Spring Intro

19 Temmuz 2018 Perşembe 11:40

- Spring is best practice of Design Patterns. Clean Code.
- Pojo model is used by Spring, no need for implementing any framework spesific interface or extends class
- Pojo model is based on simple object.
- Here are important concept to create simple pojo model.

Dependency Injection Concept
Classes in an application work together to achieve a task. To make successfull collobration , classes must be aware of their interfaces instead of implementation. Decoupling application objects from each other. DI promotes program to interface.

Problem:
If a class has relationship with another class implementation instead of interface, this class will violate single responsibility. It will has also object creation behaviour. In case of implementation of another class is changed we will change the class. It also violated Open/Close prinsible. Object creation will scatering in all system. This is a really big problem in enterprise application.

If a class uses another class for functioning, these classes are dependent.

In dependency inversion principle; we have use classes via abstraction(interface). Code should upon interfaces not the implementation
Normally;
Class A->Class B directly
In this principle Class A->Interface->Class B

Dependecy injection is just pushed dependecy outside the class. It means you shouldnt use new to instante class B inside class A. Instead use constructor or setter to inject dependecy.

Manually injecting dependecy in enterprise app is diffucult. That's why we need container.
With IOC container we will resolve dependecy via container. When we need chanhe dependecy, we change container parameters.

Dependency Injection step by step:

1- Giving dependency via constructor or setter: this is hardcoded injecting dependency. If we give dependency via constructor hardcoded, given class will have extra behaviour. This violates Single Responsibility prinsiple.
2- Giving dependency via Factory pattern. In this case, when the number of dependency increased factory class will be complex. In case of Factory pattern, we need many factory class for each object. It is not optimal solution
3- IOC container: This is the best solution. Dependencies are defined in xml or we use annotations and then container will use reflection and hashMap to create single instance for every defined item by default. On program startup all dependencies are loaded by the container

Lets little talk about **AOP**

AOP is simply modularization of cross cutting concerns.
What is cross cutting concern? Cross cutting concern is simply an additional functionalty for example: logging, security, exception handling, performance monitoring

There are two problems are solved by AOP;

1- Code Tangling: it defines as simply mixed salad. If a method has cross cutting concern as addition to business logic, this method will violates Single Responsbility.
2- Code Scattering: If cross cutting concern is scattered around the system, system has code scattering problem.

Solution is Aspect Oriented programming. Define aspect, weave it.
It uses Proxy pattern

Lets little talk about **Template** in Spring
For example for jdbc connection we have to take a connection, execute a statement and close the connection. There are too much things repeated. This is know as Boiler plate code. Out actual business logic is giving a SQL statement to execute and Mapping.
Spring has JDBCTemplate, RestTemplate..etc it decreases code we write.