Bank Work Test Assignment for Java EE Developers

1 Introduction

This document describes the requirements for completing a basic Java development assignment. The purpose of the assignment is to get an indication of a developer's ability to design and build solutions using modern Java SE/EE technology.

This document together with the source code informs you what the assignment is about. There are some strict guidelines on how the work needs to be performed. You should read these instructions carefully before you begin working on the solution.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

1.1 Contents

The distribution archive consists of the following:

- Source code that defines a contract you need to implement
- Unit tests that demonstrates the key functionality
- A complete project structure and build file

1.2 Mission

Your task is to implement a double-entry bookkeeping banking service, simply called the **Bank**. Fortunately, all API design work for the project is already completed. The functionality of the service is defined by the business interfaces and test classes.

Your mission is to deliver a fully working solution by implementing these business interfaces.

1.3 Project Status

The Bank service contains the following interfaces that MUST be implemented in order to complete the project:

- **AccountService** – Service for creating monetary accounts and query for account balance
- **TransferService** – Service for executing multi-legged monetary transactions towards accounts
- **BankFactory** – Factory interface for providing instances of the business services

2 Business Requirements

This section lists the functionality that MUST be implemented to successfully complete the assignment.

Functional Requirements

- Ability to create monetary accounts. For simplicity, a client is allowed to create its own accounts with an initial balance
- Ability to transfer money between accounts following the *double-entry bookkeeping principle *)*
- An account MUST NOT be overdrawn, i.e. have a negative balance

- A monetary transaction MAY support multiple currencies as long as the total balance for the transaction legs with the same currency is zero

*) Double-entry bookkeeping involves making at least two entries for every transaction. A debit in one account and a corresponding credit in another account. The sum of all debits should always equal the sum of all credits, providing a simple way to check for errors. The concepts of debit and credit are simplified here by specifying that a transaction can have either a positive or negative value.

Non-functional Requirements

- The solution MUST be able to handle concurrent requests from multiple threads and JVMs in a safe way
- Account details and financial transactions MUST be stored in a relational database

Terminology

- **Client -** A client is an application using the service, in this case unit tests
- **Account** - A system record with an arbitrary soft reference and a Money balance
- **Money -** A monetary unit consisting of an amount and ISO-4217 currency code
- **Transaction -** A monetary transaction between at least two different accounts
- **Transaction Leg** - An element of a monetary transaction defining a credit or debit towards a single account
- **Transaction Type -** A loosely defined type or context for a monetary transaction

3 Implementation Guidelines

This section lists the guidelines your implementation MUST follow to successfully complete the assignment.

Source Code and Dependencies

You MUST NOT modify any existing Java source code including the test classes. These classes are annotated with @Sealed as a reminder.

- You MAY edit the provided pom.xml but only for adding dependencies or plugin configurations.
- The dependencies listed in the pom.xml MAY be used.
- You MAY provide your own test classes.

Implementation

- All methods in the main business services described above MUST be implemented.
- A working implementation of the **BankFactory** interface MUST be named and placed in the same namespace, i.e: **"com.unibet.worktest.bank.BankFactoryImpl"**.
- All other source code you write MUST be placed in another namespace than **"com.unibet.worktest.bank".**

Documentation

The code you write SHOULD be clear, consistent and self-documenting. You MUST also include a solution document that describes:

- Your major design choices and the reason for those choices.
- The exact JDK and JRE versions used
- RDBMS product and version used
- The location of any setup scripts and how to execute them if needed

Platform and Technology

You are free to use any implementation technology of choice as long as it's based on Java platform JDK 1.7 or later. Use of 3rd party frameworks and products is allowed and encouraged as long as they are open-source.

The solution MAY support Java EE container deployments but your solution must not require it. Acceptable RDBMS choices are PostgreSQL, MySQL or H2 in embedded mode.

Unspecified functionality

Avoid adding functionality that is not specified, such as:

- Currency conversion
- Presentation logic
- Remoting endpoint

4 Deliverables

The solution MUST be built with Maven 3.x and MUST be possible to build and test using the "mvn test" goal. The deliverable must consist of a single zip file or tar.gz. The pom.xml is already configured to build the deliverable archive through the assembly plugin. You only need to ensure it contains all required artifacts. The archive MUST contain the original directory hierarchy along with added files or directories needed for the implementation, including:

- Source code for new classes and any modified build files, SQL script files and so forth
- Your solution description, **solution.txt** or **solution.pdf** at root level

5 Evaluation Process

First we run the code ensuring that all requirements are fulfilled. Then we investigate the design and implementation details of the solution. Equally important areas of interest to the reviewer are:

- **General**
  - Overall design choices and implementation
  - Technology choices and motivations for those choices
  - Functional and non-functional correctness
- **Documentation**
  - Javadoc source documentation and comments
  - Design documentation and motivations
- **Code Quality**
  - Error handling
  - Testability
  - Use of coding standards and code readability

6 Final Words

There is no time limit on completing the assignment. Good luck!