 property를 이용해서 property base로 프로젝트를 셋업 하면 거의 코드의 변화 없이 원하는 설정을 만들거나 바꿀 수 있다.

@SpringBootApplication

기본적으로 configuration 클래스이다. 즉, 하나의 Configuration Annotation의 스테레오 타입이다. 내부에 Configuration Annotation, Enable Auto Configuration 그리고 Component Scanning Annotation 3개가 담겨 있는 어노테이션이다.

* @SprringBootConfiguration: 스프링 부트용 @Configuration
* @EnableAutoConfiguration: 사전에 정의한 라이브러리 빈을 등록
* @ComponentScan: 각종 스프링 빈 annotation을 base package에서부터 스캔하여 스프링 빈으로 스프링 IoC 컨테이너에 등록

@SpringBootApplication의 속성(attribute)들

* Exclude: AutoConfiguration을 제외 시킨다. Ex) WebMvc
* excludeNmae: 이름 “문자열”로 제외 가능 \* 풀 패키지명을 다 넣어야지 가동됨 ex): “org.springframework~~~WebMvcAutoConfiguration” class path에 접근할 수 없을 때 유용하다. Import를 사용할 수 없는 상황에 쓴다.
* scanBasePackages: ComponentScan 옵션을 밖으로 빼낸다. “ 베이스 패키지명”. 그러면 Componentscan이 베이스 패키지부터 검색을 한다. 어디서부터 스캔할 거다라는 걸 지정할 수 있는데 그걸 밖으로 드러나게끔 옵션을 꺼내 놓은 것이다.
* scanBasePackageClasses
* nameGenerator
* proxyBeanMethods

SpringApplication 인스턴스를 만들어 run() 전에 각종 설정 가능

@Component vs @Configuration

@Component: “이 클래스는 커스텀 빈이다.”

* @ComponentScan 🡺 base package 에서부터 모든 @Component 검색
* 인스턴스화: 필요한 의존성을 모두 주입
* 스프링 컨테이너에 등록: 필요한 곳에 주입
* 커스텀인 이유는 클래스를 사용자가 짰기 때문에 커스텀 빈이라고 칭함
* 사용 흐름은 @ComponentScan으로 스프링 부트 프로젝트가 전체를 베이스 패키지부터 쫙 검색을 한다. 모든 @Component가 달려있는 클래스들을 찾는다. 그 후 그것들을 인스턴스화 하는데 이때 필요한 내부 의존성들을 모두 주입하고 스프링 컨테이너에 등록한다. 그리고 나중에 필요한 곳에 다 주입을 넣어주는 식이다.

참고: 빈을 만드는 방법 2+1 가지

1. @Component
2. @Bean (in @Configuration)
3. @Bean (in @Component): Lite Mode

@Component

* Class-level annotation
* 등록하려는 빈의 클래스 소스가 편집 가능한 경우 사용
* Auto-detection에 걸림

@Bean

* Method-level annotation
* 좀 더 읽기 쉬움
* 인스턴스화 하는 코드가 수동으로 작성
* 빈의 인스턴스 코드와 클래스 정의가 분리된 구조
* 따라서 외부 라이브러리, third 파티 클래스도 빈으로 등록 가능

@Component: Stereotype Annotations - @Component와 같은 기능들을 하지만 이름만 다른 annotations

컴포넌트에 해당하는 stereotype Annotations

* @Controller
* Service
* Repository

@Configuration: “이 클래스는 각종 bean 설정을 담고 있다.”

