Spring Framework

1. Framework 개념

Tacademy

CONTENTS

학습 목표

- SW 재사용성을 높일 수 있는 방안에 대해 이해할 수 있습니다
- 디자인패턴과 프레임워크의 관련성에 대해 이해할 수 있습니다
- 프레임워크 구성요소와 종류에 대해 이해할 수 있습니다.

1. SW 재사용 방안들

복사(Copy) & 붙이기(Paste)

초보적인 재사용 방식으로 비슷한 예제를 다른 Source에서 복사해서 사용함.

```
GregorianCalendar date = (GregorianCalendar)Calendar.getInstance();
SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
String date = df.format(date);
```

- 예를 들어, A라는 클래스에서 Date 타입을 String 타입으로 변환하는 코딩을 하고, 클래스 B에서 동일한 로직이 필요하여 복사했다고 가정한 경우

JDK 버전이 바뀌어 동일한 기능을 제공하는 향상된 인터페이스가 나오면 위의 코드를 사용한 A, B 클래스를 모두 변경해야 한다.

메서드 호출

자주 사용되고, 유사한 기능들을 모아 메서드로 정의하여 재사용함.

```
public class DateUtility {
   public static String toStringToday(String format) {
          GregorianCalendar date = (GregorianCalendar)Calendar.getInstance();
          SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
          String date = df.format(date);
   }
}
```

```
Sting sdate = DateUtility.toStringToday("yyyMMdd");
```

메서드 호출

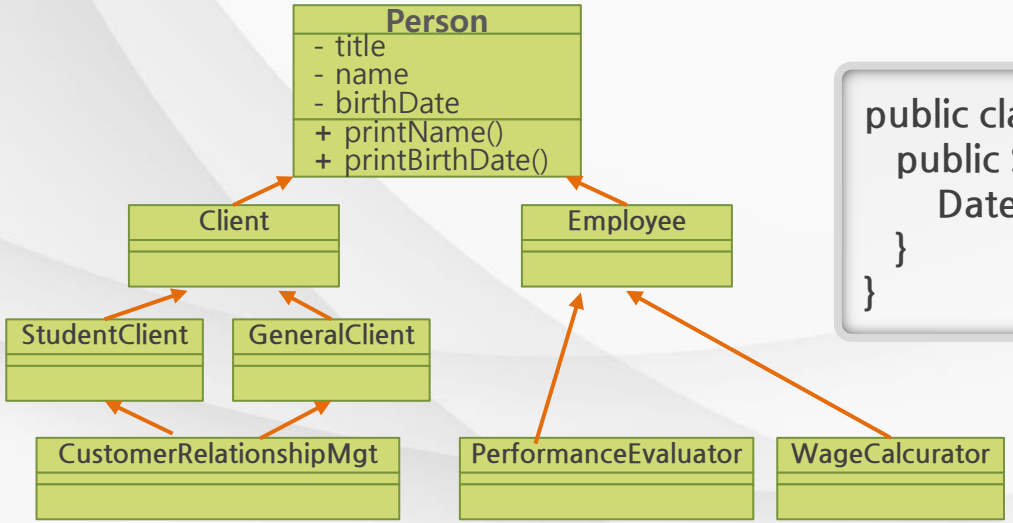
- JDK 버전이 바뀌거나 메서드의 내용이 수정되더라도 해당 클래스를 모두 수정할 필요 없이 toStringToday() 메서드의 내용만 수정하면 된다.

toStringToday() 메서드의 Signature를 변경하면 이 메서드를 사용하는 모든 클래스에 영향을 준다.

메서드 재사용 방법은 '복사 & 붙이기'보다는 진보된 방식이지만, 작업 영역간의 결합도(Coupling) 문제는 여전히 존재한다.

클래스 재사용 (상속)

자주 사용되고, 유사한 기능들을 모아 메서드로 정의하여 재사용함.

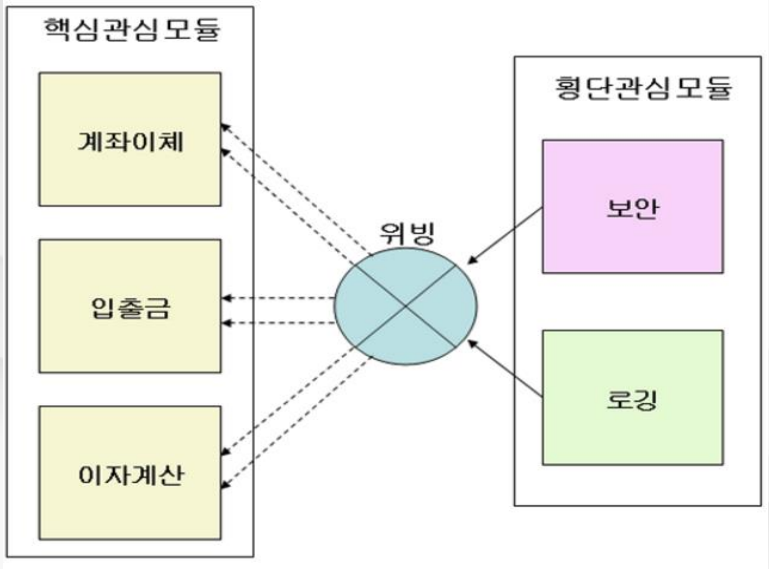



```
public class Person {
   public String printBirthDate(String format) {
      DateUtility.toStringToday(birthDate, format);
   }
}
```

