Ebberød Bank

Flow 1 project

Project information

Project name:
Ebberød bank

Date:
18-02-2021

Flow & semester name:
Flow 1, Semester 2

Education name:
Datamatiker / Computer Science

School name:
CPHBusiness Lyngby

Group name:
Group B1

Group members:
Andreas Buch, Albert Emil Hyrde Jensen, Albert Havn Kops & Aleksander Christian Rolander langkjær.

GitHub project:
Link

Table of contents

Requirements

Functional requirements
- Program must have multiple customers.
- Each customer has one account associated with them.
- Customers must be able to put money into their accounts, take money out of their accounts, and they're not allowed to overdraw.
- Customers must be able to print out a transaction statement.
- Bank employees must be able to move money from one customer account to another, once again without overdrawing.

Non-functional requirements
- It's okay to work in whole numbers, as the program is only a proof of concept.
- The program must have a GUI but it can run in the terminal.
- Program must use data which persists in a mySQL database.
- Project has to be written in Java and be created as a Maven project.
- Use GitHubs "project & issues" feature for project oversight.

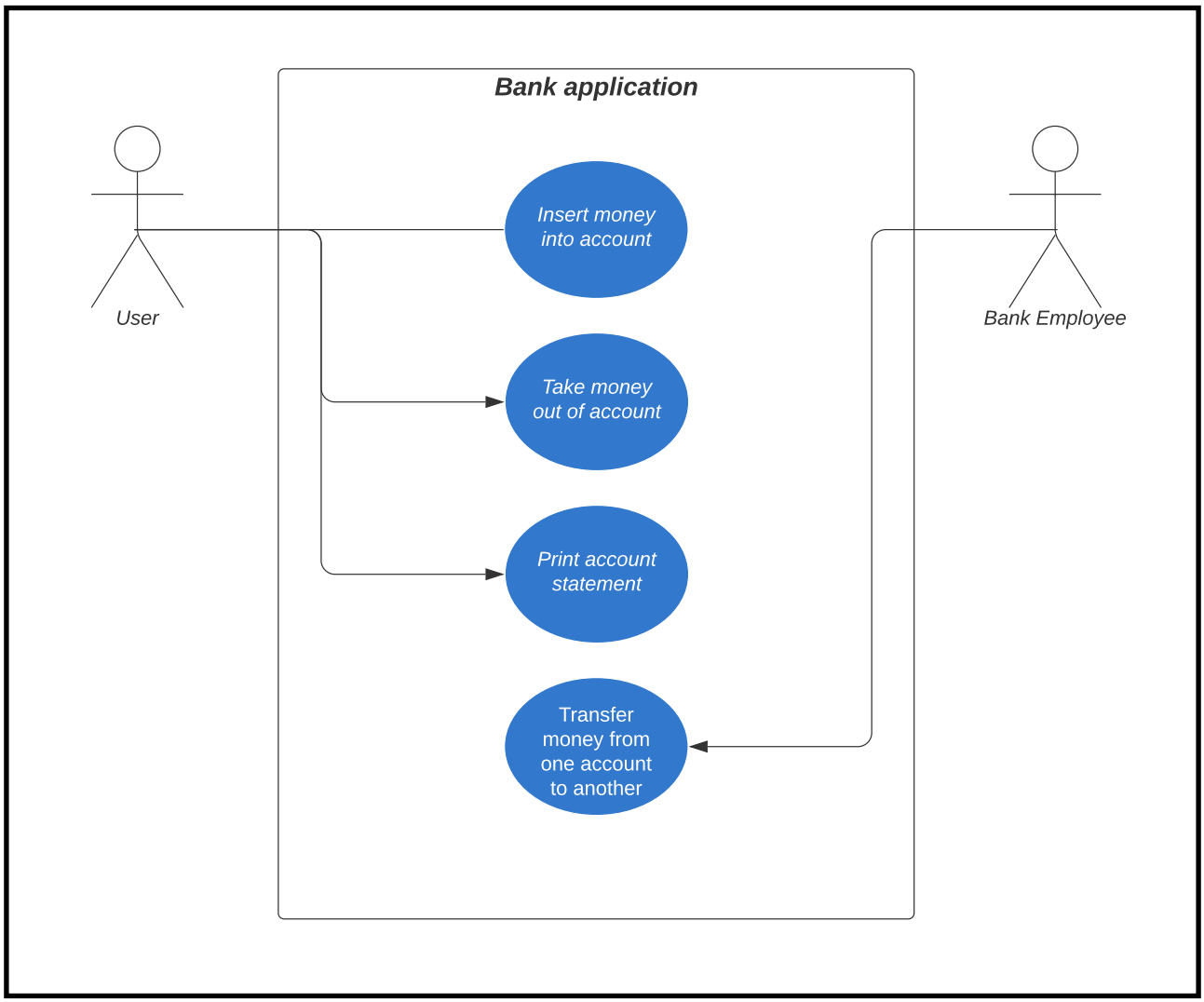
Project status

We've managed to make pretty much every requirement work before the assignment's deadline. Our program has a simple GUI with which users can interact. It also works with data persisting in a database, and we're able to deposit, withdraw and transfer money between accounts. Furthermore, the program is able to print an account statement for users, as specified in one of our user stories. With that, our banking app is practically feature complete as far as our initial requirements are concerned.

