

Spring Framework (Part 2)

Gold - Chapter 4 - Topic 2

Selamat datang di **Chapter 4 Topic 2**
online course **Back End Java** dari
Binar Academy!

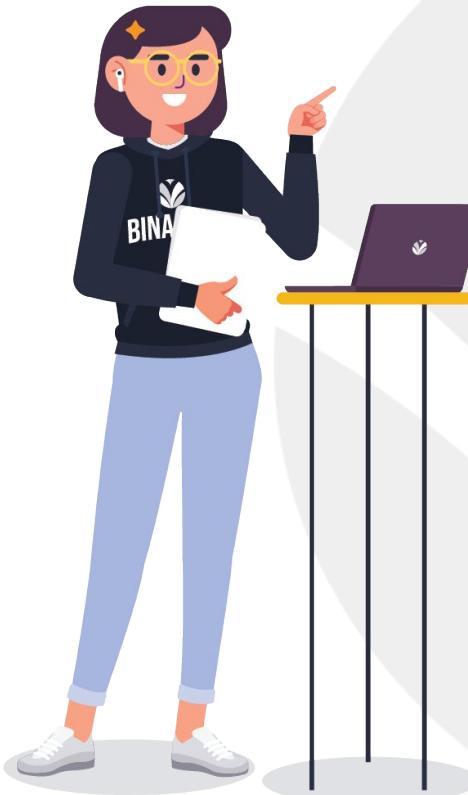


Selamat datang, Binarian! ☺

Gimana belajar kamu di topik sebelumnya? Masih seputar Spring Framework, kali ini kita bakal lanjutin pembahasan **Spring Framework di part 2.**

Pokoknya, mulai dari konsep Bean Containers, Spring Bean Lifecycle, Konsep Autowire, sampai pom.xml dan configuration-nya, kita bakal belajar semua dari awal!

Yuk, langsung aja kita kepoin~



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Konsep dan Penggunaan Spring Batch
- Konfigurasi dan Implementasi Spring Batch
- Mengenal Bean Containers
- Tahapan Spring Bean Lifecycle
- Dependency Injection dan Inversion of Control
- Annotation @Bean, @Component, @PostConstruct, @Service dan @Repository
- Konsep autowire
- Konfigurasi pada Spring Framework berupa pom.xml dan configuration
- Pengantar Spring AOP
- Konfigurasi Spring AOP



Binarian, nggak bisa dipungkiri bahwa terkadang kita perlu mengelola data yang besar.

Bukan tanpa solusi, tentunya udah ada jalan keluar untuk mengatasi situasi itu.

Caranya, kamu bisa pakai **Framework Spring Batch** ✨



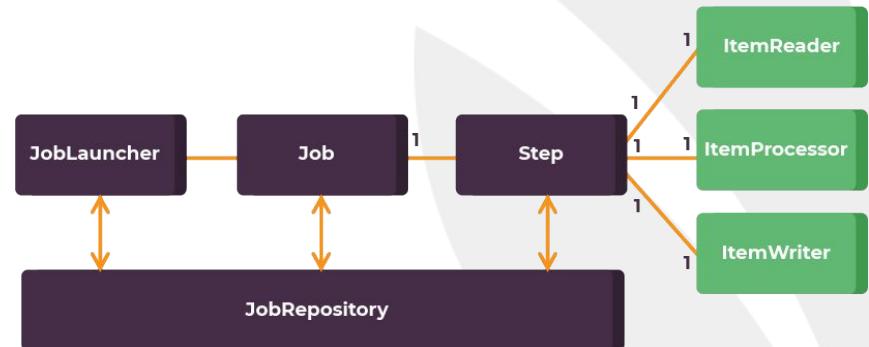
Apa itu Spring Batch?

Spring Batch merupakan salah satu framework dari Spring yang bisa mengelola data yang sangat besar sampai bisa menghandle kebutuhan aplikasi berskala enterprise.

Secara lebih spesifik, framework ini akan melakukan proses batch yang mengeksekusi serangkaian job. Di mana job tersebut terdiri dari banyak step.

Setiap step terdiri dari read-process-write atau single operation yang alurnya kayak gambar disamping ini.

SPRING BATCH ARCHITECTURE



Alright. So far kita tahu nih kalau Spring Batch bisa mendukung pemrosesan data yang besar. Tapi kamu kepo nggak sih kenapa bisa gitu?

Jawabannya karena Spring Batch punya fungsi-fungsi kayak yang ada di bawah ini:

- Logging/tracing
- Transaction management
- Job processing statistics
- Job restart, skip
- Resource management

Gimana? keren kan?

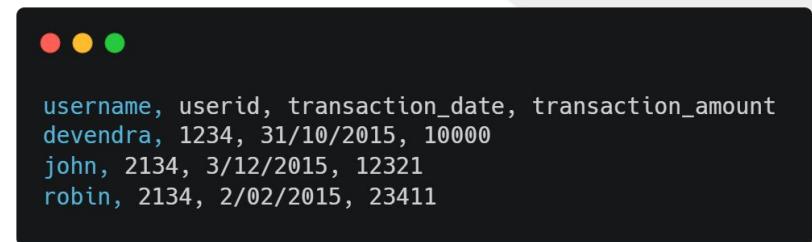


Spring Batch Use Case

Lanjuut~ Supaya pemahaman kita tentang Spring Batch makin ajib, kita coba pakai case ya.

Kasus use case yang akan kita tangani di sini adalah memigrasikan beberapa data transaksi keuangan dari CSV ke XML.

Etss.. ada catatan. File input memiliki struktur yang sangat sederhana. Berisi transaksi per baris yang terdiri dari nama pengguna, id pengguna, tanggal transaksi dan jumlah.



A screenshot of a terminal window with a dark background. At the top, there are three colored dots: red, yellow, and green. Below them, the CSV data is displayed:

```
username, userid, transaction_date, transaction_amount
devendra, 1234, 31/10/2015, 10000
john, 2134, 3/12/2015, 12321
robin, 2134, 2/02/2015, 23411
```

Guys! Pengenalan dan kegunaan Spring Batch kayaknya udah cukup kebayang, ya.

Abis ini kita akan lebih detail ngomongin **Konfigurasi dan Implementasi Spring Batch**. Let's go!



