

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.

CODE:

```
package com.pvt;

class Employee {

    private static String companyName = "ABC Corp";

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

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

    public double getEmployeeSalary() {
        return employeeSalary;
    }

    public void setEmployeeSalary(double newSalary) {
        if (newSalary > 0) {
            this.employeeSalary = newSalary;
        } else {
            System.out.println("Salary must be positive.");
        }
    }

    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 double calculateSalary(double bonus) {
        return employeeSalary + bonus;
    }

    public double calculateSalary(double bonus, double deduction) {
        return employeeSalary + bonus - deduction;
    }
}
```

```

public static void main(String[] args) {
    Employee employee1 = new Employee("John Doe", 101, 50000);
    employee1.displayEmployeeDetails();

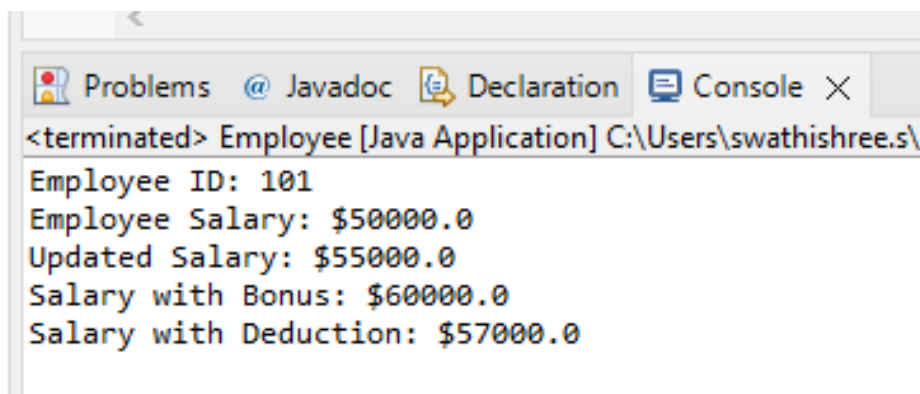
    employee1.setEmployeeSalary(55000);
    System.out.println("Updated Salary: $" + employee1.getEmployeeSalary());

    double salaryWithBonus = employee1.calculateSalary(5000);
    System.out.println("Salary with Bonus: $" + salaryWithBonus);

    double salaryWithDeduction = employee1.calculateSalary(5000, 3000);
    System.out.println("Salary with Deduction: $" + salaryWithDeduction);
}
}

```

OUTPUT:

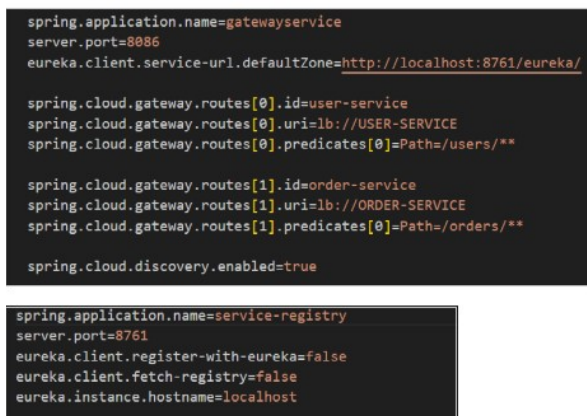


```

<terminated> Employee [Java Application] C:\Users\swathishree.s\
Employee ID: 101
Employee Salary: $50000.0
Updated Salary: $55000.0
Salary with Bonus: $60000.0
Salary with Deduction: $57000.0

```

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



```

spring.application.name=gateway-service
server.port=8086
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

spring.cloud.gateway.routes[0].id=user-service
spring.cloud.gateway.routes[0].uri=lb://USER-SERVICE
spring.cloud.gateway.routes[0].predicates[0]=Path=/users/**

spring.cloud.gateway.routes[1].id=order-service
spring.cloud.gateway.routes[1].uri=lb://ORDER-SERVICE
spring.cloud.gateway.routes[1].predicates[0]=Path=/orders/**

spring.cloud.discovery.enabled=true

```

```

spring.application.name=service-registry
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.instance.hostname=localhost

```

Answer:

Overall Architecture:

- Service Registry (Eureka Server):
 - Name: service-registry
 - Port: 8761
 - Role: This is a Eureka Server, which acts as a Service Registry. It maintains a list of all available microservices and their instances, allowing for dynamic discovery of services. Other microservices and the gateway will register themselves with this Eureka server, making it possible to discover and communicate with each other.
- API Gateway (Spring Cloud Gateway):
 - Name: gateway-service
 - Port: 8086
 - Role: The API Gateway acts as a single entry point for all client requests. It routes incoming requests to the appropriate microservice based on predefined routes and predicates.

Microservices Configuration:

- USER-SERVICE:
 - Routing Configuration:
 - Gateway Route ID: user-service
 - URI: lb://USER-SERVICE
 - Path Predicate: Path=/users/**
 - Description:
 - The USER-SERVICE is a microservice that is responsible for user-related operations, such as user registration, login, profile management, etc.
 - The route in the gateway is configured to forward any requests with the path /users/** to this microservice.
 - The lb:// prefix indicates that the gateway will use Eureka's load-balancing capabilities to route the requests to one of the available instances of USER-SERVICE.
- ORDER-SERVICE:
 - Routing Configuration:
 - Gateway Route ID: order-service
 - URI: lb://ORDER-SERVICE
 - Path Predicate: Path=/orders/**
 - Description:
 - The ORDER-SERVICE handles operations related to orders, such as placing, viewing, or updating orders.
 - The route in the gateway forwards any requests with the path /orders/** to this microservice.
 - Similar to USER-SERVICE, the lb:// prefix is used for load-balancing across instances of ORDER-SERVICE.

Spring Cloud Gateway Configuration:

- **Service Discovery:** The gateway is integrated with Eureka (`eureka.client.service-url.defaultZone=http://localhost:8761/eureka/`), which allows it to dynamically discover the microservices (USER-SERVICE and ORDER-SERVICE) and route the requests accordingly.
- **Discovery Enabled:** The property `spring.cloud.discovery.enabled=true` confirms that the gateway will rely on Eureka for service discovery instead of using static service URLs.

Summary of Communication Flow:

- **Client Requests:** Clients send requests to the API Gateway (gateway-service).
- **Routing:**
 - If the request URL matches `/users/**`, it is routed to USER-SERVICE.
 - If the request URL matches `/orders/**`, it is routed to ORDER-SERVICE.
- **Service Discovery:** The gateway uses Eureka to find the appropriate instances of the microservices (USER-SERVICE, ORDER-SERVICE) to handle the requests, enabling load balancing and fault tolerance.
- **Service Interaction:** The microservices might also register themselves with Eureka and communicate with each other via Eureka, allowing them to scale independently and discover each other dynamically.