**Banking System Programmer’s Manual**

Overview

This manual is designed to give software engineers a solid foundation on the structure and functionality of the banking system program. You do not need to read it all the way through and may use it merely as a reference if desired. There are 4 main sections to the manual, “Design Diagrams”, “Class Diagrams”, “Architecture Diagram”, and “Functions and Main Drivers”.

Before diving into the specifics of the architecture, it is important to know some basic details about the program. First, the program has a very simple tech stack. It utilizes Kotlin for the frontend interface and Java for the backend functionality. There is also a very simple method for utilizing a database. The method is to simply use CRUD operations on “records” recognized in .csv files. Each .csv file can be viewed as if it were a table in an SQL database. This allows us to not actually have to run any “server” for the database and instead, run the program on an as needed basis before saving the information to .csv files and ultimately shutting down until the next use.

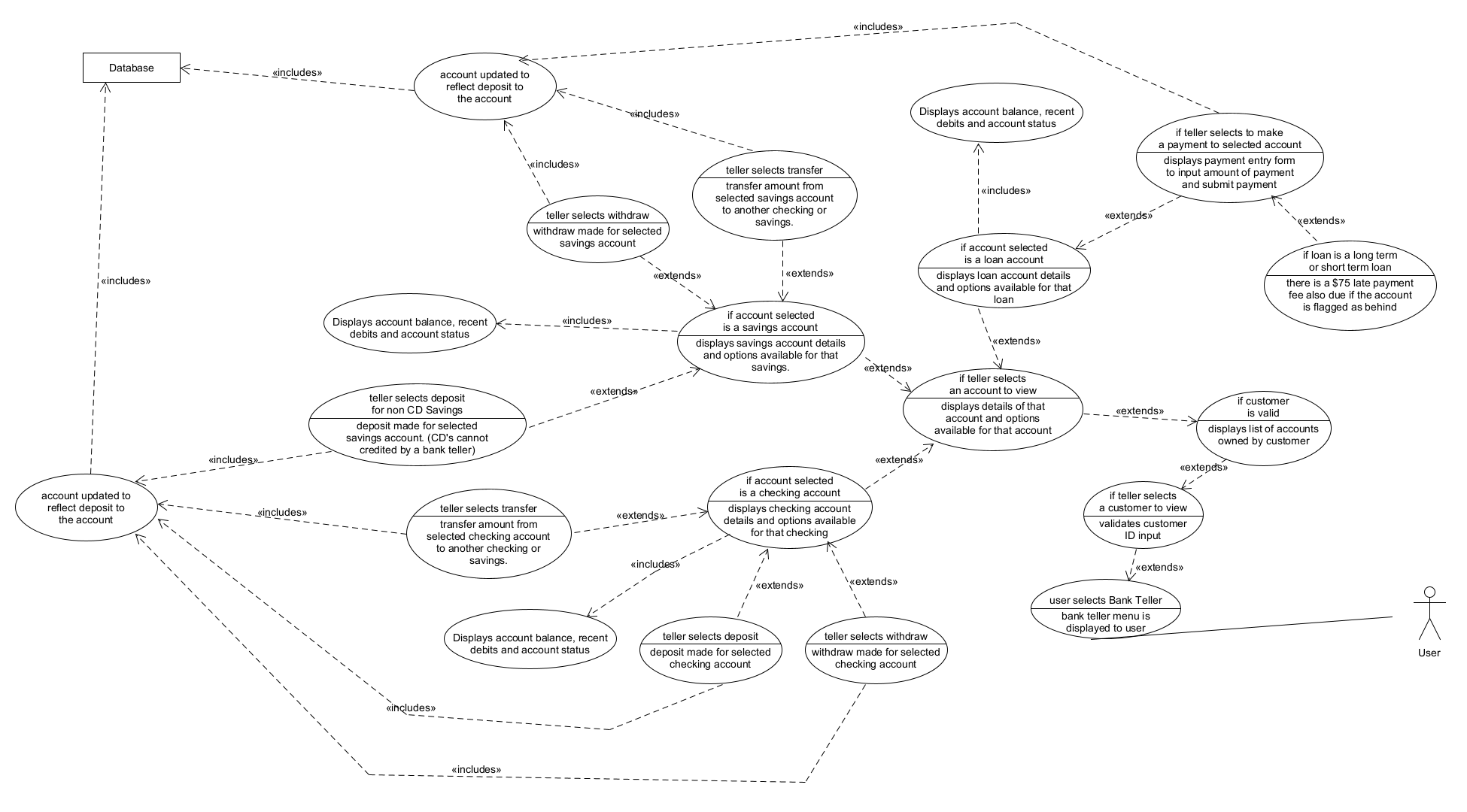
Lastly, it would be helpful to have an understanding of the basic operations and initial interfaces of the program and the purpose from a user perspective. The purpose of this program is to mock a banking system application that maintains user accounts and transactions at a bank. The application is built for 3 types of users, a customer, bank teller, and bank manager. The customer can manage their account with limited privileges, the bank teller can lookup accounts and make adjustments, and the bank manager can manage customer accounts as well as execute administrative tasks. Hopefully this overview will give enough context to adequately understand the following information.

Design Diagrams

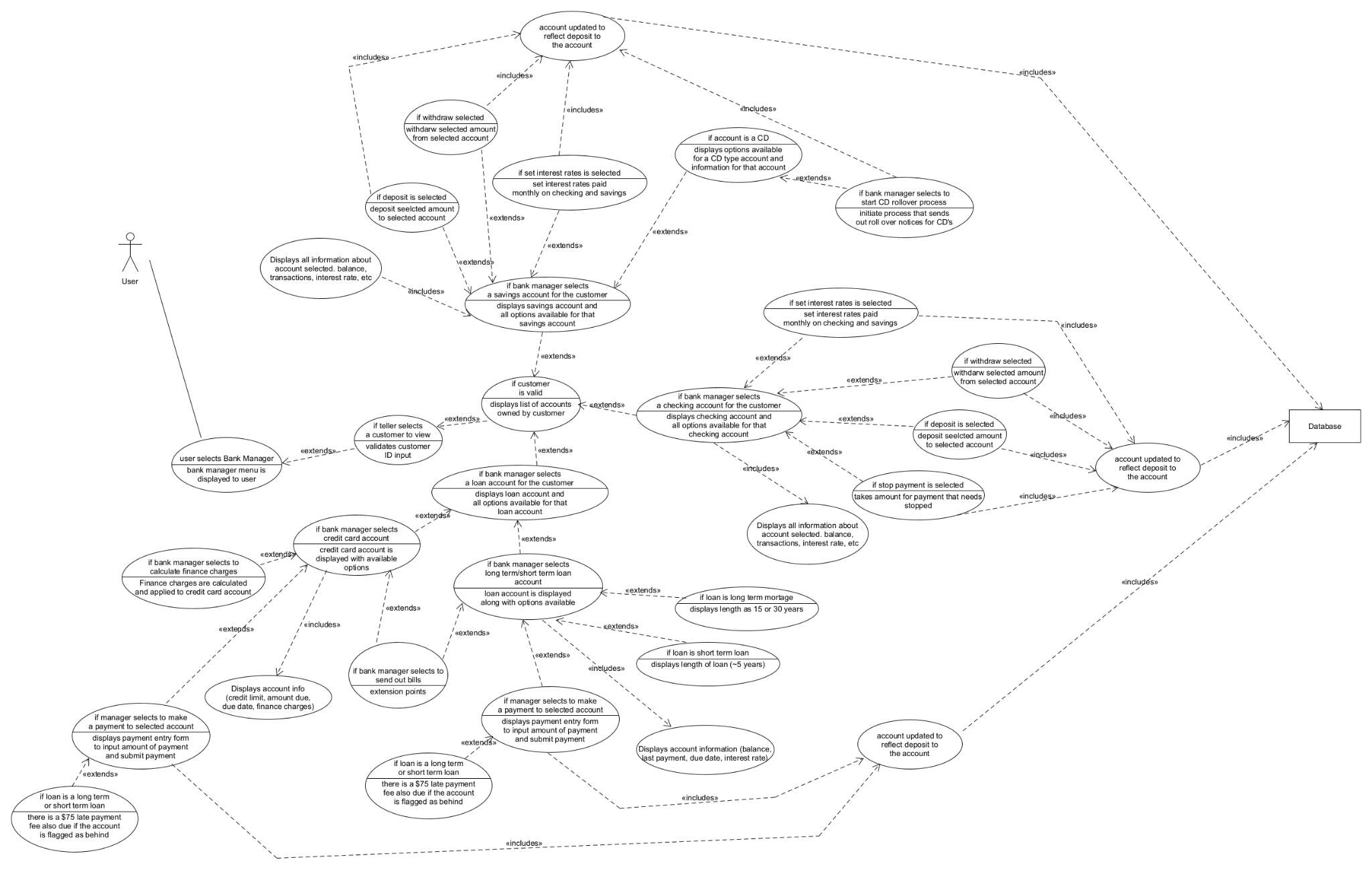
Below are 3 design diagrams that may be utilized to conceptualize the process of the interfaces and functionality of the program. First there is a use case diagram, then an activity diagram, and lastly a set of sequence diagrams detailing both the user repositories as well as the user data access objects.

