# 1. 오브젝트와 의존관계

2021.12.10

# Contents

# Spring이란

- Spring
- 클래스 호출 방식(객체지향)

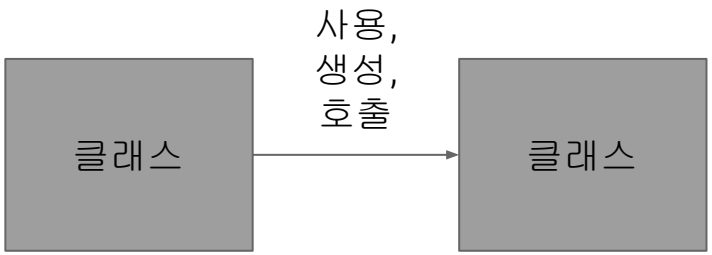
# Spring

어떻게 **오브젝트**가 설계되고, 만들어지고, 어떻게 관계를 맺고 사용되는지에 **관심**을 갖는 프레임워크

- 프레임워크 vs 라이브러리

  흐름에 대한 제어 권한이 어디에 있느냐의 차이
  ⇒ 제어의 역전(IoC)

# 클래스 호출방식

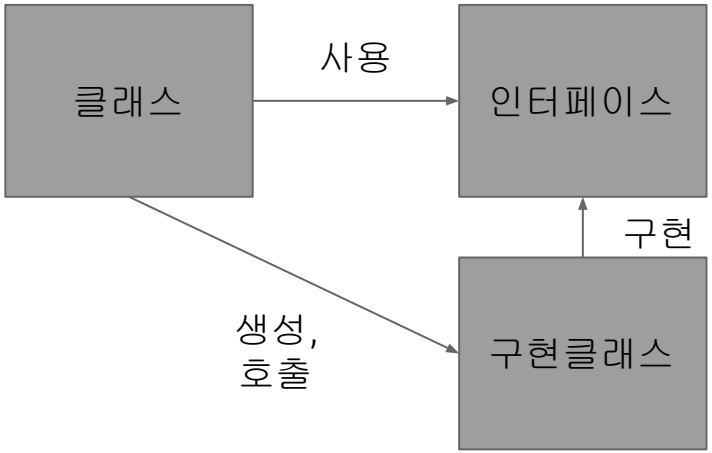



관심사의 분리,
리팩토링

- **리팩토링**:
  기존의 코드를 외부의 동작방식에는
  변화없이 내부 구조를 변경해서
  재구성하는 작업 또는 기술

- **메소드 추출 기법**:
  공통의 기능을 담당하는 메소드로
  중복된 코드를 뽑아 내는 것

# 클래스 호출방식




**인터페이스**를 이용한 클래스 호출

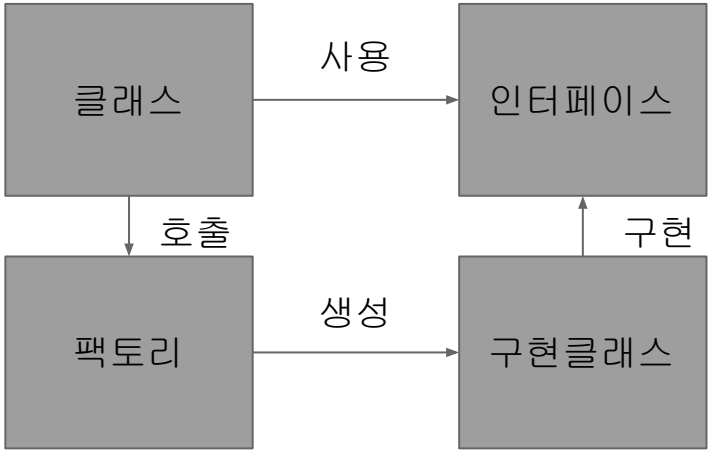
- **개방 폐쇄 원칙**

  클래스나 모듈은 확장에 열려 있어야하고 변경에는 닫혀있어야함

- **전략패턴**

  변경이 필요한 알고리즘을 인터페이스를 통해 통째로 외부로 분리, 이를 구현한 구체적인 알고리즘 클래스를 필요에 따라 바꿔서 사용할 수 있게하는 디자인 패턴

# 클래스 호출방식




**팩토리**를 이용한 클래스 호출

- **팩토리**

  객체의 생성 방법을 결정하고 그렇게 만들어진 오브젝트를 돌려주는 오브젝트

⇒ 오브젝트를 생성하는 쪽과 생성된 오브젝트를 사용하는 쪽의 역할과 책임을 분리하려는 목적으로 사용

# Spring IoC/DI

- IoC 용어 정리
- Spring IoC/DI
- 싱글톤 레지스토리

# 스프링 IoC 용어 정리

- 컨테이너

인스턴스의 생명주기를 관리하며, 생성된 인스턴스에게 추가적인 기능을 제공함

- 빈(Bean)

빈 또는 빈 오브젝트는 스프링이 IoC 방식으로 관리하는 오브젝트라는 뜻이다.

- 빈 팩토리(Bean Factory)

스프링의 IoC를 담당하는 핵심 컨테이너를 가리킨다. 빈을 등록, 생성, 조회, 관리하는 기능을 담당한다.

