Zach Evans – Team Leader

Ryan A James - Co-Programmer

Trey Scarborough - Co-Programmer

Zach Czarnecki – Documentation Lead

CSC-406 Banking system final

Team 2

Systems Table of Contents

**Requirements Document4**

**Use Case Diagram**

Open/Close Account6

Savings Account6

Checking Account7

Loan Account8

ATM8

**Activity Diagram**

Open/Close Account9

Savings Account10

Checking Account11

Loan Account12

ATM13

**Sequence Diagram**

Open/Close Account14

Checking/Savings Deposit and Withdraw Transaction15

Overdraft Charge16

Open Loan/Credit Card17

Late Payment17

Close CD18

ATM19

**Class Diagram20**

**Users Manual**

Log-In18

Open/Close Account18

Withdraw and Deposit19

Overdraft Charge13

Open Loan/Credit Card14

Late Payment14

Close CD15

ATM22

**System Manual**

Open/Close Account6

Savings Account7

Checking Account8

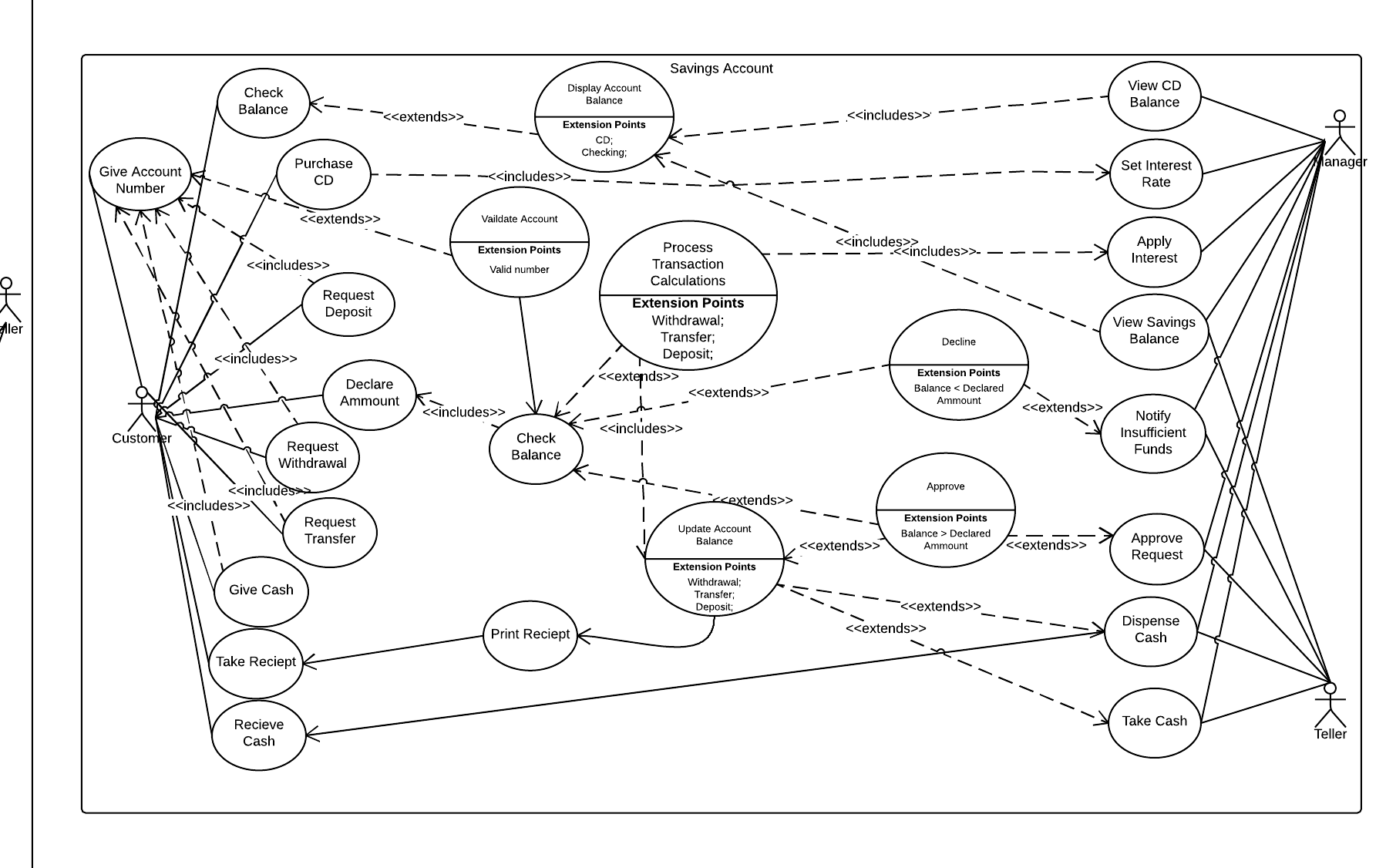
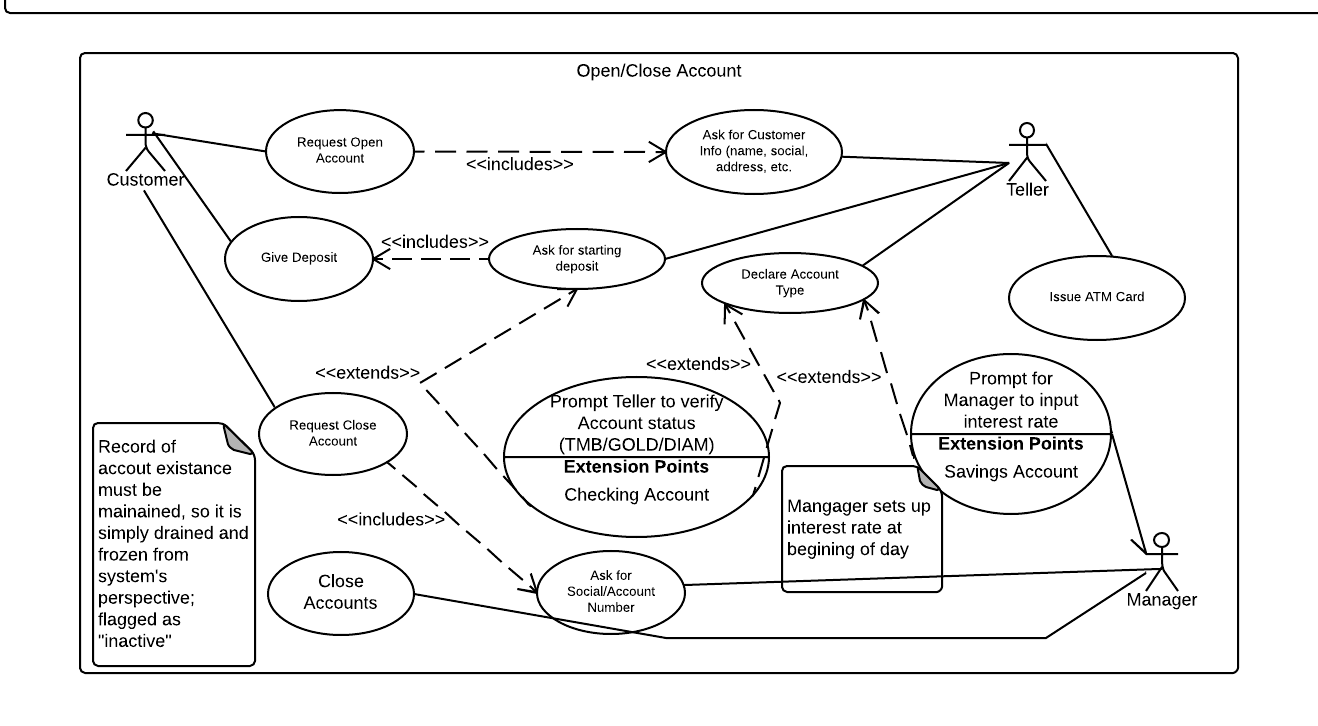
Loan Account9

