

**Team 9 - Anthony Dawson,**

**Osman Khan**

**Erick Morales**

**Kevin Vu**

## **ATM System Design and Application**

### **Table of Contents**

<b>Problem Statement</b>	<b>2</b>
<b>Use Cases and Scenarios</b>	<b>4</b>
<b>Domain analysis</b>	<b>9</b>
<b>Requirements modeling</b>	<b>11</b>
Class Diagrams	11
Detailed class diagrams	15
CRC	16
Activity diagram	17
Sequence diagram	22
Collaboration diagram	28
State diagram	29
<b>Component Design</b>	<b>30</b>
<b>User Interface Design</b>	<b>31</b>
<b>Deployment Design</b>	<b>31</b>
<b>Coding</b>	<b>32</b>
Main	32
ATM	32
Menu	44
Account	49
Savings	51
Checking	52
Operations	54
CheckBalance	55
Deposit	56
TransferFunds	57

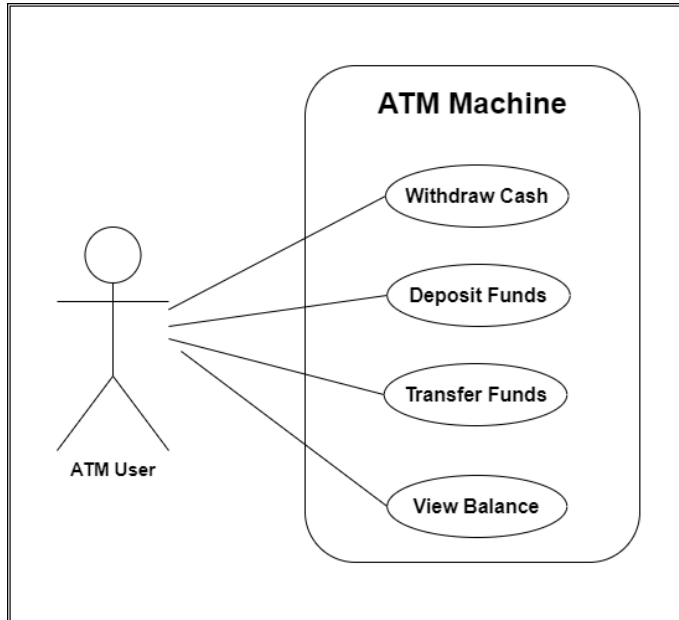
Withdraw	59
<b>Unit Testing</b>	60
<b>Runtime output</b>	62
<b>Summary</b>	66

## **1. Problem Statement**

Our team will be creating an ATM application which will be coded in Java on Eclipse. The program will run on the desktop (OS will be Windows 10). The bank's ATM machine will welcome users with a message followed by a graphical user interface menu providing all of the

possible functions that can be done. The functions provided in the menu will allow the user to select between withdrawing, depositing, checking balance, transferring funds between the customer's checkings and savings account, and exiting if they no longer wish to use the ATM machine. Users will only have the capability of having one account which will include savings and checkings. Each account will have a corresponding pin that allows the customer to access their account. With every function there should always be a way to cancel the current action. The "show balance" action will allow users to see the balance of their savings or their checking. The "withdrawal" action will allow the account holder to withdraw a certain amount of money from their checking or their savings. The "deposit" function allows you to deposit a certain amount of money into their accounts. The "transfer" function allows the user to move funds between their checking and savings. A receipt will be displayed if the user successfully deposits, transfers, or withdraws.

## 2. Use Cases and Scenarios



Case#1 - Kevin Vu

**Use Case Name:** Withdraw Cash

**Summary:** Customer will be able to use a pin number to withdraw funds from their bank account

**Actor:** ATM Customer

**Precondition:** Displays the welcome message and gives the user an option to choose cash withdraw

**Description:**

1. Customer enters their PIN account number into ATM
2. System will check if the pin number is correct
3. If the customer enters the PIN incorrectly, it will prompt the user again for the PIN.
4. The system will give a selection of options for the user to choose from
5. If the customer chooses withdraw funds, it will give the user an option to select which account type they want to withdraw from

6. The user selects savings or checking and then the system will check if the customer has enough funds to withdraw from and check if the daily limit has not been met
7. If the conditions are met, the machine will dispense the selected amount
8. System will print a receipt
9. System displays the home screen

**Alternative Flows:**

(3) If the customer enters the wrong PIN, it will prompt an incorrect PIN message and ask the user to type in the PIN again.

(6) If the customer types in an amount that exceeds the existing funds in the account, the system will display an exceeding message and the system will ask the user to type in a new amount

**Post Condition:** Funds are withdrawn from the customer's account

**Non-Functional Requirement:** User friendly, Accuracy

Case #2 Erick Morales

**Use Case Name:** Deposit Funds

**Summary:** Customers will be able to use a PIN to deposit funds onto their account.

**Actor:** Customer

**Precondition:** Displays the welcome message and gives the user an option to choose deposit funds.

**Description:**

1. Customer enters their account PIN number into the ATM
2. System will recognize the PIN number
3. If the customer enters the PIN incorrectly, it will prompt the user again for the PIN.
4. The system will give a selection of options to choose from

5. Deposit function is selected
6. System will then prompt the customer to enter to his/her checking or savings account
7. System will ask the customer how much they want to deposit to their account
8. System will display a successful message and print a receipt
9. System returns to the welcome screen

**Alternative Flows:**

(3) If the customer enters in the wrong PIN, it will prompt an incorrect PIN message and ask the user to type in the PIN again.

(8) If nothing is inserted into the deposit cartridge the system will reset to the home screen.

**Post Condition:** Funds are deposited into the customer's account

**Non-Functional Requirement:** User friendly, Accuracy

Case #3 Osman Khan

**Use Case Name:** Transfer Funds

**Summary:** Customers will be able to use a pin number to transfer funds between their checking and savings account.

**Actor:** ATM Customer

**Precondition:** Displays the welcome message and gives the user an option to choose transfer funds

**Description:**

1. System prompts the user to type in a PIN
2. If the customer enters the PIN incorrectly, it will prompt the user again for PIN.

3. The system will give a selection of options for the user to choose from one of them being transfer balance
4. Picking transfer balance will lead to a new menu that displays an option to pick which type of account they want to transfer from, either savings or checking
5. After an account type is picked the system will prompt a new menu of how much the user would like to transfer between the two balances
6. If the amount selected is enough for the account type, the balance is sent from the initial account to the selected one
7. The system displays a digital receipt that shows the amount transferred between the accounts
8. After the receipt displays the menu is taken back to the home screen

**Alternative Flows:**

(2) If the customer enters in the wrong PIN, it will prompt an incorrect PIN message claiming no account is associated with that number and ask the user to type in the PIN again.

(6) If the account type has insufficient funds for transferring an error message will be displayed and the user is asked to enter an amount again

**Post Condition:** Funds are transferred between the customer's account types

**Non-Functional Requirement:** User friendly, Accuracy

Case #4 - Anthony Dawson

**Use Case Name:** View Balance

**Summary:** The bank customer will use a PIN to access their account and view the balance within their checkings or savings through the ATM.

