Bài tập Java và database:

Hệ Thống Quản Lý Tài Khoản Ngân Hàng

Hệ thống giúp quản lý tài khoản ngân hàng với các nghiệp vụ:

- Thêm, sửa, xóa, tìm kiếm tài khoản.
- Quản lý nhân viên ngân hàng.
- Thực hiện các giao dịch như nạp tiền, rút tiền, và chuyển khoản.
- Lưu trữ và quản lý các giao dịch.

Chi Tiết Các Class

1. Lớp Account (Tài Khoản)

Mục Đích: Quản lý thông tin tài khoản ngân hàng của khách hàng.

Thuộc Tính:

1. accountNumber (String): Số tài khoản duy nhất.
    - **Ý nghĩa:** định danh tài khoản.
2. accountHolder (String): Tên chủ tài khoản.
    - **Ý nghĩa:** lưu thông tin khách hàng sở hữu tài khoản.
3. balance (double): Số dư tài khoản.
    - **Ý nghĩa:** quản lý số tiền hiện có.
4. employeeInCharge (Employee): Nhân viên phụ trách tài khoản.
    - **Ý nghĩa:** xác định ai quản lý tài khoản.

Phương Thức:

1. deposit(double amount): Nạp tiền vào tài khoản.
    - **Logic**: Cộng số tiền vào balance.
2. withdraw(double amount): Rút tiền từ tài khoản.
    - **Logic**: Trừ số tiền từ balance. Nếu không đủ, ném ngoại lệ.
3. addTransaction(Transaction transaction): Gắn giao dịch với tài khoản.

---

2. Lớp Employee (Nhân Viên)

Mục Đích: Quản lý thông tin nhân viên của ngân hàng.

Thuộc Tính:

1. id (String): ID nhân viên duy nhất.
    - **Ý nghĩa:** định danh nhân viên.
2. name (String): Tên nhân viên.
    - **Ý nghĩa:** lưu thông tin cá nhân.
3. salary (double): Lương cơ bản.
    - **Ý nghĩa:** tính toán thưởng và chi trả.
4. managedAccounts (List<Account>): Danh sách tài khoản mà nhân viên quản lý.
    - **Ý nghĩa:** lưu các tài khoản thuộc trách nhiệm của nhân viên.

Phương Thức:

1. calculateBonus(): Tính thưởng dựa trên vai trò nhân viên.
    - **Logic**: Trừu tượng, mỗi lớp con sẽ tự định nghĩa.
2. addAccount(Account account): Gắn tài khoản vào danh sách quản lý.
3. processTransaction(Transaction transaction): Xử lý giao dịch liên quan đến tài khoản.

---

3. Lớp Transaction (Giao Dịch)

Mục Đích: Quản lý thông tin giao dịch liên quan đến tài khoản.

Thuộc Tính:

1. transactionId (String): ID giao dịch duy nhất.
    - **Ý nghĩa:** định danh giao dịch.
2. account (Account): Tài khoản liên quan đến giao dịch.
    - **Ý nghĩa:** xác định tài khoản thực hiện giao dịch.
3. employee (Employee): Nhân viên thực hiện giao dịch.
    - **Ý nghĩa:** xác định ai phụ trách giao dịch.
4. type (String): Loại giao dịch (Deposit, Withdraw, Transfer).
    - **Ý nghĩa:** mô tả bản chất giao dịch.
5. amount (double): Số tiền giao dịch.
    - **Ý nghĩa:** quản lý số tiền liên quan.
6. timestamp (LocalDateTime): Thời gian thực hiện giao dịch.
    - **Ý nghĩa:** lưu trữ thông tin thời gian.

Phương Thức:

1. Transaction(String id, Account account, Employee employee, String type, double amount): Constructor tạo giao dịch.
2. toString(): Hiển thị chi tiết giao dịch.

Chi Tiết Mối Quan Hệ

1. **Account - Employee**:

    - **Quan hệ**: Một nhân viên có thể quản lý nhiều tài khoản. Một tài khoản chỉ thuộc về một nhân viên.
    - **Ý nghĩa:** : Quản lý trách nhiệm của nhân viên đối với khách hàng.
2. **Account - Transaction**:
    - **Quan hệ**: Một tài khoản có thể có nhiều giao dịch. Một giao dịch chỉ thuộc về một tài khoản.
    - **Ý nghĩa:** : Ghi nhận lịch sử giao dịch.
3. **Employee - Transaction**:
    - **Quan hệ**: Một nhân viên có thể thực hiện nhiều giao dịch. Một giao dịch chỉ được thực hiện bởi một nhân viên.
    - **Ý nghĩa:** : Theo dõi hiệu suất làm việc của nhân viên.

Mô Tả Chi Tiết Yêu Cầu Nghiệp Vụ

Dưới đây là mô tả chi tiết từng yêu cầu nghiệp vụ kèm theo các bước thực hiện, điều kiện kiểm tra và logic xử lý:

---

1. Mô Tả Chi Tiết Yêu Cầu Nghiệp Vụ

1.1. Thêm Tài Khoản

Mô Tả:

- Hệ thống cần cho phép thêm một tài khoản mới vào danh sách quản lý.
- Tài khoản được liên kết với một nhân viên phụ trách.

Quy Trình:

1. **Nhập thông tin tài khoản mới**:
    - Số tài khoản (accountNumber) – phải là duy nhất.
    - Tên chủ tài khoản (accountHolder).
    - Số dư ban đầu (balance).
2. **Kiểm tra số tài khoản**:
    - Nếu accountNumber đã tồn tại, từ chối thêm và thông báo lỗi.
3. **Liên kết nhân viên phụ trách**:
    - Tìm nhân viên phù hợp dựa trên ID hoặc chọn ngẫu nhiên một giao dịch viên.
4. **Lưu tài khoản**:
    - Thêm tài khoản vào danh sách tài khoản trong AccountManager.

Logic Business:

- Một nhân viên có thể quản lý nhiều tài khoản.
- Số tài khoản không được trùng lặp.

1.2. Sửa Thông Tin Tài Khoản

Mô Tả:

- Cập nhật thông tin chủ tài khoản hoặc số dư tài khoản.

Quy Trình:

1. **Tìm kiếm tài khoản**:
    - Nhập accountNumber để xác định tài khoản cần sửa.
    - Nếu không tìm thấy, thông báo lỗi.
2. **Cập nhật thông tin**:
    - Cho phép sửa accountHolder (tên chủ tài khoản).
    - **Không cho phép** sửa số tài khoản.
3. **Lưu thay đổi**:
    - Cập nhật thông tin mới trong danh sách.

Logic Business:

- Không được phép sửa số tài khoản (accountNumber).
- Tên chủ tài khoản có thể được thay đổi theo yêu cầu.

1.3. Xóa Tài Khoản

Mô Tả:

- Xóa một tài khoản khỏi danh sách quản lý.

Quy Trình:

1. **Tìm kiếm tài khoản**:
    - Nhập accountNumber để xác định tài khoản cần xóa.
    - Nếu không tìm thấy, thông báo lỗi.
2. **Kiểm tra điều kiện xóa**:
    - Nếu tài khoản còn số dư lớn hơn 0, yêu cầu khách hàng rút hết trước khi xóa.
    - Nếu tài khoản có giao dịch liên quan, không được phép xóa (hoặc yêu cầu hủy giao dịch trước).
3. **Xóa tài khoản**:
    - Loại bỏ tài khoản khỏi danh sách.

Logic Business:

- Tài khoản có số dư > 0 không được phép xóa.
- Giao dịch liên quan đến tài khoản phải được xử lý trước khi xóa.

1.4. Tìm Kiếm Tài Khoản

Mô Tả:

- Tìm kiếm thông tin tài khoản theo số tài khoản hoặc tên chủ tài khoản.

Quy Trình:

1. **Nhập thông tin tìm kiếm**:
    - Nhập accountNumber hoặc accountHolder để tìm kiếm.
2. **Tìm kiếm**:
    - Duyệt qua danh sách tài khoản trong AccountManager.
    - So khớp thông tin tài khoản với điều kiện tìm kiếm.
3. **Hiển thị kết quả**:
    - Nếu tìm thấy, trả về thông tin tài khoản.
    - Nếu không tìm thấy, thông báo lỗi.

Logic Business:

- Tìm kiếm bằng số tài khoản có độ ưu tiên cao hơn.
- Tên chủ tài khoản có thể trùng lặp.

---

2. Giao dịch tài khoản

2.1. Nạp Tiền

Mô Tả:

- Nạp thêm tiền vào tài khoản của khách hàng.

Quy Trình:

1. **Tìm kiếm tài khoản**:
    - Nhập accountNumber để xác định tài khoản cần nạp tiền.
    - Nếu không tìm thấy, thông báo lỗi.
2. **Nhập số tiền cần nạp**:
    - Kiểm tra số tiền có hợp lệ không (phải > 0).
    - Nếu không hợp lệ, thông báo lỗi.
3. **Thực hiện giao dịch**:
    - Cộng số tiền vào balance.
    - Tạo giao dịch Transaction với loại Deposit.

Logic Business:

- Số tiền nạp phải > 0.
- Giao dịch được lưu vào lịch sử giao dịch.

2.2. Rút Tiền

Mô Tả:

- Rút tiền từ tài khoản khách hàng.

Quy Trình:

1. **Tìm kiếm tài khoản**:
    - Nhập accountNumber để xác định tài khoản cần rút tiền.
    - Nếu không tìm thấy, thông báo lỗi.
2. **Nhập số tiền cần rút**:
    - Kiểm tra số tiền có hợp lệ không (phải > 0).
    - Nếu số tiền lớn hơn số dư, từ chối giao dịch.
3. **Thực hiện giao dịch**:
    - Trừ số tiền từ balance.
    - Tạo giao dịch Transaction với loại Withdraw.

Logic Business:

- Số tiền rút phải > 0 và không vượt quá số dư hiện tại.
- Tài khoản không được phép có số dư âm.

2.3. Chuyển Khoản

Mô Tả:

- Chuyển tiền từ một tài khoản này sang một tài khoản khác.

Quy Trình:

1. **Tìm kiếm tài khoản**:
    - Nhập accountNumber của tài khoản gửi và nhận.
    - Nếu một trong hai tài khoản không tồn tại, thông báo lỗi.
2. **Nhập số tiền cần chuyển**:
    - Kiểm tra số tiền có hợp lệ không (phải > 0).
    - Nếu số tiền lớn hơn số dư của tài khoản gửi, từ chối giao dịch.
3. **Thực hiện giao dịch**:
    - Trừ số tiền từ balance của tài khoản gửi.
    - Cộng số tiền vào balance của tài khoản nhận.
    - Tạo hai giao dịch Transaction: một cho tài khoản gửi, một cho tài khoản nhận.

Logic Business:

- Số tiền chuyển phải > 0 và không vượt quá số dư tài khoản gửi.

- Cả tài khoản gửi và nhận phải tồn tại trong hệ thống.

---

3. Quản lý giao dịch

3.1. Hiển Thị Lịch Sử Giao Dịch Của Một Tài Khoản

Mô Tả:

- Xem danh sách các giao dịch đã được thực hiện trên một tài khoản cụ thể.