Spring Batch Configuration

Buat konfigurasinya akan kayak di bawah ini, nih. Kamu boleh coba intip-intip duluu~

```
<!-->
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="org.sqlite.JDBC" />
    <property name="url" value="jdbc:sqlite:repository.sqlite" />
    <property name="username" value="" />
    <property name="password" value="" />
</bean>

<!-- create job-meta tables automatically -->
<jdbc:initialize-database data-source="dataSource">
    <jdbc:script
        location="org/springframework/batch/core/schema-drop-sqlite.sql" />
    <jdbc:script location="org/springframework/batch/core/schema-sqlite.sql" />
</jdbc:initialize-database>

<bean id="jobRepository"
      class="org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="transactionManager" ref="transactionManager" />
    <property name="databaseType" value="sqlite" />
</bean>

<bean id="transactionManager" class=
      "org.springframework.batch.support.transaction.ResourcelessTransactionManager" />

<bean id="jobLauncher"
      class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
    <property name="jobRepository" ref="jobRepository" />
</bean>
```

Masih seputar Spring Batch Configuration, cara konfigurasinya nggak cuma satu aja loh!

Ada juga contoh konfigurasi dengan file yang berbeda. Cara ini masih pada satu rangkaian proses, kok. Buat source code-nya bisa kamu lihat [di sini](#) ya



Lanjutin, ya. Setelah file-nya selesai kita konfigurasi, selanjutnya adalah melakukan implementasi dari Spring Batch. Kamu bisa cek contoh implementasinya pada gambar di bawah ini. Mangga~

```
● ● ●

public class App {
    public static void main(String[] args) {
        // Spring Java config
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();
        context.register(SpringConfig.class);
        context.register(SpringBatchConfig.class);
        context.refresh();

        JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");
        Job job = (Job) context.getBean("firstBatchJob");
        System.out.println("Starting the batch job");
        try {
            JobExecution execution = jobLauncher.run(job, new JobParameters());
            System.out.println("Job Status : " + execution.getStatus());
            System.out.println("Job completed");
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Job failed");
        }
    }
}
```

Kayaknya udah pada jago, nih. Kalau gitu saatnya kita move on ke materi baru yaitu **Bean Containers**.

Eh.. materi ini sempet kita spoil di topik sebelumnya, masih inget?

Sekarang saatnya kita cari tahu!



“Sekarang jelasin dong, Bean Containers itu apa?!”

Okey fine, jadi Bean Containers itu terdiri dari dua kata, yaitu Bean dan Containers.

Bean merupakan object dari sebuah aplikasi. Lebih jelasnya, yaitu object dari sebuah class.

Sedangkan **containers bisa diartikan sebagai wadah**.



Jadi...

Bisa dibilang kalau seluruh bean yang ada di dalam aplikasi, diletakkan di dalam sebuah wadah (container).

Di sini, bean saling berinteraksi dengan yang lainnya. Semua interaksi bean dikelola oleh container milik Spring.



Proses pembuatan bean, hingga proses penghancuran dari bean diatur oleh **Spring container**.

Proses dari pembuatan dan penghancuran dari bean ini disebut dengan **Spring beans lifecycle** yang bakal kita bahas pada subtopic berikutnya.

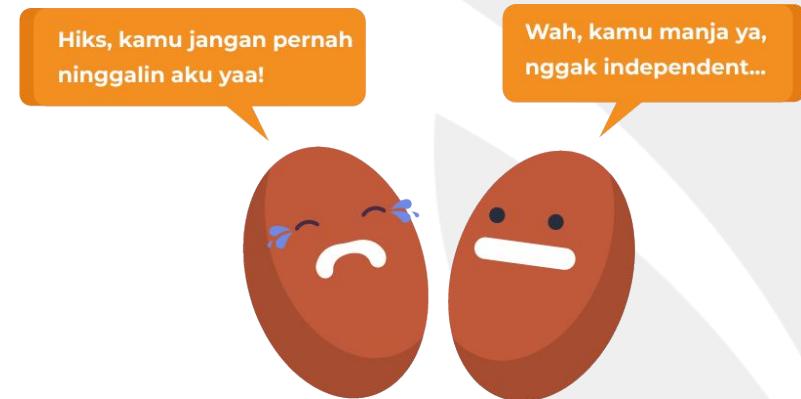
Sabar dulu yaaa~



Kita balik lagi sama interaksi Bean yang tadi, ya~

Interaksi antara bean tersebut nantinya bakal kita kenal sebagai dependency injection.

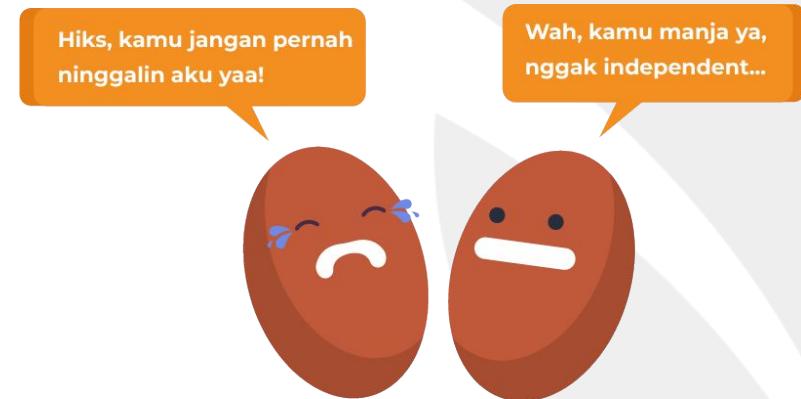
Bean saling membutuhkan antara satu dengan yang lain. **Di mana bean yang dibutuhkan oleh Bean lain disebut sebagai dependency.**



“Coba dong kasih contoh, Bean Container itu kayak gimana!”

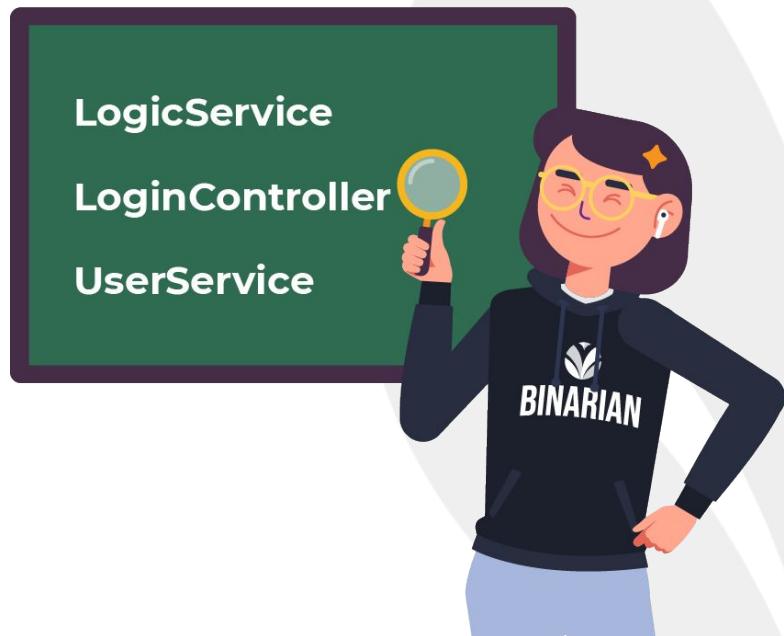
Okey, misalnya ada sebuah class yang berisi business logic dari login, kita namakan LogicService.