*Use Case Diagram*

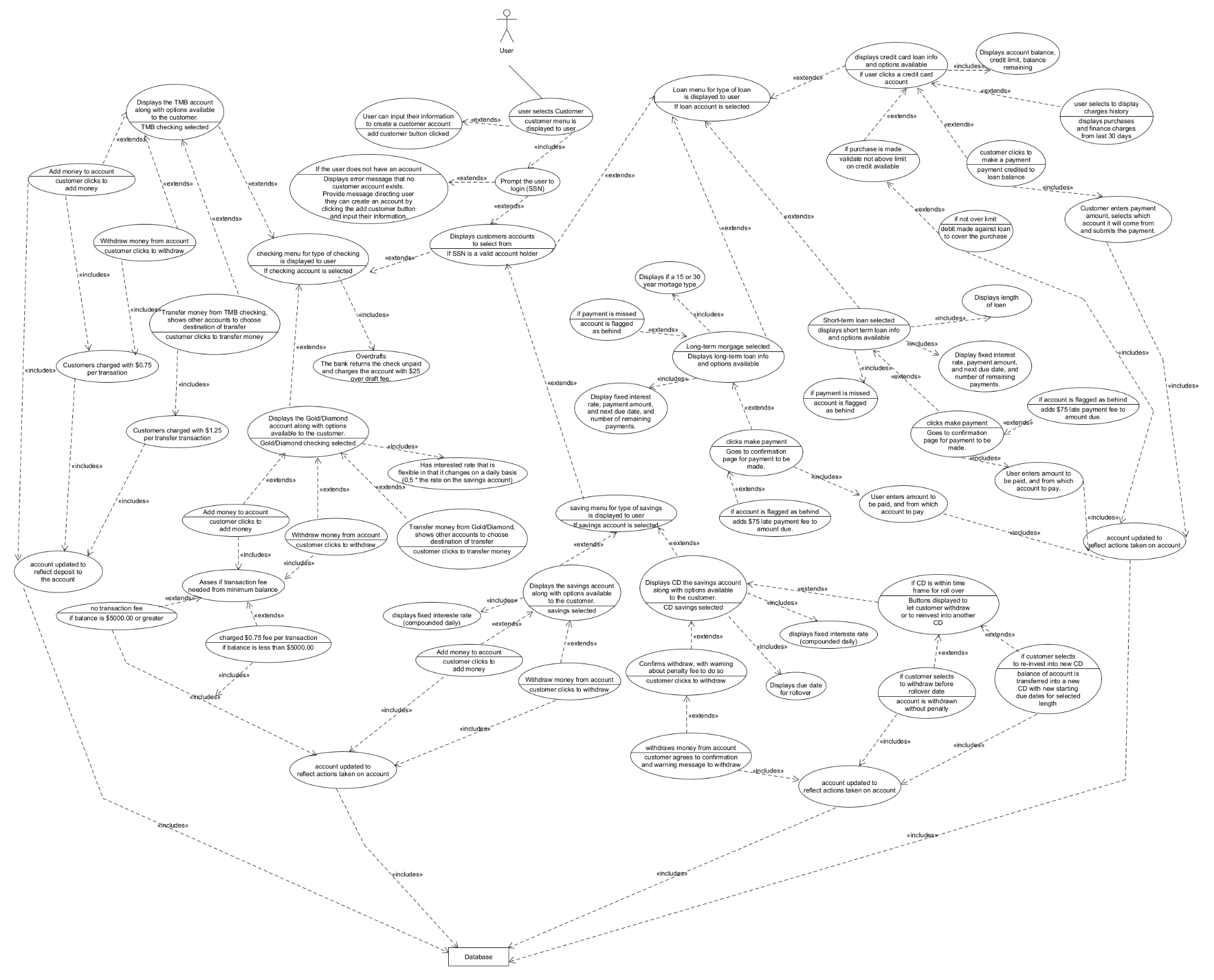
*Bank Teller*



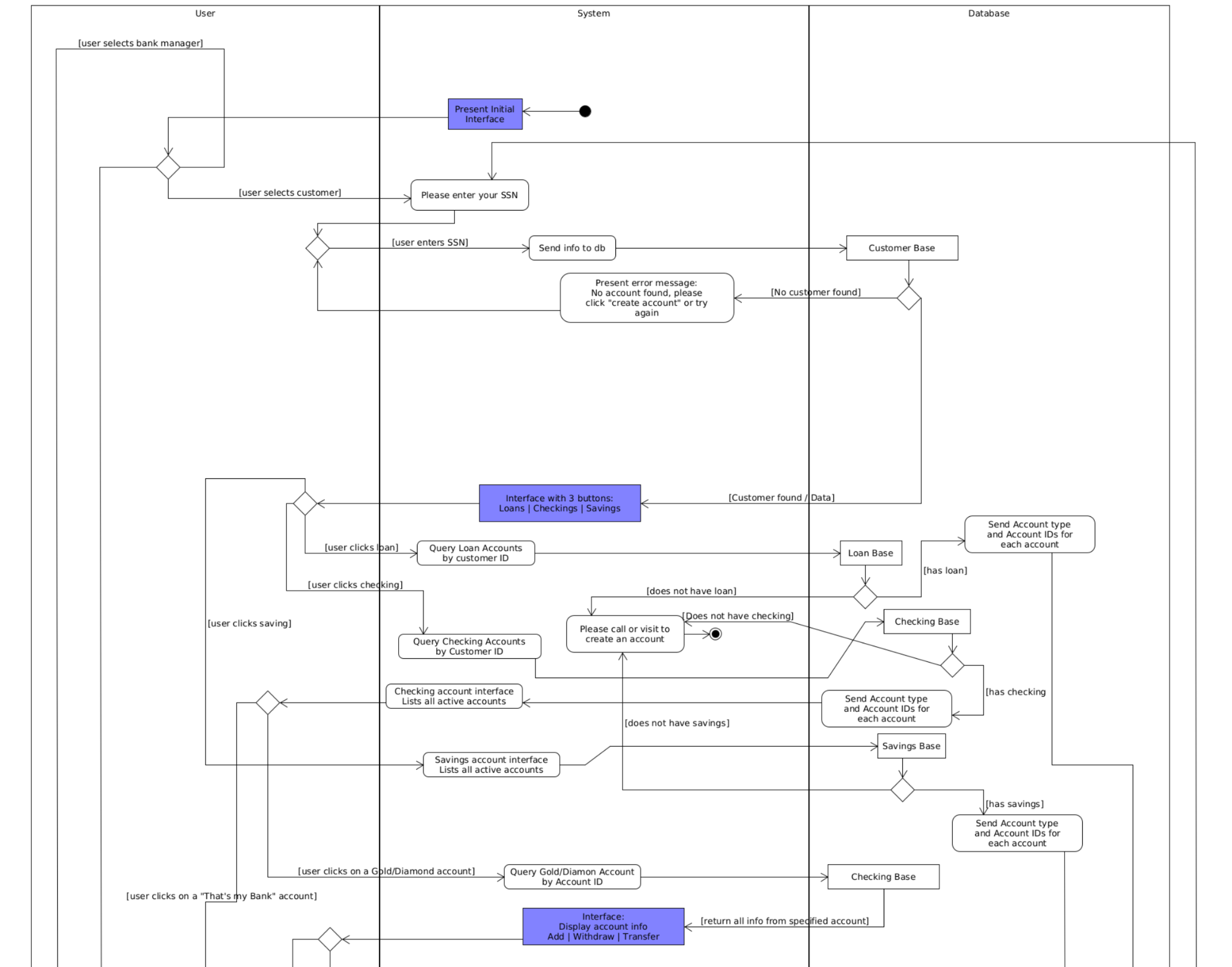
*Bank Manager*