Quy Trình:

1. **Nhập thông tin tài khoản**:
    - Yêu cầu người dùng nhập accountNumber để xác định tài khoản cần xem.
2. **Tìm kiếm tài khoản**:
    - Duyệt danh sách tài khoản trong hệ thống (AccountManager).
    - Nếu không tìm thấy tài khoản, thông báo lỗi và kết thúc quy trình.
3. **Lấy danh sách giao dịch**:
    - Truy xuất danh sách giao dịch từ tài khoản.
    - Lọc danh sách Transaction để lấy các giao dịch có liên quan đến tài khoản này.
4. **Hiển thị kết quả**:
    - Hiển thị thông tin từng giao dịch:
        - ID giao dịch (transactionId).
        - Loại giao dịch (Deposit, Withdraw, Transfer).
        - Số tiền (amount).
        - Thời gian thực hiện (timestamp).
        - Nhân viên thực hiện (employee).

Logic Business:

- Tài khoản phải tồn tại trong hệ thống.
- Lịch sử giao dịch phải được hiển thị theo thứ tự thời gian (mới nhất lên trên).
- Nếu không có giao dịch nào liên quan, thông báo "Không có giao dịch nào cho tài khoản này."

Ví Dụ:

- **Nhập**: Số tài khoản: A12345
- **Kết quả**:

Lịch sử giao dịch cho tài khoản A12345:
1. ID: T001, Loại: Deposit, Số tiền: 5000, Thời gian: 2025-01-01 10:30, Nhân viên: Alice
2. ID: T002, Loại: Withdraw, Số tiền: 2000, Thời gian: 2025-01-02 14:15, Nhân viên: Bob

---

3.2. Hiển Thị Toàn Bộ Giao Dịch Trong Hệ Thống

Mô Tả:

- Hiển thị tất cả các giao dịch đã thực hiện trong hệ thống.

Quy Trình:

1. **Lấy danh sách giao dịch**:
    - Truy xuất danh sách Transaction từ hệ thống (TransactionManager hoặc danh sách giao dịch toàn cục).
    - Nếu danh sách trống, thông báo "Không có giao dịch nào trong hệ thống."
2. **Hiển thị giao dịch**:
    - Hiển thị thông tin chi tiết từng giao dịch:
        - ID giao dịch (transactionId).
        - Tài khoản liên quan (account).
        - Loại giao dịch (Deposit, Withdraw, Transfer).
        - Số tiền (amount).
        - Thời gian thực hiện (timestamp).
        - Nhân viên thực hiện (employee).

Logic Business:

- Danh sách giao dịch phải được hiển thị theo thứ tự thời gian (mới nhất lên trên).
- Nếu danh sách quá dài, có thể phân trang hoặc giới hạn số lượng hiển thị.

Ví Dụ:

- **Kết quả**:

Danh sách toàn bộ giao dịch:
1. ID: T001, Loại: Deposit, Tài khoản: A12345, Số tiền: 5000, Thời gian: 2025-01-01 10:30, Nhân viên: Alice
2. ID: T002, Loại: Withdraw, Tài khoản: A67890, Số tiền: 2000, Thời gian: 2025-01-02 14:15, Nhân viên: Bob
3. ID: T003, Loại: Transfer, Tài khoản gửi: A12345, Tài khoản nhận: A67890, Số tiền: 1000, Thời gian: 2025-01-03 09:00, Nhân viên: Alice

---

Ghi Chú

- **Phân biệt hiển thị lịch sử giao dịch cá nhân và toàn bộ giao dịch**:
    - Hiển thị lịch sử giao dịch cá nhân tập trung vào một tài khoản cụ thể.
    - Hiển thị toàn bộ giao dịch cung cấp cái nhìn tổng quan cho hệ thống.

4. Quản Lý Nhân Viên

Dưới đây là chi tiết 3 nghiệp vụ trong quản lý nhân viên, bao gồm: thêm nhân viên mới, xem danh sách nhân viên, và gán tài khoản cho nhân viên.

4.1. Thêm Nhân Viên Mới

Mô Tả:

- Hệ thống cho phép thêm một nhân viên mới vào danh sách quản lý nhân viên.

Quy Trình:

1. **Nhập thông tin nhân viên mới**:
    - ID nhân viên (id) – phải là duy nhất.
    - Tên nhân viên (name).
    - Lương cơ bản (salary).
2. **Kiểm tra ID nhân viên**:
    - Nếu ID đã tồn tại trong hệ thống, từ chối thêm và thông báo lỗi.
3. **Lưu nhân viên**:
    - Thêm nhân viên mới vào danh sách nhân viên trong EmployeeManager.

Logic Business:

- **ID nhân viên phải là duy nhất**: Đảm bảo không có xung đột dữ liệu khi tìm kiếm hoặc quản lý.
- Nhân viên mới có thể chưa được gán tài khoản nào.

Ví Dụ:

- Nhập ID: E001, Tên: Alice, Lương: 10,000.
- Thêm nhân viên vào danh sách.

4.2. Xem Danh Sách Nhân Viên

Mô Tả:

- Hiển thị thông tin toàn bộ nhân viên trong hệ thống.

Quy Trình:

1. **Truy xuất danh sách nhân viên**:
    - Lấy danh sách tất cả nhân viên từ hệ thống (EmployeeManager).
2. **Hiển thị thông tin**:
    - Hiển thị từng nhân viên với các thông tin:
        - ID nhân viên (id).
        - Tên nhân viên (name).
        - Lương cơ bản (salary).
        - Số lượng tài khoản đang quản lý.

Logic Business:

- Danh sách phải được sắp xếp (theo ID hoặc tên) để dễ quản lý.
- Nếu không có nhân viên nào, thông báo "Không có nhân viên trong hệ thống."

Ví Dụ:

Kết quả hiển thị:

ID: E001, Tên: Alice, Lương: 10,000, Tài khoản quản lý: 5
ID: E002, Tên: Bob, Lương: 12,000, Tài khoản quản lý: 2

4.3. Gán Tài Khoản Cho Nhân Viên

Mô Tả:

- Gắn một tài khoản ngân hàng cho nhân viên để quản lý.

Quy Trình:

1. **Nhập thông tin**:
    - Nhập ID nhân viên (id) để tìm kiếm nhân viên.
    - Nhập số tài khoản (accountNumber) để tìm kiếm tài khoản.
2. **Kiểm tra tính hợp lệ**:
    - Nếu không tìm thấy nhân viên hoặc tài khoản, thông báo lỗi.
    - Nếu tài khoản đã được gán cho một nhân viên khác, từ chối gán và thông báo lỗi.
3. **Liên kết tài khoản với nhân viên**:
    - Thêm tài khoản vào danh sách managedAccounts của nhân viên.
    - Cập nhật thuộc tính employeeInCharge của tài khoản với nhân viên được gán.

Logic Business:

- **Một tài khoản chỉ có thể được gán cho một nhân viên duy nhất**.
- Nhân viên có thể quản lý nhiều tài khoản.

Ví Dụ:

- **Nhập**:
  ID nhân viên: E001
  Số tài khoản: A12345
- **Hành động**:
  Gắn tài khoản A12345 cho nhân viên E001.
- **Kết quả**:
  Tài khoản A12345 được thêm vào danh sách managedAccounts của E001.

Ghi Chú

- Mỗi nghiệp vụ đều có liên quan mật thiết đến các class chính: Account, Employee, và Transaction.

- Quy trình phải đảm bảo tính toàn vẹn dữ liệu, không xảy ra lỗi nghiêm trọng khi xử lý các nghiệp vụ.

Menu Gợi Ý Trong Lớp Main

Danh Sách Chức Năng

1. **Quản lý tài khoản**:
    - 1.1. Thêm tài khoản mới.
    - 1.2. Sửa thông tin tài khoản.
    - 1.3. Xóa tài khoản.
    - 1.4. Tìm kiếm tài khoản.
2. **Giao dịch tài khoản**:
    - 2.1. Nạp tiền vào tài khoản.
    - 2.2. Rút tiền từ tài khoản.
    - 2.3. Chuyển khoản giữa hai tài khoản.
3. **Quản lý giao dịch**:
    - 3.1. Hiển thị lịch sử giao dịch của một tài khoản.
    - 3.2. Hiển thị toàn bộ giao dịch trong hệ thống.
4. **Quản lý nhân viên**:
    - 4.1. Thêm nhân viên mới.
    - 4.2. Xem danh sách nhân viên.
    - 4.3. Gán tài khoản cho nhân viên.
5. **Thoát chương trình**:
    - 5.1. Thoát khỏi hệ thống.

Kịch Bản Chạy Chương Trình - Test Case

Dưới đây là kịch bản minh họa chương trình quản lý tài khoản ngân hàng hoạt động với **input**(là thông tin người sử dụng hệ thống nhập vào) và **output**(là các thông tin hệ thống hiển thị ra khi nhận được input ứng với các chức năng)

Sau khi hoàn thiện chương trình đảm bảo chương trình hoạt động tương tự testcase này

---

Test Case 1: Thêm Tài Khoản Mới

Input:

```
1.1  (Quản lý tài khoản -> Thêm tài khoản mới)
Số tài khoản: A12345
Tên chủ tài khoản: John Doe
Số dư ban đầu: 10000
Nhân viên phụ trách (ID): E001
```

Output:

```
Tài khoản được thêm thành công!
Số tài khoản: A12345
Tên chủ tài khoản: John Doe
Số dư ban đầu: 10,000
Nhân viên phụ trách: Alice
```

Test Case 2: Sửa Thông Tin Tài Khoản

Input:

```
1.2  (Quản lý tài khoản -> Sửa thông tin tài khoản)
Số tài khoản: A12345
Tên chủ tài khoản mới: Jane Smith
```

Output:

```
Tài khoản được cập nhật thành công!
Số tài khoản: A12345
Tên chủ tài khoản mới: Jane Smith
```

Test Case 3: Xóa Tài Khoản

Input:

```
1.3  (Quản lý tài khoản -> Xóa tài khoản)
Số tài khoản: A12345
```

Output:

```
Tài khoản A12345 đã được xóa thành công!
```

Trường Hợp Lỗi (Còn Số Dư):

```
1.3  (Quản lý tài khoản -> Xóa tài khoản)
Số tài khoản: A12345
Lỗi: Không thể xóa tài khoản A12345 vì tài khoản còn số dư 5,000.
```

Test Case 4: Tìm Kiếm Tài Khoản

Input:

```
1.4  (Quản lý tài khoản -> Tìm kiếm tài khoản)
Số tài khoản: A12345
```

Output:

```
Tìm thấy tài khoản:
Số tài khoản: A12345
Tên chủ tài khoản: John Doe
Số dư hiện tại: 10,000
Nhân viên phụ trách: Alice
```

Test Case 5: Nạp Tiền

Input:

```
2.1  (Giao dịch tài khoản -> Nạp tiền)
Số tài khoản: A12345
Số tiền nạp: 5000
```

Output:

```
Nạp tiền thành công!
Số tài khoản: A12345
Số tiền nạp: 5,000
Số dư mới: 15,000
```

Test Case 6: Rút Tiền

Input:

```
2.2  (Giao dịch tài khoản -> Rút tiền)
Số tài khoản: A12345
Số tiền rút: 2000
```

Output:

```
Rút tiền thành công!
Số tài khoản: A12345
Số tiền rút: 2,000
Số dư mới: 13,000
```

Trường Hợp Lỗi (Số dư không đủ):

```
2.2  (Giao dịch tài khoản -> Rút tiền)
Số tài khoản: A12345
Số tiền rút: 20,000
Lỗi: Số dư không đủ để thực hiện giao dịch. Số dư hiện tại: 10,000.
```