* @SpringBootApplication 이 ComponentScan을 통해 @Configuration을 찾아냄
* 안의 bean 설정(메소드)을 읽어서 스프링 컨테이너에 등록
* 필요한 곳에 주입
* 또는 각종 스프링 인터페이스의 구현에 함께 활용

Conclusion: “Annotation이 의도에 맞게 사용되었는지 잘 봐야 한다”

* Bean 설정은 @Configuration, Class bean 등록은 @Component
* 정확한 목적을 모르고 쓰면 “잘 모르겠는데 어쨌든 돌아간다” 운영의 시한폭탄
* IDE가 노란 줄 그어주면
* 돌아는 간다 (X)
* 노란 줄이 있을 수 없다(O)

Configuration Properties

Externalized Configuration: 각종 설정 값을 외부로 분리해낸 것

* 서로 다른 환경에서도 사용할 수 있음
* Application을 새로 compile 하지 않고 설정 값을 바꿀 수 있음
* 종류:
* Java properties file
* YAML
* Environment variable
* Command-line argument

외부 설정의 우선 순위: 외부설정을 읽어 들이는 순서 – 아래 설정이 위에서 읽은 것을 덮어씀

1. Default Property 🡺
2. @Comfiguration 클래스에 @PropertySource로 정의된 것 🡺
3. 설정 파일: application.properties 🡺
4. RandomValuePropertySource 🡺
5. OS 환경변수 🡺
6. Java System Property: System.getProperties() 🡺
7. JNDI 속성: java:comp/env 🡺
8. ServletContext – 초기 파라미터 🡺
9. ServletConfig – 초기 파라미터 🡺
10. SPRING\_APPLICATION\_JSON 안의 프로퍼티들 🡺
11. Command-line arguments 🡺
12. 테스트에 사용된 프로퍼티들
13. @TestPropertySource
14. Devtools 글로벌 세팅: $HOME/.config/spring-boot

설정 파일(Config data)의 우선 순위

1. JAR 패키지 안의 application.properties, application.yaml
2. JAR 패키지 안의, 프로파일이 지정된 파일: application-{profile}.properties
3. JAR 패키지 밖의 파일
4. JAR 패키지 밖의, 프로파일이 지정된 파일

@Value

* SpEL 로 프로퍼티명을 표현
* Type-safe 하지 않음
* 필드 주입 방식을 사용할 경우
* 인스턴스화 이루에 주입하므로, final을 쓸 수 없음
* 생성자 안에서 보이지 않음 (대안: @PostConstruct)
* 생성자 주입 방식이 가능
* 프로퍼티 Relaxed binding 지원
* Meta-data 없음

Environment:

* Application context에서 꺼내 오는 방법
* Environment bean을 가져오는 방법
* 눈에 잘 안 들어온

Configuration Properties

* 자바 클래스로 매핑하므로 type-safe
* 각 프로퍼티에 대응하는 meta-data 작성 가능
* Relaxed binding 지원
* 작성 방법
* 기본
* @Configuratioon 생략
* Bean 메소드
* ConstructorBinding

Lombok

“Never write another getter or equals method again”

Boilerplate code를 줄여주는 도구

생산성 향상에 기여

Lombok에서 자주 쓰이는 기능들

* @Data:
* @Getter + @Setter + @RequiredArgsConstructor + @ToString + EqualsAndHashCode
* 편함
* 근데 그래서 조심해야 함
* @Value
* 불변 객체를 만들 때 씀
* @Getter, @FieldDefaults(makeFinal=true, level=AccessLevel.PRIVATE), @AllArgsConstructor, @ToString, @EqualsAndHashCode.
* @RequiredArgsConstructor: 생성자를 만들어 주는 annotation. 필드에서 필수 필드만 골라서, 필수 필드들을 인자로 받는 생성자를 만들어준다. 필수의 기준은 final 키워드이다. Final이 붙어 있으면 생성자로 담지 않으면 컴파일 에러가 발생. 이러한 것을 부분적으로 선택해서 잡아주는 생성자이다.

주의사항:

* 각종 예상치 못한 동작
* 과도한 annotations, 관례 기반 코드 스타일: 동작 예측이 어렵다는 지적
* 명시적이고 테스트가 쉬운 코드로 회귀하려는 움직임