**PROGRAM STUDI INFORMATIKA**
**UNIVERSITAS SANATA DHARMA**

**PRAKTIKUM PBO LANJUT**

# MODUL PENGANTAR SPRING-3

## I. TUJUAN PRAKTIKUM

1. Memahami dan menerapkan **Spring AOP** untuk memisahkan logika lintas aplikasi (cross-cutting concerns) seperti logging, keamanan, dan validasi.
2. Menerapkan **Spring Transaction Management** untuk mengelola transaksi di aplikasi..

## II. DASAR TEORI

### 1. Aspect-Oriented Programming (AOP)

**Aspect-Oriented Programming (AOP)** adalah paradigma pemrograman yang bertujuan untuk memisahkan **cross-cutting concerns** (logika lintas aplikasi) dari logika utama aplikasi. Dalam konteks **Spring Framework**, AOP memungkinkan kita menerapkan fungsionalitas seperti logging, validasi, manajemen transaksi, dan pengelolaan exception tanpa mencampurkan logika tersebut ke dalam kode bisnis utama.

**Konsep Dasar AOP:**

1. **Aspect**: Komponen yang berisi kode untuk menangani **cross-cutting concerns**. Dalam Spring, ini biasanya diimplementasikan sebagai kelas yang dianotasi dengan **@Aspect**.
2. **Join Point**: Titik tertentu dalam eksekusi program, seperti pemanggilan metode, di mana sebuah aspect dapat diterapkan.
3. **Advice**: Potongan kode yang dijalankan pada **join point** tertentu. Ada beberapa jenis **advice**:
   - **@Before**: Advice dijalankan sebelum eksekusi metode.
   - **@After**: Advice dijalankan setelah metode selesai.
   - **@AfterReturning**: Advice dijalankan setelah metode berhasil dijalankan.
   - **@AfterThrowing**: Advice dijalankan jika metode melempar pengecualian.
   - **@Around**: Advice dijalankan sebelum dan sesudah eksekusi metode.
4. **Pointcut**: Ekspresi yang menentukan **join points** yang cocok untuk advice. Contoh, pointcut bisa digunakan untuk menentukan metode-metode mana yang akan diterapkan aspect.
5. **Weaving**: Proses di mana aspect terintegrasi ke dalam logika aplikasi utama. Dalam Spring, **weaving** terjadi pada waktu runtime.

**Contoh Penggunaan Aspect di Spring:**

Pada praktikum di bawah, kita menggunakan AOP untuk menerapkan validasi otomatis pada metode **addBook** di **BookServiceImpl**. Dalam kode ini:

- **Aspect** `ValidationAspect` memeriksa apakah judul buku valid sebelum buku ditambahkan melalui **BookServiceImpl**.
- **Pointcut**

`execution(* com.example.springcoreadvanced.service.BookService.addBook(..))` menentukan bahwa **aspect** ini hanya diterapkan pada metode **`addBook`** di kelas **BookServiceImpl**.

## 2. Transaction Management

**Transaction Management** adalah salah satu fitur penting dalam aplikasi enterprise, terutama saat berurusan dengan **Database Operations**. Dalam **Spring Framework**, transaksi bisa dikelola secara otomatis menggunakan anotasi **`@Transactional`**. Ini memungkinkan kita untuk menjaga **consistency** dan **integrity** data, terutama dalam situasi di mana operasi melibatkan beberapa langkah yang saling tergantung.

**Konsep Transaksi:**

1. **Atomicity**: Semua operasi dalam satu transaksi dianggap sebagai satu unit. Jika satu operasi gagal, seluruh transaksi dibatalkan (rollback).
2. **Consistency**: Transaksi membawa sistem dari satu status konsisten ke status konsisten lainnya.
3. **Isolation**: Transaksi dieksekusi secara independen dari transaksi lainnya.
4. **Durability**: Setelah transaksi selesai, perubahan bersifat permanen dan tidak akan hilang bahkan jika terjadi kegagalan sistem.

**Spring Transaction Management:**

Spring menyediakan dukungan transaksi berbasis **AOP** dengan anotasi **`@Transactional`**, yang memungkinkan pengelolaan transaksi secara deklaratif.
**`@Transactional`** dapat ditempatkan pada metode atau kelas untuk menunjukkan bahwa metode tersebut dijalankan dalam konteks transaksi.

