|  |  |
| --- | --- |
| Course | Advanced Software Design – CS525 |
| Assignment | Lab 11 |
| Week | 13 |
| Due | May 4, 2020 |
| Student name | Quan Hong Doan |
| Student ID | 986956 |

Online version:

<https://github.com/zithiat/asd/blob/master/labs%20%26%20assignments/Answers/Assignment_CS525_Lab11_986956.docx>

**Problem**:

Problem is the question of LMS Framework implementation from the PDF file.

**Answer**:

Source is on my GitHub

<https://github.com/zithiat/asd/tree/master/codes/code%20for%20labs/lab11/lab11>

**LabApplication**

**public** **class** LabApplication {

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

FWContext context = **new** FWContext();

context.start();

}

}

**FWContext**

**public** **class** FWContext {

**private** **static** List<Object> *objectMap* = **new** ArrayList<>();

**private** **static** List<Object> *serviceObjectMap* = **new** ArrayList<>();

**public** FWContext() {

**try** {

Reflections reflections = **new** Reflections("");

Set<Class<?>> types = reflections.getTypesAnnotatedWith(TestClass.**class**);

**for** (Class<?> cls : types) {

System.***out***.println(cls.getName());

*objectMap*.add((Object) cls.newInstance());

}

Set<Class<?>> serviceTypes = reflections.getTypesAnnotatedWith(Service.**class**);

**for** (Class<?> cls : serviceTypes) {

System.***out***.println(cls.getName());

*serviceObjectMap*.add((Object) cls.newInstance());

}

} **catch** (Exception e) {

e.printStackTrace();

}

performDI();

}

**public** **void** start() {

**try** {

**for** (Object cls : *objectMap*) {

Method before = **null**, test = **null**;

**for** (Method m : cls.getClass().getDeclaredMethods()) {

**if** (m.isAnnotationPresent(Before.**class**)) {

before = m;

}

**if** (m.isAnnotationPresent(Test.**class**)) {

test = m;

}

**if** (before != **null** & test != **null**) {

before.invoke(cls);

test.invoke(cls);

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **void** performDI() {

**try** {

**for** (Object cls : *objectMap*) {

// find annotated fields

**for** (Field f : cls.getClass().getDeclaredFields()) {

**if** (f.isAnnotationPresent(Inject.**class**)) {

// get the type of the field

Class<?> t = f.getType();

// get the object instance of this type

Object instance = getBeanOftype(t);

// do the injection

f.setAccessible(**true**);

f.set(cls, instance);

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

**private** Object getBeanOftype(Class<?> theFieldType) {

Object service = **null**;

**try** {

**for** (Object cls : *serviceObjectMap*) {

Class<?>[] interfaces = cls.getClass().getInterfaces();

**for** (Class<?> i : interfaces) {

**if** (i.getName().contentEquals(theFieldType.getName()))

service = cls;

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** service;

}

}

**MyTest**

@TestClass

**public** **class** MyTest {

@Inject

Calculator calculator;

@Before

**public** **void** init() {

System.***out***.println("perform initialization");

}

@Test

**public** **void** testMethod1() {

System.***out***.println("perform testMethod1");

*assertEquals*(calculator.add(3), 3);

*assertEquals*(calculator.add(6), 9);

}

@Test

**public** **void** testMethod2() {

System.***out***.println("perform testMethod2");

*assertEquals*(calculator.add(3), 3);

*assertEquals*(calculator.subtract(6), -1);

}

}

**Calculator** interface

**public** **interface** Calculator {

**public** **void** reset();

**public** **int** add(**int** newValue);

**public** **int** subtract(**int** newValue);

}

**CalculatorImpl**

@Service

**public** **class** CalculatorImpl **implements** Calculator {

**private** **int** calcValue = 0;

**public** **void** reset() {

calcValue = 0;

}

**public** **int** add(**int** newValue) {

calcValue = calcValue + newValue;

**return** calcValue;

}

**public** **int** subtract(**int** newValue) {

calcValue = calcValue - newValue;

**return** calcValue;

}

}

**Asserts**

**public** **class** Asserts {

**public** **static** **void** assertEquals(**int** x, **int** y) {

**if** (x != y)

System.***out***.println("Fail: result = " + x + " but expected " + y);

**else**

System.***out***.println("Pass: result = " + x + " expected " + y);

}

}

**TestClass** annotation class

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** TestClass {}

**Before** annotation class

@Retention(RetentionPolicy.***RUNTIME***)

//@Target(ElementType.TYPE)

**public** **@interface** Before {}

**Test** annotation class

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** Test {}

**Service** annotation class

@Retention(RetentionPolicy.***RUNTIME***)

@Target(ElementType.***TYPE***)

**public** **@interface** Service {}

**Inject** annotation class

@Retention(RetentionPolicy.***RUNTIME***)

@Target(ElementType.***FIELD***)

**public** **@interface** Inject {}