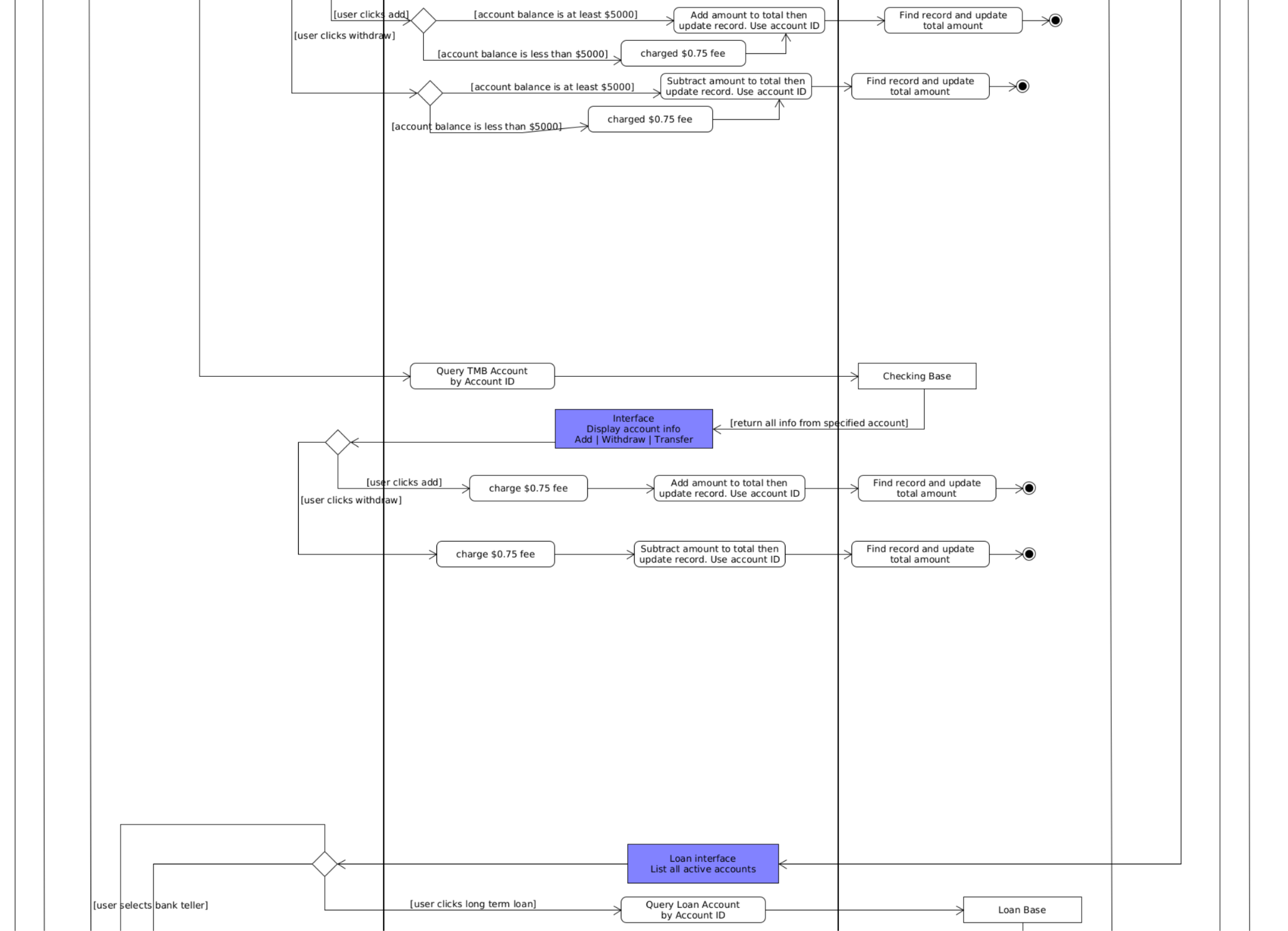
**

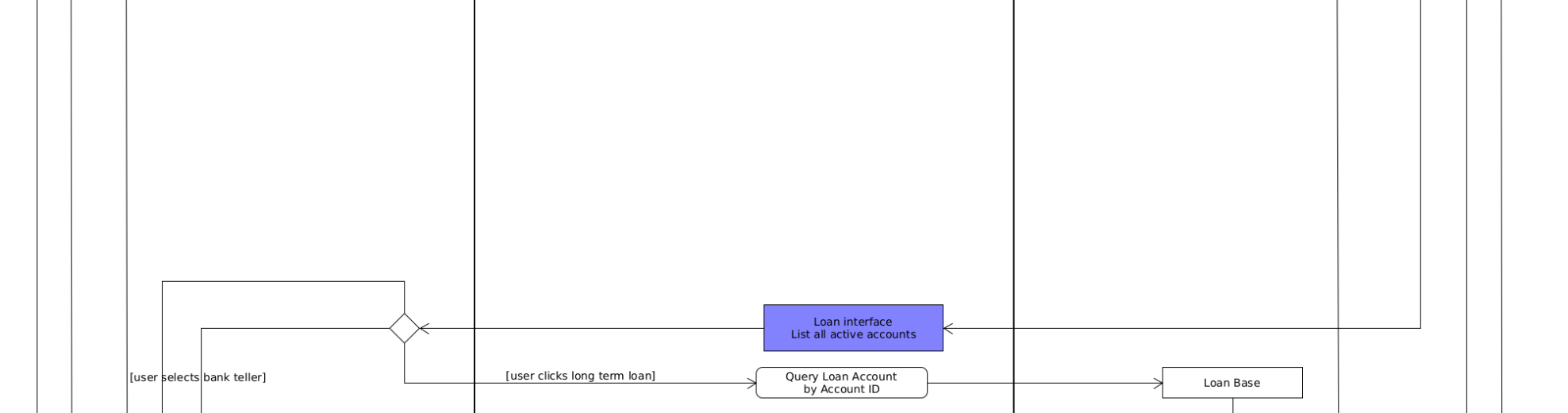
*Customer*

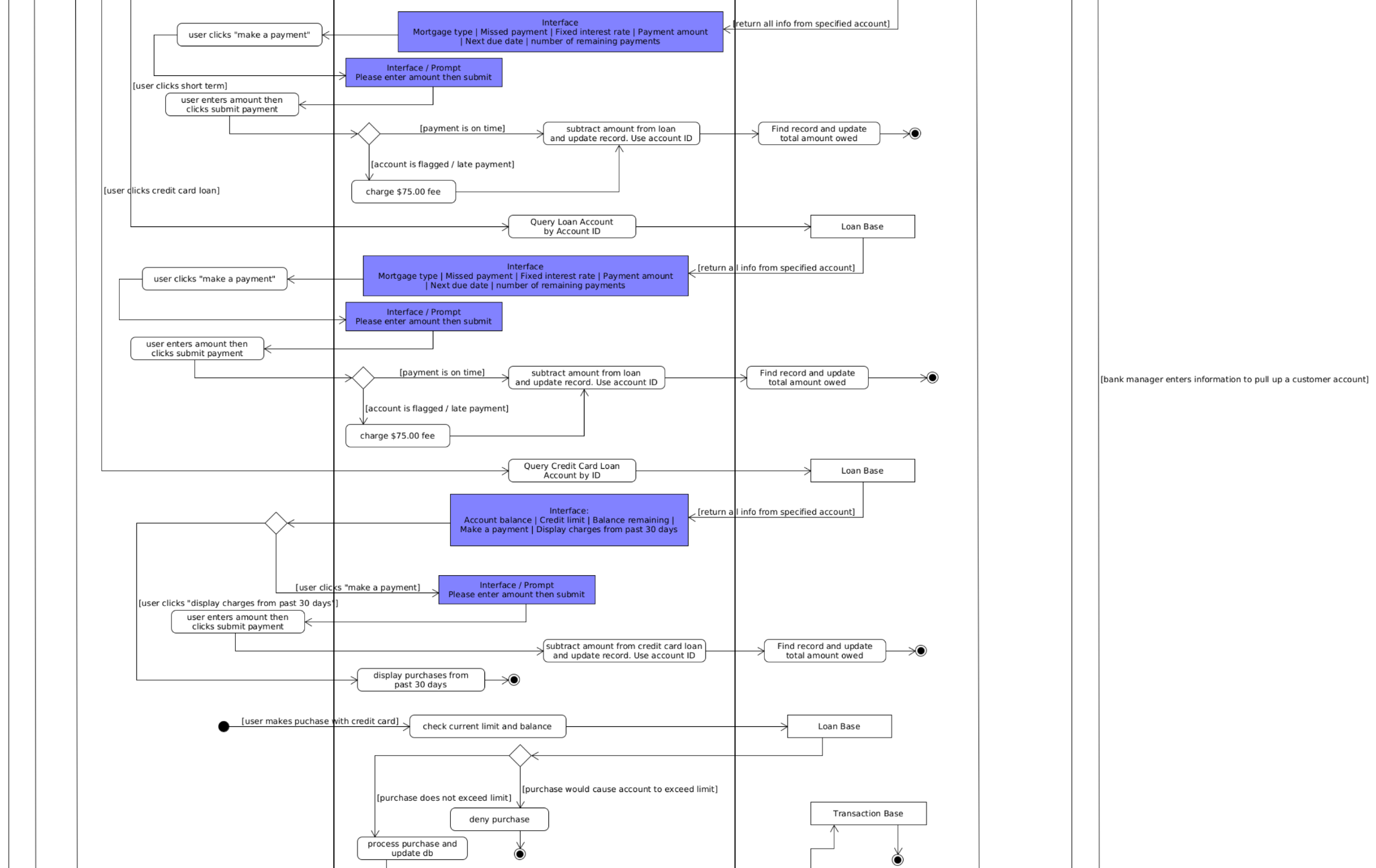
**

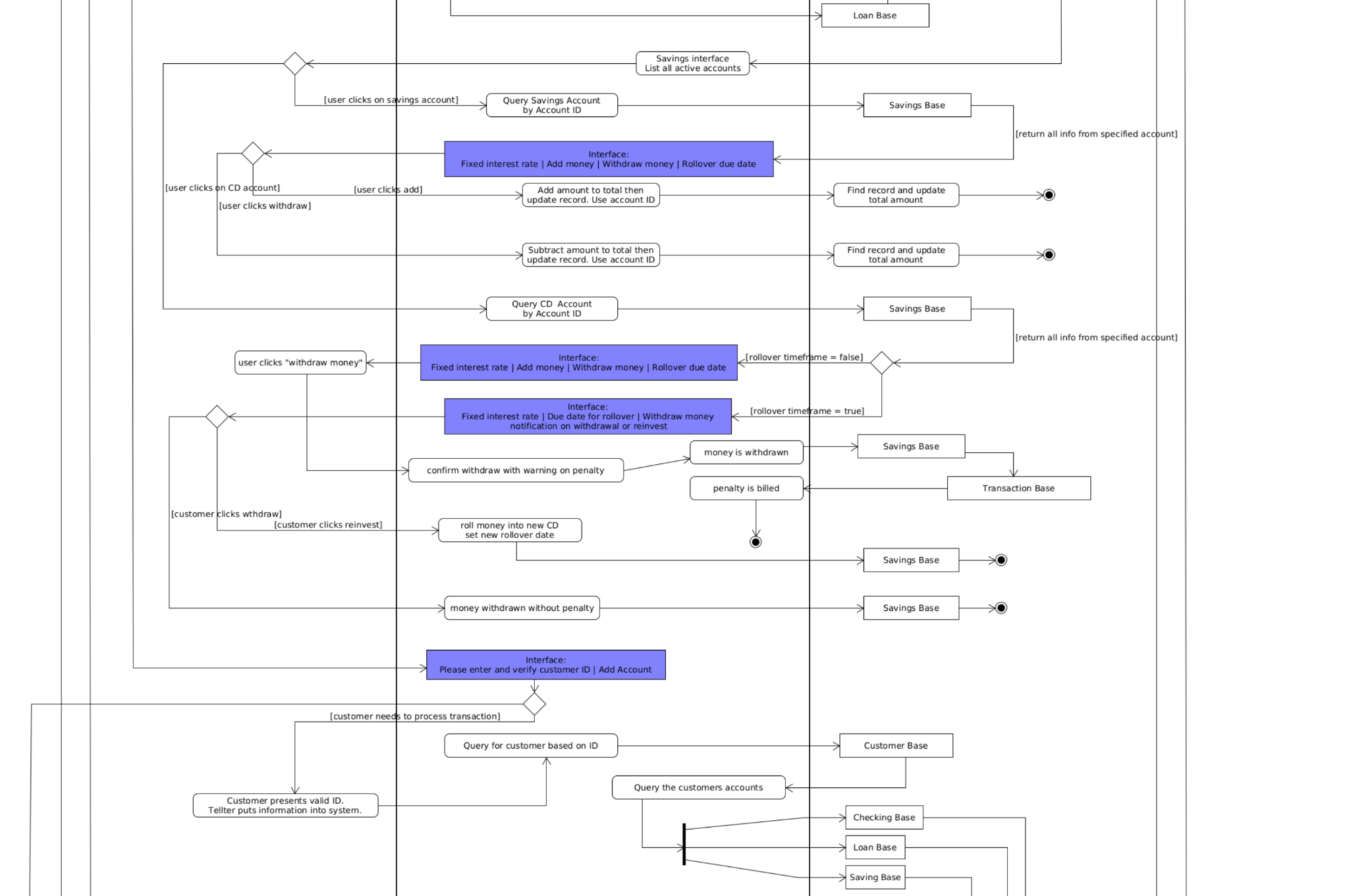
