**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.*

1. 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*

objects

produces

Configs

Configured System

Spring

Container

1. Apa itu *Bean ?*

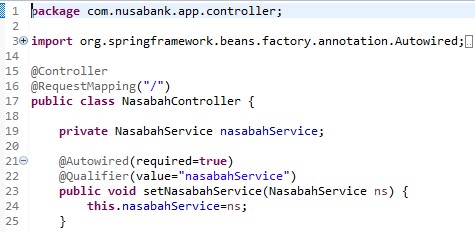
**Penjelasan:**

*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***.

1. Contoh-Contoh Implementasi ***DI, IoC, Bean*** pada project Spring MVC NusaBank
2. **Contoh deklarasi Bean class @Controller (NasabahController) dengan injecting melalui Setter.**

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



File: **dispatcher-servlet.xml**



**Penjelasan:**

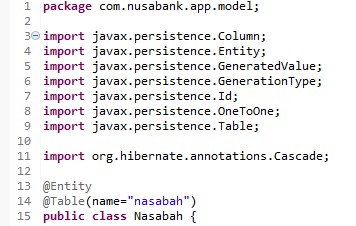
Pada class NasabahController menggunakan annotation @Controller yang mengindikasikan pada container bahwa class bean tersebut merupakan sebuah Controller.

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.

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

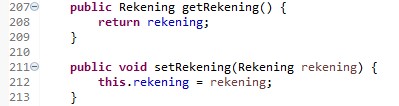
File: com.nusabank.app.model.**Nasabah.java**



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



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



**Penjelasan:**

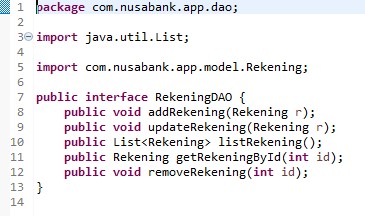
Entity Nasabah membutuhkan (*dependent*) seluruh field pada Entity Rekening untuk melakukan join table dengan hibernate dan spring harus melakukan pemetaan berdasarkan relasi entitas, disini saya menggunakan relasi ***@OneToOne*** karena masing-masing nasabah memiliki rekeningnya masing-masing olehsebab itu pula Entitas nasabah melakukan *dependency injection* dengan metode SDI dari entitas Rekening.

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

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



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

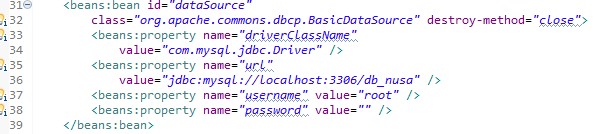


**Penjelasan:**

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.

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

File: **dispatcher-servlet.xml**



**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.

1. **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