Name:Siddarth
Date: 7-9-2024

Java and Microservices

questions:
1. Create a 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:
ANS:1,2,3

```
public class Employee {

    private static String companyName;
    private String employeeName;
    private int employeeID;
    private double employeeSalary;

    public Employee(String name, int id, double salary) {
        this.employeeName = name;
        this.employeeID = id;
        this.employeeSalary = salary;
    }

    public double getEmployeeSalary() {
        return employeeSalary;
    }

    public void setEmployeeSalary(double employeeSalary) {
        if (employeeSalary > 0) {
            this.employeeSalary = employeeSalary;
        } else {
```

```
        System.out.println("Salary cannot be negative!");
    }
}

public static void setCompanyName(String name) {
    companyName = name;
}

public static String getCompanyName() {
    return companyName;
}

public double calculateSalary(double bonus) {
    return this.employeeSalary + bonus;
}

public double calculateSalary(double deduction, boolean isDeduction) {
    if (isDeduction) {
        return this.employeeSalary - deduction;
    } else {
        return this.employeeSalary + deduction;
    }
}

public void displayEmployeeDetails() {
    System.out.println("Company Name: " + companyName);
    System.out.println("Employee Name: " + employeeName);
    System.out.println("Employee ID: " + employeeID);
    System.out.println("Employee Salary: " + employeeSalary);
}

public static void main(String[] args) {
    Employee.setCompanyName("Google");
    Employee emp1 = new Employee("Siddarth", 101, 50000);
    emp1.displayEmployeeDetails();
    emp1.setEmployeeSalary(55000);
    System.out.println("Updated Salary: " + emp1.getEmployeeSalary());
    double salaryWithBonus = emp1.calculateSalary(5000);
```

```
        System.out.println("Salary with Bonus: " + salaryWithBonus);
        double salaryWithDeduction = emp1.calculateSalary(2000, true);
        System.out.println("Salary with Deduction: " + salaryWithDeduction);
    }
}
```

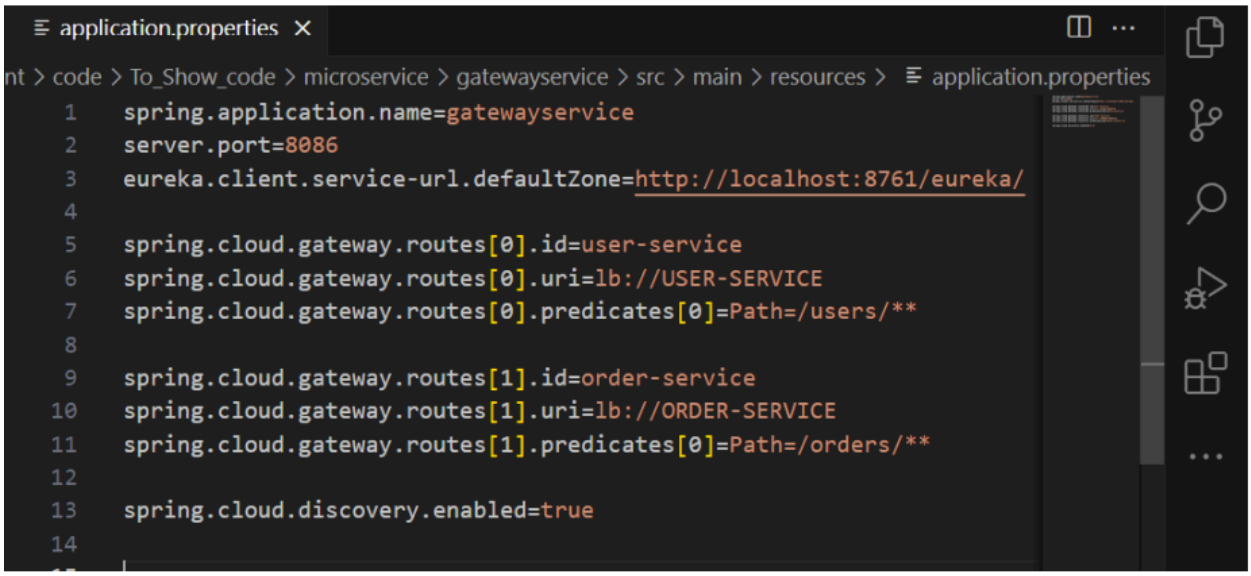
Output:

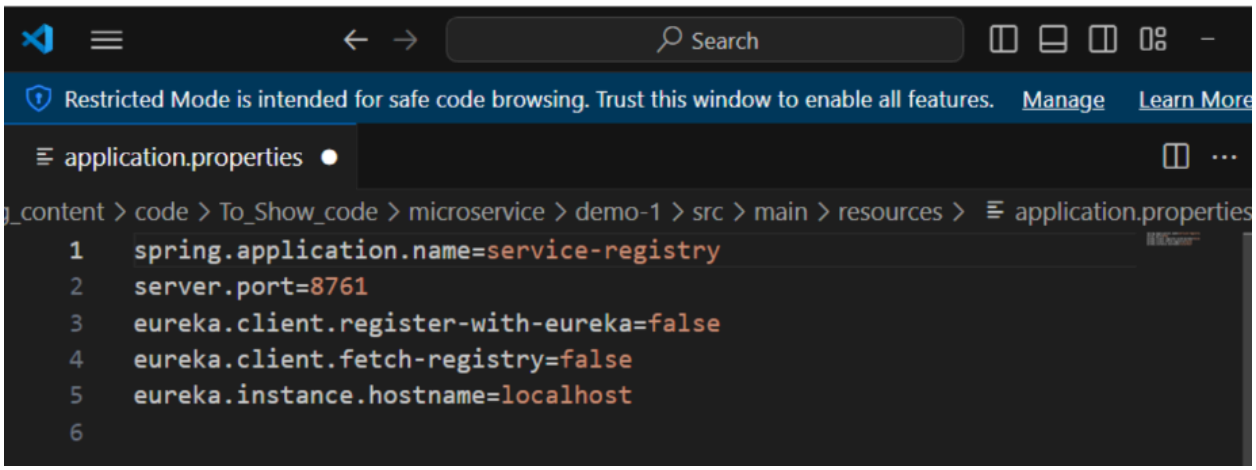
Company Name: Google
Employee Name: Siddarth
Employee ID: 101
Employee Salary: 50000.0
Updated Salary: 55000.0
Salary with Bonus: 60000.0
Salary with Deduction: 53000.0

Explanation:

- **Data Encapsulation:** The class variables are marked `private` and can only be accessed or modified through public getter and setter methods.
- **Method Overloading:** The `calculateSalary` method is overloaded to handle different scenarios: one with a bonus and another with a deduction.
- **Static Variable:** `companyName` is a static variable, meaning it's shared across all instances of the class.

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





1. Gateway Service:

Screenshot Explanation:

- The first screenshot shows the configuration for a Gateway Service using **Spring Cloud Gateway**.
- **Routes Configuration:**
  - Two services are defined: `USER-SERVICE` and `ORDER-SERVICE`.
  - The routes are based on the URL path:
    - Requests to `/users/**` are routed to `USER-SERVICE`.
    - Requests to `/orders/**` are routed to `ORDER-SERVICE`.

    - This configuration enables the gateway to act as a central point of access, directing traffic to the appropriate microservice based on the request path.

Service Discovery:

- The `eureka.client.service-url.defaultZone` is set to `http://localhost:8761/eureka/`.
- This indicates that the Gateway Service is using **Eureka** for service discovery, meaning it communicates with the Eureka server to locate the instances of `USER-SERVICE` and `ORDER-SERVICE` dynamically.

2. Service Registry (Eureka Server):

Screenshot Explanation:

- The second screenshot shows the configuration of a Service Registry using **Eureka**.
- **Service Name:** The service registry is named `service-registry`.
- **Eureka Configuration:**
    - The properties `eureka.client.register-with-eureka` and `eureka.client.fetch-registry` are set to `false`.
    - This indicates that the current instance is acting as a Eureka Server and not as a client.
    - As a Eureka Server, it functions as a service registry where other microservices (like `USER-SERVICE` and `ORDER-SERVICE`) register themselves.

Summary:

- The **Gateway Service** acts as a traffic director, routing requests to appropriate microservices based on the URL path, with routes configured for `USER-SERVICE` and `ORDER-SERVICE`.
- **Eureka** is used for service discovery, allowing the Gateway to dynamically locate services without hardcoding their addresses.
- The **Service Registry** configuration in Eureka indicates it's operating as the central directory where microservices register and discover each other.