# IoC 용어 정리

- ApplicationContext

Bean Factory + 스프링이 제공하는 각종 부가 서비스를 추가로 제공
빈의 생성과 제어의 관점 => 빈 팩토리
스프링이 제공하는 애플리케이션 지원 기능을 모두 포함 => ApplicationContext

- 설정정보/설정 메타정보

스프링의 설정정보란 애플리케이션 컨텍스트 또는 빈 팩토리가 IoC를 적용하기 위해 사용하는 메타정보

# Spring IoC/DI

IoC(Inversion of Control - 제어의 역전)

개발자가 만든 어떤 클래스나 메소드를 다른 프로그램이 대신 실행해주는 것

- 작업을 수행하는 쪽에서 Object를 생성하는 제어 흐름의 개념을 역전
- 자신이 사용할 오브젝트를 스스로 선택/생성하지 않는다.
- 오브젝트는 자신이 어떻게 생성되고 어떻게 사용되는지 알 수 없다.
- 모든 오브젝트는 제어 권한을 위임받는 특별한 오브젝트에 의해서 만들어 지고 사용된다.
  ⇒ 빈 팩토리(?)

# 싱글톤 레지스토리

- 스프링이 싱글톤으로 빈을 만드는 이유:
  스프링이 주로 적용되는 대상이 자바 엔터프라이즈 기술을 사용하는 서버환경이기 때문 => 요청이 엄청나게 많은 트래픽 사이트에서는 계속 객체를 생성하게 되면 메모리 낭비가 심하기 때문

- 싱글톤 패턴의 여러 단점 -> 싱글톤 레지스트리
  private 생성자를 갖고 있기 때문에 상속 불가 / 싱글톤은 테스트하기가 어렵다 등등

- 싱글톤 레지스트리:
  스프링은 직접 싱글톤 형태의 오브젝트를 만들고 관리하는 기능을 제공
  즉, 해당 객체는 자유롭게 사용할 수 있지만 해당 객체를 스프링 컨테이너에서 가져오면 싱글톤으로 가져오는 것이다.

# Spring IoC/DI

IoC 구현 방법

- DI(Dependency Injection) - 의존성 주입

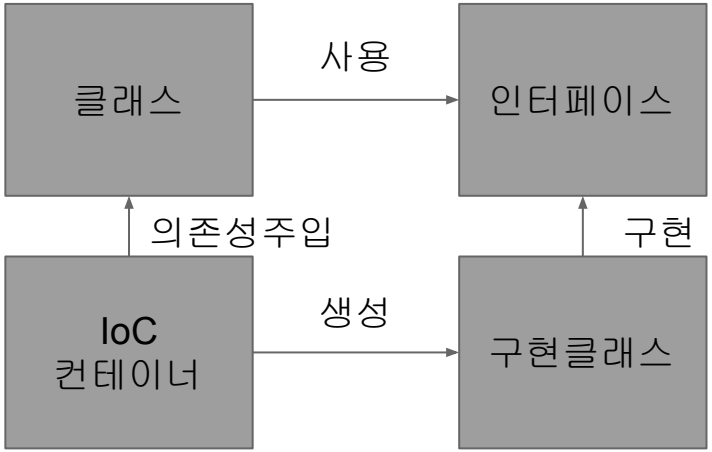
  클래스 사이의 의존 관계를 빈 설정 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것

- DL(Dependency Lookup) - 의존성 검색

  저장소에 저장되어 있는 빈에 접근하기 위하여 개발자들이 컨테이너에서 제공하는 API를 이용하여 사용하고자 하는 빈을 검색하는 것

- DL 사용 시 컨테이너 종속성이 증가하여, 이를 줄이기 위해 DI를 사용

# 클래스 호출방식




**IoC**를 이용한 클래스 호출

- **애플리케이션 컨텍스트 = IoC 컨테이너 = 스프링 컨테이너**

  실행시점에 클래스간의 관계가 형성된다.

  장점:
  - 클라이언트는 구체적인 팩토리 클래스를 알 필요가 없다.
  - 종합 IoC 서비스를 애플리케이션 컨텍스트가 제공
  - 빈을 검색하는 다양한 방법을 제공

# 헷갈리는 내용 정리

- 추상 클래스 vs 인터페이스
- 템플릿 메소드 vs 팩토리 메소드
- 자바 싱글톤 vs 스프링 싱글톤 패턴

# 헷갈리는 내용 정리

- 추상클래스 vs 인터페이스

- 추상클래스

추상메서드를 선언하여 상속을 통해 자식 클래스에서 완성하도록 유도하는 클래스 / 객체 생성 불가

- 인터페이스

객체를 어떻게 구성해야 하는지 정리한 설계도 / 다중 상속이 가능

# 헷갈리는 내용 정리

- 차이점

- 추상 클래스는 상속할 각 객체들의 공통점을 찾아 추상화시킨 것으로 상속 관계를 타고 올라갔을 때 **같은 부모 클래스를 상속하며 부모 클래스가 가진 기능**들을 구현 해야할 경우 사용

- 인터페이스는 상속 관계를 타고 올라갔을 때 **다른 조상 클래스를 상속**하더라도, **같은 기능**이 필요할 경우 사용한다.
  클래스와 별도로 구현 객체가 같은 동작을 한다는 것을 보장하기 위해 사용한다.

# 헷갈리는 내용 정리

- 템플릿 메소드 패턴 vs 팩토리 메소드 패턴

- 템플릿 메소드 패턴
  슈퍼 클래스에 기본적인 메소드가 정의되어 있고 세부적인 메소드는 추상메소드로 두고 서브 클래스에서 구현해 사용하는 방법

- 팩토리 메소드 패턴
  서브클래스에서 구체적인 오브젝트 생성 방법을 결정하게 하는 것

- 참고한 사이트

- https://www.boostcourse.org/web326/lecture/58970/?isDesc=false
- https://isstory83.tistory.com/91
- https://myjamong.tistory.com/150
- https://velog.io/@new_wisdom/Java-%EC%B6%94%EC%83%81-%ED%81%B4%EB%9E%98%EC%8A%A4%EC%99%80-%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4%EC%9D%98-%EC%B0%A8%EC%9D%B4
- https://hojak99.tistory.com/347