Selanjutnya, untuk menghandle proses login, ada class untuk membuat endpoint dari proses login yaitu LoginController.



Oh iya, ada class lain juga yang berisi business logic dari penambahan dan penghapusan user yang bernama UserService.

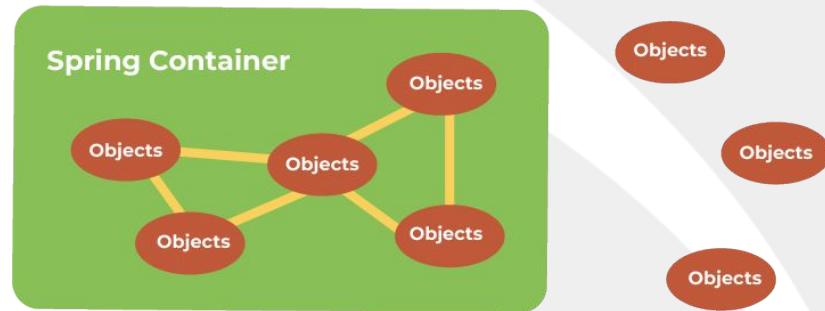
Selain itu, ada juga class yang mengatur endpoint untuk mengatur user bernama UserController.



Kita tau ada banyak class nih di sini. Pada Spring Boot, supaya class-class tersebut bisa dideteksi sebagai sebuah bean, maka **class tersebut harus menggunakan beberapa annotation**.

Annotation-nya apa? Nah, annotation-annotation ini bakal kita pelajari nanti.

Tapi sebelum ngobrolin itu, kita harus tahu bahwa ketika object udah dikenali sebagai sebuah bean, maka bean bakal diletakkan di dalam sebuah container.



Siapa yang masih inget kalau Spring Bean bisa melalui proses pembuatan sampai penghancuran?

Kita sebutnya sebagai lifecycle.

Biar ada gambaran lebih jauh, kita langsung bahas Spring Bean lifecycle. Yuk!



Bukan cuma masak mie aja yang punya tahapan, Spring Bean juga sama!

Yap, seperti namanya, Spring Bean lifecycle merupakan tahapan pembuatan (creation) bean sampai terjadinya penghancuran (destruction) bean.

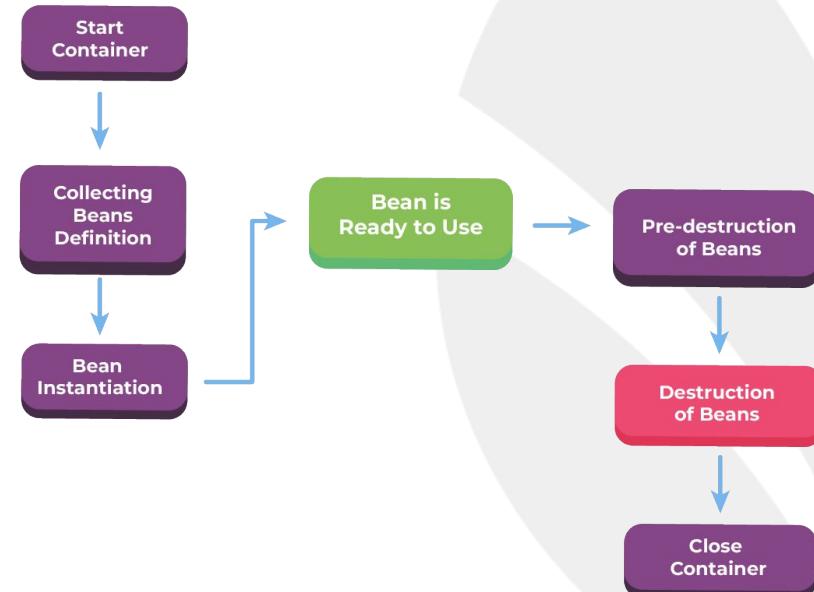
Kalau tahu lifecycle ini, kita bisa mengeksekusi code yang diperlukan di tahap-tahap tertentu.



“Emang lifecycle-nya gimana?”

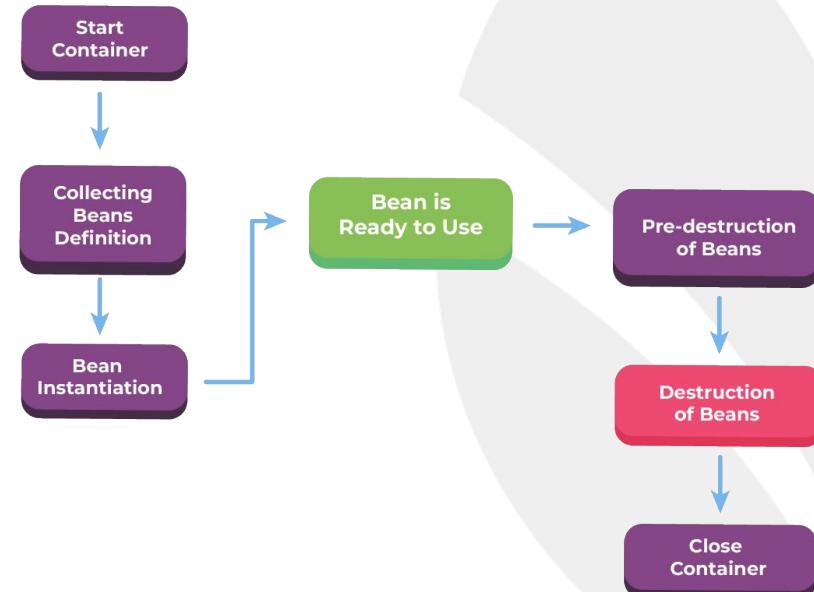
Lifecycle-nya kita bagi jadi dua bagian, ya. Berikut adalah lifecycle dari Bean:

1. Pertama, sebuah bean bakal memulai container-nya

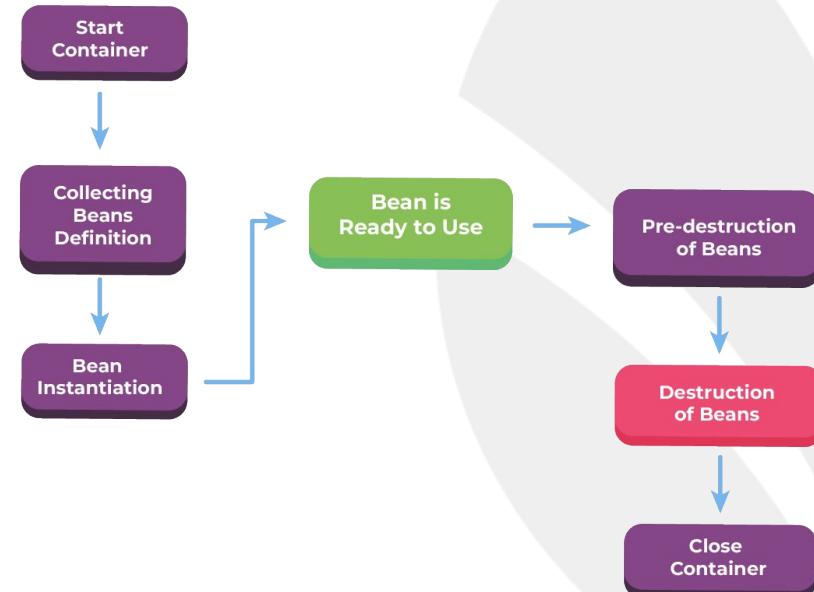


2. Tahap yang kedua, Bean yang mau dibuat akan dikumpulkan. Bean bisa berasal dari class apa aja, bisa bean singleton atau bukan. Setelah itu, interaksi satu dengan yang lainnya bakal dikumpulkan juga.

Pada tahap ini, kita bisa mengeksekusi method yang dibutuhkan melalui annotation `@PostConstruct` serta seluruh konfigurasi yang ada di properties bakal dikumpulkan (di-load).



3. Tahap yang ketiga, yaitu proses instance dari Bean. Setelah tahap ini, Bean udah bisa digunakan.



Itu dia lifecycle Spring Bean yang menjelaskan proses pembuatannya dulu.

Yang selanjutnya enih ada kaitannya sama materi dependency.

Yepp, kita akan ngobrolin tentang **Dependency Injection and Inversion of Control**.



Untuk menggambarkan fitur dependency Injection dan Inversion of control, kita pakai contoh aja ya~

Misalnya nih, ada sebuah class yang isinya business logic pembayaran suatu barang yang bernama PaymentService. Kemudian, di dalam PaymentService tersebut kita butuh pengiriman email dan pembuatan invoice yang ada di class lain yaitu EmailService dan InvoiceService.





Kalau udah gitu, berarti EmailService dan InvoiceService ini bakal di-inject sebagai sebuah dependency ke dalam PaymentService karena PaymentService butuh kedua class tersebut. Nantinya, container pada Spring ini bakal mengenal class tersebut dalam bentuk bean.

Interaksi antara bean satu dengan lainnya inilah yang disebut dengan dependency injection yang merupakan salah satu fitur dari Spring.

“Iho kok nyambung ke Spring lagi sih?”

Iya dong, kan materinya masih bahas tentang spring~

Fitur utama dari Spring Inversion of Control adalah mendefinisikan interaksi-interaksi terlebih dahulu untuk selanjutnya object-object tersebut dilakukan instance.

Kalau kita pakai Spring versi lama, maka semua interaksi harus didefinisikan dulu di xml configuration file, misalnya beans.xml.



Tapi nggak usah khawatir, sobat.

Kalau kita pakai Spring Boot, secara default developer cuma perlu **menggunakan annotation tertentu** untuk mendefinisikan dependency-dependency yang dibutuhkan dari sebuah class.

Jadi lebih praktis dengan adanya annotation ini, deh~



Okey! Dari tadi kan udah nytinggung annotation terus nih.

Berarti sekarang udah waktunya buat belajar tentang **Annotation @Bean, @Component, @PostConstruct, @Service dan @Repository**

Udah siaap? let's go~



Yuk kita kupas tuntas annotation yang sering dipakai buat meng-utilisasi fitur-fitur dari Spring!

Berikut adalah annotation-nya~

- **@Bean**
- **@Component**
- **@PostConstruct**
- **@Service**
- **@Repository**



Annotation @Bean

Annotation @Bean dipakai untuk **mendeteksi sebuah class supaya jadi sebuah bean**.

Selain itu, annotation ini juga dipakai di **level method**. Tapi, method-nya harus punya return type berupa class yang bakal dibuat jadi sebuah bean.

Kalau di Spring Boot, class yang udah dibikin, nggak perlu dibikin method kayak gini lagi karena udah menggunakan annotation lain.



Tapi nih tapi, kita bisa tetap menggunakan annotation ini buat class yang nggak kita buat.

Misalnya, pas pakai library third party yang kita tambahkan di pom.xml, terus class di library tersebut mau di-inject ke class yang udah kita buat. Berarti kita perlu bikin method yang isinya sebuah instance dari class di library tersebut, yang selanjutnya kita tambahkan Annotation @Bean.



Annotation `@Configuration`

Annotation ini berada di tingkat class untuk **mendeteksi method-method yang punya annotation `@Bean`.**

Jadi, kalau ada class yang punya method yang memakai annotation `@Bean`, berarti class tersebut harus pakai annotation `@Configuration`.

Cek halaman selanjutnya untuk melihat contoh pendefinisan suatu bean pakai annotation Bean, ya!

Jangan lupa perhatikan baik-baik, ya~



```
❶ @Configuration
❷ public class Configuration {
❸
❹     @Bean
❺     public HttpClient httpClient() {
❻         return HttpClient.create()
❼             .keepAlive(true)
⽿             .compress(true)
⽻             .responseTimeout(Duration.ofSeconds(120));
⽾     }
⽿
⽻     @Bean
⽾     public PostRouterCIF postRouterCIF(HttpClient httpClient, ConfigurationSupplier
⽻ configurationSupplier){
⽾         return new PostRouterCIF(httpClient, configurationSupplier);
⽾     }
⽿
⽻
⽻     @Bean
⽾     ConfigurationSupplier configurationSupplier(){
⽾         return new DefaultConfigurationSupplier();
⽾     }
⽿ }
```

Contohnya udah disimak kan, bestie?

