**[ 02 ] DI(Dependency Injection) 개념 및 활용** – 기초단계에서 가장 중요한 개념

1. 스프링을 이용한 객체 생성과 조립(스프링은 객체 생성과 조립을 하는 컨테이너라 볼 수 있다)
   * 지난 예제는 스프링의 특징(사용법)이 적용되지 않은 프로젝트였습니다.
   * 이번 시간에는 스프링의 특징(사용법)을 적용하여 스프링과 좀 더 친근해 질 수 있는 시간을 갖도록 하겠습니다.
   * 스프링을 사용하지 않은 프로젝트

Calculation cal = **new** Calculation();

cal.setNum1(10);

cal.setNum2(5);

cal.add();

cal.sub();

cal.mul();

cal.div();

* + 스프링 부트는 기본적으로 자바 config를 기반으로 하기 때문에 DI 설정파일을 추가하는 기능이 제공되지 않음. 그래서 sping3을 플러그인 하여야 한다.
  + 프로젝트 오른쪽 마우스 -> Configure -> Convert to Maven Project
  + Pom.xml에 다음을 추가

<dependencies>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.2.8.RELEASE</version>

</dependency>

</dependencies>

(1) pom.xml에서 dependencies에 springframework를 검색하여 추가

(2)위의 내용은 google에서 “maven repository” 검색후 나오는 첫 사이트(https://mvnrepository.com/)를 방문한 후 “springframework” search하여 나온 Spring Context를 클릭 -> 버전이 높은 것을 선택하여 가져올 소스를 복사하여 붙인다

* + 외부 컨테이너에서 부품조립을 위한 클래스들

ApplicationContext

* + - AbstractApplicationContext
      * ClassPathXmlApplicationContext(GenericXmlApplicationContext)
      * AnnotationConfigApplicationContext
  + 스프링을 적용한 프로젝트

<bean class=*"com.lec.ex01.Calculation"* id=*"calculation"*>

<property name=*"num1"*>

<value>10</value>

</property>

<property name=*"num2"*>

<value>5</value>

</property>

</bean>

// 스프링 컨테이너가 xml을 파싱하여 빈객체 생성

**public** **class** CalculationTestMain {

**public** **static** **void** main(String[] args) {

// Calculation cal = new Calculation();

// cal.setNum1(10);

// cal.setNum2(5);

AbstractApplicationContext context = **new** GenericXmlApplicationContext("com/lec/ex1/applicationCTX.xml");

//AbstractApplicationContext context2 = new ClassPathXmlApplicationContext("com/lec/ex1/applicationCTX.xml");

Calculation cal = context.getBean("calculation", Calculation.**class**);

//Calculation cal = (Calculation)context.getBean("calculation");

//Calculation cal = context.getBean(Calculation.class);

cal.add();

cal.sub();

cal.mul();

context.close();

}

}

1. 스프링 설정 파일의 이해 ; 필드에 대한 setter함수들이 있어서 property의 value값을 줄 수 있다.

**public** **class** Calculator {

**public** **void** addition(**int** n1, **int** n2) {

System.***out***.println("더하기");

System.***out***.println(n1 + " + " + n2 + " = " + (n1+n2));

}

**public** **void** subtraction(**int** n1, **int** n2) {

System.***out***.println("빼기");

System.***out***.println(n1 + " - " + n2 + " = " + (n1-n2));

}

**public** **void** multiplication(**int** n1, **int** n2) {

System.***out***.println("곱하기");

System.***out***.println(n1 + " \* " + n2 + " = " + (n1\*n2));

}

**public** **void** division(**int** n1, **int** n2) {

System.***out***.println("나누기");

System.***out***.println(n1 + " / " + n2 + " = " + (n1/n2));

}

}

@Data

**public** **class** MyCalculator {

**private** Calculator calculator;

**private** **int** num1;

**private** **int** num2;

**public** **void** add() {

calculator.addition(num1, num2);

}

**public** **void** sub() {

calculator.subtraction(num1, num2);

}

**public** **void** mul() {

calculator.multiplication(num1, num2);

}

**public** **void** div() {

calculator.division(num1, num2);

}

}

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

// MyCalculator myCalculator = new MyCalculator();

// myCalculator.setCalculator(new Calculator());

// myCalculator.setNum1(10);

// myCalculator.setNum2(5);

String resourceLocation = "classpath:applicationCTX.xml";

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext(resourceLocation);