**Actor:** ATM User

**Precondition:** The ATM displays the main menu options.

**Description:**

1. The system prompts the user to type in a PIN.

2. If the customer enters the PIN incorrectly, it will prompt the user again for the PIN.
3. The system will give a selection of options for the user to choose from one of them being transfer balance.
4. The customer selects view balance.
5. The machine displays 3 options for the user to choose from (Checking, Savings, Return).
6. The user will select either the Checking or Savings account to view.
7. The machine displays the account name, its balance, and presents a return option.
8. The user selects "Return".
9. The machine returns to display the main menu options.
10. The user selects cancel.
11. The machine ejects the card.
12. The machine returns to the idle screen.

- **Alternative flows:**

- (2) If the customer enters in the wrong PIN, it will prompt an incorrect PIN message claiming no account is associated with that number and ask the user to type in the PIN again.
- (5) If the customer selects return, the machine displays the main menu options again.

- **Postcondition:**

- The user has seen their account and its balance, and the machine is idle.

**Non-Functional Requirement:** User friendly, Accuracy



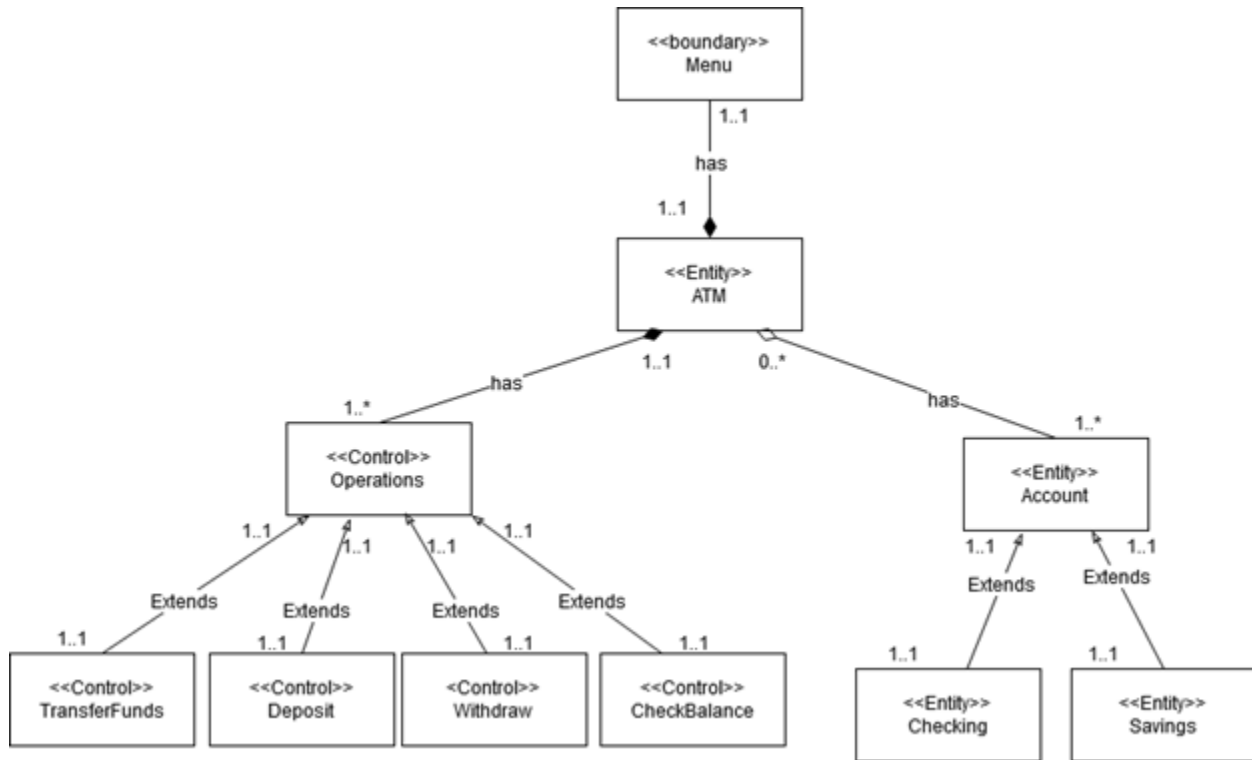
### **3. Domain analysis**

In this program the domain is “ATM and Bank Account Access”. A bank customer will use a physical ATM device to access and update their bank account’s balance. There will be a number of ways for the user to interact with their account. The ATM will verify account information from the user with the accounts from the bank servers. The user chooses an operation that handles updating the account and displays the updated account information afterwards.



#### 4. Requirements modeling

##### a. Class Diagrams



#### Class Definitions

- **ATM:** The ATM entity object provides users with access to their accounts and the operations that manipulate the funds within those accounts. The ATM entity will also verify the PIN number with the account. An ATM can have 1 to many accounts associated with it.
- **Account:** This is an entity class that represents a client's banking information every person has one and only one account. It will be the parent class to the checking class and savings class. Client's pin is associated with this class.
- **Checking:** This entity class extends the account class and stores the other way an account can hold a client's money. As with the account class a user only has one checkings account.

- **Savings:** This entity class extends the account class and stores one of the ways an account can hold a client's money. As with the account class a user only has one savings account.
- **Operation:** Control class which serves as a parent to the functions which can be performed in the ATM.
- **Withdraw:** This control class allows the user to remove money from either their savings or checkings funds.
- **Deposit:** This control class allows the user to insert money into either their savings or checking funds.
- **CheckBalance:** This control class allows the user to see the amount of funds in either one of their account types. Comes from the use case diagram.
- **TransferFunds:** This control class allows the user to transfer funds between the checking and saving account types. Comes from the use case diagram.
- **Menu:** This boundary class displays the ATM menu. Input of the keypad, enter, clear, and cancel buttons are implemented in this class. Allows the user to enter choices and data to select operations they desire.

### Class Creation Narratives

An ATM user will use the ATM to access and change the funds in their accounts through a pin.

- The ATMUser is the actor in this situation.
- The ATM entity object provides users with access to their accounts and the operations that manipulate the funds within those accounts. The ATM entity will also verify the PIN number with the account.
- The Account entity will hold the pin number and general information about the ATM Customer, such as their name, address, and phone number.

An ATM Customer will have a Checking and Savings account linked to their name and pin.

- We will create two entities that inherit the Account entity, the Checking entity and Savings entity. These entities will hold the information about the funds within the accounts.

The ATM will present the user with multiple operations to choose from.

- The Menu boundary will be created to handle the input and output between the user and ATM.
- The Operations control object will represent each transaction operation in the ATM.

The ATM user will be able to withdraw and deposit funds from their account.

- A Withdraw control object will verify the account balance is sufficient for the chosen withdrawal amount and remove it from the balance.
- A Deposit object will simply be used to add funds to the account balance

The ATM user will be able to view their different account balances and transfer funds between them.

- A CheckBalance object will be used to present the balance amount from the user chosen account.
- A TransferFunds object can take two accounts and move funds between them.