Dari contoh tersebut, kita bisa lihat penggunaan annotation `@Configuration` dan `@Bean`.

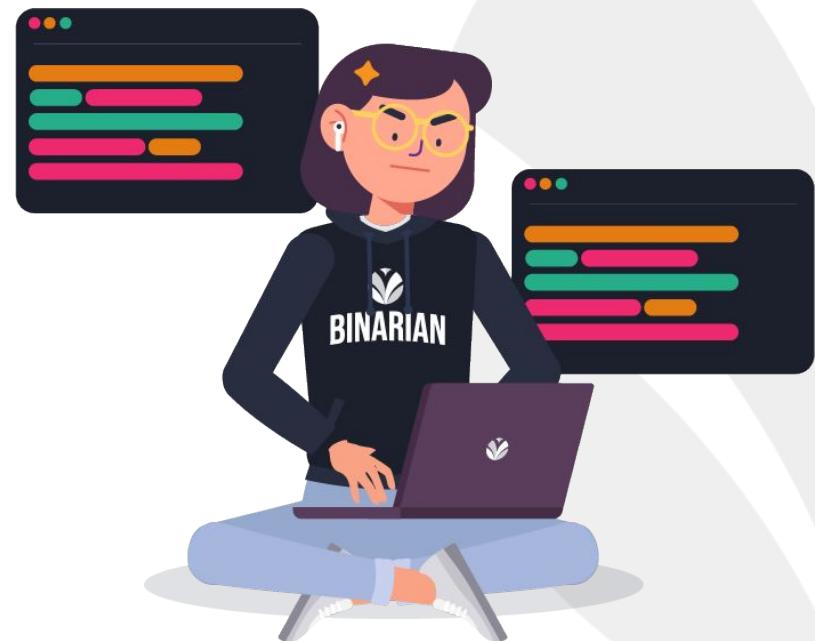
Terus ada juga Bean dari `PostRouterCIF` yang butuh `HTTPClient` dan `Configuration Supplier`, nih.

`HttpClient` merupakan library third party. Oleh karena itu kita harus mendefinisikan `HttpClient` sebagai suatu bean.



Selain itu, ada juga class ConfigurationSupplier yang jadi dependency bagi Bean PostRouterCIF.

Kalau udah begitu, berarti ConfigurationSupplier juga harus dibuatkan method yang meng-instance class ConfigurationSupplier supaya bisa dideteksi sebagai bean.



“Terus apa yang terjadi kalau salah satu argument di constructor PostRouterCIF yang jadi dependency nggak didefinisikan sebagai bean?”

Yang terjadi adalah nullpointer exception pas bean bakal di-instance. Dari situ aplikasi bakal gagal dijalankan, guys.



Annotation @Component

Yaitu annotation yang bisa kita pakai untuk **mendefinisikan class sebagai sebuah bean**.

Beda sama annotation @Bean, @Component merupakan annotation yang berada di level class.

Cek gambar di samping untuk contoh dari penggunaan annotation @Component, ya!



```
@Component  
public class ProducerService {  
    //any content  
}
```

Ada dua turunan dari annotation @Component lho, namanya @Service dan @Repository

Penjelasan dua turunan ini akan satu paket sama penjelasan fungsinya masing-masing. Coba diperhatikan baik-baik yang ada di bawah, ya:

1. **Annotation @Service**, dipakai buat service yang berisi business logic.



```
@Service  
public class ProducerServiceImpl {  
    //any content  
}
```

2. **Annotation @Repository**, dipakai di data access object (DAO) supaya bisa memberikan exception terkait dengan persisting ke database.

Nah, contoh gambar di bawah merupakan contoh dari DAO karena meng-extend JpaRepository. Coba perhatikan baik-baik ya, bestie~

```
@Repository  
public interface FieldRepository extends JpaRepository<FieldReadModel, String> {  
}
```

Annotation `@PostConstruct`

Annotation ini berada di level method yang dipakai untuk **mengeksekusi method content yang ada ketika bean mau dibuat**. Berikut contoh penggunaannya, sob!

A screenshot of a terminal window showing Java code. The code defines a class named `BaseRouter` with a single method, `initialize()`, annotated with `@PostConstruct`. This method creates a `ReactorClientHttpConnector` object and initializes a `WebClient` object named `this.oldProspera` using it.

```
public class BaseRouter {

    @PostConstruct
    void initialize() {
        final ClientHttpConnector httpConnector = new ReactorClientHttpConnector(
            httpClient
                .responseTimeout(Duration.ofMillis(OLD_PROSPERA_TIMEOUT))
                .secure());
        this.oldProspera = WebClient.builder()
            .baseUrl(oldProsperaHost)
            .clientConnector(httpConnector)
            .build();
    }
}
```

Annotation `@PreDestroy`

Annotation ini merupakan **kebalikan** dari `@PostConstruct`.

“Maksudnyaaaa?”

Iya jadi method dengan annotation ini bakal dieksekusi sebelum sebuah bean dihancurkan.

Coba simak contoh di samping, ya!

```
public class BaseRouter {  
  
    @PreDestroy  
    void destroy() {  
        //anycontent  
    }  
}
```

Sipp deh! Akhirnya annotation udah lunas, ya ☺

Sekarang kita bakal masuk ke materi selanjutnya,
yaitu **Konsep Autowire**.



Autowire sebenarnya termasuk annotation,
lho~

Annotation @Autowired diperkenalkan sejak Spring 2.5.

Ada yang unik nih, dibandingkan dengan mendefinisikan bean dan constructor untuk melakukan dependency injection -*kayak yang dibahas pada contoh implementasi @Bean-* kalau kita menggunakan **annotation @Autowired ini jauh lebih simple**.



Bahkan sebelum menggunakan annotation `@Bean` ini, dalam melakukan dependency injection harus melakukan setting dulu ke semua class yang jadi bean. Terus baru deh didefinisikan di dalam sebuah file xml.



Kira-kira, annotation `@Autowired` ini fungsinya buat apa, ya?

Annotation `@Autowired` dipakai untuk **dependency injection**. Selain itu, `@Autowired` ini bisa dipakai bukan cuma di level field aja, tapi juga di level method.

Untuk melakukan dependency injection pakai `@Autowired`, terdapat 3 cara.



