11 – DAY – TASK (08-08-2024)

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 packages;

import java.util.Scanner;
/**
* @author sanjeevkumar.v
*
*/
public class Employee {
// Class variable
private static String companyName = "Payoda Technologies";
// 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;
}
```

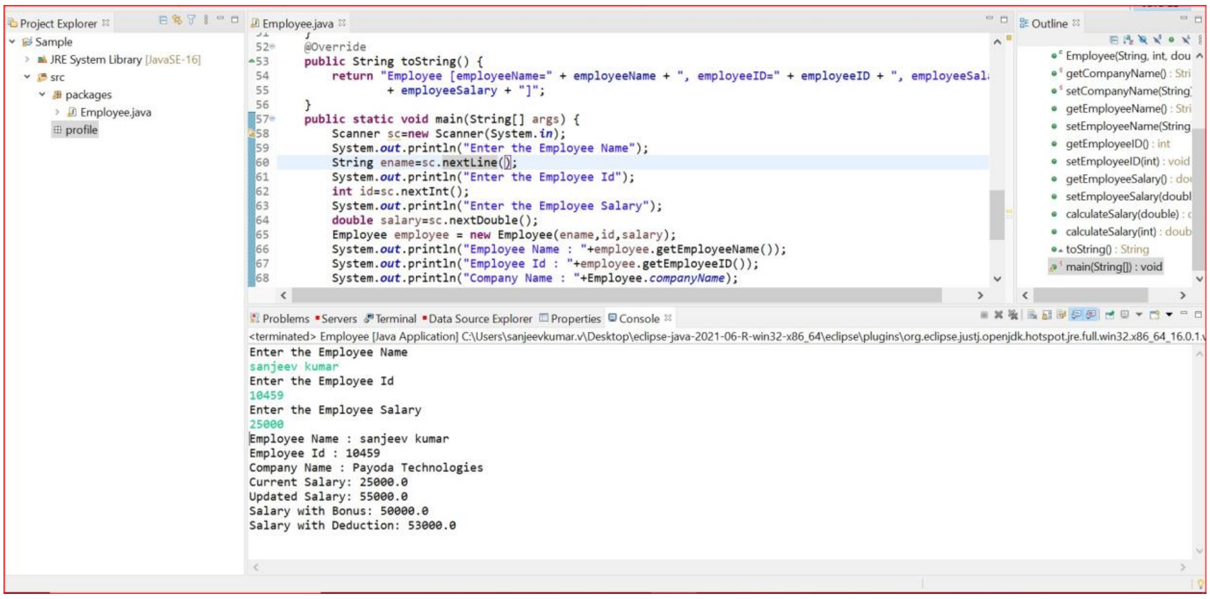
```
public String getEmployeeName() {
return employeeName;
}
public void setEmployeeName(String employeeName) {
this.employeeName = employeeName;
}

public int getEmployeeID() {
return employeeID;
}
public void setEmployeeID(int employeeID) {
this.employeeID = employeeID;
}
public double getEmployeeSalary() {
return employeeSalary;
}
public void setEmployeeSalary(double employeeSalary) {
this.employeeSalary = employeeSalary;
}
public double calculateSalary(double bonus) {
return this.employeeSalary + bonus;
}
public double calculateSalary(int deduction) {
return this.employeeSalary - deduction;
}
@Override
public String toString() {
return "Employee [employeeName=" + employeeName + ", employeeID=" +
employeeID + ", employeeSalary="
+ employeeSalary + "]";
}
public static void main(String[] args) {
Scanner sc=new Scanner(System.in);
System.out.println("Enter the Employee Name");
String ename=sc.nextLine();
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);
System.out.println("Current Salary: " + employee.getEmployeeSalary());
employee.setEmployeeSalary(55000);

System.out.println("Updated Salary: " + employee.getEmployeeSalary());
double bonusSalary = employee.calculateSalary(5000);
System.out.println("Salary with Bonus: " + bonusSalary);
double deductionSalary = employee.calculateSalary(2000);
System.out.println("Salary with Deduction: " + deductionSalary);
}

}
```

Output:



```java
    @Override
    public String toString() {
        return "Employee [employeeName=" + employeeName + ", employeeID=" + employeeID + ", employeeSal
                + employeeSalary + "]";
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Employee Name");
        String ename=sc.nextLine();
        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);
```

