스프링 1주차 공부

모각소 2차 발표(01.26)

박수인

스프링의 특징 1) IOC (Inversion Of Control, 제어의 역전)

= 프로그램의 제어 흐름을 직접 제어하는 것이 아니라, “외부”에서 관리하는 것

- 기존 자바 프로그램 : 클라이언트 구현 객체(사용자)가 스스로 필요한 서버 구현 객체를 생성, 연결, 실행함

= 사용자가 프로그램의 제어 흐름을 스스로 조종함

- IOC 적용 후 : 프로그램의 제어 흐름을 이제 **AppConfig()**가 가져감

= 프로그램의 제어 흐름에 대한 권한을 모두 **AppConfig**가 가짐

- AppConfig : 전체 동작 방식을 구성(config)하기 위한, 별도의 설정 클래스

스프링의 특징 2) DI (Dependency Injection, 의존성 주입)

= 애플리케이션 실행 시점(런타임)에 “외부”에서 실제 구현 객체 생성 -> 클라이언트에 전달 -> 클라이언트와 서버의 실제 의존관계가 연결되는 것

- 기존 자바 프로그램 : 사용하는 주체(A)가 사용하려는 객체(B)를 직접 생성함 -> A와 B의 의존성 ⬆️

- DI 적용 후 : 외부(Spring)에서 객체(B)를 직접 생성하여 관리 -> A와 B의 의존성 ⬇️

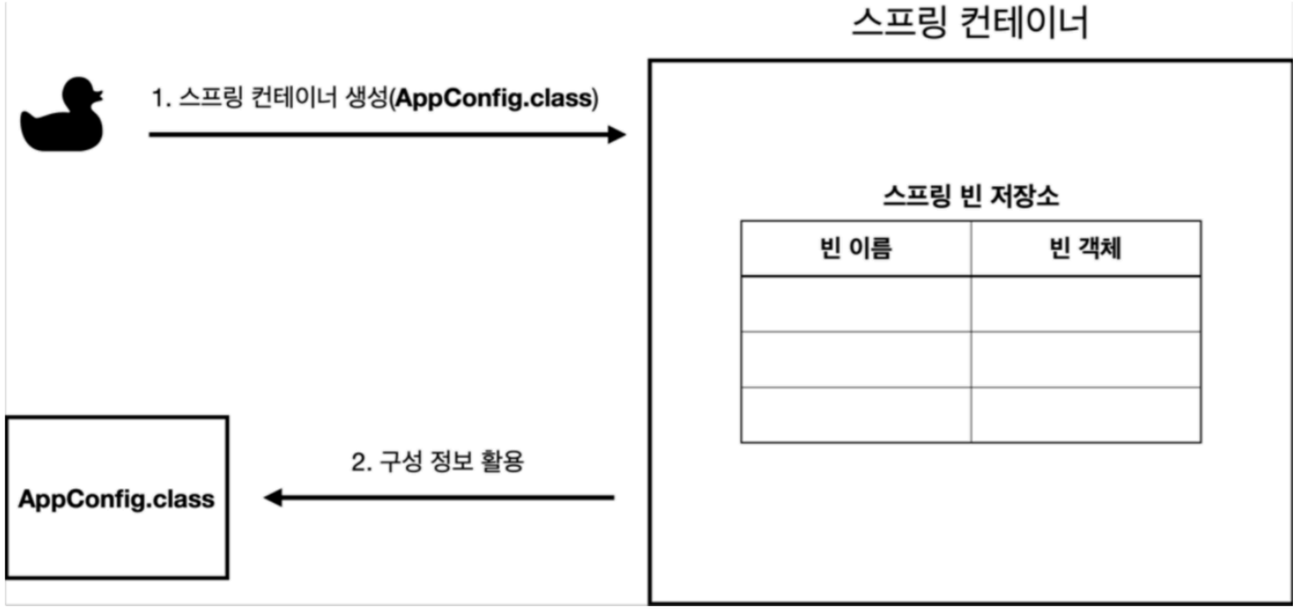
스프링 빈

= IoC 컨테이너가 관리하는 자바 객체 (Spring에 의하여 생성되고 관리되는 자바 객체)