Test Case 7: Chuyển Khoản

Input:

```
2.3  (Giao dịch tài khoản -> Chuyển khoản)
Tài khoản gửi: A12345
Tài khoản nhận: A67890
Số tiền chuyển: 1000
```

Output:

```
Chuyển khoản thành công!
Tài khoản gửi: A12345
Tài khoản nhận: A67890
Số tiền chuyển: 1,000
Số dư mới của tài khoản gửi: 12,000
Số dư mới của tài khoản nhận: 1,000
```

Test Case 8: Hiển Thị Lịch Sử Giao Dịch Của Một Tài Khoản

Input:

```
3.1  (Quản lý giao dịch -> Hiển thị lịch sử giao dịch của một tài
khoản)
Số tài khoản: A12345
```

Output:

```
Lịch sử giao dịch cho tài khoản A12345:
1. ID: T001, Loại: Deposit, Số tiền: 5,000, Thời gian: 2025-01-01
10:30, Nhân viên: Alice
2. ID: T002, Loại: Withdraw, Số tiền: 2,000, Thời gian: 2025-01-02
14:15, Nhân viên: Bob
```

Test Case 9: Hiển Thị Toàn Bộ Giao Dịch Trong Hệ Thống

Input:

```
3.2  (Quản lý giao dịch -> Hiển thị toàn bộ giao dịch trong hệ
thống)
```

Output:

```
Danh sách toàn bộ giao dịch:
1. ID: T001, Loại: Deposit, Tài khoản: A12345, Số tiền: 5,000, Thời
gian: 2025-01-01 10:30, Nhân viên: Alice
2. ID: T002, Loại: Withdraw, Tài khoản: A67890, Số tiền: 2,000, Thời
gian: 2025-01-02 14:15, Nhân viên: Bob
3. ID: T003, Loại: Transfer, Tài khoản gửi: A12345, Tài khoản nhận:
A67890, Số tiền: 1,000, Thời gian: 2025-01-03 09:00, Nhân viên:
Alice
```

Test Case 10: Thêm Nhân Viên Mới

Input:

```
4.1  (Quản lý nhân viên -> Thêm nhân viên mới)
ID nhân viên: E003
Tên nhân viên: Charlie
Lương cơ bản: 15,000
```

Output:

```
Nhân viên được thêm thành công!
ID: E003
Tên: Charlie
Lương cơ bản: 15,000
```

Test Case 11: Gán Tài Khoản Cho Nhân Viên

Input:

```
4.3  (Quản lý nhân viên -> Gán tài khoản cho nhân viên)
ID nhân viên: E001
Số tài khoản: A12345
```

Output:

```
Tài khoản A12345 đã được gán thành công cho nhân viên E001.
```

Test Case 12: Thoát Chương Trình

Input:

```
5.1  (Thoát chương trình)
```

Output:

```
Hệ thống đã thoát. Cảm ơn bạn đã sử dụng!
```

Dưới đây là một ví dụ về cách triển khai (mang tính chất tham khảo) cho bài tập "Hệ Thống Quản Lý Tài Khoản Ngân Hàng" bằng Java. Phần code được viết dưới dạng hướng đối tượng, sử dụng các class Account, Employee, Transaction, và một vài class "manager" (hoặc service) để quản lý dữ liệu.

Lưu ý:

1. Trong ví dụ này, dữ liệu được lưu trữ "trong bộ nhớ" (in-memory) thông qua các List thay vì kết nối tới CSDL thật. Khi triển khai thực tế, bạn có thể thay thế các phần List này bằng cơ chế DAO (Data Access Object), JDBC hoặc ORM (Hibernate, JPA, …) để thao tác với database.
2. Ví dụ chưa xử lý triệt để tất cả ngoại lệ hay cài đặt đầy đủ phân trang, sorting,... Bạn có thể phát triển thêm tùy yêu cầu.
3. Phần menu (Main) chỉ là demo cơ bản để minh họa cách gọi các chức năng.

---

1. Lớp Account

```
package com.bankmanagement.models;

import com.bankmanagement.models.Employee;

import com.bankmanagement.models.Transaction;


import java.util.ArrayList;

import java.util.List;


public class Account {

    private String accountNumber;    // Số tài khoản

    private String accountHolder;    // Tên chủ tài khoản

    private double balance;          // Số dư tài khoản

    private Employee employeeInCharge;    // Nhân viên phụ trách

    private List<Transaction> transactionList; // Danh sách giao dịch liên quan


    // Constructor

    public Account(String accountNumber, String accountHolder, double balance, Employee
employeeInCharge) {

        this.accountNumber = accountNumber;

        this.accountHolder = accountHolder;

        this.balance = balance;
```

```
    this.employeeInCharge = employeeInCharge;
    this.transactionList = new ArrayList<>();
}

// Getter & Setter
public String getAccountNumber() {
    return accountNumber;
}

public String getAccountHolder() {
    return accountHolder;
}

public void setAccountHolder(String accountHolder) {
    this.accountHolder = accountHolder;
}

public double getBalance() {
    return balance;
}

public Employee getEmployeeInCharge() {
    return employeeInCharge;
}

public void setEmployeeInCharge(Employee employeeInCharge) {
    this.employeeInCharge = employeeInCharge;
}

public List<Transaction> getTransactionList() {
```

```
    return transactionList;
}

// Phương thức nạp tiền
public void deposit(double amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Số tiền nạp phải lớn hơn 0.");
    }
    this.balance += amount;
}

// Phương thức rút tiền
public void withdraw(double amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Số tiền rút phải lớn hơn 0.");
    }
    if (this.balance < amount) {
        throw new IllegalArgumentException("Số dư không đủ để thực hiện giao dịch rút tiền.");
    }
    this.balance -= amount;
}

// Thêm một giao dịch vào danh sách của account
public void addTransaction(Transaction transaction) {
    transactionList.add(transaction);
}

@Override
public String toString() {
    return "Account{ " +
```

```
            "accountNumber='" + accountNumber + '\'' +
            ", accountHolder='" + accountHolder + '\'' +
            ", balance=" + balance +
            ", employeeInCharge=" + (employeeInCharge != null ? employeeInCharge.getName() : "Chưa có") +
            " }";
    }
}
```

---

2. Lớp Employee

```
package com.bankmanagement.models;

import java.util.ArrayList;
import java.util.List;

public abstract class Employee {
    private String id;              // ID nhân viên
    private String name;            // Tên nhân viên
    private double salary;          // Lương cơ bản
    private List<Account> managedAccounts; // Danh sách tài khoản mà nhân viên quản lý

    public Employee(String id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
        this.managedAccounts = new ArrayList<>();
    }

    // Getter & Setter
    public String getId() {
        return id;
```

```
}

public String getName() {
    return name;
}

public double getSalary() {
    return salary;
}

public List<Account> getManagedAccounts() {
    return managedAccounts;
}

// Mỗi lớp con của Employee sẽ định nghĩa cách tính thưởng riêng
public abstract double calculateBonus();

// Thêm tài khoản vào danh sách quản lý
public void addAccount(Account account) {
    if (account != null) {
        managedAccounts.add(account);
    }
}

// Xử lý giao dịch (ví dụ minh họa, logic thực tế có thể phức tạp hơn)
public void processTransaction(Transaction transaction) {
    // Tùy theo type mà ta xử lý
    // Ví dụ: transaction.getType() -> "Deposit", "Withdraw", "Transfer"
    // Ở đây có thể implement logic để ghi log, kiểm tra, duyệt giao dịch...
    System.out.println("Nhân viên " + this.name + " đang xử lý giao dịch: " + transaction);
```

```
    }

    @Override
    public String toString() {
        return "Employee{" +
                "id='" + id + '\'' +
                ", name='" + name + '\'' +
                ", salary=" + salary +
                ", numberOfManagedAccounts=" + managedAccounts.size() +
                '}';
    }
}
```

Ví dụ ta có thể tạo các lớp con Teller (Giao dịch viên), Manager (Quản lý) kế thừa Employee nếu muốn phân loại:

```
package com.bankmanagement.models;

public class Teller extends Employee {

    public Teller(String id, String name, double salary) {
        super(id, name, salary);
    }

    @Override
    public double calculateBonus() {
        // Giả sử giao dịch viên có thưởng = 10% lương cơ bản
        return getSalary() * 0.1;
    }
}

package com.bankmanagement.models;
```

```
public class Manager extends Employee {
    public Manager(String id, String name, double salary) {
        super(id, name, salary);
    }

    @Override
    public double calculateBonus() {
        // Giả sử quản lý có thưởng = 20% lương cơ bản
        return getSalary() * 0.2;
    }
}
```

---

3. Lớp Transaction

```
package com.bankmanagement.models;
import java.time.LocalDateTime;

public class Transaction {
    private String transactionId;    // ID giao dịch
    private Account account;         // Tài khoản liên quan
    private Employee employee;       // Nhân viên thực hiện
    private String type;             // Loại giao dịch: Deposit, Withdraw, Transfer
    private double amount;           // Số tiền
    private LocalDateTime timestamp; // Thời gian giao dịch

    public Transaction(String transactionId,
                       Account account,
                       Employee employee,
                       String type,
                       double amount) {
        this.transactionId = transactionId;
```

```
    this.account = account;
    this.employee = employee;
    this.type = type;
    this.amount = amount;
    this.timestamp = LocalDateTime.now(); // Lấy thời gian hiện tại, hoặc truyền từ ngoài
}

// Getter & Setter
public String getTransactionId() {
    return transactionId;
}

public Account getAccount() {
    return account;
}

public Employee getEmployee() {
    return employee;
}

public String getType() {
    return type;
}

public double getAmount() {
    return amount;
}

public LocalDateTime getTimestamp() {
    return timestamp;
```

```
    }

    @Override
    public String toString() {
        return "Transaction{" +
                "transactionId='" + transactionId + '\'' +
                ", type='" + type + '\'' +
                ", amount=" + amount +
                ", account=" + (account != null ? account.getAccountNumber() : "N/A") +
                ", employee=" + (employee != null ? employee.getName() : "N/A") +
                ", timestamp=" + timestamp +
                '}';
    }
}
```