*Activity Diagram*

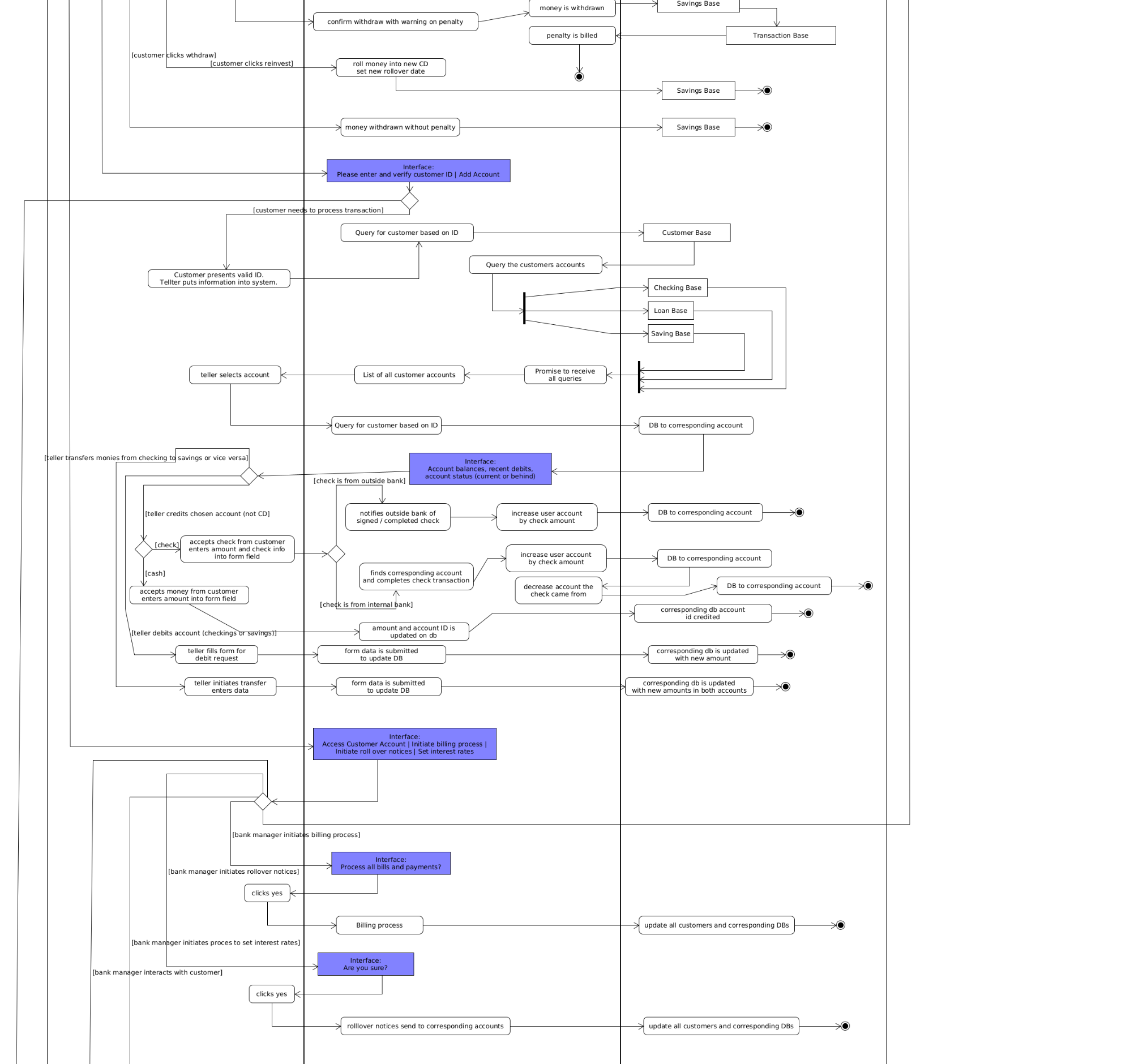
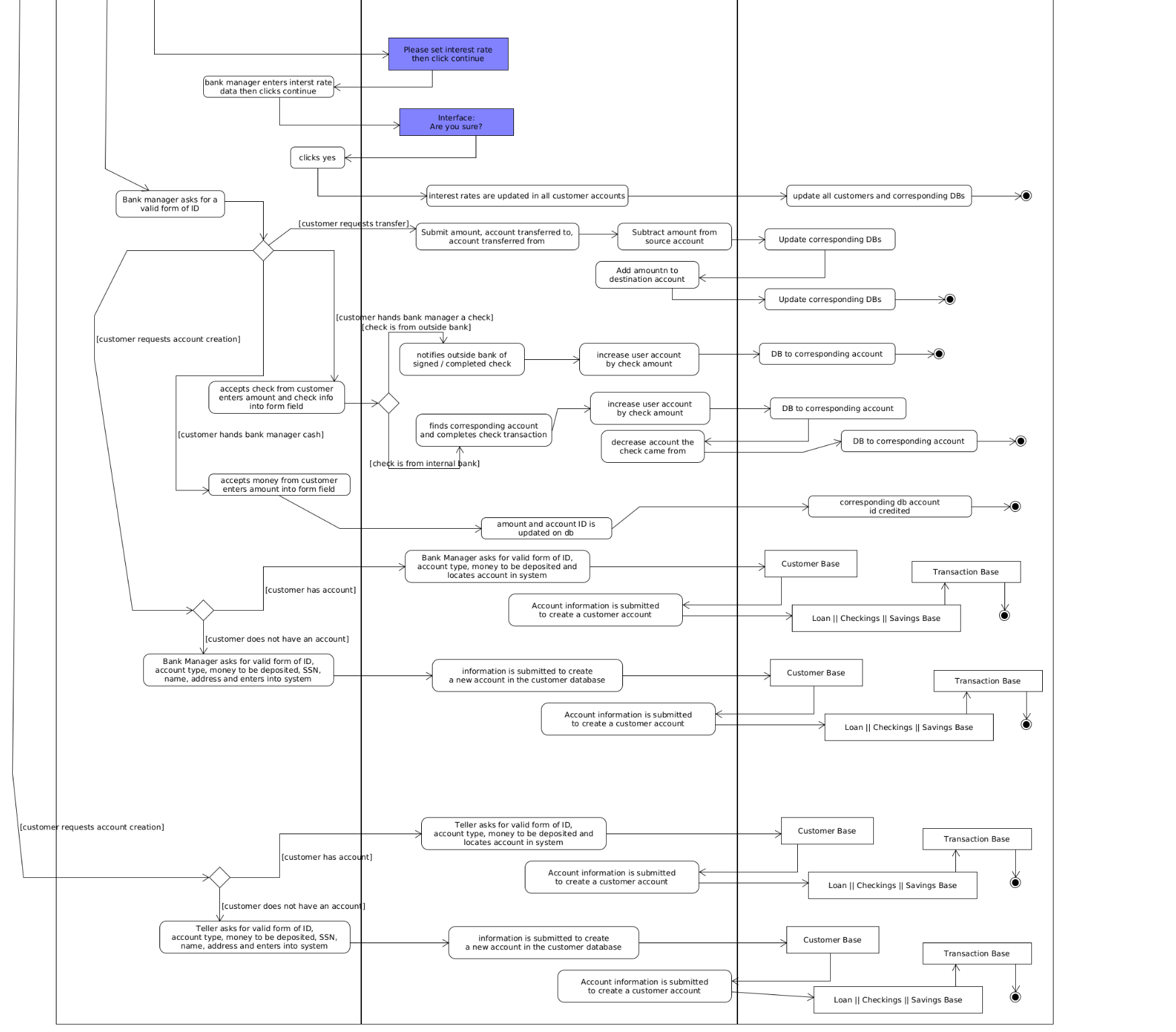
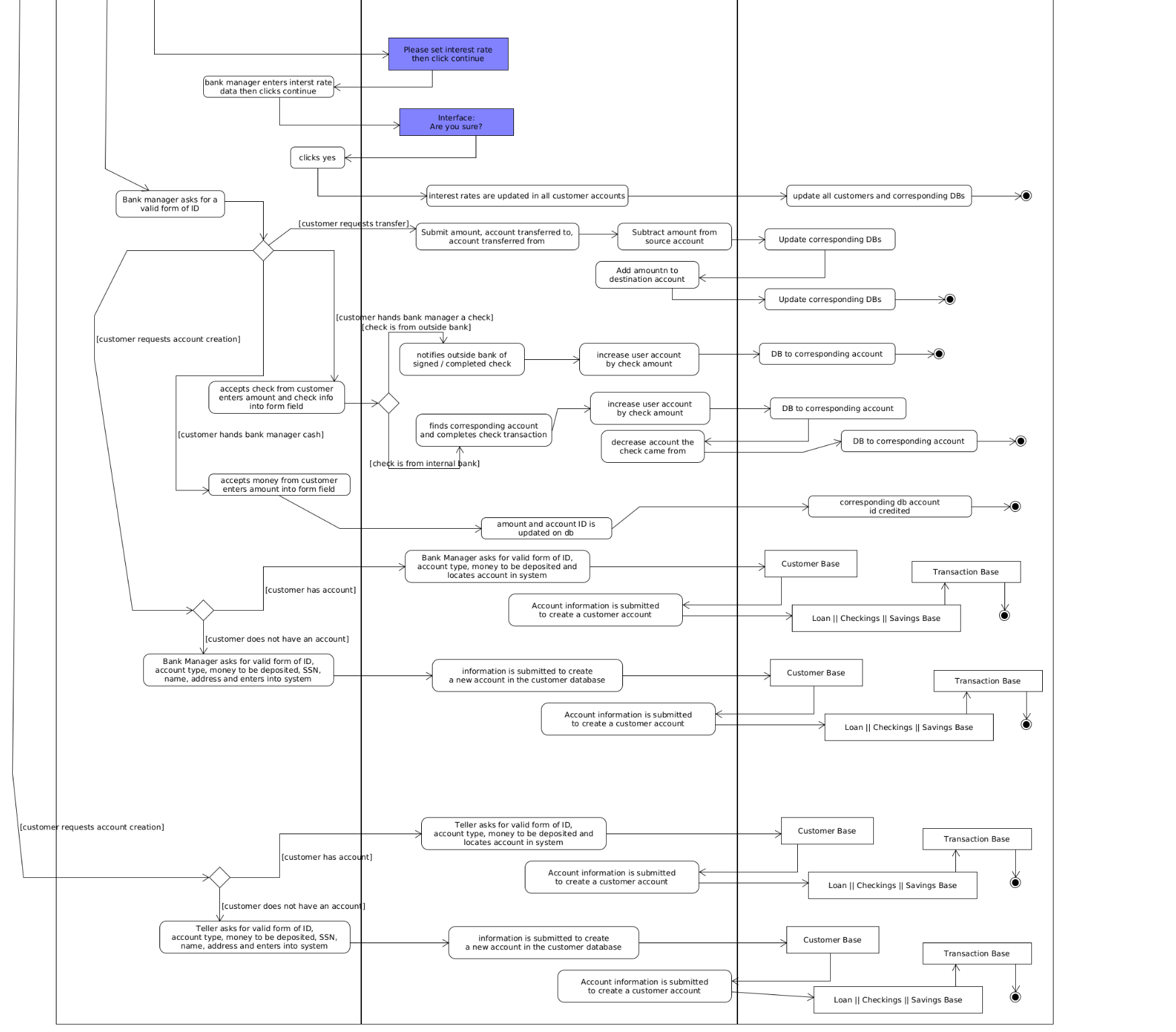