<Spring IoC Container에 스프링 빈 등록하기>

1. IoC 컨테이너 생성

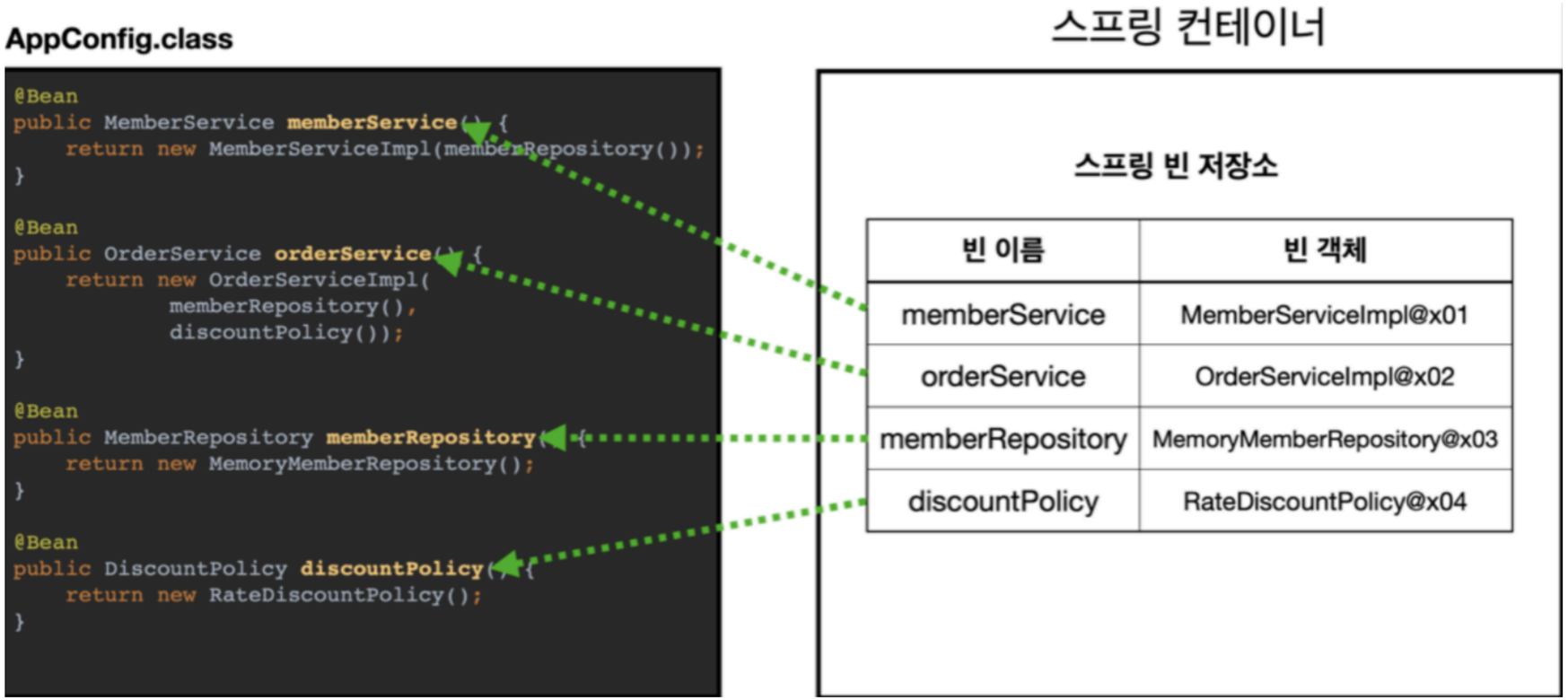
   - new AnnotationConfigApplicationContext(AppConfig.class)



스프링 빈

2. 스프링 빈 등록

- @Bean Annotation을 사용
- IoC 컨테이너는 파라미터로 넘어온 설정 클래스 정보를 사용 → 스프링 빈을 등록



스프링 빈

3. 스프링 빈 의존관계 설정

- 스프링 컨테이너는 설정 정보를 참고해서 의존관계를 주입(DI)
- IoC 컨테이너는 파라미터로 넘어온 설정 클래스 정보를 사용 → 스프링 빈을 등록

AppConfig.class

```
@Bean
public MemberService memberService() {
    return new MemberServiceImpl(memberRepository());
}

@Bean
public OrderService orderService() {
    return new OrderServiceImpl(
            memberRepository(),
            discountPolicy());
}

@Bean
public MemberRepository memberRepository() {
    return new MemoryMemberRepository();
}

@Bean
public DiscountPolicy discountPolicy() {
    return new RateDiscountPolicy();
}
```

