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?

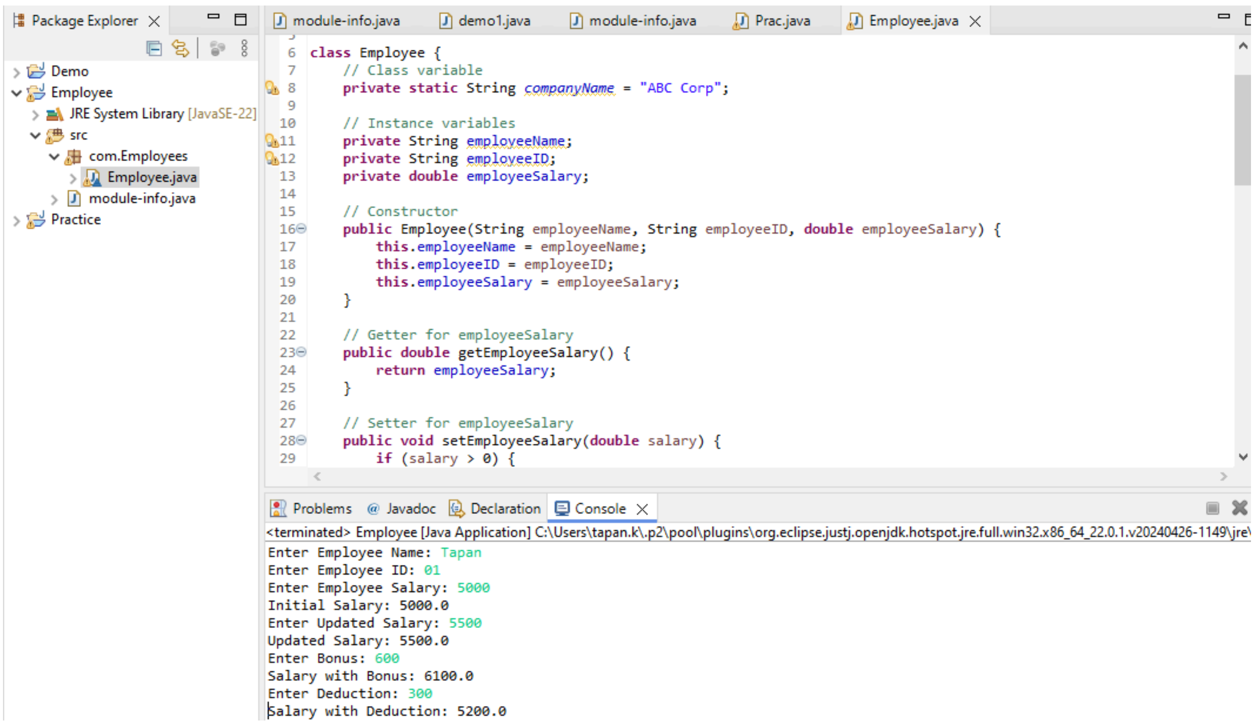
Answer:-

Code

```
package com.Employees;
import java.util.Scanner;

class Employee {
   // Class variable
   private static String companyName = "ABC Corp";

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

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

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

```
// Setter for employeeSalary
public void setEmployeeSalary(double salary) {
    if (salary > 0) {
        this.employeeSalary = salary;
    } else {
        System.out.println("Salary must be a positive number.");
    }
}

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

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

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);


    System.out.print("Enter Employee Name: ");
    String employeeName = scanner.nextLine();

    System.out.print("Enter Employee ID: ");
    String employeeID = scanner.nextLine();

    System.out.print("Enter Employee Salary: ");
    double employeeSalary = scanner.nextDouble();


    Employee emp1 = new Employee(employeeName, employeeID, employeeSalary);


    System.out.println("Initial Salary: " + emp1.getEmployeeSalary());