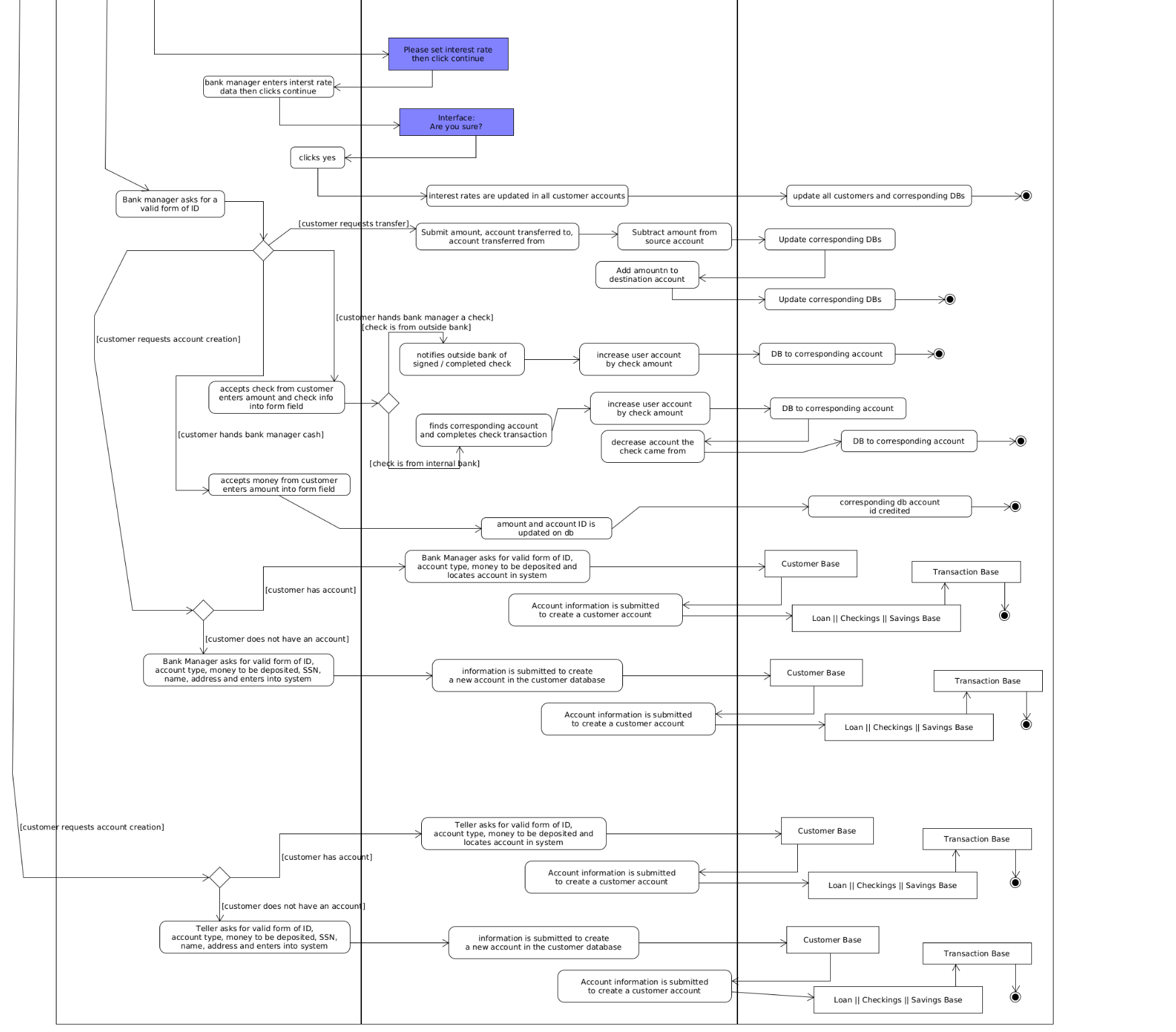






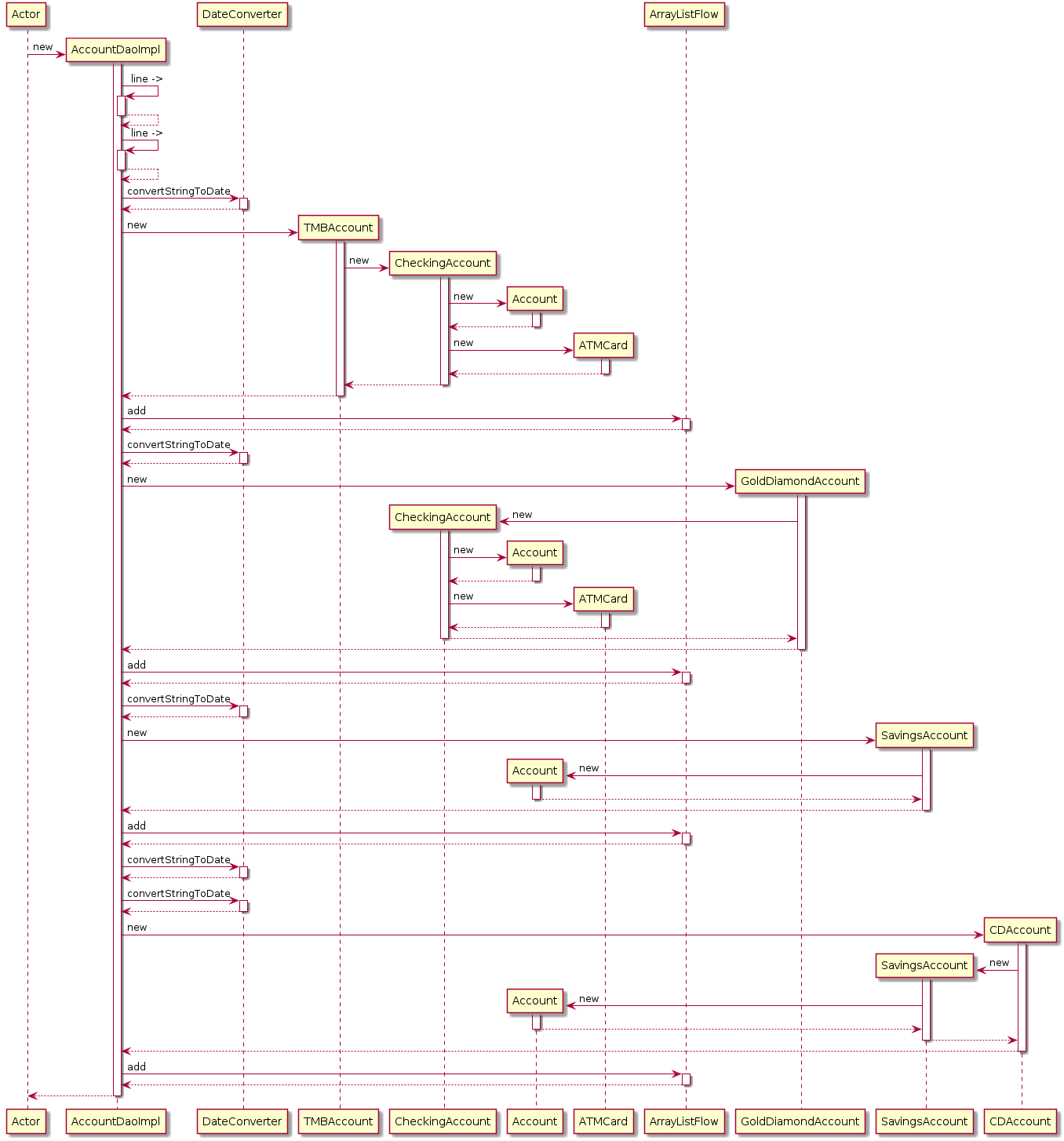




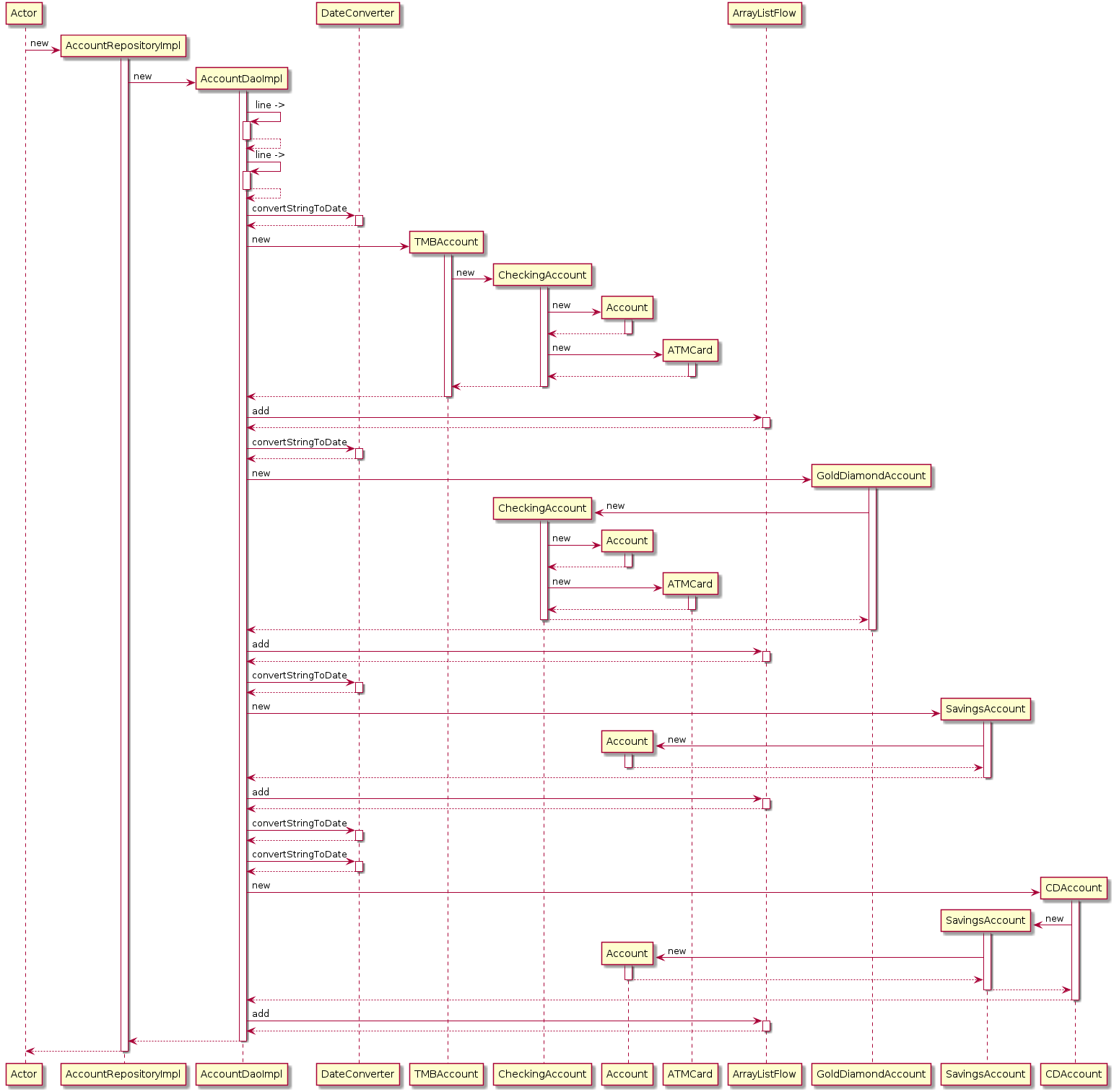


*Sequence Diagram*

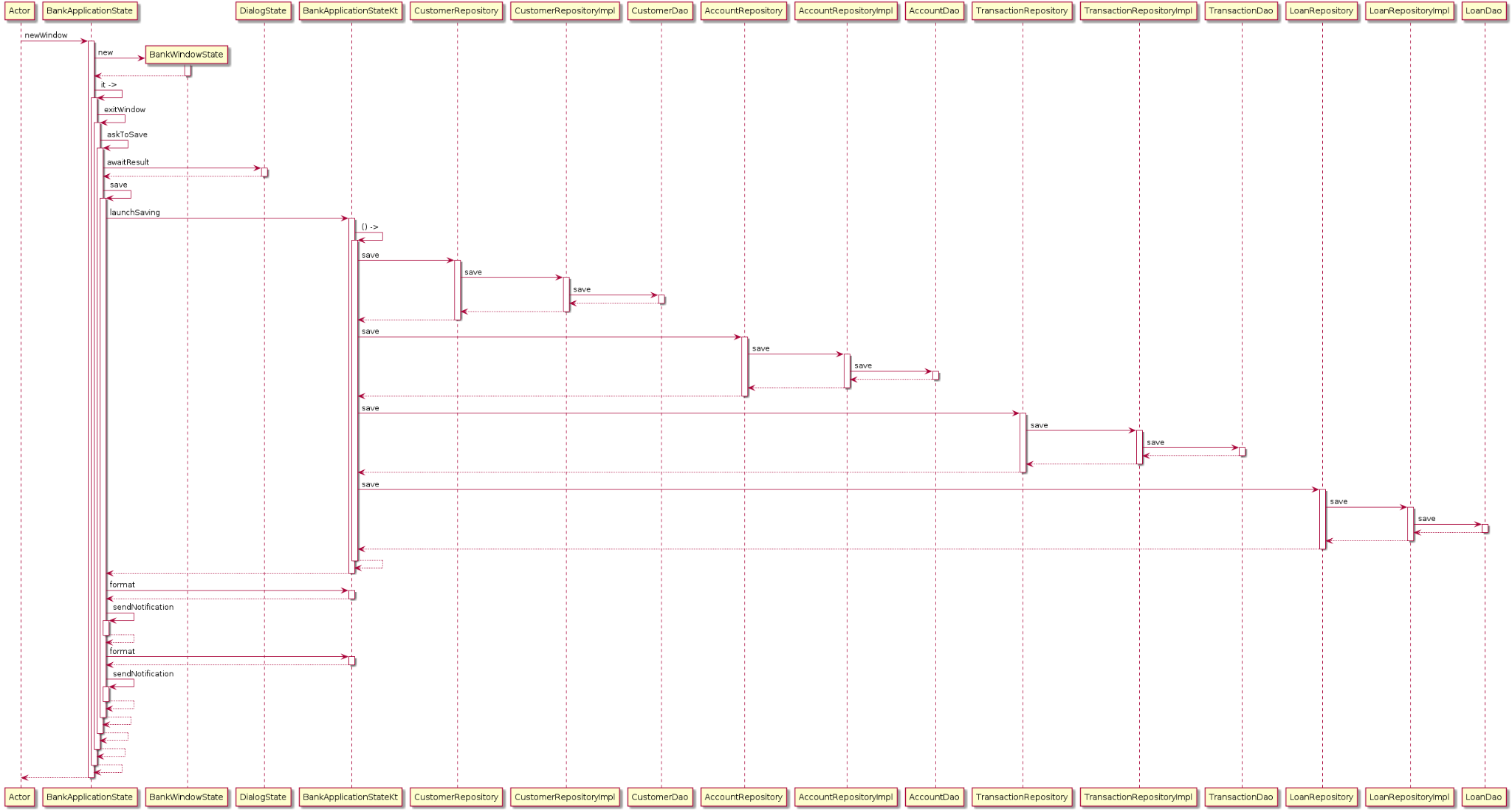
Account Dao Implementation

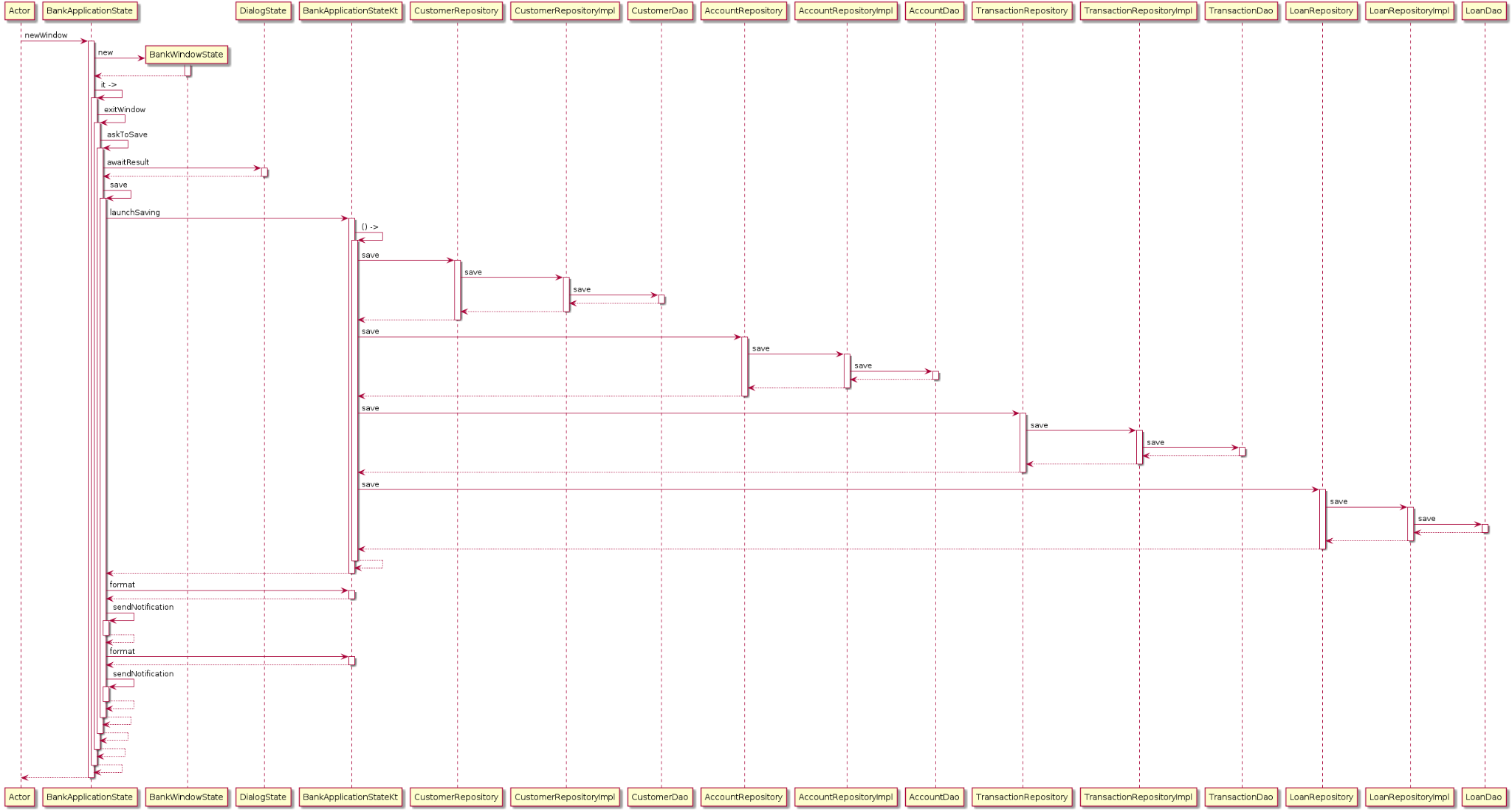


Account Repository Implementation



Bank Application State

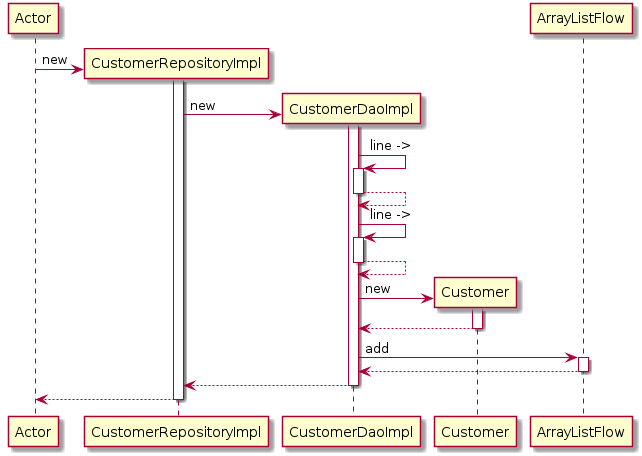




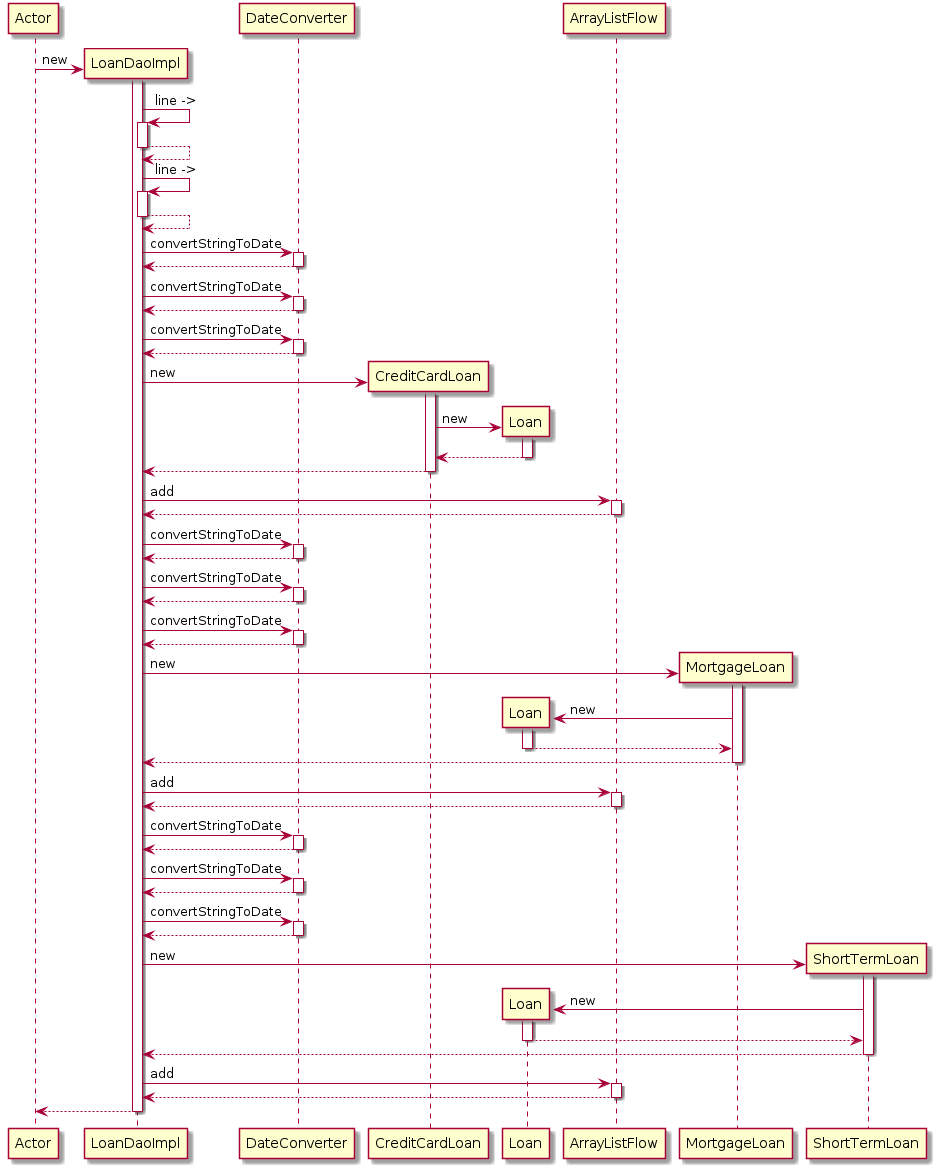
