Case study: Leave Tracking System

Advanced Object-Oriented Programming Concepts in Java

Estimated time: 10 minutes

As part of advanced object-oriented programming concepts in Java, you'll use inheritance, polymorphism, interfaces, and inner classes. Let's get started.

Inheritance

In real-life applications, inheritance creates parent-child relationships between classes. For example, in real life, vehicles share common attributes, but each kind of vehicle, such as cars, trucks, and motorcycles have their specializations. Similarly, all bank accounts have account numbers and balances, but savings and checking accounts have specific features.

In your case study, you'll apply inheritance to create the following different types of leave requests:

```
// Parent class
public class LeaveRequest {
    protected int requestId;
    protected Employee employee;
    protected String startDate;
    protected String endDate;
    protected String status;

    // Methods and constructors
    // ...
}
// Child class
public class SickLeaveRequest extends LeaveRequest {
    private boolean medicalCertificateProvided;

    public SickLeaveRequest(int requestId, Employee employee,
                        String startDate, String endDate,
                        boolean medicalCertificateProvided) {
        super(requestId, employee, startDate, endDate, "Sick Leave");
        this.medicalCertificateProvided = medicalCertificateProvided;
    }

    // Additional methods specific to sick leave
    public boolean isMedicalCertificateProvided() {
        return medicalCertificateProvided;
    }
}
```

You will complete the following practical tasks:

- Create a hierarchy of leave types (SickLeaveRequest, VacationLeaveRequest, MaternityLeaveRequest)
- Add specific attributes and methods to each leave type
- Use the parent class constructor in child classes using super()

Polymorphism

The real world purpose of polymorphism is to allow objects to take different forms. For example, TV remote buttons perform different actions depending on the device they're controlling.

Another example is how payment processing systems handle different payment methods such as credit cards, PayPal, and bank transfers, through a common interface.|

In your case study you will implement polymorphism to process different types of leave requests. Let's review the following code:

```
// In the parent LeaveRequest class
public boolean processRequest() {
    // Basic request processing logic
    System.out.println("Processing generic leave request...");
    return true;
}
// In the SickLeaveRequest class
@Override
public boolean processRequest() {
    if (!medicalCertificateProvided && getDuration() > 2) {
        System.out.println("Sick leave longer than 2 days requires a medical certificate");
        return false;
    }
    System.out.println("Processing sick leave request...");
    return true;
}
```

As part of implementing polymorphism, you'll complete the following practical tasks:

- Override the processRequest() method in each leave type subclass
- Create an array or list of different leave request types
- Process all requests using polymorphism

Interfaces and Abstract Classes

In real life, interfaces define capabilities without specifying implementation details.

For example, electrical outlets provide a standard interface for devices including mobile phone chargers, lamps, stoves washing machines and other devices.

Interestingly abstract classes provide partial implementations; similar to a semi-furnished apartment with some features already installed, like a pre-installed wall-mounted desk.

In this case study, you will create an interface for leave approval and an abstract class for leave types. Check out the following code for inspiration:

```
// Interface
public interface Approvable {
    boolean approve(String approverName);
    boolean deny(String approverName, String reason);
}
// Abstract class
public abstract class LeaveRequest implements Approvable {
    // Common fields and methods
    // ...

    // Abstract method that subclasses must implement
    public abstract int calculateLeaveDays();
}
```

The following practical tasks are part of implementing interfaces and abstract classes:

- Create an Approvable interface with approval methods
- Make LeaveRequest an abstract class that implements Approvable
- Implement the abstract methods in concrete leave type classes

Inner Classes

In real-life applications, inner classes are classes defined within other classes. Inner classes are like departments within a company, they exist within the context of the larger organization and often have special access to its resources.

In your case study, you'll use an inner class to represent leave request status history. Check out the following code:

```
public class LeaveRequest {
    // Other fields and methods
    // ...

    private ArrayList<StatusChange> statusHistory = new ArrayList<>();

    // Inner class to track status changes
    public class StatusChange {
        private String oldStatus;
        private String newStatus;
        private String changeDate;
        private String changedBy;

        public StatusChange(String oldStatus, String newStatus,
                            String changeDate, String changedBy) {
            this.oldStatus = oldStatus;
            this.newStatus = newStatus;
            this.changeDate = changeDate;
            this.changedBy = changedBy;
        }

        // Getters for the fields
        // ...
    }

    // Method to change status and record the change
    public void changeStatus(String newStatus, String changedBy) {
        String oldStatus = this.status;
        this.status = newStatus;

        // Create a new status change record
        StatusChange change = new StatusChange(
            oldStatus, newStatus, getCurrentDate(), changedBy);
        statusHistory.add(change);
    }
}
```

As part of using inner classes, you'll need to complete the following tasks:

- Create an inner class for status changes in the LeaveRequest class
- Use the inner class to track the history of status changes
- Display the full status history of a leave request

For real-life professional software development,these advanced object-oriented programming concepts help create flexible, maintainable systems. For example, HR software uses inheritance for different employee types, polymorphism to handle different leave policies, interfaces to standardize approval processes, and inner classes to manage related data.

What you can do in real life

In real life you can use advanced object-oriented programming to:

- Design class hierarchies based on "is-a" relationships
- Use polymorphism to simplify code that handles different object types
- Create interfaces to define required capabilities without specifying implementation
- Use abstract classes when you need both common implementation and specialized behavior

Author(s)

Ramanujam Srinivasan