

# 6

## Chapter 6. 싱글톤 컨테이너

### 웹 애플리케이션과 싱글톤

- 대부분의 스프링 애플리케이션은 웹 애플리케이션이다.
- 웹 애플리케이션은 보통 여러 고객이 동시에 요청을 한다.  
고객이 3번 요청을 하면 객체가 3번 생성됨 (문제임)

```

8  public class SingletonTest {
9
10     @Test
11     @DisplayName("스프링 없는 순수한 DI 컨테이너")
12     void pureContainer() {
13         AppConfig appConfig = new AppConfig();
14         // 1. 조회 : 호출할 때마다 객체를 생성
15         MemberService memberService1 = appConfig.memberService();
16
17         // 2. 조회 : 호출할 때마다 객체를 생성
18         MemberService memberService2 = appConfig.memberService();
19
20         // 참조 값이 다른 것을 확인
21         System.out.println("memberService 1 = " + memberService1);
22         System.out.println("memberService 2 = " + memberService2);
23     }
24 }
25

```

Tests passed: 1 of 1 test – 12 ms

```

/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
memberService 1 = hello.core.member.MemberServiceImpl@6a79c292
memberService 2 = hello.core.member.MemberServiceImpl@37574691

```

서로 다른 객체가 생성됨을 확인

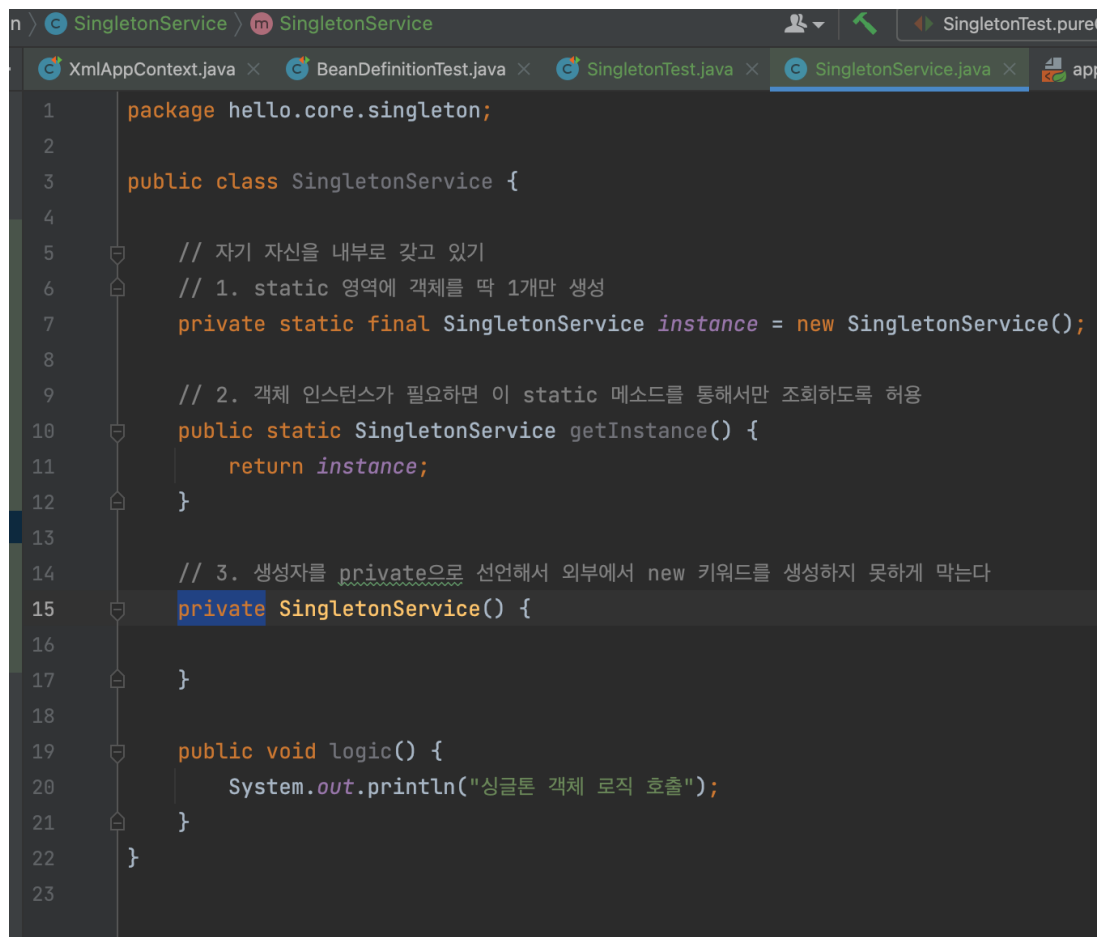
- 해결 방안은 객체는 딱 하나만 생성되고, 이 객체 인스턴스를 공유하며 사용하게 설계 !!  
→ 싱글톤 패턴

