PROGRAM STUDI INFORMATIKA
UNIVERSITAS SANATA DHARMA

PRAKTIKUM PBO LANJUT

MODUL PENGANTAR SPRING-1

A. TUJUAN PRAKTIKUM

1. Mahasiswa memahami framework, library
2. Mahasiswa memahami arsitektur Spring dan konsep DI & IoC
3. Mahasiswa dapat membuat proyek Spring dengan konfigurasi dasar

B. DASAR TEORI

I. Framework

Framework adalah struktur atau kerangka kerja yang disediakan oleh platform atau bahasa pemrograman untuk mempermudah pengembangan aplikasi. Framework memberikan serangkaian komponen, *design pattern*, dan *tool* yang dapat digunakan oleh pengembang untuk membangun aplikasi lebih cepat dan efisien.

Karakteristik Utama:

- Memiliki aturan dan struktur.
- Memungkinkan pengembang untuk mengelola dan mengatur kode dengan lebih baik.
- Meningkatkan produktivitas dan mengurangi kesalahan dengan menyediakan komponen yang dapat digunakan kembali.

Contoh:

- **Spring Framework** (Java)
- **Django** (Python)
- **Laravel** (PHP)

Beda Framework Dan Library

Library adalah kumpulan fungsi atau kelas yang dapat digunakan oleh pengembang dalam proyek mereka. Pengembang yang memanggil library, dan library bekerja sesuai dengan instruksi pengembang. Contoh di Java: `Apache Commons` adalah library yang menyediakan fungsi-fungsi umum seperti manipulasi string, data, dan angka.

Framework adalah arsitektur atau kerangka yang mengatur alur kerja aplikasi. Dalam framework, framework yang memanggil kode kita, bukan sebaliknya. Ini dikenal sebagai prinsip "Inversion of Control" (IoC)**.** Contoh **:** `Spring Framework` mengatur alur aplikasi dan memanggil kode kita saat dibutuhkan, misalnya melalui anotasi untuk dependency injection.

Perbedaan Utama:

- **Library**: Pengembang memanggil library (kendali di tangan pengembang).
- **Framework**: Framework memanggil kode pengembang (kendali di tangan framework).

Apa itu Spring Framework?

Spring Framework adalah framework open-source untuk pengembangan aplikasi berbasis Java. Awalnya dirancang untuk aplikasi enterprise, Spring sekarang digunakan secara luas untuk membangun aplikasi web, aplikasi mikroservis, dan aplikasi cloud-native. Salah satu fitur kunci yang menjadikannya unggul adalah kemampuan Dependency Injection (DI), yang memisahkan logika bisnis dari logika teknis, serta modularitas yang tinggi, memungkinkan penggunaan fitur sesuai kebutuhan.

Keunggulan dan Kekurangan Spring Framework

Keunggulan:

- **Modular**: Spring memiliki modul-modul yang dapat digunakan secara terpisah atau bersama-sama sesuai kebutuhan.
- **IoC dan DI**: Dependency Injection dan Inversion of Control membantu mengelola dependensi dengan lebih mudah, membuat aplikasi lebih fleksibel dan modular.
- **Mendukung Integrasi**: Spring mudah diintegrasikan dengan teknologi lain seperti Hibernate, JPA, dan MyBatis.
- **Spring Boot**: Menyederhanakan konfigurasi, memungkinkan pengembangan aplikasi lebih cepat dengan pengaturan minimal.

Kekurangan:

- **Waktu belajar**: Meskipun Spring sangat fleksibel, memahami seluruh ekosistem Spring memerlukan waktu.
- **Overhead konfigurasi**: Sebelum Spring Boot, Spring dikenal memiliki konfigurasi yang kompleks, terutama dalam XML.

Arsitektur Spring Framework

Spring memiliki beberapa layer arsitektur, yang mencakup beberapa modul utama.
Berikut adalah beberapa layer utama:

- **Core Container**: Ini adalah inti dari Spring yang mencakup IoC dan Dependency Injection.
- **Data Access Layer**: Mendukung akses data menggunakan JDBC, ORM (Hibernate, JPA), dan transaksi.
- **Web Layer**: Mendukung pengembangan aplikasi web berbasis Spring MVC, RESTful, dan aplikasi SOAP.
- **AOP (Aspect Oriented Programming)**: Memungkinkan pemrograman berbasis aspek seperti logging, transaksi, dan keamanan.
- **Spring Boot**: Mempermudah konfigurasi dan pembuatan aplikasi Spring dengan menambahkan auto-configuration dan server embedded.

