스프링 프레임워크

스프링 프레임워크 개요

Spring Framework

Spring Framework

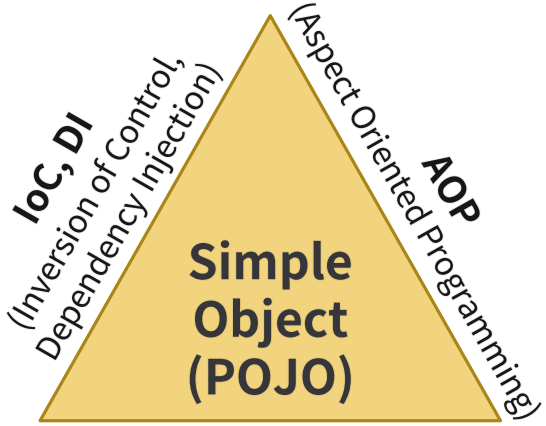
- Spring이 관리하는 하위 프로젝트 중 하나이다.
- 특정한 환경에 종속되지 않는 최신 자바 기술 기반의 기업(Enterprise) 급 응용 프로그램 개발에 적합한 환경을 제공하는 오픈소스 애플리케이션 프레임워크이다.
  - JEE(Java Platform, Enterprise Edition)
    - 복잡한 비즈니스 프로세스와 다양한 계층, 개체간 데이터 트랜잭션을 가지고 있는 엔터프라이즈 규모와 특정 운영체계 및 환경에 종속되지 않는 프로젝트를 위한 기술이다.
    - 서버, 분산 등 로우 레벨 기술의 추상화를 위해 복잡한 스펙(Over Spec)을 제공하였다.
- JEE(Java/Jakarta EE)의 스펙을 객체지향을 이용하여 가볍게 지원하도록 설계되어 있다.
- 특정 기술에 종속되지 않는 POJO(Plain Old Java Object) 프로그래밍을 지원하는 프레임워크로 객체지향형(상속, 다형성 등)의 장점을 최대한 활용한다.
- 다양한 디자인 패턴(IoC, Factory, Singleton, MVC등)을 활용하여 객체지향형 방법론을 적극적으로 적용하고 있다.
- 객체의 생성, 소멸 등 라이프 사이클(Life Cycle)을 관리하는 경량 컨테이너(Container)를 제공한다.
- Java, C#, Kotlin등의 언어에서 사용 가능하다.

Spring Framework

◈ POJO(Plain Old Java Object) 프로그래밍

- ◆ 특징
  - 특정 기술 규약 및 환경에 의존성을 줄이기 위해 순수 자바 기술(규약)로 구현한다.
  - 객체 지향형 개념을 준수해야 한다.
  - 하나의 기능은 하나의 클래스에 담는 단일 책임 원칙을 지킨다.
  - 테스트 등의 작업들에 대한 자동화 및 유지보수가 용이하다.
- ◆ 스프링 프레임워크는 POJO프로그래밍을 지원하기 위해 다음과 같은 기술을 제공한다.

Spring Triangle



Spring Framework

- 제어의 역전(IoC : Inversion of Control)
  - 객체생성이나 코드 실행을 프로그래머가 정하는 것이 아니라 프레임워크에 위임하는 설계 개념이다.
  - 프레임워크는 필요할 경우 위임된 코드를 프레임워크의 일부분으로 실행해 준다.
- 의존성 주입(DI : Dependency Injection)
  - 계층, 서비스 등 애플리케이션 구성 요소들 사이에 의존성이 발생할 경우 미리 설정된 의존성 정보를 이용하여 필요시 서로 연결하여 결합을 느슨하게 하여 애플리케이션의 유연성을 높이는 디자인 패턴이다.
- 관점 지향 프로그래밍(AOP : Aspect-Oriented Programming)
  - 클래스 중심의 객체지향형(OOP)의 단점인 공통 기능의 중복 문제를 해결하기 위한 프로그래밍 패러다임으로 기능 중심으로 모듈화를 추구한다.
  - 트랜잭션이나 로깅, 보안과 같이 공통적으로 기능을 클래스에서 분리하여 모듈로 만들고 사용 및 관리할 수 있다.
- 서비스 추상화(PSA : Portable Service Abstraction)
  - 데이터베이스 연결 및 로깅과 같은 기반 기술(JDBC, Slf4j등)들을 추상화 하여 분리한 뒤, 일관성 있는 인터페이스(어노테이션)를 제공하여 기반 기술에 대한 지식없이 이용 가능하게 서비스이다.
  - 외부 서비스에 대한 의존성을 최소화하여 애플리케이션의 유연성을 높인다.

