Section4. 스프링 컨테이너와 스프링 빈

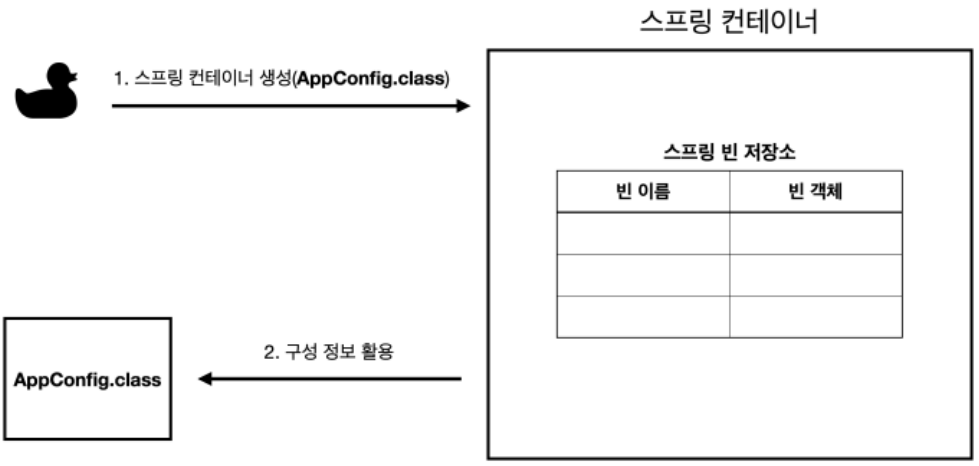
스프링 컨테이너 생성

`ApplicationContext` 는 인터페이스

스프링 컨테이너

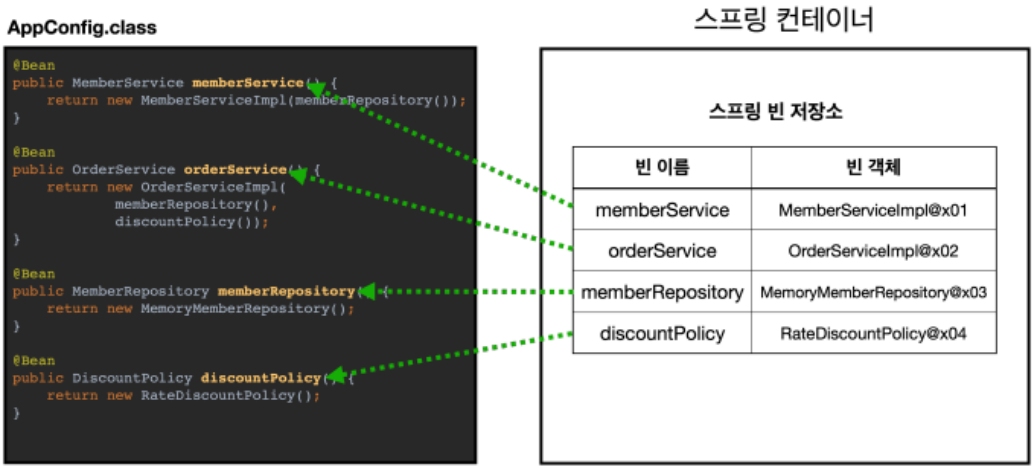
- XML 기반
- 애노테이션 기반 (선호)

