Spring Framework

스프링은 경량 프레임워크이다.

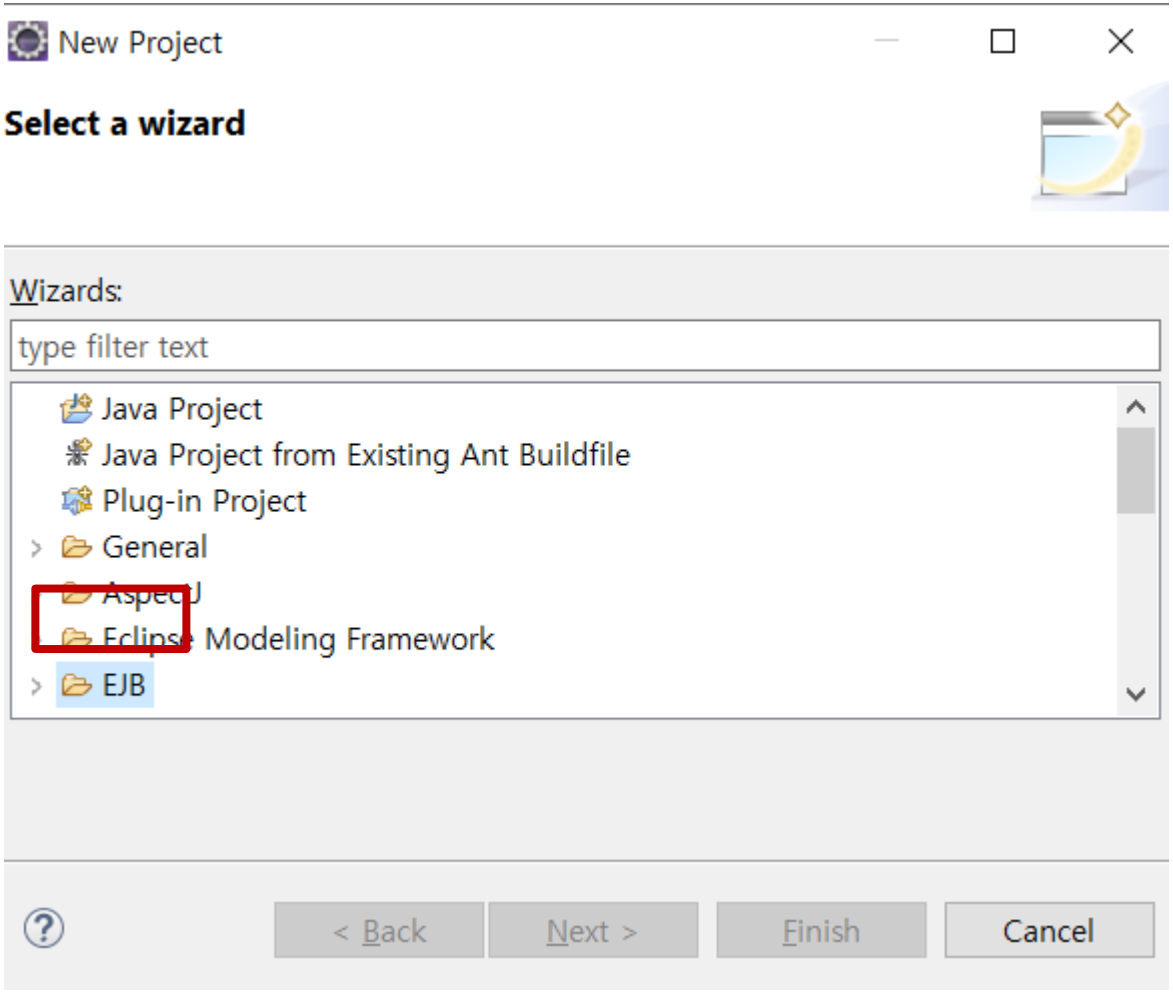
==

스프링은 필요이상으로 무겁지 않다.

스프링 이전의 프레임워크

EJB(Enterprise JavaBeans)프레임워크 :

EJB의 비전(EJB 1.0 Specification)
EJB는 애플리케이션 개발을 쉽게 만들어준다.
애플리케이션 개발자는 로우레벨의 기술들에 관심을 가질 필요도 없다.



