SPRING FRAMEWORK ASSIGNMENT

RAGHAV GUPTA

==================================

**(1) Write a program to demonstrate Tightly Coupled code.**

**package** com.example.demo;

**class** Subject {

Topic **t** = **new** Topic();

**public void** startReading()

{

**t**.understand();

}

}

**class** Topic {

**public void** understand()

{

System.***out***.println(**"Tight coupling concept"**);

}

}

**public class** Question1{

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

Subject s = **new** Subject();

s.startReading();

}

}

Here we see that, if we modify the name of understand() method in the Topic class to gotit(), then the subject class will not be able to use it because it is dependent on the Topic class. This is called a tightly coupled relationship.

(**2) Write a program to demonstrate Loosely Coupled code.**

**package** com.example.demo;

**interface** Topic

{

**void** understand();

}

**class** Topic1 **implements** Topic {

**public void** understand()

{

System.***out***.println(**"Got it"**);

}

}

**class** Topic2 **implements** Topic {

**public void** understand()

{

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

}

}

**public class** Question2 {

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

{

Topic t = **new** Topic1();

t.understand();

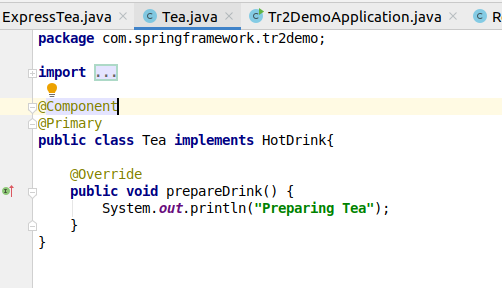
}

}

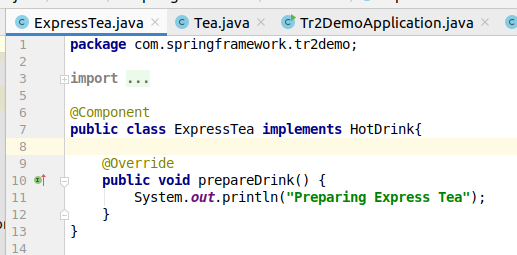
Here, we modified the previous tightly coupled code by using interfaces and now, whenever a class wants to use the Topic interface, it can use it easily by providing its own implementation and the method name in Topic interface will always be the same. There is no dependency on the Topic class now. Each class can provide their own logic irrespective of other classes. So, there is no dependency. However, many classes will use the same Topic interface, so they ARE coupled BUT loosely, due to separate implementation.

**(3) Use @Component and @Autowired annotations to in Loosely Coupled code for dependency management.**

TEA.java



ExpressTea.java



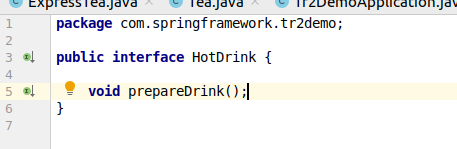
Restaurant.java



Tr2DemoApplication.java



HotDrink interface -



Here, I have made a Tea class, ExpressTea class, Restaurant class and make them beans by using the @Component annotation.

Tea and ExpressTea implement the HotDrink interface.

I use @Autowire with the hotDrink reference in the Restaurant class. It looks that if there is any bean of HotDrink type is available in the Spring, it founds 2 beans - Tea and ExpressTea. It associates it with the Tea Bean because of the @Primary annotation.

When I get the bean by ApplicationContext , we see that we didn’t create any object of the Restaurant class and Tea class, but we are able to access it and it prints the output corresponding to bean of Tead type.

**(4) Get a Spring Bean from application context and display its properties.**



I have used the getClass() method to get the class name property.

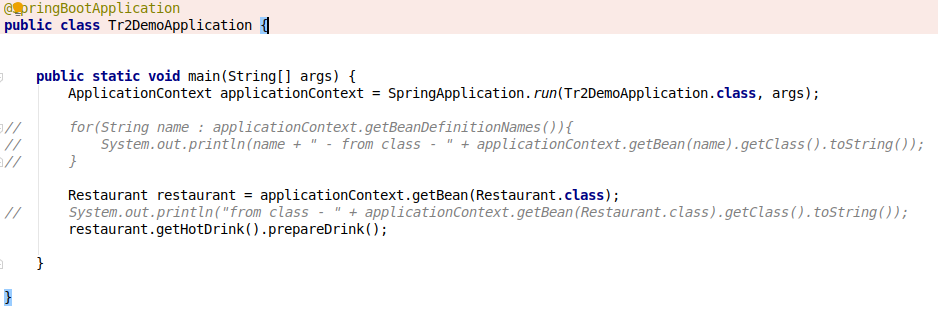
**(5) Demonstrate how you will resolve ambiguity while autowiring bean (Hint : @Primary)**

Already shown in Question 3.

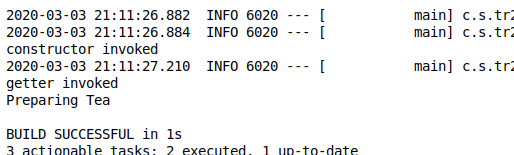
There were 2 beans available of HotDrink type and we make Tea to be primary.

**(6) Perform Constructor Injection in a Spring Bean**

****

****

**OUTPUT-**

****