Berikut adalah cara untuk melakukan dependency injection dengan `@Autowired`, yaitu:

- **field injection**
- **setter injection**
- **constructor injection**

Kita bahas satu per satu ya!



Field Injection

Untuk melakukan field injection, kita **cuma perlu menambahkan annotation `@Autowired`** pada field yang mau di-inject. Meskipun kita bisa melakukan field injection, tapi di [SonarLint](#) (pendekripsi code yang nggak baik) ini, penggunaan field injection bakal menghasilkan **code smell** atau code yang nggak bagus.

Berikut adalah contohnya!

```
● ● ●  
@Service  
public class ProducerService {  
  
    @Autowired  
    private KafkaTemplate<String, Object> kafkaTemplate;  
}
```

Setter Injection

Untuk melakukan setter injection, kita perlu bikin setter method dari sebuah field. Dimana pada setter method, bakal ditambahkan annotation `@Autowired`. Coba deh cek contohnya di bawah ini, ya!

```
● ● ●  
@Service  
public class ProducerService {  
  
    private KafkaTemplate<String, Object> kafkaTemplate;  
  
    @Autowired  
    public void setKafkaTemplate(KafkaTemplate<String, Object> kafkaTemplate) {  
        this.kafkaTemplate = kafkaTemplate;  
    }  
  
}
```

Constructor Injection

Untuk melakukan constructor injection, field yang bakal di-inject harus punya access modifier final. Kemudian field harus menjadi parameter bagi constructor.

Lalu tinggal tambahkan annotation `@Autowired` pada constructor tersebut. Berikut adalah contohnya:

```
@Service
public class ProducerService {

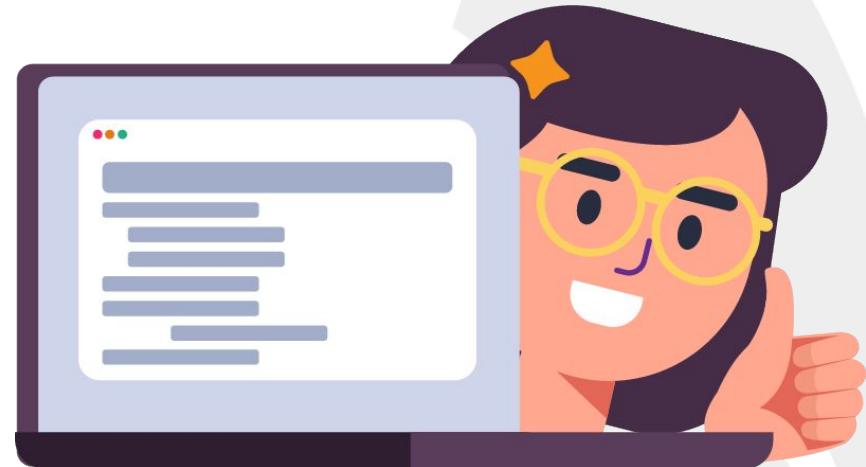
    private final KafkaTemplate<String, Object> kafkaTemplate;

    @Autowired
    public ProducerService(KafkaTemplate<String, Object> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }
}
```

Oh iya, dalam melakukan constructor injection, field yang bakal di-inject harus punya **access modifier final**. Terus field harus jadi parameter bagi constructor.

Habis itu, kita menambahkan annotation `@Autowired` pada constructor tersebut.

Coba cek contohnya di slide selanjutnya, ya!



```
@Service
public class ProducerService {

    private final KafkaTemplate<String, Object> kafkaTemplate;

    @Autowired
    public ProducerService(KafkaTemplate<String, Object> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

}
```

Wihiyy! progress kita hari ini ajib banget ya.
Kalian keren banget pokoknya!

Hampir menuju materi terakhir, kita bakal bahas
dulu pom.xml dan Configuration.

Kira-kira siapa ya mereka?



Kamu masih ingat pembahasan materi Maven tentang cara menambahkan library atau dependency?

Sama kayak project Maven lainnya untuk menambah suatu dependency, tag dependency di file pom.xml harus dimodifikasi dulu sebelum kita bisa menambahkan library apapun ke dalamnya.

Karena kita menggunakan Spring framework, tentunya ada dependency Spring Framework di salah satu dependency-nya.



Selanjutnya kita bakal bahas salah satu file yang berisi configuration dari sebuah aplikasi pada Spring Boot~

Configuration ini yaitu application.properties.

application.properties merupakan default file properties dari aplikasi yang dibuat pakai Spring Boot.

File application.properties bakal diletakkan di dalam resource folder.



“Terus fungsi dari application.properties ini apa, ya?”

Berikut beberapa fungsi dari application properties:

1. Menyimpan value yang bersifat default atau constant.
2. Menyimpan konfigurasi dari dependency. Beberapa value dari konfigurasi bisa diatur pakai file application.properties.
3. Value yang ada di properties bisa ditimpa atau di-override sama file environment, jadi nilainya bisa diubah sesuai dengan environment dimana aplikasi tersebut di-deploy.



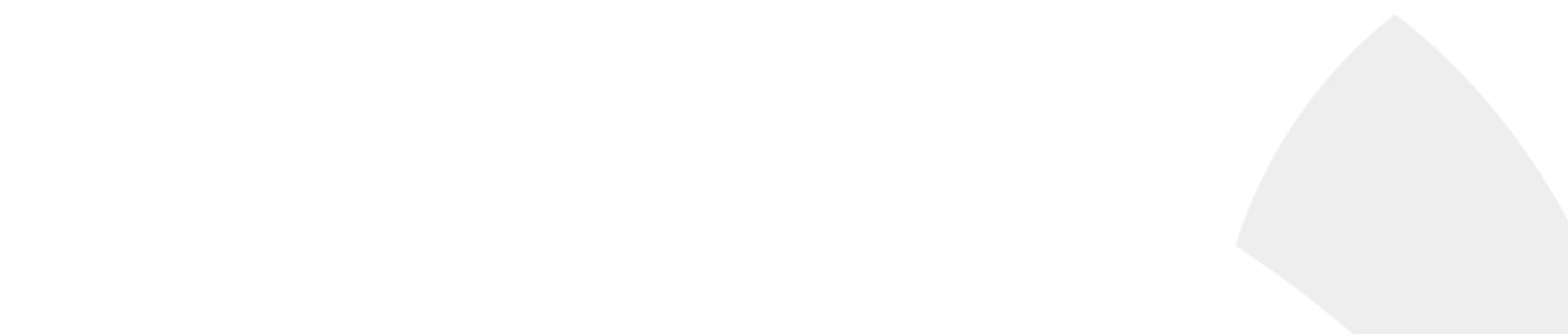
Berikut merupakan contoh konfigurasi dari sebuah dependency yang diletakkan pada properties!

