SPRING

Spring

- JAVA의 웹 프레임워크로 JAVA 언어를 기반으로 사용한다. JAVA로 다양한 어플리케이션을 만들기 위한 프로그래밍 틀이라 할 수 있다.
- 옛날에 비교하면 지금은 JAVA의 활용도가 높아졌고 따라서 프로젝트 규모도 커졌다. JAVA를 이용한 기술은 JSP, MyBatis, JPA 등 여러가지가 있는데 즉, 이 기술들이 프로젝트에 많이 쓰인다고 할 수 있다. Spring은 이 기술들을 더 편하게 사용하기 위해 만들어진 것이다.
- 프로젝트를 진행하다 보면 아무리 분업을 해도 분명 중복되는 코드가 있기 마련이다. Spring은 이런 **중복코드의 사용률을 줄여주고, 비즈니스 로직을 더 간단**하게 해줄 수 있다.
- Spring을 사용하면 다른 사람의 코드를 참조하여 쓰기 편리한데 이말의 의미는 **오픈소스를 좀더 효율적으로 가져다 쓰기 좋은 구조**라는 것이다.
- 결론적으로 Spring이란 JAVA 기술들을 더 쉽게 사용할 수 있게 해주는 오픈소스 프레임 워크이다.

Framework

- 프레임 워크는 어떠한 목적을 달성하기 위해 복잡하게 얽혀 있는 문제를 해결하기 위한 구조이자 약속이며, 소프트웨어 개발에 있어서 하나의 뼈대 역할을 한다.
- 더 간단히 말하자면 프레임 워크는 자주 쓰일 만한 기능들을 한데 모아 놓은 유틸(클래스)들의 모음(집합)이라고 정의할 수 있다.
- 설계의 기반이 되는 부분을 기술한 확장 가능한 **기반 코드**와 사용자가 이 코드를 자기 입맛대로 수정, 확장하는데 필요한 **라이브러리 이 두가지 요소가 통합되어 제공**되는 형태이다.
- 사용자가 이 기반 코드를 이용해 일정 수준 이상의 품질을 보장받는 코드를, 비교적 빠른 시간에 완성 및 유지보수할 수 있는 환경을 제공해주는 솔루션이다.
- 즉 프레임 워크는 기본적인 설계나 필요한 라이브러리는 알아서 제공해줄꺼니깐 개발자는 만들고 싶은 기능을 구현하는데 집중해라는 취지에서 만들어진 것이다.
- 개발자가 구현하고자 하는 기능을 쉽게 제공해줄수 있다는 점에서 프레임 워크는 라이브러리와 비슷한 면이 있다.

Framework VS Library

- 라이브러리는 개발자가 프로그램을 짜다가 라이브러리가 필요한 순간에 인지하고 라이브러리를 직접 추가해야겠다는 생각이 들었을때 가져다 쓰는 것이다. 라이브러리는 프로그램 기능 구현에만 도움을 줄 수 있다.
- 하지만 프레임 워크는 필요한 라이브러리와 기능 구현에 필요한 설계도 틀을 함께 제공해준다.
- 설계도 틀이란 확장 가능한 기반코드, 재사용 가능한 형태의 협업화된 클래스와 같은 뜻이다. 사용자가 세세하게 신경쓰지 않아도 수비고 빠르게 기능을 확장하거나 유지보수할 수 있게 해주는 구조에 대한 가이드라인이라 할 수 있다. 즉, 기반이 되는 부모 클래스라 생각하면된다.
- 개발자가 동일한 라이브러리를 쓰는 동일한 기능의 프로그램을 사용할때, 프레임 워크를 사용하면 클래스 관계 구조나, 데이터를 처리하는 절차, 프로그램이 화면에 그려지는 방식 등 일부 틀을 함께 제공 받을 수 있다. 즉, 프레임 워크는 **라이브러리+설계도**가 함께 온다.