Customer Dao Implementation



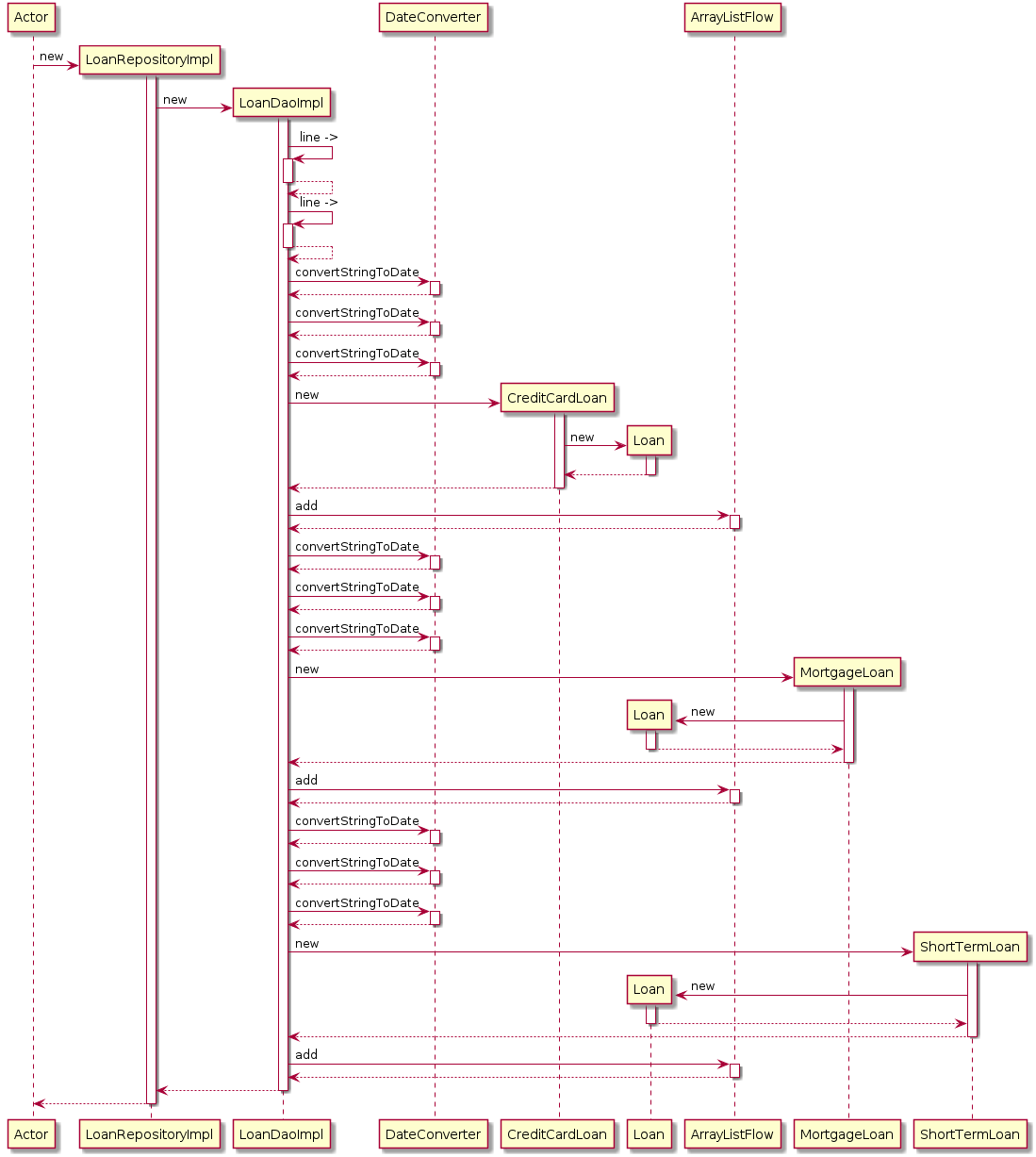
Customer Repository Implementation



Loan Dao Implementation



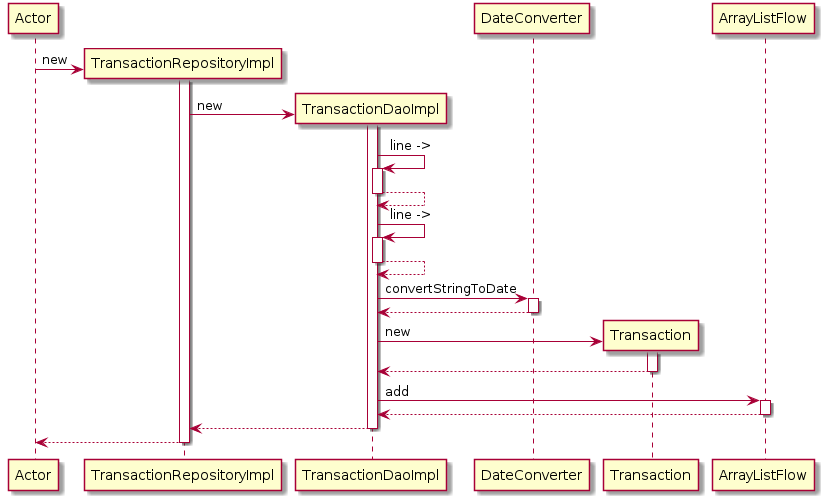
Loan Repository Implementation



Transaction Dao Implementation

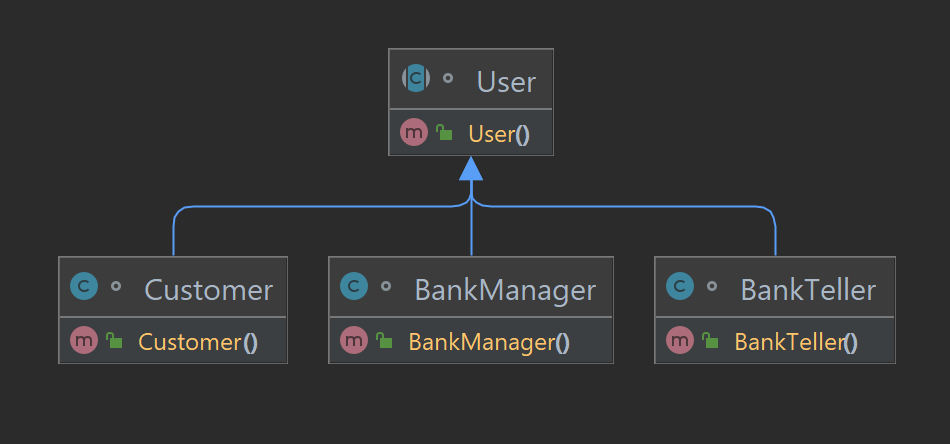


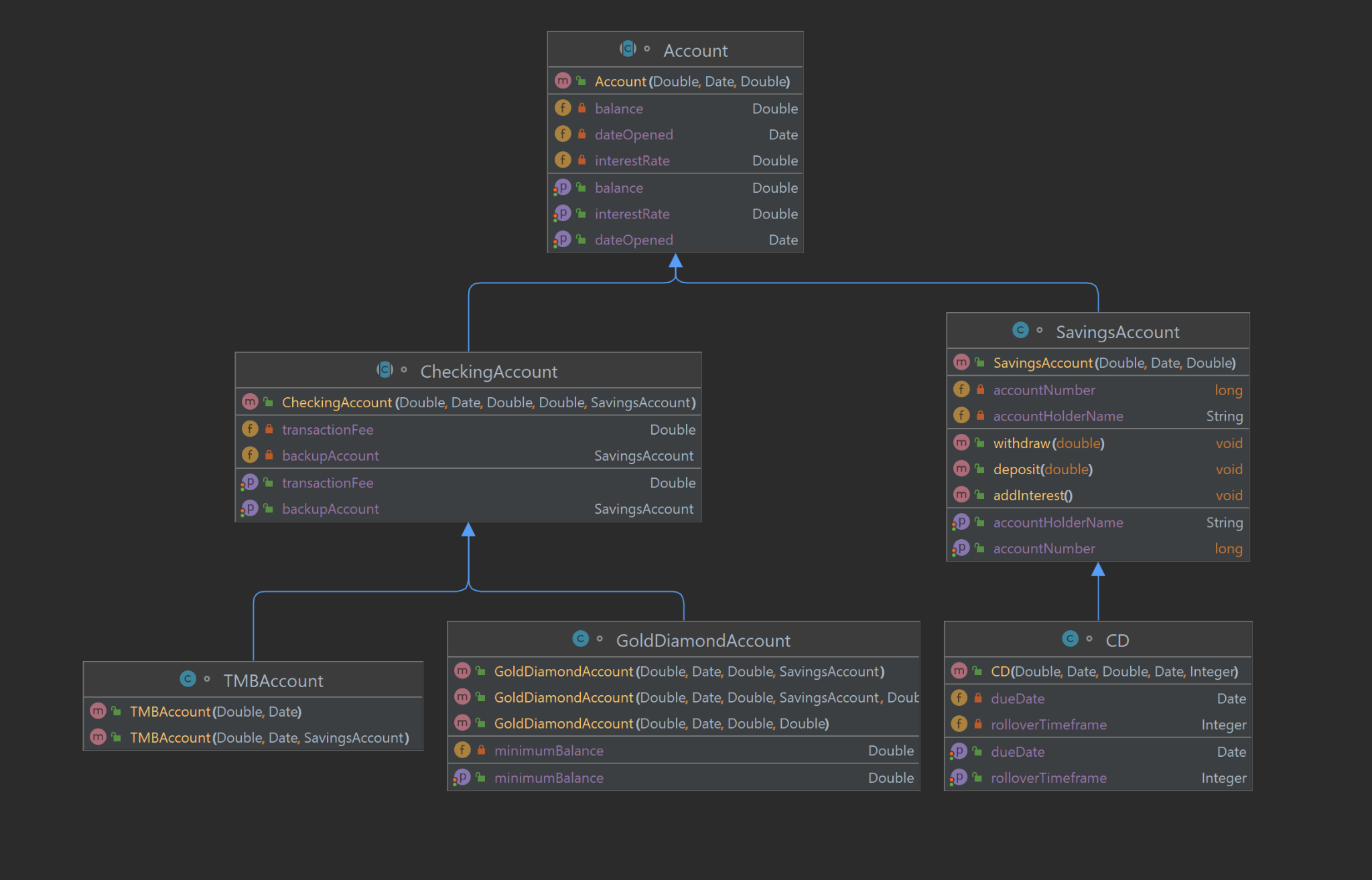
Transaction Repository Implementation

