Case study: Leave Tracking System

Working with Collections in Java

In real life, collections are like organized storage containers. Libraries use collections to organize books, hospitals use collections to manage patient records, and inventory systems use collections to track products.

In this case study, you will use various types of collections to store and manage leave requests, employee data, and work with other data in your Leave Tracking System. First, though, let's check out some sample code used to import collection data.

```
// Import necessary collections
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
```

When using collections, you will need to perform the following practical tasks:

- Identify which data in the system to store in collections
- Decide which collection types are appropriate for different data sets
- Import the necessary collection classes

Working with Lists: ArrayList and LinkedList

In the real world, lists are ordered collections similar to to-do lists or shopping lists. Online retailers use lists to show order history to customers, and music apps use lists to create playlists.

In this case study, you'll use an ArrayList to store leave requests for each employee:

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

    private ArrayList<LeaveRequest> leaveHistory = new ArrayList<>();

    public void addLeaveRequest(LeaveRequest request) {
        leaveHistory.add(request);
    }

    public ArrayList<LeaveRequest> getLeaveHistory() {
        return leaveHistory;
    }

    public LeaveRequest getLeaveRequestById(int requestId) {
        for (LeaveRequest request : leaveHistory) {
            if (request.getRequestId() == requestId) {
                return request;
            }
        }
        return null; // Request not found
    }
}
```

You'll want to complete the following practical tasks related to lists:

- Create an ArrayList to store leave requests for each employee
- Implement methods to add, retrieve, and search for leave requests
- Display all leave requests for a specific employee

Exploring Sets: HashSet and TreeSet

The real-world goal of using sets is to store unique elements with no duplicates. Universities use sets to ensure each student is enrolled only once in a course, and email systems use sets to track unique email addresses.

In this case study, you will use HashSet to track unique departments with pending leave requests. Check out the following code:

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

    private HashSet<String> departmentsWithPendingRequests = new HashSet<>();

    public void updateDepartmentsWithPendingRequests() {
```

```
        departmentsWithPendingRequests.clear();

        for (LeaveRequest request : allLeaveRequests) {
            if (request.getStatus().equals("Pending")) {
                departmentsWithPendingRequests.add(
                    request.getEmployee().getDepartment());
            }
        }
    }

    public boolean hasPendingRequests(String department) {
        return departmentsWithPendingRequests.contains(department);
    }
}
```

You'll want to complete the following practical tasks:

- Create a HashSet to track unique departments with pending requests
- Add and remove departments from the set based on request status
- Use the set to quickly check if a department has pending requests

Using Maps: HashMap and TreeMap

Maps have important real-life uses. Maps store key-value pairs like dictionaries. In your phone, the Contacts app uses maps to associate names with phone numbers, and online stores use maps to track product prices by SKU.

In this case study, you'll use a HashMap to quickly access employees by their ID. Let's check out the following code sample:

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

    private HashMap<Integer, Employee> employeeDirectory = new HashMap<>();

    public void addEmployee(Employee employee) {
        employeeDirectory.put(employee.getEmployeeId(), employee);
    }

    public Employee getEmployeeById(int employeeId) {
        return employeeDirectory.get(employeeId);
    }

    public boolean removeEmployee(int employeeId) {
        if (employeeDirectory.containsKey(employeeId)) {
            employeeDirectory.remove(employeeId);
            return true;
        }
        return false;
    }
}
```

You'll need to complete the following practical tasks:

- Create a HashMap to store employees by their ID
- Implement methods to add, retrieve, and remove employees from the map
- Use the map to quickly look up employee information

Using Queues in Java

Queues process elements in a first-in, first-out order. In the real-world, Print servers use queues to process print jobs, and customer service systems queue support tickets.

In this case study, you will use a queue to process leave requests in the order they were received. Next, review the following code:

```
import java.util.LinkedList;
import java.util.Queue;
public class LeaveApprovalSystem {
    private Queue<LeaveRequest> pendingRequests = new LinkedList<>();

    public void addPendingRequest(LeaveRequest request) {
        pendingRequests.add(request);
    }

    public LeaveRequest getNextPendingRequest() {
        return pendingRequests.poll(); // Retrieves and removes the head
    }
```

```
    public int getPendingRequestCount() {
        return pendingRequests.size();
    }

    public boolean hasPendingRequests() {
        return !pendingRequests.isEmpty();
    }
}
```

You'll want to complete the following practical tasks:

- Create a queue for pending leave requests
- Implement methods to add requests to the queue
- Process requests in the order they were received

Let's review real-life applications of using multiple application types. First, in practice, applications often need to combine multiple collection types. For example, banking systems track transactions using lists, organize unique account holders using sets, and organize account details using maps.

In this case study, you will combine collections to create a complete leave tracking system. Review the following code sample:

```
public class LeaveTrackingSystem {
    // Maps for quick lookup
    private HashMap<Integer, Employee> employeeById = new HashMap<>();
    private HashMap<Integer, LeaveRequest> requestById = new HashMap<>();

    // Lists for ordered access
    private ArrayList<LeaveRequest> allRequests = new ArrayList<>();

    // Sets for unique collections
    private HashSet<String> leaveTypes = new HashSet<>();

    // Queues for processing
    private Queue<LeaveRequest> pendingApprovals = new LinkedList<>();

    // Methods to manage these collections
    // ...
}
```

You will need to complete the following practical tasks:

- Determine which collection types work best for different aspects of the system
- Combine multiple collections to create a complete system
- Implement methods to maintain consistency across collections

In real-world professional software development, collections are essential for efficient data management. Human resource systems use collections to organize employee records, track leave balances, and manage approval workflows.

You'll want to complete the following practical, real-life tasks:

- Choose appropriate collection types based on data access patterns
- Use lists when order matters and duplicates are allowed
- Use sets when you need to track unique items
- Use maps when you need fast lookups by a key
- Use queues when processing items in a specific order

Author(s)

[Ramanujam Srinivasan]

Skills Network