## 싱글톤 패턴

- 클래스의 인스턴스가 딱 1개만 생성되는 것을 보장하는 디자인 패턴
- 객체 인스턴스를 2개 이상 생성되지 못하도록 막아야함

- private 생성자를 사용

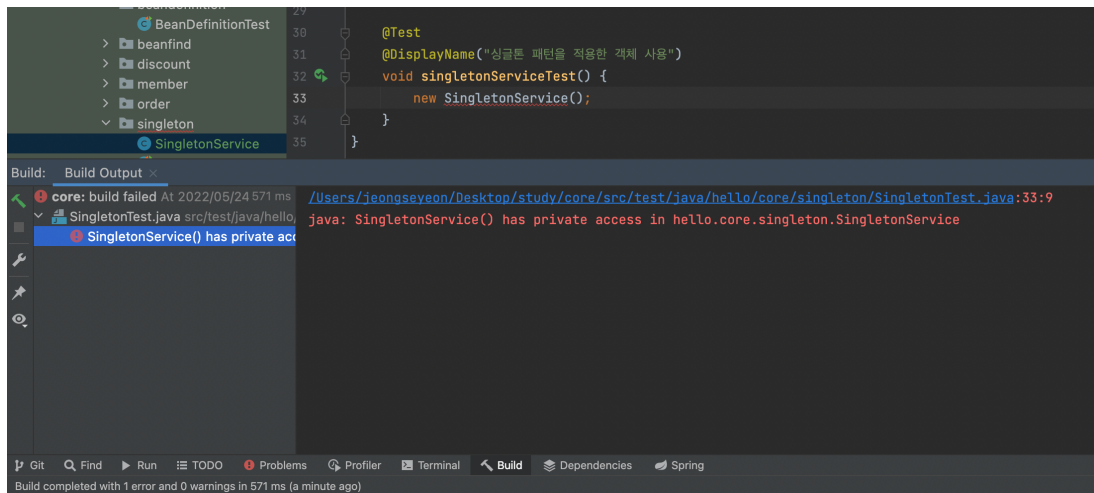
외부에서 임의로 new 키워드를 사용하지 못하도록 하기



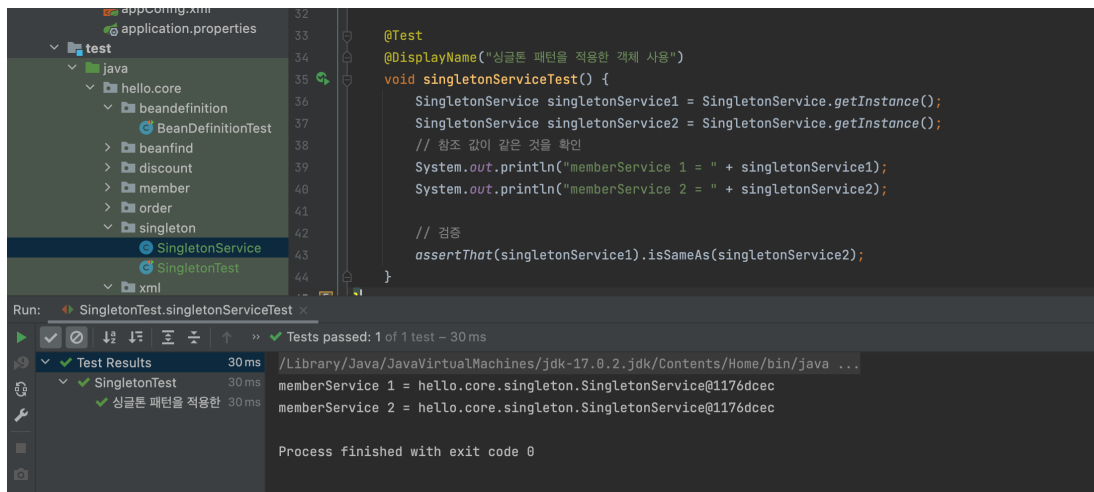
```
1 package hello.core.singleton;
2
3 public class SingletonService {
4
5     // 자기 자신을 내부로 갖고 있기
6     // 1. static 영역에 객체를 딱 1개만 생성
7     private static final SingletonService instance = new SingletonService();
8
9     // 2. 객체 인스턴스가 필요하면 이 static 메소드를 통해서만 조회하도록 허용
10    public static SingletonService getInstance() {
11        return instance;
12    }
13
14    // 3. 생성자를 private으로 선언해서 외부에서 new 키워드를 생성하지 못하게 막는다
15    private SingletonService() {
16
17    }
18
19    public void logic() {
20        System.out.println("싱글톤 객체 로직 호출");
21    }
22 }
23
```