- Person을 상속받은 모든 클래스들은 자동적으로 변경된 printBirthDate() 메서드를 사용하게 된다.
- DateUtility 클래스의 메서드가 변경되더라도 printBirthDate() 메서드의 인터페이스가 변하지 않으면 나머지 클래스들은 영향을 받지 않는다.

AOP(Aspect Oriented Programming)

❖관심의 분리 (Seperatrion of Concerns)



AOP가 핵심관심모듈의 코드를 직접 건드리지 않고 필요한 기능이 동작하도록 하는 데는 위빙(Weaving)이라고 하는 특수한 작업이 필요하다.

즉, AOP에서 위빙 작업을 통해 핵심모듈 사이 사이에 필요한 횡단 관심 코드가 동작하도록 엮어지게 만든다.

- AOP는 OOP를 더욱 OOP 답게 만들어 줄 수 있다.
- AOP는 OOP 뿐만 아니라 기존의 절차적 프로그래밍에도 적용될 수 있다.

2. 디자인 패턴과 프레임워크의 관련성

디자인패턴의 정의

프로그램 개발에서 자주 나타나는 과제를 해결하기 위한 방법 중 하나로, 소프트웨어 개발과정에서 발견된 Know-How를 축적하여 이름을 붙여 이후에 **재사용하기 좋은 형태로 특정 규약을 묶어서 정리**한 것.

- 이 용어를 소프트웨어 개발 영역에서 구체적으로 처음 제시한 곳은, GoF(Gang of Four)라 불리는 네 명의 컴퓨터 과학 연구자들이 쓴 서적 'Design Patterns: Elements of Reusable Object-Oriented Software'(재사용 가능한 객체지향 소프트웨어의 요소 - 디자인 패턴)이다.

디자인패턴의 정의

❖ 디자인 패턴을 사용하는 이유

- 요구사항은 수시로 변경-> 요구사항 변경에 대한 Source Code 변경을 최소화
- 여러 사람이 같이 하는 팀 프로젝트 진행 -> 범용적인 코딩 스타일을 적용
- 상황에 따라 인수 인계하는 경우도 발생 -> 직관적인 코드를 사용

프레임워크의 정의

비기능적(Non-Funtional) 요구사항(성능, 보안, 확장성, 안정성 등)을 만족하는 구조와 구현된 기능을 안정적으로 실행하도록 제어해주는 잘 만들어진 구조의 라이브러리의 덩어리

- 프레임워크는 애플리케이션들의 최소한의 공통점을 찾아 하부 구조를 제공함으로써 개발자들로 하여금 시스템의 하부 구조를 구현하는데 들어가는 노력을 절감하게 해줌

프레임워크의 정의

❖ 프레임워크를 사용하는 이유

- 비기능적인 요소들을 초기 개발 단계마다 구현해야 하는 불합리함을 극복해준다.
- 기능적인(Functional) 요구사항에 집중할 수 있도록 해준다.
- 디자인 패턴과 마찬가지로 반복적으로 발견되는 문제를 해결하기 위한 특화된 Solution을 제공한다.

디자인패턴과 프레임워크의 관련성

디자인 패턴은 프레임워크의 핵심적인 특징이고, 프레임워크를 사용하는 애플리케이션에 그 패턴이 적용된다는 특징을 가지고 있다. 하지만 프레임워크는 디자인 패턴이 아니다.

- 디자인 패턴은 애플리케이션을 설계할 때 필요한 구조적인 가이드라인이 되어 줄 수는 있지만 구체적으로 구현된 기반코드를 제공하지 않는다.

- 프레임워크는 디자인 패턴과 함께 패턴이 적용 된 기반 클래스 라이브러리를 제공해서 프레임워크를 사용하는 구조적인 틀과 구현코드를 함께 제공한다.

