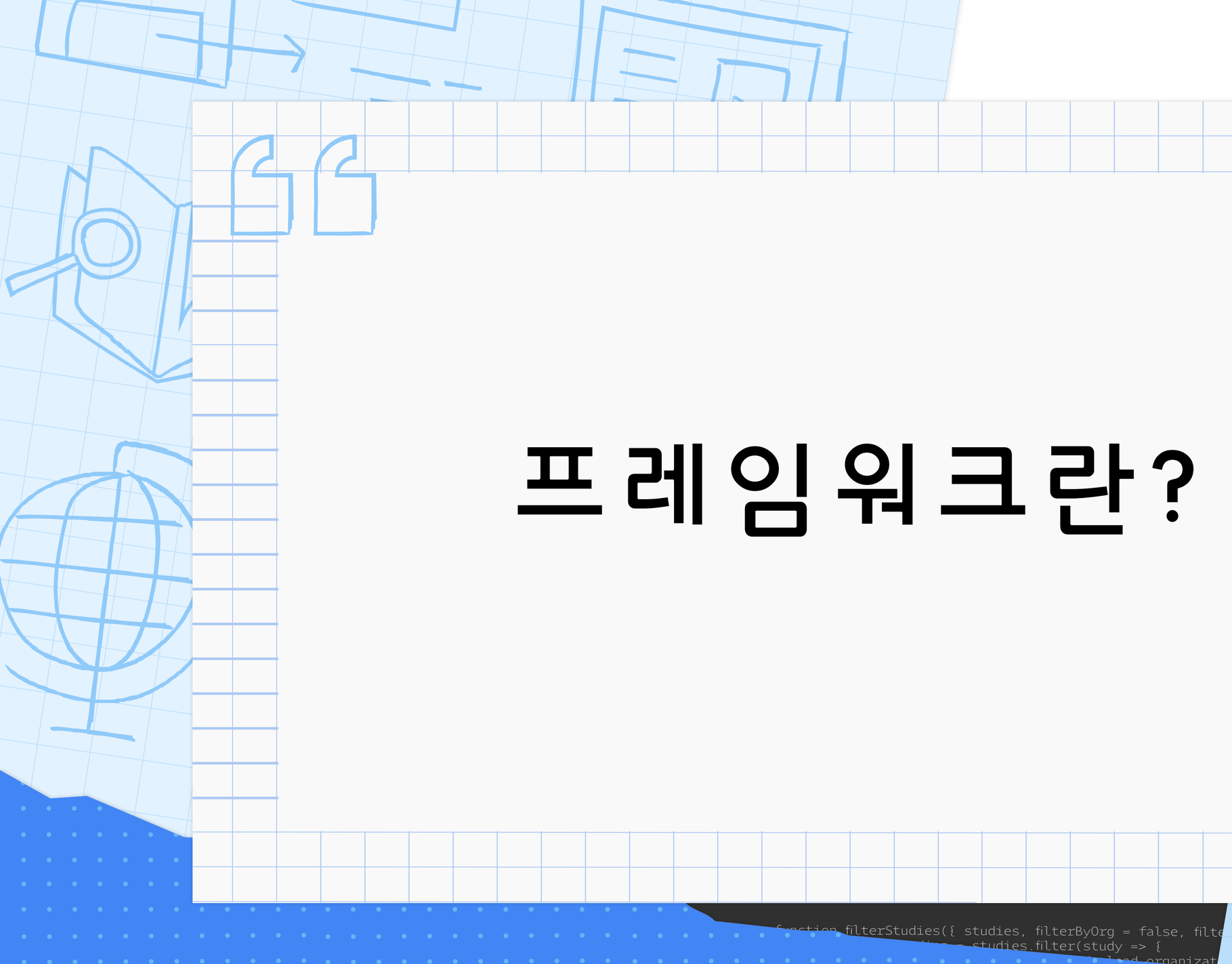
Google Developer Student Clubs

GDSC Backend 2주차 스터디

Contents

프레임워크란?


function filterStudies({ studies, filterByOrg = false, filte
studies.filter(study => {

1. 프레임워크란?

프레임워크란, 소프트웨어의 구체적인 부분에 해당하는 설계와 구현을 재사용이 가능하게끔 일련의 협업화된 형태로 클래스들을 제공하는 것

1. 프레임워크란?

프레임워크 vs 라이브러리

라이브러리란 자주 사용되는 로직을 재사용하기 편리하도록 잘 정리한 일련의 코드들의 집합을 의미합니다. (참고: 생활코딩)

쉽게 생각해서!

프레임워크 : 자동차의 프레임

라이브러리 : 자동차의 기능을 하는 부분

한 번 정해진 프레임은 바꾸지 못하고, 소형차의 뼈대로 대형차를 만들 수 없는 것! 그러나 바퀴, 헤드라이트 등은 바꿀 수 있음!

1. 프레임워크란?

프레임워크의 장점

1. 효율적.

2. 퀄리티 향상.

3. 유지 보수 Good.

1. 프레임워크란?