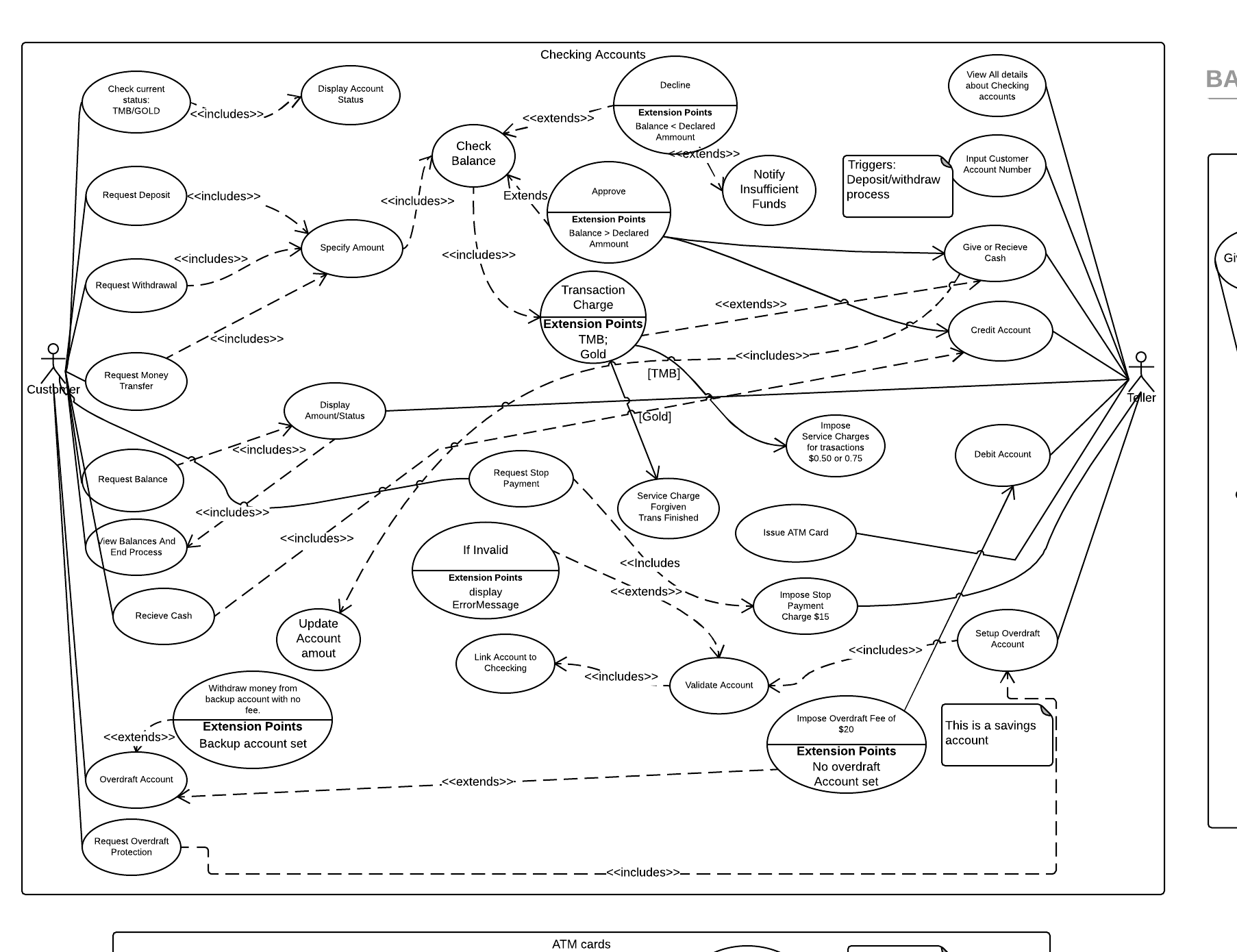
ATM10

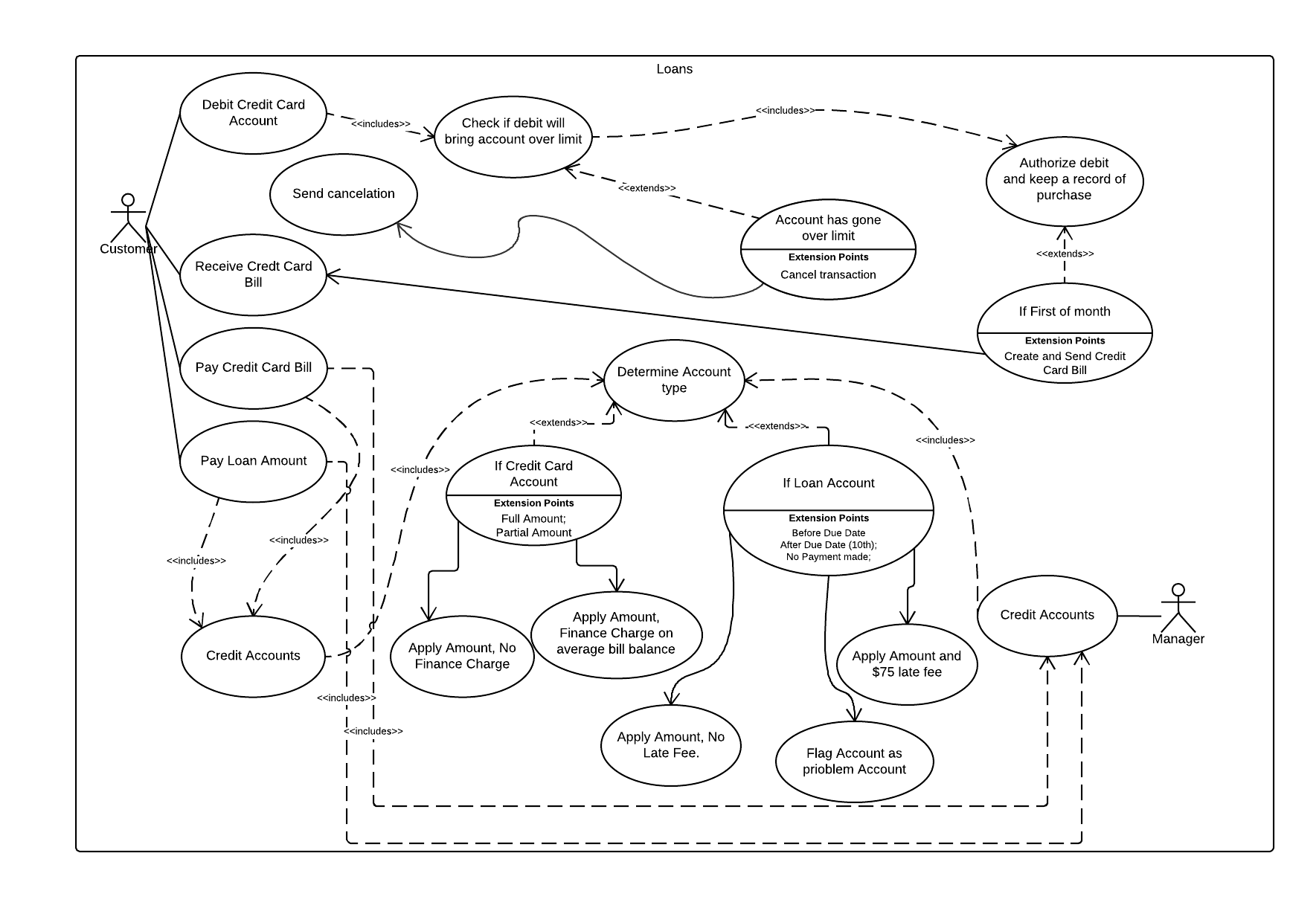
**Usability:**

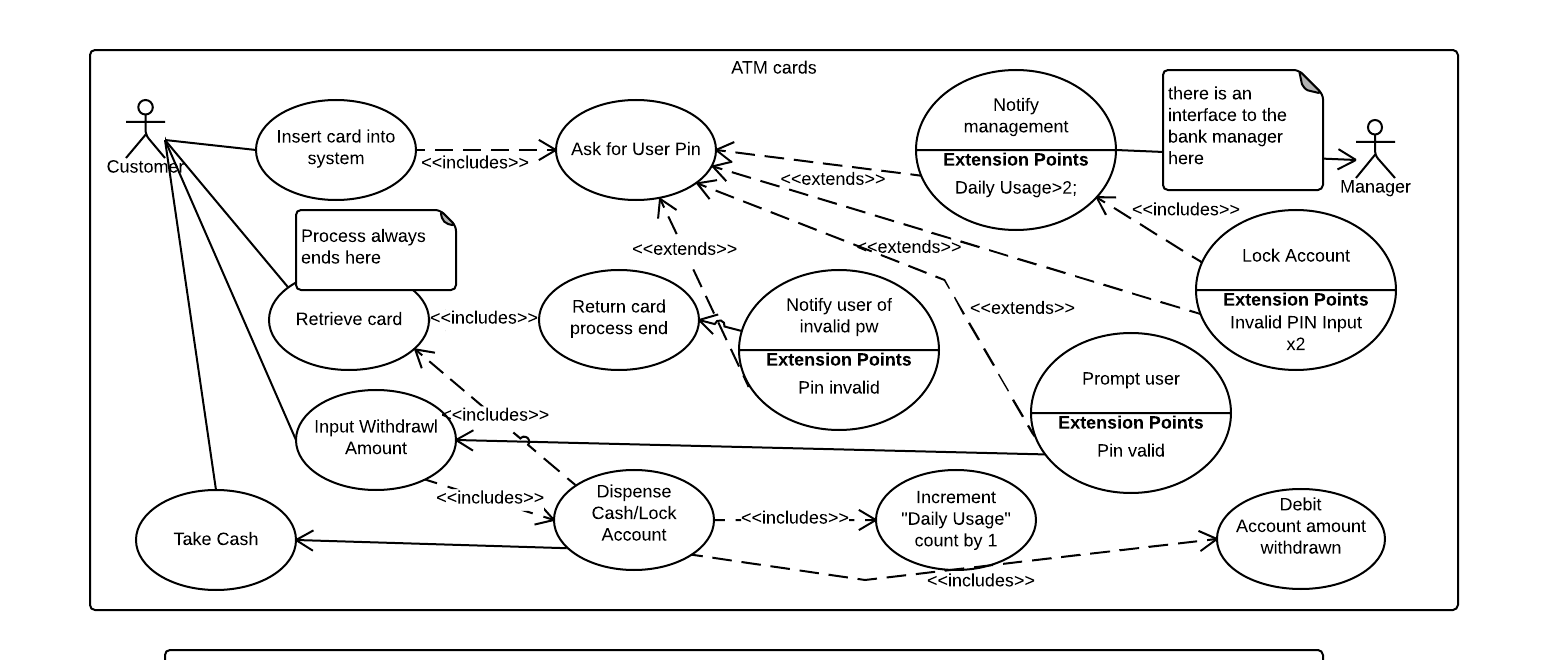
* The Customer shall have access to ATM interface, paper deposits, and/or use of credit cards to make withdrawals and deposits
* Tellers shall have access account balances, recent debits, accounts status (current or behind), loans, withdrawal, deposit, and other account operations such as opening and closing of accounts. Tellers can credit any account they have access to and transfer money between them.
* Managers shall have access to all the same systems and functions as Tellers, in addition to setting up and managing Loans and CDs, including interest rates on payments. They shall send bills each month out to credit and mortgage accounts. They shall also process rollover notices for CDs and set the interest rates paid monthly on checking and savings accounts.