디자인패턴과 프레임워크의 관련성

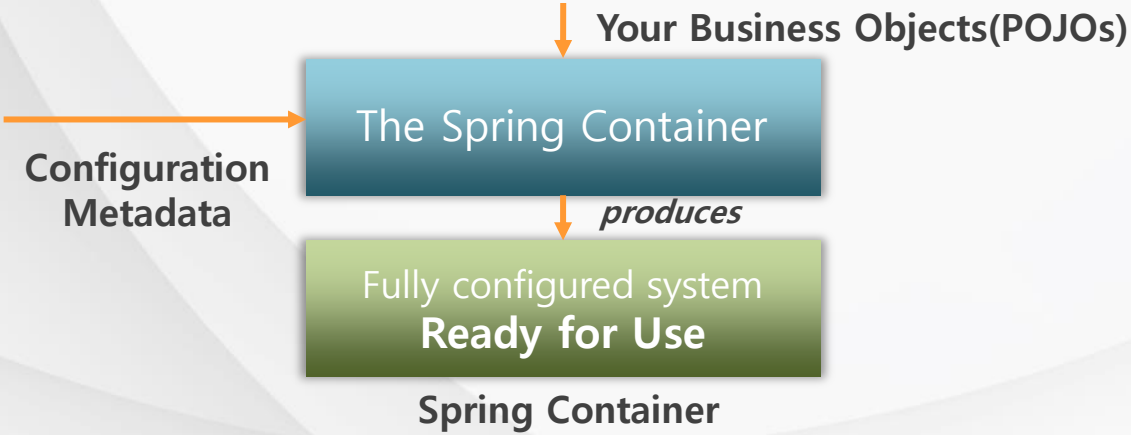
개발자는 프레임워크의 기반코드를 확장하여 사용하면서 자연스럽게 그 프레임워크에서 사용된 패턴을 적용할 수 있게 된다.

3. 프레임워크의 구성요소와 종류

IoC (Inversion of Control)

IoC란 **"제어의 역전"** 즉, 인스턴스 생성부터 소멸까지의 **인스턴스 생명주기 관리를 개발자가 아닌 컨테이너가 대신 해준다**는 뜻임.
즉, 컨테이너 역할을 해주는 프레임워크에게 제어하는 권한을 넘겨서 개발자의 코드가 신경 써야 할 것을 줄이는 전략이다.

IoC (Inversion of Control)



- 프레임워크의 동작원리를 제어흐름이 일반적인 프로그램 흐름과 반대로 동작하므로 IoC 라고 설명함.
- Spring 컨테이너는 IoC를 지원하며, 메타데이터(XML설정)를 통해 beans를 관리하고 어플리케이션의 중요부분을 형성함.
- Spring 컨테이너는 관리되는 bean들을 의존성주입(Dependency Injection)을 통해 IoC를 지원함.

클래스 라이브러리 (Class Library)

프레임워크는 특정 부분의 기술적인 구현을 라이브러리 형태로 제공한다.

Class Library라는 구성요소는 프레임워크의 정의 중 하나인 "Semi Complete(반제품)" 이다. 라고 해석하게 만들었다.

특징	프레임워크	라이브러리
유저코드의 작성	프레임워크 클래스를 서브 클래싱 해서 작성	독립적으로 작성
호출흐름	프레임워크코드가 유저코드를 호출	유저코드가 라이브러리를 호출
실행흐름	프레임워크가 제어	유저코드가 제어
객체의 연동	구조프레임워크가 정의	독자적으로 정의

클래스 라이브러리 (Class Library)

❖라이브러리와 프레임워크의 차이점

- 프레임워크와 라이브러리를 구분하는 방법은 실행제어가 어디서 일어나는 가에 달려있다.

- **라이브러리**는 개발자가 만든 클래스에서 직접 호출하여 사용하므로 실행의 흐름에 대한 **제어를 개발자의 코드가 관장**하고 있다.

- **프레임워크**는 반대로 **프레임워크에서** 개발자가 만든 클래스를 호출하여 **실행의 흐름에 대한 제어를 담당**한다.

디자인 패턴

디자인 패턴 + 라이브러리 = 프레임워크

프레임워크는 디자인 패턴과 그것이 적용된 기반 라이브러리의 결합으로 이해할 수 있다.

프레임워크의 라이브러리를 살펴볼 때도 적용된 패턴을 주목해서 살펴 본다면 그 구성을 이해하기 쉽다.

특히 프레임워크를 확장하거나 커스터마이징 할 때는 프레임워크에 적용된 패턴에 대한 이해가 꼭 필요하다.

프레임워크 종류

❖ 아키텍쳐 결정 = 사용하는 프레임워크의 종류 + 사용전략

기능	프레임워크 종류
웹(MVC)	Spring MVC, Struts2, Webwork, PlayFramework
OR(Object-Relational) 매핑	MyBatis, Hibernate, JPA, Spring JDBC
AOP(Aspect Oriented Programming)	Spring AOP, AspectJ, JBoss AOP
DI(Dependency Injection)	Spring DI, Google Guice
Build와 Library관리	Ant + Ivy, Maven, Gradle
단위 테스트	jUnit, TestNG, Cactus
JavaScript	jQuery, AngularJS, Node.js

학습정리

지금까지 [Framework 개념]에 대해서 살펴보았습니다.

SW 재사용 방안들

Copy & Paste, Method, Inheritance, AOP

디자인 패턴과 프레임워크 관련성

개발자는 프레임워크의 기반코드를 확장하여 사용하면서 자연스럽게
프레임워크에서 사용된 패턴들을 적용할 수 있게 된다.

프레임워크의 구성요소와 종류

IoC(Inversion of Control), Design Pattern, Class Library