---

4. Lớp AccountManager (hoặc AccountService)

Lớp này chịu trách nhiệm quản lý (thêm, sửa, xóa, tìm kiếm) tài khoản. Thông thường khi kết nối CSDL, ta sẽ có DAO hoặc Repository riêng. Ở đây minh họa trên List.

```
package com.bankmanagement.services;

import com.bankmanagement.models.Account;

import com.bankmanagement.models.Employee;


import java.util.ArrayList;

import java.util.List;


public class AccountManager {
    private List<Account> accounts;

    public AccountManager() {
        this.accounts = new ArrayList<>();
```

```
}

// Thêm tài khoản
public void addAccount(Account account) throws Exception {
    // Kiểm tra trùng số tài khoản
    Account existing = findByAccountNumber(account.getAccountNumber());
    if (existing != null) {
        throw new Exception("Số tài khoản đã tồn tại: " + account.getAccountNumber());
    }
    accounts.add(account);
}

// Sửa thông tin tài khoản (chủ yếu sửa tên chủ tài khoản, vì accountNumber không được phép sửa)
public void updateAccountHolder(String accountNumber, String newHolder) throws Exception {
    Account account = findByAccountNumber(accountNumber);
    if (account == null) {
        throw new Exception("Không tìm thấy tài khoản: " + accountNumber);
    }
    account.setAccountHolder(newHolder);
}

// Xóa tài khoản
public void deleteAccount(String accountNumber) throws Exception {
    Account account = findByAccountNumber(accountNumber);
    if (account == null) {
        throw new Exception("Không tìm thấy tài khoản: " + accountNumber);
    }
    // Kiểm tra số dư
    if (account.getBalance() > 0) {
        throw new Exception("Không thể xóa tài khoản " + accountNumber
```

```
                + " vì tài khoản còn số dư " + account.getBalance());
        }
        // Kiểm tra nếu có lịch sử giao dịch?
        // Tùy yêu cầu, ví dụ nếu có transaction, không cho xóa hoặc yêu cầu xử lý trước
        if (!account.getTransactionList().isEmpty()) {
            throw new Exception("Tài khoản có lịch sử giao dịch. Không thể xóa hoặc cần xác nhận hủy giao dịch trước.");
        }
        accounts.remove(account);
    }

    // Tìm kiếm tài khoản theo accountNumber
    public Account findByAccountNumber(String accountNumber) {
        for (Account a : accounts) {
            if (a.getAccountNumber().equals(accountNumber)) {
                return a;
            }
        }
        return null;
    }

    // Tìm kiếm tài khoản theo tên chủ tài khoản
    public List<Account> findByAccountHolder(String accountHolder) {
        List<Account> result = new ArrayList<>();
        for (Account a : accounts) {
            if (a.getAccountHolder().equalsIgnoreCase(accountHolder)) {
                result.add(a);
            }
        }
        return result;
```

```
    }

    public List<Account> getAllAccounts() {
        return accounts;
    }
}
```

---

5. Lớp EmployeeManager (hoặc EmployeeService)

Tương tự, lớp này quản lý danh sách nhân viên, thêm nhân viên, tìm kiếm nhân viên, gán tài khoản cho nhân viên,...

```
package com.bankmanagement.services;

import com.bankmanagement.models.Account;

import com.bankmanagement.models.Employee;


import java.util.ArrayList;

import java.util.List;


public class EmployeeManager {
    private List<Employee> employees;

    public EmployeeManager() {
        this.employees = new ArrayList<>();
    }

    // Thêm nhân viên
    public void addEmployee(Employee employee) throws Exception {
        // Kiểm tra trùng ID
        if (findById(employee.getId()) != null) {
            throw new Exception("ID nhân viên đã tồn tại: " + employee.getId());
        }
```

```
        employees.add(employee);
    }

    // Tìm nhân viên theo ID
    public Employee findById(String id) {
        for (Employee e : employees) {
            if (e.getId().equals(id)) {
                return e;
            }
        }
        return null;
    }

    // Lấy toàn bộ danh sách nhân viên
    public List<Employee> getAllEmployees() {
        return employees;
    }

    // Gán tài khoản cho nhân viên
    public void assignAccountToEmployee(Employee employee, Account account) throws Exception {
        // Kiểm tra nếu account đã có employeeInCharge
        if (account.getEmployeeInCharge() != null && account.getEmployeeInCharge() != employee) {
            throw new Exception("Tài khoản " + account.getAccountNumber() + " đã được gán cho nhân viên khác!");
        }
        // Gán
        employee.addAccount(account);
        account.setEmployeeInCharge(employee);
    }
}
```

6. Lớp TransactionManager (hoặc TransactionService)

Dùng để thao tác nạp tiền, rút tiền, chuyển khoản, lưu trữ và hiển thị danh sách giao dịch.

```
package com.bankmanagement.services;
import com.bankmanagement.models.Account;
import com.bankmanagement.models.Employee;
import com.bankmanagement.models.Transaction;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class TransactionManager {
    private List<Transaction> transactions;

    public TransactionManager() {
        this.transactions = new ArrayList<>();
    }

    // Nạp tiền
    public Transaction deposit(Account account, Employee employee, double amount) throws Exception {
        if (account == null) {
            throw new Exception("Tài khoản không tồn tại để nạp tiền.");
        }
        // Gọi hàm deposit ở account
        account.deposit(amount);
        // Tạo transaction
        Transaction transaction = new Transaction(
                generateTransactionId(),
                account,
```

```
            employee,
            "Deposit",
            amount
        );
        // Lưu transaction vào list
        transactions.add(transaction);
        // Gắn transaction vào account
        account.addTransaction(transaction);
        return transaction;
    }

    // Rút tiền
    public Transaction withdraw(Account account, Employee employee, double amount) throws Exception
{
        if (account == null) {
            throw new Exception("Tài khoản không tồn tại để rút tiền.");
        }
        account.withdraw(amount);
        Transaction transaction = new Transaction(
            generateTransactionId(),
            account,
            employee,
            "Withdraw",
            amount
        );
        transactions.add(transaction);
        account.addTransaction(transaction);
        return transaction;
    }
```

```
    // Chuyển khoản
    public List<Transaction> transfer(Account sender, Account receiver, Employee employee, double amount) throws Exception {
        if (sender == null || receiver == null) {
            throw new Exception("Một trong hai tài khoản không tồn tại.");
        }
        if (amount <= 0) {
            throw new Exception("Số tiền chuyển phải > 0");
        }
        if (sender.getBalance() < amount) {
            throw new Exception("Số dư không đủ để chuyển khoản.");
        }

        // Thực hiện
        sender.withdraw(amount);
        receiver.deposit(amount);

        // Tạo transaction cho tài khoản gửi
        Transaction senderTx = new Transaction(generateTransactionId(), sender, employee, "Transfer-Out", amount);
        transactions.add(senderTx);
        sender.addTransaction(senderTx);

        // Tạo transaction cho tài khoản nhận
        Transaction receiverTx = new Transaction(generateTransactionId(), receiver, employee, "Transfer-In", amount);
        transactions.add(receiverTx);
        receiver.addTransaction(receiverTx);

        List<Transaction> result = new ArrayList<>();
        result.add(senderTx);
```

```
    result.add(receiverTx);
    return result;
}

// Lấy toàn bộ giao dịch
public List<Transaction> getAllTransactions() {
    // Có thể sắp xếp theo timestamp mới nhất lên đầu
    // Ở đây minh họa sắp xếp giảm dần:
    List<Transaction> sorted = new ArrayList<>(transactions);
    sorted.sort((t1, t2) -> t2.getTimestamp().compareTo(t1.getTimestamp()));
    return sorted;
}

// Lấy lịch sử giao dịch của một tài khoản
public List<Transaction> getTransactionsByAccount(Account account) {
    // Ở đây ta có thể trả về danh sách transactionList bên trong account
    // Hoặc lọc từ transactions. Để chắc chắn, ta lọc từ transactions
    List<Transaction> result = new ArrayList<>();
    for (Transaction t : transactions) {
        if (t.getAccount().getAccountNumber().equals(account.getAccountNumber())) {
            result.add(t);
        }
    }
    // Sắp xếp giảm dần theo thời gian
    result.sort((t1, t2) -> t2.getTimestamp().compareTo(t1.getTimestamp()));
    return result;
}

// Giả lập gen transactionId
private String generateTransactionId() {
```

```
        return "T" + (transactions.size() + 1);
    }
}
```

