Day_9 class Assignmnet:

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.food;


import org.springframework.stereotype.Component;


@Component
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;
            }
}
```

Restaurent.java:

```
package com.example.food;

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;
         }
}
```

FoodOrderService.java:

```
package com.example.food;

import java.util.List;

public class FoodOrderService {

         private List<Restaurant> restaurants;
```

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

    public void placeOrder(Customer customer) {
        System.out.println("Placing order for " + customer.getName());

        for (Restaurant r : restaurants) {
            if (r.getAvailableCuisines().contains(customer.getPreferredCuisine())) {
                System.out.println("Found matching restaurant: " + r.getName());
                System.out.println("Order placed successfully!");
                return;
            }
        }

        System.out.println("No restaurant found for preferred cuisine: " +
customer.getPreferredCuisine());
    }
}
```

Mainapp.java:

```
package com.example.food;

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);


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


      service.placeOrder(customer);
   }
}
```

AppConfig.java:

```
package com.example.food;


import java.util.Arrays;


import org.springframework.context.annotation.Bean;
public class AppConfig {
  @Bean
  public Customer customer() {
     return new Customer("Alice", "alice@example.com", "Italian");
   }


    @Bean
    public Restaurant r1() {
        return new Restaurant("Pasta Palace", "Downtown", Arrays.asList("Italian",
"Continental"));
    }
    @Bean
    public Restaurant r2() {
        return new Restaurant("Spice Route", "Midtown", Arrays.asList("Indian", "Thai"));
   }

    @Bean
            public FoodOrderService foodOrderService() {
               return new FoodOrderService(Arrays.asList(r1(), r2()));
     }
}
```

Pom.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.travel</groupId>
  <artifactId>travel-booking-system</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
      <maven.compiler.source>17</maven.compiler.source>
      <maven.compiler.target>17</maven.compiler.target>
   </properties>

   <dependencies>
      <!-- Spring Core (includes BeanFactory and ApplicationContext) -->
      <dependency>
```

```xml
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.30</version>
    </dependency>

    <!-- Apache Commons Logging (used internally by Spring) -->
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
                <!-- <dependency>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-beans</artifactId>
                    <version>7.0.0-M7</version>
                </dependency> -->

</dependencies>

<build>
    <plugins>
        <!-- Compiler Plugin to support Java 17 -->
        <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.example.Home;

import org.springframework.stereotype.Component;

@Component
public class User {
    private String name = "sai";
    private String homeId = "H123";

    public String getName() { return name; }
    public String getHomeId() { return homeId; }
}
```

Device.java:

```
package com.example.Home;

import org.springframework.stereotype.Component;

@Component
public class Device {
    private String deviceType = "Light";

    public void turnOn() {
            System.out.println(deviceType + "is on");
    }

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

AutomationService:

```
package com.example.Home;

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 AutomationService(User user, Device device) {
            this.user=user;
            this.device=device;

    }
```

```
   public void controlDevice(String action) {
          System.out.println("User " +user.getName() + " (" + user.getHomeId() + ") is
controlling the device");
          if("on".equalsIgnoreCase(action)) {
                 device.turnOff();
          } else if ("off".equalsIgnoreCase(action)) {
                 device.turnOff();
          }
          else {
                 System.out.println("Invalid action");



          }
   }
}
```

App.Config:

```
package com.example.Home;

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

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

MainApp:

```
package com.example.Home;
```

```
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();

    }

}
```

Pom.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.travel</groupId>

  <artifactId>travel-booking-system</artifactId>

  <version>0.0.1-SNAPSHOT</version>


  <properties>

        <maven.compiler.source>17</maven.compiler.source>

        <maven.compiler.target>17</maven.compiler.target>

    </properties>


    <dependencies>

        <!-- Spring Core (includes BeanFactory and ApplicationContext) -->

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.30</version>
        </dependency>

        <!-- Apache Commons Logging (used internally by Spring) -->
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
                    <!-- <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-beans</artifactId>
                        <version>7.0.0-M7</version>
                    </dependency> -->

    </dependencies>

    <build>
        <plugins>
            <!-- Compiler Plugin to support Java 17 -->
            <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>
```

Casestudy3(morning):

Case study: Create Product POJO class - product name, product description , Create Beans.xml file and configure that in App.java

Product java:

```
package com.example.product;
public class Product {
    private String name;
     private String description;
      public Product() {
        }
       public String getName() {
           return name; }
       public void setName(String name) {
             this.name = name; }
       public String getDescription() {
               return description; }
        public void setDescription(String description) {
            this.description = description; }
```

```
          public void display() {
                  System.out.println("Product Name: " + name);
                   System.out.println("Product Description: " + description);
      }
 }
```

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.product ">
                    <property name=”name”value=”car”>
                      <property name="description” value="It is a Benz car new model" />
          </bean>



</beans>
```

SpringProductDemo.java Code:

```
package com.example.product;
import org.springframework.context.ApplicationContext;
 import org.springframework.context.support.ClassPathXmlApplicationContext;
public class SpringProductDemo {
          public static void main(String[] args) {
                ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
Product product = (Product) context.getBean("productBean");

          product.display();

      }

 }
```