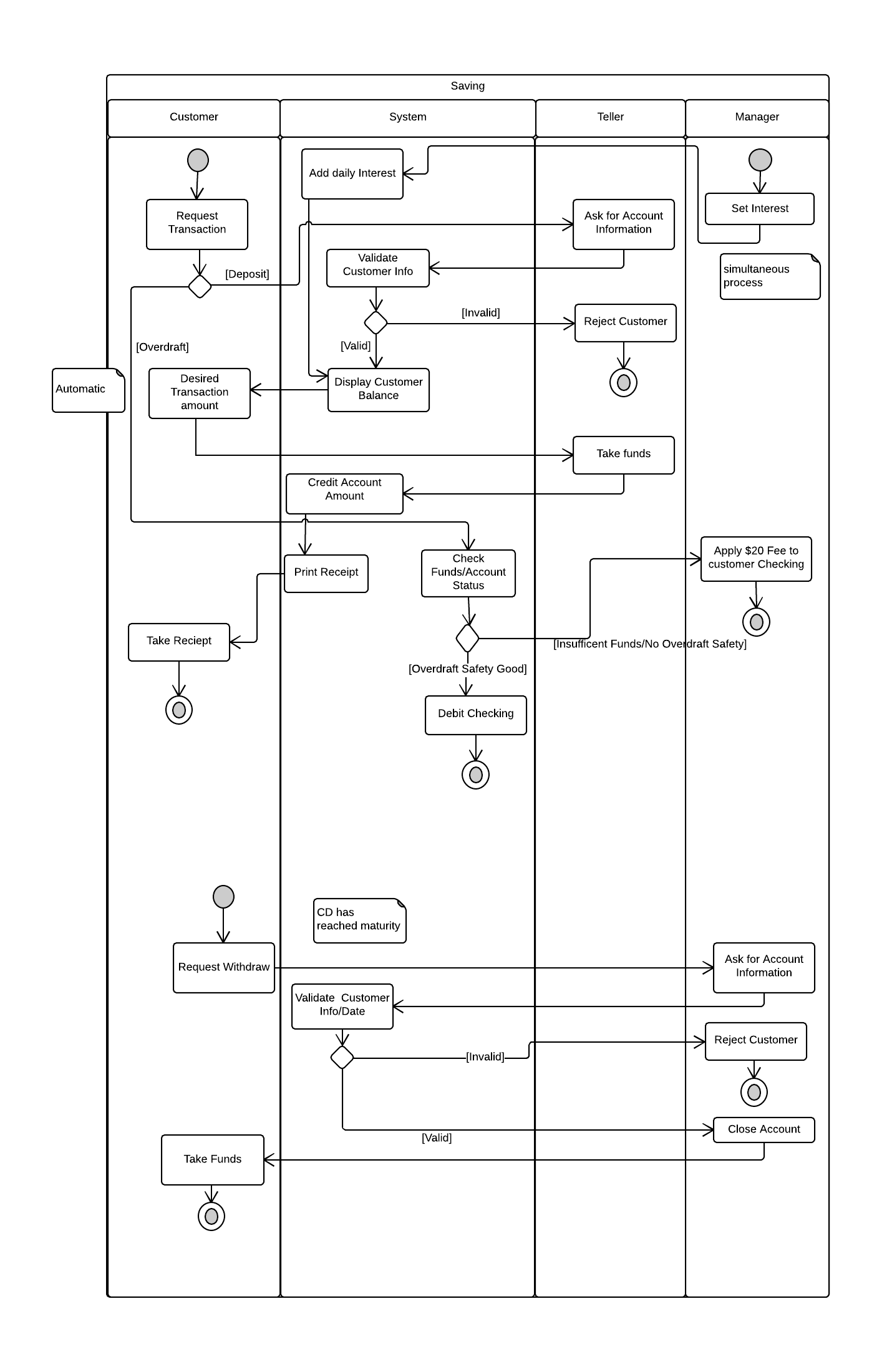
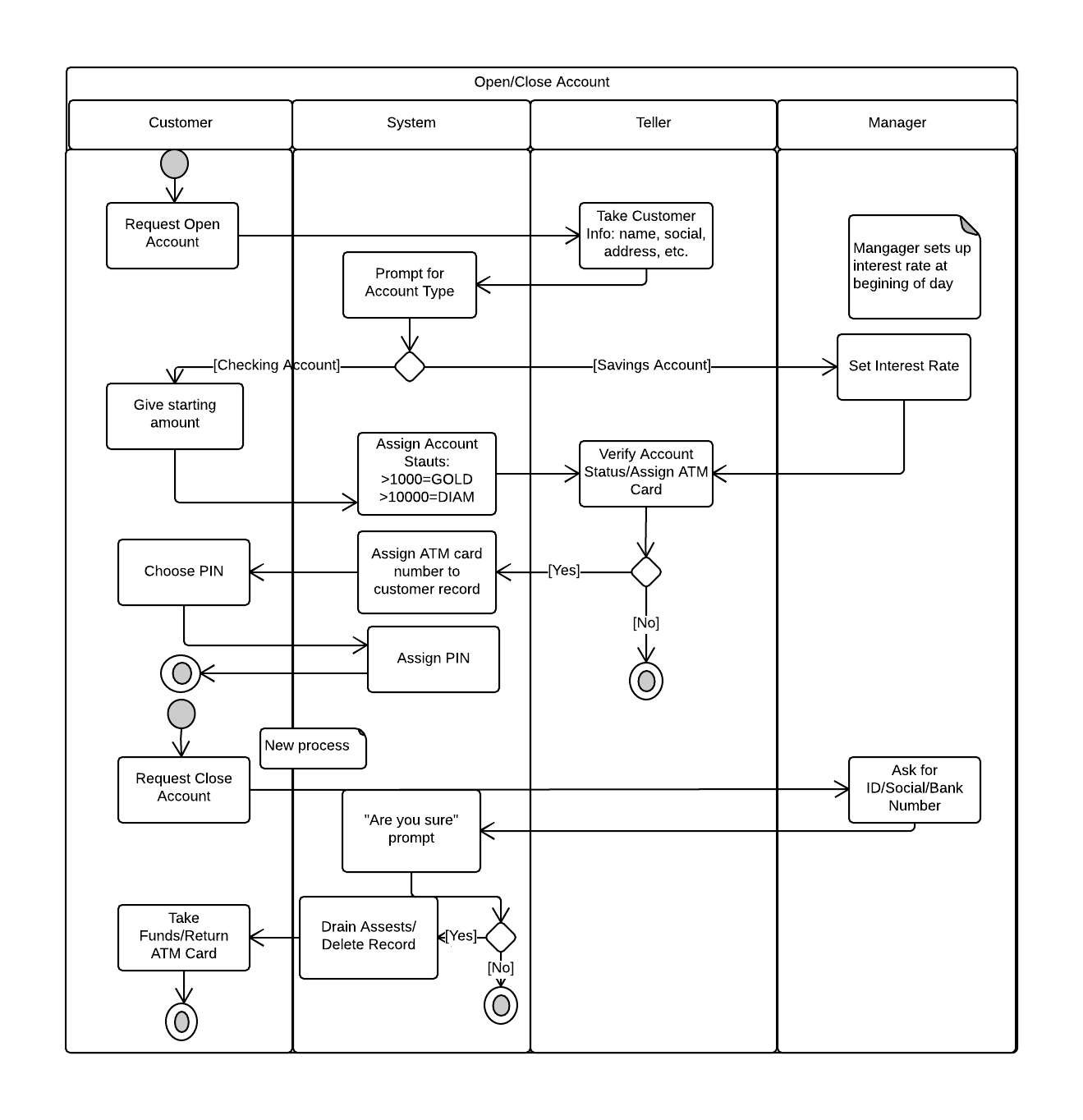
1. **Interface Functionalities:**
   1. **Checking and Saving Account Interfaces:**
      1. The Interfaces shall allow tellers to credit money from checking / saving accounts except CD’s.
      2. The Interfaces shall allow tellers to debit money from checking / saving accounts except CD’s.
      3. The Interfaces shall allow tellers to open and close checking accounts.
      4. The Interfaces shall allow managers to credit money from all accounts.
      5. The Interfaces shall allow managers to debit money from all accounts.
      6. The Interfaces shall allow tellers to open and close saving accounts except CD’s.
      7. The interface shall allow customers to enter a payment amount to balance their credit card account.(payments)
   2. **ATM interfaces:**
      1. The Interfaces shall allow tellers and managers to issue atm cards.
      2. The Interface shall have a “Yes/No” buttons on all warning prompts.
      3. The Interface shall allow customers to choose from savings or credit accounts.
      4. The Interface shall allow customers to input withdrawal amount.
      5. The Interface shall reset every 24 hours.
   3. **Loan interface**
      1. The interface shall allow managers to credit money from all accounts.
      2. The interface shall allow managers to open and close all loan accounts.
      3. The interface shall allow the customer/tellers to choose how much they would like to pay on loan and credit card payment for the month.
      4. The interface shall allow managers to set interest rate of accounts.
   4. **Employee Interfaces:**
      1. The interface shall allow Tellers to pull up all related records of a customer.
      2. The interface shall allow Managers to open and close accounts on request from customer.
      3. The Interface shall allow Managers to make short term or long term loans, as well as set interest rates for each loan
      4. The Interface shall allow Tellers and Managers to assign credit and ATM cards.
      5. The Interface shall have a set of buttons to switch between each account of a customer.
      6. The Interface shall allow Tellers to transfer money amongst accounts.
      7. The Interface shall allow Managers to run reports for credit cards and CD accounts.
      8. The Interface shall allow employees to pull up records of all of a customer's transactions
2. **System Functionality:**
   * 1. The System shall maintain a record of ALL transactions.
     2. The System shall allow for loans to be taken out at on 5, 15 or 30 year plans.
        1. If a loan payment is missed the customer is labeled a problem account.
     3. The Systems shall have an automated system to alert users of when their funds   are to low. This is tied to the overdraw accounts, and the Gold/Diamond accounts. The alert amounts are $50 and $1000, respectively.
     4. The System shall send an alert if a credit card purchase of over $1000 is made.
     5. The System shall keep a record of a customer’s daily ATM usage, and keep them from using that process more than twice a day.
     6. The System shall auto compound interest on savings and loans.
        1. Standard Savings have a daily compound rate
        2. CDs have a fixed rate of return established by management at    creation of the CD
        3. Gold/Diamond have a daily interest rate on all checking and savings accounts of 0.5 x (the rate on the day the account was established)
        4. Account Status is based on the average account balance by adding balance on first and last days of the month divided by 2.
     7. The System shall be able to impose various charges such as bill 50 cents per small withdrawals and deposits on a “That’s my Bank” account, and 75 cents for each automatic debt pay (mortgages, loans, etc.)
     8. The System shall bill no cents per any transaction on a “Gold/Diamond” account.
     9. The System shall assign a late fee of $75 on all missed loan payments.
     10. The System shall assign a fee of $20 on all overdrafted accounts without backup savings set up.
     11. The System shall debit savings accounts for amount overdrafted if that account is setup as a backup account with no fee.
     12. The System shall allow a manager to run reports on a CD or credit card account.
     13. The System shall allow a manager to set interest on checking and savings accounts.
     14. The System shall allow for Stop-Payment for checks issued by “check number”
     15. The System shall allow a user to cancel a transaction at any time; there is a $15 stop payment fee.
     16. The System shall have a checkpoint modification protocol that allows the system to resume where it left off.
     17. The System shall “terminate accounts” by draining the assets to the customer and freezing the account, marking it as inactive.
3. **Screen Functionality:**
   * 1. The Screen shall display a login screen
        1. ATM shall ask for PIN
        2. Teller workstation shall ask for Employee ID and password
     2. The Screen shall display a list of accounts.
     3. The Screen shall allow Management to look at the status of user accounts.
     4. The Screen shall display recent transactions debited to each credit card account.
     5. The Screen shall have an error display for withdrawing more than twice in one day.
     6. The Screen shall have an error display if the user inputs the wrong PIN.
     7. The Screen shall have a display stating change of status for checking accounts.
     8. The Screen shall have a display showing account details such as balance, etc.
     9. The screen shall display a “Are You sure prompt?” on all termination transactions.
     10. The Screen shall display a warning if a withdrawal (ATM) or credit payment (manager) will go over the funds/credit limit of the account in question.

