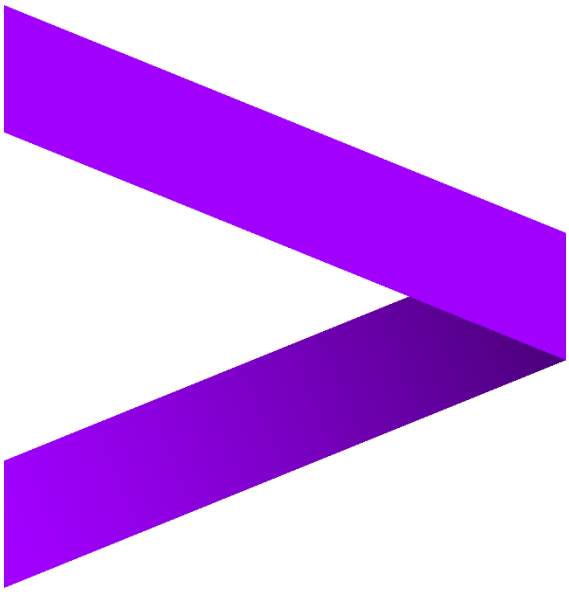
accenture

Java Full Stack - NINJA

Java Full Stack with React Coding based Learning.



Coding Challenge - 1

Revision History

Sr no	Version	Author	Date	Comments
1	1.0	jagadish.sanjeevi	12-March-2024	Initial draft

Coding Challenge -1

Title: Banking Application – Transaction Analyzer

Description:

You are tasked with building an advanced banking application that leverages Java 8 features such as Lambda expressions, Stream API, and Optional classes. The application should handle database operations using MySQL and JPA for persistence. Additionally, you are responsible to write a JUNIT test classes to demonstrate the proper code coverage by writing the appropriate testcases.

This is basically a Console based applications so you can write a Java class with main() to execute the application.

Challenge Overview:

1. Database Setup:

Set up a MySQL database with tables for Customer, Account, Transaction, and any other necessary entities. Here is a simplified database schema setup using MySQL:

Customers Table:

Field	Type	PRIMARYKEY	Extra
Id	int	YES	AUTO_INCREMENT
first_name	varchar(50)	NO	
last_name	varchar(50)	NO	
Email	varchar(100)	NO	
Password	varchar(64)	NO	Password should be Hashed

Accounts Table:

Field	Type	PRIMARYKEY	Extra
account_number	bigint	YES	
customer_id	Int	Foreign Key MUL	Refers to Customer table id
account_type	enum('SAVINGS','SALARY','CURRENT','FIXEDDEPOSIT')	NO	
balance	decimal(10,2)	NO	

Transactions Table:

Field	Type	PRIMARYKEY	Extra
id	int	YES	AUTO_INCREMENT
account_number	bigint	Foreign Key MUL	Refers to Accounts table account_number
transaction_type	enum('DEPOSIT','WITHDRAWAL')	NO	
amount	decimal(10,2)	NO	
transaction_date	timestamp	NO	

Note: with this schema setup each customer can have multiple accounts (one-to-many relationship between Customers and Accounts) and each account can have multiple transactions (one-to-many relationship between Accounts and Transactions).