스프링 컨테이너

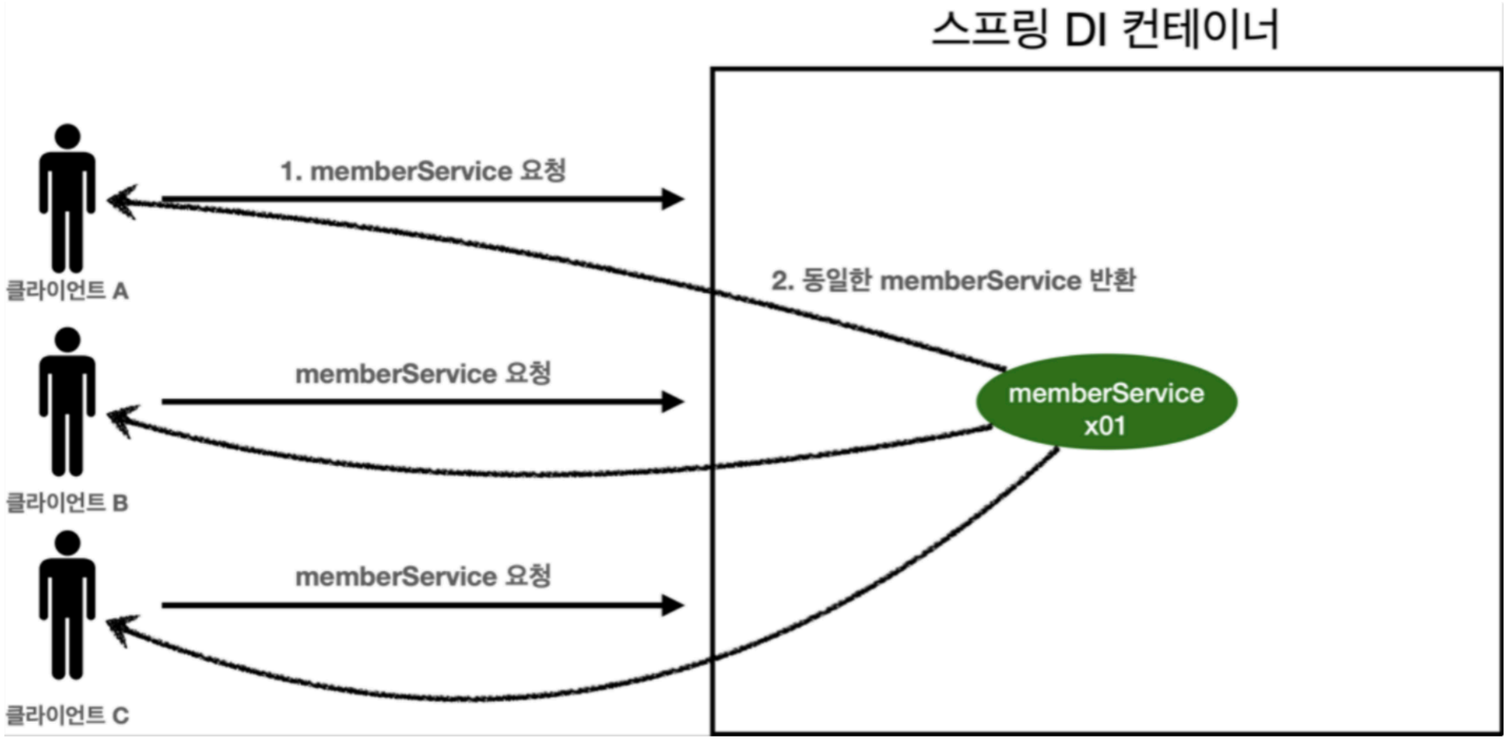
스프링 빈 저장소

빈 이름	빈 객체
memberService	MemberServiceImpl@x01
orderService	OrderServiceImpl@x02
memberRepository	MemoryMemberRepository@x03
discountPolicy	RateDiscountPolicy@x04

싱글톤 컨테이너

- **싱글톤 패턴** : 클래스의 인스턴스가 딱 1개만 생성되는 것을 보장하는 디자인 패턴
  - ➡️ 생성자를 private으로 막아서 혹시라도 외부에서 new키워드로 객체 인스턴스가 생성되는 것을 막음

- **싱글톤 컨테이너** : 싱글톤 패턴을 적용하지 않아도, 객체 인스턴스를 싱글톤으로 관리
  - 요청이 올 때마다 객체를 생성하는 것이 아니라, 이미 만들어진 객체를 공유해서 효율적으로 재사용



스프링에서 자주 사용하는 Annotation

Annotation : 클래스와 메서드에 추가 -> 다양한 기능 부여하는 역할

➡️ 역할 : 해당 클래스가 어떤 역할인지 정함 or Bean 주입함 or 자동으로 getter나 setter 생성 등

장점 : 코드량 감소 & 유지보수 용이 & 생산성 증가

- **@Component** - 개발자가 생성한 Class를 Spring의 Bean으로 등록할 때 사용
- **@ComponentScan** - 스프링 빈으로 만들기 위한 컴포넌트를 스캔할 클래스(@Component, @Repository, @Service, @Controller, @Configuration 중 등록된 클래스)를 설정하는 역할
- **@Bean** - 개발자가 제어가 불가능한 외부 라이브러리와 같은 것들을 Bean으로 만들 때 사용
- **@Controller** - 해당 클래스가 Controller의 역할을 한다고 명시
- **@AutoWired** - 개발자가 생성한 Class를 Spring의 Bean으로 등록할 때 사용
- **@RequestHeader, @RequestParam, @RequestBody** - 개발자가 생성한 Class를 Spring의 Bean으로 등록할 때 사용
- **@RequestMapping** - 개발자가 생성한 Class를 Spring의 Bean으로 등록할 때 사용