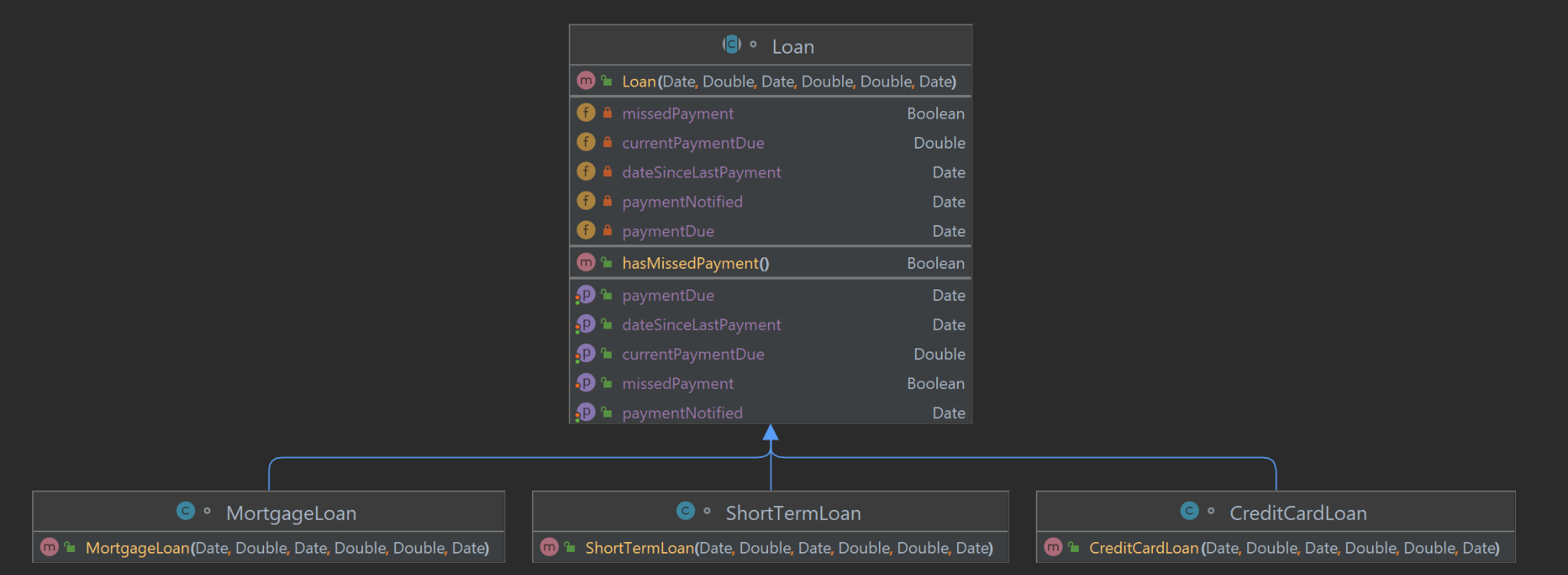
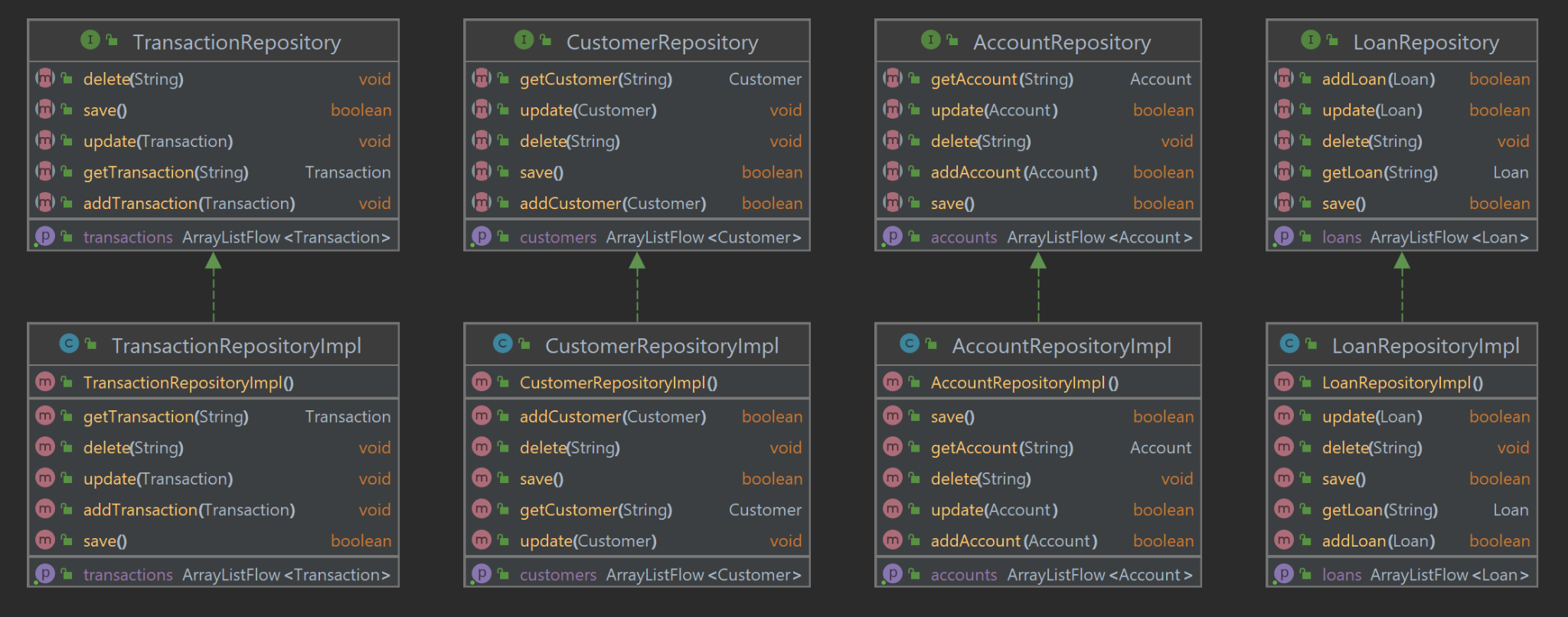


Class Diagrams

Below we have the class diagrams. Here you can easily visualize the polymorphism taking place with the abstract classes User, Account and Loan at the top and then more specific classes catered to more specific applications as you follow down the various charts. The last set of class diagrams describes the