Spring 주요 특징

- Spring은 개발을 더 쉽게 해주는 프로그램 틀이다. 개발을 더 쉽게 해주는 기술들이 Spring에 존재하는데, IoC, Di, AOP 등등이 있다.
- 개발을 더 쉽게 해주는 것들 중 몇가지를 알아보겠다.

1. IoC(Inversion of Control, 제어 반전)

- 개발자는 JAVA 코딩시 new 연산자, 인터페이스 호출, 데이터 클래스 호출 방식으로 객체를 생성하고 소멸시킨다.
- IoC란 인스턴스 (객체)의 생성부터 소멸까지 객체 생명주기 관리를 개발자가 하는게 아닌 스피링(컨테이너)가 대신 해주는 것을 말한다.

- 다들 겪어봤겠지만 자신을 믿어선 안된다ㅋㅋ.. 분명 틀린 코드가 없다고 생각했는데 알고보면 항상 오타라던가 이상한 곳에 코드가 있더라.. IoC는 개발자가 실수할 수 있는 생명주기 관리를 대신 해준다.
- 프로젝트의 규모가 커질수록 객체와 자원을 이용하는 방법이 더 복잡해지고 어디서 코드가 꼬일지 모르는 것을 Spring의 IoC는 자동으로 관리해준다.
- 즉, **제어권이 개발자가 아닌 IoC에게 있으며** IoC가 개발자의 코드를 호출하여 그 코드로 생명주기를 제어하는 것이다.

2. DI(Dependency Injection, 의존성 주입)
- 프로그래밍에서 구성요소 간의 의존 관계가 소스코드 내부가 아닌 외부의 설정파일을 통해 정의되는 방식이다.
- 코드 재사용을 높여 재사용을 높여 소스코드를 다양한 곳에 사용할 수 있으며 모듈간의 결합도도 낮출 수 있다.
- 쉽게 말하자면 게임 캐릭터라는 하나의 객체가 존재하는데, 그 객체를 더 잘 이용하기 위해서 무기, 방패 등 아이템을 가져와 결합시키는 것이다. 이 객체는 무기와 방패를 뺐다 꼈다 자유자재로 할 수 있으며 아이템을 갈아끼우는데 어떤 상황에 구애받지도 않는다.
- JAVA에서 데이터를 저장하고 가져오는 기능을 외부의 Oracle Database를 사용할 수도 있고, JDBC, iBatis, JPA 등 다른 프레임 워크를 이용해 짤 수도 있다. 이때 Spring을 이용하면 그때마다 **필요한 부분을 뺐다 꼈다 하면서 적절한 상황에 필요한 기능**을 해낼 수 있다.

3. AOP(Aspect Object Programming, 관점 지향 프로그래밍)
- 로깅, 트랜잭션, 보안 등 여러 모듈에서 공통적으로 사용하는 **기능을 분리하여 관리** 할 수 있다.
- 각각의 클래스가 있다고 가정하자. 각 클래스들은 서로 코드와 기능들이 중복되는 부분이 많다. 코드가 중복될 경우 실용성과 가독성 및 개발 속도에 좋지 않다. 중복된 코드를 최대한 배제하는 방법은 중복되는 기능들을 전부 빼놓은 뒤 그 기능이 필요할때만 호출하여 쓰면 훨씬 효율성이 좋다.
- 즉, AOP는 여러 객체에 공통으로 적용할 수 있는 기능을 구분함으로써 재사용성을 높여주는 프로그래밍 기법이다.

4. POJO(Plain Old Java Object) 방식
- POJO는 Java EE를 사용하면서 해당 플랫폼에 종속되어 있는 무거운 객체들을 만드는 것에 반발하여 나타난 용어이다.
- 별도의 프레임 워크 없이 Java EE를 사용할 때에 비해 인터페이스를 직접 구현하거나 상속받을 필요가 없어 기존 라이브러리를 지원하기 용이하고, 객체가 가볍다.
- 즉, getter/setter를 가진 단순한 자바 오브젝트를 말한다.

