Pemrograman Framework Java

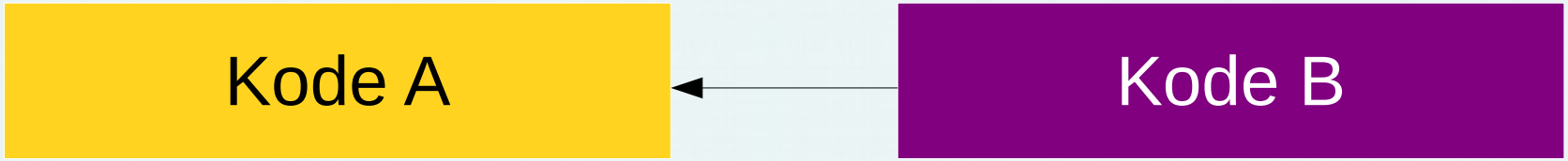
Pertemuan 3

Injeksi Ketergantungan

- Definisi Ketergantungan
- Injeksi Ketergantungan
- Injeksi Setter dan Konstruktor
- Autowiring dengan XML
- Autowiring dengan Annotation

Definisi Ketergantungan

- Ketergantungan bisa disebut di mana kode A memerlukan kode B untuk bekerja dengan baik. Jika ketergantungan tidak terpenuhi maka kode A tidak akan berjalan baik.



Ketergantungan hanya bersifat satu arah saja. Dan bisa juga disebut dengan Coupling

Baik & Buruk Ketergantungan

- Dengan ketergantungan kita bisa menambah fitur tanpa harus menulis ulang kode yang dibutuhkan, karena sudah tersedia.
- Namun dibalik manfaat ketergantungan, ada hal yang perlu diperhatikan.
  - Tight Coupling
  - Loose Coupling

Analogi Coupling

- Coupling ini bisa diibaratkan dengan penggunaan battery pada smartphone.
  - Battery di perangkat iPad/iPod merupakan battery yang sudah di solder (tight coupling). Jadi untuk menggantinya harus satu perangkatnya sekaligus
  - Berbeda dengan smartphone dengan battery bisa dilepas (loose coupling). Jadi untuk mengganti battery nya bisa diganti tanpa mengganti semua perangkatnya

Analogi Secara Kode

```
public class CalendarReader {
    public List readCalendarEvents(File calendarEventFile){
        //open InputStream from File and read calendar events.
    }
}
```

```
public class CalendarReader {
    public List readCalendarEvents(InputStream calendarEventFile)
{
        //read calendar events from InputStream
    }
}
```

Jenis Ketergantungan

- Ketergantungan Kelas (Class)
- Ketergantungan Interface
- Ketergantungan Metode

Ilustrasi 1

```
public byte[] readFileContents(String fileName){
    //open the file and return the contents as a byte array.
}
```

Kelas ini bergantung pada String, lihat parameternya

Ilustrasi 2

```
public byte[] readFileContents(CharSequence fileName){
    //open the file and return the contents as a byte array.
}
```

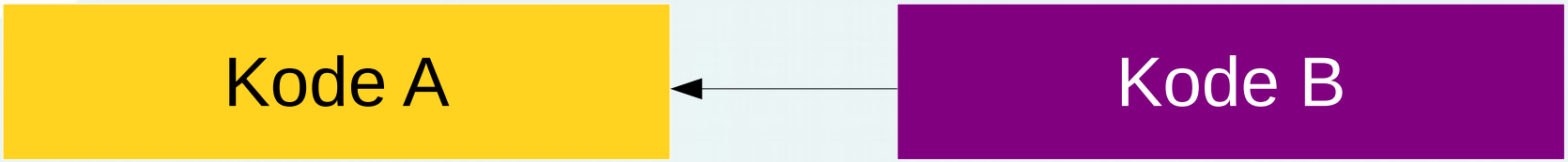
Jika dilihat di bagian parameter, metode ini bergantu pada CharSequence yang merupakan Interface standar milik Java

Ilustrasi 3

```
public byte[] readFileContents(Object fileNameContainer){

    Method method   = fileNameContainer
                        .getClass()
                        .getMethod("getFileName", null);

    String fileName = method.invoke(fileNameContainer,
null);

    //open the file and return the contents as a byte array.
}
```

Metode ini bergantung dengan metode lain dengan nama getFileName

Ketergantungan Langsung dan Tidak Langsung



Kode A memiliki ketergantungan langsung dengan Kode B

Kode A memiliki ketergantungan tidak langsung dengan Kode C

Injeksi Ketergantungan

- Injeksi Ketergantungan atau Dependency Injection merupakan suatu cara agar ketergantungan tidak dibuat secara internal.
- Ketergantungan ini akan diatur oleh Framework berdasarkan dependency graph framework itu sendiri
- Kelebihan dari injeksi ini adalah kemudahan dalam pengujian

Jenis-jenis Injeksi Ketergantungan

- Injeksi Konstruktor
  - Jenis ini memiliki layanan injeksi ketergantungan yang dilakukan melalui konstruktor klien kelas
- Injeksi Property
  - Jenis ini melakukan injeksi melalui file property dari klien kelas (Setter Injection)
- Injeksi Metode
  - Jenis ini melakukan injeksi dengan melakukan implementasi interface yang mendeklarasikan metode penyuplai ketergantungan