### **Class Function Narratives**

**Account:**

The Account class will hold the users information such as name and phone. It will also hold an account number and the users PIN number.

The class will have methods to retrieve the PIN, Account Number, and an abstract method to get and set the balance.

### **Checking and Savings:**

These classes will have attributes of the account balance and account names.

There is a method to retrieve the accounts balance.

### **Operations:**

The Operations class creates a record of the ATMs transaction and gives the user a receipt.

### **TransferFunds:**

This class will have a method that takes two accounts and moves funds between them.

### **Deposit:**

This class has a method that takes the user account and adds funds dictated by the user.

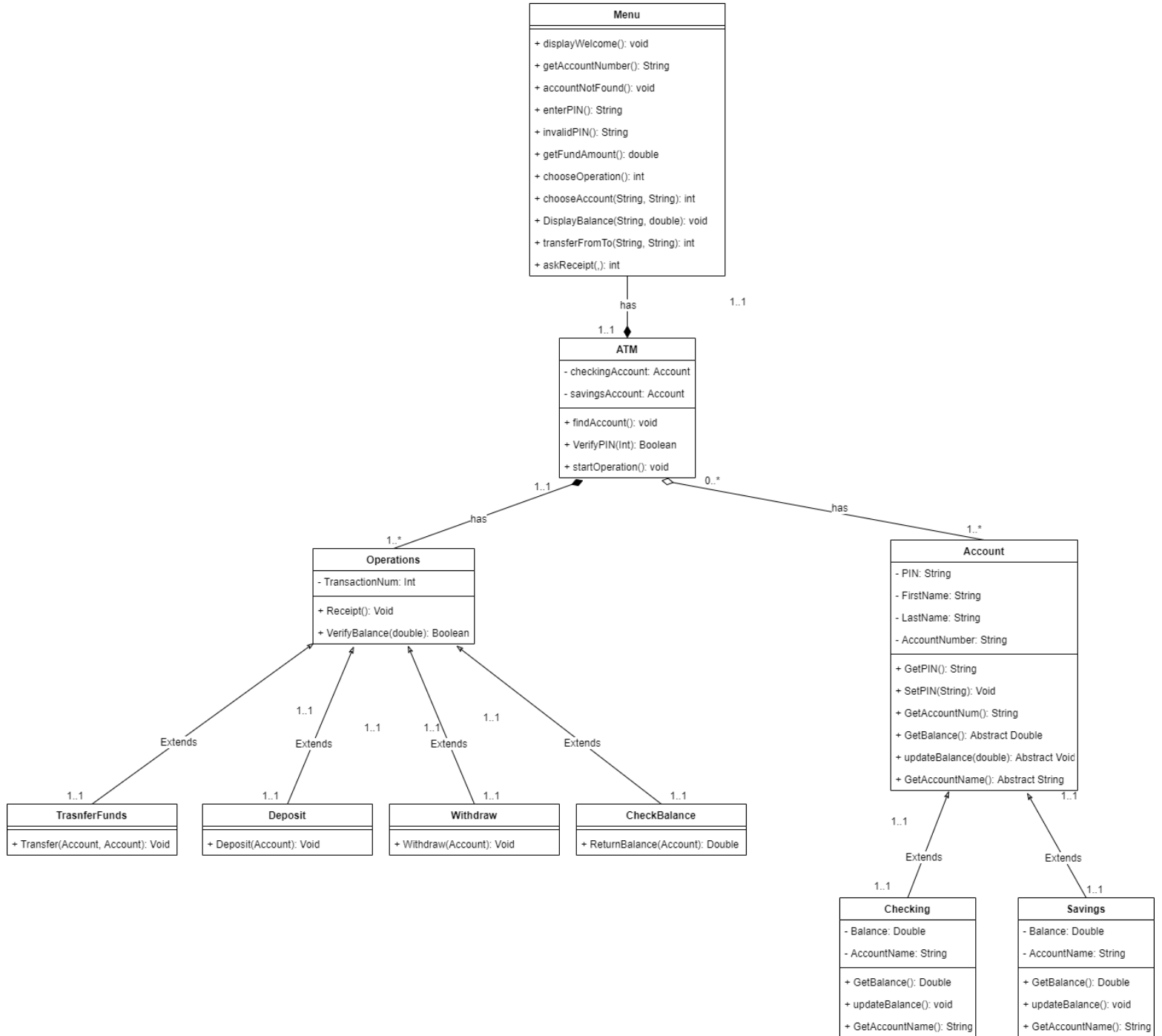
### **Withdraw:**

There will be a method that takes the account balance and verifies that the account has enough funds to withdraw from.

There will be a method that removes the funds after verification.

**CheckBalance:** This class will have a method that takes the account and returns its current balance.

## b. Detailed class diagrams



c. CRC

Menu	
<ul style="list-style-type: none"> <li>• Display welcome message</li> <li>• Getting the account number</li> <li>• Asking the customer to enter PIN</li> <li>• Displaying an error if the account the customer entered is incorrect</li> <li>• Choosing the operation to execute</li> <li>• Ask the user which account to choose from (Savings/Checkings)</li> <li>• Asking the customer if he/she wants to receipt</li> </ul>	<ul style="list-style-type: none"> <li>• ATM</li> <li>• Account</li> </ul>
Operation	
<ul style="list-style-type: none"> <li>• The operations class will have a private variable that contains the transactionNum. The operation class will also be able to print the receipt of the transaction and verifying if the customer has enough funds in their account</li> </ul>	<ul style="list-style-type: none"> <li>• TransferFunds</li> <li>• Deposit</li> <li>• Withdraw</li> <li>• CheckBalance</li> </ul>
TransferFunds	
<ul style="list-style-type: none"> <li>• The TransferFunds class will allow the customer to transfer funds from one account to another.</li> </ul>	<ul style="list-style-type: none"> <li>• Account</li> </ul>
Deposit	
<ul style="list-style-type: none"> <li>• The Deposit class will allow the customer to deposit money into their checking or savings account</li> </ul>	<ul style="list-style-type: none"> <li>• Account</li> </ul>
Withdraw	
<ul style="list-style-type: none"> <li>• The Withdraw class will allow the customer to withdraw money from their checking or savings account.</li> </ul>	<ul style="list-style-type: none"> <li>• Account</li> </ul>



<b>CheckBalance</b>	
<ul style="list-style-type: none"> <li>The CheckBalance class will allow the customer to check their checking or savings balance.</li> </ul>	<ul style="list-style-type: none"> <li>Account</li> </ul>

<b>Account</b>	
<ul style="list-style-type: none"> <li>Getting the PIN on the current account</li> <li>Setting the PIN for the account</li> <li>Getting account number</li> <li>Getting the balance on the checking or savings account</li> <li>Updating the balance on the checking or savings account</li> <li>Getting the name on the account</li> </ul>	<ul style="list-style-type: none"> <li>Checking</li> <li>Saving</li> </ul>

<b>Checking</b>	
<ul style="list-style-type: none"> <li>Updates the balance of the checking account</li> </ul>	<ul style="list-style-type: none"> <li>Account</li> </ul>