15 라인 private 이 중요 !!

- SingletonTest.java 에서 객체 생성이 불가능함을 확인



## ◦ 싱글톤 패턴 적용



같은 객체 인스턴스를 반환함을 확인

## • 싱글톤 패턴의 한계

◦ “싱글톤”을 구현하는 코드 자체가 많이 들어감

◦ 의존관계상 클라이언트가 구체 클래스에 의존

: getInstance 한 걸 가져와야함 (구체클래스.getInstance)

→ DIP 위반

→ OCP 원칙을 위반할 가능성 높음

◦ 테스트 어려움

- 내부 속성을 변경하기 어려움
- `private` 생성자를 쓴다 → 자식 클래스를 생성하기 어려움
- 결과적으로 유연성이 떨어짐

그런데 스프링 프레임워크는 이 단점을 다 없애준다 ?!

## 싱글톤 컨테이너

---

- 스프링 컨테이너는 싱글톤 패턴의 문제점을 해결하면서, 객체 인스턴스를 싱글톤으로 관리한다.
- 스프링 컨테이너는 싱글톤 패턴을 적용하지 않아도, 객체 인스턴스를 싱글톤으로 관리
- 스프링 컨테이너는 싱글톤 컨테이너 역할을 한다.
- 이렇게 싱글톤 객체를 생성하고 관리하는 기능을 싱글톤 레지스트리 라고 한다.
- DIP, OCP, 테스트, `private` 생성자로부터 자유롭게 싱글톤을 사용할 수 있음
- 지저분한 코드 필요 없어짐

```
47
48 @Test
49 @DisplayName("스프링 컨테이너와 싱글톤")
50 void springContainer() {
51     // AppConfig appConfig = new AppConfig();
52     ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
53     // 1. 조회 : 호출할 때마다 객체를 생성
54     MemberService memberService1 = ac.getBean(name: "memberService", MemberService.class);
55     MemberService memberService2 = ac.getBean(name: "memberService", MemberService.class);
56
57     // 참조 값이 다른 것을 확인
58     System.out.println("memberService 1 = " + memberService1);
59     System.out.println("memberService 2 = " + memberService2);
60
61     // test 자동화
62     // memberService1 != memberService2
63     assertThat(memberService1).isSameAs(memberService2);
64 }
65
66
```

Tests passed: 1 of 1 test – 210 ms

00:08:23.492 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'memberService'

00:08:23.493 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'memberService'

00:08:23.499 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'memberService'

00:08:23.500 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'memberService'

memberService 1 = hello.core.member.MemberServiceImpl@5cdec700

memberService 2 = hello.core.member.MemberServiceImpl@5cdec700

## • 참고

- 스프링의 기본 빈 등록 방식은 <싱글톤> 이지만, 싱글톤 방식만 지원하는 것은 아니다.
- 요청할 때마다 새로운 객체를 생성해서 반환하는 기능도 제공한다.
- 자세한 내용은 뒤에 빈 스코프에서 설명

## 싱글톤 방식의 주의점

- 싱글톤 패턴이든, 싱글톤 컨테이너(스프링)를 사용하든, 객체 인스턴스를 하나만 생성해서 공유하는 싱글톤 방식은 여러 클라이언트가 하나의 객체 인스턴스를 공유하기 때문에,  
싱글톤 객체는 상태를 유지(stateful)하게 설계하면 안된다.
- 무상태 (stateless)로 설계 !!!