Spring Framework

- POJO 프로그래밍 예

```
public class User {
    private String name;

    public void setName(String name) {
        this. name=name;
    }
    public String getName() {
        return name;
    }
}
```

```
public class HomeServlet extends HttpServlet {
    @Override
    public void init() throws ServletException {
        // code
    }
}
```

```
@Controller
public class HomeController {
    @GetMapping("/home")
    public void home(){
        // code
    }
}
```

```
public class Printer {
    public void print(String material) {
        switch(material) {
            case "ink":
                // code
            case "laser":
                // code
            case "3d":
                // code
            default:
                // code
        }
    }
}
```

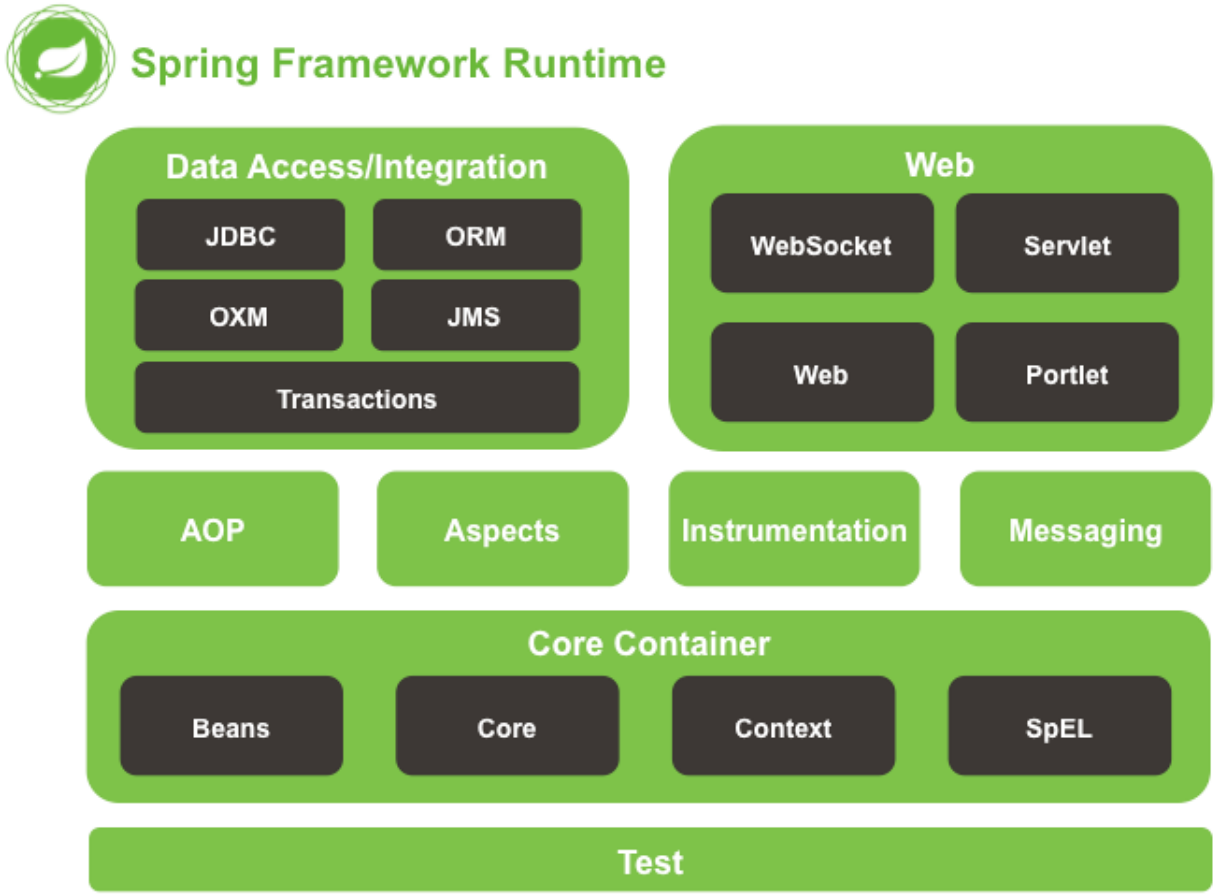
```
public class Printer {
    public void print(Ink material) {
        // code
    }
    public void print(Laser material) {
        // code
    }
    public void print(ThreeD
        // code
    }
}
```

```
Printer printer = new Printer();
printer.print("ink");

printer.print(new Ink());
printer.print(new Laser());
```

Spring Framework

- 주요 모듈(Modules) 구성
  - 스프링 프레임워크는 약 20개의 모듈로 구성되어 있다.