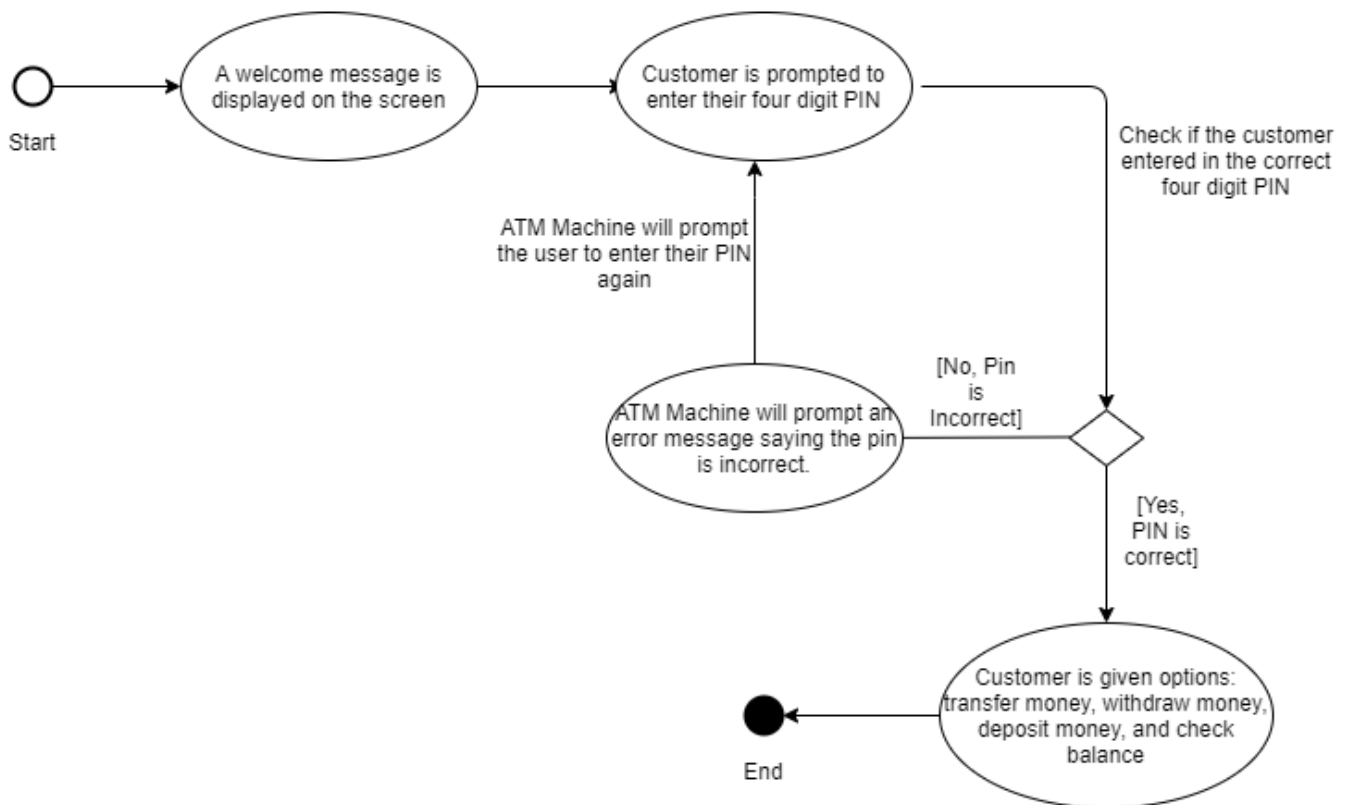
<b>Savings</b>	
<ul style="list-style-type: none"> <li>Updates the balance of the savings account</li> </ul>	<ul style="list-style-type: none"> <li>Account</li> </ul>

d. Activity diagram

The activity diagram is an important diagram that is used to describe the dynamic aspects of our ATM machine. This diagram will be displaying particular operations in the system such as the overall ATM system (how it flows), how the system deposits, how the system processes the

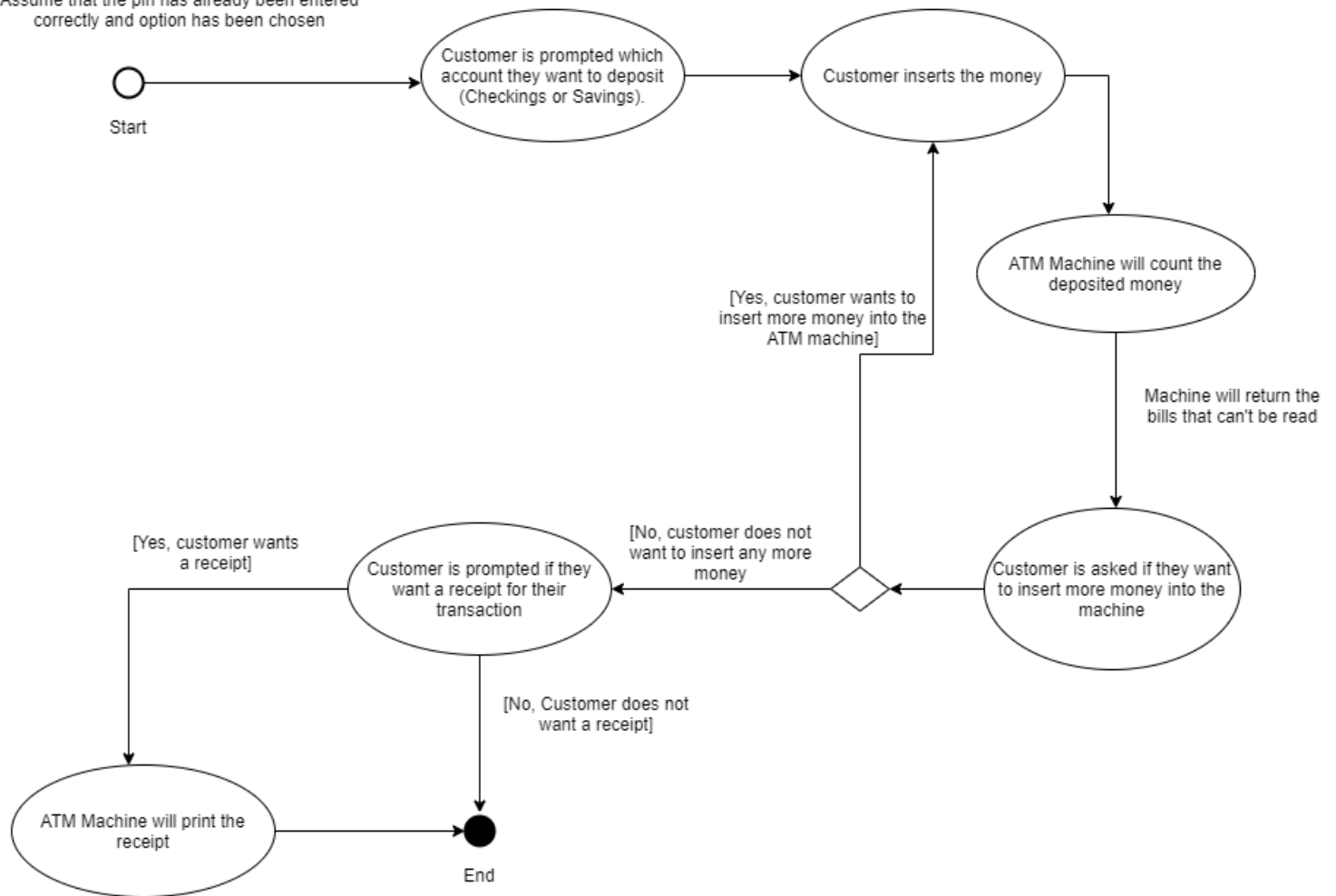
withdrawal process, how the system prints out the balance, and how the system will transfer funds from one account to another. The activity diagram will display a flowchart consisting of activities that are performed by the system. An example of this is the activity diagram of the ATM machine. First it displays a welcome message and asks the user to enter a four digit PIN. The system will then check if the four digit PIN is correct. If correct, the customer is given 4 options: transfer money, withdraw money, deposit money, and check balance. However, if PIN is incorrect, the system will prompt the user to reenter the PIN.