---

7. Lớp Main (Menu điều khiển)

Đây là ví dụ minh họa cho menu console đơn giản. Bạn có thể cải tiến thêm.

```
package com.bankmanagement;
import com.bankmanagement.models.*;
import com.bankmanagement.services.*;

import java.util.List;
import java.util.Scanner;

public class Main {
    private static AccountManager accountManager = new AccountManager();
    private static EmployeeManager employeeManager = new EmployeeManager();
    private static TransactionManager transactionManager = new TransactionManager();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        // Tạo sẵn một vài Employee để test
        try {
            Employee e1 = new Teller("E001", "Alice", 10000);
            Employee e2 = new Teller("E002", "Bob", 12000);
            employeeManager.addEmployee(e1);
            employeeManager.addEmployee(e2);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
boolean running = true;
while (running) {
    printMenu();
    String choice = scanner.nextLine();
    switch (choice) {
        case "1.1":
            addNewAccount();
            break;
        case "1.2":
            updateAccount();
            break;
        case "1.3":
            deleteAccount();
            break;
        case "1.4":
            searchAccount();
            break;
        case "2.1":
            deposit();
            break;
        case "2.2":
            withdraw();
            break;
        case "2.3":
            transfer();
            break;
        case "3.1":
            showAccountTransactions();
            break;
        case "3.2":
```

```
                showAllTransactions();
                break;
            case "4.1":
                addNewEmployee();
                break;
            case "4.2":
                showAllEmployees();
                break;
            case "4.3":
                assignAccountToEmployee();
                break;
            case "5.1":
                System.out.println("Hệ thống đã thoát. Cảm ơn bạn đã sử dụng!");
                running = false;
                break;
            default:
                System.out.println("Lựa chọn không hợp lệ.");
        }
    }
}

// In menu
private static void printMenu() {
    System.out.println("\n===== MENU =====");
    System.out.println("1. Quản lý tài khoản:");
    System.out.println("   1.1. Thêm tài khoản mới");
    System.out.println("   1.2. Sửa thông tin tài khoản");
    System.out.println("   1.3. Xóa tài khoản");
    System.out.println("   1.4. Tìm kiếm tài khoản");
    System.out.println("2. Giao dịch tài khoản:");
```

```
    System.out.println("   2.1. Nạp tiền vào tài khoản");
    System.out.println("   2.2. Rút tiền từ tài khoản");
    System.out.println("   2.3. Chuyển khoản giữa hai tài khoản");
    System.out.println("3. Quản lý giao dịch:");
    System.out.println("   3.1. Hiển thị lịch sử giao dịch của một tài khoản");
    System.out.println("   3.2. Hiển thị toàn bộ giao dịch trong hệ thống");
    System.out.println("4. Quản lý nhân viên:");
    System.out.println("   4.1. Thêm nhân viên mới");
    System.out.println("   4.2. Xem danh sách nhân viên");
    System.out.println("   4.3. Gán tài khoản cho nhân viên");
    System.out.println("5. Thoát chương trình:");
    System.out.println("   5.1. Thoát khỏi hệ thống");
    System.out.print("Lựa chọn: ");
}

// 1.1: Thêm tài khoản
private static void addNewAccount() {
    try {
        System.out.print("Nhập số tài khoản: ");
        String accNumber = scanner.nextLine();
        System.out.print("Nhập tên chủ tài khoản: ");
        String holder = scanner.nextLine();
        System.out.print("Nhập số dư ban đầu: ");
        double balance = Double.parseDouble(scanner.nextLine());
        System.out.print("Nhập ID nhân viên phụ trách: ");
        String empId = scanner.nextLine();
        Employee e = employeeManager.findById(empId);
        if (e == null) {
            System.out.println("Không tìm thấy nhân viên với ID: " + empId);
            return;
```

```
        }
        // Tạo account
        Account acc = new Account(accNumber, holder, balance, e);
        // Thêm account vào AccountManager
        accountManager.addAccount(acc);
        // Gán account cho employee
        employeeManager.assignAccountToEmployee(e, acc);

        System.out.println("Tài khoản được thêm thành công!");
        System.out.println("Số tài khoản: " + acc.getAccountNumber());
        System.out.println("Tên chủ tài khoản: " + acc.getAccountHolder());
        System.out.println("Số dư ban đầu: " + acc.getBalance());
        System.out.println("Nhân viên phụ trách: " + e.getName());
    } catch (Exception ex) {
        System.out.println("Lỗi khi thêm tài khoản: " + ex.getMessage());
    }
}

// 1.2: Sửa thông tin tài khoản
private static void updateAccount() {
    try {
        System.out.print("Nhập số tài khoản cần sửa: ");
        String accNumber = scanner.nextLine();
        System.out.print("Nhập tên chủ tài khoản mới: ");
        String newHolder = scanner.nextLine();
        accountManager.updateAccountHolder(accNumber, newHolder);
        System.out.println("Tài khoản được cập nhật thành công!");
        System.out.println("Số tài khoản: " + accNumber);
        System.out.println("Tên chủ tài khoản mới: " + newHolder);
    } catch (Exception ex) {
```

```
            System.out.println("Lỗi: " + ex.getMessage());
        }
    }

    // 1.3: Xóa tài khoản
    private static void deleteAccount() {
        try {
            System.out.print("Nhập số tài khoản cần xóa: ");
            String accNumber = scanner.nextLine();
            accountManager.deleteAccount(accNumber);
            System.out.println("Tài khoản " + accNumber + " đã được xóa thành công!");
        } catch (Exception ex) {
            System.out.println("Lỗi: " + ex.getMessage());
        }
    }

    // 1.4: Tìm kiếm tài khoản
    private static void searchAccount() {
        System.out.print("Nhập số tài khoản (nhấn Enter bỏ qua) hoặc tên chủ tài khoản: ");
        String input = scanner.nextLine();
        // Thử tìm theo accountNumber trước
        Account a = accountManager.findByAccountNumber(input);
        if (a != null) {
            System.out.println("Tìm thấy tài khoản:");
            System.out.println("Số tài khoản: " + a.getAccountNumber());
            System.out.println("Tên chủ tài khoản: " + a.getAccountHolder());
            System.out.println("Số dư hiện tại: " + a.getBalance());
            System.out.println("Nhân viên phụ trách: " + (a.getEmployeeInCharge() != null ?
a.getEmployeeInCharge().getName() : "N/A"));
        } else {
```

```
        // Tìm theo tên
        List<Account> list = accountManager.findByAccountHolder(input);
        if (list.isEmpty()) {
            System.out.println("Không tìm thấy tài khoản phù hợp.");
        } else {
            System.out.println("Các tài khoản trùng tên chủ tài khoản:");
            for (Account acc : list) {
                System.out.println(acc);
            }
        }
    }
}

// 2.1: Nạp tiền
private static void deposit() {
    try {
        System.out.print("Nhập số tài khoản: ");
        String accNumber = scanner.nextLine();
        Account account = accountManager.findByAccountNumber(accNumber);
        if (account == null) {
            System.out.println("Không tìm thấy tài khoản: " + accNumber);
            return;
        }
        System.out.print("Nhập ID nhân viên thực hiện: ");
        String empId = scanner.nextLine();
        Employee e = employeeManager.findById(empId);
        if (e == null) {
            System.out.println("Không tìm thấy nhân viên: " + empId);
            return;
        }
```

```
        System.out.print("Nhập số tiền nạp: ");
        double amount = Double.parseDouble(scanner.nextLine());
        transactionManager.deposit(account, e, amount);

        System.out.println("Nạp tiền thành công!");
        System.out.println("Số tài khoản: " + account.getAccountNumber());
        System.out.println("Số tiền nạp: " + amount);
        System.out.println("Số dư mới: " + account.getBalance());
    } catch (Exception ex) {
        System.out.println("Lỗi: " + ex.getMessage());
    }
}

// 2.2: Rút tiền
private static void withdraw() {
    try {
        System.out.print("Nhập số tài khoản: ");
        String accNumber = scanner.nextLine();
        Account account = accountManager.findByAccountNumber(accNumber);
        if (account == null) {
            System.out.println("Không tìm thấy tài khoản: " + accNumber);
            return;
        }
        System.out.print("Nhập ID nhân viên thực hiện: ");
        String empId = scanner.nextLine();
        Employee e = employeeManager.findById(empId);
        if (e == null) {
            System.out.println("Không tìm thấy nhân viên: " + empId);
            return;
        }
```

```
        System.out.print("Nhập số tiền rút: ");
        double amount = Double.parseDouble(scanner.nextLine());
        transactionManager.withdraw(account, e, amount);

        System.out.println("Rút tiền thành công!");
        System.out.println("Số tài khoản: " + account.getAccountNumber());
        System.out.println("Số tiền rút: " + amount);
        System.out.println("Số dư mới: " + account.getBalance());
    } catch (Exception ex) {
        System.out.println("Lỗi: " + ex.getMessage());
    }
}

// 2.3: Chuyển khoản
private static void transfer() {
    try {
        System.out.print("Nhập số tài khoản gửi: ");
        String senderNum = scanner.nextLine();
        Account sender = accountManager.findByAccountNumber(senderNum);

        System.out.print("Nhập số tài khoản nhận: ");
        String receiverNum = scanner.nextLine();
        Account receiver = accountManager.findByAccountNumber(receiverNum);

        System.out.print("Nhập ID nhân viên thực hiện: ");
        String empId = scanner.nextLine();
        Employee e = employeeManager.findById(empId);

        if (sender == null || receiver == null || e == null) {
            System.out.println("Không tìm thấy tài khoản gửi/nhận hoặc nhân viên thực hiện!");
```

```
            return;
        }

        System.out.print("Nhập số tiền chuyển: ");
        double amount = Double.parseDouble(scanner.nextLine());

        transactionManager.transfer(sender, receiver, e, amount);

        System.out.println("Chuyển khoản thành công!");
        System.out.println("Tài khoản gửi: " + senderNum);
        System.out.println("Tài khoản nhận: " + receiverNum);
        System.out.println("Số tiền chuyển: " + amount);
        System.out.println("Số dư mới tài khoản gửi: " + sender.getBalance());
        System.out.println("Số dư mới tài khoản nhận: " + receiver.getBalance());
    } catch (Exception ex) {
        System.out.println("Lỗi: " + ex.getMessage());
    }
}

// 3.1: Hiển thị lịch sử giao dịch của một tài khoản
private static void showAccountTransactions() {
    System.out.print("Nhập số tài khoản: ");
    String accNumber = scanner.nextLine();
    Account account = accountManager.findByAccountNumber(accNumber);
    if (account == null) {
        System.out.println("Không tìm thấy tài khoản: " + accNumber);
        return;
    }
    // Lấy lịch sử
    List<Transaction> txList = transactionManager.getTransactionsByAccount(account);
```

```
    if (txList.isEmpty()) {
        System.out.println("Không có giao dịch nào cho tài khoản này.");
    } else {
        System.out.println("Lịch sử giao dịch cho tài khoản " + accNumber + ":");
        for (Transaction tx : txList) {
            System.out.println(tx);
        }
    }
}

// 3.2: Hiển thị toàn bộ giao dịch
private static void showAllTransactions() {
    List<Transaction> txList = transactionManager.getAllTransactions();
    if (txList.isEmpty()) {
        System.out.println("Không có giao dịch nào trong hệ thống.");
    } else {
        System.out.println("Danh sách toàn bộ giao dịch:");
        for (Transaction tx : txList) {
            System.out.println(tx);
        }
    }
}

// 4.1: Thêm nhân viên mới
private static void addNewEmployee() {
    try {
        System.out.print("Nhập ID nhân viên: ");
        String id = scanner.nextLine();
        System.out.print("Nhập tên nhân viên: ");
        String name = scanner.nextLine();
```

```
            System.out.print("Nhập lương cơ bản: ");
            double salary = Double.parseDouble(scanner.nextLine());

            // Ở đây tạm thêm nhân viên vào dạng Teller
            // Nếu bạn muốn nhập vai trò (Teller hay Manager) thì có thể hỏi thêm
            Employee e = new Teller(id, name, salary);
            employeeManager.addEmployee(e);

            System.out.println("Nhân viên được thêm thành công!");
            System.out.println("ID: " + e.getId());
            System.out.println("Tên: " + e.getName());
            System.out.println("Lương cơ bản: " + e.getSalary());
        } catch (Exception ex) {
            System.out.println("Lỗi: " + ex.getMessage());
        }
    }

    // 4.2: Xem danh sách nhân viên
    private static void showAllEmployees() {
        List<Employee> list = employeeManager.getAllEmployees();
        if (list.isEmpty()) {
            System.out.println("Không có nhân viên trong hệ thống.");
        } else {
            // Có thể sắp xếp list trước khi in
            System.out.println("Danh sách nhân viên:");
            for (Employee e : list) {
                System.out.println("ID: " + e.getId() +
                                   ", Tên: " + e.getName() +
                                   ", Lương: " + e.getSalary() +
                                   ", Tài khoản quản lý: " + e.getManagedAccounts().size());
```

```
            }
        }
    }

    // 4.3: Gán tài khoản cho nhân viên
    private static void assignAccountToEmployee() {
        try {
            System.out.print("Nhập ID nhân viên: ");
            String empId = scanner.nextLine();
            Employee e = employeeManager.findById(empId);
            if (e == null) {
                System.out.println("Không tìm thấy nhân viên: " + empId);
                return;
            }
            System.out.print("Nhập số tài khoản: ");
            String accNumber = scanner.nextLine();
            Account account = accountManager.findByAccountNumber(accNumber);
            if (account == null) {
                System.out.println("Không tìm thấy tài khoản: " + accNumber);
                return;
            }

            employeeManager.assignAccountToEmployee(e, account);
            System.out.println("Tài khoản " + accNumber + " đã được gán thành công cho nhân viên " +
empId);
        } catch (Exception ex) {
            System.out.println("Lỗi: " + ex.getMessage());
        }
    }
}
```

Cách chạy chương trình

1. Tạo project Java và đặt các file vào trong các package (ví dụ com.bankmanagement như trên).
2. Thêm câu lệnh main ở lớp Main và chạy (Run) chương trình.
3. Tương tác với Console theo các mã chức năng (1.1, 1.2, v.v…) đã mô tả.

Bạn có thể kiểm tra các **Test Case** đã được liệt kê trong đề bài bằng cách nhập tương ứng trong giao diện console.

Hướng dẫn mở rộng/triển khai với Database

- Thay vì dùng List<Account> hay List<Employee> trong memory, bạn có thể sử dụng JDBC (với MySQL, PostgreSQL, v.v…) hoặc ORM (Hibernate, JPA).
- Tạo các DAO (Data Access Object) với CRUD methods (create, read, update, delete).
- Ở các hàm addAccount, deleteAccount, updateAccountHolder,… thay vì thao tác trên List, hãy gọi các DAO để thực hiện query tới database.
- Tương tự với TransactionManager, thay vì lưu transactions trong List, ta có thể lưu vào bảng transactions trong database.

Trên đây là **khung** triển khai cơ bản cho bài tập. Bạn có thể tùy chỉnh, bổ sung chức năng, kiểm tra ngoại lệ, sắp xếp, phân trang, v.v… cho phù hợp với yêu cầu thực tế. Chúc bạn hoàn thành tốt bài tập!

Dưới đây là ví dụ triển khai **JDBC với MySQL** cho bài tập "Hệ Thống Quản Lý Tài Khoản Ngân Hàng". Ý tưởng chính là thay vì lưu trữ dữ liệu "trong bộ nhớ" (sử dụng List), chúng ta sẽ thao tác trực tiếp với **CSDL MySQL** thông qua **JDBC**.