Biasanya konfigurasi di samping disediakan oleh library yang udah didukung oleh Spring. Hal ini bakal bikin code jadi lebih rapi.

Pada contoh di samping, ada **notasi \${}** yang disebut dengan **placeholder**.

Kalau pakai placeholder, kita bisa mereferensikan value dari variable dan menggunakan default value.

```
spring.kafka.bootstrap-servers=${KAFKA_HOST:dirty-kafka.abc.com:443}
spring.kafka.producer.client-id=${KAFKA_PRODUCER_CLIENT_ID:dirty-apps}
spring.kafka.producer.retries=0
spring.kafka.consumer.auto-offset-reset=latest
```



Pada property `spring.kafka.bootstrap-servers`, pertama bakal mengecek variable `KAFKA_HOST`, kalau nggak ditemukan atau null, maka bakal ada default value, yaitu `dirty-kafka.abc.com:443`.



```
spring.kafka.bootstrap-servers=${KAFKA_HOST:dirty-kafka.abc.com:443}
spring.kafka.producer.client-id=${KAFKA_PRODUCER_CLIENT_ID:dirty-apps}
spring.kafka.producer.retries=0
spring.kafka.consumer.auto-offset-reset=latest
```

“Terus, gimana caranya mengambil value dari file application.properties?”

Ada beberapa cara buat mengambil value dari application.properties nih, yaitu:

- Pakai annotation `@Value`
- Pakai library Spring framework Environment
- Mengubahnya jadi sebuah java object



Annotation @Value

Berikut adalah contoh penggunaan Annotation @Value:

- Pada file application.properties
- Pada class yang mau mengambil value dari properties



```
jdbc.username=admin
```

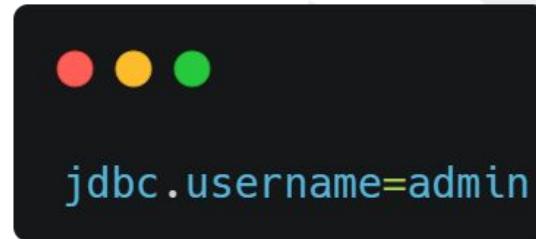


```
@Value( "${jdbc.username}" )  
private String jdbcUser;
```

Library Environment

Berikut adalah cara menggunakan Library Environment!

- Pada file `application.properties`



- Pada class yang mau mengambil value dari properties, kita harus meng-inject Environment:



```
@Autowired  
private Environment env;
```

Kemudian kita menyetor value dari properties pakai Environment:



```
String username = env.getProperty("jdbc.username");
```

Mengubahnya jadi sebuah object

Mengambil value dengan cara mengubahnya jadi sebuah object bisa kamu lihat secara lengkap pada referensi [di sini](#), yaaaa~



Guys! Kita sampai di materi terakhir, yaitu berkaitan sama yang namanya paradigma programming.

Kalau kamu masih inget OOP (Object Oriented Programming), doi merupakan salah satunya.

Tapi, bukan OOP yang bakal kita bahas, melainkan Spring AOP.



Kenalan yuk sama Spring AOP!

Spring AOP atau dikenal sebagai Spring Aspect Oriented Programming adalah implementasi AOP dengan menggunakan framework Spring.

Sebagaimana arti dalam bahasa Indonesia, AOP merupakan paradigma pemrograman berbasiskan aspect sebagai pelengkap dari OOP (Object Oriented Programming).

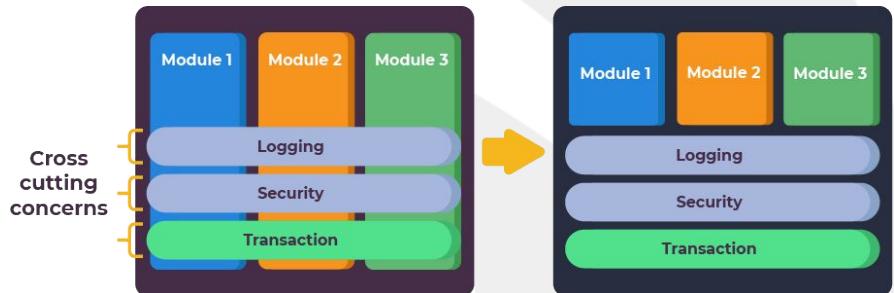
Berhubung posisi AOP sebagai **pelengkap**, ini berarti AOP **bukan** untuk menggantikan OOP.



Di sini, AOP bertujuan untuk meningkatkan modularitas dengan memungkinkan pemisahan masalah cross-cutting.

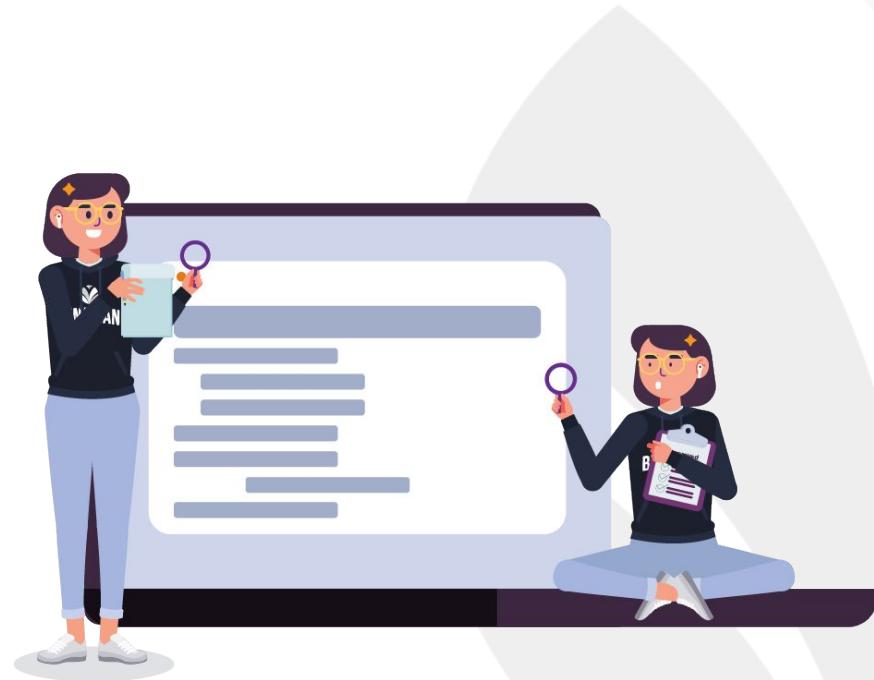
Cross cutting yaitu pemisahan code utama dari code pendukung, sehingga perubahan dan penambahan code tidak akan mengganggu satu sama lain.