#### Verification Diagram



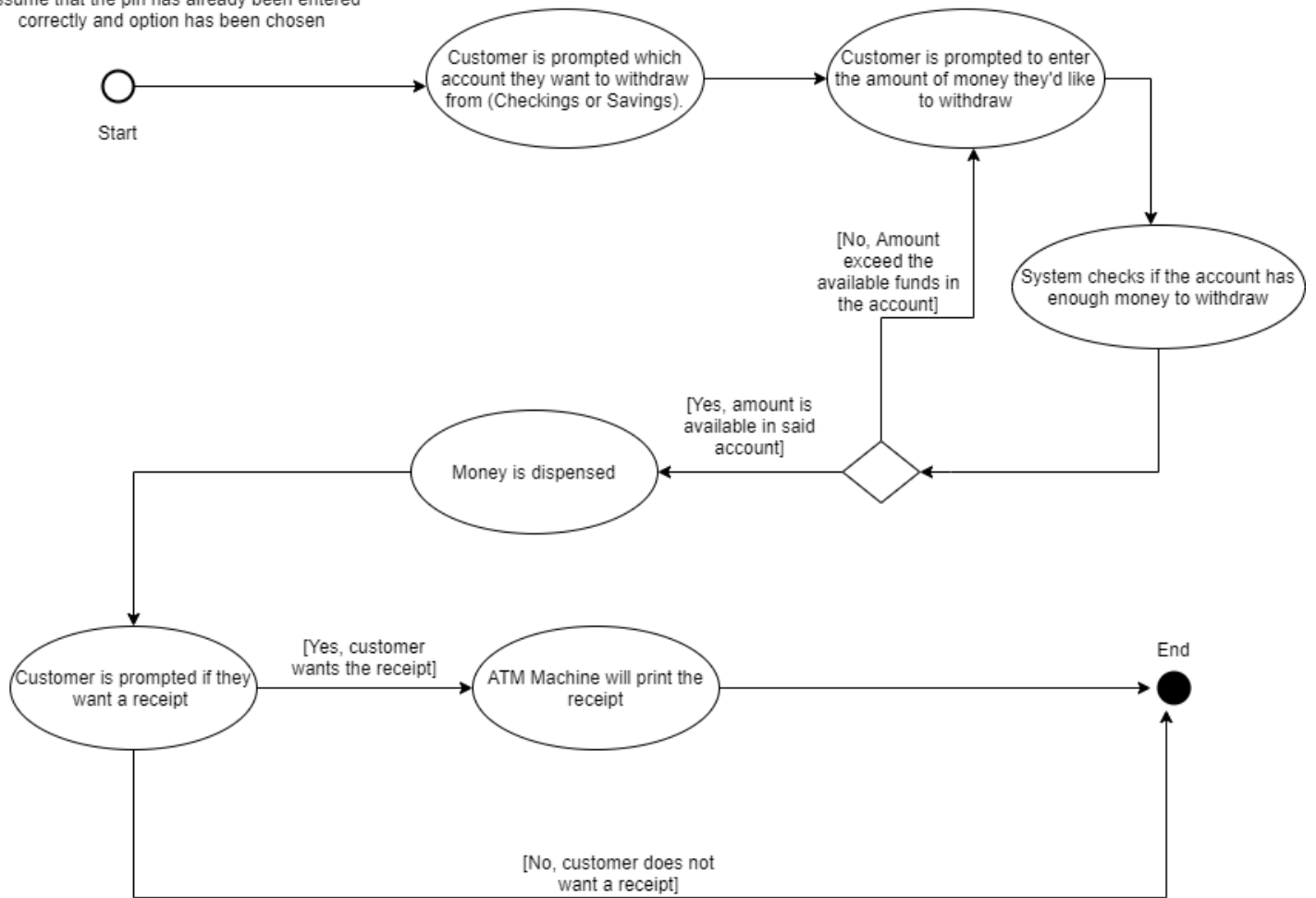
### Deposit Diagram

Assume that the pin has already been entered correctly and option has been chosen