II. Core Container / Spring Container

Spring Container adalah inti dari **Spring Framework** yang bertanggung jawab untuk mengelola siklus hidup dan konfigurasi objek dalam aplikasi. Objek-objek yang dikelola oleh Spring Container disebut **Spring Beans**. Spring Container membuat objek, mengatur dependensi mereka, mengelola siklus hidup, dan menyuntikkan dependensi yang dibutuhkan sesuai dengan konfigurasi.

Fungsi Utama Spring Container:

1. **Instansiasi Bean:** Membuat instance dari class yang dideklarasikan sebagai bean di Spring.
2. **Injeksi Dependensi (Dependency Injection):** Menyuntikkan dependensi yang diperlukan oleh bean.
3. **Mengelola Siklus Hidup Bean:** Mengelola lifecycle dari bean (dari pembuatan hingga penghancuran).
4. **Pengelolaan Konfigurasi:** Spring Container dapat dikonfigurasi menggunakan file XML atau anotasi berbasis Java.

Jenis-Jenis Spring Container:

Spring menyediakan beberapa jenis IoC Container, dua yang paling sering digunakan adalah:

1. **BeanFactory:**
   - **BeanFactory** adalah antarmuka dasar untuk IoC Container di Spring. Ini adalah container yang ringan dan hanya memuat bean saat dibutuhkan (**lazy loading**).
   - **BeanFactory** cocok digunakan dalam aplikasi yang membutuhkan performa tinggi dan hanya membutuhkan inisialisasi bean ketika diperlukan.

     Contoh Penggunaan `BeanFactory`:

     ```
     BeanFactory factory = new XmlBeanFactory(new FileSystemResource("beans.xml"));
     Car car = (Car) factory.getBean("car");
     ```
2. **ApplicationContext:**
   - **ApplicationContext** adalah ekstensi dari **BeanFactory** dengan fitur tambahan seperti event propagation, deklarasi transaksi, dan resolusi message dari file properti. Ini adalah pilihan yang lebih umum untuk aplikasi enterprise.
   - **ApplicationContext** memuat semua bean pada saat startup aplikasi (**eager loading**), yang membuat aplikasi lebih siap, meskipun membutuhkan lebih banyak memori di awal.

Contoh Penggunaan ApplicationContext:

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

Car car = context.getBean("car", Car.class);
```

III. Bean dan Spring Container:

Bean adalah objek yang dikelola oleh Spring Container. Bean didefinisikan di dalam konteks Spring, baik melalui file XML, anotasi, atau file konfigurasi berbasis Java. Saat Spring Container diinisialisasi, ia akan memuat bean-bean ini dan mengelola siklus hidup mereka.

Cara Mendaftarkan Bean di Spring:

1. **Konfigurasi XML:**
   - Anda dapat mendaftarkan bean di file XML seperti berikut:

```
<bean id="car" class="com.example.Car">
    <property name="engine" ref="engine"/>
</bean>

<bean id="engine" class="com.example.Engine"/>
```

2. **Konfigurasi dengan Anotasi:**
   - Spring juga mendukung anotasi untuk mengonfigurasi bean:

```
@Component
public class Car {
    @Autowired
    private Engine engine;
}
```

   - Di sini, **@Component** menandai kelas **Car** sebagai bean yang akan dikelola oleh Spring Container, dan **@Autowired** digunakan untuk menyuntikkan dependensi.
3. **Konfigurasi Java (Java-based Configuration):**
   - Selain XML dan anotasi, Spring mendukung konfigurasi berbasis Java.

