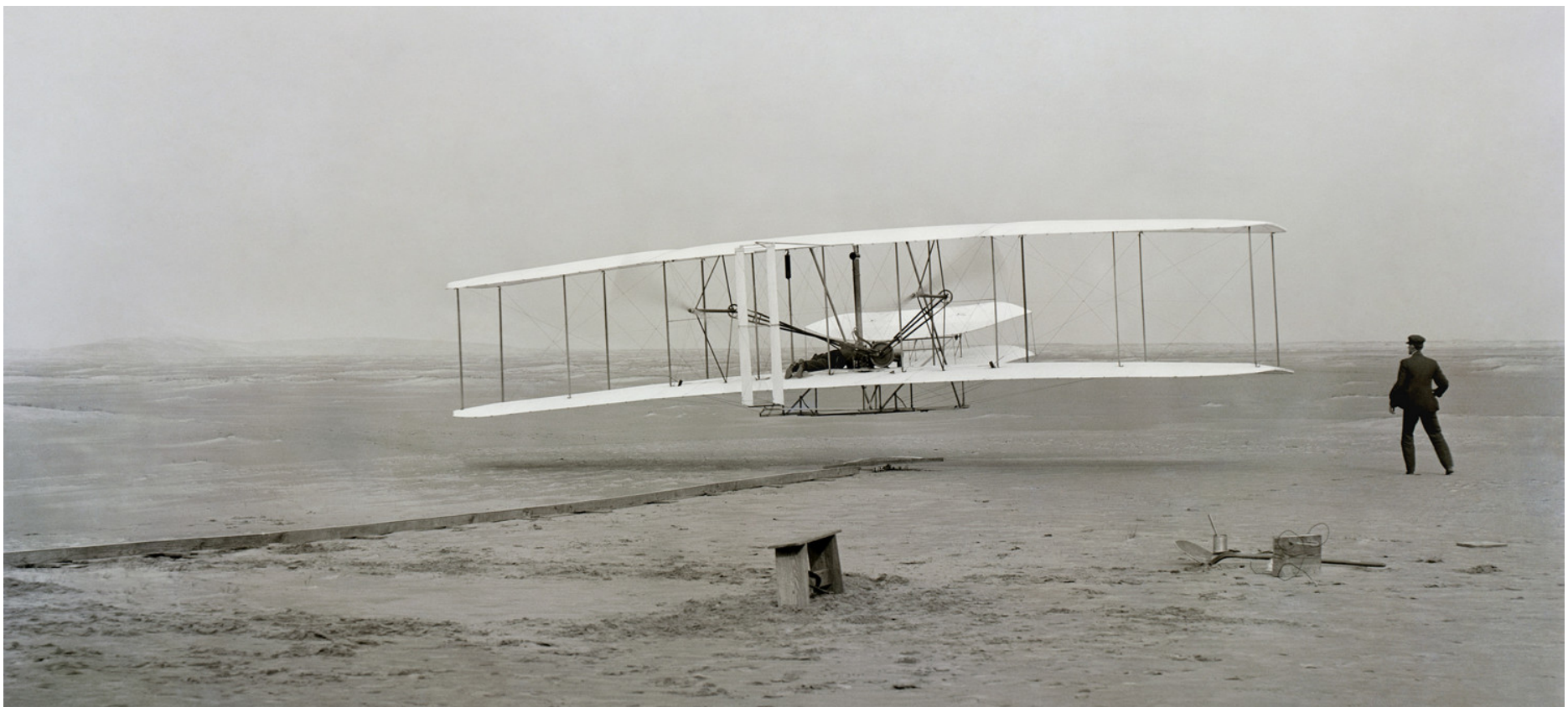


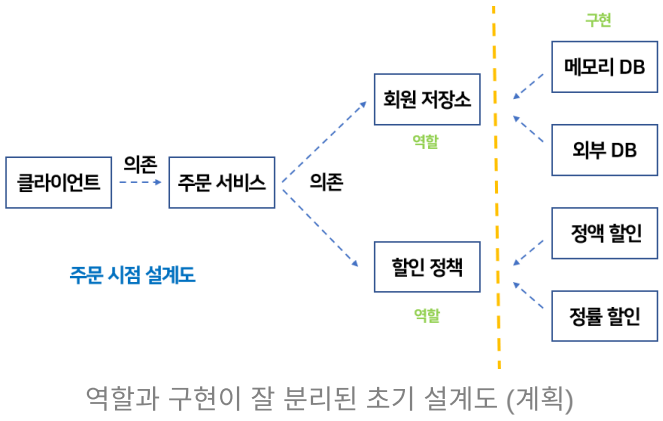
스터디 4주차(2021.09.05)

