Day 11

Name : DHANAPAL

Date : 08/08/2024

Java and Microservices

1. Create Class named Employee program with class variables as companyName, instance variables with employeeName, employeeID , employeeSalary.
2. Use Data Encapsulation and use getters and setters for updating the employeeSalary
3. Show function overloading to calculate salary of employee with bonus and salary of employee with deduction.

Solution for Question 1,2&3:

Encapsulation: The class variables are private, and the class provides public getters and setters to access and update `employeeSalary`.

Static Variable: `companyName` is a static variable since it is common across all employees.

Function Overloading: There are two `calculateSalary` methods, one that adds a bonus and another that subtracts a deduction, demonstrating function overloading.

Program:

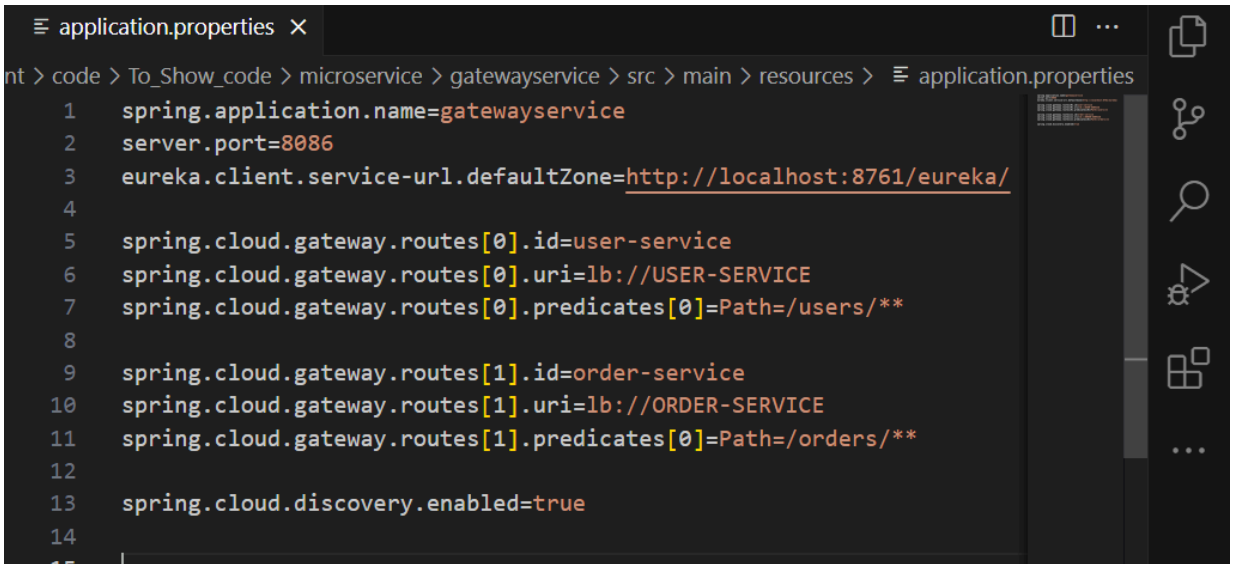
```java
package tesst;
public class Employee {
  // Class variables (companyName as a static variable)
  private static String companyName = "Payoda";
  private String employeeName;
  private int employeeID;
  private double employeeSalary;
  // Constructor to initialize employee details
  public Employee(String employeeName, int employeeID, double employeeSalary) {
    this.employeeName = employeeName;
    this.employeeID = employeeID;
    this.employeeSalary = employeeSalary;
  }
  // Getter and setter for employeeSalary
  public double getEmployeeSalary() {
    return employeeSalary;
  }
  public void setEmployeeSalary(double employeeSalary) {
    this.employeeSalary = employeeSalary;
  }
  // Getter for employeeName
  public String getEmployeeName() {
    return employeeName;
  }
  // Getter for employeeID
  public int getEmployeeID() {
    return employeeID;
  }
  // Static method to get the company name
  public static String getCompanyName() {
    return companyName;
  }
  // Overloaded method to calculate salary with bonus
  public double calculateSalary(double bonus) {
    return employeeSalary + bonus;
  }
  // Overloaded method to calculate salary with deduction
```