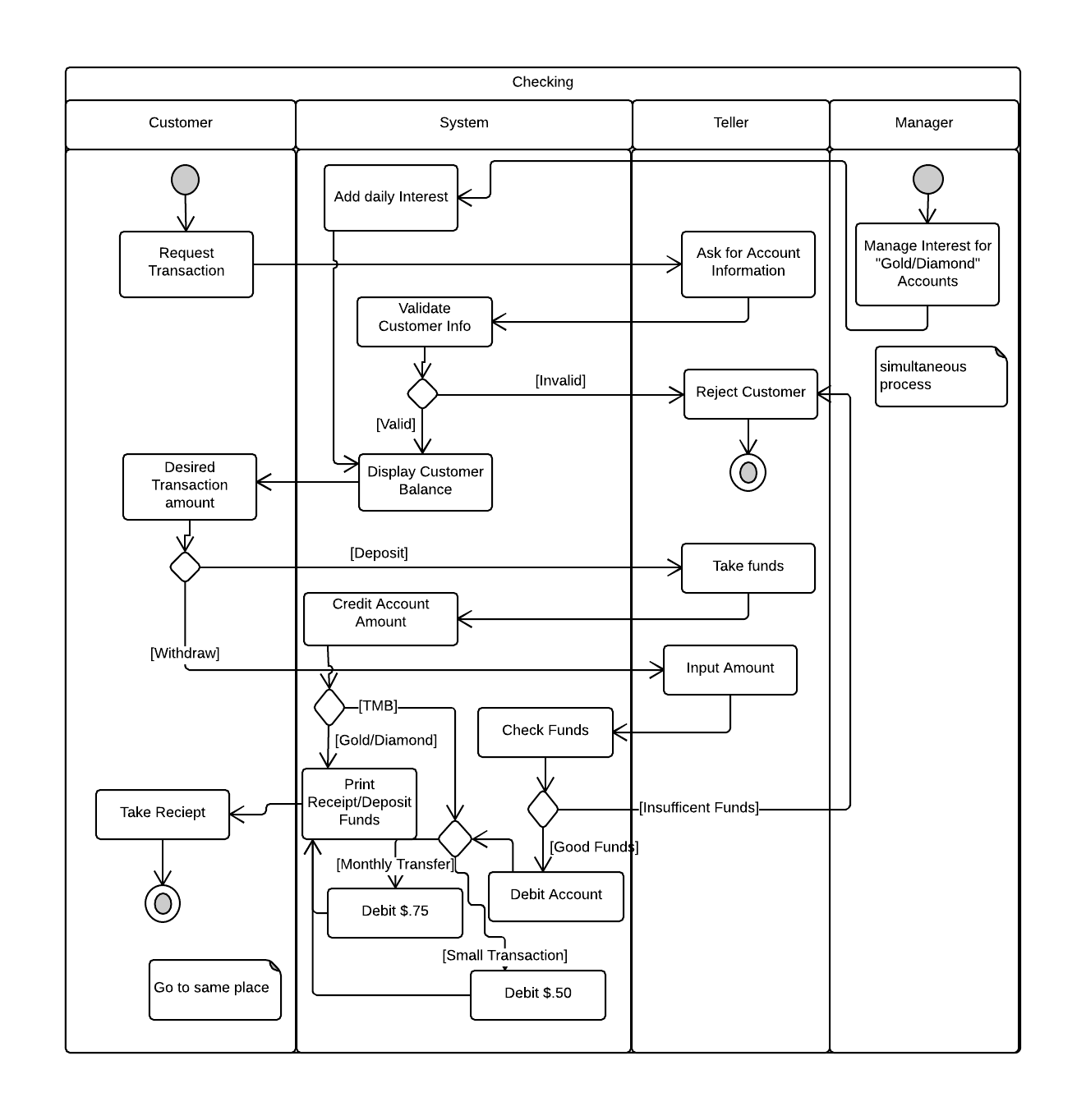


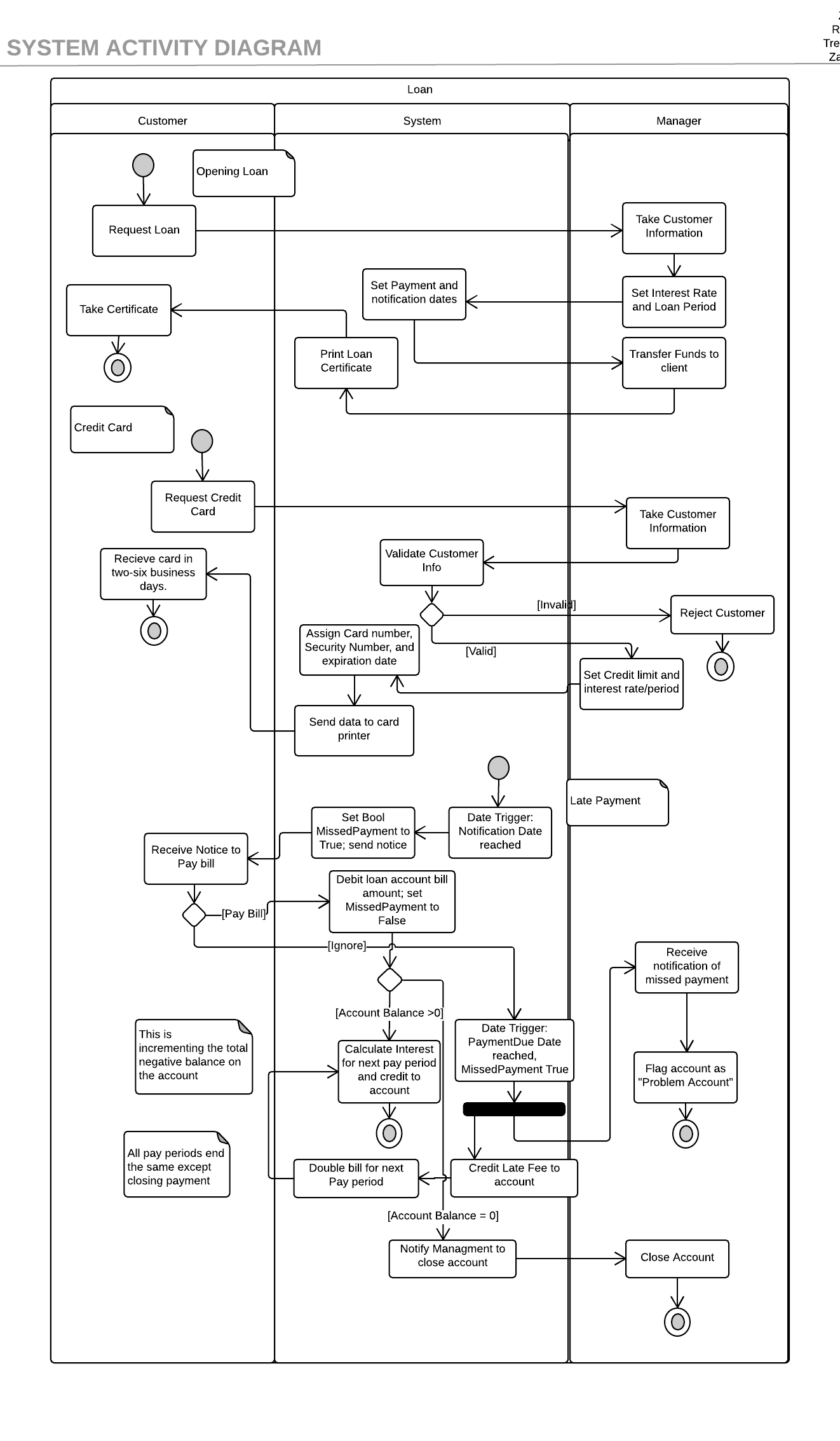


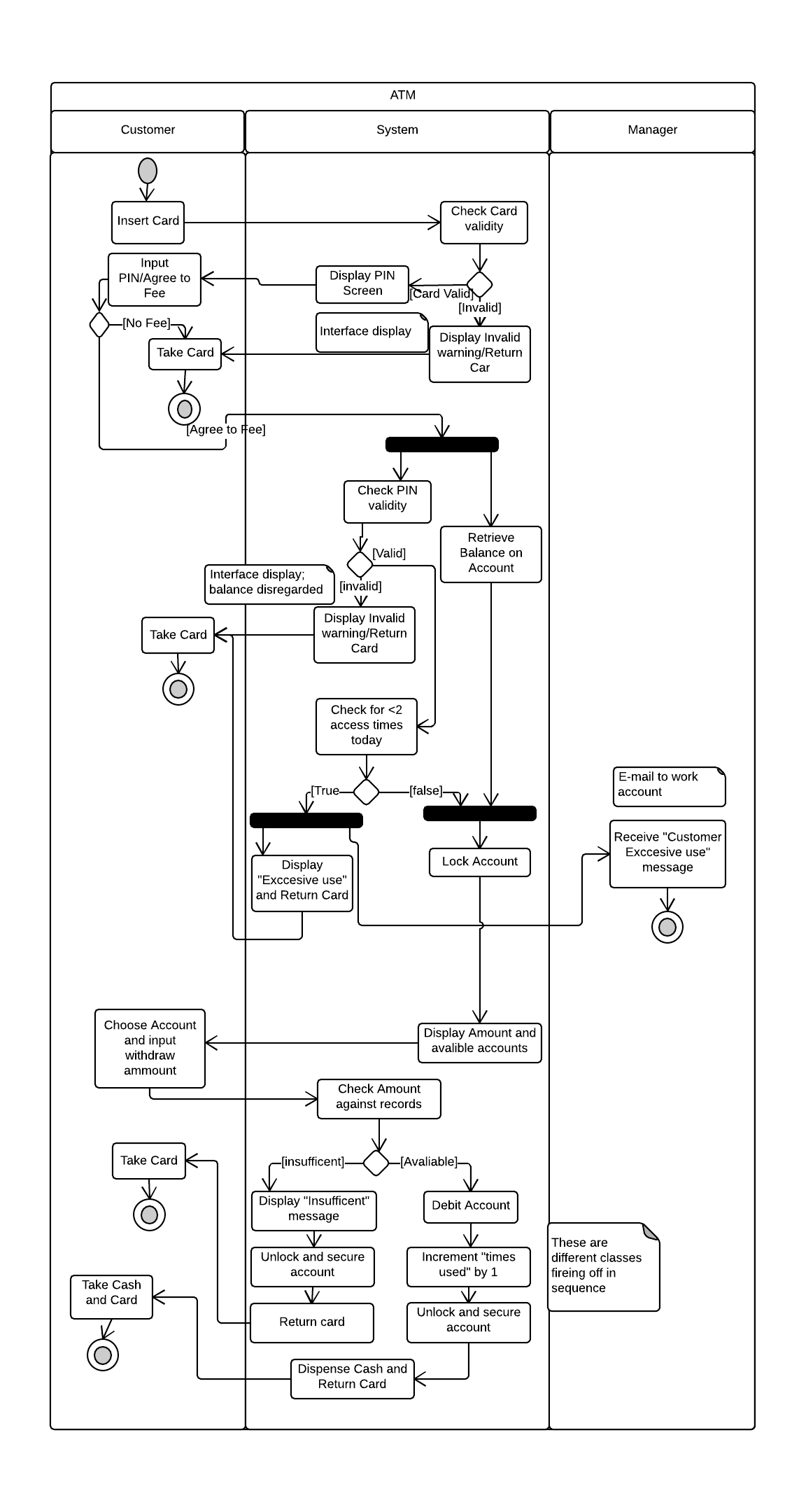


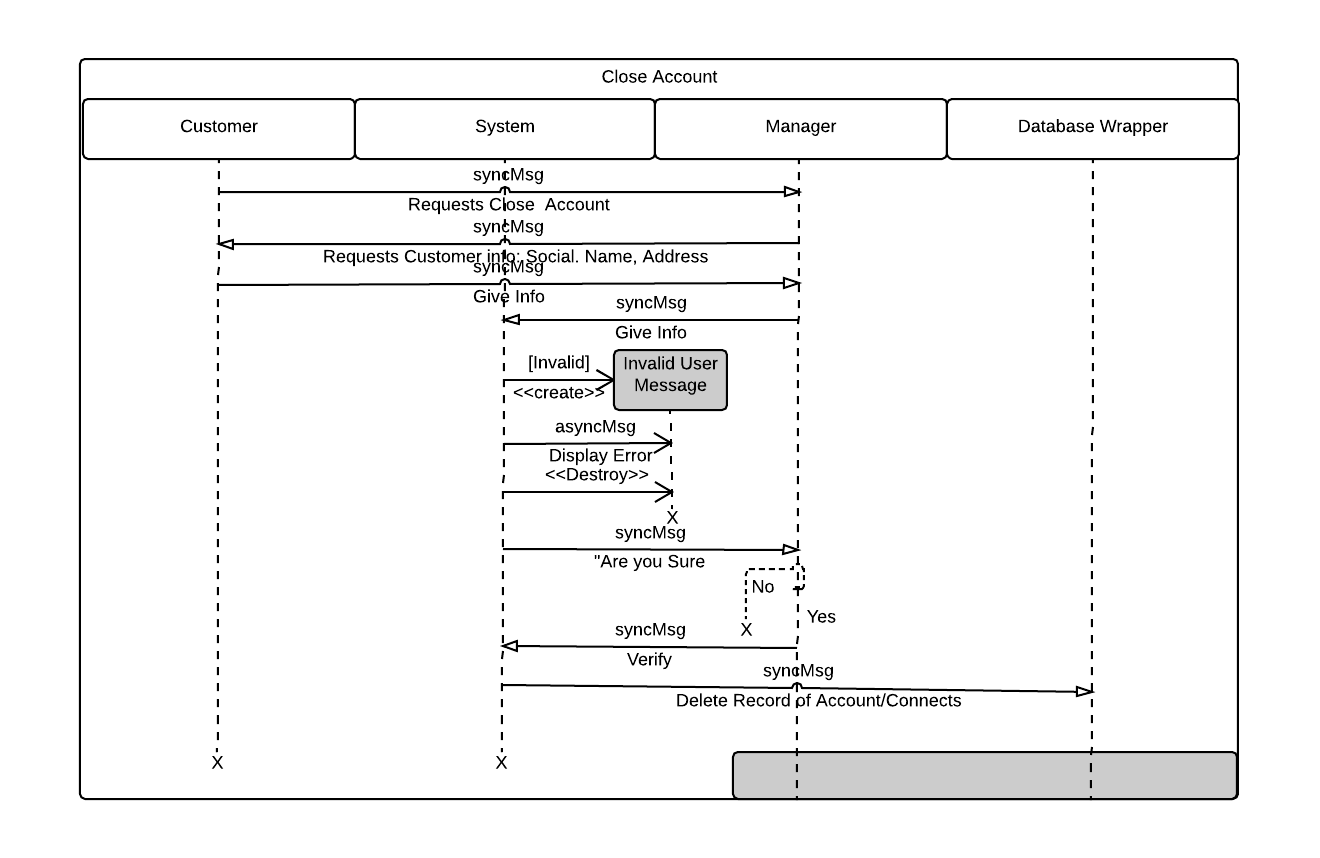
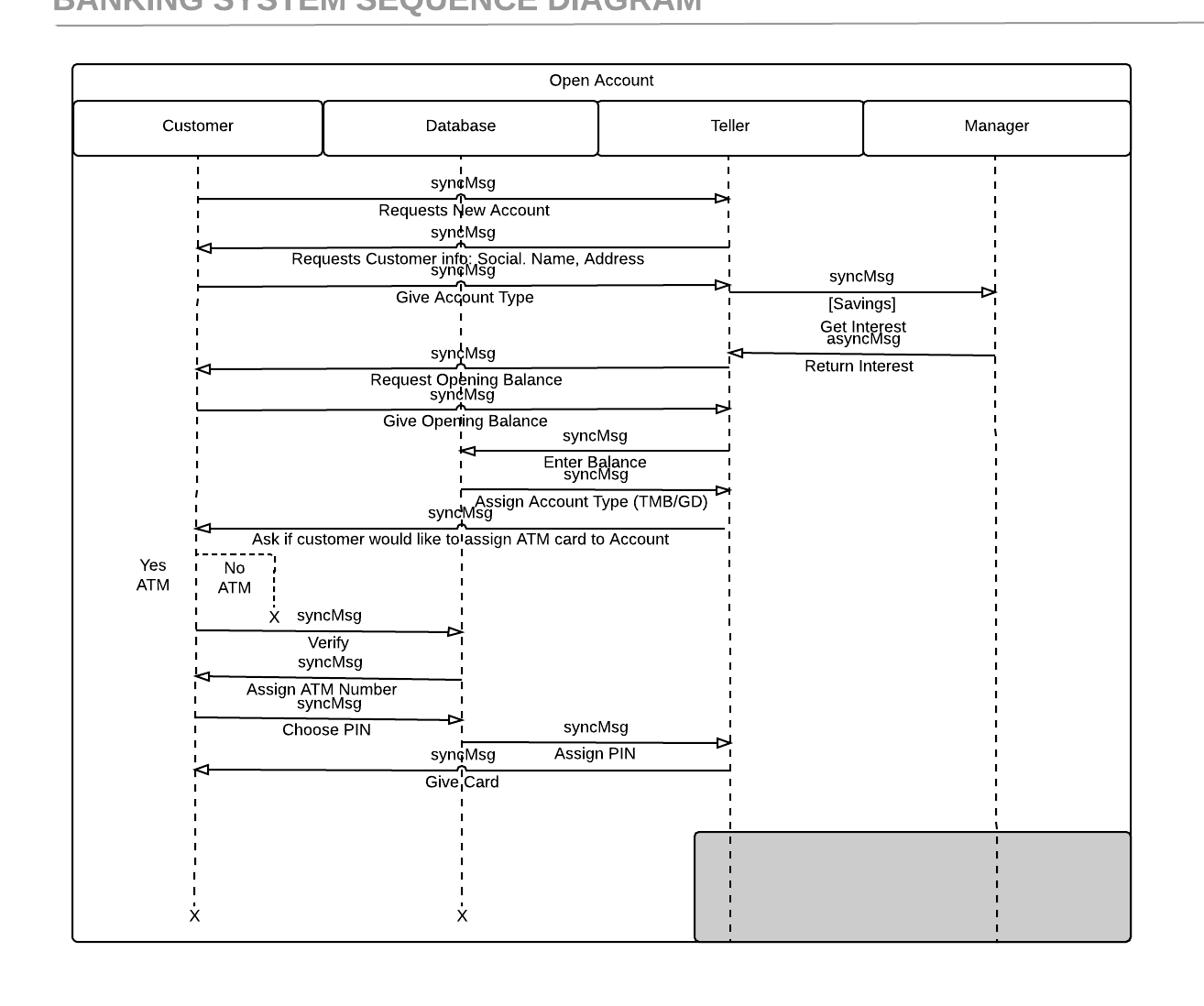