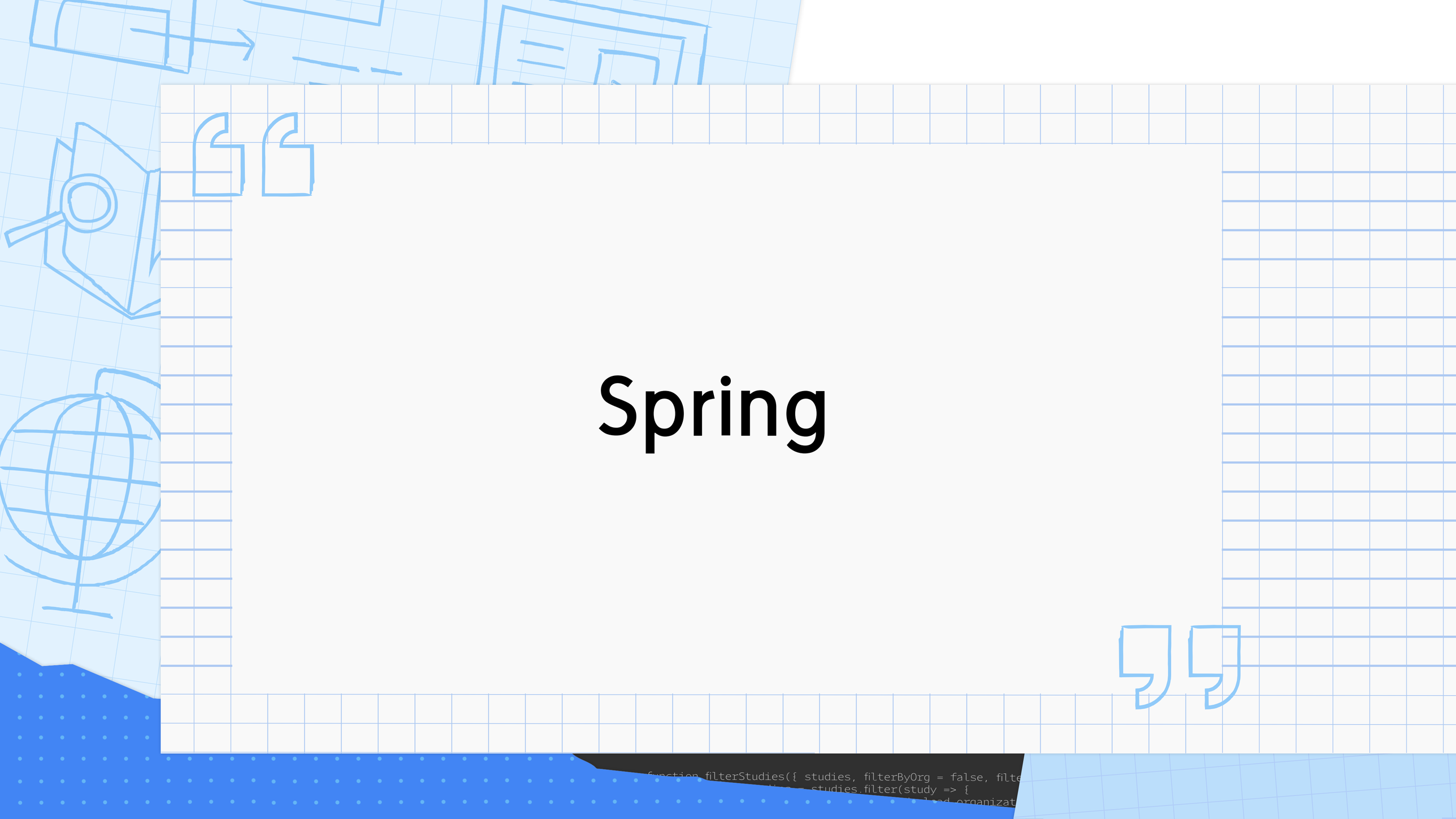
프레임워크의 단점

1. 러닝커브

2. 제작자의 의도 파악

3. 프레임워크 의존

Spring


function filterStudies({ studies, filterByOrg = false, filte
studies.filter(study => {

2. Spring

Spring?

프레임워크!

Spring

Java 언어 프레임워크 (+ Kotlin)

Java는 객체 지향 언어

스프링 – 객체지향의 특징

다형성

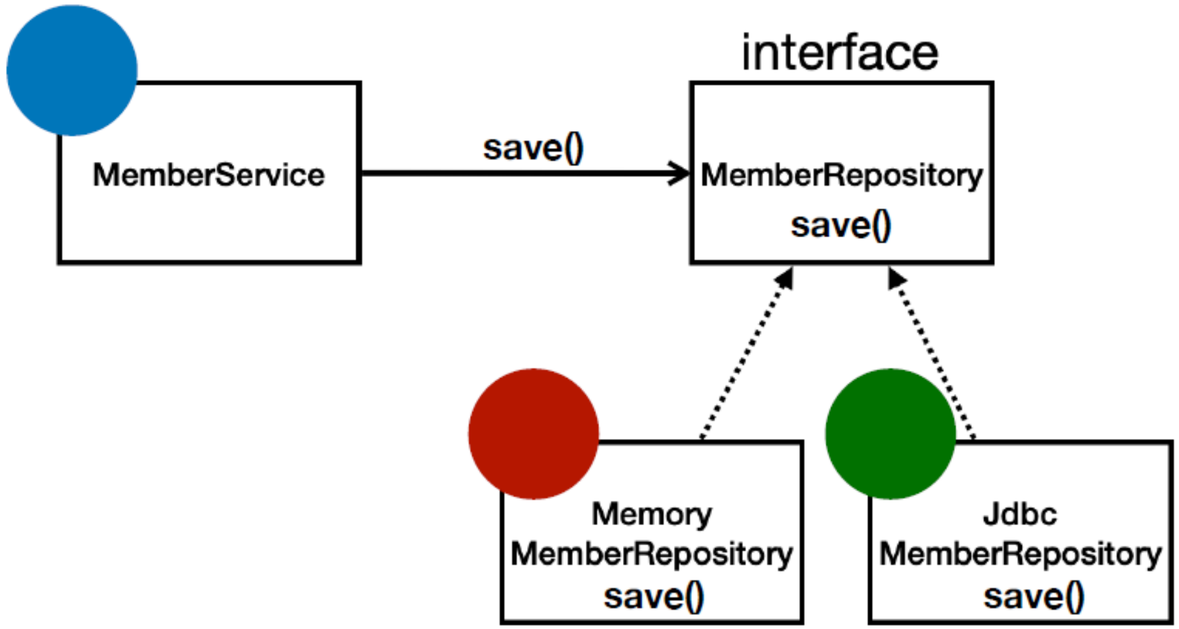
인터페이스를 구현한 객체 인스턴스를 **실행 시점**에 **유연**하게 **변경**할 수 있다.