1. 스프링 컨테이너 생성



- 스프링 컨테이너를 생성할 때는 구성 정보 지정 필요
  - `AppConfig.class`

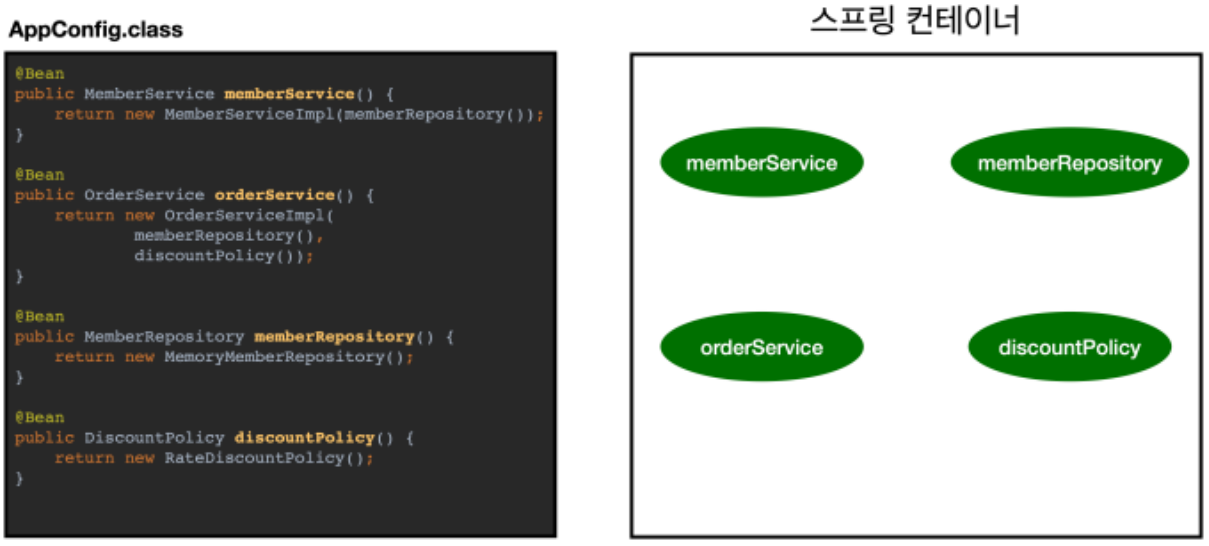
2. 스프링 빈 등록



- 파라미터로 넘어온 설정 클래스 정보를 사용해서 스프링 빈 등록

> 빈 이름은 메서드 이름을 사용한다.
> 빈 이름을 직접 부여할 수 있다.
> `Bean(name = "memberService2")`
>
> **※ 빈 이름은 항상 다른 이름 부여 ※**

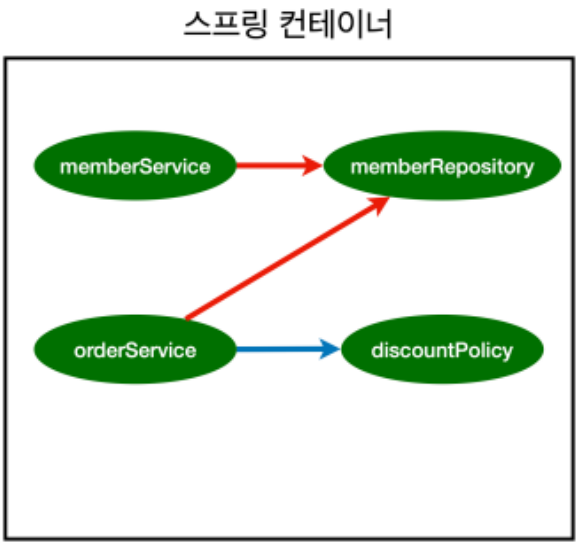
3. 스프링 빈 의존관계 설정 - 준비



4. 스프링 빈 의존관계 설정 - 완료

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 컨테이너는 설정 정보를 참고해서 의존관계 주입(DI)

컨테이너에 등록된 모든 빈 조회

💡 실습 코드 작성

▼ **Test Code**

```
public class ApplicationContextInfoTest {
    AnnotationConfigApplicationContext ac =
            new AnnotationConfigApplicationContext(AppConf

    @Test
    @DisplayName("모든 빈 출력하기")
    void findAllBean(){
        String[] beanDefinitionNames = ac.getBeanDefinitio
        for (String beanDefinitionName : beanDefinitionNam
            Object bean = ac.getBean(beanDefinitionName);
            System.out.println("name = " + beanDefinitionN
        }
    }

    @Test
    @DisplayName("애플리케이션 빈 출력하기")
```

```
    void findApplicationBean(){
        String[] beanDefinitionNames = ac.getBeanDefinitio
        for (String beanDefinitionName : beanDefinitionNam
            BeanDefinition beanDefinition = ac.getBeanDefi
            if(beanDefinition.getRole() == BeanDefinition.
                Object bean = ac.getBean(beanDefinitionNam
                System.out.println("name = " + beanDefinit
            }
        }
    }
}
```

모든 빈 출력하기

- `ac.getBeanDefinitionNames()` : 스프링에 등록된 모든 빈 이름을 조회한다.
- `ac.getBean()` : 빈 이름으로 빈 객체(인스턴스)를 조회한다.