Sample Data to insert into the Tables:

```
-- Sample Data for Customers Table
INSERT INTO Customers (id,first_name, last_name, email, password) VALUES
(1001,'JAG', 'SAN', 'jassan@example.com', SHA2('jas@123', 256)),
(1002,'Jane', 'Smith', 'jane@example.com', SHA2('jane@123', 256));

-- Sample Data for Accounts Table
INSERT INTO Accounts (customer_id, account_type, account_number, balance)
VALUES
(1001, 'SAVINGS', 1234567890, 1000.00),
(1001, 'SALARY',  2345678901, 500.00),
(1002, 'SAVINGS', 3456789012, 1500.00);

-- Sample Data for Transactions Table (30 records)
INSERT INTO Transactions (account_number, transaction_type, amount,
transaction_date) VALUES
-- For Account Number 1234567890
(1234567890, 'DEPOSIT', 500.00, '2023-11-12'),
(1234567890, 'WITHDRAWAL', 200.00, '2023-10-22'),
(1234567890, 'DEPOSIT', 100.00, '2023-08-19'),
(1234567890, 'WITHDRAWAL', 50.00, '2023-07-28'),
(1234567890, 'DEPOSIT', 700.00, '2023-06-14'),
(1234567890, 'WITHDRAWAL', 100.00, '2023-05-09'),
(1234567890, 'DEPOSIT', 300.00, '2023-03-23'),
(1234567890, 'WITHDRAWAL', 150.00, '2023-02-28'),
-- For Account Number 2345678901
(2345678901, 'DEPOSIT', 1000.00, '2023-01-11'),
(2345678901, 'WITHDRAWAL', 200.00, '2022-12-05'),
(2345678901, 'DEPOSIT', 200.00, '2022-10-29'),
(2345678901, 'WITHDRAWAL', 100.00, '2022-09-14'),
(2345678901, 'DEPOSIT', 500.00, '2023-11-12'),
(2345678901, 'WITHDRAWAL', 300.00, '2023-10-22'),
(2345678901, 'DEPOSIT', 150.00, '2023-08-19'),
(2345678901, 'WITHDRAWAL', 200.00, '2023-07-28'),
-- For Account Number 3456789012
(3456789012, 'DEPOSIT', 600.00, '2023-06-14'),
(3456789012, 'WITHDRAWAL', 100.00, '2023-05-09'),
(3456789012, 'DEPOSIT', 200.00, '2023-03-23'),
(3456789012, 'WITHDRAWAL', 50.00, '2023-02-28'),
(3456789012, 'DEPOSIT', 900.00, '2023-01-11'),
(3456789012, 'WITHDRAWAL', 150.00, '2022-12-05'),
(3456789012, 'DEPOSIT', 400.00, '2022-10-29'),
(3456789012, 'WITHDRAWAL', 300.00, '2022-09-14');
```

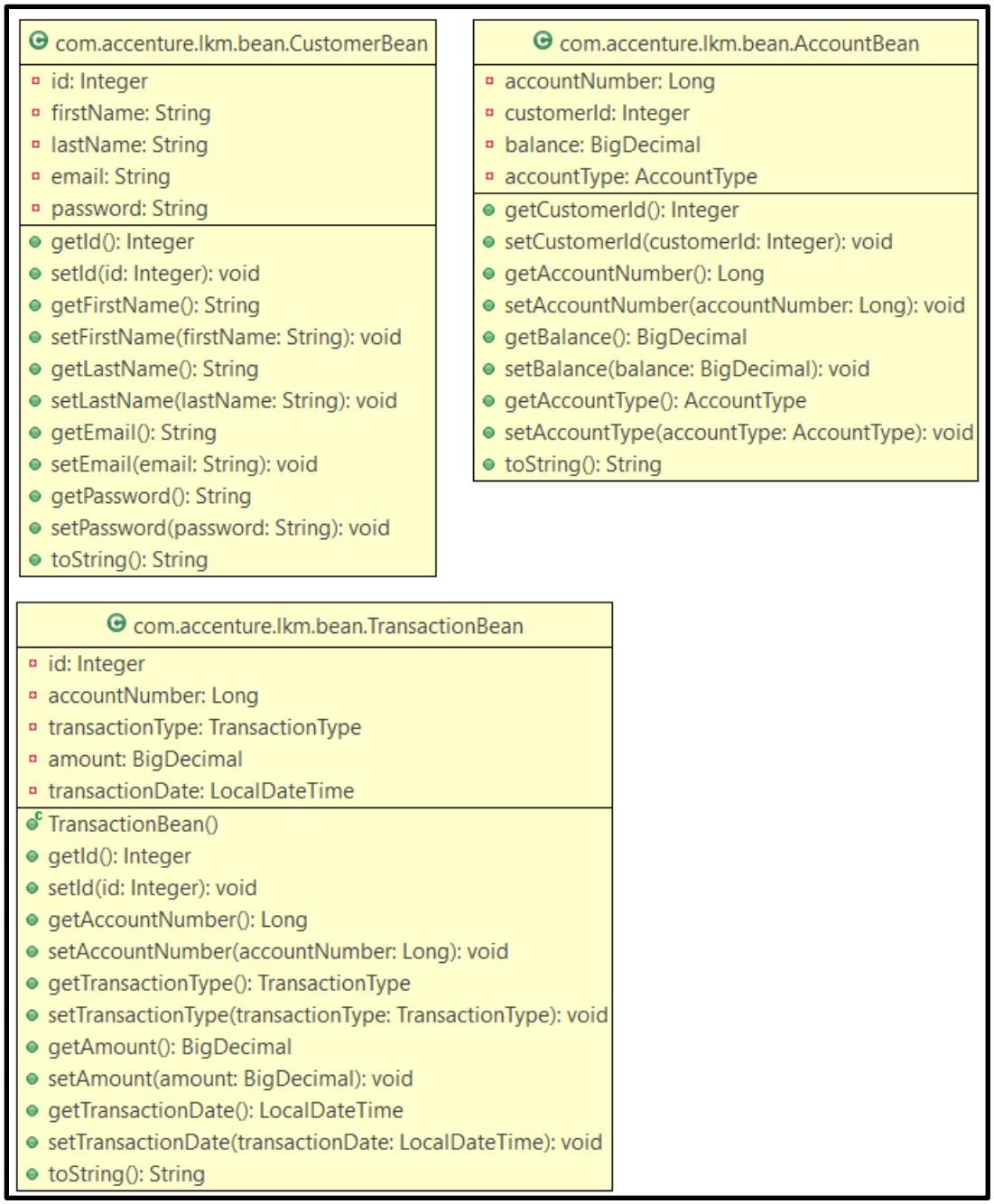
2. Backend Development:

A. Implement the DTO(Bean) and Entity classes for Customer, Account, and Transaction tables with appropriate attributes and relationships.

Create ENUM types for TransactionType and AccountType with FIXED ENUM values.

Note: Use appropriate JPA annotation with the Entity classes attributes.

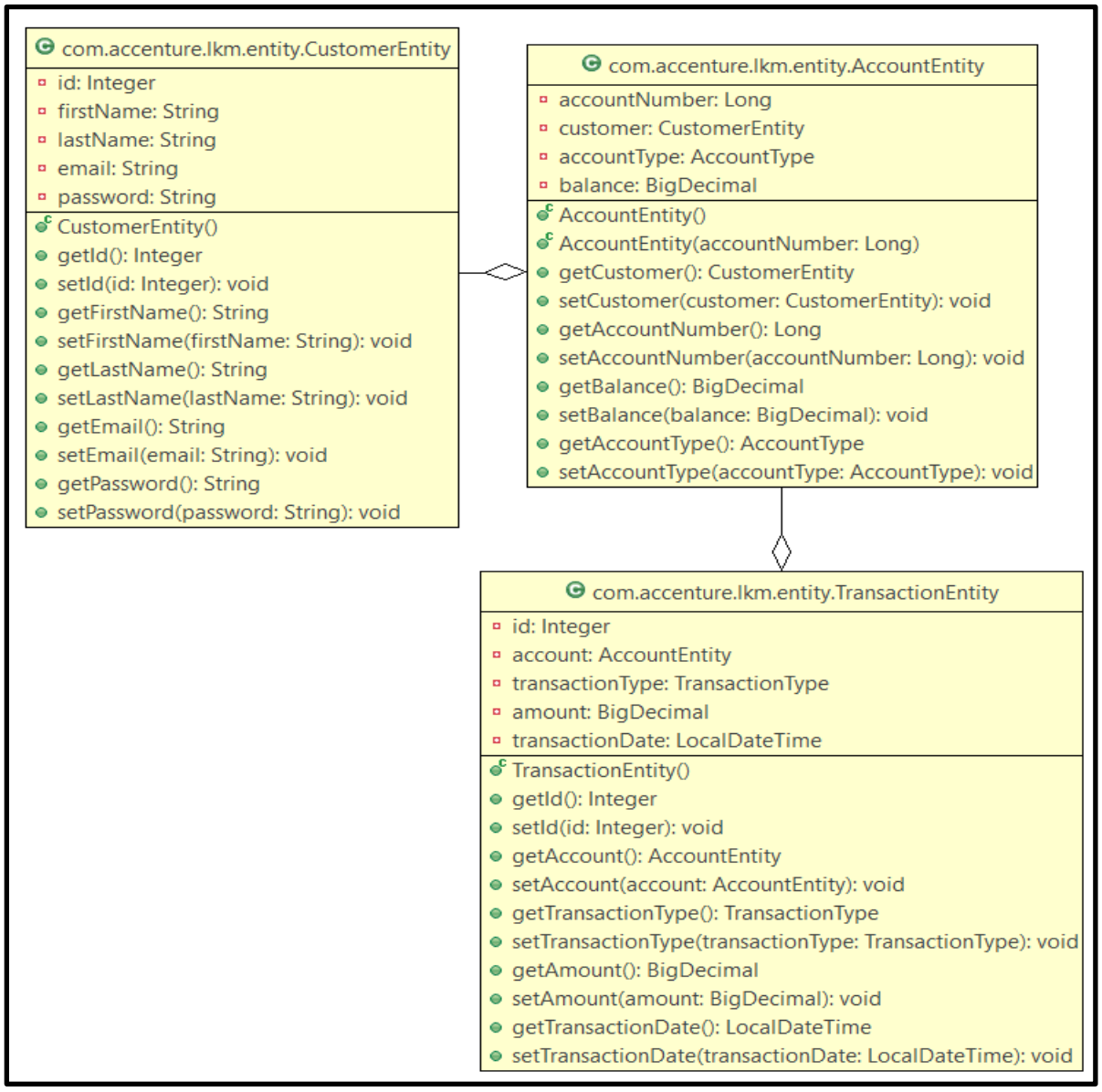
Bean Classes in com.accenture.lkm.bean package:



ENUM in com.accenture.lkm.enums package:

```
public enum AccountType {
    SAVINGS,
    CURRENT,
    SALARY,
    FIXEDDEPOSIT,
    DEMAT
}
public enum TransactionType {
    DEPOSIT,
    WITHDRAWAL
}
```

Entity Classes in com.accenture.lkm.entity package:



B. Implement the repository layers for Customer, Account, and Transaction with below mentioned functionalities.

DAO Interface/Class	Functionality
AccountDAO	Open New Account
	Balance Enquiry
CustomerDAO	Update Customer details
	Change Login Password (password is stored in SHA-256 format in DB)
	Register for a New Customer(password is stored in SHA-256 format in DB)
TransactionDAO	Get Transactions by Account Number
	Perform Deposit/Withdraw Transaction
	get all transactions from DB

For performing the above operations on the given entities in DAO layer use JPA code and if need utilize Stream API wherever it is needed [ex: Bean to Entity conversion and vice versa.]

C. Implement the service layers for Customer, Account, Transaction and Transaction Analyzer with below mentioned functionalities.

Service Interface/Class	Functionality
AccountService	Open New Account Note: 10 Digit Account number should be AUTO GENERATED use a private method for the same and make sure it is unique (not existing in the DB)
	Balance Enquiry
CustomerService	Update Customer details
	Change Login Password
	Register New Customer
TransactionService	Get Transactions by Account Number
	Perform Deposit/Withdraw Transaction Note: Account balance should be updated Withdraw not allowed if balance is less than the transaction amount
	get all transactions from DB
Transaction Analyzer	Get the total amount deposited for each account.
	Get the total amount withdrawn for each account.
	Get total Number of Deposit and Withdraw Transaction for each Account
	Get the Total Amount Transacted Per Account for A Given Month
	Get Account Numbers with Transactions Exceeding Threshold
	Get Account Numbers with Unusual Transaction Patterns (i,e where the number of withdraw is greater than twice of deposit transactions)

Use Java 8 features like Lambda expressions for functional programming paradigms for implementing functionalities in the service layer.

Ensure proper error handling using Optional classes for nullable return types where ever applicable.

Note: Concurrency and Performance:

Implement concurrent processing using Java 8 parallel streams for tasks like processing of transactions.

If required optimize database interactions and queries for better performance.

D. JUNIT Test Classes:

Write a proper JUNIT test classes for all the methods in the Service and Repository layer and ensure proper code coverage.

E. Java Classes with main() in com.accenture.lkm.ui package

Create a respective Java UITester classes in com.accenture.lkm.ui package with main() to invoke all the functionalities from the Service Layer.

This challenge is designed to be comprehensive and time-consuming, requiring proficiency in various Java 8 features, database interactions, Core Java Fundamentals, Design Patterns like DAO, Factory design patterns and software engineering principles.

Plan your time effectively and tackle each aspect systematically. Good luck!