Kalau kamu penasaran, beberapa contoh yang menggunakan pendekatan AOP yaitu transaksi management, logging, atau permasalahan security.



Oh iya, proses kerja Spring AOP yaitu dengan menambahkan behaviour tambahan ke code yang ada tanpa memodifikasi code itu sendiri.

Sebagai gantinya, kita dapat mendeklarasikan code baru dan behaviour baru secara terpisah. Gimana, keren kan? □

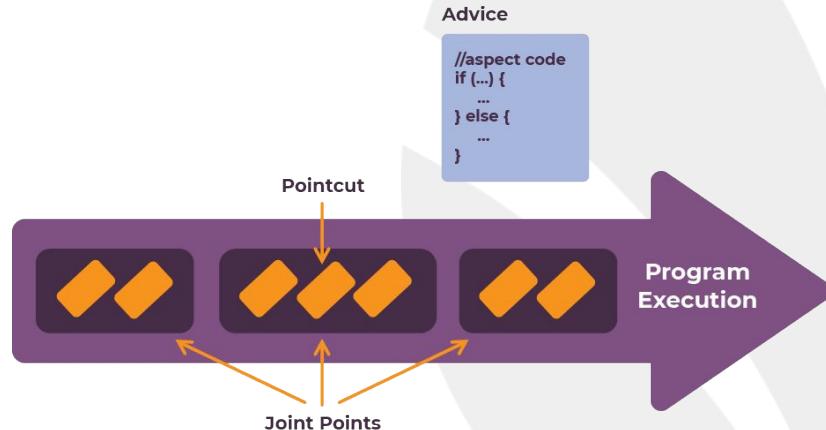


Makin kenal Terminologi Spring AOP, makin jagoo!

Pada Spring AOP, tentunya ada istilah baru yang bakal kita temui. Kalau kamu lirik gambar Program Execution disamping ada tiga keywords utama:

- **Join Points**
- **Pointcut**
- **Advice**

Yuk, geser slide buat tau penjelasannya.



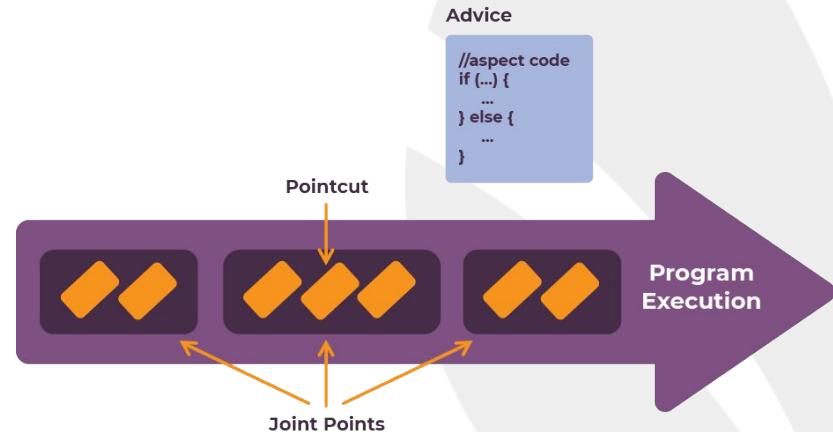
Joint Points

Joint point merupakan bagian atau point tempat aspect dieksekusi.

Pointcut

Merupakan ekspresion atau predicate yang cocok dengan joint point.

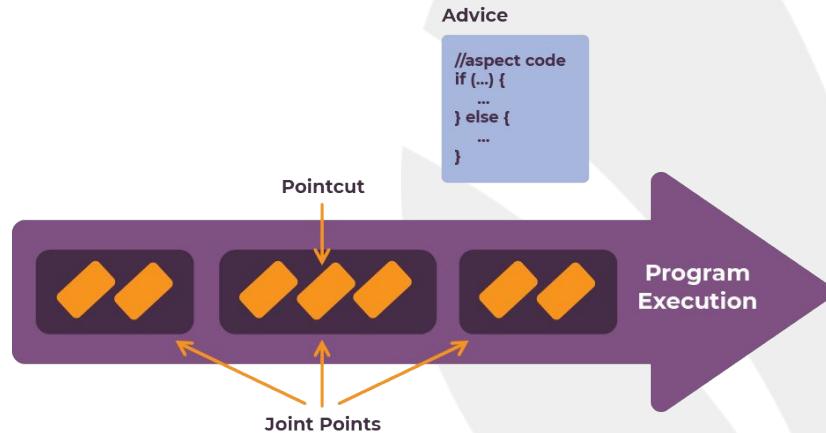
Kalau cocok, maka program akan dieksekusi. Sedangkan kalau nggak cocok maka program tidak akan dieksekusi.



Advice

Advice adalah bagian utama dan terpenting dari Spring AOP.

Advice adalah tempat action yang diambil oleh aspect pada Join Points. Kalo nggak ada Advice, maka tidak ada program yang dieksekusi



Spring AOP Config

Lastly, untuk menutup materi hari ini ada Spring AOP Configuration yang bisa kamu cek contohnya di bawah ini, ya!

```
<bean id="sampleAdder" class="org.baeldung.logger.SampleAdder" />
<bean id="doAfterReturningAspect"
      class="org.baeldung.logger.AdderAfterReturnAspect" />
<aop:config>
    <aop:aspect id="aspects" ref="doAfterReturningAspect">
        <aop:pointcut id="pointCutAfterReturning" expression=
            "execution(* org.baeldung.logger.SampleAdder+.*(..))"/>
        <aop:after-returning method="afterReturn"
            returning="returnValue" pointcut-ref="pointCutAfterReturning"/>
    </aop:aspect>
</aop:config>
```



Coba sini Sabrina mau denger suaranya yang udah belajar Spring Framework □

By the way Sabrina mau tanya nih, selain berfungsi untuk tidak mengganggu code ketika terjadi perubahan atau penambahan, apa lagi sih manfaat dari AOP?

Nah, selesai sudah pembahasan kita di Chapter 4 Topic 2 ini.

Selanjutnya, kita bakal bahas tentang **Relational Databases, ORM and Database Operation**.

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~