애플리케이션 빈 출력하기

- 스프링이 내부에서 사용하는 빈은 getRole() 로 구분할 수 있다.
  - `ROLE_APPLICATION` : 일반적으로 사용자가 정의한 빈
  - `ROLE_INFRASTRUCTURE` : 스프링이 내부에서 사용하는 빈

스프링 빈 조회 - 기본

- 스프링 빈을 찾는 가장 기본적인 방법
  - `ac.getBean(빈 이름, 타입)`
  - `ac.getBean(타입)`
- `NoSuchBeanDefinitionException` 예외 발생

> 💡 실습 코드 작성

▼ **Test Code**

```
public class ApplicationContextBasicFindTest {
    AnnotationConfigApplicationContext ac = new Annotation
```

```
    @Test
    @DisplayName("빈 이름, 타입으로 조회")
    void findByName(){
        MemberService bean = ac.getBean("memberService", M
        Assertions.assertThat(bean).isInstanceOf(MemberSer
    }

    @Test
    @DisplayName("이름없이 타입으로 조회")
    void findByType(){
        MemberService bean = ac.getBean(MemberService.clas
        Assertions.assertThat(bean).isInstanceOf(MemberSer
    }

    @Test
    @DisplayName("이름없이 타입으로 조회")
    void findByName2(){
        MemberServiceImpl bean = ac.getBean("memberService
        Assertions.assertThat(bean).isInstanceOf(MemberSer
    }

    @Test
    @DisplayName("이름으로 빈 조회 실패 테스트")
    void failFindByName(){
        // MemberService memberService = ac.getBean("xxxxx
        Assertions.assertThrows(NoSuchBeanDefinitionExcept
    }
}
```

스프링 빈 조회 - 동일한 타입이 둘 이상

- `NoUniqueBeanDefinitionException` 예외 발생

> 💡 실습 코드 작성

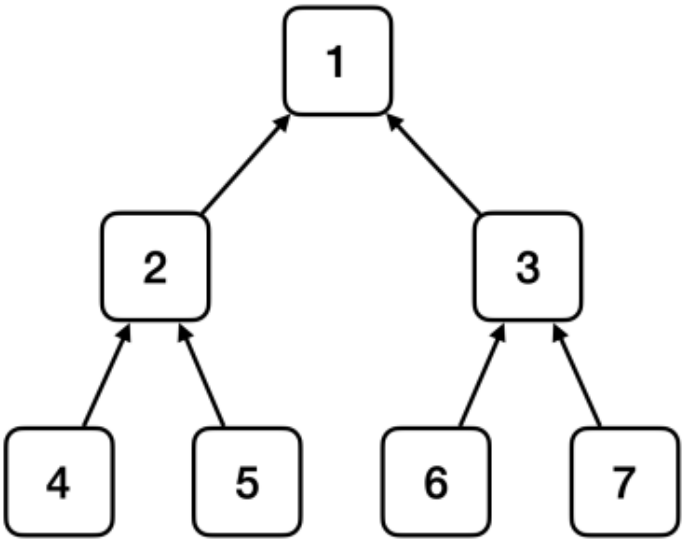
▼ **Test Code**

```
public class ApplicationContextSameBeanFindTest {
    AnnotationConfigApplicationContext ac = new Annotation

    @Test
    @DisplayName("타입으로 조회 시 같은 타입이 둘 이상 있으면, 중복
    void findBeanByTypeDuplicate(){
        // MemberRepository member = ac.getBean(MemberRepo
        Assertions.assertThrows(NoUniqueBeanDefinitionExce
    }

    @Test
    @DisplayName("타입으로 조회 시 같은 타입이 둘 이상 있으면, 빈 
    void findBeanByName(){
        MemberRepository member = ac.getBean("memberReposi
        org.assertj.core.api.Assertions.assertThat(member)
    }
    @Test
    @DisplayName("특정 타입 모든 빈 조회")
    void findAllBeanByType(){
        Map<String, MemberRepository> beansOfType = ac.getB
        for (String name : beansOfType.keySet()) {
            System.out.println("name = " + name + " value =
        }

        System.out.println("beansOfType = " + beansOfType)
        org.assertj.core.api.Assertions.assertThat(beansOfT
    }


    static class SameBeanConfig{
        @Bean
        public MemberRepository memberRepository1(){
            return new MemoryMemberRepository();
        }
        @Bean
        public MemberRepository memberRepository2(){
```

```
            return new MemoryMemberRepository();
        }
    }
}
```