relationship between the data access objects (DAOs) and the repositories.

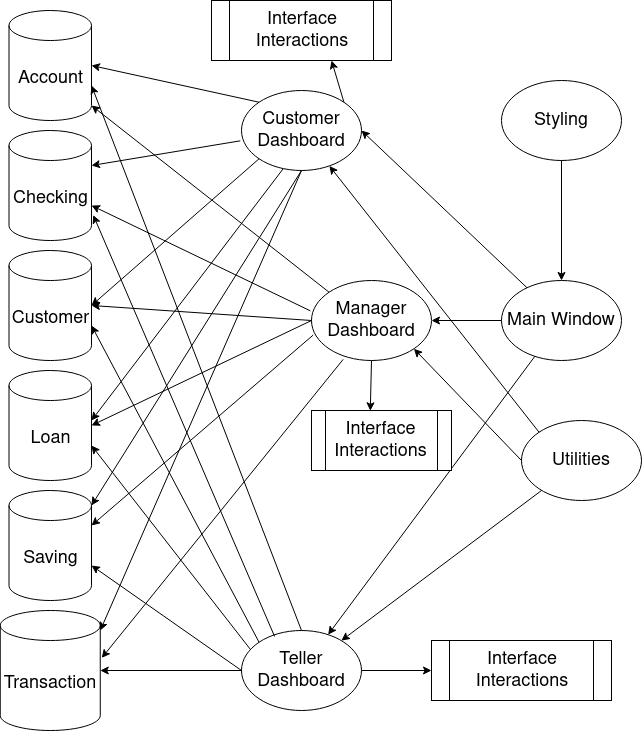






Architecture Diagram

The purpose of the architecture diagram is to provide a solid overall view of the functioning and flow of the system. Although many intermediate functions are implied, the larger overarching functions are described visually.



Functions and Main Drivers