MyCalculator myCalculator = ctx.getBean("myCalculator", MyCalculator.**class**);

myCalculator.add();

myCalculator.sub();

myCalculator.mul();

myCalculator.div();

}

}

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"calculator"* class=*"com.lec.ch02.Calculator"*/>

<bean id=*"myCalculator"* class=*"com.lec.ch02.MyCalculator"*>

<property name=*"calculator"*>

<ref bean=*"calculator"*/>

</property>

<property name=*"num1"* value=*"10"*/>

<property name=*"num2"* value=*"5"*/>

</bean>

</beans>

1. 스프링 프로퍼티 설정

; 기초데이터, list타입, 다른 빈 객체 참조 방법 등을 살펴본다

BMICalculator.java

**import** lombok.Data;

@Data

**public** **class** BMICalculator {

**private** **double** lowWeight;

**private** **double** normal;

**private** **double** overWeight;

**private** **double** obesity;

**public** **void** bmiCalculation(**double** weight, **double** height) {

**double** h = height \* .01;

**double** result = weight / (h\*h);

System.***out***.println("BMI 지수 : "+(**int**)result);

**if**(result>obesity){

System.***out***.println("비만입니다");

}**else** **if**(result>overWeight){

System.***out***.println("과체중입니다");

}**else** **if**(result>normal){

System.***out***.println("정상입니다");

}**else** **if**(result>lowWeight){

System.***out***.println("저체중입니다");

}**else** {

System.***out***.println("심각하네요. 살 좀 찌세요");

}

}

}

MyInfo.java

**import** lombok.Data;

@Data

**public** **class** MyInfo {

**private** String name;

**private** **double** height;

**private** **double** weight;

**private** ArrayList<String> hobbies;

**private** BMICalculator bmiCalculator;

**private** **void** bmiCalculation(){

bmiCalculator.bmiCalculation(weight,height);

}

**public** **void** getInfo(){

System.***out***.println("이름 : "+name);

System.***out***.println("키 : "+height);

System.***out***.println("몸무게 : "+weight);

System.***out***.println("취미:"+ hobbies);

bmiCalculation();

}

}

applicationCTX2.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"bmiCalculator"* class=*"com.lec.ch02.bmi.BMICalculator"*>

<property name=*"lowWeight"* value=*"18.5"*/>

<property name=*"normal"* value=*"23"*/>

<property name=*"overWeight"* value=*"25"*/>

<property name=*"obesity"* value=*"30"*/>

</bean>

<bean id=*"myInfo"* class=*"com.lec.ch02.bmi.MyInfo"*>

<property name=*"bmiCalculator"* ref=*"bmiCalculator"*/> <!-- 다른 빈 객체 참조 -->

<property name=*"height"* value=*"170"*/> <!--기본데이터-->

<property name=*"weight"* value=*"65"*/>

<property name=*"name"* value=*"홍길동"*/>

<property name=*"hobbies"*> <!--List 타입데이터-->

<list>

<value>수영</value>

<value>요리</value>

<value>독서</value>

</list>

</property>

</bean>

</beans>

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

String resourceLocation = "classpath:bmi/applicationCTX2.xml";

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext(resourceLocation);

MyInfo myInfo = ctx.getBean("myInfo", MyInfo.**class**);

myInfo.getInfo();

ctx.close();

}

}

1. 스프링 컨테이너의 이해
   * 스프링 컨테이너를 생성하고 컴포넌트를 사용하는 방법에 대해서 알아보자

String resourceLocation = "classpath:bmi/applicationCTX3.xml";

AbstractApplicationContext ctx = new GenericXmlApplicationContext(resourceLocation);

// 스프링 컨테이너 생성

MyInfo myInfo = ctx.getBean(“myInfo”, MyInfo.class);

// 스프링 컨테이너에서 컴포넌트 가져옴

myInfo.getInfo();

ctx.close();

// AbstractApplicationContext 추상클래스 위로는 ApplicationContext 인터페이스 밑으로 상속받는 얘들중의 하나가 GenericXmlApplicationContext. 매개변수로 위치를 주면 IOC 컨테이너(스프링 컨테이너)가 생성. 즉 부품이 된 거임. 그 부품에서 “myInfo”라는 빈(이 안에도 bmiCalculator 주입됨)을 뽑아서 myInfo라는 변수에 넣음. 객체를 생성하고 조립하는 것을 Spring 컨테이너가 하고 있음