스프링 빈 조회 - 상속 관계

1번: 1,2,3,4,5,6,7
2번: 2,4,5
3번: 3,6,7
4번: 4
5번: 5
6번: 6
7번: 7



- 부모 타입 조회하면, 자식 타입도 함께 조회
  - 모든 자바 객체의 최고 부모인 'Object' 타입 조회 시, 모든 스프링 빈 조회

💡 실습 코드 작성

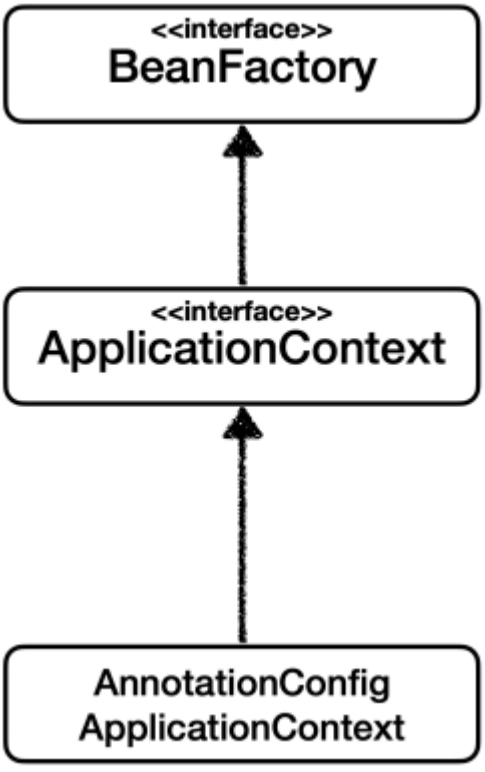
▼ **Test Code**

```
public class ApplicationContextExtendsFindTest {
    AnnotationConfigApplicationContext ac = new Annotation

    @Test
    @DisplayName("부모 타입으로 조회 시, 자식이 둘 이상 있으면, 중
    void findBeanByParentTypeDuplicate(){
        // DiscountPolicy discountPolicy = ac.getBean(Disc
        Assertions.assertThrows(NoUniqueBeanDefinitionExce
    }
```

```
    @Test
    @DisplayName("부모 타입으로 조회 시, 자식이 둘 이상 있으면, 빈
    void findBeanByParentTypeBeanName(){
        DiscountPolicy discountPolicy = ac.getBean("rateDi
        org.assertj.core.api.Assertions.assertThat(discoun
    }

    @Test
    @DisplayName("특정 하위 타입으로 조회")
    void findBeanBySubType(){
        RateDiscountPolicy rateDiscountPolicy = ac.getBean
        org.assertj.core.api.Assertions.assertThat(rateDis
    }

    @Test
    @DisplayName("부모 타입으로 모두 조회")
    void findAllBeanByParentType(){
        Map<String, DiscountPolicy> beansOfType = ac.getBe
        for (String key : beansOfType.keySet()) {
            System.out.println("name = " + key + " value "
        }
    }

    @Test
    @DisplayName("부모 타입으로 모두 조회 - Object")
    void findAllBeanByObject(){
        Map<String, Object> beansOfType = ac.getBeansOfTyp
        for (String key : beansOfType.keySet()) {
            System.out.println("name = " + key + " values =
        }
    }

    @Configuration
    static class TestConfig {
        @Bean
        public DiscountPolicy rateDiscountPolicy(){
            return new RateDiscountPolicy();
```

```
        }
        @Bean
        public DiscountPolicy fixDiscountPolicy(){
            return new FixDiscountPolicy();
        }
    }

}
```

