INTEGRATION TESTING ASSIGNMENT

BANK SYSTEM: BANK ENTITIES + MAPPER

For a bank system, integration tests need to be set up for a bank mapper capable of managing accounts and credit cards in a database

The database layer tests should be created using a simple direct database connection and a SQL script file, mocks, an embedded database, an in-memory database and a database testing framework such as DBUnit or another similar or better database testing framework

1. Implement account and credit card classes based on the following interfaces…

Account interface:

```
public interface AccountInterface
{
   public void setId(int id);
   public int getId();
   public void setBalance(double balance);
   public double getBalance();
   public void deposit(double amount);
   public void withdraw(double amount);
   public void deposit(double amount);
   public void withdraw(double amount);
}
```

CreditCard interface:

```
public interface CreditCardInterface
{
   public void setId(int id);
   public int getId();
   public void setAccount(AccountInterface account);
   public AccountInterface getAccount();
   public void setCreated(Date created);
   public Date getCreated();
   public void setLastUsed(Date lastUsed);
   public Date getLastUsed();
   public void setPinCode(int pinCode);
   public int getPinCode();
   public void setWrongPinCodeAttempts(int wrongPinCodeAttempts);
   public void addWrongPinCodeAttempt();
   public int getWrongPinCodeAttempts();
   public void resetWrongPinCodeAttempts();
   public void setBlocked(boolean blocked);
   public boolean isBlocked();
}
```

2. Implement a database bank mapper class based on the following interface…

BankMapperInterface

```
public interface BankMapperInterface
{
   public void setDataSource(DataSource ds);
   public CreditCardInterface createCreditCard(CreditCardInterface creditCard);
   public CreditCardInterface updateCreditCard(CreditCardInterface creditCard);
   public CreditCardInterface getCreditCard(int id);
   public List<CreditCardInterface> getCreditCards();
   public AccountInterface createAccount(AccountInterface account);
   public void updateAccount(AccountInterface account);
   public AccountInterface getAccount(int id);
   public List<AccountInterface> getAccounts();
}
```

3. Create a main method and try out some of the different methods of the bank mapper on a production database

4. Test the different methods of the database bank mapper sufficiently using different approaches, involving all the following details…
    - Executing all tests on a test databases instead of the production database
    - Focusing on cleaning the database and keeping a known state for all the tests
    - Setting up a simple direct database connection and a SQL script to set a test database to a known state to be used by some tests
    - Using DBUnit along with XML files to set a test database to a known state to be used by some tests or do something similar with another database testing framework
    - Using an embedded database for some of the tests
    - Using an in-memory database for some of the tests
    - Using mocks to test the bank mapper without any test database
    - Setting a test database to a known state, both before each test is run and before all tests are run, for either some of the database testing framework tests, the embedded tests, the in-memory tests or the mock tests
    - Setting up all the tests with JUnit5 and Hamcrest or other testing frameworks / libraries, that will improve readability, simplicity, effectiveness and thoroughness of the database bank mapper tests in general