Case Study: Leave Tracking System

Object-Oriented Programming Basics

Estimated time: 15 minutes

Let's explore some real life applications of object-oriented programming basics related to this case study.

Classes and Objects

In real-world software development, classes are blueprints that define the properties and behaviors of objects. For example, banking apps create Account objects, and ride-sharing apps create Ride objects.

For this Leave Tracking System, you'll begin by creating the following two fundamental classes:

- Employee: Represents a company employee
- LeaveRequest: Represents a request for time off

Let's examine a sample employee class.

Sample Employee class

```
public class Employee {
    // Properties (attributes)
    private int employeeId;
    private String name;
    private String department;
    private String email;

    // Constructor
    public Employee(int employeeId, String name, String department, String email) {
        this.employeeId = employeeId;
        this.name = name;
        this.department = department;
        this.email = email;
    }
    // Methods will be added here
}
```

Next let's examine the practical related tasks you will need to complete:

- Create an Employee class with appropriate attributes
- Create a LeaveRequest class with start date, end date, and status attributes
- Create instances (objects) of these classes to represent different employees and their leave requests

Encapsulation

Encapsulation is another real-life task you'll implement. Encapsulation is similar to a medicine capsule. Encapsulation hides the internal details, just as a medicine capsule hides or holds medication, and encapsulation exposes only what's necessary for the function to happen, such as determining if the medication is the correct medication.

Real-life encapsulation uses include automatic teller machines (ATMs) encapsulating complex banking operations behind simple buttons, and smart home systems encapsulating complex device controls behind simple interfaces.

In this case study, you will encapsulate employee and leave requests data using private fields and public getters/setters such as you'll see in the following code example:

```
// In the Employee class
private int leaveBalance = 20; // Annual leave balance in days
// Getter method
public int getLeaveBalance() {
    return leaveBalance;
}
// Setter method with validation
public void setLeaveBalance(int leaveBalance) {
    if (leaveBalance >= 0) {
        this.leaveBalance = leaveBalance;
    } else {
        System.out.println("Leave balance cannot be negative.");
    }
}
```

So as part of encapsulation, you'll perform the following practical tasks:

- Add getters and setters for all attributes in Employee and LeaveRequest classes
- Add validation in setters (to enable rejection of negative leave balances)
- Create a method to calculate remaining leave balance after a request

Constructors

In real-life applications, constructors are like startup procedures. For example, when a new car is manufactured, the car needs its vehicle identification number (VIN), engine, and initial settings configured before it can operate.

In this case study, you will create constructors to properly initialize your object. Check out the following code example:

```
public class LeaveRequest {
    private int requestId;
    private Employee employee;
    private String startDate;
    private String endDate;
    private String status; // "Pending", "Approved", "Denied"
    private String reason;

    // Constructor
    public LeaveRequest(int requestId, Employee employee, String startDate,
                        String endDate, String reason) {
        this.requestId = requestId;
        this.employee = employee;
        this.startDate = startDate;
        this.endDate = endDate;
        this.status = "Pending"; // Default status
        this.reason = reason;
    }

    // Methods will be added here
}
```

For this application, you would perform the following practical tasks related to constructors:

- Create a default constructor for Employee that sets default values
- Create an overloaded constructor for LeaveRequest that accepts different parameters
- Use constructors to create new Employee and LeaveRequest objects

Real-life application of basic object-oriented programming in Java

In professional software development environments, these practices help create structured, maintainable code. Companies build their HR systems using objects to represent real-world entities, use encapsulation to protect data integrity, and implement constructors to ensure proper initialization.

What you can do in real life

Here are some of the basic object-oriented programming skills you can apply in real life:

- Design classes based on real-world objects in any domain
- Use proper encapsulation to hide implementation details
- Create appropriate constructors to initialize objects correctly

Author(s)

Ramanujam Srinivasan