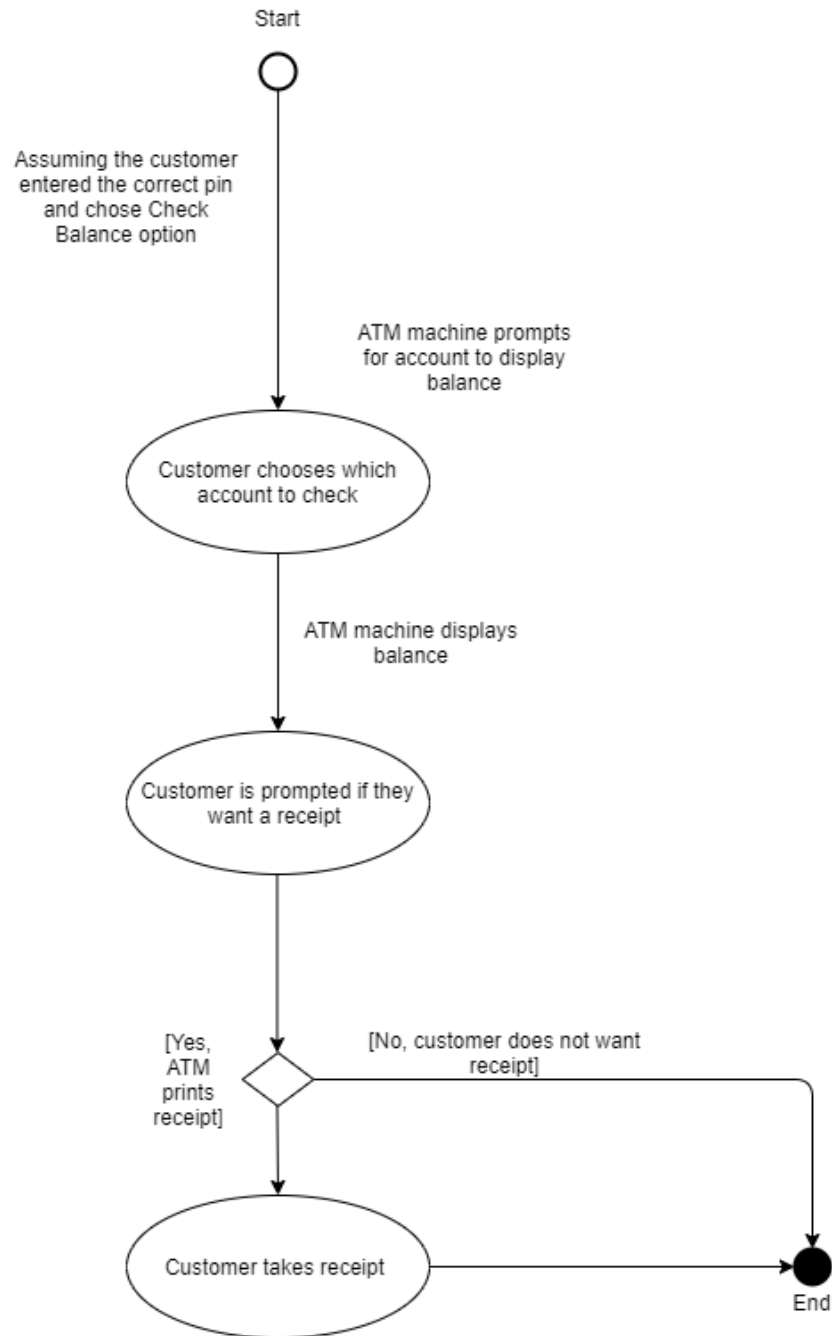


### Withdraw Diagram

Assume that the pin has already been entered correctly and option has been chosen



