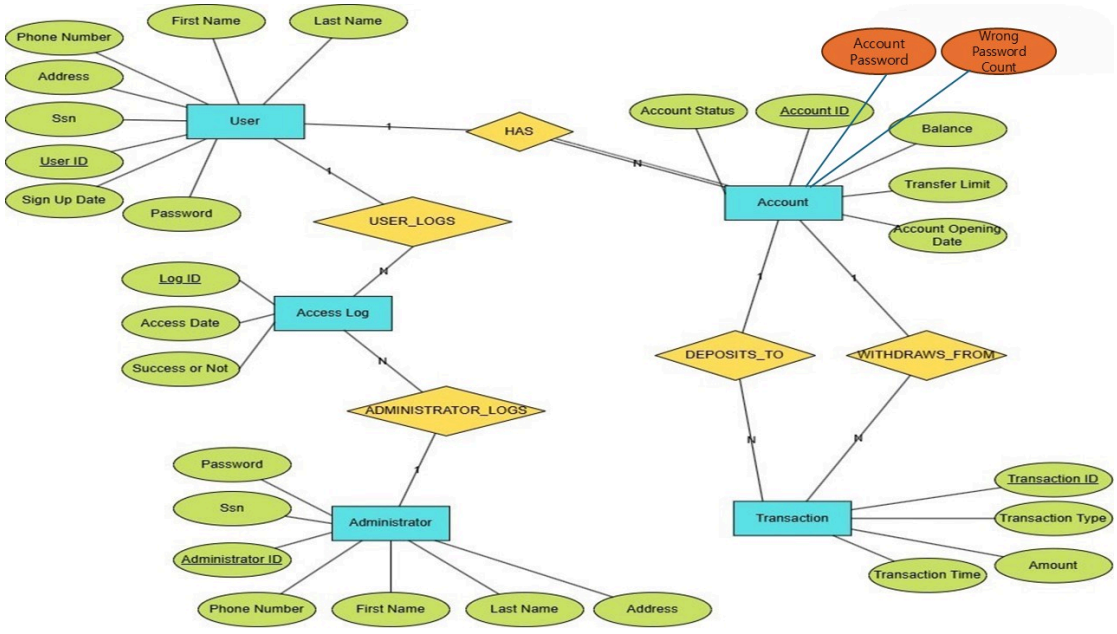
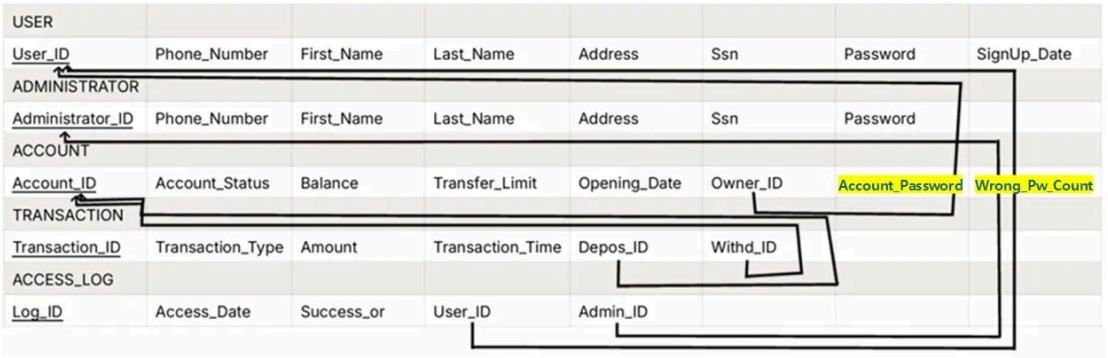
DB application assignment 3

2021007138_서교빈_P4_보고서

1. ER Diagram / Relational Diagram



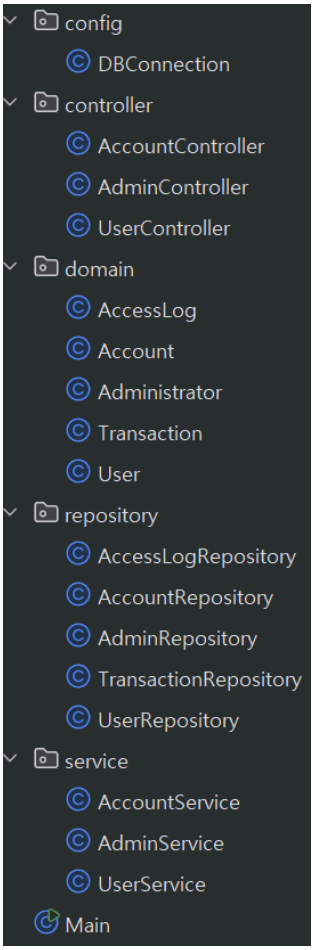
- 기존 Diagram에서 변경된 점은 Account Entity에 Account password, Wrong Password Count 속성을 포함했다. 후술할 기능 설명에서 각각 계좌 비밀번호, 연속 비밀번호 오류 횟수의 기능을 담당한다.



- 기존 Diagram와 참조가 변경된 점은 없다. ACCOUNT table에 Account_Password, Wrong_Pw_Count 속성을 추가했다.

2. 코드

- 프로그램 코드는 MVC 아키텍처 패턴으로 작성했다.
- JDBC 드라이버는 Gradle을 통해 빌드했으며, DBConnection에서 MySQL에 연결한다.
- Java는 23버전을 사용했다.
- 코드에 대한 설명은 상단 글머리와 주석에 표시했다.




```
dependencies {
    testImplementation platform('org.junit:junit-bom:5.10.0')
    testImplementation 'org.junit.jupiter:junit-jupiter'

    implementation 'com.mysql:mysql-connector-j:8.4.0'
    compileOnly 'org.projectlombok:lombok:1.18.30'
    annotationProcessor 'org.projectlombok:lombok:1.18.30'
}
```

```
package com.bank.config;

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

public class DBConnection {

    // DB 접속 정보 설정
    // localhost:3306 : (MySQL 기본 포트)
    // /bank_db : 데이터베이스 이름
    // ?serverTimezone=UTC : 시차 문제로 에러 나는 것을 방지
    private static final String URL = "jdbc:mysql://localhost:3306/bank_db?serverTimezone=Asi
a/Seoul&useSSL=false&allowPublicKeyRetrieval=true";
    private static final String USER = "root";     // MySQL 아이디
    private static final String PASSWORD = "<비밀번호>"; // MySQL 비밀번호

    // 연결 객체 메서드
    public static Connection getConnection() {
        Connection conn = null;
        try {
            // 드라이버 로드 (Gradle에서 mysql-connector-j 라이브러리)
            Class.forName("com.mysql.cj.jdbc.Driver");

            // DB 연결
            conn = DriverManager.getConnection(URL, USER, PASSWORD);

        } catch (ClassNotFoundException e) {
            System.err.println("JDBC 드라이버를 찾을 수 없습니다.");
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("DB 연결 실패. URL, ID, Password 확인");
            e.printStackTrace();
        }
        return conn;
    }
}
```

Domain

- DB연동을 위한 기본적인 Entity 구조만 정의한다. 모든 Entity는 아래와 동일하게 구성된다.

```
package com.bank.domain;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
```

```
import lombok.NoArgsConstructor;

import java.sql.Timestamp;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Account {
    private int accountId;
    private String accountStatus;
    private long balance;
    private long transferLimit;
    private Timestamp openingDate;
    private String ownerId;
    //추가 필드
    private String accountPassword;
    private int wrongPwCount;
}
```

Repository

- 데이터베이스와의 직접적인 통신을 담당한다. Service 계층의 비즈니스 로직에 따라 실제 SQL 쿼리를 작성하고 실행한다. 여러 쿼리에 따른 트랜잭션 관리도 담당한다.

AccessLogRepository.java

- `save()`: 사용자 및 관리자의 로그인 시도 시, 접속 시간과 성공 여부를 기록한다.
- `printAllLogs()`: 시스템에 저장된 모든 접속 로그를 최신순으로 조회하여 출력한다.

```
package com.bank.repository;

import com.bank.config.DBConnection;
import com.bank.domain.AccessLog;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class AccessLogRepository {

    public void save(AccessLog log) {
        String sql = "INSERT INTO access_log (access_date, success_or_not, user_id, admin_id) V
ALUES (?, ?, ?, ?)";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
```

```
            preparedStatement.setTimestamp(1, log.getAccessDate());
            preparedStatement.setBoolean(2, log.isSuccessOrNot());
            // User 로그인이면 userId 넣고, 아니면 NULL
            preparedStatement.setObject(3, log.getUserId());
            // Admin 로그인이면 adminId 넣고, 아니면 NULL
            preparedStatement.setObject(4, log.getAdminId());

            preparedStatement.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // 전체 접속 로그 조회
    public void printAllLogs() {
        String sql = "SELECT * FROM Access_Log ORDER BY access_date DESC";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql);
             ResultSet rs = preparedStatement.executeQuery()) {

            System.out.println("\n--- [시스템 전체 접속 로그] ---");
            System.out.printf("%-5s %-20s %-10s %-10s %-10s\n", "ID", "시간", "성공여부", "User",
"Admin");
            System.out.println("-----------------------------------------------------------");

            while (rs.next()) {
                System.out.printf("%-5d %-20s %-10s %-10s %-10s\n",
                        rs.getInt("log_id"),
                        rs.getTimestamp("access_date"),
                        rs.getBoolean("success_or_not") ? "성공" : "실패",
                        rs.getString("user_id") == null ? "-" : rs.getString("user_id"),
                        rs.getString("admin_id") == null ? "-" : rs.getString("admin_id")
                );
            }
            System.out.println("-----------------------------------------------------------");

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

AccountRepository.java

- `save()`: 새로운 계좌 정보를 생성하여 DB에 저장한다.

- `findAllByOwnerId()` : 특정 사용자가 소유한 모든 계좌 목록을 조회한다.
- `updateBalance()` : 특정 계좌의 잔액을 갱신(입금 시 증가, 출금 시 감소)한다.
- `findById()` : 계좌번호를 통해 해당 계좌의 상세 정보를 조회한다.
- `transfer()` : 출금, 입금, 거래 기록 `INSERT` 를 하나의 트랜잭션으로 묶어 atomic을 보장하며 이체를 처리한다.
  - Atomic을 보장하기 위해 `Connection.setAutoCommit(false)` 상태에서 3개의 쿼리가 모두 성공해야 commit 된다.
- `updateAccountStatus()` : 계좌의 상태(정상, 휴면, 잠금, 해지)를 변경한다.
- `findAll()` : 시스템에 존재하는 모든 계좌 목록을 조회한다.
- `increaseWrongPwCount()` : 계좌 비밀번호 오류 시, 오류 횟수 카운트를 1 증가시킨다.
- `resetWrongPwCount()` : 계좌 비밀번호 인증 성공 시, 오류 횟수 카운트를 0으로 초기화한다.
- `updateAccountPassword()` : 계좌의 비밀번호를 새로운 값으로 변경한다.
- `updateTransferLimit()` : 계좌의 1회 이체 한도 금액을 변경한다.

