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.

CODE:

```
package com.profile;

import java.util.Scanner;

public class Employee {
   // Class variable
   private static String companyName = "Payoda";

   // Instance variables
   private String employeeName;
   private int employeeID;
   private double employeeSalary;

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

   public static String getCompanyName() {
            return companyName;
         }

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

         public String getEmployeeName() {
            return employeeName;
         }

         public void setEmployeeName(String employeeName) {
            this.employeeName = employeeName;
         }
```

```
        public int getEmployeeID() {
                return employeeID;
        }

        public void setEmployeeID(int employeeID) {
                this.employeeID = employeeID;
        }

// Getter for employeeSalary
public double getEmployeeSalary() {
   return employeeSalary;
}

// Setter for employeeSalary
public void setEmployeeSalary(double employeeSalary) {

   this.employeeSalary = employeeSalary;
}

// Method overloading to calculate salary with bonus
public double calculateSalary(double bonus) {
   return this.employeeSalary + bonus;
}

// Method overloading to calculate salary with deduction
public double calculateSalary(int deduction) {
   return this.employeeSalary - deduction;
}

// Main method to test the Employee class
public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Employee Name");
        String ename=sc.next();
        System.out.println("Enter the Employee Id");
        int id=sc.nextInt();
        System.out.println("Enter the Employee Salary");
        double salary=sc.nextDouble();
   Employee employee = new Employee(ename,id,salary);
   System.out.println("Employee Name : "+employee.getEmployeeName());
   System.out.println("Employee Id : "+employee.getEmployeeID());
   System.out.println("Company Name : "+Employee.companyName);
   // Getting the salary
   System.out.println("Current Salary: " + employee.getEmployeeSalary());

   // Setting the salary
   employee.setEmployeeSalary(55000);
```

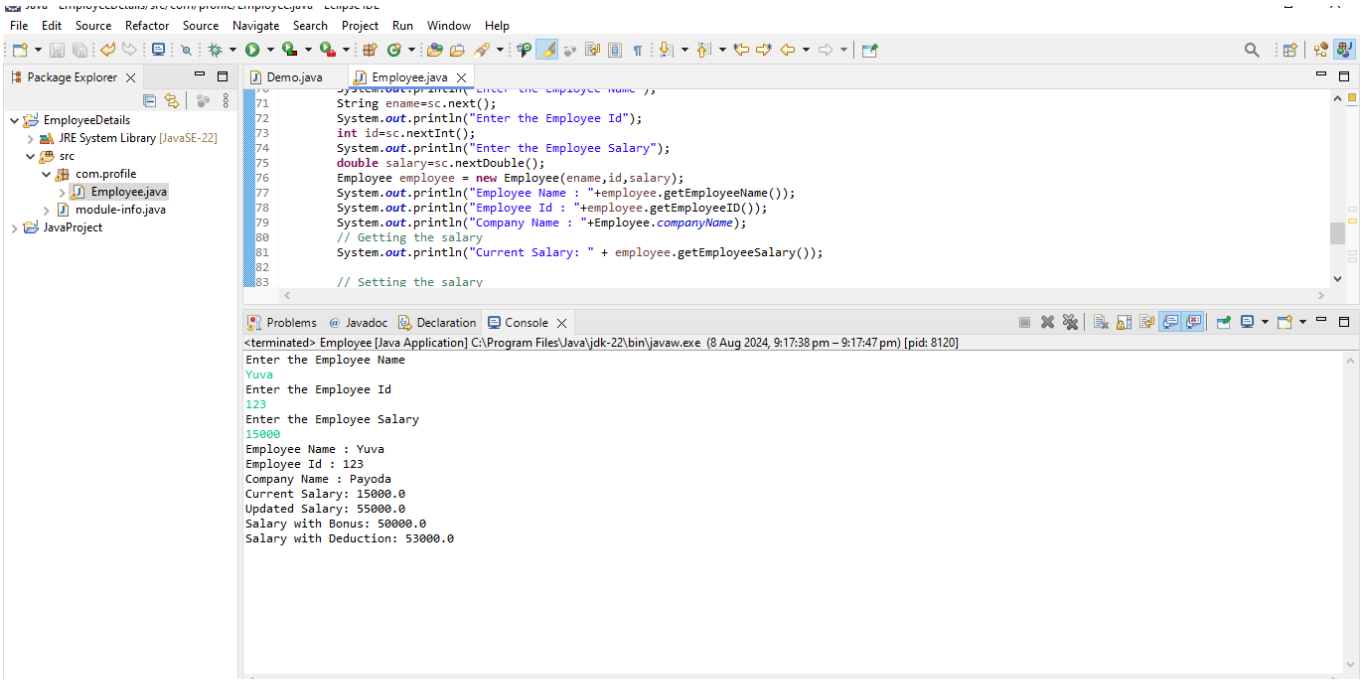
```
        System.out.println("Updated Salary: " + employee.getEmployeeSalary());

        // Calculating salary with bonus
        double bonusSalary = employee.calculateSalary(5000);
        System.out.println("Salary with Bonus: " + bonusSalary);

        // Calculating salary with deduction
        double deductionSalary = employee.calculateSalary(2000);
        System.out.println("Salary with Deduction: " + deductionSalary);
    }


}
```

OUTPUT:



4. What are the Microservices – that use this Gateway and Service Discovery methods

```
spring.application.name=gatewayservice
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
```

using below screen shot:

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**: gatewayservice
  - **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 (gatewayservice).
- **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.