섹션 2 : 스프링 핵심 원리 이해1 - 예제 만들기

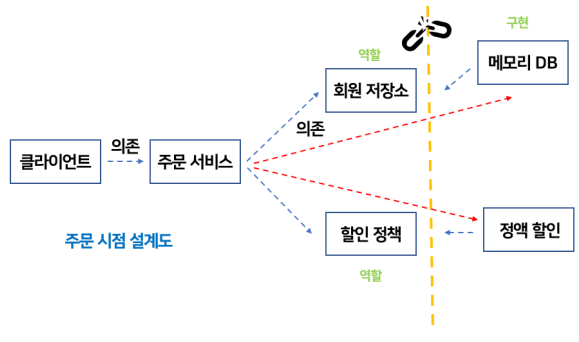
섹션 3 : 스프링 핵심 원리 이해2 - 객체 지향 원리 적용

섹션 4 : 스프링 컨테이너와 스프링 빈

예제 만들기 요약



역할과 구현이 잘 분리된 초기 설계도 (계획)



의존 파티의 상황을 설계도에 반영한 것 (오류)

```
private final MemberRepository memberRepository = new MemoryMemberRepository();
// private final DiscountPolicy discountPolicy = new FixDiscountPolicy();
private final DiscountPolicy discountPolicy = new RateDiscountPolicy();
```

의존 파티 (OrderServiceImpl)

설계도 상으로는 역할과 구현을 분리했지만, AppConfig 없이 구현한 결과를 보면 OCP, DIP를 위배

OCP(개방-폐쇄 원칙, Open-Closed Principle) ⭐⭐⭐

> 💡 SW 요소는 확장에는 열려있으나 변경에는 닫혀 있어야 한다

—> DiscountPolicy의 구현체를 활용하기에 확장은 가능하나, 변경 시 클라이언트 코드에 영향을 줌

DIP(의존관계 역전 원칙, Dependency Inversion Principle)

> 💡 추상화에 의존해야지, 구체화에 의존하면 안된다

—> DiscountPolicy에도 의존함과 동시에 RateDiscountPolicy(구현체)에도 의존하고 있음

✅ 생성자를 통해 누군가 의존 관계를 주입해줘야 한다(AppConfig)

의존성 주입

```
@Configuration
public class AppConfig {

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

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

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

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

appConfig 객체가 구현체를 적재적소로 잘 생성하고, 해당 참조값을 생성자를 통해 클래스로 넘겨준다

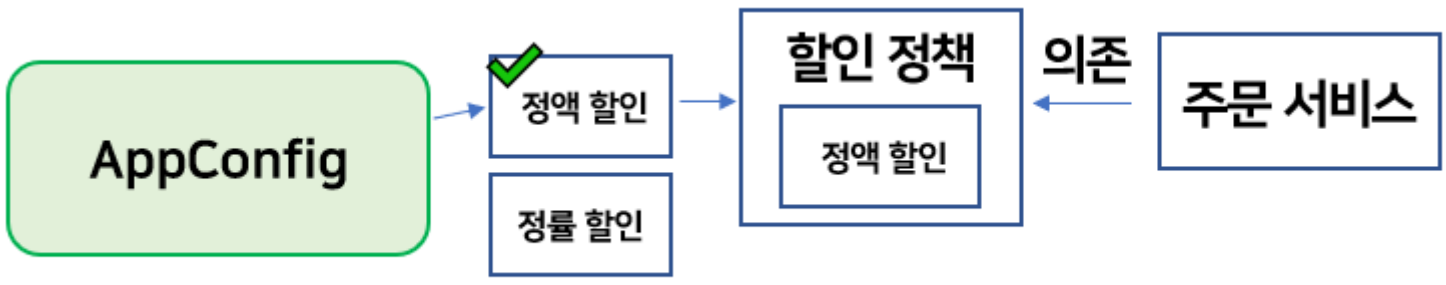
—> 클라이언트에서 보면 의존관계를 외부에서 주입하는 것이기 때문에 DI(Dependency injection)