```
package com.bank.repository;

import com.bank.config.DBConnection;
import com.bank.domain.Account;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class AccountRepository {

    public void save(Account account) {
        String sql = "INSERT INTO Account (account_id, account_status, balance, transfer_limit, o
pening_date, owner_id, account_password) " +
                "VALUES (?, ?, ?, ?, ?, ?, ?)";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setInt(1, account.getAccountId());
            preparedStatement.setString(2, account.getAccountStatus());
            preparedStatement.setLong(3, account.getBalance());
            preparedStatement.setLong(4, account.getTransferLimit());
            preparedStatement.setTimestamp(5, account.getOpeningDate());
            preparedStatement.setString(6, account.getOwnerId()); // 계좌 소유 FK
            preparedStatement.setString(7, account.getAccountPassword());

            int result = preparedStatement.executeUpdate();
```

```
            if (result > 0) {
                System.out.println("[성공] 계좌가 개설되었습니다. 계좌번호: " + account.getAccountId());
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 내 계좌 목록
    public List<Account> findAllByOwnerId(String ownerId) {
        List<Account> accounts = new ArrayList<>();
        String sql = "SELECT * FROM account WHERE owner_id = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, ownerId);

            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                while (resultSet.next()) {
                    Account account = mapRowToAccount(resultSet);
                    accounts.add(account);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return accounts;
    }
    // 잔액 변경 (입금/출금)
    public void updateBalance(int accountId, long amount) {
        // 현재 잔액에 + amount (출금 시 음수)
        String sql = "UPDATE account SET balance = balance + ? WHERE account_id = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setLong(1, amount);
            preparedStatement.setInt(2, accountId);

            int result = preparedStatement.executeUpdate();
            if (result == 0) {
                System.out.println("[오류] 계좌번호를 찾을 수 없습니다.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
    }
    // 계좌 ID로 상세 정보 조회 (비밀번호, 잔액 확인)
    public Account findById(int accountId) {
        String sql = "SELECT * FROM account WHERE account_id = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setInt(1, accountId);

            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                if (resultSet.next()) {
                    return mapRowToAccount(resultSet);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null; // 찾는 계좌 없음
    }

    // 계좌 이체 (트랜잭션: 출금 -> 입금 -> 기록)
    public boolean transfer(int fromId, int toId, long amount) {
        Connection conn = null;
        PreparedStatement preparedStatement = null;

        // 이체 쿼리
        String withdrawSql = "UPDATE account SET balance = balance - ? WHERE account_id =
?"; // 보내는 사람
        String depositSql = "UPDATE account SET balance = balance + ? WHERE account_id = ?";
// 받는 사람
        // 이체 시 depos_id와 withd_id 모두
        String logSql = "INSERT INTO transaction (transaction_type, amount, transaction_time, dep
os_id, withd_id) VALUES ('TRANSFER', ?, ?, ?, ?)";

        try {
            conn = DBConnection.getConnection();

            // 자동 커밋 끄기 (트랜잭션 시작, rollback 적용)
            conn.setAutoCommit(false);

            // 출금 (보내는 사람 감소)
            preparedStatement = conn.prepareStatement(withdrawSql);
            preparedStatement.setLong(1, amount);
            preparedStatement.setInt(2, fromId);
            int withdrawResult = preparedStatement.executeUpdate();
            preparedStatement.close();
```

```
            if (withdrawResult == 0) throw new SQLException("출금 실패");

            // 입금 (받는 사람 증가)
            preparedStatement = conn.prepareStatement(depositSql);
            preparedStatement.setLong(1, amount);
            preparedStatement.setInt(2, toId);
            int depositResult = preparedStatement.executeUpdate();
            preparedStatement.close();

            if (depositResult == 0) throw new SQLException("입금 실패 (존재하지 않는 계좌)");

            // 거래 내역 기록
            preparedStatement = conn.prepareStatement(logSql);
            preparedStatement.setLong(1, amount);
            preparedStatement.setTimestamp(2, new java.sql.Timestamp(System.currentTimeMillis
()));
            preparedStatement.setInt(3, toId);   // 입금 계좌
            preparedStatement.setInt(4, fromId); // 출금 계좌
            preparedStatement.executeUpdate();

            // **모든 단계 성공 시 커밋
            conn.commit();
            System.out.println("[DB 알림] 이체 처리가 완료되었습니다.");
            return true;

        } catch (SQLException e) {
            // 하나라도 실패하면 rollback
            try {
                if (conn != null) conn.rollback();
                System.out.println("[DB 알림] 오류 발생으로 롤백되었습니다.");
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            e.printStackTrace();
            return false;
        } finally {
            // 자원 해제
            try {
                if (preparedStatement != null) preparedStatement.close();
                if (conn != null) {
                    conn.setAutoCommit(true); // 원래대로 복구
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

```
    // 계좌 상태 변경
    public void updateAccountStatus(int accountId, String newStatus) {
        String sql = "UPDATE Account SET account_status = ? WHERE account_id = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, newStatus);
            preparedStatement.setInt(2, accountId);

            int result = preparedStatement.executeUpdate();
            if (result > 0) {
                System.out.println("[성공] 계좌(" + accountId + ") 상태가 '" + newStatus + "'로 변경되었
습니다.");
            } else {
                System.out.println("[오류] 해당 계좌를 찾을 수 없습니다.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 모든 계좌 목록
    public List<Account> findAll() {
        List<Account> accounts = new ArrayList<>();
        String sql = "SELECT * FROM Account ORDER BY account_id ASC";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql);
             ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                Account account = mapRowToAccount(resultSet);
                accounts.add(account);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return accounts;
    }

    // account 매핑 메서드
    private Account mapRowToAccount(ResultSet rs) throws SQLException {
        return Account.builder()
                .accountId(rs.getInt("account_id"))
                .accountStatus(rs.getString("account_status"))
                .balance(rs.getLong("balance"))
                .transferLimit(rs.getLong("transfer_limit"))
```

```
            .openingDate(rs.getTimestamp("opening_date"))
            .ownerId(rs.getString("owner_id"))
            .accountPassword(rs.getString("account_password"))
            // [NEW] 아까 추가한 비밀번호 오류 횟수 매핑
            .wrongPwCount(rs.getInt("wrong_pw_count"))
            .build();
    }
    // 비밀번호 오류 1 증가
    public void increaseWrongPwCount(int accountId) {
        String sql = "UPDATE Account SET wrong_pw_count = wrong_pw_count + 1 WHERE accou
nt_id = ?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
            preparedStatement.setInt(1, accountId);
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // 비밀번호 오류 횟수 초기화
    public void resetWrongPwCount(int accountId) {
        String sql = "UPDATE Account SET wrong_pw_count = 0 WHERE account_id = ?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
            preparedStatement.setInt(1, accountId);
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // 계좌 비밀번호 변경
    public void updateAccountPassword(int accountId, String newPassword) {
        String sql = "UPDATE Account SET account_password = ? WHERE account_id = ?";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
            preparedStatement.setString(1, newPassword);
            preparedStatement.setInt(2, accountId);
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // 이체 한도 변경
    public void updateTransferLimit(int accountId, long newLimit) {
        String sql = "UPDATE Account SET transfer_limit = ? WHERE account_id = ?";
```

```
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
            preparedStatement.setLong(1, newLimit);
            preparedStatement.setInt(2, accountId);
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

AdminRepository.java

- `findByAdminIdAndPassword()` : 관리자 ID와 비밀번호를 통해 관리자 로그인 인증을 수행한다.
- `findByAdminId()` : ID를 통해 특정 관리자의 정보를 조회한다.

```
package com.bank.repository;

import com.bank.config.DBConnection;
import com.bank.domain.Administrator;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class AdminRepository {

    // 관리자 로그인 검증
    public Administrator findByAdminIdAndPassword(String adminId, String password) {
        String sql = "SELECT * FROM Administrator WHERE admin_id = ? AND password = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, adminId);
            preparedStatement.setString(2, password);

            try (ResultSet rs = preparedStatement.executeQuery()) {
                if (rs.next()) {
                    return Administrator.builder()
                            .adminId(rs.getString("admin_id"))
                            .password(rs.getString("password")) // 비번 확인용
                            .firstName(rs.getString("first_name"))
                            .lastName(rs.getString("last_name"))
                            .build();
                }
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    public Administrator findByAdminId(String adminId) {
        String sql = "SELECT * FROM Administrator WHERE admin_id = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, adminId);

            try (ResultSet rs = preparedStatement.executeQuery()) {
                if (rs.next()) {
                    return Administrator.builder()
                            .adminId(rs.getString("admin_id"))
                            .password(rs.getString("password"))
                            .firstName(rs.getString("first_name"))
                            .lastName(rs.getString("last_name"))
                            .build();
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

TransactionRepository.java

- `save()` : 입금, 출금, 이체 발생 시 거래 내역을 DB에 저장한다.
- `findAllByAccountId()` : 특정 계좌가 입금처 or 출금처인 모든 거래 내역을 조회한다.
- `findAll()` : 모든 거래 내역을 최신순으로 조회한다.

```
package com.bank.repository;

import com.bank.config.DBConnection;
import com.bank.domain.Transaction;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
```

```
public class TransactionRepository {

    public void save(Transaction transaction) {
        // 입금, 출금, 이체 모두 처리하는 쿼리
        String sql = "INSERT INTO transaction (transaction_type, amount, transaction_time, depos
_id, withd_id) " +
                "VALUES (?, ?, ?, ?, ?)";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, transaction.getTransactionType());
            preparedStatement.setLong(2, transaction.getAmount());
            preparedStatement.setTimestamp(3, transaction.getTransactionTime());

            // deposId가 null이면 setNull 처리 (입금 시엔 값 있음, 출금 시엔 null)
            if (transaction.getDeposId() != null) {
                preparedStatement.setInt(4, transaction.getDeposId());
            } else {
                preparedStatement.setNull(4, java.sql.Types.INTEGER);
            }

            // withdId가 null이면 setNull 처리 (입금 시엔 null, 출금 시엔 값 있음)
            if (transaction.getWithdId() != null) {
                preparedStatement.setInt(5, transaction.getWithdId());
            } else {
                preparedStatement.setNull(5, java.sql.Types.INTEGER);
            }

            preparedStatement.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    // 특정 계좌의 거래 내역 조회 (입금 + 출금)
    public List<Transaction> findAllByAccountId(int accountId) {
        List<Transaction> transactions = new ArrayList<>();
        // 내 계좌가 입금이거나 출금인 경우 모두 조회
        String sql = "SELECT * FROM Transaction WHERE depos_id = ? OR withd_id = ? ORDER BY
transaction_time DESC";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setInt(1, accountId);
            preparedStatement.setInt(2, accountId);
```

```
        try (ResultSet rs = preparedStatement.executeQuery()) {
            while (rs.next()) {
                transactions.add(mapRowToTransaction(rs));
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return transactions;
}

// 전체 거래 내역 조회
public List<Transaction> findAll() {
    List<Transaction> transactions = new ArrayList<>();
    String sql = "SELECT * FROM Transaction ORDER BY transaction_time DESC";

    try (Connection conn = DBConnection.getConnection();
         PreparedStatement preparedStatement = conn.prepareStatement(sql);
         ResultSet rs = preparedStatement.executeQuery()) {

        while (rs.next()) {
            transactions.add(mapRowToTransaction(rs));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return transactions;
}
// ResultSet → Transaction 매핑
private Transaction mapRowToTransaction(ResultSet rs) throws SQLException {
    int deposId = rs.getInt("depos_id");
    if (rs.wasNull()) deposId = 0; // NULL이면 0 처리

    int withdId = rs.getInt("withd_id");
    if (rs.wasNull()) withdId = 0;

    return Transaction.builder()
            .transactionId(rs.getInt("transaction_id"))
            .transactionType(rs.getString("transaction_type"))
            .amount(rs.getLong("amount"))
            .transactionTime(rs.getTimestamp("transaction_time"))
            // 0이면 null로 변환
            .deposId(deposId == 0 ? null : deposId)
            .withdId(withdId == 0 ? null : withdId)
            .build();
```

```
    }
}
```

