**Case Study 1: Java-Based Configuration**

**Project Title: Online Food Ordering System**

**Configuration Type: Java-based Spring Configuration**

**POJO Classes: Restaurant and Customer**

**Scenario:**

An online food ordering platform allows customers to order food from various restaurants. The

system must manage customer information and restaurant offerings. The logic for selecting

restaurants and placing orders is handled in a service class. Java-based configuration is used to wire

beans explicitly.

**Components:**

**• Customer.java**: Holds customer details like name, contact info, and preferred cuisine.

**• Restaurant.java**: Holds restaurant details like name, location, and available cuisines.

**• FoodOrderService.java**: Service that processes the food order by matching customer

preferences with restaurant availability.

**• AppConfig.java**: A @Configuration class that defines and wires all beans manually

using @Bean methods.

**• MainApp.java**: Initializes the Spring context using

AnnotationConfigApplicationContext and executes the order flow.

**Why Java-Based Config?**

• Useful when full control over bean creation is required.

• Suitable for projects where configuration is centralized and separated from the POJO classes

(which may not be editable).

**Customer.java**

public class Customer {

private String name;

private String contactInfo;

private String preferredCuisine;

public Customer(String name, String contactInfo, String preferredCuisine) {

this.name = name;

this.contactInfo = contactInfo;

this.preferredCuisine = preferredCuisine;

}

public String getName() {

return name;

}

public String getContactInfo() {

return contactInfo;

}

public String getPreferredCuisine() {

return preferredCuisine;

}

}

**Restaurant.java**

import java.util.List;

public class Restaurant {

private String name;

private String location;

private List<String> availableCuisines;

public Restaurant(String name, String location, List<String> availableCuisines) {

this.name = name;

this.location = location;

this.availableCuisines = availableCuisines;

}

public String getName() {

return name;

}

public String getLocation() {

return location;

}

public List<String> getAvailableCuisines() {

return availableCuisines;

}

public boolean servesCuisine(String cuisine) {

return availableCuisines.contains(cuisine);

}

}

**FoodOrderService.java**

public class FoodOrderService {

private Customer customer;

private Restaurant restaurant;

public FoodOrderService(Customer customer, Restaurant restaurant) {

this.customer = customer;

this.restaurant = restaurant;

}

public void processOrder() {

System.out.println("Processing order for customer: " + customer.getName());

String preferredCuisine = customer.getPreferredCuisine();

System.out.println("Preferred cuisine: " + preferredCuisine);

if (restaurant.servesCuisine(preferredCuisine)) {

System.out.println("Restaurant '" + restaurant.getName() + "' serves " + preferredCuisine + ".");

System.out.println("Order placed successfully!");

} else {

System.out.println("No restaurants available that serve " + preferredCuisine + ".");

}

}

}

**AppConfig.java**

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.util.Arrays;

@Configuration

public class AppConfig {

@Bean

public Customer customer() {

return new Customer("Alice", "alice@example.com", "Italian");

}

@Bean

public Restaurant restaurant() {

return new Restaurant("Luigi's", "Downtown", Arrays.asList("Italian", "Mexican", "Indian"));

}

@Bean

public FoodOrderService foodOrderService() {

return new FoodOrderService(customer(), restaurant());

}

}

**MainApp.java**

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

FoodOrderService orderService = context.getBean(FoodOrderService.class);

orderService.processOrder();

context.close();

}

}

**beans.xml**

<project xmlns="<http://maven.apache.org/POM/4.0.0>"

         xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

         xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <https://maven.apache.org/xsd/maven-4.0.0.xsd>">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>

  <artifactId>BankServiceDemo</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <packaging>jar</packaging>

  <dependencies>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-context</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-aop</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>org.aspectj</groupId>

      <artifactId>aspectjweaver</artifactId>

      <version>1.9.20</version>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-compiler-plugin</artifactId>

        <version>3.11.0</version>

        <configuration>

          <source>17</source>

          <target>17</target>

        </configuration>

      </plugin>

    </plugins>

  </build>

</project>

**Case Study 2: Annotation-Based Configuration**

**Project Title: Smart Home Automation System**

**Configuration Type: Annotation-based Spring Configuration**

**POJO Classes: Device and User**

**Scenario:**

A smart home system manages various IoT devices like lights, fans, and ACs. Users can control

these devices through an application. Each user can register and manage multiple devices. Spring

annotations like @Component, @Autowired, and @Service are used to auto-wire

dependencies and manage components.

**Components:**

**• User.java**: Annotated with @Component, contains user details like name and home ID.

**• Device.java**: Annotated with @Component, represents smart devices with attributes like

device type and status.

**• AutomationService.java**: Annotated with @Service, uses @Autowired to inject both

User and Device beans to manage device control logic.

**• AppConfig.java**: A minimal @Configuration class with @ComponentScan to

auto-detect components in the package.

**• MainApp.java**: Loads the context and triggers methods to control devices.

**Why Annotation-Based Config?**

• Reduces boilerplate and simplifies bean wiring.

• Ideal for component-based development where classes are self-contained and annotated.

• Encourages cleaner separation of concerns with automatic scanning and DI.

**User.java**

package com.smarthome.model;

import org.springframework.stereotype.Component;

@Component

public class User {

private String name = "Alice";

private String homeId = "H001";

public String getName() {

return name;

}

public String getHomeId() {

return homeId;

}

@Override

public String toString() {

return "User{name='" + name + "', homeId='" + homeId + "'}";

}

}

**Device.java**

package com.smarthome.model;

import org.springframework.stereotype.Component;

@Component

public class Device {

private String deviceType = "Light";

private boolean status = false;

public String getDeviceType() {

return deviceType;

}

public boolean isStatus() {

return status;

}

public void toggleStatus() {

status = !status;

}

@Override

public String toString() {

return "Device{deviceType='" + deviceType + "', status=" + (status ? "ON" : "OFF") + "}";

}

}

**AutomationService.java**

package com.smarthome.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.smarthome.model.User;

import com.smarthome.model.Device;

@Service

public class AutomationService {

@Autowired

private User user;

@Autowired

private Device device;

public void controlDevice() {

System.out.println("User Info: " + user);

System.out.println("Device Before Action: " + device);

device.toggleStatus();

System.out.println("Device After Action: " + device);

}

}

**AppConfig.java**

package com.smarthome;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan(basePackages = "com.smarthome")

public class AppConfig {

}

**MainApp.java**

package com.smarthome;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.smarthome.service.AutomationService;

public class MainApp {

public static void main(String[] args) {

AnnotationConfigApplicationContext context =

new AnnotationConfigApplicationContext(AppConfig.class);

AutomationService service = context.getBean(AutomationService.class);

service.controlDevice();

context.close();

}

}

**beans.xml**

<project xmlns="<http://maven.apache.org/POM/4.0.0>"

         xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

         xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <https://maven.apache.org/xsd/maven-4.0.0.xsd>">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>

  <artifactId>BankServiceDemo</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <packaging>jar</packaging>

  <dependencies>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-context</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-aop</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>org.aspectj</groupId>

      <artifactId>aspectjweaver</artifactId>

      <version>1.9.20</version>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-compiler-plugin</artifactId>

        <version>3.11.0</version>

        <configuration>

          <source>17</source>

          <target>17</target>

        </configuration>

      </plugin>

    </plugins>

  </build>

</project>