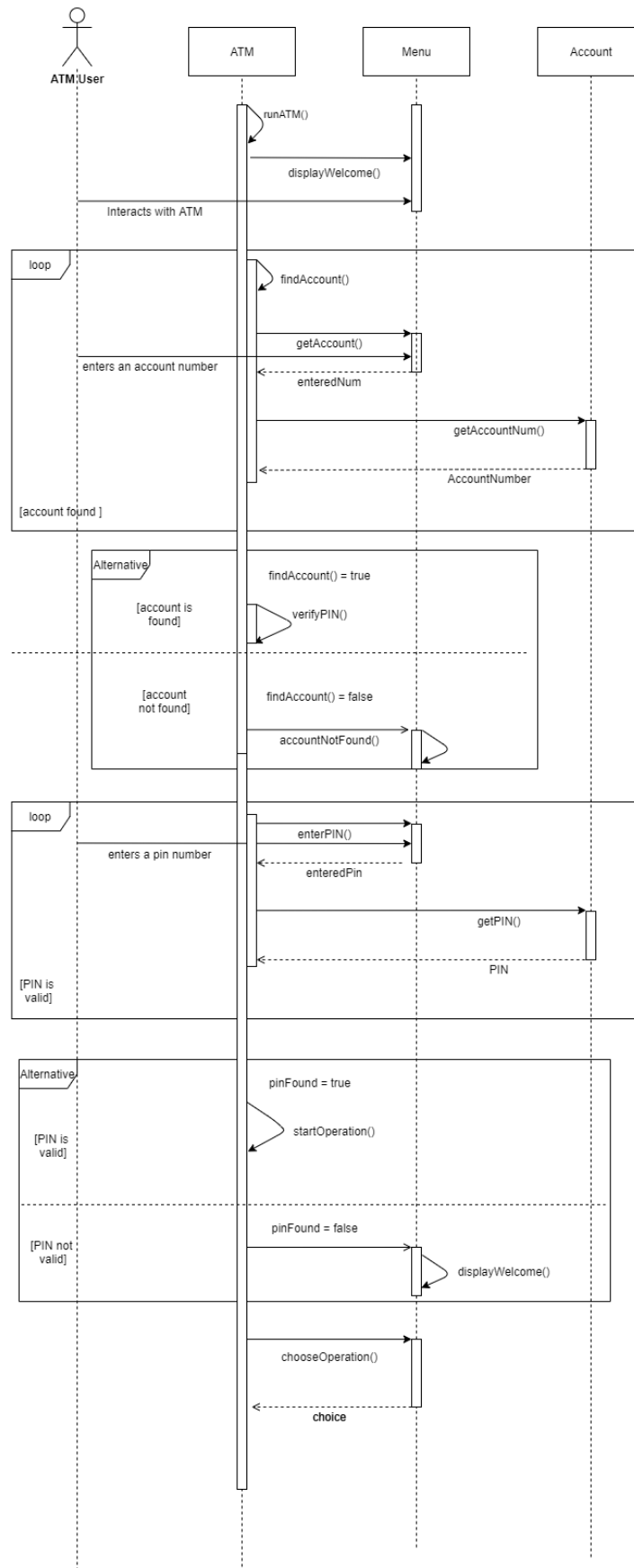
## Check Balance Diagram



e. Sequence diagram

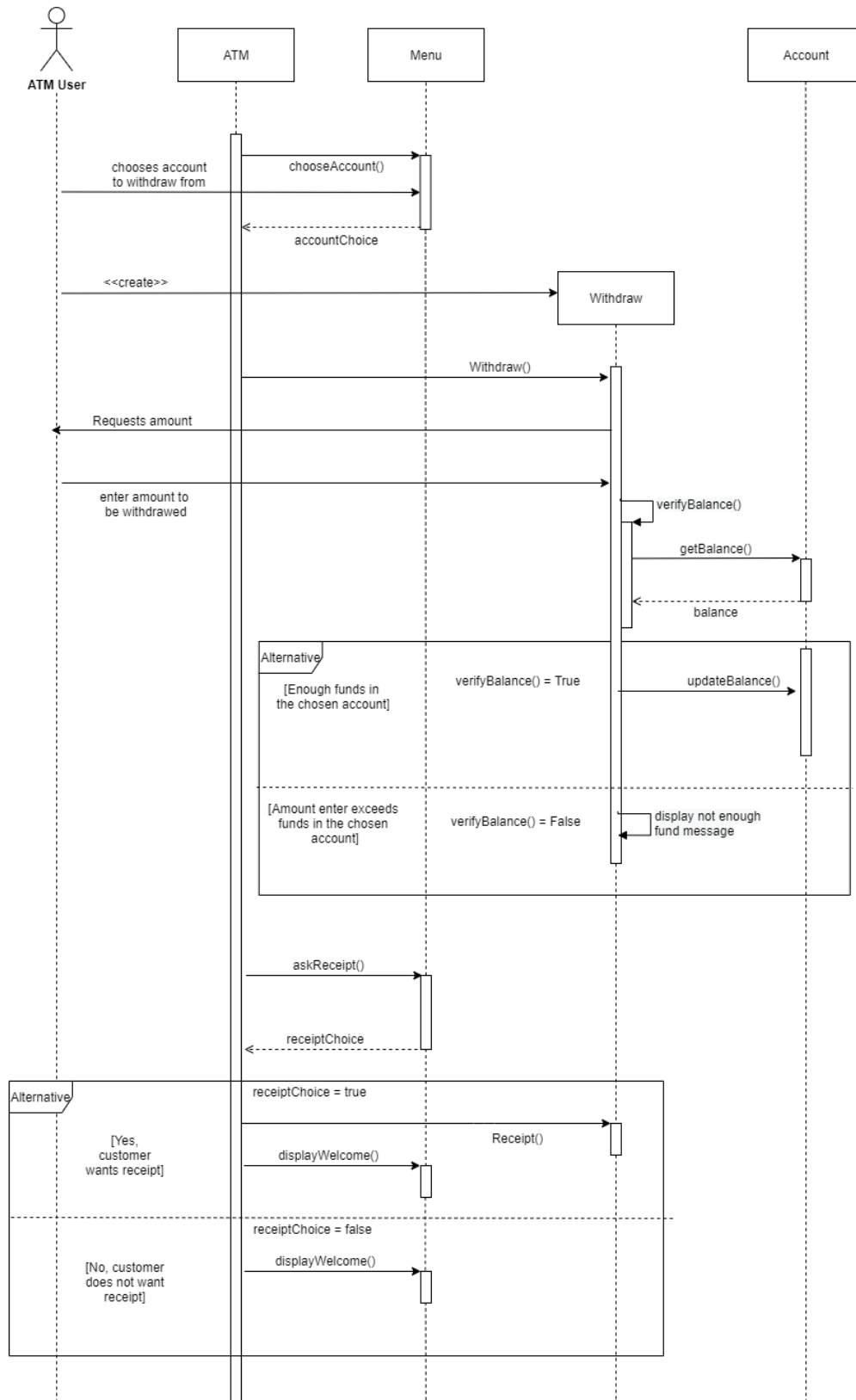
The sequence diagram starts with the verification process. The ATM object has the Menu object ask the user to enter an account number and a PIN. The ATM object uses this information to search for the matching account. Once the account is validated the Menu has the user choose an operation. Before the Operation object is created the Menu asks which account to perform the operation on. The Operation object may ask the user to enter a fund amount to update the account balance with. Once the operation is complete the Menu will ask the user if they want a receipt, if yes then the Operation will display it.