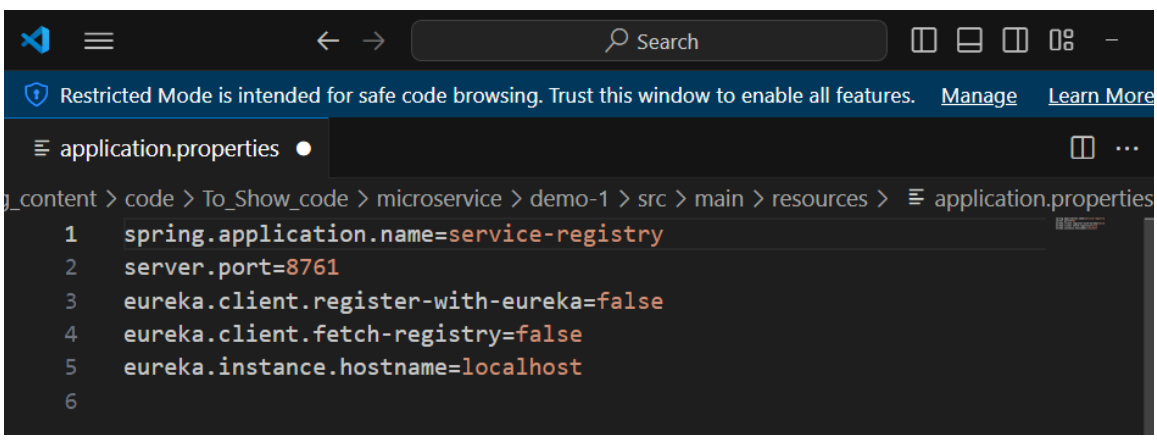
```java
  public double calculateSalary(double deduction, boolean isDeduction) {
    return employeeSalary - deduction;
  }
  public static void main(String[] args) {
    // Creating an employee object with name Dhanapal and company Payoda
    Employee emp = new Employee("Dhanapal", 101, 50000);
    // Displaying employee details
    System.out.println("Company: " + Employee.getCompanyName());
    System.out.println("Employee Name: " + emp.getEmployeeName());
    System.out.println("Employee ID: " + emp.getEmployeeID());
    System.out.println("Employee Salary: " + emp.getEmployeeSalary());
    // Calculating salary with bonus
    double salaryWithBonus = emp.calculateSalary(5000);
    System.out.println("Salary with Bonus: " + salaryWithBonus);
    // Calculating salary with deduction
    double salaryWithDeduction = emp.calculateSalary(2000, true);
    System.out.println("Salary with Deduction: " + salaryWithDeduction);
  }
}
```

Output:

```
Company: Payoda
Employee Name: Dhanapal
Employee ID: 101
Employee Salary: 50000.0
Salary with Bonus: 55000.0
Salary with Deduction: 48000.0
```

4. What are the Microservices – that use this Gateway and Service Discovery methods using below screenshot:





1. Gateway Service:

- The first screenshot shows the configuration for a Gateway Service that uses Spring Cloud Gateway.
- Routes Configuration:
  - It defines routes for two services: USER-SERVICE and ORDER-SERVICE.

    - These services are routed based on the path. For example, requests to /users/** are routed to USER-SERVICE and requests to /orders/** are routed to ORDER-SERVICE.
- Service Discovery: The eureka.client.service-url.defaultZone points to a Eureka server (http://localhost:8761/eureka/), indicating that this gateway service uses Eureka for service discovery.

2. Service Registry (Eureka Server):

- The second screenshot shows the configuration of a Service Registry using Eureka.
- Service Name: service-registry
- Eureka Configuration:
    - The eureka.client.register-with-eureka and eureka.client.fetch-registry are set to false, indicating that this instance is acting as a Eureka server (service registry) and not as a client.

Summary:

- Gateway Service: Acts as a central point for routing requests to different microservices (USER-SERVICE, ORDER-SERVICE) based on predefined routes.
- Service Discovery: Eureka Server (service-registry) is used to manage the service instances and allow services to discover each other.