다형성의 본질을 이해하려면 **협력**이라는 객체사이의 관계에서 시작해야 한다.

클라이언트를 변경하지 않고, 서버의 구현 기능을 유연하게 변경할 수 있다.

2. Spring

다형성


interface
MemberService
save()
MemberRepository
save()
Memory
MemberRepository
save()
Jdbc
MemberRepository
save()

정리

1. 실세계의 역할과 구현이라는 편리한 컨셉을 다형성을 통해 객체 세상으로 가져올 수 있음

2. 유연하고, 변경이 용이

3. 확장 가능한 설계

4. 클라이언트에 영향을 주지 않는 변경 가능

5. 인터페이스를 안정적으로 잘 설계하는 것이 중요

2. Spring

Spring

1. 다형성이 가장 중요하다!

2. 스프링은 다형성을 극대화해서 이용할 수 있게 도와준다.

3. IOC(제어의 역전), DI(의존성 주입)은 다형성을 활용해서 역할과 구현을 편리하게 다룰 수 있도록 지원하는 것이다.

4. 레고 블록을 조립하듯 구현을 편리하게 해주는 것이 스프링이다.

2. Spring

IoC(제어의 역전)

IoC(제어의 역전)

코드나 객체의 호출작업을 개발자가 결정하는 것이 아니라,
외부에서 결정되는 것을 의미한다.

더욱 쉽게 말하면 대신해준다(IoC)라는 뜻.

2. Spring IoC(제어의 역전)

```
class SoccerPlayer {
    2 usages
    private final NikeSoccerBall nikeBall;

    public SoccerPlayer() {
        this.nikeBall = new NikeSoccerBall();
    }

    public void playSoccer() {
        System.out.println("축구선수가 공을 찼다!");
        this.nikeBall.touchBall();
    }
}
1 usage
class AdidasSoccerBall {
    public void touchBall() {
        System.out.println("아디다스 축구공이 굴러간다!");
    }
}

2 usages
class NikeSoccerBall {
    1 usage
    public void touchBall() {
        System.out.println("나이키 축구공이 굴러간다!");
    }
}
```

```
class Driver {
    public static void main(String[] args) {
        SoccerPlayer sp = new SoccerPlayer();
        sp.playSoccer();
    }
}
```

SoccerPlayer -> playSoccer -> NikeSoccerBall
SoccerPlayer는 NikeSoccerBall 의존

2. Spring

IoC(제어의 역전)

```java
interface SoccerBall {
    1 usage  2 implementations
    void touchBall();
}
1 usage
class AdidasSoccerBall implements SoccerBall {
    1 usage
    public void touchBall() {
        System.out.println("아디다스 축구공이 굴러간다!");
    }
}
1 usage
class NikeSoccerBall implements SoccerBall {
    1 usage
    public void touchBall() {
        System.out.println("나이키 축구공이 굴러간다!");
    }
}
```

DIP.
But. Set method.

```java
class SoccerPlayer {
    2 usages
    private SoccerBall ball;

    2 usages
    public void setSoccerBall(SoccerBall ball) {
        this.ball = ball;
    }

    2 usages
    public void playSoccer() {
        System.out.println("축구선수가 공을 찼다!");
        this.ball.touchBall();
    }
}
class Driver {
    public static void main(String[] args) {
        SoccerPlayer sp = new SoccerPlayer();

        // NikeSoccerBall
        SoccerBall nikeBall = new NikeSoccerBall();
        sp.setSoccerBall(nikeBall);
        sp.playSoccer();

        // AdidasSoccerBall
        SoccerBall adidasBall = new AdidasSoccerBall();
        sp.setSoccerBall(adidasBall);
        sp.playSoccer();
    }
}
```

2. Spring

IoC(제어의 역전)

Spring Bean?

Java

객체 변수명 = new 객체();

직접 할당 및 생성

Spring Bean?

스프링 컨테이너에 의해 관리되는 객체

Ioc, DI..etc

2. Spring

IoC(제어의 역전)

Spring Container?

Ioc Container, DI Container, Bean Container

스프링의 컨테이너는 프로그래머가 작성한 코드의 처리과정을 위임받아 독립적으로 처리하는 존재이다.

컨테이너의 사전적 의미는 무언가를 담는 용기

즉 컨테이너는 객체관리를 주로 수행하는 용기 정도로 이해할 수 있다.

