**OBJECT-ORIENTED ANALYSIS**

**“TH-BANK”**

**Requirements / Analysis**

In the first part of our analysis, we looked through the provided backlog to identify objects that we would be working with in our project.

We identified seven different objects that we could work with:

**Users, loans, accounts, records, transactions. menus and currencies.**

Then we went through them one by one to define what each object needs to be able to do. We decided that records were superflous and currency objects were lower priority.

**User:** There needs to be at least two different user types, Customer and Admin. We also considered adding a “System Owner” and “Chief Economics User”.

Customers need to be able to interact with accounts, transactions and loans.

Admins need to be able to interact with transactions, currencies and users.

They also have several common properties (like id and password), these ones will be put in an abstract User class.

**Menu:** Users interact with the program using a menu.

We decided that there would be one menu object per user, that inherits from an abstract Menu class. This means we can easily add either general functionality, or specific functionality to the menus we have, or add new menus as we add new types of users. What type of menu loads depends on the type of user logging in.

**Accounts:** Customers can have different types of accounts. We will use an abstract account class that all accounts inherit common properties from. Each account will have a unique account number, balance, an owner and a currency.

**Transactions:**  Customers can create transactions between accounts, using unique account numbers to find the correct recipient.

**Loans:** Customers can apply for different types of loans, that are automatically declined or approved depending on the customers current account balance and previous loans. Loans also inherit from an abstract base class with common properties. The applying process needs to be able to interact with users and accounts.

**Currencies:** Needs to be able to interact with accounts, admins, transactions and loans. Each currency inherits from an abstract base Currency class and contains unique exchange rates for each other currency available.

We then identified what types of properties and methods each object needs and wrote them down (**appendix 1**).

**Other requirements**

**Transactions should be executed every 15 minutes:** We decided put the handling of this in a sealed class that runs in the background after it is instantiated. It makes use of the singleton pattern to make sure its only instantiated once. All transactions are sent here and put in a list that is saved to file and emptied every 15 minutes, based on a timer.

**Currency exchange rates should be updated daily by Admin:** This will be handled by a static class that contains exchange rates for all currencies. Transactions and accounts can also use this information when needed.

**Interaction**

We also concluded that the program needs to **save and load information from file** to work properly. We decided to create separate classes (called DataHandlers) to do this. DataHandlers will inherit common properties/methods from interfaces and there will be one for each type of object. We will also create a separate DataHandler for general information (unique id:s etc.)

Different classes can then use these DataHandlers to load information they need, and save information to be used in other parts of the program, if needed.

To further solidify our object-oriented design we will create simple **factory classes** for each base type of object that will create and return the correct subclass when we want it.

We only want the logged in user to be able to interact with their own properties. We created a singleton class called **ActiveUser** (that inherits from abstract User class). This class is instantiated on a successful login with the properties of the logged in user, and can’t be instantiated again unless the user logs out first.

**The Menu objects** will contain most of the other logic of the program. If we have time, we will create facade classes for each type of object that will handle communication with datahandlers and factories. But this is low priority.

We also decided on a common way to input via methods using ConsoleKeyInfo objects. These methods will be placed in a static class for easy access troughout the program.

**Summary**

* We will focus on creating objects of as many parts of the program as possible, and abstract classes for most objects, to have a solid base to stand on if we would be tasked with expanding the scope of the program.
* The program will create data using Factory classes
* The program will load/save/delete data using DataHandler classes
* The program will use the data in the Menu class (or Facade classes)

**Appendix 1: Notes from OOAD planning:**

* Internals
  + Abstract User -
    - ID;
    - Password;
    - IsLoggedIn?
    - Menu;
    - DataHandler object;
    - Abstract UserType;
  + User Customer
    - IsBlocked?;
    - FirstName, LastName,
    - List: Accounts();
    - List: Logs();
  + User Admin
  + Abstract Menu –
    - ShowMenu();
    - MenuTitle;
    - LogOut() / Exit()
    - User Menu
      * Transactions
      * Balance
      * Loans (+ simulator)
      * User Action History
  + Admin Menu
    - Unblock User
    - Add User
    - Change Currency exchange rate
  + Abstract Account
    - Amount;
    - Currency;
    - AccountNumber;
    - OwnerID;
    - AccountType;
    - History;
    - UpdateHistory();
    - ViewHistory();
    - DoTransaction();
    - Withdraw();
  + Account SalaryAccount;
  + Account SavingsAccount;
    - Interest;
    - GetInterest();
  + DataHandler Class
    - Interface
    - Saves to file
    - Loads from file
    - Objects can use class to retrieve data / info
  + Transaction Class
    - Sum
    - Sender Account (currency)
    - Receiver Account (currency?)
    - DateTime
    - Typ
  + Log Class
    - DateTime
    - All info from Transaction class
    - LogDataHandler
  + Loan
    - Sum
    - RepaymentPeriod
    - Owner
    - Interest
    - StartDate
    - MonthlyPayment
    - LeftToPay
  + (Currency)
    - Lista med exchange rates?
    - Lista med currencies?
    - CurrencyDataHandler?
  + (Login)
    - Metod i Program class
* Design
  + The Menu class is the home of the logic facing the user. The different methods in this class calls other classes to load/save/create/delete data.