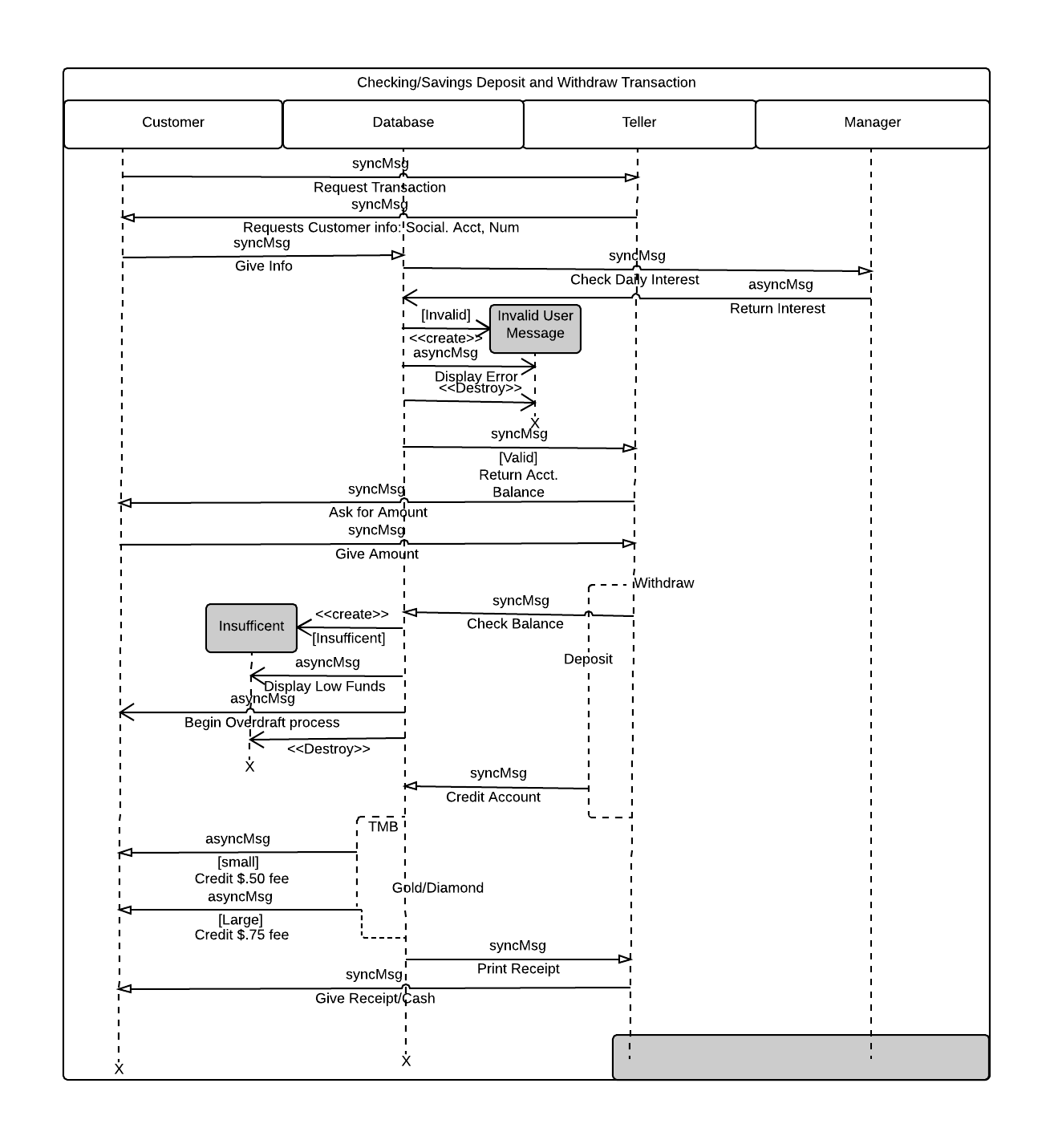


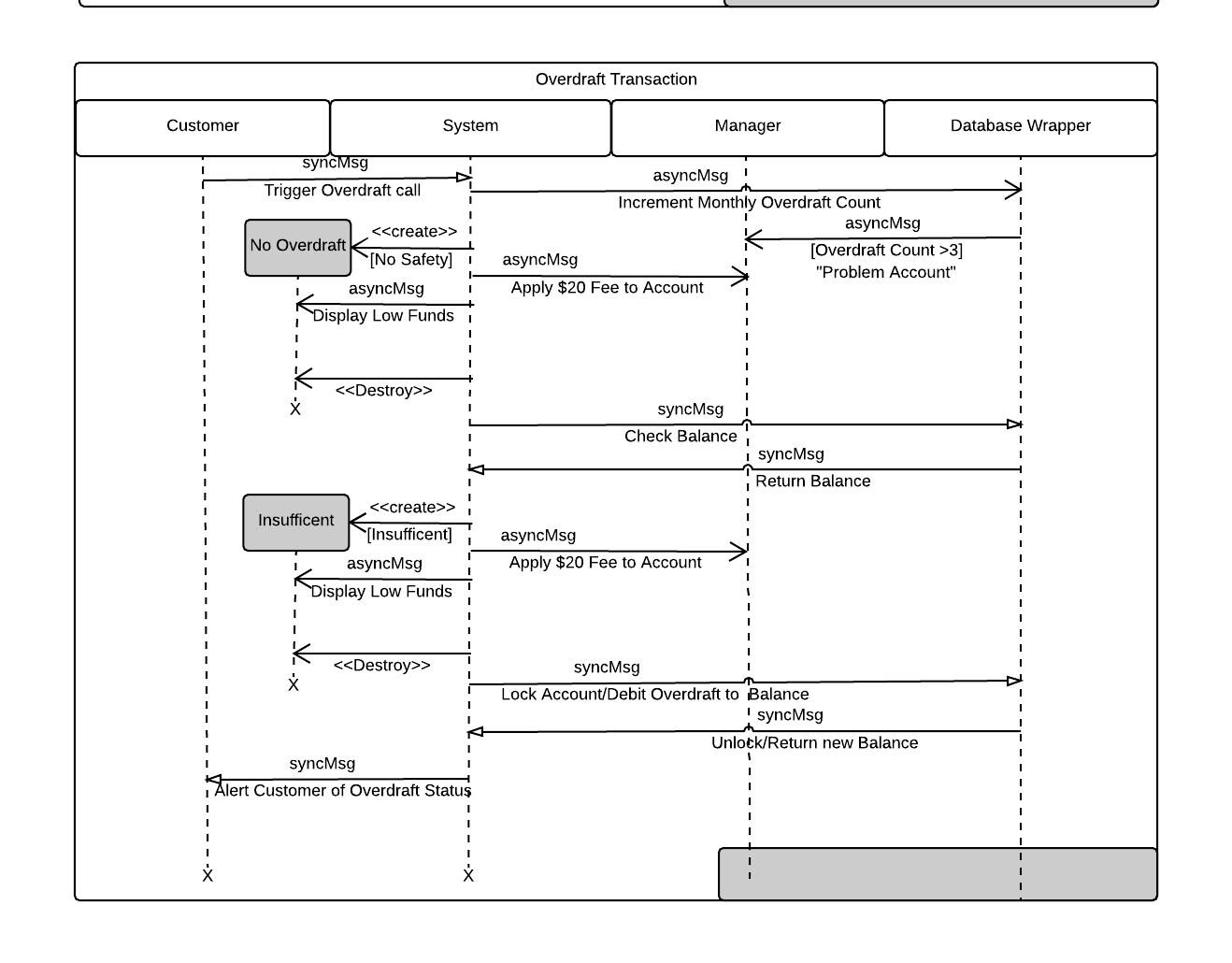


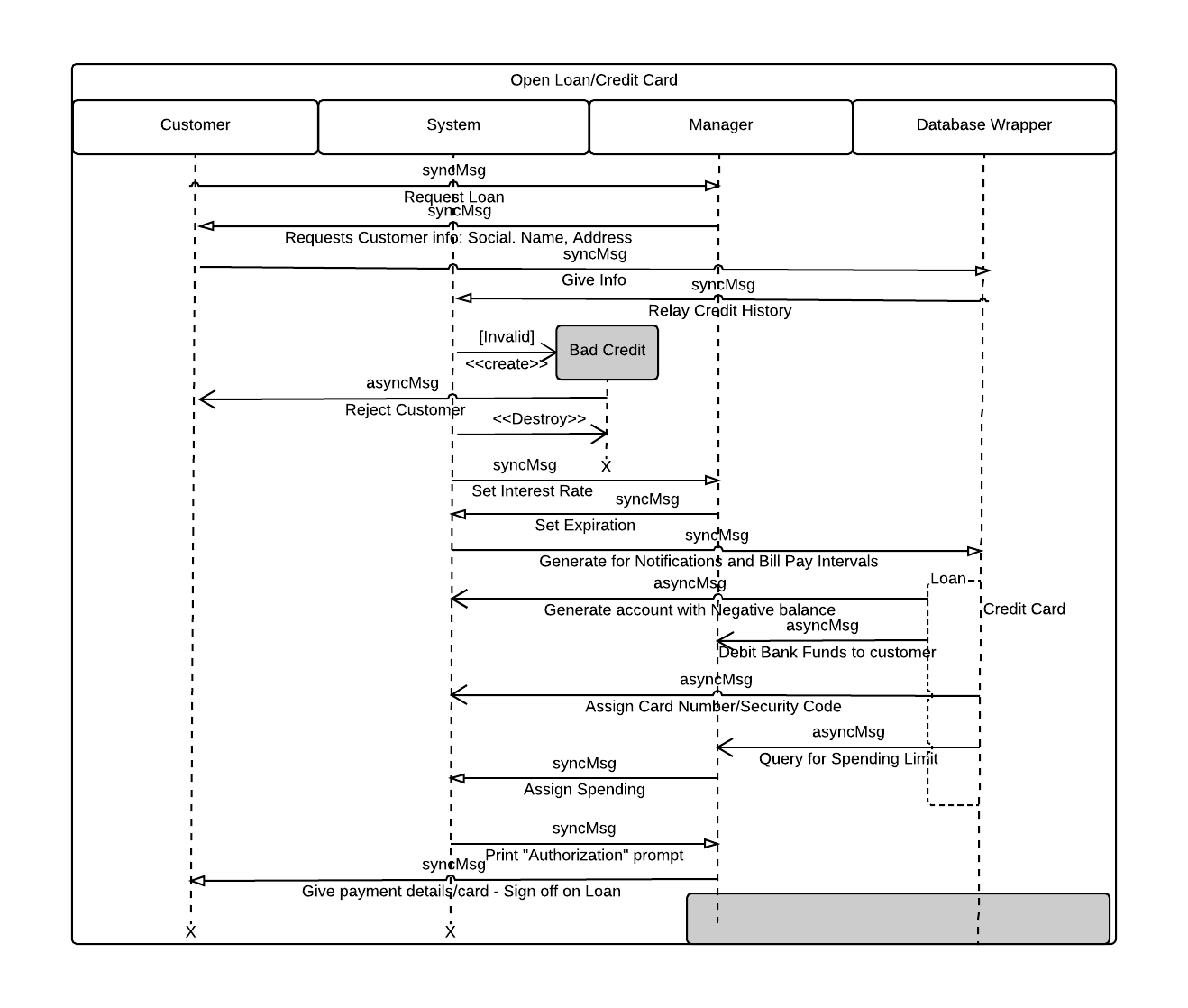


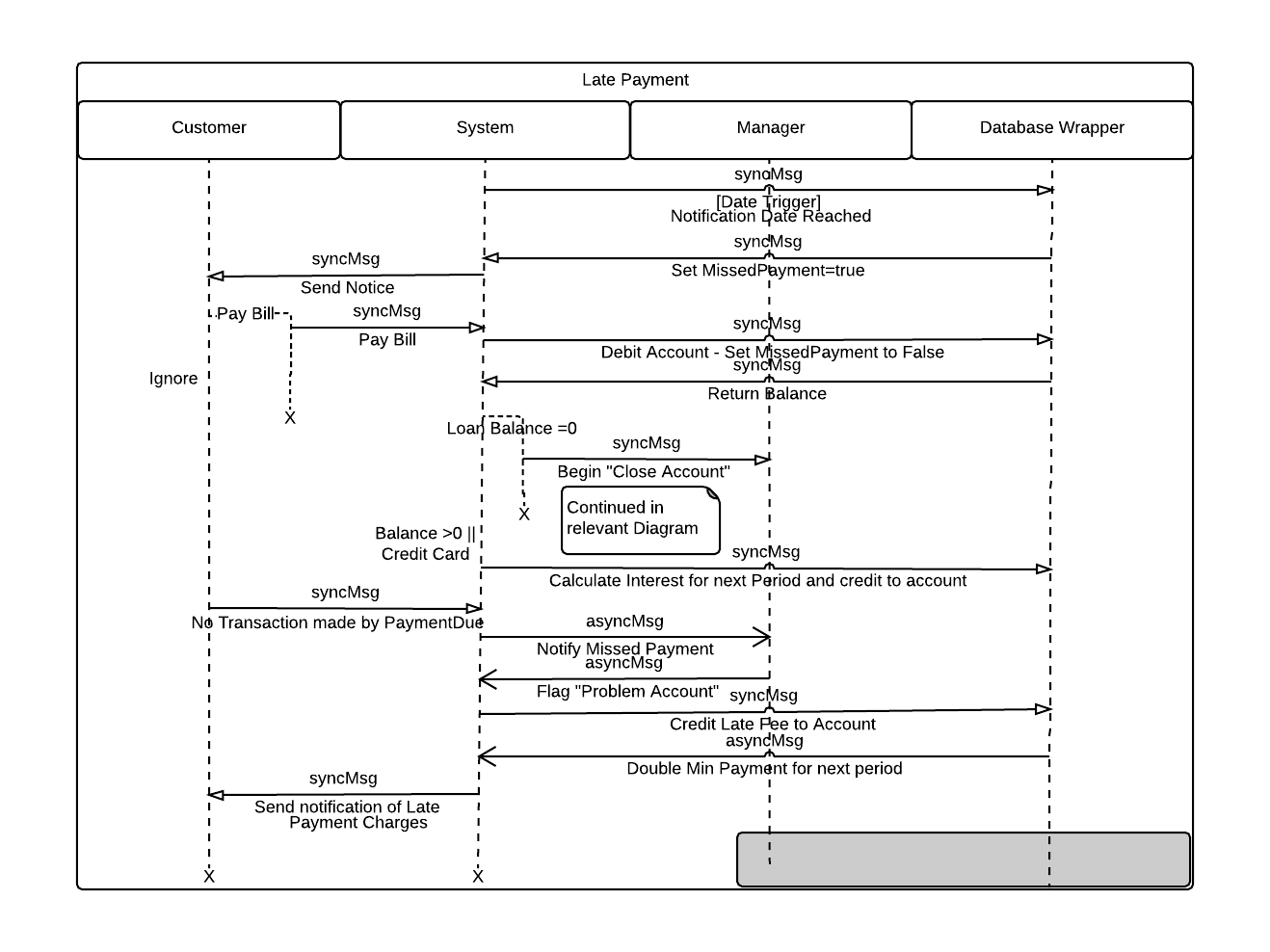


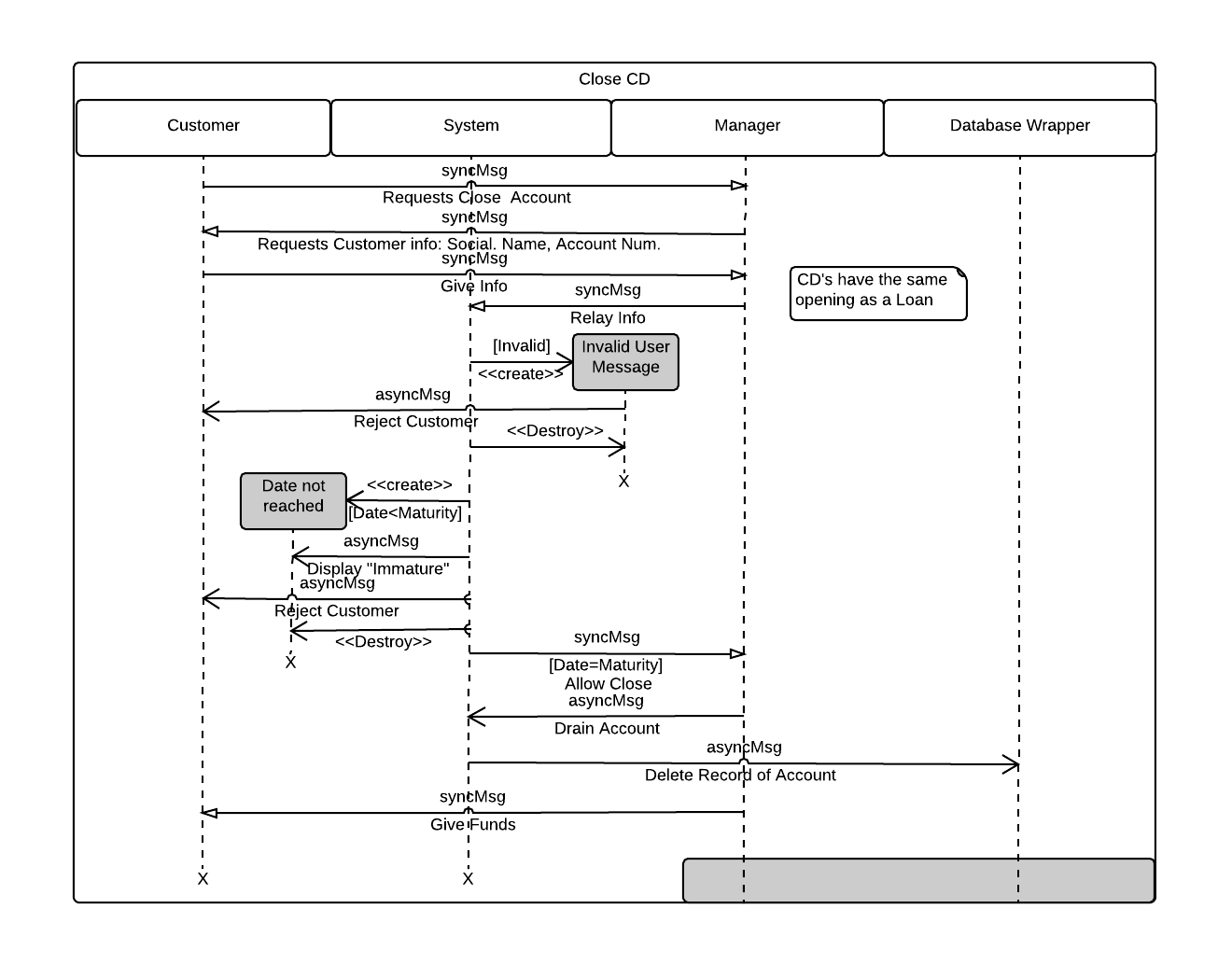
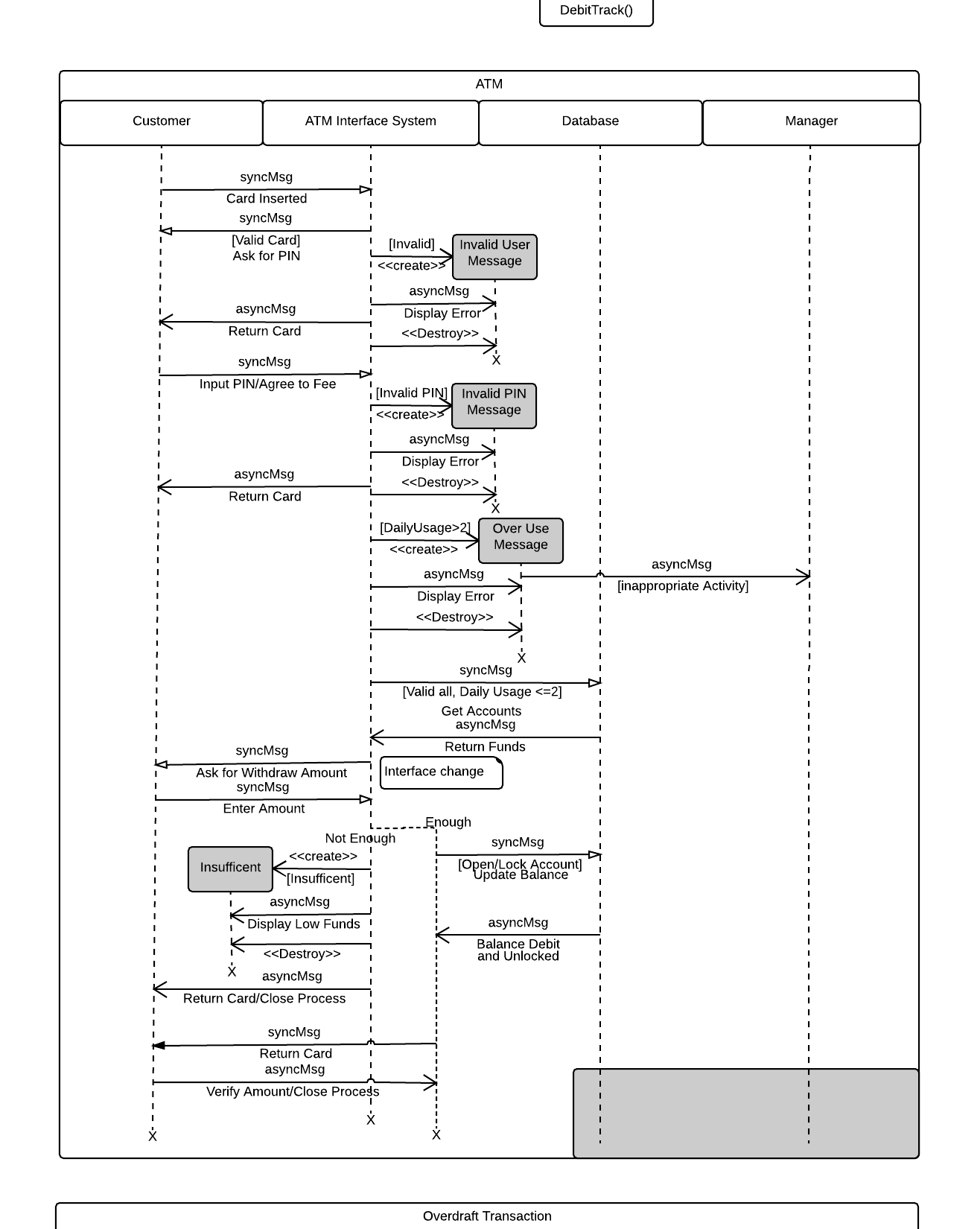