```
<terminated> Employee [Java Application] C:\Users\sanjeevkumar.v\Desktop\eclipse-java-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1
Enter the Employee Name
sanjeev kumar
Enter the Employee Id
10459
Enter the Employee Salary
25000
Employee Name : sanjeev kumar
Employee Id : 10459
Company Name : Payoda Technologies
Current Salary: 25000.0
Updated Salary: 55000.0
Salary with Bonus: 50000.0
Salary with Deduction: 53000.0
```

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

1. Spring Cloud Gateway (API Gateway)

- **Role:** Acts as a central entry point for all incoming client requests. It routes these requests to the appropriate microservices based on predefined routing rules.
- **Configuration:**
  - **Service Name:** `gatewayservice`
  - **Port:** 8886

- **Routes Configuration:**
    - Routes requests matching `/users/**` to the `USER-SERVICE` using load balancing (`lb://USER-SERVICE`).
    - Routes requests matching `/orders/**` to the `ORDER-SERVICE` using load balancing (`lb://ORDER-SERVICE`).

2. Eureka Service Registry (Service Discovery)

- **Role:** Provides a centralized registry where microservices can register themselves and discover other services. It enables dynamic service discovery and load balancing.
- **Configuration:**
    - **Service Name:** `service-registry`
    - **Port:** `8761`
    - **Does Not Register Itself:** `eureka.client.register-with-eureka=false`
    - **Does Not Fetch Other Registries:** `eureka.client.fetch-registry=false`

3. Microservices

- **User Service:**
    - **Role:** Handles operations related to users, such as user management and retrieval.
    - **Service Name:** `USER-SERVICE`
    - **Path:** `/users/**`
- **Order Service:**
    - **Role:** Manages operations related to orders, including order creation, retrieval, and processing.
    - **Service Name:** `ORDER-SERVICE`
    - **Path:** `/orders/**`

Data Flow and Interactions

Client Request: A client sends a request to the API Gateway (Spring Cloud Gateway).
Routing by Gateway: The API Gateway examines the request path and routes it to the appropriate microservice based on the route configuration.

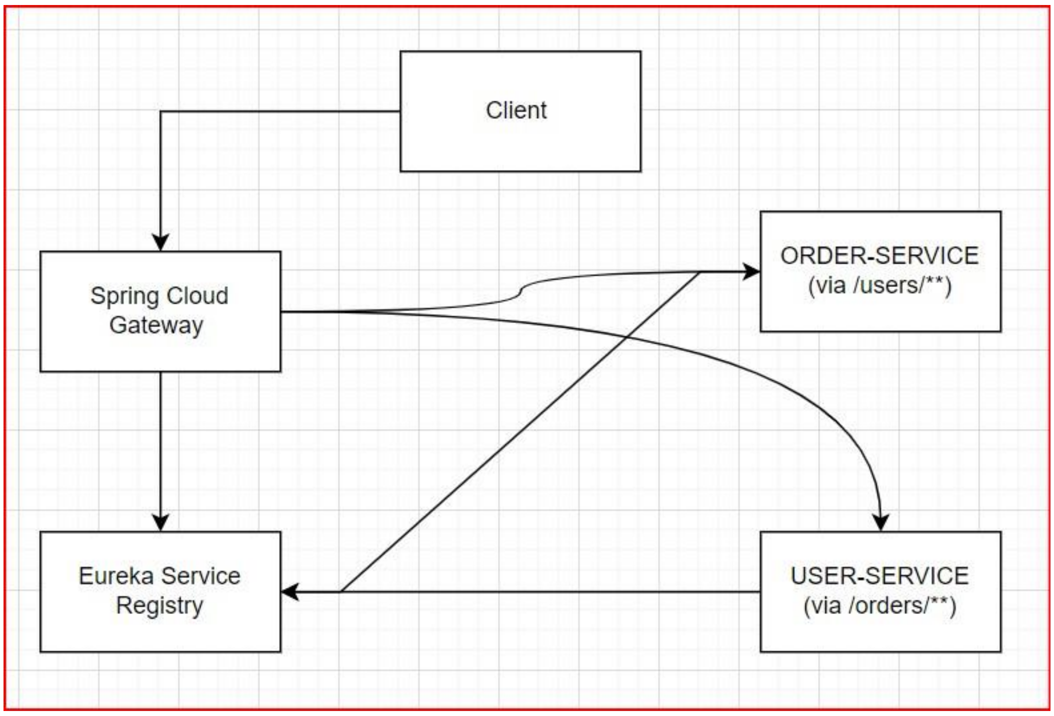
- Requests with paths starting with `/users/` are forwarded to the `USER-SERVICE`.

- Requests with paths starting with `/orders/` are forwarded to the `ORDER-SERVICE`.

Service Discovery: The Gateway uses Eureka to discover the instances of `USER-SERVICE` and `ORDER-SERVICE` dynamically, ensuring load balancing and failover capabilities.

Microservice Response: The microservice processes the request and returns a response to the API Gateway.

Response to Client: The API Gateway sends the response back to the client.


Client
Spring Cloud Gateway
ORDER-SERVICE (via /users/**)
Eureka Service Registry
USER-SERVICE (via /orders/**)