BeanFactory 와 ApplicationContext



- `ApplicationContext` 은 `BeanFactory` 의 기능을 상속받는다.
- 부가기능이 포함된 `ApplicationContext` 사용한다.
- 둘 다 스프링 컨테이너라 한다.

다양한 설정 형식 지원 - 자바 코드, XML

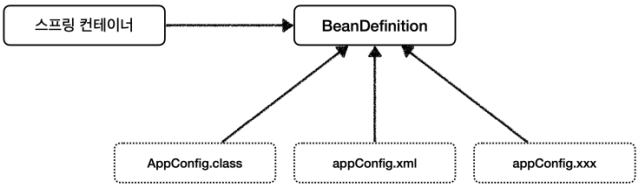
- 지금까지 했던 것 = `AppConfig`

- 최근에는 스프링 부트를 많이 사용하여 XML 기반의 설정을 잘 사용하지는 않는다.
- 레거시 프로젝트의 경우 XML 이 많아 한번쯤은 배우면 좋다.
- `GenericXmlApplicationContext` 를 사용하면서 `xml` 설정 파일을 넘기면 된다.
- AppConfig.java 와 xml 거의 유사

▼ **Xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
       xsi:schemaLocation="http://www.springframework.org/

    <bean id = "memberService" class="hello.core.member.Me
        <constructor-arg name = "memberRepository" ref="me
    </bean>

    <bean id = "memberRepository" class = "hello.core.memb

    <bean id = "orderService" class="hello.core.order.Orde
        <constructor-arg name = "memberRepository" ref = "
        <constructor-arg name = "discountPolicy" ref="disc
    </bean>

    <bean id = "discountPolicy" class = "hello.core.discou
</beans>
```

스프링 빈 설정 메타 정보 - BeanDefinition



- `BeanDefinition` 이라는 추상화
  - **역할과 구현을 개념적으로 나눈 것**
    - XML을 읽어서 `BeanDefinition` 을 만든다.

    - 자바 코드를 읽어서 `BeanDefinition` 을 만든다.
    - 스프링 컨테이너는 오직 `BeanDefinition` 만 알면 된다.
- `BeanDefinition` 을 빈 설정 메타정보라 한다.

> 💡 `BeanClassName`: 생성할 빈의 클래스 명(자바 설정 처럼 팩토리 역할의 빈을 사용하면 없음)
>
> `factoryBeanName`: 팩토리 역할의 빈을 사용할 경우 이름, 예) `appConfig`
>
> `factoryMethodName`: 빈을 생성할 팩토리 메서드 지정, 예) `memberService`
>
> `Scope`: 싱글톤(기본값)
>
> `lazyInit`: 스프링 컨테이너를 생성할 때 빈을 생성하는 것이 아니라, 실제 빈을 사용할 때 까지 최대한 생성을 지연 처리 하는지 여부
>
> `InitMethodName`: 빈을 생성하고, 의존관계를 적용한 뒤에 호출되는 초기화 메서드 명
>
> `DestroyMethodName`: 빈의 생명주기가 끝나서 제거하기 직전에 호출되는 메서드 명
>
> `Constructor arguments`, `Properties`: 의존관계 주입에서 사용한다. (자바 설정 처럼 팩토리 역할의 빈을 사용하면 없음)

팩토리 메서드 등록

- 자바 코드를 이용해 빈 설정 하는 방식은 **팩토리 메서드** 방식이다.
    - 외부에서 메서드를 호출 해 생성이 되는 방식

- AppConfig 에서 MemberService 생성자 호출 → MemberService 객체 생성