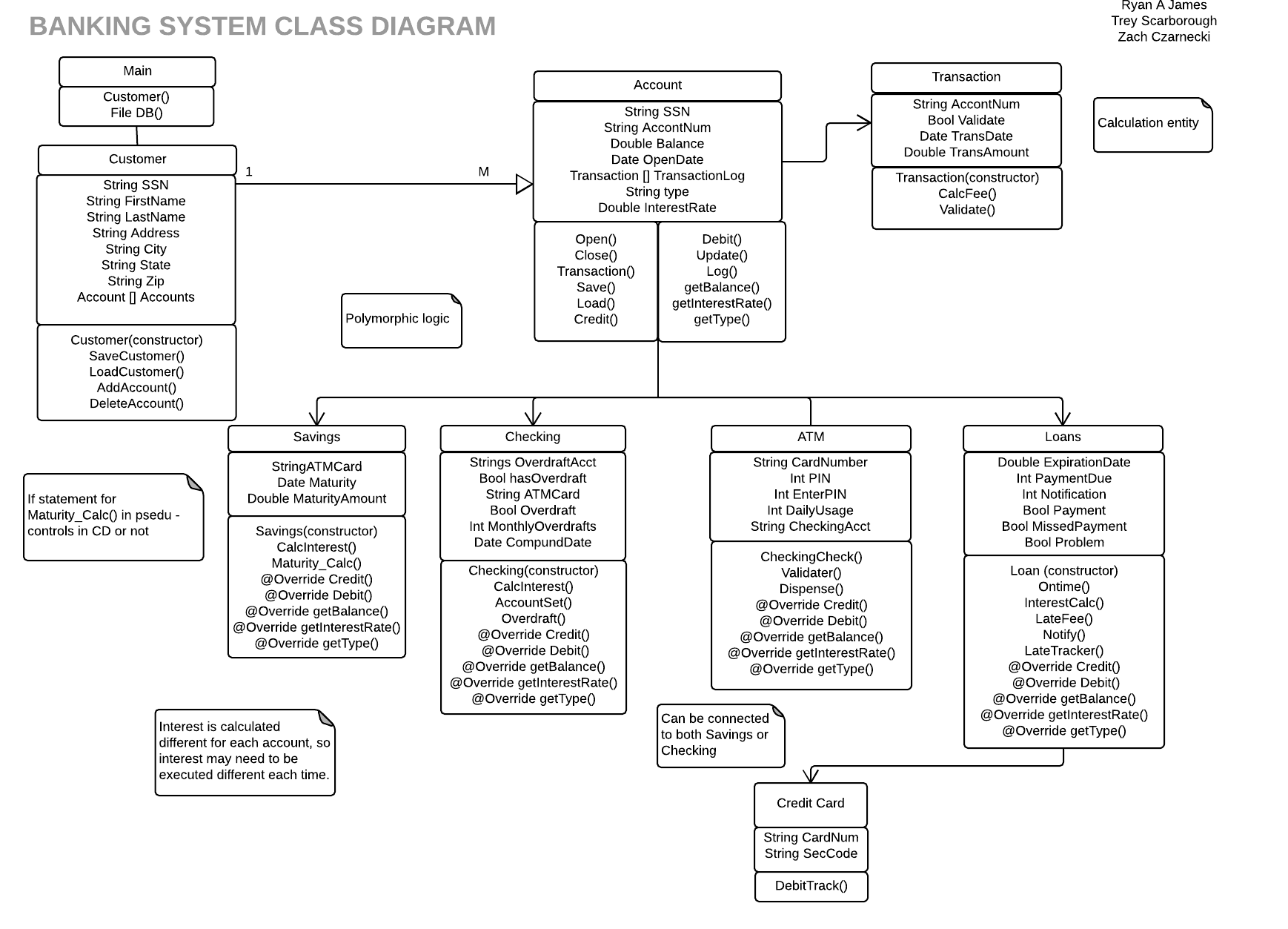










Welcome to Pickett Savings & Loans, the easiest and cheapest way to bank. This manual section will outline all the various functions and systems that rank us one of the “cheapest” and “certainly exists” banks in the nation.