Class / Object / Instance / Method

Class
객체를 만들어 내기 위한 설계도 혹은 틀
연관되어 있는 변수와 메서드의 집합

Object
소프트웨어 세계에 구현할 대상
클래스에 선언된 모양 그대로 생성된 실체
특징
‘클래스의 인스턴스(instance)’ 라고도 부른다.
객체는 모든 인스턴스를 대표하는 포괄적인 의미를 갖는다.
oop의 관점에서 클래스의 타입으로 선언되었을 때 ‘객체’라고 부른다.

Instance
설계도를 바탕으로 소프트웨어 세계에 구현된 구체적인 실체
즉, 객체를 소프트웨어에 실체화 하면 그것을 ‘인스턴스’라고 부른다.
실체화된 인스턴스는 메모리에 할당된다.
특징
인스턴스는 객체에 포함된다고 볼 수 있다.
oop의 관점에서 객체가 메모리에 할당되어 실제 사용될 때 ‘인스턴스’라고 부른다.
추상적인 개념(또는 명세)과 구체적인 객체 사이의 관계 에 초점을 맞출 경우에 사용한다.
‘~의 인스턴스’ 의 형태로 사용된다.
객체는 클래스의 인스턴스다.
객체 간의 링크는 클래스 간의 연관 관계의 인스턴스다.
실행 프로세스는 프로그램의 인스턴스다.
즉, 인스턴스라는 용어는 반드시 클래스와 객체 사이의 관계로 한정지어서 사용할 필요는 없다.
인스턴스는 어떤 원본(추상적인 개념)으로부터 ‘생성된 복제본’을 의미한다.
예시

```
/* 클래스 */
public class Animal {
  ...
}
/* 객체와 인스턴스 */
public class Main {
  public static void main(String[] args) {
    Animal cat, dog; // '객체'

    // 인스턴스화
    cat = new Animal(); // cat은 Animal 클래스의 '인스턴스'(객체를 메모리에 할당)
    dog = new Animal(); // dog은 Animal 클래스의 '인스턴스'(객체를 메모리에 할당)
  }
}
```

클래스, 객체, 인스턴스의 차이

클래스(Class) VS 객체(Object)

클래스는 ‘설계도’, 객체는 ‘설계도로 구현한 모든 대상’을 의미한다.
객체(Object) VS 인스턴스(Instance)

클래스의 타입으로 선언되었을 때 객체라고 부르고, 그 객체가 메모리에 할당되어 실제 사용될 때 인스턴스라고 부른다.
객체는 현실 세계에 가깝고, 인스턴스는 소프트웨어 세계에 가깝다.
객체는 '실체', 인스턴스는 '관계'에 초점을 맞춘다.
객체를 '클래스의 인스턴스'라고도 부른다.
'방금 인스턴스화하여 레퍼런스를 할당한' 객체를 인스턴스라고 말하지만, 이는 원본(추상적인 개념)으로부터 생성되었다는 것에 의미를 부여하는 것일 뿐 엄격하게 객체와 인스턴스를 나누긴 어렵다.
참고

추상화 기법
분류(Classification)
객체 -> 클래스
실재하는 객체들을 공통적인 속성을 공유하는 범부 또는 추상적인 개념으로 묶는 것
인스턴스화(Instantiation)
클래스 -> 인스턴스
분류의 반대 개념. 범주나 개념으로부터 실재하는 객체를 만드는 과정
구체적으로 클래스 내의 객체에 대해 특정한 변형을 정의하고, 이름을 붙인 다음, 그것을 물리적인 어떤 장소에 위치시키는 등의 작업을 통해 인스턴스를 만드는 것을 말한다.
'예시(Exemplification)'라고도 부른다.

