Name: Andi Dwi Saputro

Date: April 7th, 2020

Topik: 1. Spring DI (*Dependency Injection*)

2. Spring IoC (*Inversion of Control) Container*

3. Bean

4. Contoh-contoh implementasi DI dan Bean

5. Kesimpulan

1. Apa itu *Dependency Injection*?

   Penjelasan:

   Dependency Injection merupakan salah satu fungsi utama yang disediakan oleh framework Spring. Modul utama (*Spring-Core*) pada Spring framework bertanggungjawab penuh terhadap proses injecting/penyisipan *dependencies* (kebutuhan-kebutuhan dari suatu class) melalui berbagai macam cara, baik melalui *Constructor* ataupun *Setter methods*. Singkatnya, *Dependency Injection* merupakan sebuah proses dimana *object* mendeklarasikan *dependencies* yang dibutuhkan lalu disisipkan/*inject* melalui ***Contructor, Setter,*** ataupun ***Interface****.*

   Selain itu pada Spring framework juga dapat melakukan proses DI dengan memanfaatkan ***Beans*** yakni dengan menggunakan *annotations* seperti **@Bean**, ***@Autowired, @Controller, @Component, @Service, @Repository*** dan lain sebagainya pada class-class yang membutuhkan proses *injection dependencies.*

2. Apa itu *Spring Inversion of Control Container* ?

   Penjelasan:

   Spring IoC merupakan sebuah *container* atau wadah sebuah aplikasi dimana wadah ini akan menampung *Beans* yang akan digunakan dalam aplikasi tersebut menggunakan Spring framework. Spring IoC ini tidak hanya menampung objek saja, tapi juga bertugas untuk menginstansiasi, meng-*configure* dan merangkai *beans* yang ada sesuai dengan konfigurasi yang digunakan. Spring Container pada spring framework sering disebut sebagai ***Application Context.***

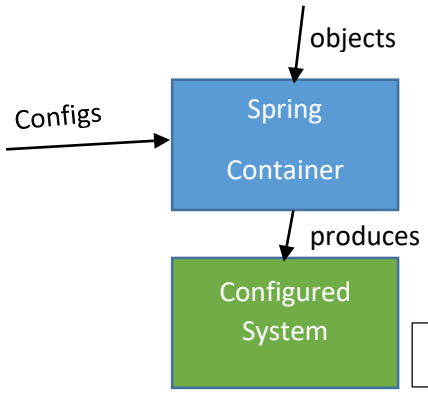

Fig1: Spring IoC Container

3. Apa itu *Bean ?*

Penjelasan:

Bean merupakan sebuah object/instance yang digunakan dalam aplikasi spring framework dan di manage oleh Spring IoC *Container* dengan adanya penggunaan *annotations* seperti ***@Component, @Controller, @Service, @Bean, @Repository***, dsb pada class Bean tersebut.

Secara *default* Bean pada spring framework memiliki *scope design pattern singleton* dimana setiap bean hanya memiliki 1 *instance* per *container*. Hal ini menandakan bahwa penggunaan memori pada aplikasi dengan ***singleton pattern*** akan lebih sedikit. Untuk membuat *scope* lainnya seperti *prototype, session, request*, dsb perlu digunakan anotasi ***@Scope*** pada bean class tersebut. Tanpa anotasi ***@Scope*** secara *default* patternnya adalah ***singleton***.

4. Contoh-Contoh Implementasi ***DI, IoC, Bean*** pada project Spring MVC NusaBank

   a. **Contoh deklarasi Bean class @Controller (NasabahController) dengan injecting melalui Setter.**

   File: com.nusabank.app.controller.**NasabahController.java**

```
1  package com.nusabank.app.controller;
2
3  import org.springframework.beans.factory.annotation.Autowired;
14
15 @Controller
16 @RequestMapping("/")
17 public class NasabahController {
18
19     private NasabahService nasabahService;
20
21     @Autowired(required=true)
22     @Qualifier(value="nasabahService")
23     public void setNasabahService(NasabahService ns) {
24         this.nasabahService=ns;
25     }
```

   File: **dispatcher-servlet.xml**

```
82     <context:component-scan
83         base-package="com.nusabank.app" />
```

   Penjelasan:

   Pada class NasabahController menggunakan annotation @Controller yang mengindikasikan pada container bahwa class bean tersebut merupakan sebuah Controller.

Proses Dependency Injection dilakukan menggunakan metode SDI atau Setter Dependency Injection (line: 19-25). Dimana class NasabahController membutuhkan (dependent) semua fields/properties dan fungsi dari NasabahService class untuk melakukan proses Transaction.

