Spring 이해하기

웹상에서 프로그램이 필요하면 서비스를 처리하는 클래스가 클래스를 직접 만들어야 할까?

필요한 클래스가 한둘은 아닐 테고 그 많은 클래스를 어떻게 관리하지? 객체를 필드로 선언해서 1개만 공유해도 될 것 같은데 그렇다면 자원을 공유할 때 문제가 생길 것 같으면서도 아닌 것 같고...

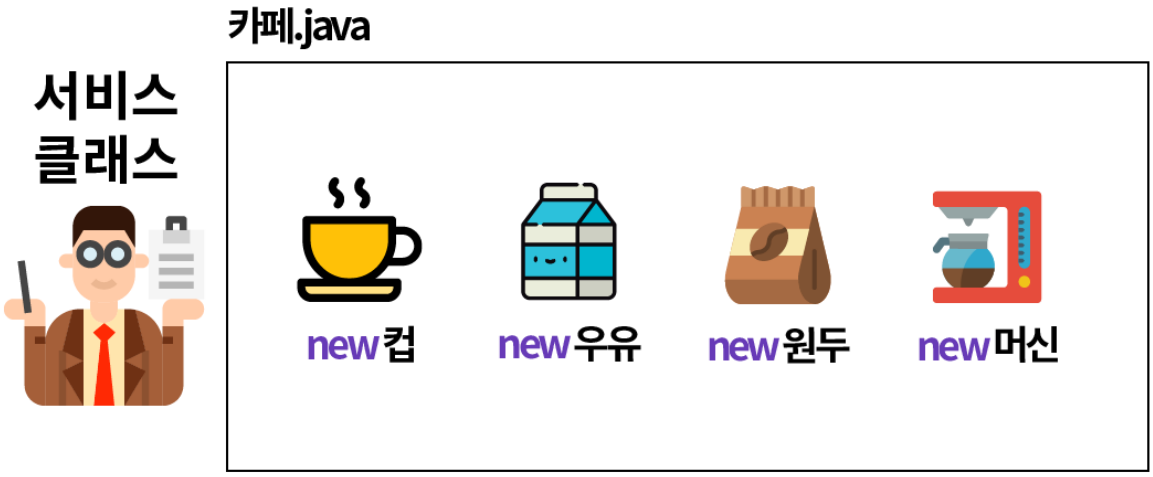
바로 이러한 개념에서 객체를 내 서비스 클래스에서 생성하지 않고 누군가가 내 서비스의 메서드를 호출하고 그 메서드에 클래스를 보내주는 방식으로 구현을 하고 싶었기 때문에 이러한 기술이 등장한 것 같다.

특징1. 객체 의존 주입 DI(Dependency Injection)

> "객체를 생성하고 관리하는 건 내가(스프링 프레임워크) 할 테니까 넌 서비스만 해!"

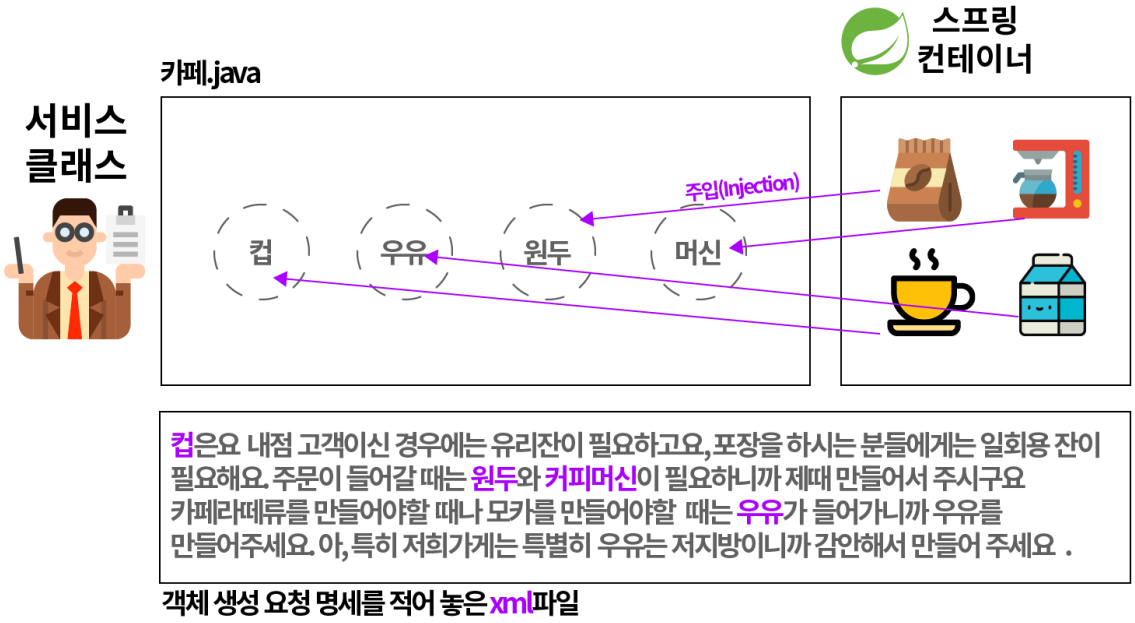
이러한 프레임워크의 등장 덕분에 우리는 객체를 생성해서 어떻게 자원을 관리할지 생각해야 하는 부담이 줄었다. 이제부터는 스프링 프레임워크의 특징에 대해 간단하게 알아보도록 하자.