1. 의존 관계
   * DI는 Dependency Injection의 약자로 우리말로 하면 ‘의존하는 객체를 주입하는 것‘ 입니다.
   * 의존주입은 setter를 통한 주입과 생성자를 통한 주입이 있습니다.
   * xml파일을 이용한 의존주입(아래의 예제는 studentInfo객체는 student객체에 의존하고 있다)

<bean id=*"studentInfo"* class=*"com.lec.ex.StudentInfo"*>

<!-- <constructor-arg ref="student1"/> -->

<constructor-arg>

<ref bean=*"student1"*/>

</constructor-arg>

</bean>

* + java파일을 이용한 의존주입

Student student2 = ctx.getBean(“student2”,Student.class);

studentInfo.setStudent(student2);

studentInfo.getStudentInfo();

1. 의 경우 예제 : 생성자 함수를 통해 주입하고 있다

**Student.java**

@Data

@NoArgsConstructor

@AllArgsConstructor

**public** **class** Student {

**private** String name;

**private** String age;

**private** String gradeNum;

**private** String classNum;

}

**StudentInfo.java**

**import** lombok.AllArgsConstructor;

**import** lombok.Data;

@Data

@AllArgsConstructor

**public** **class** StudentInfo {

**private** Student student;

**public** **void** getInfo(){

**if**(student!=**null**){

System.***out***.println("이름:"+student.getName());

System.***out***.println("나이:"+student.getAge());

System.***out***.println("학년:"+student.getGradeNum());

System.***out***.println("반:"+student.getClassNum());

System.***out***.println("======================");

}**else** {

System.***out***.println("없어");

}

}

}

**applicationCTX.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"student1"* class=*"com.lec.ch02.student.Student"*>

<constructor-arg >

<value>홍길동</value>

</constructor-arg>

<constructor-arg>

<value>10</value>

</constructor-arg>

<constructor-arg>

<value>3학년</value>

</constructor-arg>

<constructor-arg>

<value>20반</value>

</constructor-arg>

</bean>

<bean name=*"student2"* class=*"com.lec.ch02.student.Student"*>

<constructor-arg value=*"성춘향"*/>

<constructor-arg value=*"10"*/>

<constructor-arg value=*"3학년"*/>

<constructor-arg value=*"20반"*/>

</bean>

<bean id=*"studentInfo"* class=*"com.lec.ch02.student.StudentInfo"*>

<constructor-arg>

<ref bean=*"student1"*/>

</constructor-arg>

</bean>

</beans>

**MainClass.java**

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

String configLocation = "classpath:applicationStudent.xml";

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext(configLocation);//스트링 설정파일을 파싱

StudentInfo studentInfo = ctx.getBean("studentInfo", StudentInfo.**class**); //컴포넌트 가져온다

studentInfo.getInfo();

Student student2 = ctx.getBean("student2", Student.**class**); //컴포넌트 가져온다

studentInfo.setStudent(student2);

studentInfo.getInfo();

ctx.close();

}

}

Student2

Java코드이용

studentInfo 객체

생성자() 또는 setter

student1

Xml이용

Xml에 bean으로 주입해 쓰다가 맘에 안 들면 java 코드에서 다른 bean을 주입해서 쓸 수도 있다

1. DI 사용에 따른 장점
   * 아직은 스프링의 DI사용에 따른 장점을 많이 느끼지 못할 수 있습니다. 또는 DI를 사용하니 더욱 복잡하고 시간이 더 많이 소요된다고 생각 할 수 있습니다. (oop언어를 배울 때, class 나온 후 interface가 나온다. Interface는 구현도 없고 선언만 하는데 굳이 interface를 만들까 생각하지만 interface를 만들므로 타입을 통일시킬 수 있고, 강제로 오버라이드하게 해서 작업을 통일시킬 수 있다는 여러가지 장점이 있다. 이 장점은 규모가 큰 프로젝트를 할수록 느낄 수 있게 된다)
   * 사실 작은 규모의 프로젝트에서는 스프링의 DI사용을 하는 것 보다 일반적인 방법을 사용하여 개발하는 것이 더욱 빠르고, 개발에 따른 스트레스를 줄일 수 있습니다. 하지만 규모가 어느 정도 커지고, 추후 유지보수 업무가 발생시에는 DI를 이용한 개발의 장점을 느낄 수 있습니다.
   * 스프링 DI사용에 따른 장점을 예제를 통해 살펴 봅니다. **자바파일의 수정 없이** 스프링 설정 파일만을 수정하여 부품들을 생성/조립하고 있는 예제입니다

