Case Study: Leave Tracking System

File and Directory Management

Estimated time: 10 minutes

Let's explore some real-life applications of file and directory management related to this case study.

Introduction to Java File Handling

In real-life applications, file handling is crucial for data persistence. For example, banking applications save transaction records, document editors save your in-progress work, and email clients store your messages drafts in files.

In this case study, you will use file handling to save and load leave requests and employee data. Check out the following code:

```
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class LeaveTrackingSystem {
    private File employeeDataFile = new File("employees.csv");
    private File leaveRequestsFile = new File("leave_requests.csv");

    // Methods for saving and loading data
    // ...
}
```

As part of your work, you'll need to complete the following practical tasks:

- Create File objects for employee and leave request data
- Implement basic file existence checking
- Create directory structures for storing application data

Understanding Byte Streams in Java

In real-life applications, byte streams handle raw binary data. For example, image editors use byte streams to read and write picture files, and video games use byte streams to load game assets.

In this case study, you will use byte streams to save employee profile images. Consider the following example:

```
public void saveEmployeeImage(Employee employee, File imageFile) {
    try (FileInputStream in = new FileInputStream(imageFile);
         FileOutputStream out = new FileOutputStream(
             "images/" + employee.getEmployeeId() + ".jpg")) {

        byte[] buffer = new byte[1024];
        int length;

        // Copy the file content byte by byte
        while ((length = in.read(buffer)) > 0) {
            out.write(buffer, 0, length);
        }

        System.out.println("Employee image saved successfully");
    } catch (IOException e) {
        System.out.println("Error saving employee image: " + e.getMessage());
    }
}
```

As part of your work, you'll need to complete the following practical tasks:

- Create a method to save employee profile images using byte streams
- Implement error handling for file operations
- Create a method to load employee profile images

Working with File Input and Output Streams

In real life, character streams handle text data. Word processors use character streams to save documents, and configuration managers use character streams to store settings.

In this case study, you will use character streams to save and load employee data similar to what you see in the following code:

```
public void saveEmployeeData() {
    try (FileWriter writer = new FileWriter(employeeDataFile)) {
        // Write header
        writer.write("ID,Name,Department,Email,LeaveBalance\n");

        // Write each employee
        for (Employee employee : employeeDirectory.values()) {
            writer.write(
                employee.getEmployeeId() + "," +
                employee.getName() + "," +
                employee.getDepartment() + "," +
                employee.getEmail() + "," +
                employee.getLeaveBalance() + "\n"
            );
        }

        System.out.println("Employee data saved successfully");
    } catch (IOException e) {
        System.out.println("Error saving employee data: " + e.getMessage());
    }
}
public void loadEmployeeData() {
    if (!employeeDataFile.exists()) {
        System.out.println("No employee data file found");
        return;
    }

    try (BufferedReader reader = new BufferedReader(new FileReader(employeeDataFile))) {
        // Skip header
        String line = reader.readLine();

        // Read each employee
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length == 5) {
                int id = Integer.parseInt(parts[0]);
                String name = parts[1];
                String department = parts[2];
                String email = parts[3];
                int leaveBalance = Integer.parseInt(parts[4]);

                Employee employee = new Employee(id, name, department, email);
                employee.setLeaveBalance(leaveBalance);

                employeeDirectory.put(id, employee);
            }
        }

        System.out.println("Employee data loaded successfully");
    } catch (IOException | NumberFormatException e) {
        System.out.println("Error loading employee data: " + e.getMessage());
    }
}
```

As part of your leave application creation process, you will need to complete the following practical tasks:

- Create methods to save and load employee data to/from CSV files
- Implement proper file handling with try-with-resources statements
- Add error handling for file operations

Managing Directories in Java

As part of its real-life applications, directory management organizes files into hierarchical structures. For example, photo organizers create folders by date, and project management tools organize files by project.

In your study, you will create a directory structure for your Leave Tracking System. Let's examine some code:

```
public void initializeFileStructure() {
    // Create main data directory
    File dataDir = new File("leavetracker_data");
    if (!dataDir.exists()) {
        if (dataDir.mkdir()) {
            System.out.println("Created data directory");
        } else {
            System.out.println("Failed to create data directory");
            return;
        }
    }

    // Create subdirectories
    File[] subdirs = {
        new File(dataDir, "employees"),
        new File(dataDir, "requests"),
        new File(dataDir, "reports"),
        new File(dataDir, "backups")
    };

    for (File dir : subdirs) {
```

```
            if (!dir.exists()) {
                if (dir.mkdir()) {
                    System.out.println("Created directory: " + dir.getName());
                } else {
                    System.out.println("Failed to create directory: " + dir.getName());
                }
            }
        }
    }
```

As part of your work, you will also need to complete the following practical tasks related to directory management:

- Create a method to initialize the directory structure for the application
- Implement backup functionality by copying files to a backup directory
- Create a method to list all leave request files for a specific employee

Implementing File Management for Leave Requests

In real life, complete file management, also referred to as file handling, involves creating, reading, updating, and deleting files. Inventory systems save and update product information, and Customer Relationship Management (CRM) systems save customer data.

In this case study, you will implement CRUD (Create, Read, Update, Delete) operations for leave requests. Next, review the following code:

```
    public void saveLeaveRequest(LeaveRequest request) {
        File requestFile = new File(
            "leavetracker_data/requests/" +
            request.getRequestId() + ".txt");

        try (PrintWriter writer = new PrintWriter(new FileWriter(requestFile))) {
            writer.println("RequestID: " + request.getRequestId());
            writer.println("EmployeeID: " + request.getEmployee().getEmployeeId());
            writer.println("StartDate: " + request.getStartDate());
            writer.println("EndDate: " + request.getEndDate());
            writer.println("Status: " + request.getStatus());
            writer.println("Reason: " + request.getReason());

            if (request instanceof SickLeaveRequest) {
                SickLeaveRequest slr = (SickLeaveRequest) request;
                writer.println("Type: Sick");
                writer.println("MedicalCertificate: " +
                               slr.isMedicalCertificateProvided());
            } else {
                writer.println("Type: Regular");
            }

            System.out.println("Leave request saved: " + request.getRequestId());
        } catch (IOException e) {
            System.out.println("Error saving leave request: " + e.getMessage());
        }
    }
```

As part of implementing file management, you will need to complete the following practical tasks:

- Create methods to save, load, update, and delete leave request files
- Implement functions to search for leave requests by date or status
- Create a backup system that creates copies of important files

As part of the real-life application of directory and file management, effective file management is essential for data persistence. Human resource (HR) systems use file handling to store employee records, leave balances, and historical data for reporting and compliance.

What you can do in real life

In real life, you can use directory and file management coding techniques to:

- Design file formats appropriate for your data (text, binary, CSV, JSON, etc.)
- Create organized directory structures to manage application data
- Implement robust error handling for file operations
- Use file operations to save application state between sessions

Author(s)

Ramanujam Srinivasan