스프링 이전의 프레임워크

EJB(Enterprise JavaBeans)

EJB에서는 현실에서 1% 미만의 애플리케이션에서만 필요한 멀티 DB 분산 트랜잭션을 위해
나머지 99%의 애플리케이션에서도 무거운 JTA기반의 글로벌 트랜잭션 관리 기능을 사용해야 했다.

EJB의 혜택을 얻기 위해 모든 기능이 다 필요하지도 않은 고가의 WAS를 구입 해야했고,
고급 IDE의 도움 없이는 손쉽게 다룰 수 없는 복잡한 설정 파일 속에서 허우적대야 했다.

EJB 컴포넌트는 컨테이너 밖에서는 정상적으로 동작할 수 없으므로
개발자들은 끝도 없이 반복되는 **수정-빌드-배포-테스트**의 지루한 과정으로 많은 시간을 낭비 해야했다.

가장 최악의 문제점은 EJB 스펙을 따르는 비즈니스 오브젝트들은 객체지향적인 특징과 장점을 포기해야 했다는 것이다.

EJB 빈은 상속과 다형성 등의 혜택을 제대로 누릴 수 없었다.

토비의 스프링 저자 이일민

가장 최악의 문제점은 EJB 스펙을 따르는 비즈니스 오브젝트들은
객체지향적인 특징과 장점을 포기해야 했다는 것이다.

객체지향적인 특징과 장점?
그게 뭐죠?

객체지향 프로그래밍

앨런 튜링

2차 세계대전 때 독일군의 암호를 해독하기 위해
1936년에 세계 최초의 컴퓨터를 만듦

1940년대 : 탄도 미사일 거리 계산, 폭발 계산 등 군사목적으로 컴퓨터 사용

1950년대 : 전 과학 분야에서 컴퓨터를 사용하기 시작

1970년대 : 개인에게 컴퓨터가 보급되기 시작

1980년대 : 컴퓨터가 폭발적으로 보급, 그래픽 기반 소프트웨어 등장

1990년대 : WEB의 등장, IT버블

2000년대 : 구글 등장, wiki 등장, 페이스북 등장, 빅데이터 등장

2010년대 : 빅데이터, 인공지능, 블록체인

2020년대 : ???

객체지향 프로그래밍

컴퓨터가 사용되는 분야가 계속 해서 넓어진다.
다양한 분야의 소프트웨어 개발이 가능한 유연한 언어의 필요성이 대두

HOW?

객체지향 프로그래밍

세상 모든 것을 나타내고 표현하는 언어의 구조를 본 따 프로그래밍언어를 만들기 위한 연구를 시작

객체지향이론의 기본 개념은 '실제 세계는 사물(객체)로 이루어져 있으며, 발생하는 모든 사건들은 사물간의 상호작용이다.' 라는 것이다.

자바의 정석 / p.230

사태는 대상(객체)들의 결합이다.
사태는 존립하거나 비존립 한다.
존립하는 사태들의 결합이 세계이다.