We've also been able to finish the required UML models for the project, meaning we have a finished use case diagram showing the most common use cases, an EERD showing our database relationships, and several user stories detailing the most fundamental ways our users will be interacting with the bank app, as well details the user stories' respective acceptance criteria.

The ultimate way to check up on project status, however, is by visiting our GitHub project using the link at the very top. GitHub is a great tool for version controlling a project when working with multiple people, but its other uses are great too; we're able to see every change that was made to the project, who it was made by, and when exactly it was made. So visit that link to find some raw data for questions that were left answered unsatisfactory in this segment.

Use case diagram



Use case diagram final, can be found in project folder under “PDF & UML”

User stories & acceptance criteria

User story:
As a customer of the bank, I want to be able to deposit money into my account as I accumulate more cash.

Acceptance criteria:
- Deposit amount needs to be added onto the account balance.
- Success message has to be implemented so customers know that the deposit has come through successfully.

User story:
As a customer of the bank, I want to be able to withdraw money from my account as I need it.

Acceptance criteria:
- Bank must check account balance against withdrawal amount before moving money, so that customer doesn't overdraw.
- Program must have exceptions in place for any errors, such as not having a high enough balance.
- Success message has to be implemented so customers know that a withdrawal request has come through.

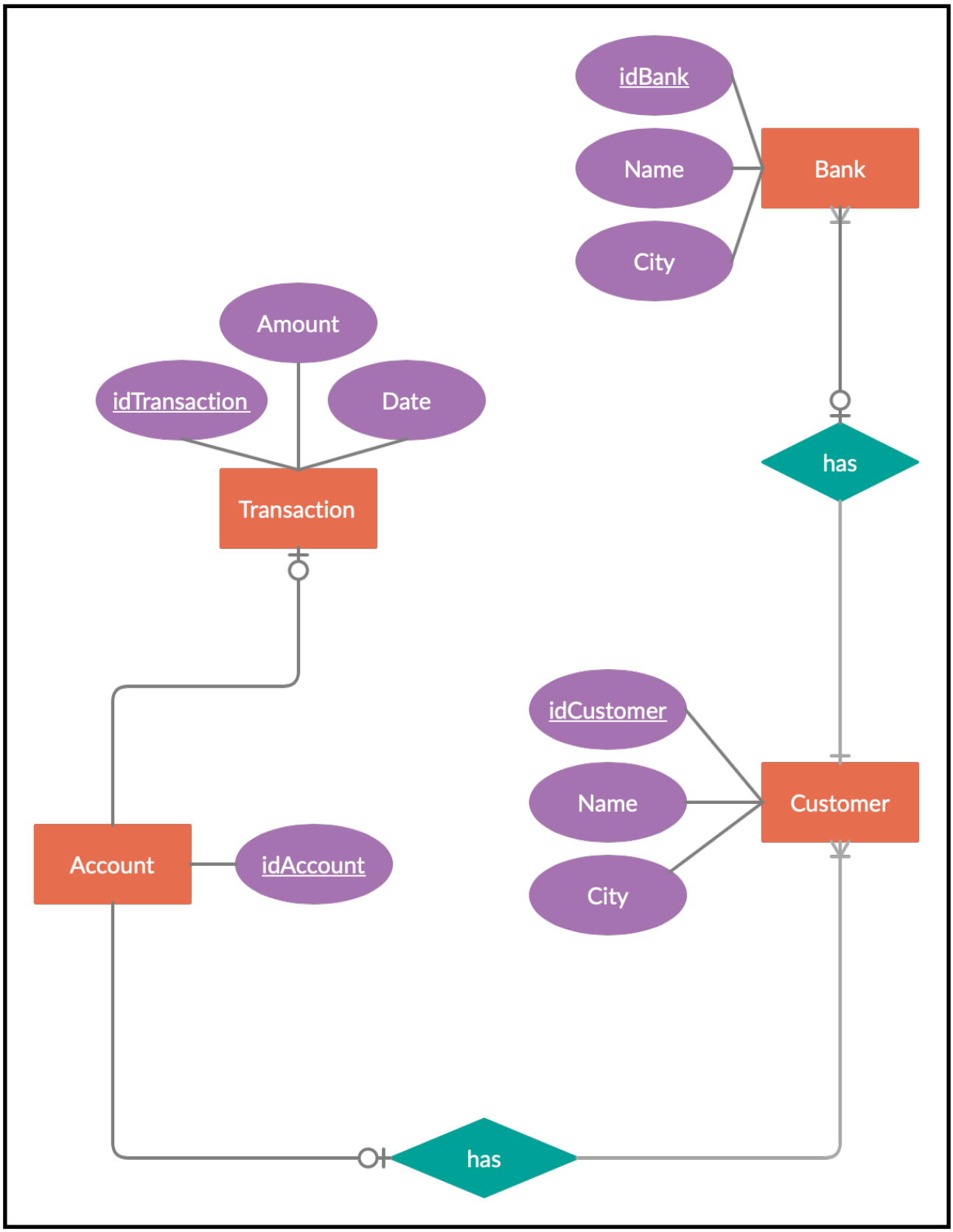
User story:
As an employee of the bank, I want to be able to move funds from one customer account to another, so as to better help with our customers requests.