**Contoh Transactional di Praktikum:**

Pada praktikum di bawah, kita menandai metode **`addBook`** dengan **`@Transactional`** untuk memastikan bahwa operasi penambahan buku dijalankan dalam satu unit transaksi. Jika terjadi pengecualian, transaksi akan dibatalkan.

```
    @Override
    @Transactional
    public void addBook(Book book) {
        System.out.println("Adding book: " +
book.getTitle());
        bookRepository.save(book);

        // Simulasi kegagalan jika buku yang ditambahkan
adalah "Spring in Action"
        if ("Spring in Action".equals(book.getTitle())) {
            throw new RuntimeException("Error: Cannot add
this book!");
        }
    }
```

Jika metode ini memicu pengecualian, seluruh transaksi dibatalkan dan perubahan data yang dilakukan (misalnya, menyimpan buku) akan di-rollback.

**Transaction Manager:**
Spring memerlukan **Transaction Manager** untuk mengelola transaksi. Ini didefinisikan dalam konfigurasi seperti berikut:

```
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> <property name="dataSource" ref="dataSource"/> </bean>
```

Dengan **DataSourceTransactionManager**, Spring secara otomatis mengelola transaksi database dengan cara yang konsisten.

## 3. Exception Handling

**Exception Handling** adalah proses menangani error atau kejadian yang tidak terduga selama eksekusi program. Dengan exception handling yang baik, aplikasi tidak akan berhenti tiba-tiba saat terjadi error, melainkan bisa menangani error tersebut secara terstruktur.

**Jenis-Jenis Exception di Java:**

1. **Checked Exception**: Error yang bisa diprediksi dan harus ditangani oleh program, misalnya **`IOException`**, **`SQLException`**. Kompilator Java akan mengharuskan Anda untuk menangani atau melempar ulang checked exception.
2. **Unchecked Exception**: Kesalahan yang terjadi akibat kesalahan logika dalam program, seperti **`NullPointerException`** atau **`ArrayIndexOutOfBoundsException`**. Unchecked exceptions tidak harus ditangani secara eksplisit dalam kode.
3. **Error**: Kesalahan serius yang biasanya tidak bisa ditangani oleh aplikasi, seperti **`OutOfMemoryError`**.

**Exception Handling dalam Java:**
Untuk menangani pengecualian, Java menyediakan mekanisme **try-catch-finally**:

```
try {
    // Kode yang mungkin memicu exception
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero");
} finally {
    System.out.println("This block always executes");
}
```

- **try**: Blok kode yang mungkin memicu pengecualian.
- **catch**: Blok yang menangani pengecualian jika terjadi.
- **finally**: Blok kode yang selalu dieksekusi, baik pengecualian terjadi atau tidak.

**Exception Handling di Praktikum:**
Pada praktikum ini, pengecualian seperti **`RuntimeException`** dilempar untuk mensimulasikan kegagalan dalam transaksi. Untuk menangani pengecualian tersebut agar program tidak langsung berhenti, kita menggunakan **`try-catch`**.
Dalam kasus ini, jika metode **`addBook`** memicu pengecualian, program akan menampilkan pesan kesalahan tetapi tidak keluar secara tiba-tiba. Ini menjaga agar aplikasi tetap berjalan meskipun terjadi error.

# III. LATIHAN

## Bagian 1: Menambahkan AOP untuk Logging dan Validasi

### Langkah 1.1: Setup AOP untuk Logging

1. **Tambahkan dependensi Spring AOP di `pom.xml`:**

```
<!-- AspectJ -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.9.7</version>
        </dependency>
        <!-- Spring AOP -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>5.3.10</version>
        </dependency>
```

2. **Tambahkan Aspek Logging**
   - Buat package baru bernama `aspect`, dan tambahkan kelas `LoggingAspect.java` di dalamnya:

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
@Component
public class LoggingAspect {
    @Before("execution(*
com.example.library.management.service.BookService.*(..))")
    public void logBeforeServiceMethods() {
        System.out.println("LoggingAspect: A method in
BookService is being called");
    }
}
```

**Penjelasan:**

- **@Aspect**: Menandakan bahwa kelas ini adalah sebuah aspek.

- **@Before**: Menjalankan metode `logBeforeServiceMethods()` sebelum setiap metode di **BookService** dipanggil. Ini berfungsi sebagai logging otomatis sebelum metode seperti `addBook` atau `deleteBook` dipanggil.

3. **Perbarui `applicationContext.xml` untuk mendukung AOP:**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:context="http://www.springframework.org/schema/c
ontext"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/sch
ema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd
http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-
context.xsd
           http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-
aop.xsd">
    <!-- Aktifkan scanning komponen -->
    <context:component-scan base-
package="com.example.library.management" />
    <!-- Aktifkan AOP Auto Proxy -->
    <aop:aspectj-autoproxy/>
     <bean id="loggingAspect"
class="com.example.library.management.aspect.LoggingAs
pect"/>
</beans>
```

