Spring Class Case Studies[Day-09]

Case Study-1: Create Product POJO Class-Product name,Product Description,Create Beans.xml file and conifigure that in App.java

Product.java

```
package com.example.spring_hello_world_demo;


public class Product {
	String productName;
	String productDescription;
	public void getProductName() {
		System.out.println(productName);
	}
	public void setProductName(String productName) {
		this.productName = productName;
	}
	public void getProductDescription() {
		System.out.println(productDescription);
	}
	public void setProductDescription(String productDescription) {
		this.productDescription = productDescription;
	}
}
```

Beans.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
      https://www.springframework.org/schema/beans/spring-beans.xsd">

        <bean id="product" class="com.example.spring_hello_world_demo.Product">
```

```
        <property name="productName" value="Laptop"/>
        <property name="productDescription" value="hp intel i5"/>
    </bean>
</beans>
```

App.java:

```
package com.example.spring_hello_world_demo;


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

public class App {

    public static void main(String[] args) {
        //SpringApplication.run(SpringHelloWorldDemoApplication.class, args);
        ApplicationContext context=new ClassPathXmlApplicationContext("beans.xml");
        Product obj=(Product) context.getBean("product");
        obj.getProductName();
        obj.getProductDescription();
    }

}
```

Output:

```
Laptop
hp intel i5
```

Case Study 2: 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).

Restaurant.java

```
package com.example.onlinefoodordering;


import java.util.List;
```

```
public class Restuarant {
    private String name;
    private String location;
    private List<String> availableCuisines;

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

    public String getName() {
        return name;
    }

    public String getLocation() {
        return location;
    }

    public List<String> getAvailableCuisines() {
        return availableCuisines;
    }
}
```

Customer.java

```
package com.example.onlinefoodordering;

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 getName() {
        return name;
    }

    public String getContact() {
        return contact;
    }

    public String getPreferredCuisine() {
        return preferredCuisine;
    }
}
```

FoodOrderService.java

```
package com.example.onlinefoodordering;

import java.util.List;

public class FoodOrderService{
private List<Restuarant> restaurants;

public FoodOrderService(List<Restuarant> restaurants) {
    this.restaurants = restaurants;
}
```

```
public void placeOrder(Customer customer) {
    System.out.println("Processing order for: " + customer.getName());
    boolean found = false;
    for (Restuarant r : restaurants) {
        if (r.getAvailableCuisines().contains(customer.getPreferredCuisine())) {
            System.out.println("Restaurant matched: " + r.getName() + " at " + r.getLocation());
            found = true;
        }
    }
    if (!found) {
        System.out.println("No restaurants found for cuisine: " + customer.getPreferredCuisine());
    }
}
}
```

AppConfig.java

```
package com.example.onlinefoodordering;

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

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

@Configuration
public class AppConfig {

    @Bean
    public Customer customer() {
        return new Customer("Hari", "hari@example.com", "Indian");
    }
```

```
    @Bean
    public Restuarant r1() {
        return new Restuarant("Bawarchi", "Downtown", Arrays.asList("Italian", "Mexican"));
    }

    @Bean
    public Restuarant r2() {
        return new Restuarant("Curry House", "Uptown", Arrays.asList("Indian", "Thai"));
    }

    @Bean
    public List<Restuarant> restaurants() {
        return Arrays.asList(r1(), r2());
    }

    @Bean
    public FoodOrderService foodOrderService() {
        return new FoodOrderService(restaurants());
    }
}
```

MainApp.java:

```
package com.example.onlinefoodordering;

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

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

```
        Customer customer = context.getBean(Customer.class);
        FoodOrderService service = context.getBean(FoodOrderService.class);

        service.placeOrder(customer);
        ((AbstractApplicationContext) context).close();
    }
}
```

OUTPUT:

Processing order for: Alice

Restaurant matched: Pizza Palace at Downtown

Case Study 3: 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.homeautomation;


import org.springframework.stereotype.Component;


@Component
public class User {
	private String name;
	private int homeId;


	public String getName() {
		return name;
	}
	public int getHomeId() {
		return homeId;
	}
	@Override
	public String toString() {
		return "User [name=" + name + ", homeId=" + homeId + "]";
	}
```

}

Device.java

```
package com.example.homeautomation;


import org.springframework.stereotype.Component;


@Component
public class Device {
	private String deviceType;
	private String deviceStatus;
	public String getDeviceType() {
		return deviceType;
	}
	public void setDeviceType(String deviceType) {
		this.deviceType = deviceType;
	}
	public String getDeviceStatus() {
		return deviceStatus;
	}
	public void setDeviceStatus(String deviceStatus) {
		this.deviceStatus = deviceStatus;
	}
	public void turnOn() {
		deviceStatus="ON";
		System.out.println(deviceType+"is"+deviceStatus);


	}
	@Override
	public String toString() {
		return "Device [deviceType=" + deviceType + ", deviceStatus=" + deviceStatus + "]";
```

```
	}
	public void turnOff() {
		deviceStatus="OFF";
		System.out.println(deviceType+"is"+deviceStatus);
	}

}
```

AutomationService.java

```
package com.example.homeautomation;

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

@Configuration
@ComponentScan(basePackages="com.example.homeautomation")
public class AppConfig {

}
```

MainApp.java

```
package com.example.homeautomation;


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

public class MainApp {

	public static void main(String[] args) throws BeansException	{
		// TODO Auto-generated method stub
```

```
                ApplicationContext context=new
AnnotationConfigApplicationContext(AppConfig.class);
                AutomationService service=context.getBean(AutomationService.class);
                service.controlDevice();


        }


}
```