Acceptance criteria:
- Bank must check account balance before moving money, so that customers account doesn't go into the negatives.
- Program must have exceptions in place for any errors that might occur, such as input mistakes or not having a high enough account balance to go through with a given transfer.
- Success message has to be implemented so customers know that a transfer has come through.

User story:
As a customer of the bank, I want to print out an account statement so that I can stay on top of how much I've spent in a given time period.

Acceptance criteria:
- Bank must be able to show current account balance to give a clear picture of a customers current financial state.
- Bank must be able to show an account statement which encompasses all three types of transaction types made on the account.
- Bank must also distinguish between these types, so that it's clear to the customer wether an amount was deposited, withdrawn or transferred to or from another account.

Enhanced entity relational diagram



Enhanced entity relational diagram final, can be found in project folder under “PDF & UML”

Reflection

Teamwork was great in our group from the beginning of the projects development. We got off to a slow start, not beginning work on the project until Tuesday, however, once we got going everyone picked something to start working on. The communication within our group was informal, and delegation of tasks was up to each individual. So we picked the tasks on a ‘first come, first served’ basis, and got to work on building the skeletal structure of the project.

Andreas started working on the pdf and use case diagram, while the two Alberts started working on the java project itself, building out the classes and program logic we’d need. Aleksander began working on the user stories and accept criteria.

In a days work, the project’s structure had been build out and we had a somewhat working prototype, which worked as proof of concept. It hadn’t yet been integrated with a database or anything fancy like that, but the logic had been built out.

GitHub didn’t present any big issues for us. We discussed to which extend we wanted to use GitHub, and came to an agreement on how to safely operate branches so as to not end up accidentally deleting or overriding any data while pushing. We shared our knowledge with each other on merging to make sure everyone was up to date, and to minimise risks, and because of this we never encountered any big issues.

Coding together wasn’t an issue for the most part as we all started working on our own subtasks and branches. But as we inched closer to completion, we had less tasks needing work and so less available tasks to choose from, which presented us with the minor problem of effectively working together on a single task. We didn’t employ any plugins like ‘Code Together’ or similar, and so we usually had one person coding while others gave verbal suggestions while watching.

As we get bigger assignments in the future and more time to complete the projects, we might need to become a bit more formal in our planning. Perhaps we will need to become better at the planning phase as the project’s and report’s scope gets bigger, and we probably have to find a way to solve our coding bottleneck once we’ve built out the structure and start working on the same classes.

Overall, the issues we faced were minor ones in the face of the assignments scope. But in the future, as our project’s scope increases, we will have to make minor adjustments to our workflow in order to follow the demand of said project.