# Log-In

As a bank employee, you will be prompted to enter an employee ID(your own social) as well as a password in order to access the system. Your capabilities will be different depending on if you are a teller or a manager.

# Opening Account

As a teller, you are responsible for making checking and savings accounts.

First, check to see if the customer is already in the system by searching for them based on their social security number.

* If no results pop up, that means they are a new customer. Create a new customer by entering their Social Security number, address, city, state (2 letters), zip code, Last Name, and First Name

Once you’ve found the customer, select which type of account the customer wants: Savings or Checking. Simply enter the initial deposit, and one of two possibilities will happen.

**Savings:**

* Interests rates are assigned daily by management (see Loans). All you as a Teller need to do is enter the starting deposit, and the system will log that interest rate and attach the appropriate amount to the account for the foreseeable future.

**Checking:**

* If a client gives a starting balance of over $1000, or $10,000 they are automatically assigned Gold/Diamond account status. Accounts lower than $1000 are called “That’s my Bank.” Interest is then assigned based on the rate management set at the beginning of the day

Customers also have the option to get and ATM card to access either type of account from any PSL ATM upon opening an account. If they so desire one, ask them what they would like their PIN to be and input into the interface. The system will generate a card number. The information will then be sent to our third-party card printing partners, and the card will be mailed to the customer in 5-10 business days.

# Closing Account

Managers have the ability to close savings and checking accounts on request from the customer/a legally appointed third party. Search for a customer’s accounts based on their social security number.

If such a customer does not exist, the following error will display

If a customer is found, the following warning will display

Choosing yes will drain the account amount from bank assets, and delete the record of the account from the bank data base.

# Withdraw and Deposit

As a Teller, customers come to you for deposits and withdraws.

Step 1: The customer gives you their social security number, and the number of the account they wish to interact with.

* The system checks the validity of the customer information. If it isn’t the following error window is displayed
* At the same time, the system will que up the interest rate for the account to add the daily interest to the account if applicable

Step 2: Ask the customer for the amount they would like to withdraw/deposit.

Step 3: For Withdraw: the system will check that the user has enough funds to make the transaction. If not, the overdraft process is initiated. For Deposit - the amount is simply credited to the account.

Step 4: If the account had under $1000 at the start of transaction (“That’s my Bank”), a fee is applied to the transaction, and debited from the account. $.50 for amounts lower than $500, and $.75 for larger.

Step 5: A digital receipt is displayed/printed.

# Overdraft

This is a safety net a customer can request in the event of an overdraft on either their ATM card, or during an in-person transaction. A customer can request that their savings account can be set up as an “overdraft account” for their checking in the event that they try to withdraw too much from a checking.

The Overdraft process is automatically triggered and works as follows:

1. The system looks to see if that checking account is set up with and overdraft account. If not, then the account is fined $20 for their next interest period.
2. If there is an overdraft account, the system will query the amount of the overdraft. If the overdraft account does not have adequate funds, the $20 fee is still applied.
3. If the account does have adequate funds, the extra funds are automatically withdrawn from that account.

Customer accounts are flagged for each instance of an overdraft. Those with more than 3 in one month are “problem accounts.”

# Opening Loan, CD or Credit Card

Managers are the only people with this ability. Similar to opening a savings or checking account, the manager must first ask and input the customer’s Social Security Number, name and Address. Only pre-existing customers may open a loan/CD/credit account.

With this information, the system will query the client’s credit history\* and relay it back for interest calculation purposes. This step is skipped if applying for CD.

\*A base score will be used across all accounts in the demonstration

If the customer has bad credit, the system will reject them. If they have good credit, the system will give a recommended Interest Rate, though the manager has the ability to overwrite this.

* If a customer wishes to open a CD, they give an initial deposit at this point

The manager must then set an expiration/maturity rate for the account. From here, the system will generate bill pay and notification intervals, and create the account.

* If it is a Loan, a negative account will be created with a debit of the requested loan amount, and the funds will be credited to the customer/customer’s checking account. The starting balance is the negative of amount loaned out
* If it is a credit card, the database will assign a Card number and Security code to the account, and prompt for a spending limit to be set by the manager. The starting balance is zero.
* If it is a CD, then there is no additional step here. The balance is the starting investment.

The display will then print out a confirmation of the establishment of the account. Credit cards will be mailed to the customer in 5-10 business days.

# Notification and Late Payment

The system is set up to automatically track the pay period and lateness of payment on a loan or accrued spending on a credit account. Once a date is reached (usually 5 days before the due date of the payment for that month), the system will send an e-mail to the customer, as well as setting an alert variable (MissedPayement) to true.

1. They pay the bill:
   1. The amount is debited to the account, and alert variable is set to false, and the customer goes about their day.
   2. The system then calculates the interest for the next pay period and credits it too amount.
   3. If it is a loan account and the balance is $0 the closing process initiates.
2. The Bill is not paid:
   1. If the variable has not been reset to false by the date of payment, management is notified to set the account to “Problem account.”
   2. The database will then credit a late fee to the account balance of $20.
   3. The bill pa amount for the next period is doubled.
   4. The customer is sent a notification of all this.

# Closing CD/Loan

While Credit Cards will reset on reaching their expiration date (this process is automatic in the system), a CD or Loan reach a maturity date. For loans, the maturity date is more of a calculated estimation of when the account will reach a balance of $0, but for CD’s it is when the customer can cash out the accrued interest.

Again, the customer gives their social security number, name, and account number. If the system detects that the account has not matured, or that the information provided is incorrect, then it will reject the customer.

If the date has been reached, the system will drain the funds of the CD to the customer/customer’s checking account, and delete the record of the account from the database.

# ATM

The following are simple instructions on how to use a PSL ATM/Customer Interface.

1. Insert card/input card number
2. If the card is valid, a prompt will appear asking for the User’s Pin. Otherwise, they are rejected and the card is returned.
3. If the PIN is valid, the customer’s daily usage is checked and incremented. If the usage is already greater than 2 they are rejected and the card is returned.
4. If the Usage is allowable, the funds attached to the card are returned and displayed.
5. From here the user is allowed to input the funds they wish to withdraw.
6. The user inputs the desire amount and the system checks available funds. If there are not enough funds, the following error displays and the card is returned.
7. If there are enough funds, the account is debited and the following message is displayed. The card is returned.

**All classes link to a same named “.form” file that is auto-generated by the compiling software as all interfaces are assembled graphically.**

**Main:**

Initialize customer ArrayList: Main Data structure for program – stores all customers; active User: Stores user type that logged in, myDate: Stores current Date for the system, and dateString: String version of date for database storage.

Main():

Initializes database read and loads interface.

ReadDB();

Method Called from Main that parses Database.txt pulling out database records from each line. Then parses each line based on ‘|’ delimiter storing each record as a String[] and passing to the updateUser() function.

updateCustomerArray();

Method called from Main that takes in a String[] of record. It then Creates user objects based on the values of the String[]. It then checks if the User already exists in the Customers[]customers and if they do not, adds user to System Users[]List, if they do exist gets index of User in []customers. Creates Account object based on string[] and Adds account to user object in User[].

switch (type):

Controls and sets account to polymorphic class type, adds variables based on extra “record” array fields not previously called. “Savings,” “CD;” “TMB,” “Gold,” and “Diamond” are Checking class sub-types and add an overdraft count. “CC” is credit cards, and adds Last Payment, Monthly Payment date, open date, and payment post date. “long” and “short” are loans that set length, and all the variables from credit card. The default will error out "Invalid Account Type for User: " followed by first and last name and SSN.

hasOverdraftAccount();

Checks to see a checking account has an overdraft attached. Based on count of “flag” variable.

saveDB();

Updates outside database file with new entries on close. Parses based on variables established in updateCustomerArray for records array and compiles into String DBRecord.

checkingDBRecord();

Establishes that the database record instance is a checking account based on record of interestrate and overdraft count/Boolean. Also uses the latter to control overdraftFlag and pushes to database.

creditDBRecord();

Establishes that the database record instance is a credit account based on presence of time fields.

**LogInScreen:**

This controls the first screen of the program; the Log-In. How a user interacts with this code determines what their additional permissions are.

LogInScreen():

Set up an initialize button layout and action settings. Also allows for keyboard input.

main():

Initializes dialog between Log-In Dialogue and System.

checkCredentials():

Uses Else If function to checks user input of desired system privileges for matching username and password pair to allow access; such as “(userName.compareToIgnoreCase("teller") == 0 && password.compareToIgnoreCase("password") == 0)” to control access to the Teller menu by way of the username teller, and password, password. Errors and resets if incorrect match.

onOK():

Checks to see if there is entry in both username and password fields. If not, resets page on click. Runs checkCredentials() if input is found.

onCancel():

Resets page on selection of relevant button.

**FirstMainMenu():**

After the LogIn Page, this is the main nerve center of the program. As such, almost all interface calls run through here; with three versions depending on user: teller, manager, and ATM (bank customer).

FirstMainMenu():

This constructor sets Teller as the default view. Except those specifically for management, all buttons display here. If the user logged in as a manager, openLoanButton, openCDButton, closeAccountButton, and setInterestRatesButton are set to true. “ATM” has all but withdrawlButton and depositButton set to false. The duttons are as follows

openAccountButton:

Go to onOpenAccount().

updateDateButton:

Go to setSystemDateTime().

closeAccountButton:

Go to onCloseAccount().

debitAnAccountButton:

Go to onDebitAccount().

closeSystemButton

Go to onCloseSystem().

lookupCustomerButton

Go to onLookupCustomer().

openCDButton

Go to onOpenCD().

openLoanButton:

Go to onOpenLoan().

main():

Initializes dialog between Log-In Dialogue and System.

setSystemDateTime():

Takes in the system dateStr variable, formats it, and applies it to the variable Date newDate for interest calculations.

onOpenAccount():

Create Account instance and unpack relevant interface page.

onUpdateDate():

Compute current date from system memory and previous date and format.

onCloseAccount():

Create instance of CloseAccount and unpack relevant interface page.

onCloseSystem():

Disposes of input, closes system

onLookupCustomer():

Create DebitAccount instance and unpack relevant interface page.

onOpenCD():

Set acctType to “CD” and create instance of New\_Existing. Force New\_Existing to load up as CD page.

onOpenLoan():

Set acctType to “Loan” and create instance of New\_Existing. Force New\_Existing to load up as Loan page.

onDebitAccount():

Create DebitAccount instance and unpack relevant interface page.

**CustomerAccountsMenu2:**

Creates a GUI table to pull accounts from without searching.

CustomerAccountsMenu2():

Using an instance of Customer and default table variables, a String array called columns is used as the header of a 2D String array, data, which displays the first three customer accounts by default. Three for loops control the dimensions of data. Table jt is then instantiated and filled with the two arrays. Following prepareRender, jt is then set to be able to scroll through more accounts.

Component prepareRender:

Visually formats jt to white and light gray.

Main():

Creates recursive instance called “dialog.” This is the variable that controls the visibility of all other interfaces.

setIndex():

Set the database index.

onCancel():

Dispose of input and close system.

**LookupCustomer:**

This is the search function that allows a user to look up a customer with their SSN or Account using text input.