Object Oriented

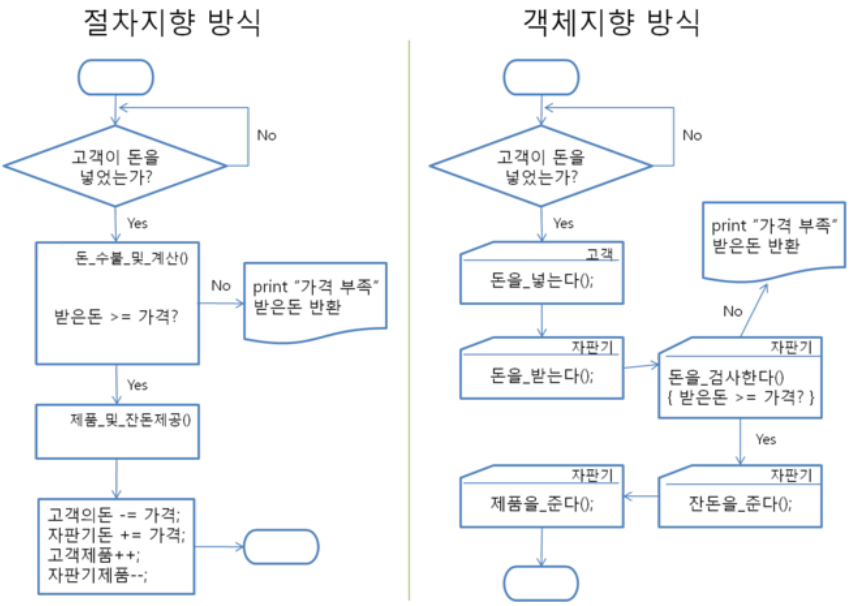
객체 지향(Object oriented) 프로그래밍이란?

우리가 실생활에서 쓰는 모든 것을 **객체**라 하며, 객체 지향 프로그래밍은 프로그램 구현에 필요한 객체를 파악하고 각각의 객체들의 역할이 무엇인지를 정의하여 객체들 간의 상호작용을 통해 프로그램을 만드는 것을 말한다. 객체는 클래스라는 틀에서 생겨난 실체(instance)이다. 따라서 객체 지향 프로그램은 객체와 객체 간의 연결로 되어 있으며 각각의 객체 안에 자료구조와 알고리즘이 들어있는 것이다.

프로그램 설계방법론이자 개념의 일종

- 프로그램을 단순히 데이터와 처리 방법으로 나누는 것이 아니라, 프로그램을 수많은 '객체(object)'라는 단위로 나누고 이들의 상호작용으로 서술하는 방식이다. 객체란 하나의 역할을 수행하는 '메소드와 변수(데이터)'의 묶음으로 봐야 한다.

절차 지향과 객체 지향



절차 지향

절차 지향 모델링은 프로그램을 **기능중심**으로 바라보는 방식으로 **"무엇을 어떤 절차로 할 것인가?"**가 핵심이 된다. 즉, 어떤 기능을 어떤 순서로 처리하는가에 초점을 맞춘다.

객체 지향

객체 지향 모델링은 기능이 아닌 **객체가 중심**이 되며 **"누가 어떤 일을 할 것인가?"**가 핵심이 된다. 즉, 객체를 도출하고 각각의 역할을 정의해 나가는 것에 초점을 맞춘다.

절차 지향 VS 객체 지향

- 대형 프로그래밍의 경우 많은 기능을 수반하기 때문에 절차 지향보다는 객체 지향이 적합
  - 각 객체가 하는 역할이 많아도, 많은 역할을 객체로 묶을 수 있기 때문
- 소형 프로그래밍의 경우 작은 기능을 수반하기 때문에 객체 지향보다는 절차 지향이 적합
  - 작은 기능을 객체별로 나눌 경우, 오히려 복잡해질 수 있기 때문

객체 지향 프로그래밍의 특징

1. 추상화(abstraction)
   - 객체들의 공통적인 특징(기능, 속성)을 도출하는 것
   - 객체지향적 관점에서는 클래스를 정의하는 것을 추상화라고 할 수 있다.(클래스가 없는 객체지향 언어도 존재 ex.JavaScript)