Lưu ý chung

1. Bạn cần cài đặt **MySQL** trên máy và tạo **database** phù hợp.
2. Thêm **JDBC driver** (ví dụ: mysql-connector-java-x.x.x.jar) vào classpath.
3. Các đoạn code dưới đây chỉ là **skeleton** minh họa; bạn cần điều chỉnh lại cho khớp với cấu trúc project, tên package, exception handling, v.v…

---

1. Chuẩn Bị CSDL (Database Schema)

Ví dụ, tạo một schema tên là **bank_db** và các bảng như sau:

-- Tạo Database (nếu chưa có)

CREATE DATABASE IF NOT EXISTS bank_db;

USE bank_db;

```
-- Bảng Nhân viên
CREATE TABLE IF NOT EXISTS employees (
    id VARCHAR(10) PRIMARY KEY,
    name VARCHAR(100),
    salary DOUBLE,
    role VARCHAR(50)  -- Có thể dùng để phân loại: Teller, Manager...
);

-- Bảng Tài khoản
CREATE TABLE IF NOT EXISTS accounts (
    accountNumber VARCHAR(20) PRIMARY KEY,
    accountHolder VARCHAR(100),
    balance DOUBLE,
    employeeInChargeId VARCHAR(10),
    CONSTRAINT fk_employee_in_charge
      FOREIGN KEY (employeeInChargeId) REFERENCES employees(id)
      ON DELETE SET NULL ON UPDATE CASCADE
);

-- Bảng Giao dịch
CREATE TABLE IF NOT EXISTS transactions (
    transactionId VARCHAR(20) PRIMARY KEY,
    accountNumber VARCHAR(20),
    employeeId VARCHAR(10),
    type VARCHAR(20),      -- Deposit, Withdraw, Transfer-In, Transfer-Out
    amount DOUBLE,
    timestamp DATETIME,
    CONSTRAINT fk_account
      FOREIGN KEY (accountNumber) REFERENCES accounts(accountNumber)
      ON DELETE CASCADE ON UPDATE CASCADE,
```

```
    CONSTRAINT fk_employee
        FOREIGN KEY (employeeId) REFERENCES employees(id)
        ON DELETE SET NULL ON UPDATE CASCADE
);
```

Bạn có thể điều chỉnh **các kiểu dữ liệu** (VARCHAR, DOUBLE, v.v…) và **ràng buộc** (FOREIGN KEY, ON DELETE CASCADE, ...) cho phù hợp với yêu cầu chi tiết.

---

2. Cấu Hình Kết Nối JDBC

Tạo một lớp tiện ích (hoặc có thể để chung trong DAO) để quản lý Connection:

```
package com.bankmanagement.utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/bank_db";
    private static final String USER = "root";
    private static final String PASSWORD = "123456"; // Thay bằng mật khẩu MySQL của bạn

    public static Connection getConnection() throws SQLException {
        // Trong MySQL connector mới không bắt buộc gọi Class.forName() nữa
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

- Thay bank_db bằng tên DB bạn tạo.
- Thay **USER** và **PASSWORD** cho đúng với cấu hình MySQL của bạn.

---

3. Triển Khai EmployeeDAO (Ví dụ)

Dưới đây là một ví dụ về lớp **DAO** để quản lý nhân viên (thêm, tìm kiếm, lấy danh sách).

```
package com.bankmanagement.dao;

import com.bankmanagement.models.Employee;
import com.bankmanagement.models.Teller;   // Hoặc Manager, tùy bạn
import com.bankmanagement.utils.DBConnection;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeDAO {

    // Thêm nhân viên
    public void insertEmployee(Employee e) throws SQLException {
        String sql = "INSERT INTO employees (id, name, salary, role) VALUES (?,?,?,?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, e.getId());
            pstmt.setString(2, e.getName());
            pstmt.setDouble(3, e.getSalary());
            // Ở đây có thể dùng e.getClass().getSimpleName() hoặc 1 thuộc tính role
            // Giả sử ta set cứng là "Teller"
            pstmt.setString(4, "Teller");

            pstmt.executeUpdate();
        }
    }

    // Tìm nhân viên theo ID
    public Employee findById(String id) throws SQLException {
```

```
    String sql = "SELECT * FROM employees WHERE id = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, id);

        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            String eId = rs.getString("id");
            String name = rs.getString("name");
            double salary = rs.getDouble("salary");
            String role = rs.getString("role");

            // Tùy vào role, bạn có thể tạo đối tượng Employee phù hợp
            // Ở đây ví dụ role = Teller
            if ("Teller".equalsIgnoreCase(role)) {
                return new Teller(eId, name, salary);
            } else {
                // Hoặc return new Manager(eId, name, salary);
                return new Teller(eId, name, salary);
            }
        }
    }
    return null;
}

// Lấy tất cả nhân viên
public List<Employee> getAllEmployees() throws SQLException {
    List<Employee> list = new ArrayList<>();
    String sql = "SELECT * FROM employees";
    try (Connection conn = DBConnection.getConnection();
```

```
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                String eId = rs.getString("id");
                String name = rs.getString("name");
                double salary = rs.getDouble("salary");
                String role = rs.getString("role");

                // Tạo đối tượng Employee tuỳ theo role
                if ("Teller".equalsIgnoreCase(role)) {
                    list.add(new Teller(eId, name, salary));
                } else {
                    // Hoặc Manager...
                    list.add(new Teller(eId, name, salary));
                }
            }
        }
        return list;
    }
}
```

---

4. Triển Khai AccountDAO

Tương tự, ta có lớp **DAO** để quản lý tài khoản (thêm, sửa, xóa, tìm kiếm...).

```
package com.bankmanagement.dao;

import com.bankmanagement.models.Account;
import com.bankmanagement.models.Employee;
import com.bankmanagement.utils.DBConnection;

import java.sql.*;
```

```
public class AccountDAO {


    // Thêm tài khoản
    public void insertAccount(Account account) throws SQLException {
        String sql = "INSERT INTO accounts (accountNumber, accountHolder, balance, employeeInChargeId) VALUES (?,?,?,?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, account.getAccountNumber());
            pstmt.setString(2, account.getAccountHolder());
            pstmt.setDouble(3, account.getBalance());
            // Lưu id của nhân viên phụ trách
            String empId = (account.getEmployeeInCharge() != null) ? account.getEmployeeInCharge().getId() : null;
            pstmt.setString(4, empId);


            pstmt.executeUpdate();
        }
    }


    // Tìm tài khoản theo accountNumber
    public Account findByAccountNumber(String accountNumber, Employee employee) throws SQLException {
        String sql = "SELECT * FROM accounts WHERE accountNumber = ?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, accountNumber);


            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
```

```
            String accNum = rs.getString("accountNumber");
            String accHolder = rs.getString("accountHolder");
            double balance = rs.getDouble("balance");
            String empId = rs.getString("employeeInChargeId");

            // Ở đây bạn cần tìm Employee theo empId từ EmployeeDAO
            // Hoặc nếu có sẵn Employee, ta dùng tạm => Thực tế nên query ra
            // Dưới đây giả sử ta xài employee tạm truyền vào
            return new Account(accNum, accHolder, balance, employee);
        }
    }
    return null;
}

// Hoặc viết hàm findByAccountNumber (không truyền Employee):
public Account findByAccountNumber(String accountNumber) throws SQLException {
    String sql = "SELECT * FROM accounts WHERE accountNumber = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, accountNumber);

        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            String accNum = rs.getString("accountNumber");
            String accHolder = rs.getString("accountHolder");
            double balance = rs.getDouble("balance");
            String empId = rs.getString("employeeInChargeId");

            // TODO: Lấy employeeInCharge từ DB
            // Chẳng hạn: EmployeeDAO eDao = new EmployeeDAO();
```

```
            // Employee e = eDao.findById(empId);

            // Ở ví dụ này ta để null tạm
            return new Account(accNum, accHolder, balance, null);
        }
    }
    return null;
}

// Update tên chủ tài khoản
public void updateAccountHolder(String accountNumber, String newHolder) throws SQLException {
    String sql = "UPDATE accounts SET accountHolder = ? WHERE accountNumber = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, newHolder);
        pstmt.setString(2, accountNumber);
        pstmt.executeUpdate();
    }
}

// Xóa tài khoản
public void deleteAccount(String accountNumber) throws SQLException {
    String sql = "DELETE FROM accounts WHERE accountNumber = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, accountNumber);
        pstmt.executeUpdate();
    }
}
```

```
    // Update balance (khi nạp/rút/chuyển)
    public void updateBalance(String accountNumber, double newBalance) throws SQLException {
        String sql = "UPDATE accounts SET balance = ? WHERE accountNumber = ?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, newBalance);
            pstmt.setString(2, accountNumber);
            pstmt.executeUpdate();
        }
    }
}
```

Giải thích: Khi nạp tiền, rút tiền hoặc chuyển khoản, bạn cần:

1. Tìm Account trong DB (để kiểm tra balance, v.v…).
2. Cộng / trừ số dư balance.
3. Gọi hàm updateBalance(...) để lưu số dư mới.
4. Tạo 1 bản ghi Transaction tương ứng trong bảng transactions.

---

5. Triển Khai TransactionDAO

Cũng tương tự, ta cần ghi lại các bản ghi giao dịch vào bảng transactions.

```
package com.bankmanagement.dao;

import com.bankmanagement.models.Transaction;
import com.bankmanagement.utils.DBConnection;

import java.sql.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
```

```
public class TransactionDAO {


    // Thêm 1 giao dịch
    public void insertTransaction(Transaction tx) throws SQLException {
        String sql = "INSERT INTO transactions (transactionId, accountNumber, employeeId, type, amount, timestamp) " +
                     "VALUES (?,?,?,?,?,?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, tx.getTransactionId());
            pstmt.setString(2, tx.getAccount().getAccountNumber());
            pstmt.setString(3, (tx.getEmployee() != null) ? tx.getEmployee().getId() : null);
            pstmt.setString(4, tx.getType());
            pstmt.setDouble(5, tx.getAmount());

            // Chuyển LocalDateTime thành dạng DATETIME
            // Ở đây minh họa format yyyy-MM-dd HH:mm:ss
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
            pstmt.setString(6, tx.getTimestamp().format(formatter));

            pstmt.executeUpdate();
        }
    }

    // Lấy tất cả giao dịch (có thể sắp xếp theo timestamp)
    public List<Transaction> getAllTransactions() throws SQLException {
        List<Transaction> list = new ArrayList<>();
        String sql = "SELECT * FROM transactions ORDER BY timestamp DESC";
        try (Connection conn = DBConnection.getConnection();
             Statement stmt = conn.createStatement();
```

```
            ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                String tId = rs.getString("transactionId");
                String accNumber = rs.getString("accountNumber");
                String empId = rs.getString("employeeId");
                String type = rs.getString("type");
                double amount = rs.getDouble("amount");
                Timestamp ts = rs.getTimestamp("timestamp");
                LocalDateTime dateTime = ts.toLocalDateTime();

                // TODO: Tìm Account, Employee từ DB
                // Account account = ...
                // Employee employee = ...
                // Tạm thời để null
                list.add(new Transaction(tId, null, null, type, amount) {
                    {
                        // Chèn timestamp thủ công (override giá trị trong constructor)
                        // Vì constructor gốc set LocalDateTime.now()
                        // Ta có thể tạo constructor khác cho Transaction
                    }
                });
                // Lưu ý: Code trên chỉ minh họa, bạn cần override timestamp hoặc có constructor cho
timestamp
            }
        }
        return list;
    }