```
@Configuration
public class AppConfig {
    @Bean
    public Car car() {
        return new Car(engine());
    }

    @Bean
    public Engine engine() {
        return new Engine();
    }
}
```

IV. Siklus Hidup Bean di Spring Container:

Spring Container tidak hanya membuat bean, tetapi juga mengelola seluruh siklus hidup bean. Ini berarti bahwa Spring bertanggung jawab untuk:

1. **Membuat Bean:** Spring akan meng-instantiate objek bean saat konteks aplikasi dimuat.
2. **Injeksi Dependensi:** Setelah bean dibuat, Spring akan menyuntikkan dependensi yang diperlukan oleh bean.
3. **Inisialisasi:** Setelah dependensi di-inject, bean akan dipersiapkan untuk digunakan (jika ada metode `init()`).
4. **Penghancuran:** Saat aplikasi berhenti atau saat bean tidak lagi diperlukan, Spring akan memanggil metode `destroy()` untuk menghancurkan bean.

Siklus Hidup Bean dengan Callback Methods:

- **@PostConstruct**: Dapat digunakan untuk menjalankan kode setelah bean dibuat dan setelah dependensi disuntikkan.
- **@PreDestroy**: Dapat digunakan untuk membersihkan sumber daya sebelum bean dihancurkan.

Contoh:

```
@Component
public class Car {

    @PostConstruct
    public void init() {
        System.out.println("Car bean is initialized");
    }

    @PreDestroy
    public void destroy() {
        System.out.println("Car bean is about to be
destroyed");
    }
}
```

V. Konsep Dependency Injection (DI) dan Inversion of Control (IoC)

Inversion of Control (IoC):

IoC adalah prinsip di mana framework yang mengontrol aliran program, bukan pengembang. Artinya, framework memanggil kode pengembang berdasarkan konfigurasi yang sudah ditentukan. Contoh: Spring menggunakan IoC untuk membuat dan mengelola obyek, pengembang tidak perlu melakukan instansiasi obyek secara manual.

Dependency Injection (DI):
DI adalah teknik di mana obyek atau dependensi disuntikkan ke dalam obyek lain melalui constructor, setter, atau metode. Hal ini mengurangi *tight coupling* dan mempermudah pengujian. Contoh: Dengan Spring, kita bisa menggunakan anotasi `@Autowired` untuk melakukan dependency injection.

Jenis-Jenis Dependency Injection:

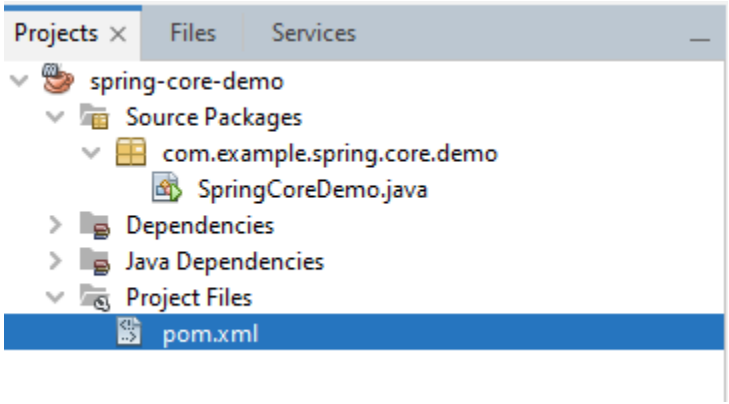
1. **Constructor Injection:** Dependensi disuntikkan melalui konstruktor.
2. **Setter Injection:** Dependensi disuntikkan melalui metode setter.
3. **Field Injection:** Dependensi disuntikkan langsung ke field dengan menggunakan anotasi **@Autowired**.

C. LATIHAN

Bagian 1: Setup Proyek Spring Core

Langkah 1.1: Membuat Proyek Maven

1. **Buka IDE** dan buat proyek Maven baru:
   a. Pilih **File > New Project > Maven > Java Application**.
   b. Beri nama proyek: `spring-core-demo`.
   c. Isi **Group ID**: `com.example`.
   d. Isi **Artifact ID**: `spring-core-demo`.
   e. Klik **Finish**.
2. **Tambahkan Spring Core Dependensi**:
   2. Buka file `pom.xml` dan tambahkan dependensi untuk Spring Core.




```
<dependencies>
  <!-- Spring Core →
 <dependency>
     <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-core</artifactId>
    <version>5.3.10</version> <!-- versi bisa disesuaikan -->
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.10</version> <!-- versi bisa disesuaikan -->
  </dependency>
</dependencies>
```

3. Update Proyek:

- Klik kanan proyek > Build with Dependencies untuk memastikan Maven mengunduh semua dependensi yang diperlukan.

Bagian 2: Implementasi Dependency Injection (DI) dengan XML

Langkah 2.1: Membuat Bean dan Konfigurasi XML

1. **Buat kelas sederhana `Engine.java`:**
   - Di package `com.example.springcore`, buat kelas berikut:

```
public class Engine {
  private String type;


  public void setType(String type) {
    this.type = type;
  }
```

```
    public void start() {
        System.out.println("Engine type " + type + " started!");
    }
}
```

2. **Buat kelas `Car.java`:**
   - Buat kelas `Car.java` yang memiliki dependency pada `Engine`.