논리철학논고

객체지향 프로그래밍

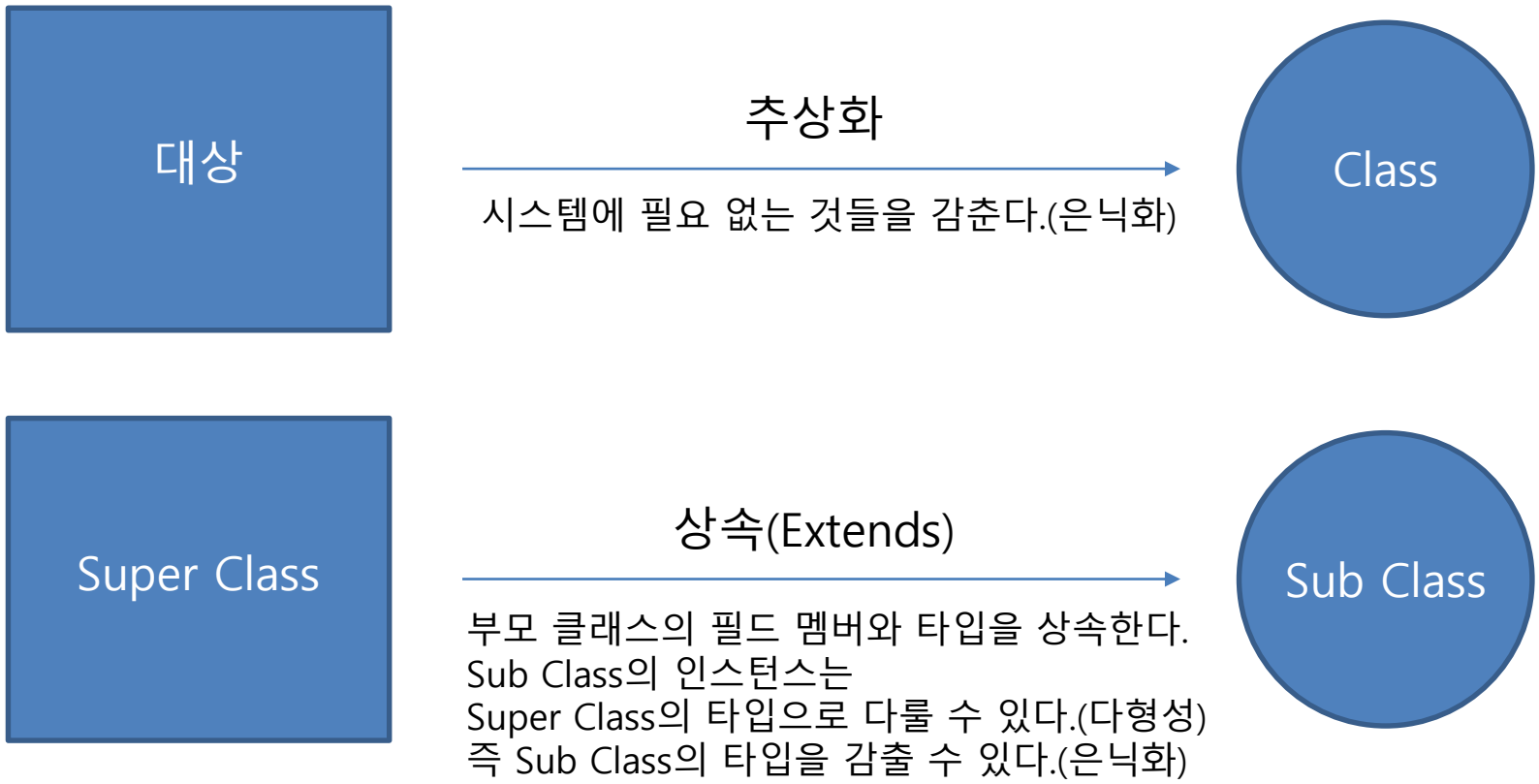
언어는 추상적이다.

추상적이라는 것은 구체적이지 않다는 뜻이다.

언어는 전달하고자 하는 맥락(message)과 무관한 부분은 과감히 지워버린다.

그래서 언어는 유연하다.

객체지향 프로그래밍


대상
추상화
시스템에 필요 없는 것들을 감춘다.(은닉화)
Class
Super Class
상속(Extends)
부모 클래스의 필드 멤버와 타입을 상속한다.
Sub Class의 인스턴스는
Super Class의 타입으로 다룰 수 있다.(다형성)
즉 Sub Class의 타입을 감출 수 있다.(은닉화)
Sub Class

객체지향 프로그래밍

구체적인 것은 감추자.

구체적인 타입은 감추자.(다형성)

구체적인 작동방식은 감추자.(facade pattern**)**

반드시 다른 객체가 알아야 할 것만 드러내고

나머지는 모두 다 감추자 (캡슐화)

감추면 유연해진다.