Proses DI tersebut melibatkan penggunaan annotation @Autowired dan @Qualifier dimana dengan anotasi tersebut dependency object yang dibutuhkan akan langsung terhubung/di inject ke container dan akan discan otomatis oleh spring menggunakan component-scan pada config xml sesuai dengan tipe datanya, jika field tidak sesuai atau class tidak ditemukan pada base package maka akan error saat initializing context/container.

b. Contoh deklarasi Bean class @Entity Nasabah dengan injecting melalui Setter methods dari class @Entity Rekening.

File: com.nusabank.app.model.**Nasabah.java**

```
1  package com.nusabank.app.model;
2
3  import javax.persistence.Column;
4  import javax.persistence.Entity;
5  import javax.persistence.GeneratedValue;
6  import javax.persistence.GenerationType;
7  import javax.persistence.Id;
8  import javax.persistence.OneToOne;
9  import javax.persistence.Table;
10
11 import org.hibernate.annotations.Cascade;
12
13 @Entity
14 @Table(name="nasabah")
15 public class Nasabah {
```

File: com.nusabank.app.model.**Nasabah.java**

```
63     @OneToOne(mappedBy = "nasabah")
64     @Cascade(value = org.hibernate.annotations.CascadeType.ALL)
65     private Rekening rekening;
```

File: com.nusabank.app.model.**Nasabah.java**

```
207    public Rekening getRekening() {
208        return rekening;
209    }
210
211    public void setRekening(Rekening rekening) {
212        this.rekening = rekening;
213    }
```

Penjelasan:

Entity Nasabah membutuhkan (*dependent*) seluruh field pada Entity Rekening untuk melakukan join table dengan hibernate dan spring harus melakukan pemetaan berdasarkan

relasi entitas, disini saya menggunakan relasi ***@OneToOne*** karena masing-masing nasabah memiliki rekeningnya masing-masing olehsebab itu pula Entitas nasabah melakukan *dependency injection* dengan metode SDI dari entitas Rekening.

c. Contoh deklarasi Bean class @Service RekeningServiceImpl dengan injecting melalui interface RekeningDAO.

File: com.nusabank.app.service.**RekeningServiceImpl.java**

```
1  package com.nusabank.app.service;
2
3  import java.util.List;
10
11 @Service
12 public class RekeningServiceImpl implements RekeningService {
13
14     private RekeningDAO rekeningDAO;
15
16     public void setRekeningDAO(RekeningDAO rekeningDAO) {
17         this.rekeningDAO = rekeningDAO;
18     }
```

File: com.nusabank.app.dao.**RekeningDAO**

```
1  package com.nusabank.app.dao;
2
3  import java.util.List;
4
5  import com.nusabank.app.model.Rekening;
6
7  public interface RekeningDAO {
8      public void addRekening(Rekening r);
9      public void updateRekening(Rekening r);
10     public List<Rekening> listRekening();
11     public Rekening getRekeningById(int id);
12     public void removeRekening(int id);
13 }
14
```

Penjelasan:

Bean RekeningServiceImpl dideklarasikan menggunakan anotasi ***@Service*** yang menandakan bahwa class tersebut adalah sebuah service. Class tersebut melakukan *dependency injection* dari interface Rekening DAO. Sehingga class tersebut dapat menginitialize pada container dan menggunakan seluruh methods yang ada pada RekeningDAO.

d. Contoh deklarasi Bean dataSource pada config dispatcher-servlet.xml / FrontController untuk injecting konfigurasi DataSource.

File: **dispatcher-servlet.xml**

```
31    <beans:bean id="dataSource"
32        class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
33        <beans:property name="driverClassName"
34            value="com.mysql.jdbc.Driver" />
35        <beans:property name="url"
36            value="jdbc:mysql://localhost:3306/db_nusa" />
37        <beans:property name="username" value="root" />
38        <beans:property name="password" value="" />
39    </beans:bean>
```

Penjelasan:

Dengan mendeklarasikan bean *data source* untuk konfigurasi koneksi database dengan aplikasi spring framework kita tidak perlu lagi membuat atau memanggil *object data source* yang sama pada class-class yang membutuhkan *object data source* tersebut, karena semua sudah di*handle* oleh spring IoC.

5. Kesimpulan

Spring IoC / Spring Framework pada umumnya dapat mengatasi ketergantungan berlebih antar class dengan menerapkan ***Dependency Injection***, yang mana dengan penerapan ini dapat membuat code lebih mudah untuk di ***reuse*** dan ***test. Loose coupling**** antar class bisa dengan mudah dilakukan dengan men-*define* ***interfaces*** untuk beberapa fungsionalitas yang bersangkutan dan ***injector*** spring IoC akan meng-*instansiate* objek-objek bean yang diperlukan untuk di implementasikan. Tugas-tugas instansiasi objek dilakukan oleh *container / app context* berdasarkan konfigurasi-konfigurasi yang telah spesifikasikan oleh developer.

***Loose coupling**: independensi, atau proses mengurangi ketergantungan antar class.

Contoh: implementasi interface RekeningDAO pada RekeningServiceImpl