DAY 8 SPRING CLASS ASSIGNMENT

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

```
package com.example.foodorder;

public class Customer {
    private String name;
    private String contact;
    private String preferredCuisine;

    public Customer(String name, String contact, String preferredCuisine) {
        this.name = name;
        this.contact = contact;
        this.preferredCuisine = preferredCuisine;
    }

    public String getPreferredCuisine() {
        return preferredCuisine;
    }

    public String getName() {
        return name;
    }

    public String getContact() {
        return contact;
    }
}
```

Hotel.java

```
package com.example.foodorder;

import java.util.List;

public class Hotel {
```

```java
    private String name;
    private String location;
    private List<String> cuisines;

    public Hotel(String name, String location, List<String> cuisines) {
        this.name = name;
        this.location = location;
        this.cuisines = cuisines;
    }

    public boolean servesCuisine(String cuisine) {
        return cuisines.contains(cuisine);
    }

    public String getName() {
        return name;
    }

    public String getLocation() {
        return location;
    }
}
```

FoodOrderService.java

```java
package com.example.foodorder;

import java.util.List;
```

```
public class FoodOrderService {
    private Customer customer;
    private List<Hotel> Hotels;

    public FoodOrderService(Customer customer, List<Hotel> restaurants) {
        this.customer = customer;
        this.Hotels = restaurants;
    }

    public void processOrder() {
        System.out.println("Searching restaurants for " + customer.getName() +
                " (Cuisine: " + customer.getPreferredCuisine() + ")");

        boolean found = false;
        for (Hotel r : Hotels) {
            if (r.servesCuisine(customer.getPreferredCuisine())) {
                System.out.println("Found: " + r.getName() + " at " + r.getLocation());
                found = true;
            }
        }
        if (!found) {
            System.out.println("No restaurants found serving " + customer.getPreferredCuisine());
        }
    }
}
```

AppConfig.java

```java
package com.example.foodorder;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Arrays;
import java.util.List;

@Configuration
public class AppConfig {

    @Bean
    public Customer customer() {
        return new Customer("Sanjay", "999-888-7777", "Indian");
    }

    @Bean
    public Hotel r1() {
        return new Hotel("Spice Hub", "Chennai", Arrays.asList("Indian", "Chinese"));
    }

    @Bean
    public Hotel r2() {
        return new Hotel("Pasta Palace", "Delhi", Arrays.asList("Italian", "Continental"));
    }
    @Bean
    public List<Hotel> Hotels() {
        return Arrays.asList(r1(), r2());
    }

    @Bean
    public FoodOrderService foodOrderService() {
        return new FoodOrderService(customer(), Hotels());
    }
}
```

Main.java

```
package com.example.foodorder;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

        FoodOrderService service = context.getBean(FoodOrderService.class);
        service.processOrder();
    }
}
```

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.example.smarthome;


import org.springframework.stereotype.Component;


@Component
public class User {
    private String name = "Sanjay";
    private String homeId = "HOME123";
    public String getName() {
        return name;
    }
    public String getHomeId() {
        return homeId;
    }
}
```

Device.java

```
package com.example.smarthome;

import org.springframework.stereotype.Component;

@Component
public class Device {
    private String type = "Light";
    private boolean status = false;
    public void turnOn() {
        status = true;
        System.out.println(type + " is ON.");
    }
    public void turnOff() {
        status = false;
        System.out.println(type + " is OFF.");
    }
    public String getType() {
        return type;
    }
    public boolean getStatus() {
        return status;
    }
}
```

AutomationService.java

```
package com.example.smarthome;

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

@Service
public class AutomationService {

    @Autowired
    private User user;

    @Autowired
    private Device device;

    public void controlDevice() {
        System.out.println("User " + user.getName() + " is controlling device: " +
device.getType());

        device.turnOn();
```

```java
        System.out.println("Device status: " + (device.getStatus() ? "ON" : "OFF"));
    }
}
```

AppConfig.java

```java
package com.example.smarthome;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("com.example.smarthome")
public class AppConfig {
}
```

MainApp.java

```java
package com.example.smarthome;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
```

```
        AutomationService service = context.getBean(AutomationService.class);
        service.controlDevice();
    }
}
```