- 특정 클라이언트에 의존적인 필드 x
- 특정 클라이언트가 값을 변경할 수 있는 필드가 있으면 안된다
- 가급적 읽기만

```

1 package hello.core.singleton;
2
3 public class StatefulService {
4
5     private int price; // 상태를 유지하는 필드
6
7     public void order (String name, int price) {
8         System.out.println("name = " + name + "price = " + price);
9         this.price = price;    // 여기가 문제 !
10    }
11
12    public int getPrice() {
13        return price;
14    }
15 }

```

```

46 }
47
48 @Test
49 @DisplayName("스프링 컨테이너와 싱글톤")
50 void springContainer() {
51     // AppConfig appConfig = new AppConfig();
52     ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
53     // 1. 조회 : 호출할 때마다 객체를 생성
54     MemberService memberService1 = ac.getBean("memberService", MemberService.class);
55     MemberService memberService2 = ac.getBean("memberService", MemberService.class);
56
57     // 참조 값이 다른 것을 확인
58     System.out.println("memberService 1 = " + memberService1);
59     System.out.println("memberService 2 = " + memberService2);
60
61     // test 자동화
62     // memberService1 != memberService2
63     assertThat(memberService1).isSameAs(memberService2);
64 }
65 }
66

```

Tests passed: 1 of 1 test - 217 ms  
 00:30:39.549 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton  
 00:30:39.550 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton  
 00:30:39.550 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton  
 memberService 1 = hello.core.member.MemberServiceImpl@5cdec700  
 memberService 2 = hello.core.member.MemberServiceImpl@5cdec700  
 Process finished with exit code 0

- StatefulService 의 price 필드는 공유되는 필드인데, 특정 클라이언트가 값을 변경한다.
- 공유 필드는 항상 조심 해야함
- 스프링 빈은 <무상태>로 설계
- 무상태 설계

```

1 package hello.core.singleton;
2
3 public class StatefulService {
4
5     // private int price; // 상태를 유지하는 필드
6
7     public int order(String name, int price) {
8         System.out.println("name = " + name + " price = " + price);
9         // this.price = price; // 여기가 문제 !
10        return price;
11    }
12

```

private 영역 주석, public에 return price 로 변경

```

11 class StatefulServiceTest {
12     @Test
13     void statefulServiceSingleton() {
14         ApplicationContext ac = new AnnotationConfigApplicationContext(TestConfig.class);
15         StatefulService statefulService1 = ac.getBean(StatefulService.class);
16         StatefulService statefulService2 = ac.getBean(StatefulService.class);
17
18         // Thread A : A 사용자 10000원 주문
19         int userAPrice = statefulService1.order( name: "userA", price: 10000);
20         // Thread B : B 사용자 20000원 주문
21         int userBPrice = statefulService2.order( name: "userB", price: 20000);
22
23         // Thread A : 사용자 A 주문 금액 조회
24         // int price = statefulService1.getPrice();
25         System.out.println("price = " + userAPrice);
26
27         // Assertions.assertThat(statefulService1.getPrice()).isEqualTo(20000);
28     }
29
30     static class TestConfig {
31         @Bean
32         public StatefulService statefulService() {
33             return new StatefulService();
34         }
35     }
36 }