기존 java처럼 서비스 클래스에서 new하여 사용한 경우



기존에는 new로 객체를 생성하여 객체를 생성하고 삭제하는 연산을 모두 해당 서비스 클래스에서 삭제한 모습이다.

기존 java환경과 달리 스프링 컨테이너에게 객체 생성을 맡긴 경우



기존 java환경과는 다르게 스프링 컨테이너에게 객체 생성을 맡긴 경우, bean객체를 가져와서 서비스 클래스가 활용할 수 있도록 도와준다.

단, 기존에 bean객체는 싱글톤 패턴 기반으로 관리된다는 것에 유의한다.

(처음에는 bean객체에 관해서도 모르고 시작했기 때문에 더 어렵게만 느껴지는 스프링프레임워크였다. bean객체에 관해서는 짤막하게 다음 포스팅에서 소개하도록 하겠다. 일단 모르는 개념이라면 자바에서 객체를 생성하는 하나의 규약인데 이 규약을 지켜 생성되는 객체를 bean이다 하는 정도만 보고 넘어갈 것!)

이전에 알던 객체 생성 방법

자, 학교와 학생이 있다고 치고! 학교를 거대한 자바 프로그램이라고 생각해 봅시다.

그리고 학생은 연필과 지우개를 가지고 있어야 공부할 수 있다고 생각해 봅니다.


```
class Student{
    private Pencil pencil;
    private Eraser eraser;

    public void doStudy(String subject) {
        System.out.println(subject + "을 공부하다");
    }
}

public class School {
    public static void main(String[] args) {
        Student student = new Student();
        student.doStudy("스프링");
    }
}
```

> 위의 관계에서 학교와 학생은 "의존 관계에 있다"고 말할 수 있습니다.

이제부터 알아야 할 객체 생성 방법

스프링 웹 프로젝트가 아니라면 웬만해서 자바 언어로는 대부분 콘솔 창에서 입력하는 것을 연습하셨을 것인데요.

그것은 정말 소규모이고 자바 언어를 익히고자 하는 것에 중점을 두었다면 이제는 그게 아닙니다.

웹에는 정말 많은 사람들이 접속하는 플랫폼이고 많은 자원들을 효율적으로 관리해야 합니다.

우리 신생 개발자들이 이런 모든 처리들을 다 할 수 있으면 얼마나 좋을까요?

마치 고급 언어 필요없이 모든 프로그램을 어셈블리어 내지는 기계어로 코딩하는 능력을 가졌다면요...

꼭 신생 개발자가 아니더라도 시니어 프로그래머들도 자원을 관리할 수 있는 충분한 능력은 가질 수 있지만, 반대로 자원 관리에만 치중하다가 정작 본인들이 서비스해야 할 개발 부분을 놓친다면요?

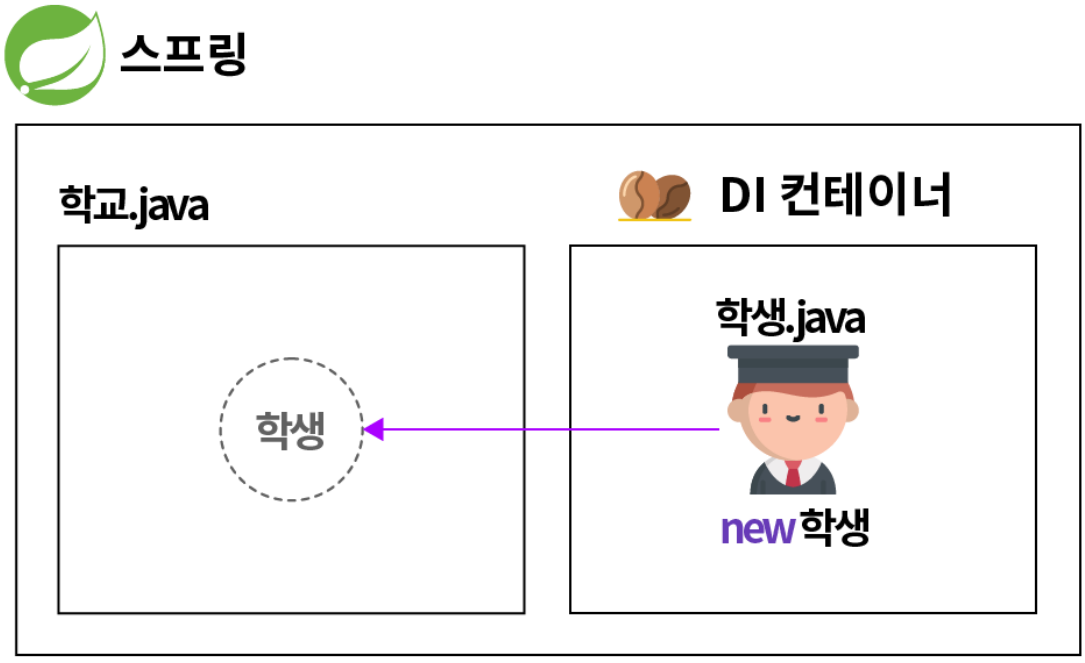
이러한 의미에서 "너넨 너네 사업 프로그래밍에나 신경 써. 앞으로 웹 자원 관리는 내(스프링)가 할게."라고 하며 프레임워크가 등장하게 된 것이죠.