2. Spring

IoC(제어의 역전)

Why Spring Container?

(Java) 객체를 사용하기 위해서 new 생성자 – Setter, Getter 사용

객체가 무수히 많으면 의존성 UP, 결합도 Down

OOP에 대한 위반

의존성 제어, 즉 객체 간의 의존성을 낮추기 위해 바로 Spring 컨테이너 사용

2. Spring

IoC(제어의 역전)

BeanFactory vs ApplicationContext

BeanFactory

Bean 객체를 생성하고 관리하는 인터페이스
컨테이너 구동 시 생성 X
Lazy init

ApplicationContext

BeanFactory를 상속받은 interface + 부가적인 기능
컨테이너 구동 시 Bean 객체 스캔 후 객체화
Eager init

추가 기능

국제화 지원 텍스트 메시지 관리
이미지 파일 로드
Listener로 등록된 Bean에게 이벤트 발생 통보

2. Spring

IoC(제어의 역전)

Bean 등록 하는 방법

@Bean

개발자가 컨트롤 불가능한
주로 외부라이브러리 들에 사용
Method에 사용

@Component

개발자가 직접 컨트롤 가능한
경우 사용
Class에 사용

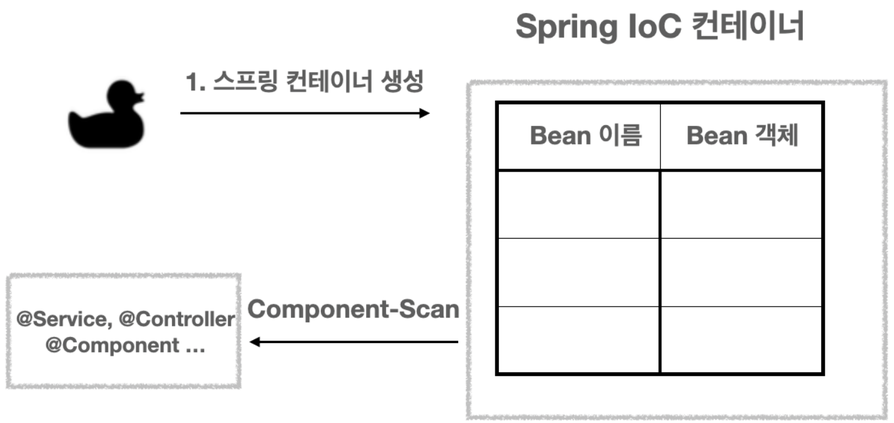
그럼 개발자가 생성한 Class에 @Bean은 선언이 가능할까?

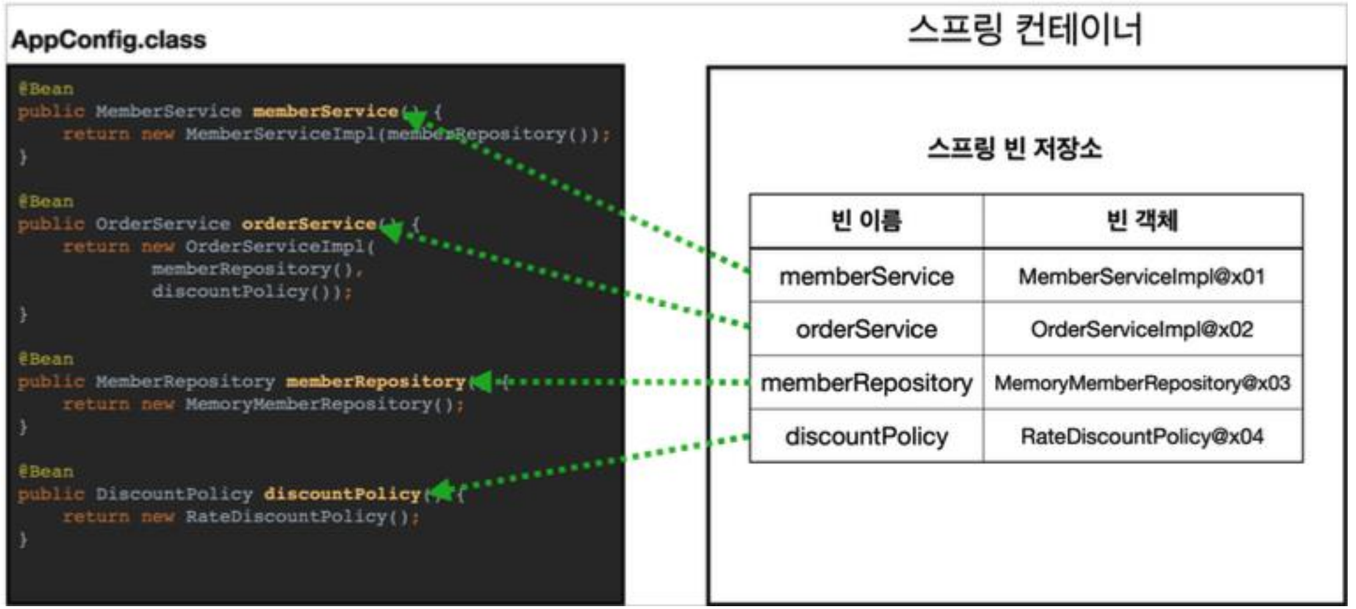
이외에도 ApplicationConfig, @Configuration, etc..
정해진 방법은 없음 – 상황에 맞춰서 사용하면 됨.