Vehicle.java

**public** **interface** Vehicle {

**public** **void** ride(String name);

}

Bicycle.java

**public** **class** Bicycle **implements** Vehicle {

**public** **void** ride(String name) {

System.***out***.println(name + "은(는) 자전거를 10km/h 속도로 탄다");

}

}

Car.java

**public** **class** Car **implements** Vehicle {

**public** **void** ride(String name) {

System.***out***.println(name + "은(는) 자동차를 70km/h 속도로 탄다");

}

}

Airplane.java

**package** com.ch.ex;

**public** **class** Airplane **implements** Vehicle {

@Override

**public** **void** ride(String name) {

System.***out***.println(name + "은(는) 비행기를 500km/h 속도로 탄다");

}

}

TestMain.java

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

String location = "classpath:applicationVehicleCTX.xml";

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext(location);

Vehicle vh = ctx.getBean("vh",Vehicle.**class**);

vh.ride("홍길동");

ctx.close();

}

}

applicationVehicleCTX.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<!-- <bean id="vh" class="com.lec.ch02.vehicle.Bicycle"/> -->

<!-- <bean id="vh" class="com.lec.ch02.vehicle.Car"/> -->

<bean id=*"vh"* class=*"com.lec.ch02.vehicle.Airplane"*/>

</beans>

DI장점(ex1의 진화버전) ex2.

VehicleImpl.java

**import** com.lec.ch02.vehicle.Vehicle;

**import** lombok.Data;

@Data

**public** **class** VehicleImpl **implements** Vehicle {

**private** String vehicle;

**private** **int** speed;

**public** **void** ride(String name) {

System.***out***.println(name + "은(는) " + vehicle + "을(를) "+speed+ "km/h 속도로 탄다");

}

}

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<!-- <bean id="vh" class="com.lec.ch02.vehicle.Bicycle"/> -->

<!-- <bean id="vh" class="com.lec.ch02.vehicle.Car"/> -->

<!-- <bean id="vh" class="com.lec.ch02.vehicle.Airplane"/> -->

<bean id=*"vh"* class=*"com.lec.ch02.vehicle2.VehicleImpl"*>

<property name=*"vehicle"* value=*"멋진차"*/>

<property name=*"speed"* value=*"70"*/>

</bean>

</beans>

두번째소스

**public** **interface** Exam {

**int** total();

**double** avg();

}

**import** lombok.Data;

@Data

**public** **class** ExamImpl **implements** Exam {

**private** **int** kor;

**private** **int** eng;

**private** **int** mat;

**private** **int** com;

**public** **int** total() {

// **TODO** Auto-generated method stub

**return** kor+eng+mat+com;

}

**public** **double** avg() {

// **TODO** Auto-generated method stub

**return** total()/4.0;

}

}

**import** lombok.AllArgsConstructor;

@AllArgsConstructor

**public** **class** InlineExamConsole **implements** ExamConsole {

**private** Exam exam;

**public** **void** print() {

System.***out***.printf("total is %d점, avg is %f점\n", exam.total(), exam.avg());

}

}

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<bean id=*"exam"* class=*"com.lec.ch02.exam.ExamImpl"*>

<property name=*"kor"* value=*"100"*/>

<property name=*"eng"* value=*"90"*/>

<property name=*"mat"* value=*"95"*/>

<property name=*"com"* value=*"100"*/>

</bean>

<!-- <bean id="console" class="com.lec.ch02.exam.InlineExamConsole"> -->

<!-- <constructor-arg ref="exam2"></constructor-arg> -->

<!-- </bean> -->

<bean id=*"exam2"* class=*"com.lec.ch02.exam.ExamImpl2"*>

<property name=*"kor"* value=*"100"*/>

<property name=*"eng"* value=*"90"*/>

<property name=*"mat"* value=*"95"*/>

</bean>

<bean id=*"console"* class=*"com.lec.ch02.exam.GradeExamConsole"*>

<constructor-arg ref=*"exam2"*></constructor-arg>

</bean>

</beans>

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

String location = "classpath:applicationCTX5.xml";

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext(location);

ExamConsole console = ctx.getBean("console",ExamConsole.**class**);

console.print();

ctx.close();

}

}

**기존의 구현을 수정하지 않고, 시험 과목을 조정할 경우 xml과 class를 추가하여 구현하시고**