UserRepository.java

- `save()` : 신규 사용자 정보를 DB에 저장하여 회원가입을 처리한다 .
- `findByUserIdAndPassword()` : ID와 비밀번호가 일치하는 사용자를 조회하여 로그인 인증을 수행한다.
- `findByUserId()` : 사용자 ID를 기준으로 사용자의 상세 정보를 조회한다.
- `deleteById()` : 사용자 정보를 삭제하며, 연관된 계좌 정보도 삭제한다.
- `findAll()` : 등록된 모든 사용자 목록을 가입일 기준으로 조회한다.

```
package com.bank.repository;

import com.bank.config.DBConnection;
import com.bank.domain.User;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class UserRepository {
    public void save(User user) {

        String sql = "INSERT INTO user (user_id, phone_number, first_name, last_name, address, s
sn, password, signup_date) " +
                "VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, user.getUserId());
            preparedStatement.setString(2, user.getPhoneNumber());
            preparedStatement.setString(3, user.getFirstName());
            preparedStatement.setString(4, user.getLastName());
            preparedStatement.setString(5, user.getAddress());
            preparedStatement.setString(6, user.getSsn());
            preparedStatement.setString(7, user.getPassword());
            preparedStatement.setTimestamp(8, user.getSignUpDate());

            int result = preparedStatement.executeUpdate();

            if (result > 0) {
                System.out.println("User saved successfully: " + user.getUserId());
            } else {
                System.out.println("Failed to save user: " + user.getUserId());
```

```
            }
        } catch (SQLException e) {
            if (e.getErrorCode() == 1062) {
                System.err.println("Error: Duplicate entry for user ID " + user.getUserId());
            } else {
                System.err.println("SQL Error Code: " + e.getErrorCode());
                e.printStackTrace();
            }
        }
    }

    public User findByUserIdAndPassword(String userId, String password) {
        String sql = "SELECT * FROM User WHERE user_id = ? AND password = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, userId);
            preparedStatement.setString(2, password);

            try (ResultSet rs = preparedStatement.executeQuery()) {
                if (rs.next()) {
                    // 일치하는 유저가 있으면 객체로 매핑해서 리턴
                    return User.builder()
                            .userId(rs.getString("user_id"))
                            .password(rs.getString("password"))
                            .firstName(rs.getString("first_name"))
                            .lastName(rs.getString("last_name"))
                            .phoneNumber(rs.getString("phone_number"))
                            .address(rs.getString("address"))
                            .ssn(rs.getString("ssn"))
                            .signUpDate(rs.getTimestamp("signup_date"))
                            .build();
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null; // 해당하는 유저 없음
    }

    public User findByUserId(String userId) {
        String sql = "SELECT * FROM User WHERE user_id = ?";

        try (Connection conn = DBConnection.getConnection();
             PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

            preparedStatement.setString(1, userId);
```

```
        try (ResultSet rs = preparedStatement.executeQuery()) {
            if (rs.next()) {
                // 일치하는 유저가 있으면 객체로 매핑해서 리턴
                return User.builder()
                        .userId(rs.getString("user_id"))
                        .password(rs.getString("password"))
                        .firstName(rs.getString("first_name"))
                        .lastName(rs.getString("last_name"))
                        .phoneNumber(rs.getString("phone_number"))
                        .address(rs.getString("address"))
                        .ssn(rs.getString("ssn"))
                        .signUpDate(rs.getTimestamp("signup_date"))
                        .build();
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null; // 해당하는 유저 없음
}

public void deleteById(String userId) {
    String sql = "DELETE FROM User WHERE user_id = ?";

    try (Connection conn = DBConnection.getConnection();
         PreparedStatement preparedStatement = conn.prepareStatement(sql)) {

        preparedStatement.setString(1, userId);
        int affectedRows = preparedStatement.executeUpdate();

        if (affectedRows > 0) {
            System.out.println("User deleted successfully: " + userId);
        } else {
            System.out.println("Failed to delete user or user not found: " + userId);
        }

    } catch (SQLException e) {
        System.err.println("Error deleting user " + userId + ": " + e.getMessage());
        e.printStackTrace();
    }
}

public List<User> findAll() {
    List<User> users = new ArrayList<>();
    String sql = "SELECT * FROM User ORDER BY signup_date DESC";

    try (Connection conn = DBConnection.getConnection();
```

```
            PreparedStatement preparedStatement = conn.prepareStatement(sql);
            ResultSet rs = preparedStatement.executeQuery()) {

            while (rs.next()) {
                User user = User.builder()
                        .userId(rs.getString("user_id"))
                        .password(rs.getString("password"))
                        .firstName(rs.getString("first_name"))
                        .lastName(rs.getString("last_name"))
                        .phoneNumber(rs.getString("phone_number"))
                        .address(rs.getString("address"))
                        .ssn(rs.getString("ssn"))
                        .signUpDate(rs.getTimestamp("signup_date"))
                        .build();
                users.add(user);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

Service

- 애플리케이션의 비즈니스 로직을 담당한다. Controller 계층의 요청을 전달 받아, 실제 유효성을 검증하고 데이터를 가공한다. Repository에서 전달된 데이터를 활용한다.

AccountService.java

- `createAccount()` : 중복되지 않는 고유한 계좌번호를 생성하고, 초기 상태(ACTIVE, balance = 0)로 계좌를 개설한다.
- `checkIfAccountIdExists()` : 랜덤 생성된 계좌번호가 이미 DB에 존재하는지 확인하여 중복을 방지한다.
- `getMyAccounts()` : 로그인한 사용자가 소유한 모든 계좌 리스트를 반환한다.
- `deposit()` : 계좌 상태(잠금/해지 여부)를 확인한 후, 잔액을 증가시키고 입금 내역을 기록한다.
- `withdraw()` : 본인 확인, 비밀번호 검증, 잔액 및 상태 확인 후 출금을 수행하고 내역을 기록한다.
- `transfer()` : 송금인과 수취인의 계좌 상태, 잔액, 이체 한도, 비밀번호를 모두 검증한 후 이체를 실행한다.
- `changeStatusByUser()` : 사용자가 직접 계좌 상태를 변경(휴면 전환 등)하며, 유효하지 않은 상태 변경을 차단한다.
- `checkPasswordAndHandleStatus()` : 비밀번호 일치 여부를 검사하고, 3회 연속 오류 시 계좌를 자동으로 잠금(LOCKED) 처리한다.
- `getTransactionHistory()` : 본인 확인 및 비밀번호 검증 성공 시, 해당 계좌의 입출금 내역을 조회한다.
- `changeAccountPassword()` : 기존 비밀번호를 검증한 후, 새로운 4자리 비밀번호로 변경한다.
- `changeTransferLimit()` : 본인 확인 및 비밀번호 검증 후, 계좌의 1회 이체 한도 금액을 수정한다.

```
package com.bank.service;

import com.bank.domain.Account;
import com.bank.domain.Transaction;
import com.bank.repository.AccountRepository;
import com.bank.repository.TransactionRepository;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class AccountService {

    private final AccountRepository accountRepository;
    private final TransactionRepository transactionRepository;
    private final Random random = new Random();

    public AccountService(AccountRepository accountRepository, TransactionRepository transa
ctionRepository) {
        this.accountRepository = accountRepository;
        this.transactionRepository = transactionRepository;
    }

    // 계좌 개설
    public void createAccount(String userId, long transferLimit, String password) {
        int newAccountId;
        // 고유한 계좌번호 생성 (중복 체크)
        do {
            newAccountId = 1000 + random.nextInt(9000); // 1000 ~ 9999
        } while (checkIfAccountIdExists(newAccountId));


        // 계좌 객체 기본값
        Account newAccount = Account.builder()
                .accountId(newAccountId)
                .accountStatus("ACTIVE")      // 초기 상태: 정상
                .balance(0L)                  // 초기 잔액: 0원
                .transferLimit(transferLimit) // 이체 한도 (사용자 설정)
                .openingDate(new Timestamp(System.currentTimeMillis()))
                .ownerId(userId)              // FK: 로그인한 사용자 ID
                .accountPassword(password)
                .build();

        // 저장 요청
        accountRepository.save(newAccount);
    }
```

```
    // 계좌번호 존재 확인
    private boolean checkIfAccountIdExists(int accountId) {
        return accountRepository.findById(accountId) != null;
    }

    // 내 계좌 목록 가져오기
    public List<Account> getMyAccounts(String userId) {
        return accountRepository.findAllByOwnerId(userId);
    }
    // 입금
    public void deposit(int accountId, long amount) {
        // 금액 체크
        if (amount <= 0) {
            System.out.println("[오류] 입금액은 0원보다 커야 합니다.");
            return;
        }

        // 계좌 정보 조회 (상태 확인)
        Account account = accountRepository.findById(accountId);

        // 계좌 존재 여부
        if (account == null) {
            System.out.println("[오류] 존재하지 않는 계좌번호입니다.");
            return;
        }

        // 계좌 상태에 따른 입금 제한
        String status = account.getAccountStatus();

        if (status.equals("LOCKED")) {
            System.out.println("[거부] 잠금(LOCKED)된 계좌입니다. 입금이 불가능합니다.");
            return;
        }
        if (status.equals("DORMANT")) {
            System.out.println("[거부] 휴면(DORMANT) 계좌입니다. 상태를 '정상'으로 변경 후 이용하세
요.");
            return;
        }
        if (status.equals("CLOSED")) {
            System.out.println("[거부] 해지(CLOSED)된 계좌입니다. 입금이 불가능합니다.");
            return;
        }

        // 모든 검증 통과 후 입금
        accountRepository.updateBalance(accountId, amount);

        // 거래 내역
```

```
        Transaction transaction = Transaction.builder()
                .transactionType("DEPOSIT")
                .amount(amount)
                .transactionTime(new java.sql.Timestamp(System.currentTimeMillis()))
                .deposId(accountId)
                .withdId(null)
                .build();

        transactionRepository.save(transaction);

        System.out.println("[성공] " + amount + "원이 입금되었습니다.");
    }
    // 출금
    public void withdraw(String currentUserId, int accountId, long amount, String password) {
        // 계좌 존재 여부 및 정보
        Account account = accountRepository.findById(accountId);
        if (account == null) {
            System.out.println("[오류] 존재하지 않는 계좌입니다.");
            return;
        }
        // 본인 계좌 확인
        if (!account.getOwnerId().equals(currentUserId)) {
            System.out.println("[거부] 본인의 계좌에서만 출금할 수 있습니다.");
            return;
        }

        // 비밀번호 검증
        if (!checkPasswordAndHandleStatus(account, password)) {
            return; // 검증 실패 시 중단
        }