2. Spring IoC(제어의 역전)

Bean LifeCycle

스프링 IoC 컨테이너가 Bean 관리





2. Spring

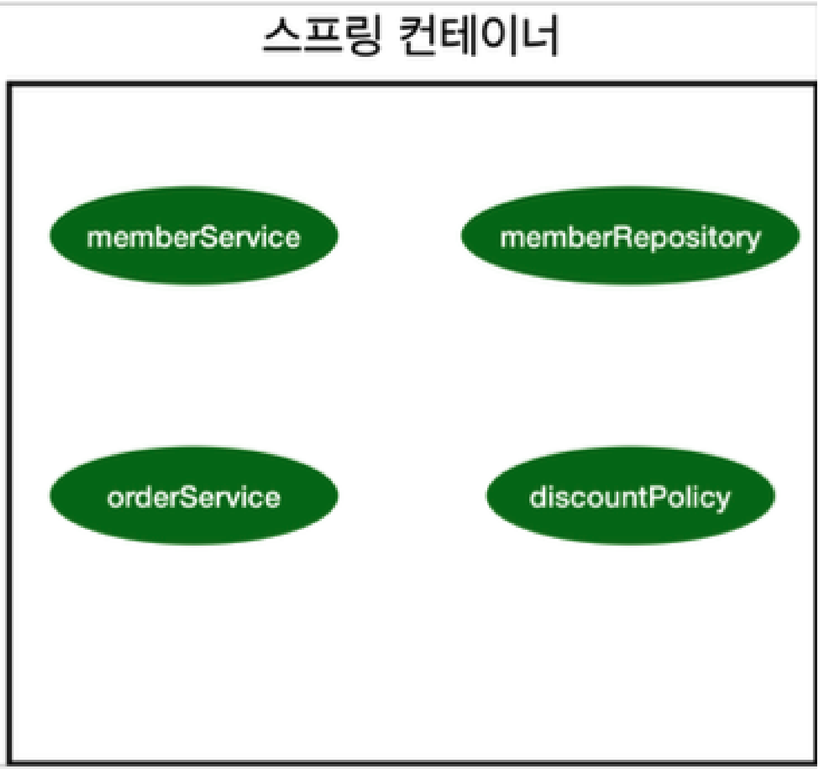
IoC(제어의 역전)