```
private final MemberRepository memberRepository;
private final DiscountPolicy discountPolicy;

public OrderServiceImpl(MemberRepository memberRepository, DiscountPolicy discountPolicy) {
    this.memberRepository = memberRepository;
    this.discountPolicy = discountPolicy;
}
```

더이상 구현체에 의존하지 않고, 인터페이스(역할)에만 의존하고 있다.

클라이언트 입장에선 어떤 구현 객체가 주입될 지 알 수 없으며, 그저 본인의 역할에만 충실하면 됨



주문 서비스는 Config가 생성자로 주입시켜준 객체에 의존하게 된다.

> Config를 통해 SOLID 원칙 중 지킬 수 있는 3가지
>
> 1. SPR(단일 책임 원칙) :: 한 클래스는 하나의 책임만 가져야 한다.
> 👉 구현 객체를 생성하고 연결하는 책임은 Config가 맡고, 클라이언트 객체는 실행만 담당

2. DIP(의존관계 역전 원칙) :: 추상화에 의존해야한다.
👉 클라이언트는 인터페이스에만 의존하며, 생성자를 통해 주입받은 객체를 사용한다.

3. OCP(개방-폐쇄 원칙) :: 확장에는 열려있으나, 변경에는 닫혀있어야 한다.
👉 더이상 클라이언트 코드에 변경이 일어나지 않으며 Config만 변경하면 된다.

IoC, DI, 컨테이너

IoC(Inversion of Control, 제어의 역전) : 프로그램 흐름을 프레임워크가 주도하는 것, 스프링의 경우 컨테이너가 객체의 생성 - 생명주기 관리를 도맡아서 한다.

—> 기존에는 구현 객체가 필요한 객체를 생성하고 실행했음

—> 제어권이 컨테이너로 넘어가게 되기 때문에 흐름이 바뀌었다고 하여 제어의 역전

DI(Dependency Injection, 의존성 주입) : 객체간의 의존성을 자신이 아닌 외부에서 주입하는 것

—> 런타임 시점에 의존관계가 결정되기에 제 3의 존재가 필요함

—> IoC를 구현하는 하나의 방법

Container : 객체를 생성하고 관리하면서 의존관계를 연결해주는 것

스프링 컨테이너

```
ApplicationContext applicationContext = new AnnotationConfigApplicationContext(AppConfig.class);
MemberService memberService = applicationContext.getBean( name: "memberService", MemberService.class);
OrderService orderService = applicationContext.getBean( name: "orderService", OrderService.class);
```

스프링 컨테이너에 Config와 생성자들을 bean으로 등록한 뒤 호출하는 모습

ApplicationContext로 컨테이너를 선언한 뒤, getBean()를 통해 원하는 타입, 메서드를 호출한다.

—> 이때 @Configuration이 붙은 AppConfig와 @Bean annotation이 붙은 메서드가 있어야 함

—> 위의 규칙이 잘 구현된 AppConfig는 실행 시 메서드를 모두 호출해서 컨테이너에 등록함

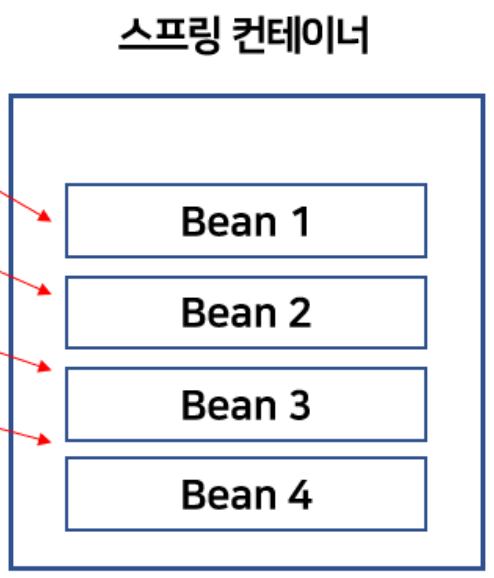
```
@Configuration
public class AppConfig {

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

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

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

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



빈 이름은 메서드 이름을 사용한다.
빈 이름은 직접 부여할 수 있다.
빈 이름은 항상 다른 이름을 부여해야 한다.

💡 동일한 타입의 스프링 빈이 두개 이상이라면, 빈 이름까지 지정해주면 된다.

getBean(JaeyoonImpl.class) —> getBean("JaeYoon", JaeyoonImpl.class)

💡 만약 해당 타입을 모두 불러오고 싶다면? ac.getBeansOfType() 메서드를 Map에 담는다.

💡 이때 부모 타입으로 조회하면, 자식 타입도 함께 조회된다. (Object의 경우 모든 스프링 빈 조회)

스프링은 컨테이너 등록 시 BeanDefinition에 읽어온 정보들을 메타정보로 생성하여 사용한다.

—> 때문에 XML이나 java 파일로 구현된 Config를 능수능란하게 처리할 수 있음 (XML은 레거시)

Source.

본 문서는 인프런의 스프링 핵심 원리 - 기본편(김영한) 강의를 수강하면서 정리한 내용입니다.

https://www.inflearn.com/course/%EC%8A%A4%ED%94%84%EB%A7%81-%ED%95%B5%EC%8B%AC-%EC%9B%90%EB%A6%AC-%EA%B8%B0%EB%B3%B8%ED%8E%B8/dashboard