2. 캡슐화(encapsulation)
    - 실제로 구현되는 부분을 외부에 드러나지 않도록 하여 정보를 은닉할 수 있다.
    - 객체가 독립적으로 역할을 할 수 있도록 데이터와 기능을 하나로 묶어 관리하는 것
    - 코드가 묶여있어서 오류가 없어 편리하다.
    - 데이터를 보이지 않고 외부와 상호작용을 할 때는 메소드를 이용하여 통신을 한다. 보통 라이브러리로 만들어서 업그레이드해 사용할 수 있다.
3. 상속성(inheritance)
    - 하나의 클래스가 가진 특징(함수, 데이터)을 다른 클래스가 그대로 물려받는 것
    - 이미 작성된 클래스를 받아서 새로운 클래스를 생성하는 것
    - 기존 코드를 재활용해서 사용함으로써 객체지향 방법의 중요한 기능 중 하나에 속한다.
4. 다형성(polymorphism)
    - 약간 다른 방법으로 동작하는 함수를 동일한 이름으로 호출하는 것
    - 동일한 명령의 해석을 연결된 객체에 의존하는 것
    - 오버라이딩(Overriding), 오버로딩(Overloading)
        - 오버라이딩(Overriding) - 부모클래스의 메소드와 같은 이름을 사용하며 매개변수도 같되 내부 소스를 재정의하는 것
        - 오버로딩(Overloading) - 같은 이름의 함수를 여러 개 정의한 후 매개변수를 다르게 하여 같은 이름을 경우에 따라 호출하여 사용하는 것
5. 동적바인딩(Dynamic Binding)
    - 가상 함수를 호출하는 코드를 컴파일할 때, 바인딩을 실행시간에 결정하는 것.
    - 파생 클래스의 객체에 대해, 기본 클래스의 포인터로 가상 함수가 호출될 때 일어난다.
    - 함수를 호출하면 동적 바인딩을 통해 파생 클래스에 오버라이딩 된 함수가 실행
    - 프로그래밍의 유연성을 높여주며 파생 클래스에서 재정의한 함수의 호출을 보장(다형 개념 실현)

객체 지향 프로그래밍의 장점

소프트웨어의 생산성 향상

객체지향 프로그래밍은 다형성, 객체, 캡슐화 등 소프트웨어의 재사용을 지향한다. 이미 만들어진 클래스를 상속받거나 객체를 가져다 재사용하거나, 부분 수정을 통해, 소프트웨어를 작성하는 부담을 대폭 줄일 수 있다.

- 신뢰성 있는 소프트웨어를 손쉽게 작성할 수 있다. (개발자가 만든 데이터를 사용하기 때문에 신뢰할 수 있다.)
- 코드를 재사용하기 쉽다 (상속, 캡슐화, 다형성으로 인해 재사용할 수 있다.)
- 업그레이드가 쉽다.
- 디버깅이 쉽다.

실세계에 대한 쉬운 모델링

컴퓨터가 산업 전반에 다양하게 활용되는 요즘 시대에는 응용 소프트웨어를 하나의 절차로 모델링하기 어렵다. 산업 전반에서 요구되는 응용 소프트웨어 특성상, 절차나 과정보다 관련된 많은 물체(객체)들의 상호 작용으로 묘사하는 것이 더 쉽고 적합하다.

- 실세계에 대한 모델링을 좀 더 쉽게 해준다. (모든 것을 객체들의 상호작용으로 생각)

보안성 향상

객체 지향적 프로그래밍의 캡슐화 특징으로 실제로 구현되는 부분을 외부에 드러나지 않도록 하여 정보를 은닉할 수 있다.

- 보안성이 높다 (캡슐화, 데이터 은닉, 다형성으로 인해 필요한 정보를 재정의하거나 getter, setter를 이용하기 때문에 보안성이 높다.)

객체 지향 프로그래밍의 단점

느린 실행 속도

객체 지향 프로그래밍은 캡슐화와 격리구조에 때문에 절차지향 프로그래밍과 비교하면 실행 속도가 느리다. 또한, 객체지향에서는 모든 것을 객체로 생각하기 때문에 추가적인 포인터 크기의 메모리와 연산에 대한 비용이 들어가게 된다.

- 절차지향 프로그래밍에 비해 느린 실행 속도

- 필요한 메모리양의 증가