Bean LifeCycle

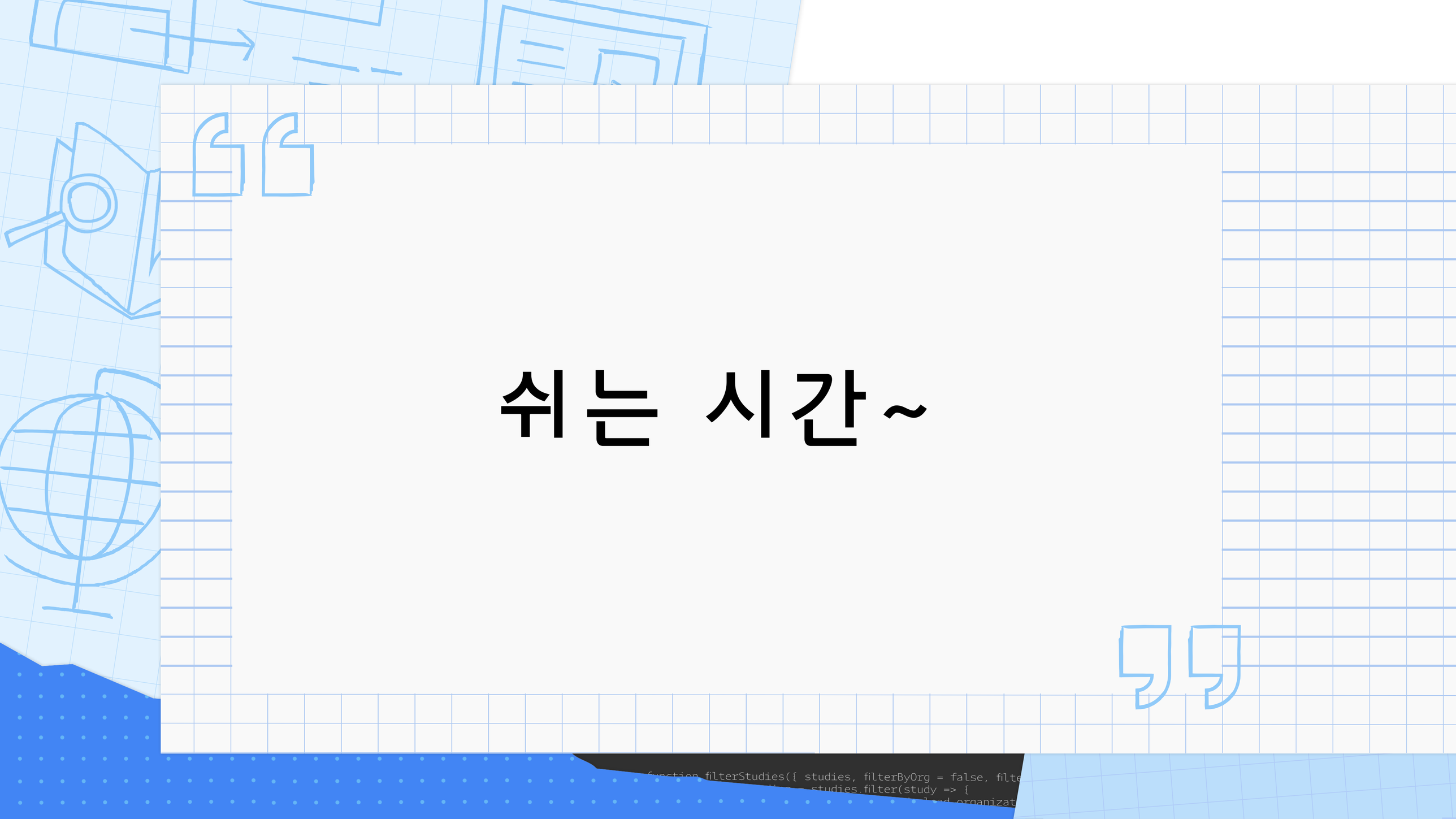
의존 관계 주입 단계

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();
}
```



쉬는 시간~


function filterStudies({ studies, filterByOrg = false, filte
studies.filter(study => {

2. Spring

DI(의존성 주입)

의존성 주입 3가지 방법

필드 주입(Field Injection)

```java
@RestController
public class SoccerController {
    // SoccerPlayer라는 타입을 가진 Bean을 찾아서 주입시킴
    1 usage
    @Autowired
     private SoccerPlayer soccerPlayer;
```

1. @Autowired 사용

2. 사용이 너무 편함

3. SRP 위반 -> 너무 많은 의존성을 가짐 -> 의존성이 눈에 보이지 않음 ->bean 구현을 하나하나 전부 뜯어봐야됨-> 테스트 할 때 new로 생성 해줘야됨.

4. Final 선언 불가 -> 새로 할당 하는 시점이 있을 수 있음.

5. DI(스프링) 컨테이너와 강한 결합

2. Spring

DI(의존성 주입)

의존성 주입 3가지 방법

세터 주입(Field Injection)

```java
@RestController
public class SoccerController {
    // SoccerPlayer라는 타입을 가진 Bean을 찾아서 주입시킴
     2 usages
     private SoccerPlayer soccerPlayer;

     @Autowired
    public void setSoccerPlayer(final SoccerPlayer soccerPlayer) {
        this.soccerPlayer = soccerPlayer;
    }
```

1. @Autowired 사용

2. Final 선언 불가

3. OCP 위반 -> 언제든지 변경 될 수 있음
-> 상황에 따라서 이점이 아예 없을 수 있다.

2. Spring

DI(의존성 주입)

의존성 주입 3가지 방법

생성자 주입(Constructer Injection)

```java
@RequiredArgsConstructor
@RestController
public class SoccerController {
    // SoccerPlayer라는 타입을 가진 Bean을 찾아서 주입시킴
    1 usage
    private final SoccerPlayer soccerPlayer;
```

1. @Autowired 생략 가능
2. Final 사용 가능 -> 불변성 보장
3. 테스트 용이함
4. Lombok이라는 라이브러리와 결합이 좋음
5. 순환 참조를 막을 수 있음.

POJO!! 궁금하면 찾아보세요~~

2. Spring

IoC(제어의 역전)

Bean LifeCycle

의존 관계 주입 단계

생성자 주입

객체 생성과 의존성 주입이 동시에 일어남

세터, 필드 주입

객체 생성 후 의존성 주입이 일어남

2. Spring

IoC(제어의 역전)

Bean LifeCycle

의존 관계 주입 단계

```
private final SoccerPlayer soccerPlayer;

public SoccerController(final SoccerPlayer soccerPlayer) {
    this.soccerPlayer = soccerPlayer;
}
```

