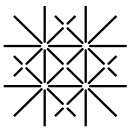

University of Basel

Prof. Dr. Florina Ciorba
Prof. Dr. Heiko Schuldt
Prof. Dr. Christian Tschudin
Prof. Dr. Isabel Wagner

Foundations of Distributed Systems Fall 2025

Exercise 4 Hand-in: 2025 November 09 (11:59pm)

Advisors: Yiming Wu (yiming.wu@unibas.ch)

Modalities of work: The exercise is solved in teams of 4 people.

Modalities of the exercise: The solutions to the exercise must be uploaded to ADAM before the deadline. There will be an interview scheduled between November 11th and November 14th. During the interview, the work and the understanding of the technical background, which includes contents from both the lectures and the assignments, will be evaluated. Therefore, each group member has to attend this interview in order to be awarded points for the exercise. Please be prepared to demonstrate your solutions on your machine! The spokesperson is required to schedule an appointment by reserving a slot on `https://xoyondo.com/dp/hewygmpgb59g5ep` for the whole team. Please note, that reservations are made on a "first come first served" basis. If there is any question or issue regarding the interview, please send an email to yiming.wu@unibas.ch.

Motivation and Application

In this exercise, you will implement a distributed banking application with different systems and tools. The example application offers bank transfers from a source account to a destination account. This *transfer* procedure consists of two basic operations. *Withdraw* will reduce the balance of a given account and *deposit* will increase its balance. In our distributed system setting, a banking transfer may of course involve different database servers.

Question 1: Distributed Transactions with Java (20 points)

The system environment for this exercise will consist of Oracle 21c XE database servers installed on two servers of the DBIS group (dmi-dbis-v10 and dmi-dbis-v11). You will

receive user accounts for the exercises, and it is recommended to use these servers for the exercise. Please note that the servers are only reachable from within the University network. Thus, you need to be in the VPN. For the purpose of this exercise, you will implement a Java application, which connects to two bank servers in order to facilitate a money transfer from one account to another. For this exercise, you do not need to implement a user interface for your Java application. Since the focus is on the distributed transaction, you can have the transfer details hard-coded in your Java code.

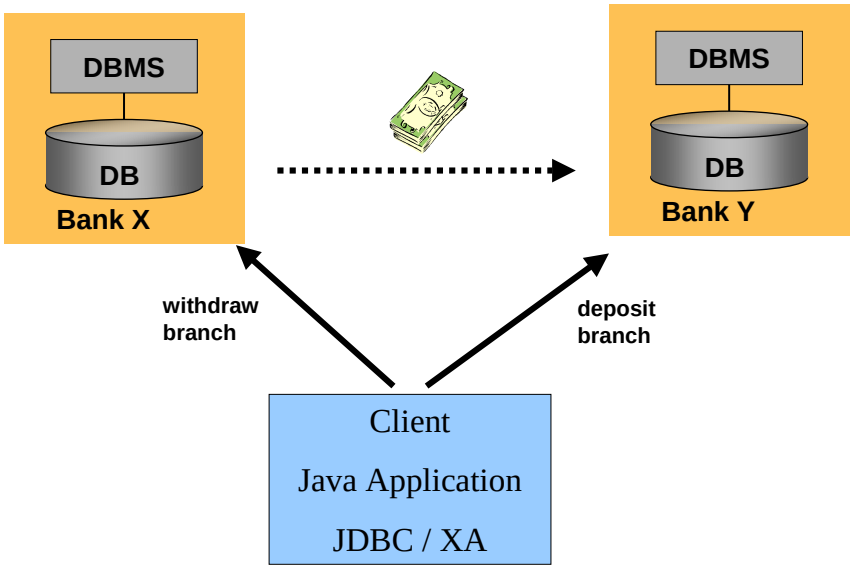

Figure 1: Money transfer facilitated by a Java application.

The Java application will consist of two different transaction branches. One transaction branch is the *withdraw* operation at the first bank. The other transaction branch is the *deposit* operation at the second bank. In order to guarantee global atomicity, you will apply the 2PC protocol in your application. In summary, you must *'prepare to commit'* foreach each successfully executed branch. Once *'prepare to commit'* of each branch have been successful, you are allowed to *commit* all branches otherwise you have to *rollback*. It is important that your application checks, if a branch has been executed successfully, (e. g., no withdrawal from a non-existing account)!

For this exercise, we provide you with a Gradle Java project (fds-2pc). It contains a JUnit test and the `OracleXaBank` class, which you can use as a starting point for your solution. In the *src/main/resources* folder, you find an SQL file (`database_schema.sql`), which allows you to create the database schema on both database servers manually (in case you write your solution on your own from scratch). However, the abstract `AbstractOracleXaBank` class already sets up the database at class initialization time.

Hint: You can find a good tutorial on the implementation of Oracle JDBC-based dis-

tributed transactions in the JDBC Developer's Guide available at:
https://docs.oracle.com/cd/E11882_01/java.112/e16548/xadistra.htm#JJDBC28000

a) Extend the `AbstractOracleXaBank` and `OracleXaBank` classes in order to facilitate a money transfer between two banks. In particular, you have to use the XA extensions of JDBC to guarantee global atomicity of the distributed transaction. In this exercise, you must not use stored procedures and database links. Keep in mind to check for proper exception handling and to throw exceptions, if error situations occur (e. g., transferring money from a non-existing account). Finally, every exception has to lead to a rollback of the complete distributed transaction. Do not forget to call the *commit* and *rollback* methods at the proper locations!

   Test your Java application for the various failure scenarios that may arise (you may use the failure scenarios defined in the previous question). In particular, check if atomicity of the distributed transaction is guaranteed. You can extend the provided JUnit test (`XaBankingAppTest`) for this task.

(15 points)

b) In the lecture, you have seen two variants of 2PC: *Presumed Abort 2PC* and *Transfer of Coordination 2PC* (It is not possible for you to implement these variants in the code, since XA does not support them.). Would it make sense to apply any of these variants to the current scenario? If so, annotate your code at the proper locations and explaining what exactly would happen at this stage of the process in either case. If the variants cannot be applied to the scenario, explain why exactly.

(5 points)

Interview (0 points)

The spokesperson is required to schedule an appointment for an interview by reserving a slot on `https://xoyondo.com/dp/hewygmpgb59g5ep` for the whole team. During the interview, the work and the understanding of the technical background is evaluated. Thus, you will be asked questions related to the topic of Exercise 3 and 4 as well as your specific solutions.

Please note that reservations are made on a "first come, first served" basis. Each group member has to attend this meeting in order to be awarded points for the exercise. Please be prepared to demonstrate your solutions on your machine!