        // 거래 가능 상태인지 확인
        String status = account.getAccountStatus();
        if (status.equals("DORMANT")) {
            System.out.println("[거부] 휴면(DORMANT) 계좌입니다. 상태를 '정상'으로 변경 후 이용하세
요.");
            return;
        }
        if (status.equals("CLOSED")) {
            System.out.println("[거부] 해지(CLOSED)된 계좌입니다.");
            return;
        }

        // 잔액 검증
        if (account.getBalance() < amount) {
            System.out.println("[오류] 잔액이 부족합니다. (현재 잔액: " + account.getBalance() + "원)");
            return;
        }
```

```
        // 출금 실행
        accountRepository.updateBalance(accountId, -amount);

        // 거래 내역 기록
        Transaction transaction = Transaction.builder()
                .transactionType("WITHDRAW")
                .amount(amount)
                .transactionTime(new Timestamp(System.currentTimeMillis()))
                .deposId(null)       // 입금 계좌 없음
                .withdId(accountId)  // 출금 계좌 기록
                .build();

        transactionRepository.save(transaction);

        System.out.println("[성공] " + amount + "원이 출금되었습니다.");
    }
    // 이체
    public void transfer(String currentUserId, int fromAccountId, int toAccountId, long amount, St
ring password) {
        Account fromAccount = accountRepository.findById(fromAccountId);
        // 보내는 사람 계좌 정보 확인
        if (fromAccount == null) {
            System.out.println("[오류] 보내는 계좌가 존재하지 않습니다.");
            return;
        }
        // 자기 자신에게 이체 불가
        if (fromAccountId == toAccountId) {
            System.out.println("[오류] 출금 계좌와 입금 계좌가 같습니다.");
            return;
        }
        // 본인 계좌 확인
        if (!fromAccount.getOwnerId().equals(currentUserId)) {
            System.out.println("[거부] 본인의 계좌에서만 출금할 수 있습니다.");
            return;
        }

        // 금액 체크
        if (amount <= 0) {
            System.out.println("[오류] 이체 금액은 0원보다 커야 합니다.");
            return;
        }

        // 비밀번호 검증
        if (!checkPasswordAndHandleStatus(fromAccount, password)) {
            return; // 검증 실패 시 중단
        }
```

```
        // 거래 가능 상태인지 확인
        String status = fromAccount.getAccountStatus();
        if (status.equals("DORMANT")) {
            System.out.println("[거부] 휴면(DORMANT) 계좌입니다. 상태를 '정상'으로 변경 후 이용하세
요.");
            return;
        }
        if (status.equals("CLOSED")) {
            System.out.println("[거부] 해지(CLOSED)된 계좌입니다.");
            return;
        }

        // 잔액 검증
        if (fromAccount.getBalance() < amount) {
            System.out.println("[오류] 잔액이 부족합니다.");
            return;
        }

        // 이체 한도 검증
        if (amount > fromAccount.getTransferLimit()) {
            System.out.println("[오류] 1회 이체 한도를 초과했습니다. (한도: " + fromAccount.getTransfer
Limit() + "원)");
            return;
        }

        // 받는 사람 계좌 존재 확인
        Account toAccount = accountRepository.findById(toAccountId);
        if (toAccount == null) {
            System.out.println("[오류] 받는 사람의 계좌번호가 존재하지 않습니다.");
            return;
        }

        // 받는 사람 계좌 상태 확인
        String toStatus = toAccount.getAccountStatus();
        if (toStatus.equals("LOCKED")) {
            System.out.println("[거부] 상대방 계좌가 잠금(LOCKED) 상태라 이체할 수 없습니다.");
            return;
        }
        if (toStatus.equals("DORMANT")) {
            System.out.println("[거부] 상대방 계좌가 휴면(DORMANT) 상태라 이체할 수 없습니다.");
            return;
        }
        if (toStatus.equals("CLOSED")) {
            System.out.println("[거부] 상대방 계좌는 해지(CLOSED)된 계좌입니다.");
            return;
        }

        // 모든 검증 통과 - 트랜잭션 실행
```

```
        boolean success = accountRepository.transfer(fromAccountId, toAccountId, amount);

        if (success) {
            System.out.println("[성공] " + amount + "원을 이체했습니다.");
            System.out.println("받는 분: " + toAccount.getOwnerId() + " (계좌: " + toAccountId + ")");
        } else {
            System.out.println("[실패] 시스템 오류로 이체가 취소되었습니다.");
        }
    }
    // 유저가 직접 상태 변경
    public void changeStatusByUser(String currentUserId, int accountId, String newStatus, Strin
g password) {
        // 계좌 확인
        Account account = accountRepository.findById(accountId);
        if (account == null) {
            System.out.println("[오류] 존재하지 않는 계좌입니다.");
            return;
        }
        // 본인 계좌 확인
        if (!account.getOwnerId().equals(currentUserId)) {
            System.out.println("[거부] 본인 소유의 계좌가 아닙니다.");
            return;
        }

        // 비밀번호 검증
        if (!checkPasswordAndHandleStatus(account, password)) {
            return;
        }

        // 현재 상태 체크
        String currentStatus = account.getAccountStatus();

        if (currentStatus.equals("LOCKED")) {
            System.out.println("[거부] 해당 계좌는 잠금 상태입니다. 관리자에게 문의하여 해제하세요.");
            return;
        }

        if (currentStatus.equals("CLOSED")) {
            System.out.println("[거부] 이미 해지된 계좌는 상태를 변경할 수 없습니다.");
            return;
        }

        // 입력된 상태값 검증
        // 유저는 LOCKED 불가
        if (!newStatus.equals("ACTIVE") && !newStatus.equals("DORMANT") && !newStatus.equals
("CLOSED")) {
            System.out.println("[오류] 변경 가능한 상태가 아닙니다. (ACTIVE, DORMANT, CLOSED 만 가
능)");
```

```
            return;
        }

        // 상태 업데이트
        accountRepository.updateAccountStatus(accountId, newStatus);
        //System.out.println("[성공] 계좌 상태가 '" + newStatus + "'(으)로 변경되었습니다.");
    }
    // 비밀번호 검증 및 상태 관리 메서드
    // 리턴: true/false
    private boolean checkPasswordAndHandleStatus(Account account, String inputPassword) {
        // 이미 잠긴 계좌인지 먼저 확인
        if (account.getAccountStatus().equals("LOCKED")) {
            System.out.println("[거부] 비밀번호 3회 오류로 잠긴 계좌입니다. 관리자에게 문의하세요.");
            return false;
        }

        // 비밀번호 일치 확인
        if (account.getAccountPassword().equals(inputPassword)) {
            // 맞았으면 카운트 초기화
            accountRepository.resetWrongPwCount(account.getAccountId());
            return true;
        } else {
            // 틀렸으면 카운트 증가
            accountRepository.increaseWrongPwCount(account.getAccountId());

            // +1
            int newCount = account.getWrongPwCount() + 1;
            System.out.println("[오류] 비밀번호가 일치하지 않습니다. (오류 횟수: " + newCount + "/3)");

            // 3회 도달 시 잠금 처리
            if (newCount >= 3) {
                accountRepository.updateAccountStatus(account.getAccountId(), "LOCKED");
                System.out.println("!!! [경고] 비밀번호 3회 연속 오류로 계좌가 잠금(LOCKED) 처리되었습니
다. !!!");
            }
            return false;
        }
    }
    // 내 계좌 거래내역
    public List<Transaction> getTransactionHistory(String currentUserId, int accountId, String pa
ssword) {
        // 계좌 확인
        Account account = accountRepository.findById(accountId);
        if (account == null) {
            System.out.println("[오류] 존재하지 않는 계좌입니다.");
            return new ArrayList<>();
        }
```

```
        // 본인 확인
        if (!account.getOwnerId().equals(currentUserId)) {
            System.out.println("[거부] 본인의 계좌 내역만 조회할 수 있습니다.");
            return new ArrayList<>();
        }

        // 비밀번호 확인
        if (!checkPasswordAndHandleStatus(account, password)) {
            return new ArrayList<>();
        }

        // 조회 실행
        return transactionRepository.findAllByAccountId(accountId);
    }
    // 계좌 비밀번호 변경
    public void changeAccountPassword(String currentUserId, int accountId, String oldPasswor
d, String newPassword) {
        Account account = accountRepository.findById(accountId);
        if (account == null) { System.out.println("[오류] 계좌 없음"); return; }
        if (!account.getOwnerId().equals(currentUserId)) { System.out.println("[거부] 본인 계좌 아
님"); return; }

        // 기존 비밀번호 검증, 틀리면 카운트 증가
        if (!checkPasswordAndHandleStatus(account, oldPassword)) {
            return;
        }

        // 새 비밀번호 유효성 검사
        if (newPassword.length() != 4) {
            System.out.println("[오류] 비밀번호는 4자리 숫자여야 합니다.");
            return;
        }

        accountRepository.updateAccountPassword(accountId, newPassword);
        System.out.println("[성공] 계좌 비밀번호가 변경되었습니다.");
    }

    // 이체 한도 변경
    public void changeTransferLimit(String currentUserId, int accountId, long newLimit, String pa
ssword) {
        Account account = accountRepository.findById(accountId);
        if (account == null) { System.out.println("[오류] 계좌 없음"); return; }
        if (!account.getOwnerId().equals(currentUserId)) { System.out.println("[거부] 본인 계좌 아
님"); return; }

        // 비밀번호 검증
        if (!checkPasswordAndHandleStatus(account, password)) {
            return;
```

```
        }

        // 한도 값 검증
        if (newLimit < 0) {
            System.out.println("[오류] 한도는 0원 이상이어야 합니다.");
            return;
        }

        accountRepository.updateTransferLimit(accountId, newLimit);
        System.out.println("[성공] 이체 한도가 " + newLimit + "원으로 변경되었습니다.");
    }
}
```

AdminService.java

- `login()` : 관리자 ID와 비밀번호를 검증하고, 로그인 시도 결과를 접속 로그에 기록한다.
- `deleteUser()` : 특정 사용자를 삭제하며, 연동된 모든 계좌 데이터가 함께 삭제되도록 처리한다.
- `viewAllAccessLogs()` : 시스템에 기록된 모든 사용자 및 관리자의 접속 로그를 조회한다.
- `changeAccountStatus()` : 관리자 권한으로 계좌 상태를 변경하며, 'ACTIVE' 전환 시 비밀번호 오류 횟수를 초기화한다.
- `getAllAccounts()` : 은행 시스템에 등록된 전체 계좌 목록을 조회한다.
- `getAllUsers()` : 은행 시스템에 가입된 전체 사용자 목록을 조회한다.
- `getAllTransactions()` : 시스템 내에서 발생한 모든 거래 내역을 조회한다.