```
class demo{
    SoccerController soccerController = new SoccerController(new SoccerPlayer());
}
```

객체 생성과 의존성 주입이 동시에 일어남

2. Spring

IoC(제어의 역전)

Bean LifeCycle

생성자 주입 이점

1. null을 주입하지 않는 한 NullPointerException은 발생하지 않는다.

2. 의존관계를 주입하지 않은 경우 객체를 생성할 수 없다. 즉, 의존관계에 대한 내용을 외부로 노출시킴으로써 컴파일 타임에 오류를 잡아낼 수 있다.

2. Spring

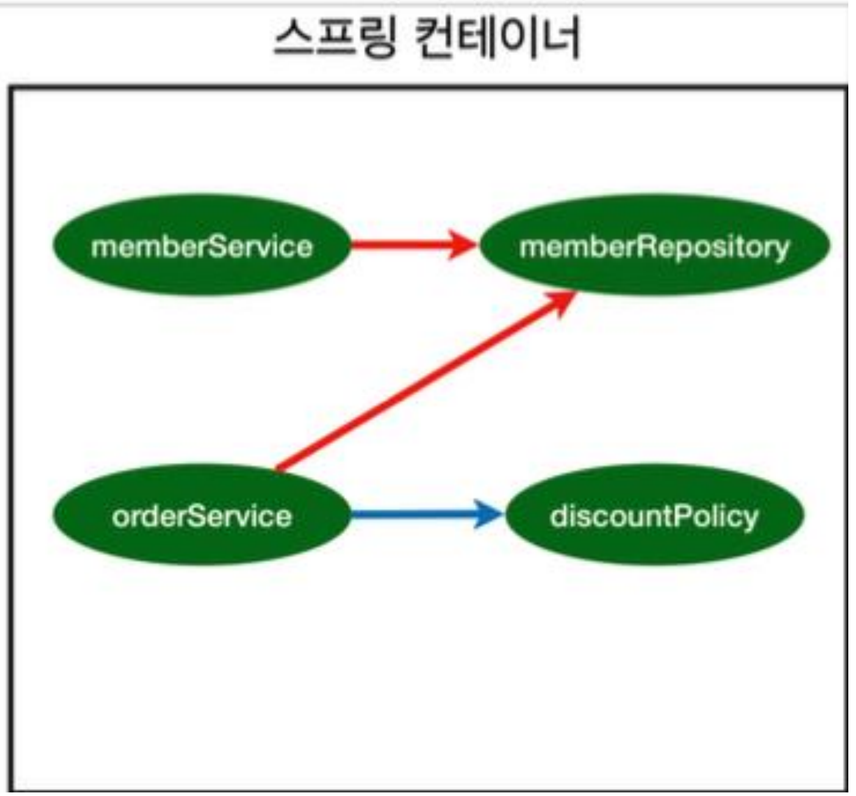
IoC(제어의 역전)

Bean LifeCycle

의존성 주입

AppConfig.class

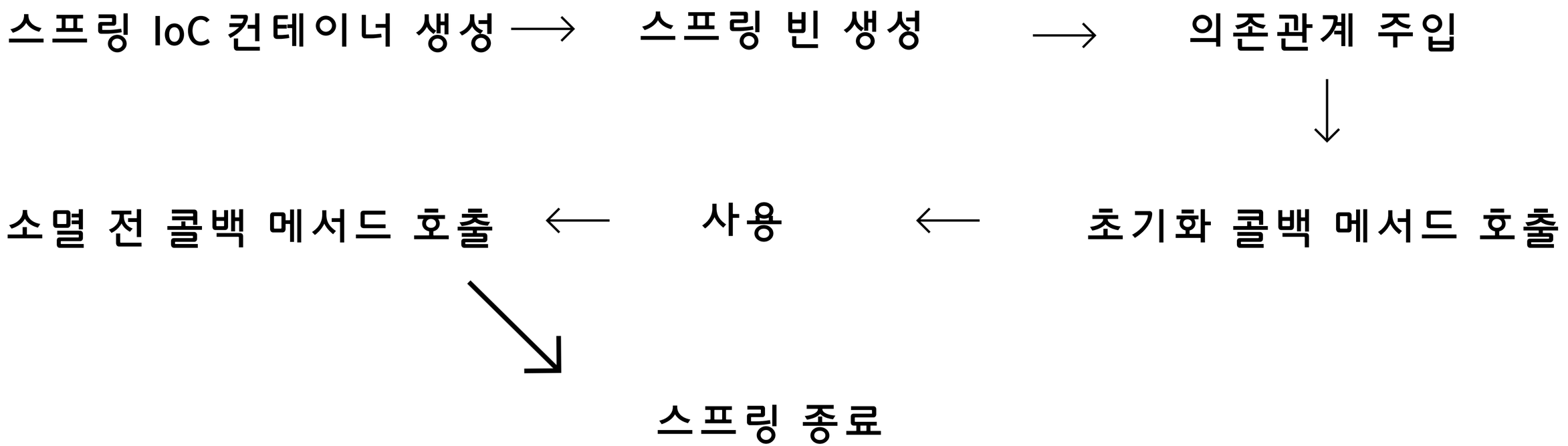
```
@Bean
public MemberService memberService() {
    return new MemberServiceImpl(memberRepository());
}

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

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

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



2. Spring

IoC(제어의 역전)

Bean LifeCycle



2. Spring

IoC(제어의 역전)

Bean LifeCycle

그러면 스프링 빈 라이프 사이클을 압축시키기 위해 생성자 주입을
통해 빈 생성과 초기화를 동시에 진행하면 되지 않을까?

2. Spring

IoC(제어의 역전)

Bean LifeCycle

객체의 생성과 초기화를 분리하자.

생성자는 파라미터를 받고, 메모리 할당을 책임.
초기화는 생성된 값을 외부 커넥션에 연결하는 책임. -> 무거움

따라서, 생성자 안에서 무거운 작업을 하지말자!
명확하게 나눠야 유지보수 관점에서 좋다!

2. Spring

IoC(제어의 역전)

Bean LifeCycle
Bean 생명주기 콜백 관리

1. 인터페이스(InitializingBean, DisposableBean)

2. 설정 정보에 초기화 메서드, 종료 메서드 지원

3. @PostConstruct, @PreDestory 어노테이션지원

2. Spring

IoC(제어의 역전)

Bean LifeCycle

@PostConstruct, @PreDestory 어노테이션지원

```
class ExampleBean {
    @PostConstruct
    public void initialize() throws Exception {
        // 초기화 콜백 (의존관계 주입이 끝나면 호출)
    }
    @PreDestroy
    public void close() throws Exception {
        // 소멸 전 콜백 (메모리 반납, 연결 종료와 같은 과정)
    }
}
```

1. 스프링 권장 방식

2. 어노테이션으로 간편함

3. ComponentSacn과 잘어울림

2. Spring

Spring Code!