```

Tests passed: 1 of 1 test - 154 ms

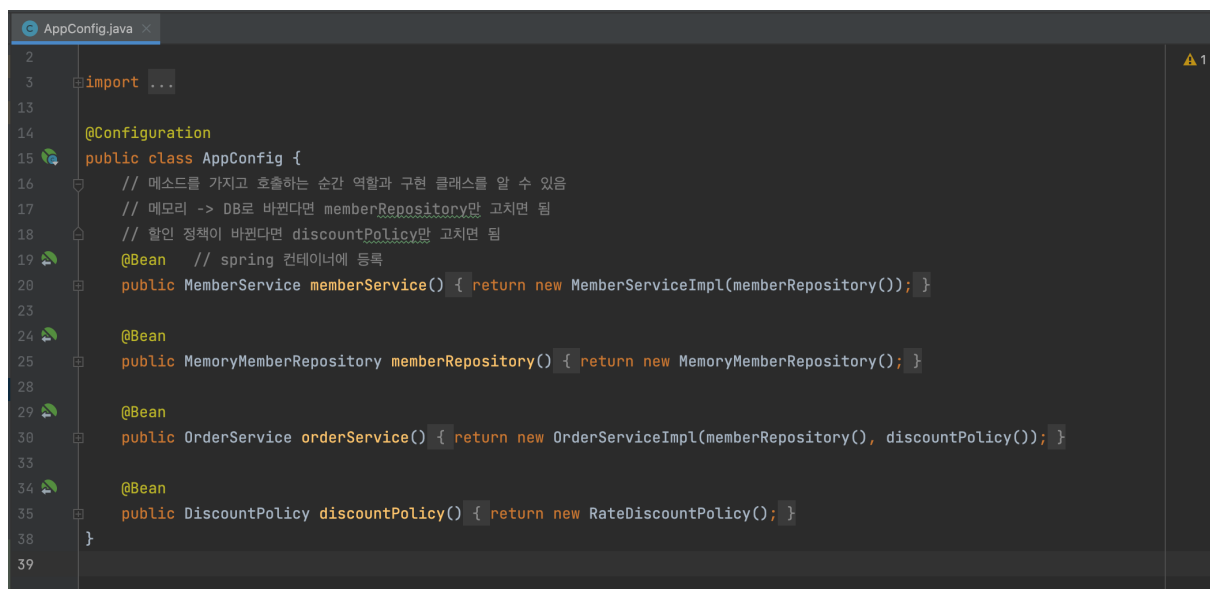
```

00:34:19.424 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton
00:34:19.426 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton
name = userAPrice = 10000
name = userBPrice = 20000
price = 10000
Process finished with exit code 0

```

## @Configuration과 싱글톤

- Configuration이란?
  - @Configuration이라고 하면 설정파일을 만들기 위한 애노테이션 or Bean을 등록하기 위한 애노테이션이다.
- 역할
  - Bean을 등록할때 싱글톤(singleton)이 되도록 보장해준다.
  - 스프링컨테이너에서 Bean을 관리할수있게 됨.



```
AppConfig.java
2
3 import ...
13
14 @Configuration
15 public class AppConfig {
16     // 메소드를 가지고 호출하는 순간 역할과 구현 클래스를 알 수 있음
17     // 메모리 -> DB로 바뀐다면 memberRepository만 고치면 됨
18     // 할인 정책이 바뀐다면 discountPolicy만 고치면 됨
19     @Bean // spring 컨테이너에 등록
20     public MemberService memberService() { return new MemberServiceImpl(memberRepository()); }
23
24     @Bean
25     public MemoryMemberRepository memberRepository() { return new MemoryMemberRepository(); }
28
29     @Bean
30     public OrderService orderService() { return new OrderServiceImpl(memberRepository(), discountPolicy()); }
33
34     @Bean
35     public DiscountPolicy discountPolicy() { return new RateDiscountPolicy(); }
38
39 }
```

- memberService 빈을 호출하면 memberRepository() 호출
- orderService 빈을 호출하면 memberRepository() 호출

→ 싱글톤이 깨지는 것이 아닌가?! (클래스 인스턴스 2개 생성)



```

10
11 public class ConfigurationSingletonTest {
12
13     @Test
14     void configurationTest() {
15         ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
16
17         MemberServiceImpl memberService = ac.getBean( name: "memberService", MemberServiceImpl.class);
18         OrderServiceImpl orderService = ac.getBean( name: "orderService", OrderServiceImpl.class);
19         MemberRepository memberRepository = ac.getBean( name: "memberRepository", MemberRepository.class);
20
21         MemberRepository memberRepository1 = memberService.getMemberRepository();
22         MemberRepository memberRepository2 = orderService.getMemberRepository();
23
24         System.out.println("memberService -> memberRepository = " + memberRepository1);
25         System.out.println("orderService -> memberRepository = " + memberRepository2);
26         System.out.println("memoryRepository = " + memberRepository);
27     }
28 }
29