출처: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/images/spring-overview.png

Spring Framework

모듈명	설명
•Spring Core Container	
spring-beans	스프링 빈의 의존성 주입(DI)기능을 제공한다. 빈 팩토리(Bean Factory)가 구현되어 있다.
spring-core	다른 스프링 모듈이 사용하는 공통 기능을 제공한다.
spring-context	빈 팩토리를 상속받는 어플리케이션 컨텍스트(Application Context)를 제공한다. 리소스 관리 및 국제화를 지원한다.
spring-expression	스프링EL(표현어, Expression Language), SpEL을 이용하여 쿼리(Query) 및 조작 기능을 제공한다.
•Spring AOP and Instrumentation	
spring-aop	제한적인 AOP(Aspect Oriented Programming, 관점지향 프로그래밍)에 대한 기능을 제공한다.
spring-aspects	AspectJ와 통합을 지원한다.
spring-instrument	기본적인 인스트루먼테이션(instrumentation) 기능을 제공한다.
•Spring Test	
spring-test	단위 또는 통합 테스트에 관련된 기능을 제공한다.
•Spring Data Access/Integration	
spring-jdbc	JDBC(Java Database Connectivity)를 추상화 하여 간편하게 사용 가능하게 한다.
spring-orm	JPA(Java Persistence API), JDO(Java Data Objects), Hibernate와 같은 ORM(Object Relational Mapping)관련 기능을 제공한다.
spring-oxm	JAXB(Java Architecture for XML Binding), XMLBeans 와 같이 객체/XML 매핑 기능을 제공한다.
spring-messaging	JMS, AMQP(Advanced Message Queuing Protocol)등을 추상화 하여 간편하게 사용 가능하게 한다.
spring-jms	JMS(Java Message Service)를 추상화 하여 간편하게 사용 가능하게 한다.

Spring Framework

모듈명	설명
Spring-tx	트랜잭션(Transaction)을 추상화 하여 간편하게 사용 가능하게 한다.
•Spring Web	
spring-web	Multipart-FileUpload등 애플리케이션 제작에 필요한 기능을 제공한다.
spring-webmvc	Web MVC 프레임워크를 제공한다.
spring-websocket	Websocket에 관련된 기능을 제공한다.
spring-webmvc-portlet	Portlet 환경을 지원하는 MVC 프레임워크를 제공한다.

상속과 구성

- 상속(Inheritance), 구성(Composition)
  - 다형성과 코드 재사용을 위한 객체지향형 프로그래밍 기법 들이다.
  - 상속(Inheritance)
    - 클래스 중심적인 방법으로 포함관계를 의미한다.
    - is-a : A is a B.
      - A는 B를 상속 받는다. (Americano is a Coffee.)
    - 컴파일(compile)시 의존성이 정해지고 정적이다.
  - 구성, 합성(Composition)
    - 객체 중심적인 방법으로 구성관계를 의미한다.
    - has-a : A has a B.
      - B는 A에 속해 있다. (Car has a Wheel.)
    - 실행(Runtime)시 의존성이 정해지고 동적이다.
    - 인터페이스(Interface)를 이용하여 유연성을 확보한다.
  - 함수의 합성(합성함수, Function Composition)
    - 홀수 함수: f(x) = 2x+1, 2배수 함수: g(y) = 2y
    - 홀수를 2배수하여라 g(f(x)) = 2*f(x) = 2*(2x+1)

의존성 주입

의존성 주입

◈ 제어의 역전(IoC : Inversion of Control)

- 프로그래머가 자신이 작성한 코드 실행을 프레임워크에 위임하는 설계 개념이다. 프레임워크는 필요할 경우 위임된 코드를 프레임워크의 일부분으로 실행해 준다.
- 프로그래머는 확장성 및 테스트 등을 직접 구현하지 않고 프레임워크를 이용하며 클래스 및 계층, 라이브러리와의 의존성을 줄일 수 있다.
- 제어의 역전은 의존성 검색(DL, Dependency Lookup)과 의존성 주입(DI, Dependency Injection) 디자인 패턴으로 구현된다.
- 스프링 프레임워크는 IoC 컨테이너를 이용하여 제어의 역전 기능을 구현한다.
- 콜백(CallBack), 템플릿 메서드 패턴(Template Method Pattern)등이 포함된다.

```
public abstract class TemplateMethods {
      abstract void methodB();

      private void methodA() { // Code };
      private void methodC() { // Code };
      public void run() {
            methodA();
            methodB();
            methodC();
      }
}
```

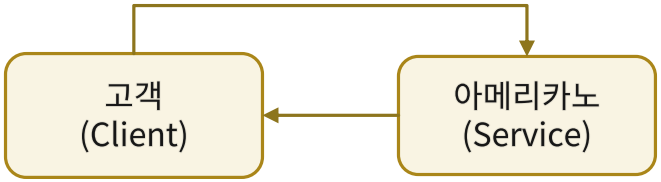
```
public class ClassA extends TemplateMethods {
      @Override
      void methodB() { // Code };
}
```

```
public static void main(String[] args) {
      TemplateMethods tmp;
      tmp = new ClassA();
      tmp.run();
}
```