Refactoring

1. 20분간 제공 받은 코드를 읽고 어떤 기능을 하는 프로그램인지 분석하세요.
2. 30분간 코드를 어떻게 Refactoring 할 지 팀 별로 토론하세요.
3. 다음 시간에 팀 별로 발표합니다. 간단한 발표자료를 준비해주세요

커피 재고 프로그램 Refactoring

Class

Sales

판매 등록

환불 등록

Coffee

판매 등록

누적 판매량 등록

재고 등록

재고 차감

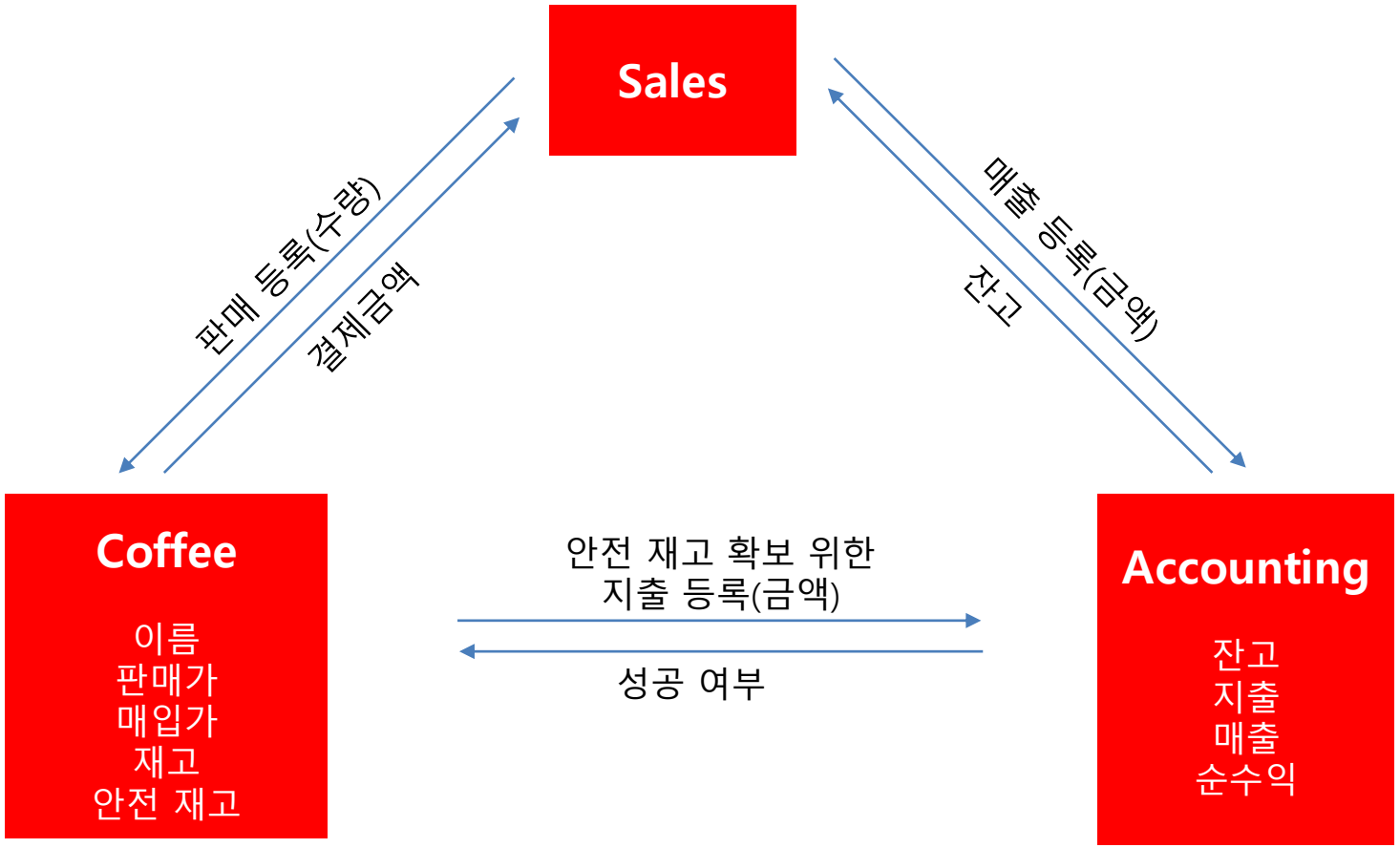
안전 재고 확보

Accounting

매출 등록

지출 등록

커피 재고 프로그램 Refactoring



커피 재고 프로그램 Refactoring

Before

1. 커피 종류가 1개 늘어날 때마다 40줄 가량의 코드가 추가된다.
2. 기능을 파악하기 위해 모든 코드를 처음부터 끝까지 읽어야 한다.
3. 기능을 확장하기 위해 전체 로직을 고려해야 한다.