**Langkah 1.2: Implementasi Validasi dengan AOP**

1. **Tambahkan Aspek Validasi**
   - Tambahkan kelas `ValidationAspect.java` di package `aspect` untuk validasi data sebelum ditambahkan:

```
@Aspect
@Component
public class ValidationAspect {

        // Pointcut untuk metode addBook di
BookService yang menerima objek Book sebagai parameter
    @Before("execution(*
com.example.library.management.service.BookService.add
Book(..)) && args(book)")
```

```
    public void validateBook(Book book) {
        if (book.getTitle() == null ||
book.getTitle().trim().isEmpty()) {
            throw new IllegalArgumentException("Book
title cannot be null or empty!");
        }
        System.out.println("ValidationAspect: Book
title is valid");
    }
}
```

- **Penjelasan:**
  - **@Before("execution(..)")**: Dijalankan sebelum metode `addBook` di **BookService**. Aspek ini memastikan bahwa judul buku yang dimasukkan valid sebelum ditambahkan ke dalam repository.

2. **Perbaiki method addBook di kelas BookServiceImpl untuk memastikan proses tambah buku**

```
public void addBook(Book book) {
        System.out.println("Adding book: " +
 book.getTitle());
        bookRepository.save(book);
    }
```

3. Perbaiki kelas aplikasi sehingga setiap kali Anda menambahkan buku, Spring AOP akan memeriksa apakah judul buku valid, dan juga akan mencatat log ketika metode dijalankan. Perhatikan bahwa ketika kita menambahkan buku dengan title null atau empty maka akan muncul pesan kesalahan.

```
ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

        LibraryController libraryController =
context.getBean(LibraryController.class);
        try {
            libraryController.addNewBook("978-1234567897",
"Spring Framework", "Craig Walls");
            libraryController.addNewBook("978-9876543210",
null, "James Gosling");
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("All books in library:");
        libraryController.displayAllBooks();
```

## Bagian 2: Menambahkan Transaksi dengan Spring Transaction Management

**Langkah 2.1: Menyiapkan Manajemen Transaksi**

1. **Tambahkan dependensi untuk transaksi dan database H2 di `pom.xml`:**

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>5.3.10</version>
</dependency>

<!-- H2 Database -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

2. Tambahkan konfigurasi manajemen transaksi di `applicationContext.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:context="http://www.springframework.org/schema/c
ontext"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/sch
ema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-
context.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-
aop.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-
tx.xsd">
    <!-- DataSource untuk H2 Database -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManag
erDataSource">
        <property name="driverClassName"
value="org.h2.Driver"/>
        <property name="url"
value="jdbc:h2:mem:testdb"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
```

```
    </bean>

    <!-- Transaction Manager -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceT
ransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <tx:annotation-driven />

    <!-- Aktifkan scanning komponen -->
    <context:component-scan base-
package="com.example.library.management" />

    <!-- Aktifkan AOP Auto Proxy -->
    <aop:aspectj-autoproxy/>

    <bean id="loggingAspect"
class="com.example.library.management.aspect.LoggingAs
pect"/>
</beans>
```

**Langkah 2.2: Menambahkan Transaksi di Service**

1. **Modifikasi `BookServiceImpl` untuk mendukung transaksi:**

```
@Transactional
    public void addBook(Book book) {
        System.out.println("Adding book: " +
book.getTitle());

        if ("Spring in
Action".equals(book.getTitle())) {
            throw new RuntimeException("Error: Cannot
add this book!");
        }
        bookRepository.save(book);
    }
```

Metode `addBook` ditandai dengan anotasi **@Transactional**. Jika terjadi kesalahan saat menambahkan buku, seluruh transaksi akan dibatalkan.

2. **Uji Transaksi:**
   - Uji dengan menambahkan buku pada kelas aplikasi :
     `libraryController.addNewBook("978-9876543210", "Spring in Action", "James Gosling");`

     Perhatikan apa yang terjadi saat aplikasi dijalankan.
   - Perbaiki try-catch pada kelas aplikasi menjadi

```
catch (Exception e) {
            System.out.println(e.getMessage());
        }
```

Perhatikan apa yang terjadi saat aplikasi dijalankan.

## IV. LAPORAN

1. Penjelasan kode program, capture GUI untuk setiap soal (9 x (@10%)

**Referensi Utama:**


1. Craig Walls, "Spring in Action," 6th Edition, Manning Publications, 2022.
2. Juergen Hoeller, "Spring Framework Reference Documentation," Spring.io.