LookupCustomer():

Set up an initialize button layout and action settings. Also allows for keyboard input.

main():

Initializes dialog between user and System.

lookupUserIndex():

Allows user to search for account based on “ssn” variable. Gets Main.customers array and re-divides to obtain list of Social Security numbers. List is matched to user input, and display is triggered by boolean alertNeeded. Else, displays error pop-up. Returns Index.

getCustomer():

Uses “index” returned from lookupUserIndex() to retrieve full customer record.

onCancel():

Resets page on selection of relevant button.

onOK():

This triggers when lookupUserIndex()checks to see if the SSN is in the database already. If it is, this function will trigger the editCustomer GUI to load. If there is no match, this function will reset the page.

handleInvalidUser():

Display error message if no matching SSN is found, or the user inputs the wrong format.

**editCustomer:**

Allows Teller or manager to edit customer information.

Main:

Displays buttons created in AddBankAccount for user interaction via instance of class AddBankAccount.

onOK():

Gets customer information from database with series of getters and establishes database commit function

onCancel:

Resets GUI to FirstMainMenu.

setCustomer():

Setter for onOK.

setCustomerDetails():

Fill text boxes with relevant customer record information

**InputCustomerInformation:**

This lets a Teller/Manager input the information of a customer.

InputCustomerInformation():

Sets up and initiates user interface and button functions.

Main():

Construct user input as dialogue with system; allow display.

onOK():

Set up getters and confirmation abilities of user input for relevant fields to generate new instance of class data type “customer.” Nested if sequence of “FirstMainMenu.acctType.equalsIgnoreCase” controls what sub-section of the ArrayList it is added to based on AcctType. Also checks to see if user already exists by calling public class New\_Existing in a cyclical fashion, as the GUI will not run if New\_Existing finds pre-existing record.

onCancel():

Disposes of input and resets user interface.

**New\_Existing:**

Allows Teller/Manager to differentiate new and pre-existing customers when interacting with database. Automatic class.

New\_Existing():

Attaches class call to button call that is activated in other classes. Uses actionPerformed stock functions to pass along information to onNewCustomer() or onExistingCustomer() classes. Also allows for close on cancel.

Main():

Construct user input as dialogue with system.

onCancel():

Disposes of input and resets user interface; for insurance mostly.

onNewCustomer():

Verifies that input is new customer, and sends back to InputCustomerInformation.

onExistingCustomer():

Launches menu to add new account if customer is found in database.

**Customer:**

This is a data type defining class. Customer is the main variable of the system and it is through these that all other data types relate back to.

Customer():

Constructor that takes in Social Security Number, street Address, city, state, zip code, first name and last name.

setSSN() – setFirstName() – setLastName() – setStreetAddress() -setCity() – setState() – setZip(): These are variable setters.

getSSN() – getFirstName() – getLastName() – getStreetAddress() -getCity() – getState() – getZip(): These are variable getters.

getAccounts():

This is a special getter that relates the customer to the different accounts associated with the customer.

**Account:**

This is the constructor class for the Accounts, setting up the basic details.

setBalance()- setInterestRate() – setAccountNumber() – setType() – setOwnerID() – setDate(): These are variable setters.

getAccountNumber – getOwnerID() – getDate() – getBalance() – getInterestRate() – getType(): These are the necessary variable getters.

getOverdraftAccount() – getBackupAccountNumber()– getLastPaymentDate():

These getters are for special functions that are not necessary for an account to be valid, but can be attached by the user.

credit() – calcInterest() - debit():

These abstract classes allow will allow for interaction with the data type.

updateTransactionList():

Adds a record to a special array each time to class is called.

**Checking:**

This is the default account type. It takes in most of it’s information from the super class, but has additional Boolean controls that allow for backup and overdrafts.

Checking():

Constructor for the class that sets Customer SSN (OwnerId), balance, Interest Rate, Account Number, type, and opening date from the super class. The Boolean hasOverdraftAccount is also set here, and if it is set to true, an override constructor is called that will add the overdraft account number.

setBackupAccountNumber():

Searches for savings account associated with same OwnerID if hasOverdraftAccount is selected as true. Will return error if no “type=Savings” is found.

getBackupAccountNumber():

Getter for overdraftAccount.

getAccountNumber – getOwnerID() – getDate() – getBalance() – getInterestRate() – getType(): These are the necessary variable getters.

credit():

Credits (add funds) account and calculates new balance.

debit():

Debits (subtracts funds) account and calculates new balance. Can also trigger overdraft or change account type if amount dips below thresholds of $1500, $1000, or $0.

stopPay():

Disposes of input and resets interface to previous page.

calcInterest():

Auto credit interest on account on time based intervals.

**Savings:**

This is the class for savings accounts. They can also be called as overdraft/backup accounts.

Savings():

Constructor that also sets SSN (OwnerID), balance, Interest Rate, Account Number, Type (always “savings” here), and Date.

getAccountNumber – getOwnerID() – getDate() – getBalance() – getInterestRate() – getType(): These are the necessary variable getters.

credit():

Credits (add funds) account and calculates new balance.

debit():

Debits (subtracts funds) account and calculates new balance. It will terminate the program if the account hits 0.

stopPay():

Disposes of input and resets interface to previous page.

calcInterest():

Auto credit interest on account on time based intervals.

**CD:**

This allows a manager to create an instance of a CD savings account.

CD():

Constructor that also sets SSN (OwnerID), balance, Interest Rate, Account Number, Type (always “savings” here), and Maturity Date.

getAccountNumber – getOwnerID() – getMaturity() – getBalance() – getInterestRate() – getType(): These are the necessary variable getters.

cashOutCD():

This function contains an If condition that calls compareDates() each day. When it returns true, the account is automatically cashed out and closed.

compareDates():

Compares the current date to the pre-established Maturity Date. These must be converted to Date type variables. With this conversion, the former strings can now be mathematically compared and return a condition to cashOutCD(): true when they equal, false when the calculation (current date-maturity date) is negative. The entire class is in a try/catch loop.

**Loan:**

This allows managers to start loan accounts.

Loan():

Constructor that takes in the card number, starting balance, monthly due date, interest rate, Type (always “CC” as established in Account super class), current date, last payment date, monthly due date, opening date, and a missed Payment variable. Also contains setters.

getBalance() – getInterestRate() – get Type() – getCardNumber() – getDate() – getLastPayment() – getMonthlyPayment() – getOpenDate() – getMissedPayment() – getMonthlyDueDate():These are the necessary variable getters. They override the parent.

missedPayment():

This is a date function that triggers and flags an account when the user misses a payment.

credit() – calcInterest() - debit():

These classes are not abstract here, and control the payments and misses of the account.

**CC:**

This is the data type for credit card accounts.

CC():

Constructor that takes in the card number, starting balance, monthly due date, interest rate, Type (always “CC” as established in Account super class), current date, last payment date, monthly due date, opening date, and a missed Payment variable. Also contains setters.

getBalance() – getInterestRate() – get Type() – getCardNumber() – getDate() – getLastPayment() – getMonthlyPayment() – getOpenDate() – getMissedPayment() – getMonthlyDueDate():These are the necessary variable getters. They override the parent.

missedPayment():

This is a date function that triggers and flags an account when the user misses a payment.

getOverdraftAccount():

Links to Savings account for in case credit limit is overdrawn

credit() – calcInterest() - debit():

These abstract classes allow will allow for interaction with the data type. Override parent.

creditCharge():

Calls the transaction data type to set up connection that fills transaction Array List of account. This is the main difference standard loans.

**Transaction:**

This allows class CC to log each transaction and use of the card.

Transaction():

A simple constructor that takes in the location, date, and time of a purchase.

Transaction(Overload):

An extra constructor for purchases with no location: ex; Amazon.

**AddBankAccount:**

Allows user to add new checking and savings accounts to the database as controller of GUI “AddBankAccount.”

AddBankAccount():

Set up and initialize user interface for button control.

buttonOK: Set’s up confirmation button to run on click with actionperformed() class.

butonCancel: Set’s up cancel button to run on click with actionperformed() class.

addWindowListener: uses windowclosing() to call onCancel().

actionperformed(): a duplicated class that calls for the end of an operation.

Main:

Displays buttons created in AddBankAccount for user interaction via instance of class AddBankAccount.

onOK:

Triggers user query search upon click of relevant button, and returns result if found using same named Boolean variable. This will then call up the relevant option of the search. After the proceeding switch statement the record is saved to the database and the page is reset to Main.

switch (acctTypecomboBox1.getSelectedItem().toString());

Displays relevant GUI based on user query selection/search.

case "Checking": gets relevant text entry fields for creation of checking account; namely allowing for calculation of account type.

case “Savings”: gets relevant text entry fields for creation of savings account; namely for interest calculation.

case “CD”: gets relevant text entry fields for creation of CD account; namely maturity dates and rates.

case “CC” gets relevant text entry fields for creation of Credit Card account; namely expiration dates.

checkingStatus():

Simply calculates, sets, and returns the account type of a checking account based on levels of funds: >$1500 sets to “Diamond,” >$1000 is “Gold,” and all lower amounts are “TMB.”

onCancel():

Closes the current GUI and resets it to the previous interface using pre-fabricated SystemDateTime class.

**AddCDAccount:**

This allows managers to create new CD savings accounts.

AddCDAccount():

Sets up and initializes user interface and functions.

buttonOK: Set’s up confirmation button to run on click with actionperformed() class.

butonCancel: Set’s up cancel button to run on click with actionperformed() class.

addWindowListener: uses windowclosing() to call onCancel().

actionperformed(): a duplicated class that calls for the end of an operation.

main():

Opens dialogue between interface and system.

onOK():

Controls Lookup function that allows for customer search via social security number. If the customer is found, this function will also take the basic information of the account and allow for the creation an instance of a “CD” datatype; namely SSN, balance, interest, account number, and maturity date. It will then send this instance back to the main account arrayList. It will also reset the interface after running.

onCancel():

Disposes of input and resets interface to FirstMainMenu.

**AddLoanAccount:**

This allows for the creation of Loan accounts.

AddLoanAccount():

Sets up and initializes user interface and functions.

buttonOK: Set’s up confirmation button to run on click with actionperformed() class.

butonCancel: Set’s up cancel button to run on click with actionperformed() class.

addWindowListener: uses windowclosing() to call onCancel().

actionperformed(): a duplicated class that calls for the end of an operation.

main():

Opens dialogue between interface and system.

onOK():

Controls Lookup function that allows for customer search via social security number. If the customer is found, this function will also allow for the creation of new instance of the data type “Loan,” with further calculations controlled by the selection in a combo-box: “Long term,” “Short term,” and “Credit Card.” Each selection will change what interest and payment attributes are assigned to the account, such as interest or payment date. Some of these are then set by the user before the Interface closes on completion of the creation of the instance.

onCancel():

Disposes of input and resets interface to FirstMainMenu.

**DebitAccount():**

This is the controller for all screens that involve removing funds from an account.

DebitAccount():

Sets up and initializes user interface and functions.

buttonOK: Set’s up confirmation button to run on click with actionperformed() class.

butonCancel: Set’s up cancel button to run on click with actionperformed() class.

addWindowListener: uses windowclosing() to call onCancel().

actionperformed(): a duplicated class that calls for the end of an operation.

main():

Opens dialog with user and sets to visible.

isDouble():

Converts the text input of amount to double for calculation.

onOK():

First, double checks that desired account to withdraw from exists with Boolean wasAccountFound. If it doesn’t, it errors out process. If it is found, a prompt will display to confirm the account information, as well as an echo of the amount. This window is linked to “JOptionPane.YES\_NO\_OPTION.” If JOptionPane.NO\_OPTION: onCancel() is called. Else, the Account in question (referenced here as “acct”) is debited and locked.

onCancel():

Return to FirstMainMenu and dispose of input. Reset current time.

**CloseAccount():**

This controls the screen that lets a manager delete an account. It is very similar to DebitAccount(), but lacks a field for amount input; simply asking for SSN and account numbers.

CloseAccount():

Sets up and initializes user interface and functions.

buttonOK: Set’s up confirmation button to run on click with actionperformed() class.

butonCancel: Set’s up cancel button to run on click with actionperformed() class.

addWindowListener: uses windowclosing() to call onCancel().

actionperformed(): a duplicated class that calls for the end of an operation.

main():

Opens dialog with user and sets to visible.

onOK():

First, double checks that desired account to withdraw from exists with Boolean wasAccountFound. If it doesn’t, it errors out process. If it is found, a prompt will display to confirm the account information, and if the person is sure they wish to close the account. This window is linked to “JOptionPane.YES\_NO\_OPTION.” If JOptionPane.NO\_OPTION: onCancel() is called. Else, the Account in question (referenced here as “acct”) is deleted from the customer ArrayList. On completion, returns system to FirstMainMenu.

onCancel():

Return to FirstMainMenu and dispose of input. Reset current time.