Dependency Injection

우리는 스프링이 밑단에서 하드캐리 해준다고 하니, 관리 능력을 얻은 셈입니다.

그 대신 우리에게는 더는 자원을 생성할 권한이 없습니다. (엄밀히 따지면 할 수는 있지만 그렇게 하지 않고 모두 스프링에 위탁하겠다는 의미입니다.)

대신 스프링에게 우리가 사용해야 할 자원을 명시해 두는 겁니다. 바로 아래의 그림처럼 스프링에게 자원을 맡겨 두고 필요할 때 가져다가 쓰는 것입니다.



어떤가요? 그림으로는 매우 간단하죠.

그림으로 보면 학교는 학생이라는 객체를 생성해서 써야 합니다.

스프링 DI컨테이너에서는 학생이라는 객체를 잘 생성하여 보관해 두었다가 학교가 필요할 때 서비스 루틴에 잘 주입해 주고 있네요.

다시말해 이전에는 학교라는 클래스 "내부"에서 객체를 자체적으로 생성했다면, 이번에 스프링DI 컨테이너라는 "외부"에서 학교로 객체를 "주입"해 주고 있습니다.

이것이 바로 "의존성 주입"이라고 하는 것입니다. 한자에 너무 겁먹을 필요 없죠?

IoC(Inversion Of Control)

그리고 바로 여기서 새로 등장하는 용어가 바로 "제어의 역전"을 뜻하는 Ioc(Inversion Of Control)입니다.

그동안 프로그램 실행의 주도권은 public static void main(String[] args)를 선언한 "내" 프로그램이 지녔다면,

이제부터 스프링 프레임워크를 이용하는 한 나의 객체를 생성하고 메서드를 불러주는 "스프링"이 제어권을 가지고 있다고 봐야 합니다.

어떻게 의존성 주입을 할 것인가?

이제는 의존성 주입이라는 용어를 이해했다면, 어떻게 "주입할 것인가"에 관해 알아보도록 하겠습니다.

너와 나의 연결고리, XML

스프링 DI컨테이너는 객체를 보관하여 필요할 때 꺼내서 주입시켜줄 수는 있지만, 프로그램이 실행되는 동안 어떤 객체를 사용할 것인지에 관해서는 일일이 알 수 없습니다.

왜냐하면 스프링 프레임워크는 AI기반으로 돌아가는 프로그램 로직이 아니기 때문이죠.

그 때문에 프로그래머가 필요한 객체를 미리 선언해 두고, 스프링 DI컨테이너에 맡긴 객체를 그때 그때 불러다 쓰는 방식으로 가야 합니다.

이때 이 프로그램에서 "나는 이때 학생 객체가 필요하니까 미리 만들어 줄래?" 하는 구문을 명시해야 합니다. 그러한 구문으로서 XML로 작성을 하게 되는 것입니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springf
ramework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schem
a/aop/spring-aop-4.3.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/sche
ma/util/spring-util-4.3.xsd">

<!--bean 스프링님, 연필을 생성해 주세요. /--><!--bean 스프링님, 학생을 생성해 주세요. 학생은 연필을 꼭
가져야 하니 생성자로 연필을 꼭 넘겨주시구요. /--><!--bean 스프링님, 학교를 생성해 주세요. /--></beans>
```

나는 Java가 좋아, 어노테이션

XML을 이용하여 선언하는 반면, Java만으로도 얼마든지 XML에 선언한 효력과 동일한 선언을 할 수도 있습니다.

@Configuration, @Autowired, @Controller, @Service 등 정말 많은 어노테이션을 사용하게 될 텐데, 2020년 포스팅을 작성하는 시각을 기준으로는 XML보다는 어노테이션을 이용한 방법이 좀 더 대세라고 생각해야 할 것입니다. 기존에는 XML이라는 체계적인 틀을 짜두어 유지보수를 편리하게 하고자 의도했지만, 오히려 XML을 많이 사용하면 내용이 지나치게 길어져서 오히려 유지보수하는 데 역효과를 가져왔기 때문이 아닐까 싶습니다. 하지만, 회사마다 사용하는 방식과 문화가 다를 테니 주니어 개발자라면 두 가지 모든 방식을 다 알아두어야 합니다.