```
package com.bank.service;

import com.bank.domain.*;
import com.bank.repository.*;

import java.sql.Timestamp;
import java.util.List;

public class AdminService {

    private final AdminRepository adminRepository;
    private final AccountRepository accountRepository;
    private final AccessLogRepository accessLogRepository;
    private final TransactionRepository transactionRepository;
    private final UserRepository userRepository;

    public AdminService
        (AdminRepository adminRepository, AccountRepository accountRepository, AccessLog
Repository accessLogRepository, TransactionRepository transactionRepository, UserRepository
userRepository) { // [수정]
        this.adminRepository = adminRepository;
        this.accountRepository = accountRepository;
```

```
        this.accessLogRepository = accessLogRepository;
        this.transactionRepository = transactionRepository;
        this.userRepository = userRepository;
    }

    // 관리자 로그인
    public Administrator login(String adminId, String password) {
        // ID로 관리자가 존재 확인
        Administrator existingAdmin = adminRepository.findByAdminId(adminId);

        // 관리자가 존재하지 않으면, 로그를 남기지 않고 로그인 실패 처리
        if (existingAdmin == null) {
            return null;
        }

        // 관리자가 존재하면, 로그를 남김
        Administrator admin = adminRepository.findByAdminIdAndPassword(adminId, password);
        boolean isSuccess = (admin != null);

        AccessLog log = AccessLog.builder()
                .accessDate(new Timestamp(System.currentTimeMillis()))
                .successOrNot(isSuccess)
                .userId(null)       // 유저는 NULL
                .adminId(adminId)   // 존재하는 관리자 ID
                .build();

        accessLogRepository.save(log);

        return admin;
    }

    // 관리자 권한 유저 삭제
    public void deleteUser(String userId) {
        // 유저 존재 여부 확인
        User userToDelete = userRepository.findByUserId(userId);
        if (userToDelete == null) {
            System.out.println("[오류] 삭제하려는 유저 (" + userId + ")를 찾을 수 없습니다.");
            return;
        }

        // 유저 삭제 (DB ON DELETE CASCADE)
        try {
            userRepository.deleteById(userId);
            System.out.println("[성공] 유저 (" + userId + ")가 삭제되었습니다. 관련 계좌도 함께 삭제되었습
니다.");
        } catch (Exception e) {
            System.err.println("[오류] 유저 (" + userId + ") 삭제 중 오류 발생: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }

    // 전체 로그 조회
    public void viewAllAccessLogs() {
        accessLogRepository.printAllLogs();
    }

    // 계좌 상태 변경
    public void changeAccountStatus(int accountId, String status) {
        // 유효한 상태값 확인
        if (!status.equals("ACTIVE") && !status.equals("LOCKED") && !status.equals("DORMANT")
&& !status.equals("CLOSED")) {
            System.out.println("[오류] 유효하지 않은 상태입니다. (ACTIVE, LOCKED, DORMANT, CLOSE
D 중 선택)");
            return;
        }
        accountRepository.updateAccountStatus(accountId, status);
        if (status.equals("ACTIVE")) {
            accountRepository.resetWrongPwCount(accountId);
            System.out.println("  ㄴ 계좌의 비밀번호 오류 횟수가 0으로 초기화되었습니다.");
        }
    }
    // 모든 계좌 목록
    public List<Account> getAllAccounts() {
        return accountRepository.findAll();
    }


    // 전체 유저 목록
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    // 전체 거래 내역
    public List<Transaction> getAllTransactions() {
        return transactionRepository.findAll();
    }
}
```

UserService.java

- `signUp()` : ID 중복 및 필수 입력값 존재 여부를 검증한 후, 신규 사용자를 등록한다.
- `validateUser()` : 회원가입 시 입력된 정보의 유효성을 검사한다.
- `login()` : 사용자 ID와 비밀번호를 검증하고, 로그인 성공/실패 여부를 접속 로그에 기록한다.

```
package com.bank.service;
```

```
import com.bank.domain.AccessLog;
import com.bank.domain.User;
import com.bank.repository.AccessLogRepository;
import com.bank.repository.UserRepository;

import java.sql.Timestamp;

public class UserService {

    private final UserRepository userRepository;
    private final AccessLogRepository accessLogRepository;

    public UserService(UserRepository userRepository, AccessLogRepository accessLogReposit
ory) {
        this.userRepository = userRepository;
        this.accessLogRepository = accessLogRepository;
    }

    // 회원가입 처리
    public void signUp(User user) {
        // 유효성 검사
        validateUser(user);
        userRepository.save(user);
    }

    private void validateUser(User user) {
        // ID 중복 검사
        if (userRepository.findByUserId(user.getUserId()) != null) {
            throw new IllegalArgumentException("[오류] 이미 존재하는 아이디입니다: " + user.getUserId
());
        }

        // 필수 값 검사
        if (user.getUserId() == null || user.getUserId().trim().isEmpty()) {
            throw new IllegalArgumentException("[오류] 아이디는 필수 입력 항목입니다.");
        }
        if (user.getPassword() == null || user.getPassword().trim().isEmpty()) {
            throw new IllegalArgumentException("[오류] 비밀번호는 필수 입력 항목입니다.");
        }
        if (user.getFirstName() == null || user.getFirstName().trim().isEmpty() ||
            user.getLastName() == null || user.getLastName().trim().isEmpty()) {
            throw new IllegalArgumentException("[오류] 이름은 필수 입력 항목입니다.");
        }
        if (user.getSsn() == null || user.getSsn().trim().isEmpty()) {
            throw new IllegalArgumentException("[오류] 주민등록번호는 필수 입력 항목입니다.");
        }
        if (user.getPhoneNumber() == null || user.getPhoneNumber().trim().isEmpty()) {
            throw new IllegalArgumentException("[오류] 전화번호는 필수 입력 항목입니다.");
```

```
        }
        if (user.getAddress() == null || user.getAddress().trim().isEmpty()) {
            throw new IllegalArgumentException("[오류] 주소는 필수 입력 항목입니다.");
        }
    }

    public User login(String userId, String password) {
        // ID로 사용자가 확인
        User existingUser = userRepository.findByUserId(userId);

        // 존재하지 않으면, 로그 남기지 않고 로그인 실패 처리
        if (existingUser == null) {
            return null;
        }

        // 존재하면, 로그를 남김
        User user = userRepository.findByUserIdAndPassword(userId, password);
        boolean isSuccess = (user != null);

        AccessLog log = AccessLog.builder()
                .accessDate(new Timestamp(System.currentTimeMillis()))
                .successOrNot(isSuccess)
                .userId(userId)
                .adminId(null)
                .build();

        accessLogRepository.save(log);

        return user; //user or null
    }
}
```

Controller

- 사용자와 시스템 사이의 상호작용 인터페이스를 담당한다. 사용자의 입력에 따른 비즈니스 로직을 호출하고, 결과를 출력한다.

AccountController.java

- `createAccountMenu()` : 사용자로부터 비밀번호와 이체 한도를 입력받아 계좌 개설 요청을 처리한다.
- `showMyAccounts()` : 로그인한 사용자의 모든 계좌 목록을 받아와 출력한다.
- `showDepositMenu()` : 입금할 계좌번호와 금액을 입력받고, 입금 서비스를 호출한다.
- `showWithdrawMenu()` : 출금 계좌번호, 금액, 비밀번호를 입력받아 출금 요청을 전달한다.
- `showTransferMenu()` : 출금 계좌, 입금 계좌, 금액, 비밀번호를 입력받아 계좌 이체 로직을 호출한다.
- `showChangeStatusMenu()` : 사용자가 자신의 계좌 상태를 직접 변경할 수 있도록 메뉴를 제공하고 입력을 처리한다.

- `showTransactionHistoryMenu()` : 계좌번호와 비밀번호를 입력받아 인증 후, 해당 계좌의 거래 내역을 조회하여 출력한다.
- `showChangePasswordMenu()` : 기존 비밀번호와 새 비밀번호를 입력받아 계좌 비밀번호 변경 요청을 처리한다.
- `showChangeLimitMenu()` : 계좌 비밀번호와 새로운 한도 금액을 입력받아 이체 한도 변경을 처리한다.

```
package com.bank.controller;

import com.bank.domain.Account;
import com.bank.domain.Transaction;
import com.bank.domain.User;
import com.bank.service.AccountService;

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

public class AccountController {

    private final AccountService accountService;
    private final Scanner scanner;

    public AccountController(AccountService accountService, Scanner scanner) {
        this.accountService = accountService;
        this.scanner = scanner;
    }

    public void createAccountMenu(User loggedInUser) {
        System.out.println("\n--- [계좌 개설] ---");
        System.out.println("고객님(" + loggedInUser.getFirstName() + ") 명의의 새 계좌를 개설합니다.");

        System.out.print("사용하실 계좌 비밀번호(4자리)를 입력하세요: ");
        String password = scanner.nextLine();

        if (password.length() != 4) {
            System.out.println("[오류] 비밀번호는 4자리 숫자로 설정해야 합니다.");
            return;
        }


        System.out.print("설정할 1회 이체 한도를 입력하세요 (예: 1000000): ");
        try {
            long limit = Long.parseLong(scanner.nextLine());

            // 로그인한 유저의 ID 넘김
            accountService.createAccount(loggedInUser.getUserId(), limit, password);

        } catch (NumberFormatException e) {
```

```
            System.out.println("숫자만 입력해주세요.");
        }
    }

    // 내 계좌 목록
    public void showMyAccounts(User loggedInUser) {
        System.out.println("\n--- [내 계좌 목록] ---");

        List<Account> myAccounts = accountService.getMyAccounts(loggedInUser.getUserId());

        if (myAccounts.isEmpty()) {
            System.out.println("보유하신 계좌가 없습니다.");
        } else {
            // 헤더
            System.out.printf("%-15s %-10s %-15s %-15s\n", "계좌번호", "상태", "잔액", "이체한도");
            System.out.println("-------------------------------------------------------------");

            for (Account acc : myAccounts) {
                System.out.printf("%-15d %-10s %-15d %-15d\n",
                        acc.getAccountId(),
                        acc.getAccountStatus(),
                        acc.getBalance(),
                        acc.getTransferLimit());
            }
            System.out.println("-------------------------------------------------------------");
        }
    }
    // 입금 메뉴
    public void showDepositMenu(User loggedInUser) {
        System.out.println("\n--- [입금] ---");

        // 계좌 목록
        showMyAccounts(loggedInUser);

        System.out.print("입금할 계좌번호 입력: ");
        try {
            int accountId = Integer.parseInt(scanner.nextLine());

            System.out.print("입금액 입력: ");
            long amount = Long.parseLong(scanner.nextLine());

            accountService.deposit(accountId, amount);

        } catch (NumberFormatException e) {
            System.out.println("[오류] 숫자로 입력해주세요.");
        }
    }
    // 출금 메뉴
```

```
    public void showWithdrawMenu(User loggedInUser) {
        System.out.println("\n--- [출금] ---");
        showMyAccounts(loggedInUser);

        try {
            System.out.print("출금할 계좌번호 입력: ");
            int accountId = Integer.parseInt(scanner.nextLine());

            System.out.print("출금액 입력: ");
            long amount = Long.parseLong(scanner.nextLine());

            System.out.print("계좌 비밀번호(4자리) 입력: ");
            String password = scanner.nextLine();

            accountService.withdraw(loggedInUser.getUserId() ,accountId, amount, password);

        } catch (NumberFormatException e) {
            System.out.println("[오류] 숫자로 입력해주세요.");
        }
    }
    // 이체 메뉴
    public void showTransferMenu(User loggedInUser) {
        System.out.println("\n--- [계좌 이체] ---");
        showMyAccounts(loggedInUser); // 내 계좌 목록

        try {
            System.out.print("내 출금 계좌번호: ");
            int fromId = Integer.parseInt(scanner.nextLine());