After

1. 추상화를 통해 Coffee클래스를 만들었기 때문에 커피 종류를 추가하기 위해 코드를 수정할 필요가 없다.
2. 반복되는 코드를 메서드로 만들어 재사용 하고 있어서 수정을 편하게 할 수 있다.
3. 구체적인 내용이 은닉화 되어 있어 전체 기능을 쉽게 파악할 수 있다.
4. 관심사에 따라서 코드가 분리 되어 있어 기능을 쉽게 확장할 수 있다.

커피 재고 프로그램 Refactoring

기능 확장 해보기

1. 환불 기능 추가하기
    1. 재고 + 환불 상품 수량
    2. 잔고 – 환불 상품 수량 * 환불 상품 판매가
    3. 판매개수 감소

2. 반품 기능 추가하기
    1. 재고 – 반품 상품 수량
    2. 잔고 + 반품 상품 수량 * 매입가

커피 재고 프로그램 Refactoring

실습하기 : Premium Coffee 추가하기

1. 사용자의 요청으로 프리미엄 등급의 커피도 관리할 수 있게끔 프로그램을 확장하려 한다.

2. 프리미엄 등급의 커피는 기존의 커피에 고급스러운 포장을 더한 제품이다.
   프리미엄 등급의 커피를 만들기 위해 포장 비용이 2000원 발생하게 된다.

3. 프리미엄 등급의 커피를 반품 처리 할 경우, 거래처로부터 매입가만 환불 받게 된다.
   즉 프리미엄 등급의 커피에 반품 처리 할 때 마다 2000원씩 손해가 발생한다.

4. 위 조건을 고려하여 프로그램을 확장해보세요.

객체지향 SOLID원칙

SRP(Single Responsibility Principle), 단일 책임 원칙

-> 하나의 클래스는 하나의 책임(역할)을 가져야 한다.

OCP (Open-Closed Principle), 개방-폐쇄 원칙

-> 기존의 코드를 수정하지 않고 기능을 확장할 수 있어야 한다.
(확장에는 개방적이고 수정에는 폐쇄적이다.)

LSP (Liskov Substitution Principle), 리스코프 치환 원칙

-> 자식클래스는 언제나 부모클래스를 대체할 수 있어야 한다. Is a 관계

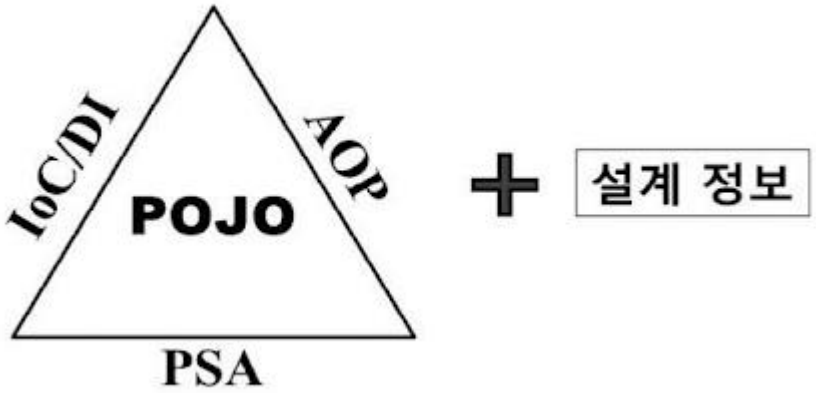
ISP (Interface Segregation Principle), 인터페이스 분리 원칙

-> 하나의 거대한 인터페이스를 만들기 보다는 기능별로 하나씩의 인터페이스를 만들어야 한다.

DIP (Dependency Inversion Principle), 의존 역전 원칙

-> 자신보다 추상성이 높은 클래스에 의존 해야한다.

Spring Framework



IOC/DI : 의존성 주입을 통한 제어 반전

AOP : 관점 지향 프로그래밍

PSA : 이식 가능한 서비스 추상화

Spring Framework

POJO : Plain Old Java Object

순수한 자바 객체로 돌아가자!

객체지향적인 원리에 충실하면서, 환경과 기술에 종속되지 않고
필요에 따라 재활용될 수 있는 방식으로 설계된 오브젝트