    // Lấy lịch sử giao dịch theo 1 tài khoản
```

```
    public List<Transaction> getTransactionsByAccount(String accountNumber) throws SQLException {
        List<Transaction> list = new ArrayList<>();
        String sql = "SELECT * FROM transactions WHERE accountNumber = ? ORDER BY timestamp DESC";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, accountNumber);

            ResultSet rs = pstmt.executeQuery();
            while (rs.next()) {
                String tId = rs.getString("transactionId");
                String empId = rs.getString("employeeId");
                String type = rs.getString("type");
                double amount = rs.getDouble("amount");
                Timestamp ts = rs.getTimestamp("timestamp");
                LocalDateTime dateTime = ts.toLocalDateTime();

                // Tạo đối tượng Transaction, v.v…
                // Tương tự phần trên
            }
        }
        return list;
    }
}
```

Do **Transaction** trong code gốc **không** có constructor truyền timestamp, bạn có thể tạo thêm một constructor khác cho việc load từ DB:

```
public Transaction(String transactionId,
                   Account account,
                   Employee employee,
                   String type,
                   double amount,
```

```
            LocalDateTime timestamp) {
    this.transactionId = transactionId;
    this.account = account;
    this.employee = employee;
    this.type = type;
    this.amount = amount;
    this.timestamp = timestamp;
}
```

---

6. Tích Hợp Trong Lớp "Service" / "Manager" (Hoặc Trực Tiếp Trong Main)

Ví dụ: AccountManager gọi AccountDAO

```
package com.bankmanagement.services;

import com.bankmanagement.dao.AccountDAO;
import com.bankmanagement.models.Account;
import java.sql.SQLException;

public class AccountManager {
    private AccountDAO accountDAO;

    public AccountManager() {
        this.accountDAO = new AccountDAO();
    }

    // Thêm tài khoản
    public void addAccount(Account account) throws SQLException {
        // Kiểm tra trùng lặp => gọi findByAccountNumber
        Account existing = accountDAO.findByAccountNumber(account.getAccountNumber());
        if (existing != null) {
            throw new SQLException("Số tài khoản đã tồn tại: " + account.getAccountNumber());
```

```
        }
        accountDAO.insertAccount(account);
    }

    // Sửa tên chủ tài khoản
    public void updateAccountHolder(String accountNumber, String newHolder) throws SQLException {
        Account a = accountDAO.findByAccountNumber(accountNumber);
        if (a == null) {
            throw new SQLException("Không tìm thấy tài khoản: " + accountNumber);
        }
        accountDAO.updateAccountHolder(accountNumber, newHolder);
    }

    // Xóa tài khoản
    public void deleteAccount(String accountNumber) throws SQLException {
        // Bạn nên kiểm tra balance hoặc transaction ở đây,
        // tùy logic yêu cầu (lấy balance => if > 0 => throw exception)
        accountDAO.deleteAccount(accountNumber);
    }

    // ...
}
```

Ví dụ: TransactionManager gọi TransactionDAO

```
package com.bankmanagement.services;

import com.bankmanagement.dao.AccountDAO;
import com.bankmanagement.dao.TransactionDAO;
import com.bankmanagement.models.Account;
import com.bankmanagement.models.Employee;
import com.bankmanagement.models.Transaction;
```

```
import java.sql.SQLException;
import java.time.LocalDateTime;

public class TransactionManager {
    private TransactionDAO transactionDAO;
    private AccountDAO accountDAO;

    public TransactionManager() {
        this.transactionDAO = new TransactionDAO();
        this.accountDAO = new AccountDAO();
    }

    // Nạp tiền
    public void deposit(String accountNumber, double amount, Employee employee) throws
SQLException {
        // Tìm account
        Account acc = accountDAO.findByAccountNumber(accountNumber);
        if (acc == null) {
            throw new SQLException("Không tìm thấy tài khoản: " + accountNumber);
        }
        // Kiểm tra amount
        if (amount <= 0) {
            throw new SQLException("Số tiền nạp phải > 0");
        }
        // Cộng balance
        double newBalance = acc.getBalance() + amount;
        accountDAO.updateBalance(accountNumber, newBalance);

        // Tạo transaction
```

```
        // ID => Tùy generate
        String txId = "T" + System.currentTimeMillis();
        Transaction tx = new Transaction(txId, acc, employee, "Deposit", amount, LocalDateTime.now());
        transactionDAO.insertTransaction(tx);
    }

    // Rút tiền
    public void withdraw(String accountNumber, double amount, Employee employee) throws
SQLException {
        Account acc = accountDAO.findByAccountNumber(accountNumber);
        if (acc == null) {
            throw new SQLException("Không tìm thấy tài khoản: " + accountNumber);
        }
        if (amount <= 0) {
            throw new SQLException("Số tiền rút phải > 0");
        }
        if (acc.getBalance() < amount) {
            throw new SQLException("Số dư không đủ để rút tiền.");
        }
        double newBalance = acc.getBalance() - amount;
        accountDAO.updateBalance(accountNumber, newBalance);

        String txId = "T" + System.currentTimeMillis();
        Transaction tx = new Transaction(txId, acc, employee, "Withdraw", amount, LocalDateTime.now());
        transactionDAO.insertTransaction(tx);
    }

    // Chuyển khoản
    public void transfer(String senderAccNum, String receiverAccNum, double amount, Employee
employee) throws SQLException {
        Account sender = accountDAO.findByAccountNumber(senderAccNum);
```

```
        Account receiver = accountDAO.findByAccountNumber(receiverAccNum);
        if (sender == null || receiver == null) {
            throw new SQLException("Tài khoản gửi/nhận không tồn tại.");
        }
        if (amount <= 0) {
            throw new SQLException("Số tiền chuyển phải > 0");
        }
        if (sender.getBalance() < amount) {
            throw new SQLException("Số dư không đủ.");
        }
        // Update balance
        double senderNewBalance = sender.getBalance() - amount;
        double receiverNewBalance = receiver.getBalance() + amount;

        // Cập nhật DB
        accountDAO.updateBalance(senderAccNum, senderNewBalance);
        accountDAO.updateBalance(receiverAccNum, receiverNewBalance);

        // Tạo transaction "Transfer-Out" cho sender
        String txOutId = "T" + System.currentTimeMillis();
        Transaction txOut = new Transaction(txOutId, sender, employee, "Transfer-Out", amount,
LocalDateTime.now());
        transactionDAO.insertTransaction(txOut);

        // Tạo transaction "Transfer-In" cho receiver
        String txInId = "T" + (System.currentTimeMillis() + 1);
        Transaction txIn = new Transaction(txInId, receiver, employee, "Transfer-In", amount,
LocalDateTime.now());
        transactionDAO.insertTransaction(txIn);
    }
```

```
    // ...
}
```

---

7. Tích Hợp Vào Lớp Main

Ở lớp Main (có hàm public static void main), thay vì thao tác trên List, ta sẽ gọi đến các **Manager / Service**. Bên trong mỗi manager/service lại gọi đến **DAO** (JDBC).

Ví dụ:

```
package com.bankmanagement;

import com.bankmanagement.models.Account;
import com.bankmanagement.models.Employee;
import com.bankmanagement.models.Teller;
import com.bankmanagement.services.AccountManager;
import com.bankmanagement.services.TransactionManager;

import java.sql.SQLException;
import java.util.Scanner;

public class Main {
    static Scanner scanner = new Scanner(System.in);
    static AccountManager accountManager = new AccountManager();
    static TransactionManager transactionManager = new TransactionManager();
    // Còn EmployeeManager cũng tương tự...

    public static void main(String[] args) {
        // Giả sử đã có nhân viên E001
        Employee e1 = new Teller("E001", "Alice", 10000);
        // TODO: Thêm e1 vào DB employees (gọi EmployeeDAO hay EmployeeManager)...

        // Hiển thị menu loop...
```

```
    boolean running = true;
    while (running) {
        printMenu();
        String choice = scanner.nextLine();
        switch (choice) {
            case "1.1":
                addNewAccount();
                break;
            case "2.1":
                deposit();
                break;
            // ...
            case "5.1":
                System.out.println("Thoát chương trình.");
                running = false;
                break;
            default:
                System.out.println("Lựa chọn không hợp lệ!");
        }
    }
}

private static void printMenu() {
    // In menu
}

private static void addNewAccount() {
    try {
        System.out.print("Nhập số tài khoản: ");
        String accNum = scanner.nextLine();
```

```
        System.out.print("Nhập tên chủ tài khoản: ");
        String holder = scanner.nextLine();
        System.out.print("Nhập số dư ban đầu: ");
        double balance = Double.parseDouble(scanner.nextLine());

        // Giả sử gán cho nhân viên E001
        Employee e = new Teller("E001", "Alice", 10000);
        // Thực tế, nên tìm trong DB => Employee e = employeeDAO.findById("E001");

        Account acc = new Account(accNum, holder, balance, e);
        accountManager.addAccount(acc);

        System.out.println("Thêm tài khoản thành công!");
    } catch (SQLException ex) {
        System.out.println("Lỗi SQL: " + ex.getMessage());
    }
}

private static void deposit() {
    try {
        System.out.print("Nhập số tài khoản: ");
        String accNum = scanner.nextLine();
        System.out.print("Nhập số tiền nạp: ");
        double amount = Double.parseDouble(scanner.nextLine());

        // Tạm gán cho employee E001
        Employee e = new Teller("E001", "Alice", 10000);
        transactionManager.deposit(accNum, amount, e);
        System.out.println("Nạp tiền thành công!");
    } catch (SQLException ex) {
```

```
            System.out.println("Lỗi SQL: " + ex.getMessage());
        }
    }
}
```

Tóm lược các bước tích hợp JDBC:

1. **Tạo CSDL MySQL** và **các bảng** employees, accounts, transactions.
2. Viết lớp **DBConnection** để lấy Connection.
3. Cho **mỗi entity** (Employee, Account, Transaction) bạn tạo **1 lớp DAO** với các **CRUD methods**.
4. Trong các lớp **Service** (hoặc Manager), khi thực hiện **nghiệp vụ** (thêm, sửa, xóa, nạp, rút, chuyển…) thì:
    - Gọi **DAO** để **lấy** hoặc **cập nhật** dữ liệu trong DB.
    - Xử lý logic (vd: kiểm tra số dư) xong thì **lưu transaction** vào DB.
5. Ở **lớp Main**, khi người dùng chọn chức năng, bạn gọi sang các phương thức **Service** (thay vì thao tác trên List).

---

8. Lưu Ý Triển Khai Thực Tế

- **Transaction** (giao dịch) thường cần **Transaction SQL** (cùng tên nhưng khác ngữ cảnh).
    - Nếu bạn muốn **đảm bảo** tính **ACID** (vd: chuyển khoản phải "rút" và "nạp" cùng thành công hoặc cùng thất bại), bạn nên dùng **transaction** trong SQL (setAutoCommit(false), commit(), rollback()) khi thực hiện rút-tiền + nạp-tiền.
- Xử lý **ngoại lệ** cẩn thận (cả SQLException lẫn RuntimeException).
- Có thể dùng **ORM** (Hibernate/JPA) để giảm bớt code JDBC thuần.

Trên đây là hướng dẫn cơ bản cách chuyển từ lưu trữ trong bộ nhớ sang sử dụng **JDBC với MySQL**. Bạn hãy điều chỉnh tùy ý để phù hợp với project (cấu trúc package, exception handling, logging, v.v…).

Để tránh phải nhập data nhiều lần. tôi nghĩ bạn có thể tạo table rồi đọc dữ liệu ra sử dụng mysql và jdbc. Tránh phải nhập đi nhập lại data

Dưới đây là hướng dẫn **tích hợp MySQL + JDBC** để **không phải nhập thủ công dữ liệu** sau mỗi lần chạy. Ý tưởng tổng quát:

1. **Tạo bảng** trong MySQL (hoặc RDBMS khác) để lưu trữ dữ liệu cho các entity (Account, Employee, Transaction).
2. **Kết nối** Java với MySQL qua JDBC.
3. **Viết các DAO (Data Access Object)** hoặc Repository để thực hiện CRUD (Create, Read, Update, Delete) trên cơ sở dữ liệu.
4. **Sử dụng DAO** trong các lớp manager (AccountManager, EmployeeManager, TransactionManager) thay vì thao tác trên List<...>.
5. Mỗi lần chạy chương trình, ta **đọc dữ liệu** từ database (thay vì nhập tay) và **đổ vào** các đối tượng tương ứng.

1. Cấu trúc bảng MySQL ví dụ

1.1. Bảng employees