            System.out.print("계좌 비밀번호(4자리): ");
            String password = scanner.nextLine();

            System.out.print("받을 계좌번호(상대방): ");
            int toId = Integer.parseInt(scanner.nextLine());

            System.out.print("이체할 금액: ");
            long amount = Long.parseLong(scanner.nextLine());

            accountService.transfer(loggedInUser.getUserId(),fromId, toId, amount, password);

        } catch (NumberFormatException e) {
            System.out.println("[오류] 숫자로 입력해주세요.");
        }
    }

    // 상태 변경 메뉴
    public void showChangeStatusMenu(User loggedInUser) {
        System.out.println("\n--- [계좌 상태 변경] ---");
```

```
        showMyAccounts(loggedInUser); // 목록 보여주기

        try {
            System.out.print("상태를 변경할 계좌번호: ");
            int accId = Integer.parseInt(scanner.nextLine());

            System.out.print("계좌 비밀번호: ");
            String password = scanner.nextLine();

            System.out.println("변경할 상태를 선택하세요:");
            System.out.println("1. 정상 (ACTIVE) - 입출금 가능");
            System.out.println("2. 휴면 (DORMANT) - 거래 제한");
            System.out.println("3. 해지 (CLOSED) - [주의] 복구 불가");
            System.out.print("선택 >> ");
            String choice = scanner.nextLine();

            String newStatus = "";
            switch (choice) {
                case "1": newStatus = "ACTIVE"; break;
                case "2": newStatus = "DORMANT"; break;
                case "3":
                    // 해지 시 재확인
                    System.out.print("정말로 해지하시겠습니까? 해지 후에는 복구가 불가능합니다. (Y/N): ");
                    String confirm = scanner.nextLine();
                    if (!confirm.equalsIgnoreCase("Y")) {
                        System.out.println("취소되었습니다.");
                        return;
                    }
                    newStatus = "CLOSED";
                    break;
                default:
                    System.out.println("잘못된 선택입니다.");
                    return;
            }

            accountService.changeStatusByUser(loggedInUser.getUserId(), accId, newStatus, pass
word);

        } catch (NumberFormatException e) {
            System.out.println("[오류] 숫자로 입력해주세요.");
        }
    }
    // 거래 내역 조회 메뉴
    public void showTransactionHistoryMenu(User loggedInUser) {
        System.out.println("\n--- [거래 내역 조회] ---");
        showMyAccounts(loggedInUser);

        try {
```

```
            System.out.print("조회할 계좌번호: ");
            int accId = Integer.parseInt(scanner.nextLine());
            System.out.print("계좌 비밀번호: ");
            String pw = scanner.nextLine();

            List<Transaction> history = accountService.getTransactionHistory(loggedInUser.getUse
rId(), accId, pw);

            if (history.isEmpty()) {
                System.out.println("거래 내역이 없거나 조회에 실패했습니다.");
            } else {
                System.out.println("\n[거래 내역]");
                System.out.printf("%-20s %-10s %-15s %-10s %-10s\n", "시간", "유형", "금액", "보낸
분", "받은분");
                System.out.println("----------------------------------------------------------------
----");
                for (Transaction t : history) {
                    System.out.printf("%-20s %-10s %-15d %-10s %-10s\n",
                            t.getTransactionTime().toString().substring(0, 19),
                            t.getTransactionType(),
                            t.getAmount(),
                            t.getWithdId() == null ? "-" : t.getWithdId().toString(), // 출금계좌
                            t.getDeposId() == null ? "-" : t.getDeposId().toString()  // 입금계좌
                    );
                }
                System.out.println("----------------------------------------------------------------
----");
            }

        } catch (NumberFormatException e) {
            System.out.println("숫자만 입력하세요.");
        }
    }

    // 비밀번호 변경 메뉴
    public void showChangePasswordMenu(User loggedInUser) {
        System.out.println("\n--- [계좌 비밀번호 변경] ---");
        showMyAccounts(loggedInUser);

        try {
            System.out.print("대상 계좌번호: ");
            int accId = Integer.parseInt(scanner.nextLine());

            System.out.print("현재 비밀번호: ");
            String oldPw = scanner.nextLine();

            System.out.print("새로운 비밀번호(4자리): ");
            String newPw = scanner.nextLine();
```

```
            accountService.changeAccountPassword(loggedInUser.getUserId(), accId, oldPw, new
Pw);

        } catch (NumberFormatException e) {
            System.out.println("숫자만 입력하세요.");
        }
    }

    // 이체 한도 변경 메뉴
    public void showChangeLimitMenu(User loggedInUser) {
        System.out.println("\n--- [이체 한도 변경] ---");
        showMyAccounts(loggedInUser);

        try {
            System.out.print("대상 계좌번호: ");
            int accId = Integer.parseInt(scanner.nextLine());

            System.out.print("계좌 비밀번호: ");
            String pw = scanner.nextLine();

            System.out.print("변경할 1회 한도 금액: ");
            long newLimit = Long.parseLong(scanner.nextLine());

            accountService.changeTransferLimit(loggedInUser.getUserId(), accId, newLimit, pw);

        } catch (NumberFormatException e) {
            System.out.println("숫자만 입력하세요.");
        }
    }
}
```

AdminController.java

- `showAdminMenu()` : 관리자 전용 메인 화면을 출력하고, 선택된 메뉴에 따라 기능을 분기 처리한다.
- `showDeleteUserMenu()` : 관리자가 입력한 ID에 해당하는 회원을 삭제시킨다.
- `showAllAccountsWithStatus()` : 모든 계좌의 상세 정보를 출력한다.
- `showAllTransactions()` : 은행 시스템에서 발생한 모든 입출금 및 이체 내역을 조회하여 출력한다.

```
package com.bank.controller;

import com.bank.domain.Account;
import com.bank.domain.Administrator;
import com.bank.domain.Transaction;
import com.bank.domain.User;
import com.bank.service.AdminService;
```

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

public class AdminController {

    private final AdminService adminService;
    private final Scanner scanner;

    public AdminController(AdminService adminService, Scanner scanner) {
        this.adminService = adminService;
        this.scanner = scanner;
    }

    // 관리자 메인 메뉴
    public void showAdminMenu(Administrator admin) {
        while (true) {
            System.out.println("\n--- [관리자 모드] " + admin.getFirstName() + " ---");
            System.out.println("1. 전체 접속 로그 조회 (Access Logs)");
            System.out.println("2. 계좌 상태 변경 (Manage Account Status)");
            System.out.println("3. 전체 거래 내역 조회 (Audit Transactions)");
            System.out.println("4. 유저 삭제 (Delete User)");
            System.out.println("9. 로그아웃 (Logout)");
            System.out.print("선택 >> ");

            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    adminService.viewAllAccessLogs();
                    break;
                case "2":
                    // 상태 변경 전 전체 목록 출력
                    showAllAccountsWithStatus();

                    System.out.println("\n[관리자 권한] 계좌 상태를 변경합니다.");
                    System.out.print("대상 계좌번호: ");
                    try {
                        int accId = Integer.parseInt(scanner.nextLine());
                        System.out.print("변경할 상태 (ACTIVE / LOCKED / DORMANT / CLOSED): ");
                        String status = scanner.nextLine().toUpperCase();

                        adminService.changeAccountStatus(accId, status);

                    } catch (NumberFormatException e) {
                        System.out.println("[오류] 숫자를 입력해주세요.");
                    }
                    break;
                case "3":
```

```
                // 전체 거래 내역 조회
                showAllTransactions();
                break;
            case "4": // 유저 삭제 메뉴 호출
                showDeleteUserMenu();
                break;
            case "9":
                System.out.println("관리자 로그아웃.");
                return;
            default:
                System.out.println("잘못된 입력입니다.");
        }
    }
}
// 유저 삭제 메뉴
private void showDeleteUserMenu() {
    // 전체 유저 목록
    List<User> users = adminService.getAllUsers();

    System.out.println("\n--- [전체 유저 목록] ---");
    System.out.printf("%-15s %-15s %-15s %-15s\n", "User ID", "이름", "전화번호", "가입일");
    System.out.println("-----------------------------------------------------------------");
    for (User user : users) {
        System.out.printf("%-15s %-15s %-15s %-15s\n",
                user.getUserId(),
                user.getLastName() + user.getFirstName(),
                user.getPhoneNumber(),
                user.getSignUpDate().toString().substring(0, 10));
    }
    System.out.println("-----------------------------------------------------------------");

    // 삭제할 유저 ID
    System.out.print("삭제할 유저의 ID를 입력하세요: ");
    String userIdToDelete = scanner.nextLine();

    // 삭제 요청
    adminService.deleteUser(userIdToDelete);
}

// 전체 계좌 출력 메서드
private void showAllAccountsWithStatus() {
    List<Account> accounts = adminService.getAllAccounts();

    System.out.println("\n--- [전체 계좌 현황] ---");
    System.out.printf("%-10s %-10s %-10s %-15s %-10s %-10s\n",
            "계좌번호", "소유자", "상태", "잔액", "오류횟수", "개설일");
    System.out.println("-----------------------------------------------------------------------
--");
```

```
        for (Account acc : accounts) {
            System.out.printf("%-10d %-10s %-10s %-15d %-10d %-10s\n",
                    acc.getAccountId(),
                    acc.getOwnerId(),
                    acc.getAccountStatus(),
                    acc.getBalance(),
                    acc.getWrongPwCount(),
                    acc.getOpeningDate().toString().substring(0, 10));
        }
        System.out.println("-------------------------------------------------------------------------
--");
    }

    // 전체 거래 내역
    private void showAllTransactions() {
        List<Transaction> transactions = adminService.getAllTransactions();
        System.out.println("\n--- [전체 거래 내역] ---");
        if (transactions.isEmpty()) {
            System.out.println("거래 내역이 없거나 조회에 실패했습니다.");
        } else {
            System.out.println("\n[거래 내역]");
            System.out.printf("%-20s %-10s %-15s %-10s %-10s\n", "시간", "유형", "금액", "보낸분",
"받은분");
            System.out.println("-------------------------------------------------------------------------
-");
            for (Transaction t : transactions) {
                System.out.printf("%-20s %-10s %-15d %-10s %-10s\n",
                        t.getTransactionTime().toString().substring(0, 19),
                        t.getTransactionType(),
                        t.getAmount(),
                        t.getWithdId() == null ? "-" : t.getWithdId().toString(),
                        t.getDeposId() == null ? "-" : t.getDeposId().toString()
                );
            }
            System.out.println("-------------------------------------------------------------------------
-");
        }
    }
}
```