```

ConfigurationSingletonTest.configurationTest

Tests passed: 1 of 1 test - 195 ms

Test Results 195 ms 00:49:55.974 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance

ConfigurationSingle 195 ms 00:49:55.981 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance

configurationTest 195 ms 00:49:55.982 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance

memberService -> memberRepository = hello.core.member.MemoryMemberRepository@30c8681

orderService -> memberRepository = hello.core.member.MemoryMemberRepository@30c8681

memoryRepository = hello.core.member.MemoryMemberRepository@30c8681

- 같은 인스턴스를 사용하는 것을 확인 !!
- 어떻게 이렇게 되는거지 ?!
  - 다음시간에 계속

## @Configuration과 바이트 코드 조작의 마법

- 스프링 컨테이너 == 싱글톤 레지스트리

```

35     @Test
36     void configurationDeep() {
37         ApplicationContext ac = new AnnotationConfigApplicationContext(AppConfig.class);
38         AppConfig bean = ac.getBean(AppConfig.class);
39
40         System.out.println("Bean = " + bean.getClass());
41     }
42 }

```

configurationDeep x

✓ Tests passed: 1 of 1 test - 223 ms

```

23:41:40.150 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig.memberService'
23:41:40.150 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig.memberRepository'
23:41:40.151 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig.orderService'
23:41:40.156 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig'
23:41:40.164 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig'
23:41:40.165 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig'
23:41:40.166 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'AppConfig'
Bean = class hello.core.AppConfig$$EnhancerBySpringCGLIB$$9db1a227

```

- 내가 만든 클래스가 아니라, 스프링이 CGLIB이라는 바이트 코드 조작 라이브러리를 사용해서 AppConfig 클래스를 상속받은 임의의 다른 클래스를 만든다
- 이 다른 클래스를 스프링 빈으로 등록함
- 이 임의의 다른 클래스가 싱글톤이 보장되도록 해준다.
- AppConfig == instance : AppConfig@CGLIB
- AppConfig@CGLIB → AppConfig
- @Configuration을 적용하고 @Bean을 적용하면 싱글톤 레지스트리가 보장됨

```

14 @Configuration
15 public class AppConfig {
16     // 메소드를 가지고 호출하는 순간 역할과 구현 클래스를 알 수 있음
17     // 메모리 -> DB로 바뀐다면 memberRepository만 고치면 됨
18     // 할인 정책이 바뀐다면 discountPolicy만 고치면 됨
19     @Bean // spring 컨테이너에 등록
20     public MemberService memberService() {
21         System.out.println("AppConfig.memberService");
22         return new MemberServiceImpl(memberRepository());
23     }
24
25     @Bean
26     public MemoryMemberRepository memberRepository() {
27         System.out.println("AppConfig.memberRepository");
28         return new MemoryMemberRepository();
29     }
30
31     @Bean
32     public OrderService orderService() {
33         System.out.println("AppConfig.orderService");
34         return new OrderServiceImpl();
35     }
36 }

```

configurationDeep x

✓ Tests passed: 1 of 1 test - 223 ms

- @Configuration이 없으면, memberRepository가 세번 호출되어 싱글톤이 깨짐

```
14 // @Configuration
15 public class AppConfig {
16     // 메소드를 가지고 호출하는 순간 역할과 구현 클래스를 알 수 있음
17     // 메모리 -> DB로 바뀐다면 memberRepository만 고치면 됨
18     // ...
19 }

configurationDeep x
Tests passed: 1 of 1 test - 155 ms

/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
23:49:19.814 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing org.springframework.context
23:49:19.820 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
23:49:19.852 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
23:49:19.852 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
23:49:19.853 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
23:49:19.854 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
23:49:19.858 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
23:49:19.859 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
AppConfig.memberService
AppConfig.memberRepository
23:49:19.861 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
AppConfig.memberRepository
23:49:19.861 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
AppConfig.orderService
AppConfig.memberRepository
23:49:19.861 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean
Bean = class hello.core.AppConfig
```