    System.out.print("Enter Updated Salary: ");
```

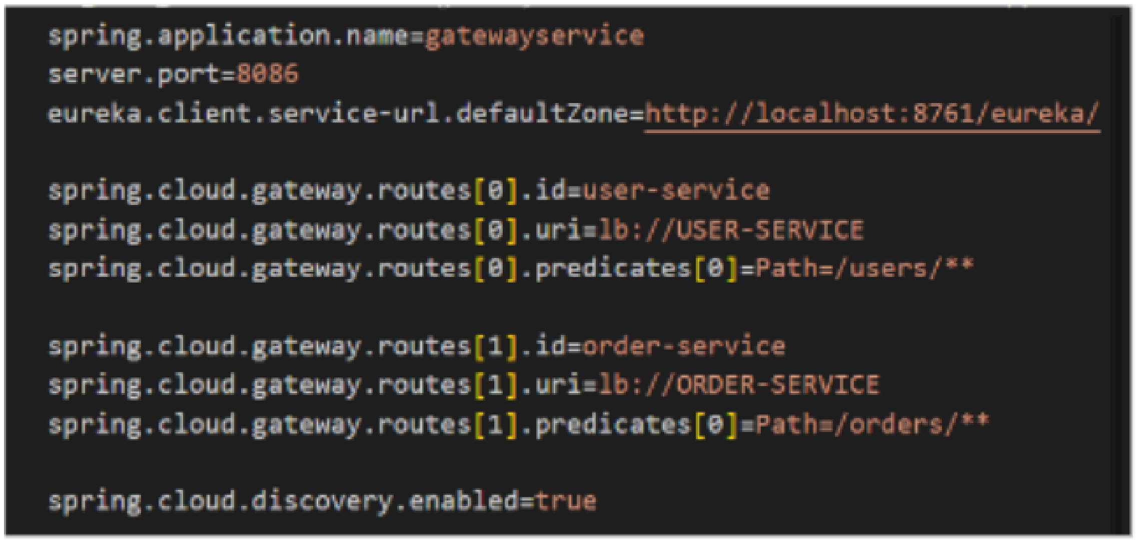
```
        double updatedSalary = scanner.nextDouble();
        emp1.setEmployeeSalary(updatedSalary);
        System.out.println("Updated Salary: " + emp1.getEmployeeSalary());

        // Calculating salary with bonus
        System.out.print("Enter Bonus: ");
        double bonus = scanner.nextDouble();
        double salaryWithBonus = emp1.calculateSalary(bonus);
        System.out.println("Salary with Bonus: " + salaryWithBonus);

        // Calculating salary with deduction
        System.out.print("Enter Deduction: ");
        int deduction = scanner.nextInt();
        double salaryWithDeduction = emp1.calculateSalary(deduction);
        System.out.println("Salary with Deduction: " + salaryWithDeduction);

        scanner.close();
    }
}
```

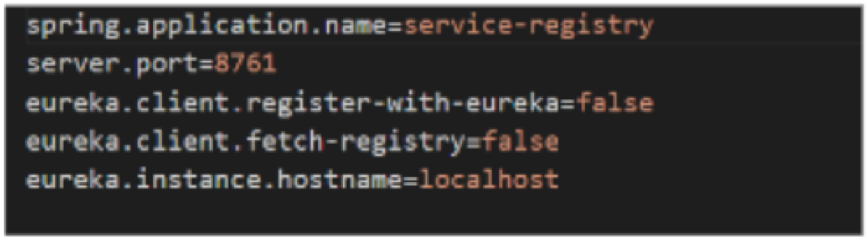


```
class Employee {
    // Class variable
    private static String companyName = "ABC Corp";

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

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

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

    // Setter for employeeSalary
    public void setEmployeeSalary(double salary) {
        if (salary > 0) {
```

```
<terminated> Employee [Java Application] C:\Users\tapan.k\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\
Enter Employee Name: Tapan
Enter Employee ID: 01
Enter Employee Salary: 5000
Initial Salary: 5000.0
Enter Updated Salary: 5500
Updated Salary: 5500.0
Enter Bonus: 600
Salary with Bonus: 6100.0
Enter Deduction: 300
Salary with Deduction: 5200.0
```

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


```
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
```

Answer:-

the microservices that use the Gateway and Service Discovery methods are:

1. **User Service (USER-SERVICE)**:
   - This service is identified by user-service.
   - The gateway routes requests with the path /users/** to the USER-SERVICE using the load balancer (lb://USER-SERVICE).
2. **Order Service (ORDER-SERVICE)**:
   - This service is identified by order-service.
   - The gateway routes requests with the path /orders/** to the ORDER-SERVICE using the load balancer (lb://ORDER-SERVICE).

Explanation:

- **Gateway Service** (gatewayservice):

        - The gateway service acts as an API Gateway and is configured to route requests to different microservices based on the URL path. In this case:
            - Requests that match the path /users/** are routed to the USER-SERVICE.
            - Requests that match the path /orders/** are routed to the ORDER-SERVICE.
        - It is registered with Eureka Service Discovery using the URL http://localhost:8761/eureka/.
- **Service Registry** (service-registry):
    - This is the Eureka server that runs on port 8761.
    - Microservices like USER-SERVICE and ORDER-SERVICE would register themselves with this Eureka server, making it possible for the Gateway service to discover them and route traffic appropriately.

In summary, the microservices involved are:

- **User Service** (identified as USER-SERVICE)
- **Order Service** (identified as ORDER-SERVICE)

These services use the Gateway service to route traffic based on the request paths and rely on the Eureka service registry for service discovery.