2. Spring IoC(제어의 역전)

```java
interface SoccerBall {
    1 usage  2 implementations
    String TouchBall();
}
@Component("adidasBall") // adidasBall이란 이름을 가진 Bean으로 등록
class AdidasSoccerBall implements SoccerBall {
    1 usage
    @Override
    public String TouchBall() {
        return "아디다스 축구공이 굴러간다!";
    }
}

@Component("nikeBall") // nikeBall이란 이름을 가진 Bean으로 등록
class NikeSoccerBall implements SoccerBall {

    1 usage
    @Override
    public String TouchBall() {
        return "나이키 축구공이 굴러간다!";
    }
}
```

SoccerBall

```java
@Component // 의존성을 주입받는 객체도 Bean으로 등록되어야 한다.
public class SoccerPlayer {
    2 usages
    private final SoccerBall ball;

    public SoccerPlayer(@Qualifier("nikeBall") SoccerBall ball) {
        this.ball = ball;
    }

    1 usage
    public String playSoccer() {
        return "축구선수가 공을 찼다! \n" + this.ball.TouchBall();
    }
}
```

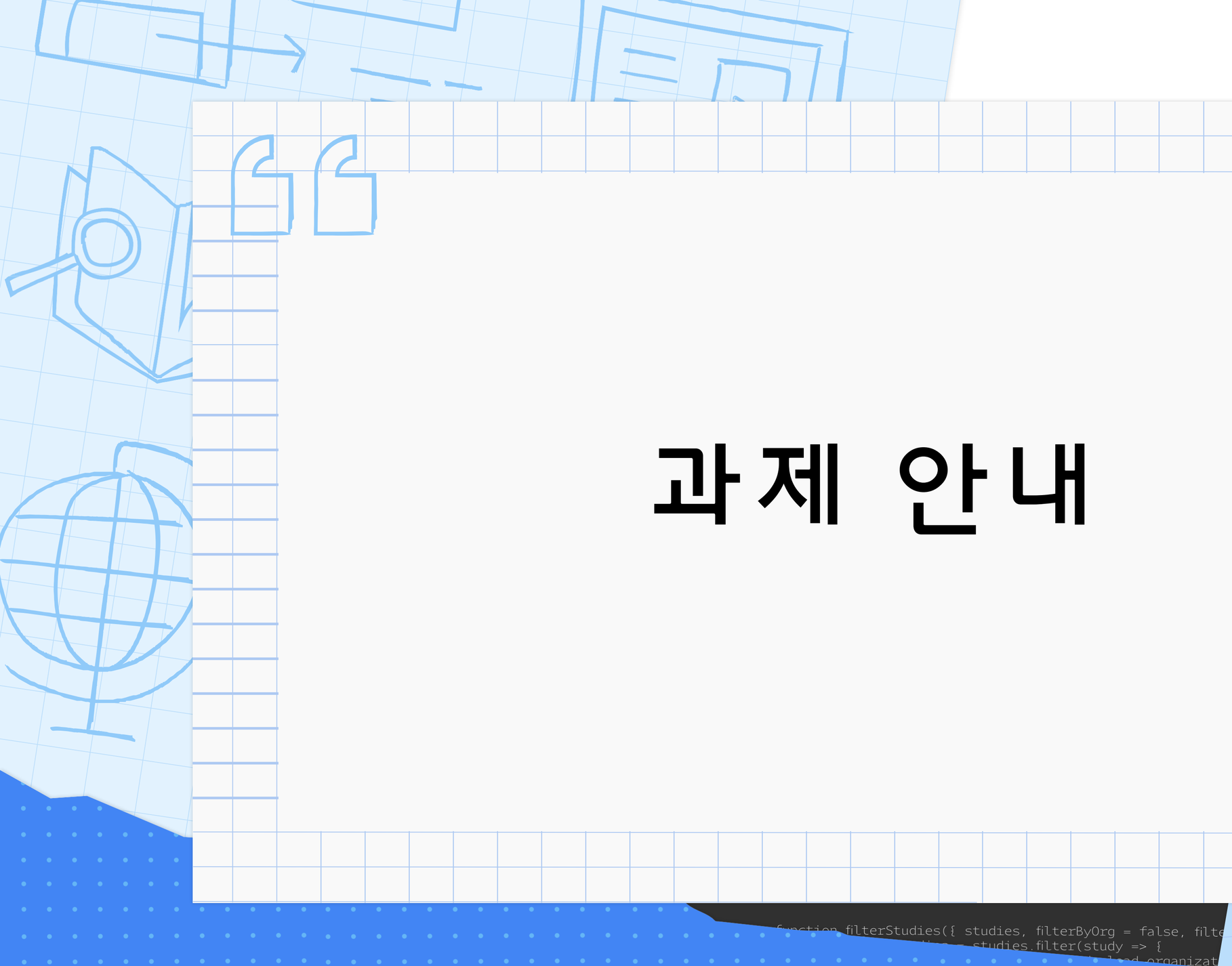
SoccerPlayer

IoC(제어의 역전)

```java
@RestController
public class SoccerController {
    // SoccerPlayer라는 타입을 가진 Bean을 찾아서 주입시킴
     2 usages
     private final SoccerPlayer soccerPlayer;

    public SoccerController(SoccerPlayer soccerPlayer) {
        this.soccerPlayer = soccerPlayer;
    }

    @RequestMapping("/soccer")
    public String soccerDriver() {
        return soccerPlayer.playSoccer();
    }
}
```

SoccerController

과제 안내


function filterStudies({ studies, filterByOrg = false, filte
studies.filter(study => {

시험 끝나고 배울 것!

Bean 싱글톤

Bean 스코프

다양한 Bean 조회

Spring MVC

다들 시험 잘보세요~

안보시는 분들은 복습 and 과제 열심히!