```
public class Car {
    private Engine engine;

    // Dependency Injection menggunakan setter
    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    public void drive() {
        engine.start();
        System.out.println("Car is moving.");
    }
}
```

3. **Buat file `applicationContext.xml`:**

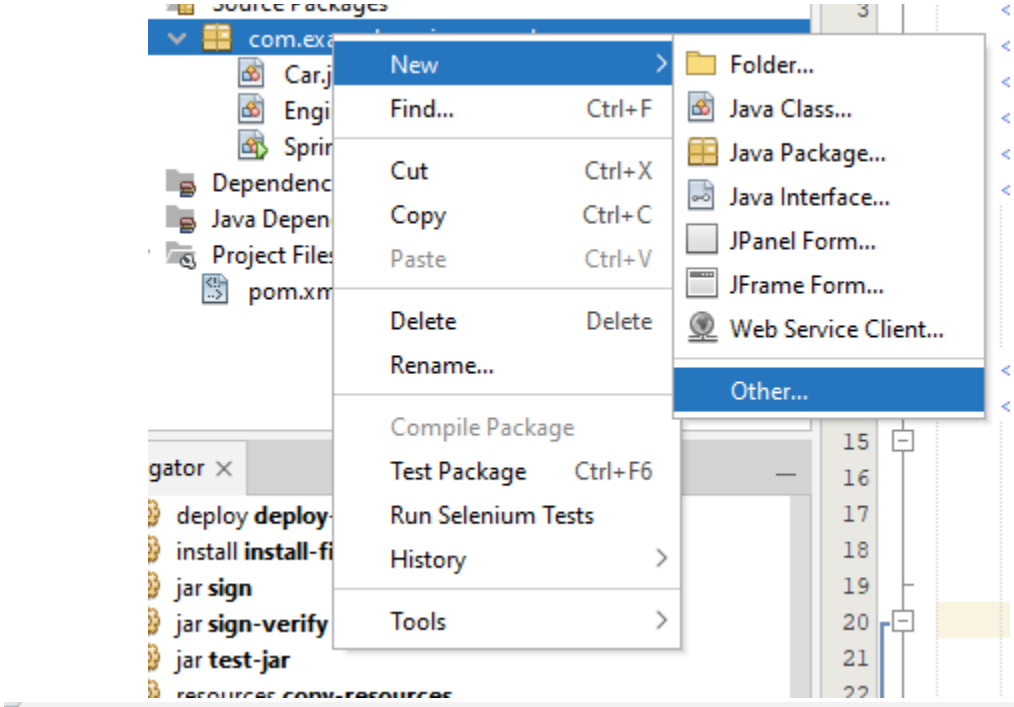
- Di folder **src/main/resources**, buat file XML bernama `applicationContext.xml` dan tambahkan konfigurasi berikut:

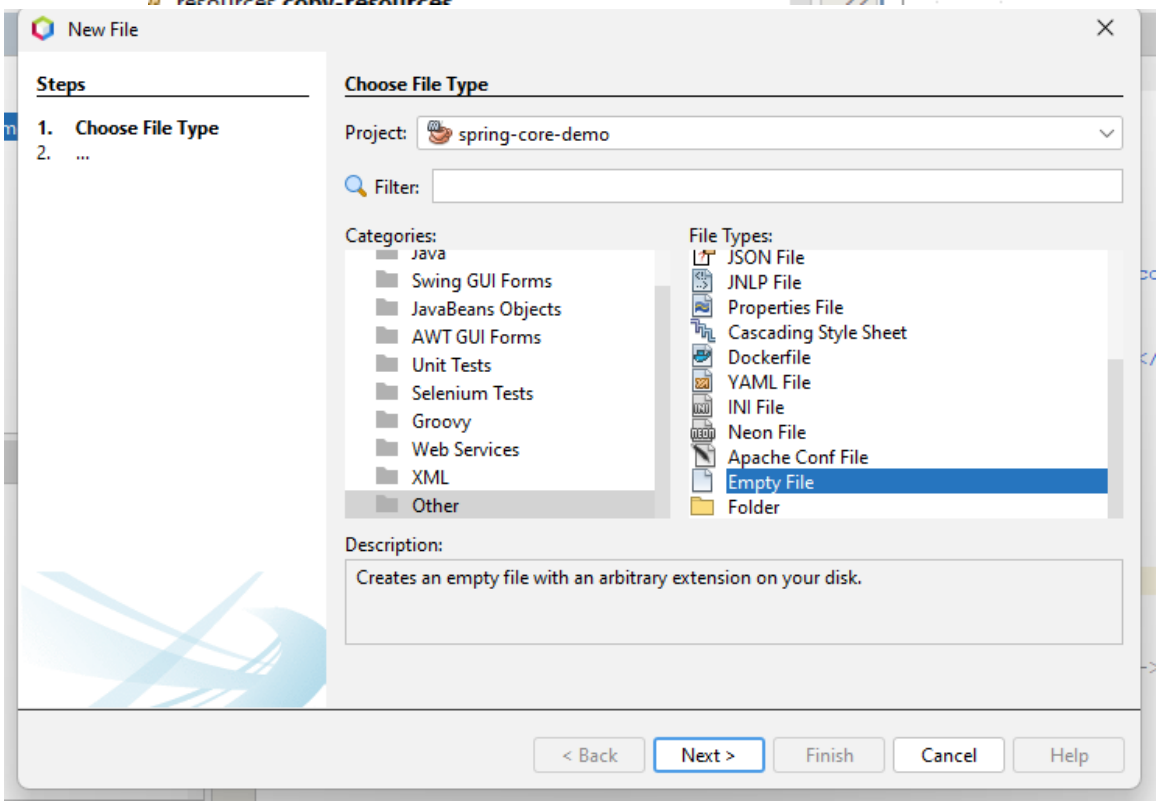
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-
beans.xsd">

  <!-- Konfigurasi bean untuk Engine -->
  <bean id="engine" class="com.example.springcore.Engine">
    <property name="type" value="V8"/>
  </bean>

  <!-- Konfigurasi bean untuk Car -->
  <bean id="car" class="com.example.springcore.Car">
    <property name="engine" ref="engine"/>
  </bean>
```