UserController.java

- `showSignUpMenu()` : 사용자로부터 개인정보를 입력 받아 객체를 생성하고 회원가입 서비스를 호출한다.

```
package com.bank.controller;

import com.bank.domain.User;
import com.bank.service.UserService;
```

```
import java.sql.Timestamp;
import java.util.Scanner;

public class UserController {

    private final UserService userService;
    private final Scanner scanner;

    public UserController(UserService userService, Scanner scanner) {
        this.userService = userService;
        this.scanner = scanner;
    }

    public void showSignUpMenu() {
        System.out.println("\n--- [회원가입] 정보를 입력해주세요. ---");

        System.out.print("아이디(ID): ");
        String userId = scanner.nextLine();

        System.out.print("비밀번호: ");
        String password = scanner.nextLine();

        System.out.print("이름(First Name): ");
        String firstName = scanner.nextLine();

        System.out.print("성(Last Name): ");
        String lastName = scanner.nextLine();

        System.out.print("연락처: ");
        String phoneNumber = scanner.nextLine();

        System.out.print("주소: ");
        String address = scanner.nextLine();

        System.out.print("주민번호(SSN): ");
        String ssn = scanner.nextLine();

        // User 객체 생성
        User newUser = User.builder()
                .userId(userId)
                .password(password)
                .firstName(firstName)
                .lastName(lastName)
                .phoneNumber(phoneNumber)
                .address(address)
                .ssn(ssn)
                .signUpDate(new Timestamp(System.currentTimeMillis())) // 가입일은 현재 시간
```

```
            .build();

        try {
            userService.signUp(newUser);
            System.out.println("[성공] 회원가입이 완료되었습니다. 메인 메뉴로 이동합니다.");
        } catch (IllegalArgumentException e) {
            // 서비스에서 발생한 유효성 검사 오류를 잡아서 메시지 출력
            System.out.println(e.getMessage());
        }
    }

}
```

Main

- 각 Controller에 의존성을 주입하고, Main화면 로직을 수행한다.
- `showCustomerMenu()` : 유저 전용 메인 화면을 출력하고, 선택된 메뉴에 따라 기능을 분기 처리한다.

```
package com.bank;

import com.bank.config.DBConnection;
import com.bank.controller.AccountController;
import com.bank.controller.AdminController;
import com.bank.controller.UserController;
import com.bank.domain.Administrator;
import com.bank.domain.User;
import com.bank.repository.*;
import com.bank.service.AccountService;
import com.bank.service.AdminService;
import com.bank.service.UserService;

import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try { // UTF-8 출력 설정
            System.setOut(new PrintStream(System.out, true, StandardCharsets.UTF_8.name()));
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        UserRepository userRepository = new UserRepository();
        AccessLogRepository accessLogRepository = new AccessLogRepository();
        AccountRepository accountRepository = new AccountRepository();
        TransactionRepository transactionRepository = new TransactionRepository();
```

```
        AdminRepository adminRepository = new AdminRepository();

        UserService userService = new UserService(userRepository, accessLogRepository);
        AccountService accountService = new AccountService(accountRepository, transactionRe
pository);
        AdminService adminService = new AdminService(adminRepository, accountRepository, ac
cessLogRepository, transactionRepository, userRepository);

        Scanner scanner = new Scanner(System.in);
        UserController userController = new UserController(userService, scanner);
        AccountController accountController = new AccountController(accountService, scanner);
        AdminController adminController = new AdminController(adminService, scanner);

        while (true) {
            System.out.println("1. 로그인 (Login)");
            System.out.println("2. 회원가입 (Register)");
            System.out.println("0. 종료 (Exit)");
            System.out.print("선택 >> ");

            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    System.out.println("\n--- [통합 로그인] ---");
                    System.out.print("ID: ");
                    String id = scanner.nextLine();
                    System.out.print("Password: ");
                    String pw = scanner.nextLine();

                    // 1. 일반 유저 로그인 시도
                    User user = userService.login(id, pw);
                    if (user != null) {
                        System.out.println(">> 사용자 로그인 성공");
                        showCustomerMenu(scanner, user, accountController);
                        break;
                    }

                    // 2. 유저 실패 시 관리자 로그인 시도
                    Administrator admin = adminService.login(id, pw);
                    if (admin != null) {
                        System.out.println(">> 관리자 로그인 성공");
                        // 관리자 메뉴
                        adminController.showAdminMenu(admin);
                        break;
                    }

                    System.out.println("[실패] 아이디 또는 비밀번호를 확인하세요.");
                    break;
```

```
            case "2":
                userController.showSignUpMenu();
                break;
            case "0":
                System.out.println("---프로그램을 종료합니다.---");
                return;
            default:
                System.out.println("잘못된 선택입니다. 다시 시도하세요.");
        }
    }
}

    private static void showCustomerMenu(Scanner scanner, User user, AccountController acco
untController) {
        while (true) {
            System.out.println("\n--- User Menu (" + user.getFirstName() + ") ---");
            System.out.println("1. 계좌 개설 (Open Account)");
            System.out.println("2. 내 계좌 조회 (My Accounts)");
            System.out.println("3. 입금 (Deposit)");
            System.out.println("4. 출금 (Withdraw)");
            System.out.println("5. 이체 (Transfer)");
            System.out.println("6. 계좌 상태 변경 (Change Account Status)");
            System.out.println("7. 거래 내역 조회 (Transaction History)");
            System.out.println("8. 비밀번호 변경 (Change Password)");
            System.out.println("9. 이체 한도 변경 (Change Transfer Limit)");
            System.out.println("0. 로그아웃 (Logout)");
            System.out.print("선택 >> ");
            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    accountController.createAccountMenu(user);
                    break;
                case "2":
                    accountController.showMyAccounts(user);
                    break;
                case "3":
                    accountController.showDepositMenu(user);
                    break;
                case "4":
                    accountController.showWithdrawMenu(user);
                    break;
                case "5":
                    accountController.showTransferMenu(user);
                    break;
                case "6":
                    accountController.showChangeStatusMenu(user);
                    break;
```

```
            case "7":
                accountController.showTransactionHistoryMenu(user);
                break;
            case "8":
                accountController.showChangePasswordMenu(user);
                break;
            case "9":
                accountController.showChangeLimitMenu(user);
                break;
            case "0": // 로그아웃
                System.out.println("로그아웃 되었습니다.");
                return;
            default:
                System.out.println("잘못된 입력입니다.");
        }
    }
  }
}
```

3. 기능 구현

- 초기 기능 요구사항 분석 결과 이후로 추가된 기능은 다음과 같다. 관리자 기능에 접속(로그인) 내역 관리, 사용자 삭제 기능이 추가되었고, 사용자의 비밀번호 연속 실패로 계좌가 잠기는 기능은 편의상 사용자 기능에 작성했다.

관리자 :

계좌 입출금 내역 관리

휴면 계좌 관리

관리자 로그인

접속 내역 관리

사용자 삭제

사용자 :

계좌 개설

계좌 입금

계좌 출금

계좌 이체

계좌 잔액 / 거래내역 조회

계좌 잠금 / 해지

이체 한도

비밀번호 변경

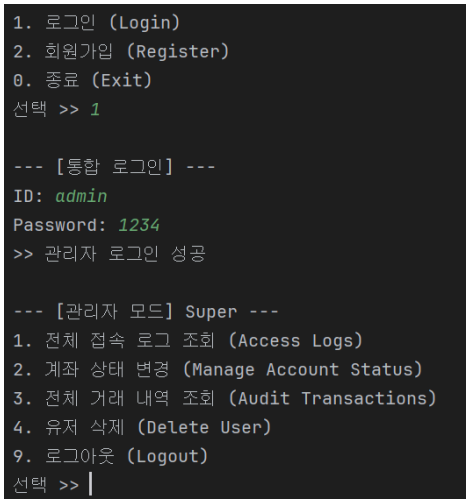
사용자 로그인

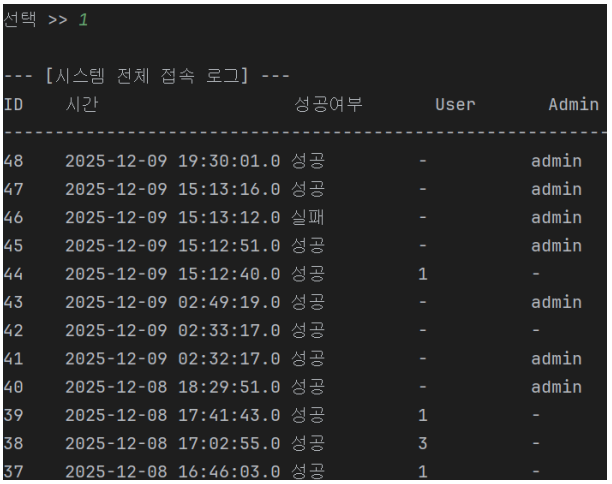
비밀번호 오류 잠금

- 기능 구현이 비슷한 일부를 제외한 전체 기능구현 스크린샷이다.

관리자 로그인

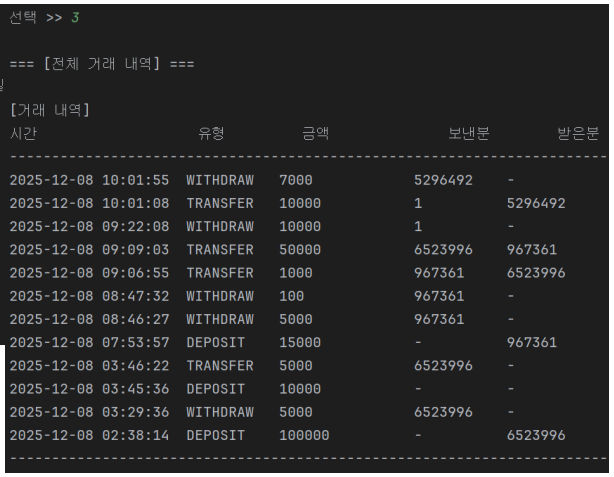
접속 내역 관리





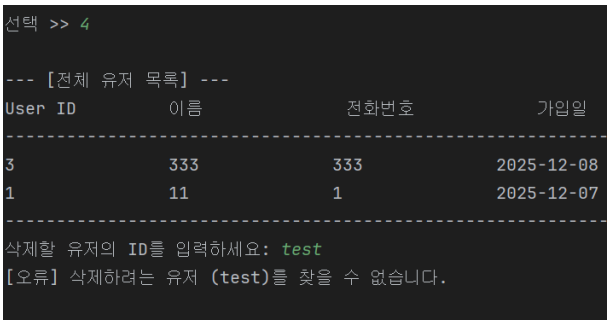
계좌 상태 변경

거래 내역 관리



사용자 삭제

사용자 로그인



계좌 개설

계좌 입금 (출금)

```
선택 >> 1

--- [계좌 개설] ---
고객님(1) 명의의 새 계좌를 개설합니다.
사용하실 계좌 비밀번호(4자리)를 입력하세요: 1111
설정할 1회 이체 한도를 입력하세요 (예: 1000000): 1000000
[성공] 계좌가 개설되었습니다. 계좌번호: 2115
```

```
--- [입금] ---

--- [내 계좌 목록] ---
계좌번호            상태          잔액              이체한도
-----------------------------------------------------------
1               DORMANT     80000           10000
2115            ACTIVE      0               1000000
967361          ACTIVE      58900           1000000
5296492         ACTIVE      3000            100000
6523996         ACTIVE      41000           1000000
-----------------------------------------------------------
입금할 계좌번호 입력: 1
입금액 입력: 10000
[거부] 휴면(DORMANT) 계좌입니다. 상태를 '정상'으로 변경 후 이용하세요.
```