```
CREATE TABLE employees (
   id VARCHAR(10) PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   salary DOUBLE NOT NULL,
   role VARCHAR(20) NOT NULL  -- ví dụ: 'TELLER' hoặc 'MANAGER'
);
```

1.2. Bảng accounts

```
CREATE TABLE accounts (
   account_number VARCHAR(20) PRIMARY KEY,
   account_holder VARCHAR(100) NOT NULL,
   balance DOUBLE NOT NULL,
   employee_id VARCHAR(10),  -- FK tới employees
   CONSTRAINT fk_employee_id
      FOREIGN KEY (employee_id) REFERENCES employees(id)
);
```

1.3. Bảng transactions

```
CREATE TABLE transactions (
    transaction_id VARCHAR(20) PRIMARY KEY,
    account_number VARCHAR(20) NOT NULL,
    employee_id VARCHAR(10) NOT NULL,
    type VARCHAR(20) NOT NULL,       -- Deposit, Withdraw, Transfer-Out, Transfer-In
    amount DOUBLE NOT NULL,
    timestamp DATETIME NOT NULL,
    CONSTRAINT fk_acc_number
        FOREIGN KEY (account_number) REFERENCES accounts(account_number),
    CONSTRAINT fk_emp_id
        FOREIGN KEY (employee_id) REFERENCES employees(id)
);
```

2. Tạo lớp DBConnection (Singleton) để quản lý kết nối JDBC

Bạn tạo 1 class chuyên quản lý Connection, ví dụ:

```
package com.bankmanagement.db;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static final String URL = "jdbc:mysql://localhost:3306/bank_db?useSSL=false";
    private static final String USER = "root";
    private static final String PASS = "123456";
    private static Connection connection;

    // private constructor
    private DBConnection() {}

    public static Connection getConnection() throws SQLException {
```

```
        if (connection == null || connection.isClosed()) {
            try {
                Class.forName("com.mysql.cj.jdbc.Driver"); // nạp driver
                connection = DriverManager.getConnection(URL, USER, PASS);
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }
}
```

Lưu ý: Thay URL, USER, PASS bằng thông tin MySQL của bạn.

3. Ví dụ DAO cho Employee (EmployeeDAO)

Thay vì lưu List<Employee>, ta sẽ thao tác với CSDL qua JDBC. Thường mỗi entity sẽ có 1 DAO riêng,

ví dụ:

```
package com.bankmanagement.dao;
import com.bankmanagement.db.DBConnection;
import com.bankmanagement.models.Employee;
import com.bankmanagement.models.Manager;
import com.bankmanagement.models.Teller;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeDAO {
    // Thêm nhân viên
    public void addEmployee(Employee e) throws SQLException {
        String sql = "INSERT INTO employees (id, name, salary, role) VALUES (?, ?, ?, ?)";
```

```
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, e.getId());
        ps.setString(2, e.getName());
        ps.setDouble(3, e.getSalary());
        // Tùy bạn mô tả role = "TELLER" hay "MANAGER"
        // Giả sử ta check instance:
        if (e instanceof Teller) {
            ps.setString(4, "TELLER");
        } else {
            ps.setString(4, "MANAGER");
        }
        ps.executeUpdate();
    }
}

// Lấy toàn bộ nhân viên
public List<Employee> getAllEmployees() throws SQLException {
    List<Employee> list = new ArrayList<>();
    String sql = "SELECT * FROM employees";
    try (Connection conn = DBConnection.getConnection();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sql)) {
        while (rs.next()) {
            String id = rs.getString("id");
            String name = rs.getString("name");
            double salary = rs.getDouble("salary");
            String role = rs.getString("role");
            Employee e = null;
            if ("TELLER".equalsIgnoreCase(role)) {
```

```
                e = new Teller(id, name, salary);
            } else {
                e = new Manager(id, name, salary);
            }
            list.add(e);
        }
    }
    return list;
}

// Tìm nhân viên theo ID
public Employee findById(String empId) throws SQLException {
    String sql = "SELECT * FROM employees WHERE id = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, empId);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                String name = rs.getString("name");
                double salary = rs.getDouble("salary");
                String role = rs.getString("role");
                if ("TELLER".equalsIgnoreCase(role)) {
                    return new Teller(empId, name, salary);
                } else {
                    return new Manager(empId, name, salary);
                }
            }
        }
    }
    return null;
```

```
    }

    // Update, Delete, ... v.v... (tương tự)
}
```

4. Ví dụ DAO cho Account (AccountDAO)

```
package com.bankmanagement.dao;

import com.bankmanagement.db.DBConnection;

import com.bankmanagement.models.Account;

import com.bankmanagement.models.Employee;


import java.sql.*;

import java.util.ArrayList;

import java.util.List;


public class AccountDAO {

    public void addAccount(Account acc) throws SQLException {

        String sql = "INSERT INTO accounts (account_number, account_holder, balance, employee_id)
VALUES (?,?,?,?)";

        try (Connection conn = DBConnection.getConnection();

             PreparedStatement ps = conn.prepareStatement(sql)) {

            ps.setString(1, acc.getAccountNumber());

            ps.setString(2, acc.getAccountHolder());

            ps.setDouble(3, acc.getBalance());

            ps.setString(4, acc.getEmployeeInCharge() != null ? acc.getEmployeeInCharge().getId() : null);

            ps.executeUpdate();

        }

    }


    public Account findByAccountNumber(String accountNumber, EmployeeDAO empDAO) throws
SQLException {

        String sql = "SELECT * FROM accounts WHERE account_number = ?";
```

```
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, accountNumber);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                String holder = rs.getString("account_holder");
                double bal = rs.getDouble("balance");
                String empId = rs.getString("employee_id");

                Employee emp = null;
                if (empId != null) {
                    emp = empDAO.findById(empId);
                }
                // Tạo đối tượng Account
                Account acc = new Account(accountNumber, holder, bal, emp);
                return acc;
            }
        }
    }
    return null;
}

public void updateAccountHolder(String accountNumber, String newHolder) throws SQLException {
    String sql = "UPDATE accounts SET account_holder = ? WHERE account_number = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, newHolder);
        ps.setString(2, accountNumber);
        ps.executeUpdate();
    }
```

```
}

public void deleteAccount(String accountNumber) throws SQLException {
    String sql = "DELETE FROM accounts WHERE account_number = ?";
    try (Connection conn = DBConnection.getConnection();
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, accountNumber);
        ps.executeUpdate();
    }
}

public List<Account> getAllAccounts(EmployeeDAO empDAO) throws SQLException {
    List<Account> list = new ArrayList<>();
    String sql = "SELECT * FROM accounts";
    try (Connection conn = DBConnection.getConnection();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sql)) {
        while (rs.next()) {
            String accNumber = rs.getString("account_number");
            String holder = rs.getString("account_holder");
            double bal = rs.getDouble("balance");
            String empId = rs.getString("employee_id");

            Employee emp = null;
            if (empId != null) {
                emp = empDAO.findById(empId);
            }
            Account acc = new Account(accNumber, holder, bal, emp);
            list.add(acc);
        }
```

```
        }
        return list;
    }

    // ... v.v... (các hàm khác nếu cần).
}
```

5. Lấy dữ liệu từ CSDL và "nạp vào" Manager

Trong lớp Main (hoặc một lớp Service khởi tạo), trước khi hiển thị menu, ta có thể:

1. Gọi employeeDAO.getAllEmployees(), sau đó nạp vào employeeManager.
2. Gọi accountDAO.getAllAccounts(), sau đó nạp vào accountManager.
3. Tương tự với transactionDAO (nếu có) để nạp các giao dịch cũ vào transactionManager.

Ví dụ:

```
// Lúc khởi động chương trình
EmployeeDAO empDAO = new EmployeeDAO();
AccountDAO accDAO = new AccountDAO();

try {
    // Lấy toàn bộ nhân viên từ DB
    List<Employee> empList = empDAO.getAllEmployees();
    for (Employee e : empList) {
        employeeManager.addEmployeeInMemory(e);
        // "addEmployeeInMemory" là hàm ta tự đặt, nó chỉ add vào List<Employee> chứ không insert DB
        // (vì DB đã insert sẵn)
    }
    // Lấy toàn bộ Account từ DB
    List<Account> accList = accDAO.getAllAccounts(empDAO);
    for (Account a : accList) {
        accountManager.addAccountInMemory(a);
    }
} catch (SQLException ex) {
```

```
    ex.printStackTrace();
}
```

Khi đó, bạn **đã có** danh sách Employee, Account sẵn trong Manager. Lúc này:

- Bạn **không cần** nhập tay các nhân viên "Alice", "Bob" nữa.
- Dữ liệu sẽ được **giữ lại** trong DB. Mỗi khi chạy lại, chương trình **đọc** ra.
- Khi thêm/sửa/xóa, bạn gọi các DAO tương ứng để DB được cập nhật ngay.

6. Lưu ý chung

1. **Bắt lỗi**: JDBC ném SQLException, nên luôn try-catch.
2. **Transaction**: Trong dự án thực tế, việc rút tiền, chuyển khoản,... có thể cần transaction ở cấp DB để đảm bảo tính toàn vẹn. (BEGIN TRANSACTION, COMMIT, ROLLBACK).
3. **Mã logic**: Thay vì transactionManager.deposit(), ta có thể di chuyển logic deposit vào AccountService + DAO, rồi ghi transaction vào bảng transactions.
4. **ORM**: Nếu dùng Hibernate/JPA, ta có thể "map" các entity Employee, Account, Transaction sang table. Lúc đó code sẽ "gọn" hơn so với JDBC thuần.

Tóm tắt

- **Tạo bảng**: employees, accounts, transactions
- **Viết class DBConnection** để kết nối MySQL
- **Xây DAO** cho mỗi entity, chứa các hàm CRUD
- **Tại Main**, thay vì thêm cứng dữ liệu vào List, ta **đọc** từ DB qua DAO, **nạp vào** Manager.
- Với cách này, **dữ liệu sẽ còn mãi** trong DB, **không phải nhập lại** sau mỗi lần chạy.

Trên đây là **khung** hướng dẫn cơ bản. Bạn có thể dựa vào để hoàn thiện dự án, tùy chỉnh theo nhu cầu và kiến trúc mong muốn (DAO, Service, Repository, v.v.).