

Unibet Bank Problem Solution

Table of Contents

Document Purpose	3
Objective	3
Technologies Used	3
Execution Environment	3
Build Tools	3
Testing Frameworks	3
Other Frameworks.....	3
Database.....	3
Code Architecture	4
New Packages	4
Class Diagram.....	5
Test Strategy	6
JUnit Testing.....	6
Mutation Testing	6
Test Case Execution.....	7
Run Instructions	7

Document Purpose

This document aims at explaining the design approaches and architecture of the solution.

Objective

To provide an efficient and optimized solution for a banking application problem.

Technologies Used

Execution Environment

Java 1.7

Build Tools

Maven

Testing Frameworks

Junit

Mockito

Pitest (Mutation Test)

Other Frameworks

JPA

Hibernate

Spring

Database

H2 (in memory)

Code Architecture

The application code has been divided into the following layers with each layer providing a specific purpose.

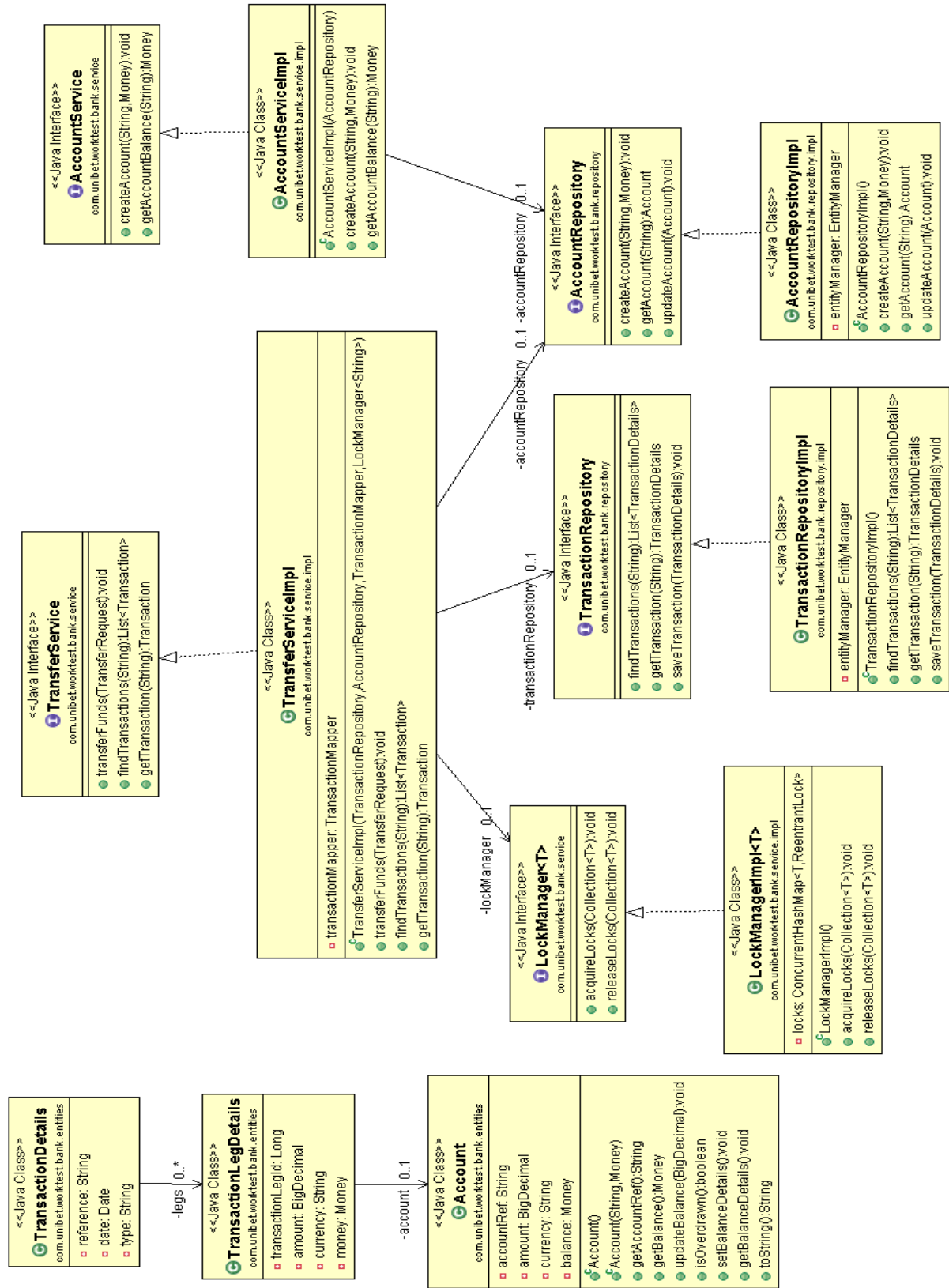
- **Entity** – A layer, which provides object relational mapping.
- **Repository** – A layer, which interacts with database to perform the required database operations.
- **Service** – A layer, which interacts with repository in order to perform the required operations.