계좌 이체

```
--- [계좌 이체] ---

--- [내 계좌 목록] ---
계좌번호            상태          잔액              이체한도
-----------------------------------------------------------
1               DORMANT     80000           10000
2115            ACTIVE      10000           1000000
967361          ACTIVE      58900           1000000
5296492         ACTIVE      3000            100000
6523996         ACTIVE      41000           1000000
-----------------------------------------------------------
내 출금 계좌번호: 2115
계좌 비밀번호(4자리): 1111
받을 계좌번호(상대방): 967361
이체할 금액: 5000
[DB 알림] 이체 처리가 완료되었습니다.
[성공] 5000원을 이체했습니다.
받는 분: 1 (계좌: 967361)
```

```
선택 >> 3

--- [입금] ---

--- [내 계좌 목록] ---
계좌번호            상태          잔액              이체한도
-----------------------------------------------------------
1               DORMANT     80000           10000
2115            ACTIVE      0               1000000
967361          ACTIVE      58900           1000000
5296492         ACTIVE      3000            100000
6523996         ACTIVE      41000           1000000
-----------------------------------------------------------
입금할 계좌번호 입력: 2115
입금액 입력: 10000
[성공] 10000원이 입금되었습니다.
```

계좌 상태 변경

```
선택 >> 6

--- [계좌 상태 변경] ---

--- [내 계좌 목록] ---
계좌번호            상태          잔액              이체한도
-----------------------------------------------------------
1               DORMANT     80000           10000
2115            DORMANT     5000            1000000
967361          ACTIVE      63900           1000000
5296492         ACTIVE      3000            100000
6523996         ACTIVE      41000           1000000
-----------------------------------------------------------
상태를 변경할 계좌번호: 5296492
계좌 비밀번호: 1111
변경할 상태를 선택하세요:
1. 정상 (ACTIVE) - 입출금 가능
2. 휴면 (DORMANT) - 거래 제한
3. 해지 (CLOSED) - [주의] 복구 불가
선택 >> 3
정말로 해지하시겠습니까? 해지 후에는 복구가 불가능합니다. (Y/N): y
[성공] 계좌(5296492) 상태가 'CLOSED'로 변경되었습니다.
```

거래 내역 조회

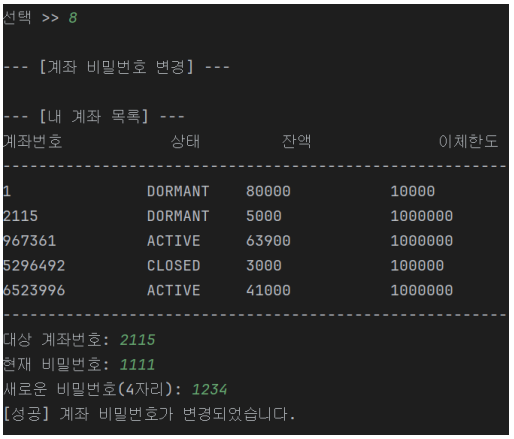
```
--- [거래 내역 조회] ---

--- [내 계좌 목록] ---
계좌번호            상태          잔액              이체한도
-----------------------------------------------------------
1               DORMANT     80000           10000
2115            DORMANT     5000            1000000
967361          ACTIVE      63900           1000000
5296492         CLOSED      3000            100000
6523996         ACTIVE      41000           1000000
-----------------------------------------------------------
조회할 계좌번호: 967361
계좌 비밀번호: 0000

[거래 내역]
시간                  유형          금액              보낸분          받은분
-------------------------------------------------------------------------
2025-12-09 19:43:35  TRANSFER    5000            2115         967361
2025-12-08 09:09:03  TRANSFER    50000           6523996      967361
2025-12-08 09:06:55  TRANSFER    1000            967361       6523996
2025-12-08 08:47:32  WITHDRAW    100             967361       -
2025-12-08 08:46:27  WITHDRAW    5000            967361       -
2025-12-08 07:53:57  DEPOSIT     15000           -            967361
-------------------------------------------------------------------------
```

계좌 비밀번호(이체한도) 변경

비밀번호 오류 잠금



4. SQL문 명세

DDL

- DELETE가 사용된 `User`의 경우, `Account`와 `Access Log`에서 FK로 참조하기 때문에 삭제시 data 삭제 시 비어있는 PK에 대한 추가 조치가 필요하다.
  - `Account`의 경우, `ON DELETE CASCADE` 옵션을 사용하여 참조하는 객체 삭제시 같이 삭제되도록 설정했다.
  - `Access Log`의 경우, `ON DELETE SET NULL` 옵션을 사용하여 참조하는 객체 삭제 시 NULL로 변환되도록 설정했다.
- DELETE로 User data삭제 시 참조하는 `Account`도 삭제되므로, `Account`를 참조하는 `Transaction`에 연쇄 작용이 발생한다.
  - `ON DELETE SET NULL` 옵션을 사용하여 `Account` 객체 삭제 시 NULL로 변환한다.

이를 통해 DELETE 시 데이터 무결성을 만족한다.

```
//초기 Database 설계

DROP DATABASE IF EXISTS bank_db;

CREATE DATABASE bank_db DEFAULT CHARACTER SET utf8;

USE bank_db;

CREATE TABLE User (
    user_id VARCHAR(50) PRIMARY KEY,     // User ID (PK)
    phone_number VARCHAR(20),
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    address VARCHAR(255),
    ssn VARCHAR(20),
```

```
    password VARCHAR(255),
    signup_date DATETIME
);

CREATE TABLE Administrator (
    admin_id VARCHAR(50) PRIMARY KEY,   // Administrator ID (PK)
    phone_number VARCHAR(20),
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    address VARCHAR(255),
    ssn VARCHAR(20),
    password VARCHAR(255)
);

CREATE TABLE Account (
    account_id INT PRIMARY KEY,         // Account ID (PK)
    account_status VARCHAR(20),
    balance BIGINT DEFAULT 0,           // 잔액 (BIGINT)
    transfer_limit BIGINT,
    opening_date DATETIME,
    owner_id VARCHAR(50),               // User_ID (FK)
    FOREIGN KEY (owner_id) REFERENCES User(user_id) ON DELETE CASCADE //DELETE 설계
고려
);

CREATE TABLE Transaction (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    transaction_type VARCHAR(20),
    amount BIGINT,
    transaction_time DATETIME,
    depos_id INT,                       // 입금 계좌 id (FK) (NULL 가능)
    withd_id INT,                       // 출금 계좌 id (FK) (NULL 가능)
    FOREIGN KEY (depos_id) REFERENCES Account(account_id) ON DELETE SET NULL,
    FOREIGN KEY (withd_id) REFERENCES Account(account_id) ON DELETE SET NULL
); //DELETE 고려

CREATE TABLE Access_Log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    access_date DATETIME,
    success_or_not BOOLEAN,             // 성공(1)/실패(0)
    user_id VARCHAR(50),                // User (FK) (NULL 가능)
    admin_id VARCHAR(50),               // Admin (FK) (NULL 가능)
    FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE SET NULL,
    FOREIGN KEY (admin_id) REFERENCES Administrator(admin_id) ON DELETE SET NULL
); //DELETE 고려

//기능 구현에 따른 Entity 수정 반영
```

```
ALTER TABLE Account ADD account_password VARCHAR(4) DEFAULT '0000';

ALTER TABLE Account ADD wrong_pw_count INT DEFAULT 0;
```

DML

Java JDBC의 `PreparedStatement`를 사용하여 SQL Injection을 방지하였으며, 다음과 같은 SQL문을 통해 수행된다.

1. 사용자 및 관리자 관리

- 사용자 회원가입 (`UserRepository.save`)
- 사용자 로그인 인증 (`UserRepository.findByUserIdAndPassword`)
- 특정 사용자 정보 조회 (`UserRepository.findByUserId`)
- 회원 탈퇴 및 정보 삭제 (`UserRepository.deleteById`)
- 관리자 로그인 인증 (`AdminRepository.findByAdminIdAndPassword`)
- 접속 로그 기록 (`AccessLogRepository.save`)

```
INSERT INTO user (user_id, phone_number, first_name, last_name, address, ssn, password, signup_date)
VALUES (?, ?, ?, ?, ?, ?, ?, ?);

SELECT * FROM User WHERE user_id = ? AND password = ?;

SELECT * FROM User WHERE user_id = ?;

DELETE FROM User WHERE user_id = ?;

SELECT * FROM Administrator WHERE admin_id = ? AND password = ?;

INSERT INTO access_log (access_date, success_or_not, user_id, admin_id)
VALUES (?, ?, ?, ?);
```

2. 계좌 관리

- 신규 계좌 개설 (`AccountRepository.save`)
- 내 계좌 목록 조회 (`AccountRepository.findAllByOwnerId`)
- 계좌 상세 정보 조회 (`AccountRepository.findById`)
- 계좌 비밀번호 변경 (`AccountRepository.updateAccountPassword`)
- 이체 한도 변경 (`AccountRepository.updateTransferLimit`)
- 비밀번호 오류 횟수 증가 (`AccountRepository.increaseWrongPwCount`)
- 비밀번호 오류 횟수 초기화 (`AccountRepository.resetWrongPwCount`)

```
INSERT INTO Account (account_id, account_status, balance, transfer_limit, opening_date, owner_id, account_password)
VALUES (?, ?, ?, ?, ?, ?, ?);

SELECT * FROM account WHERE owner_id = ?;

SELECT * FROM account WHERE account_id = ?;

UPDATE Account SET account_password = ? WHERE account_id = ?;

UPDATE Account SET transfer_limit = ? WHERE account_id = ?;

UPDATE Account SET wrong_pw_count = wrong_pw_count + 1 WHERE account_id = ?;

UPDATE Account SET wrong_pw_count = 0 WHERE account_id = ?;
```

3. 거래 처리

- 단순 잔액 변경 (입금/출금) (`AccountRepository.updateBalance`)
- 거래 내역 기록 (`TransactionRepository.save`)
- 계좌 이체 트랜잭션 (`AccountRepository.transfer`)
  1. 출금 (보내는 사람)
  2. 입금 (받는 사람)
  3. 이체 기록 저장
- 거래 내역 조회 (사용자) (`TransactionRepository.findAllByAccountId`)

```
UPDATE account SET balance = balance + ? WHERE account_id = ?;

INSERT INTO transaction (transaction_type, amount, transaction_time, depos_id, withd_id)
VALUES (?, ?, ?, ?, ?);

UPDATE account SET balance = balance - ? WHERE account_id = ?;

UPDATE account SET balance = balance + ? WHERE account_id = ?;

INSERT INTO transaction (transaction_type, amount, transaction_time, depos_id, withd_id)
VALUES ('TRANSFER', ?, ?, ?, ?);

SELECT * FROM Transaction WHERE depos_id = ? OR withd_id = ? ORDER BY transaction_time DESC;
```

4. 관리자 기능

- 전체 사용자 목록 조회 (`UserRepository.findAll`)
- 계좌 상태 변경 (휴면/잠금/해지) (`AccountRepository.updateAccountStatus`)

- 전체 계좌 목록 조회 (`AccountRepository.findAll`)
- 전체 접속 로그 조회 (`AccessLogRepository.printAllLogs`)

```
SELECT * FROM User ORDER BY signup_date DESC;

UPDATE Account SET account_status = ? WHERE account_id = ?;

SELECT * FROM Account ORDER BY account_id ASC;

SELECT * FROM Access_Log ORDER BY access_date DESC;
```