Contoh Construtor Injection

```
public class CustomerService
{
    CustomerBusinessLogic _customerBL;

    public CustomerService()
    {
        _customerBL = new CustomerBusinessLogic(new
CustomerDataAccess());
    }

    public string GetCustomerName(int id) {
        return _customerBL.GetCustomerName(id);
    }
}
```

Contoh Property Injection

```
public class CustomerBusinessLogic {
   ......
    public ICustomerDataAccess DataAccess { get; set; }
}

public class CustomerService {
    CustomerBusinessLogic _customerBL;

    public CustomerService()
    {
        _customerBL = new CustomerBusinessLogic();
        _customerBL.DataAccess = new CustomerDataAccess();
    }

    public string GetCustomerName(int id) {
        return _customerBL.GetCustomerName(id);
    }
}
```

Contoh Methode Injection

```
nterface IdataAccessDependency {
    void SetDependency(ICustomerDataAccess customerDataAccess);
}

public class CustomerService {
    CustomerBusinessLogic _customerBL;

    public CustomerService()
    {
        _customerBL = new CustomerBusinessLogic();
        ((IDataAccessDependency)_customerBL).SetDependency(new
CustomerDataAccess());
    }
}
```

Autowiring dengan XML

- Fitur dari Spring Framework yang memungkinkan Dependency Injection dilakukan secara Implisit menggunakan setter atau constructor
- Implisit memiliki arti tidak terlihat
- Injeksi dilakukan dengan cara mendefinisikan Dependency nya di file XML nya
- Autowiring bisa dilakukan dengan hanya menambahkan tag autowire

4 Jenis Autowiring

- autowire=”byName”
- autowire=”byType”
- autowire=”constructor”
- autowire=”no” - Default

Contoh Default

```
<bean id=”theFIrstTraveler” class=”Traveler”>
        <constructor-args ref="car">
        </constructor-args>

        <property name="origin" value="Jakarta"/>
        <property name="destination value="Surabaya"/>
</bean>

<bean id="theSecondTraveler" class "Traveler">
        <property name="car" ref="car"/>
        <property name="origin" value="Jakarta"/>
        <property name="destination value="Surabaya"/>
</bean>
```

Contoh byName

```
<!-- byName example -->
    <bean id="application"
class="com.websystique.spring.domain.Application"
autowire="byName"/>

    <bean id="applicationUser"
class="com.websystique.spring.domain.ApplicationUser" >
        <property name="name" value="superUser"/>
    </bean>
```

Contoh byType

```
<!-- byType example -->
    <bean id="employee"
class="com.websystique.spring.domain.Employee"
autowire="byType"/>

    <bean id="employeeAddress"
class="com.websystique.spring.domain.EmployeeAddress"
>
        <property name="street" value="112/223,SantaVila"/>
        <property name="city" value="Nebraska"/>
    </bean>
```

Contoh constructor

```
    <!-- constructor example -->
    <bean id="performer"
class="com.websystique.spring.domain.Performer"
autowire="constructor"/>

    <bean id="instrument"
class="com.websystique.spring.domain.Instrument" >
        <property name="name" value="PIANO"/>
    </bean>
```

Autowiring dengan Annotation

- Autowiring ini dilakukan dengan cara mengetikkan annotation di kode Java
- Di awali dengan tanda @ kode tersebut disebut dengan annotation
- Contoh
  - @SuppressWarnings
  - @Override
  - dll

Sebelum @Autowired

- Header dari file Beans.xml harus dimodifikasi agar mendukung annotation
- Bisa dilakukan dengan 2 cara
  - <context:annotation-config/>
  - <bean class=”org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor”/>

Autowiring

- Anotasi Autowiring bisa dilakukan langsung dengan contoh seperti ini

```
public class Traveler {
        @Autowired
        private Vehicle car;
        ...
}
```

Lainnya

```
public class Traveler {
        private Vehicle car;
        ...
        @Autowired
        public Traveler(Vehicle car) {
        ...
        }
}

public class Traveler {
        @Autowired
        public void setCar(Vehicle car) {
                this.car = car;
        }
}
```

Lanjutan

- Autowired akan melakukan pengecekan untuk memastikan referensi bean ditemukan.
- Untuk menghindari munculnya pesan kesalahan, maka dependency checking bisa di set menjadi
- @Autowired(required=false)
- Field tidak akan diinjek namun tetap bernilai null

Lanjutan

- Jika ditemukan lebih dari satu referensi bean dengan tipe atau class yang sama, maka programmer dapat memerintahkan Spring untuk memilih bean yang digunakan
- Anotasi @Qualifier dapat digunakan untuk memilih bean yang diinginkan
- @Qualifier(“theOtherCar”)

Kesimpulan

- Dependency Injection Eksplisit
  - Konstruktor
  - Setter
- Implisit
  - Autowiring XML: byNam, byType,constructor
  - Autowiring dengan anotasi: field, constructor, atau setter