In order to facilitate the design a lot of supporting classes have been developed and are kept under different packages. Following are the newly created packages :

New Packages

- com.unibet.worktest.bank.entities
- com.unibet.worktest.bank.exception
- com.unibet.worktest.bank.mapper
- com.unibet.worktest.bank.repository
- com.unibet.worktest.bank.repository.impl
- com.unibet.worktest.bank.service
- com.unibet.worktest.bank.service.impl

Class Diagram



Test Strategy

The testing has been done rigorously in order to provide maximum code coverage. The aim is to test each and every unit, which involves business logic making use of Junit, Mockito and PITTest framework. Following testing techniques are used :-

Junit Testing

Every class containing business logic has been tested in isolation with the other classes. Any dependency that may occur during the unit testing has been mocked using Mockito framework in order to test the desired functionality.

Mutation Testing

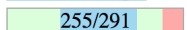
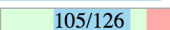
Mutation testing is a concept in which faults (or mutations) are automatically seeded into our application code. If the test case fails then the mutation is killed else it survives.

It also generates the html reports, which graphically shows the amount of test coverage that has been done.

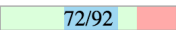
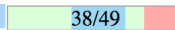
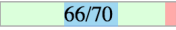
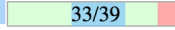
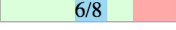
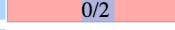
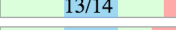
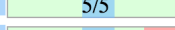
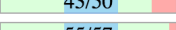
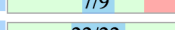
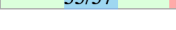
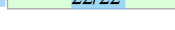
In order to facilitate mutation testing, the required dependencies have been added as maven dependencies.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
17	88%  255/291	83%  105/126

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
com.unibet.worktest.bank	5	78%  72/92	78%  38/49
com.unibet.worktest.bank.entities	3	94%  66/70	85%  33/39
com.unibet.worktest.bank.exception	2	75%  6/8	0%  0/2
com.unibet.worktest.bank.mapper	3	93%  13/14	100%  5/5
com.unibet.worktest.bank.repository.impl	2	86%  43/50	78%  7/9
com.unibet.worktest.bank.service.impl	2	96%  55/57	100%  22/22

Example Screenshot

Test Case Execution

- a. For running the test cases, go to the location where pom.xml is stored. From there you can execute maven commands :-
 - i. *“mvn clean test”* or *“mvn clean install”* to run the junit test cases.
 - ii. *mvn org.pitest:pitest-maven:mutationCoverage* to run Mutation tests. This will create html report for test coverage at c:/tmp directory inside your project folder.

Run Instructions

The file “runBankTest.bat” can be run to test the bank solution (assuming maven and java are installed on the system with PATH set correctly).