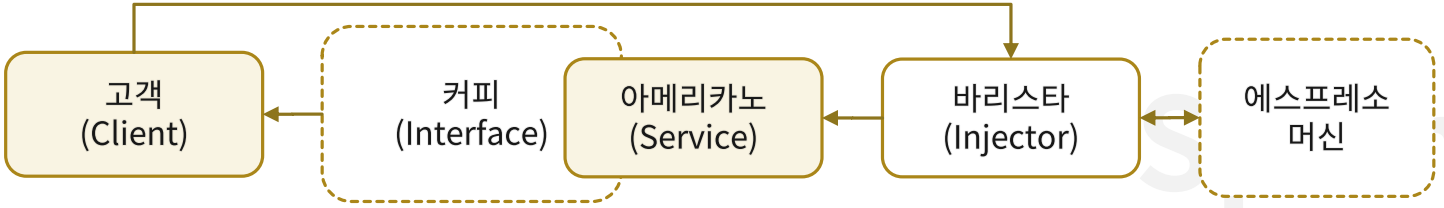
의존성 주입

◈ 의존성 주입(DI : Dependency Injection)

- 계층, 서비스 등 애플리케이션 구성 요소들 사이에 의존성이 있을 경우 프레임워크가 서전 설정된 의존성 정보를 이용하여 서로 연결시켜준다.
- 객체를 직접 생성하는 등 행동으로 의존성이 증가한 경우 수정, 개발, 테스트, 배포 등 의도에 따라 전체 코드를 수정해야 되어 자동화 구현에 비효율적이다.
- 주입을 이용하면 주입될 객체의 상태(state), 행동(behavior) 명세가 유지될 경우 객체 생성 부분만 변경하여 수정을 최소화 할 수 있다.
- 스프링프레임 워크는 IoC 컨테이너에 객체들을 미리 생성한 뒤 객체가 필요한 경우 주입하여 이용할 있게 한다.
- 의존성 - 고객이라는 클라이언트가 아메리카노라는 서비스에 의존적이다.



- 의존성 감소 - 바리스타라는 주입자를 이용하여 의존성을 줄였다.



의존성 주입

◈ 의존성 주입 구현

◆ 생성자(Constructor)

```
public class Barista {
	private Coffee coffee;
	private Map<Class<?>, Coffee> coffeeMap = new HashMap<>() {{
		put(Americano.class, new Americano());
		put(CafeMocha.class, new CafeMocha());
	}};

	public Barista(Class<?> coffee) {
		this.coffee = coffeeMap.get(coffee);
	}

	public void getCoffee() {
		System.out.println(this.coffee.getCoffee() + "가 나왔습니다.");
	}
}
```

```
barista = new Barista(Americano.class);
barista.getCoffee();
barista = new Barista(CafeMocha.class);
barista.getCoffee();
```

의존성 주입

- 세터(Setter)

```
public class Barista {
	private Coffee coffee;
	private Map<Class<?>, Coffee> coffeeMap = new HashMap<>() {{
		put(Americano.class, new Americano());
		put(CafeMocha.class, new CafeMocha());
	}};

	public void setCoffee(Class<?> coffee) {
		this.coffee = coffeeMap.get(coffee);
	}

	public void getCoffee() {
		System.out.println(this.coffee.getCoffee() + "가 나왔습니다.");
	}
}
```

```
Barista barista = new Barista();

barista.setCoffee(Americano.class);
barista.getCoffee();

barista.setCoffee(CafeMocha.class);
barista.getCoffee();
```

의존성 주입

- 팩토리 패턴(Factory DP)

```
public class Barista {
	Map<String, Coffee> coffeeMap = new HashMap<>() {{
		put("아메리카노", new Americano());
		put("카페모카", new CafeMocha());
	}};

	public Coffee getBean(String coffee) {
		return coffeeMap.get(coffee);
	}
}
```

```
Barista barista = new Barista();

Coffee coffee;
coffee = barista.getBean("아메리카노");
System.out.println(coffee.getCoffee());

coffee = barista.getBean("카페모카");
System.out.println(coffee.getCoffee());
```

참고자료

- https://guide.ncloud-docs.com/docs/sso-integration-info
- https://docs.spring.io/spring-framework/docs/current/reference/html/core.html
- https://spring.io/projects/spring-framework
- https://spring.io/projects/spring-boot
- https://ko.wikipedia.org/wiki/스프링_프레임워크
- https://www.geeksforgeeks.org/pojo-vs-java-beans/
- https://ko.wikipedia.org/wiki/의존성_주입
- https://ko.wikipedia.org/wiki/관점_지향_프로그래밍
- https://namu.wiki/w/객체 지향 프로그래밍/원칙