</beans>





Langkah 2.2: Menggunakan Spring IoC Container

4. **Buat kelas `App.java` untuk memuat Spring IoC Container:**

```
import org.springframework.context.ApplicationContext;
```

```
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {
   public static void main(String[] args) {
      // Load konfigurasi XML ke Spring IoC Container
      ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

      // Ambil bean 'car' dari container
      Car car = (Car) context.getBean("car");
      car.drive();
   }
}
```

5. **Jalankan Proyek**:

- Klik kanan proyek dan pilih **Run**. Jika berhasil, output akan menampilkan

```
Engine type V8 started!
Car is moving.
```

Bagian 3: Implementasi Dependency Injection (DI) dengan Anotasi (45 Menit)

Langkah 3.1: Menggunakan Anotasi untuk Dependency Injection

1. **Modifikasi `Car.java` dan `Engine.java`:**
   - Gunakan anotasi `@Component` dan `@Autowired` untuk menggantikan konfigurasi berbasis XML.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Car {

   private Engine engine;

   @Autowired
   public void setEngine(Engine engine) {
      this.engine = engine;
   }

   public void drive() {
```

```
        engine.start();
        System.out.println("Car is moving.");
    }
}
import org.springframework.stereotype.Component;

@Component
public class Engine {
    private String type = "V6";

    public void start() {
        System.out.println("Engine type " + type + " started!");
    }
}
```

2. Modifikasi `applicationContext.xml` untuk anotasi:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/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">

    <!-- Aktifkan pencarian anotasi -->
    <context:component-scan base-package="com.example.springcore" />

</beans>
```

3. **Modifikasi `App.java`:**

- Tidak ada perubahan, tetapi sekarang Spring IoC Container akan mencari bean berbasis anotasi.

4. **Jalankan Proyek:**

- Klik kanan proyek dan pilih **Run**. Output akan menampilkan:
  Engine type V6 started!
  Car is moving.

D. LAPORAN

1. Penjelasan kode program, capture GUI untuk setiap soal (12 x (@8%)

Referensi Utama:

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