## Verification Sequence



# Withdraw Sequence

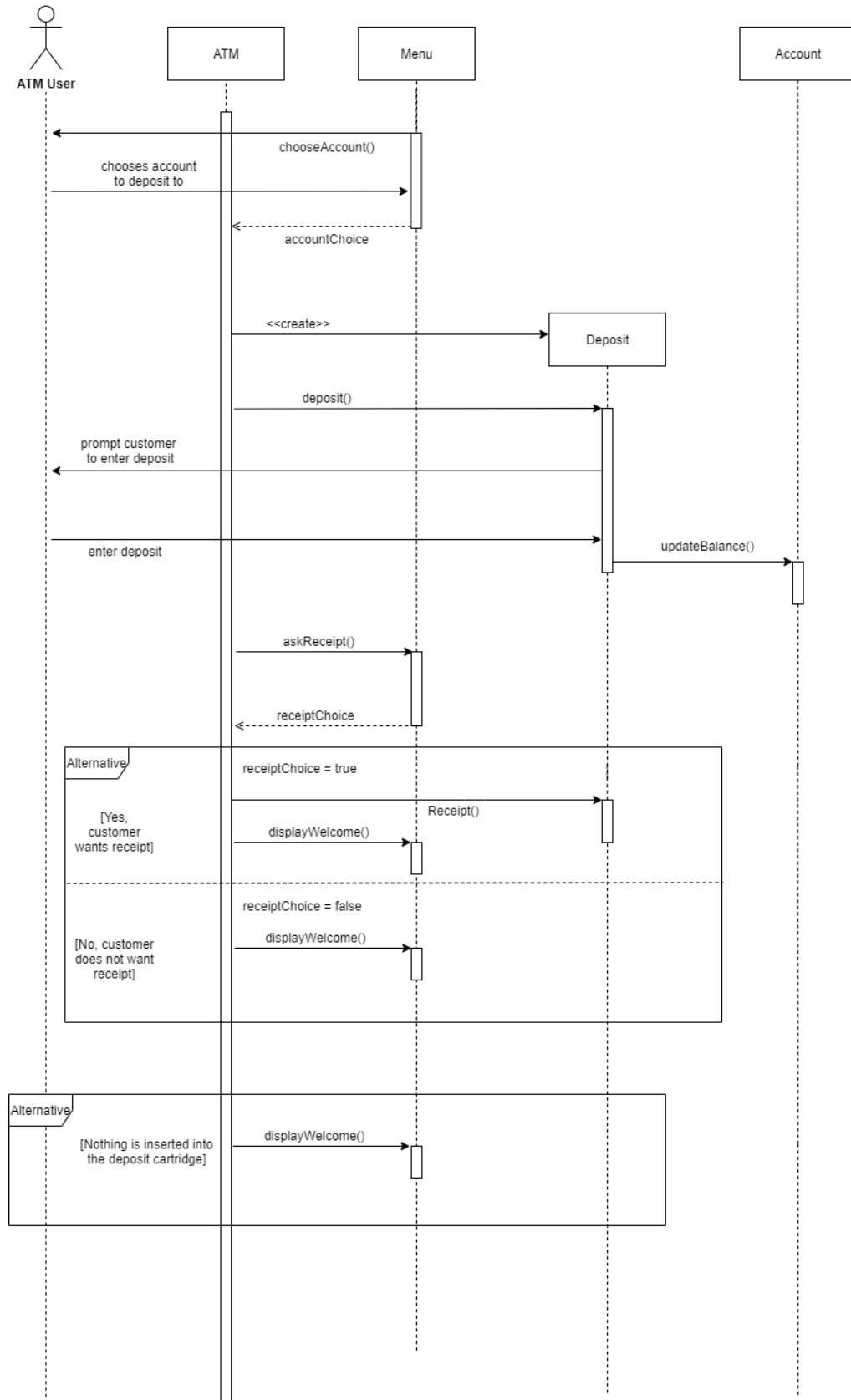
Assume Withdraw already selected





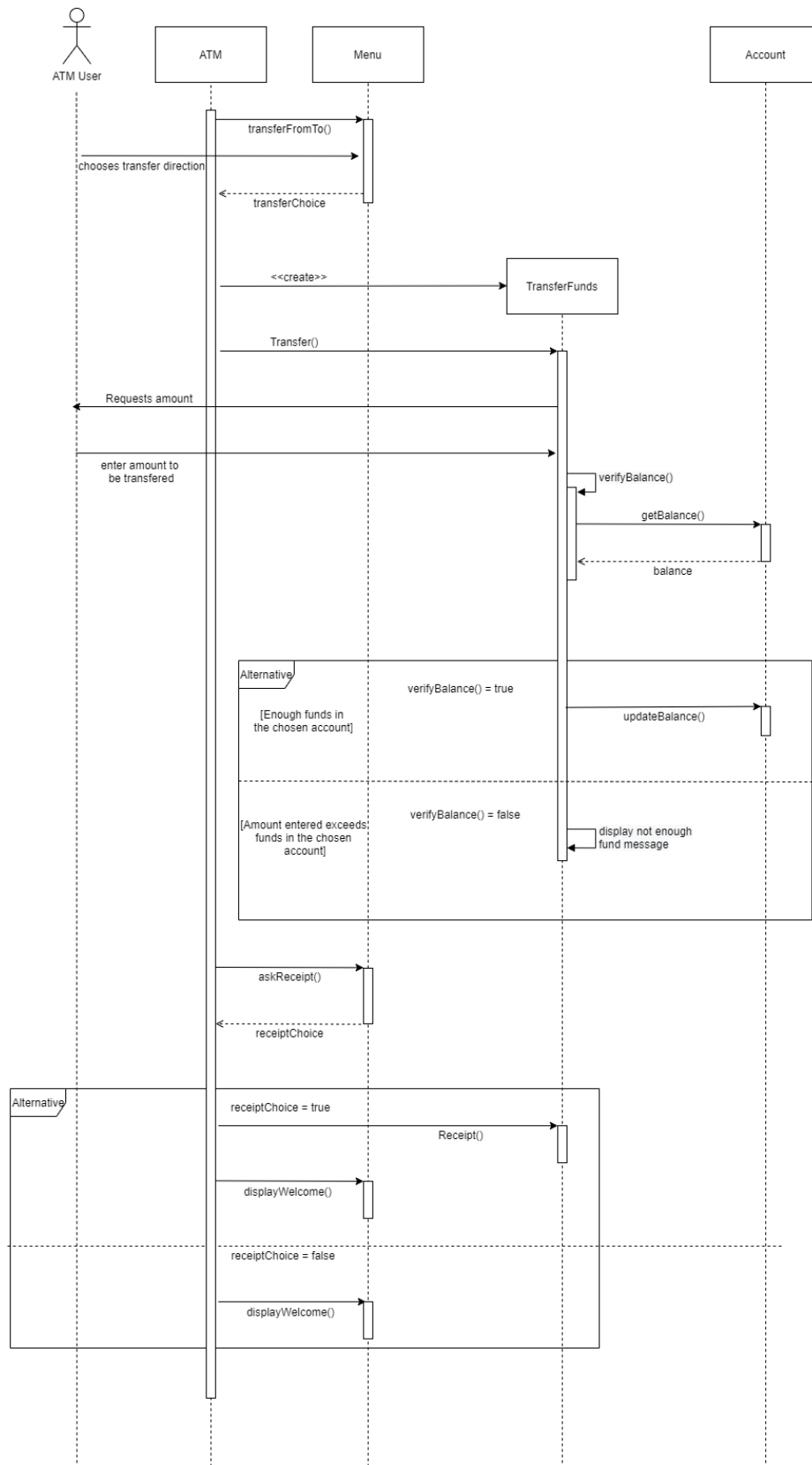
# Deposit Sequence

Assume Deposit already selected



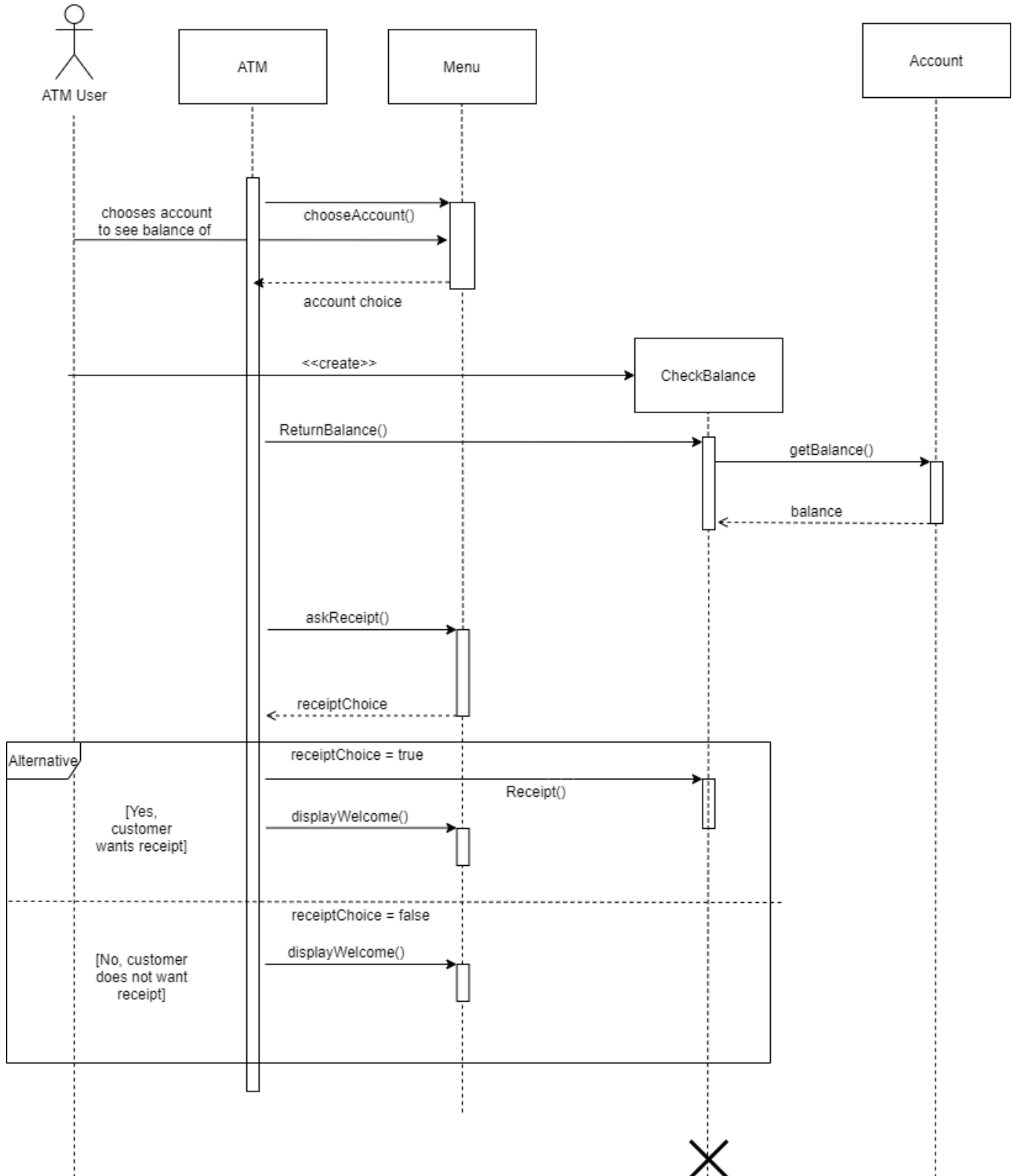
## Transfer Sequence

Assume TrasnferFunds already selected



# Check Balance Sequence

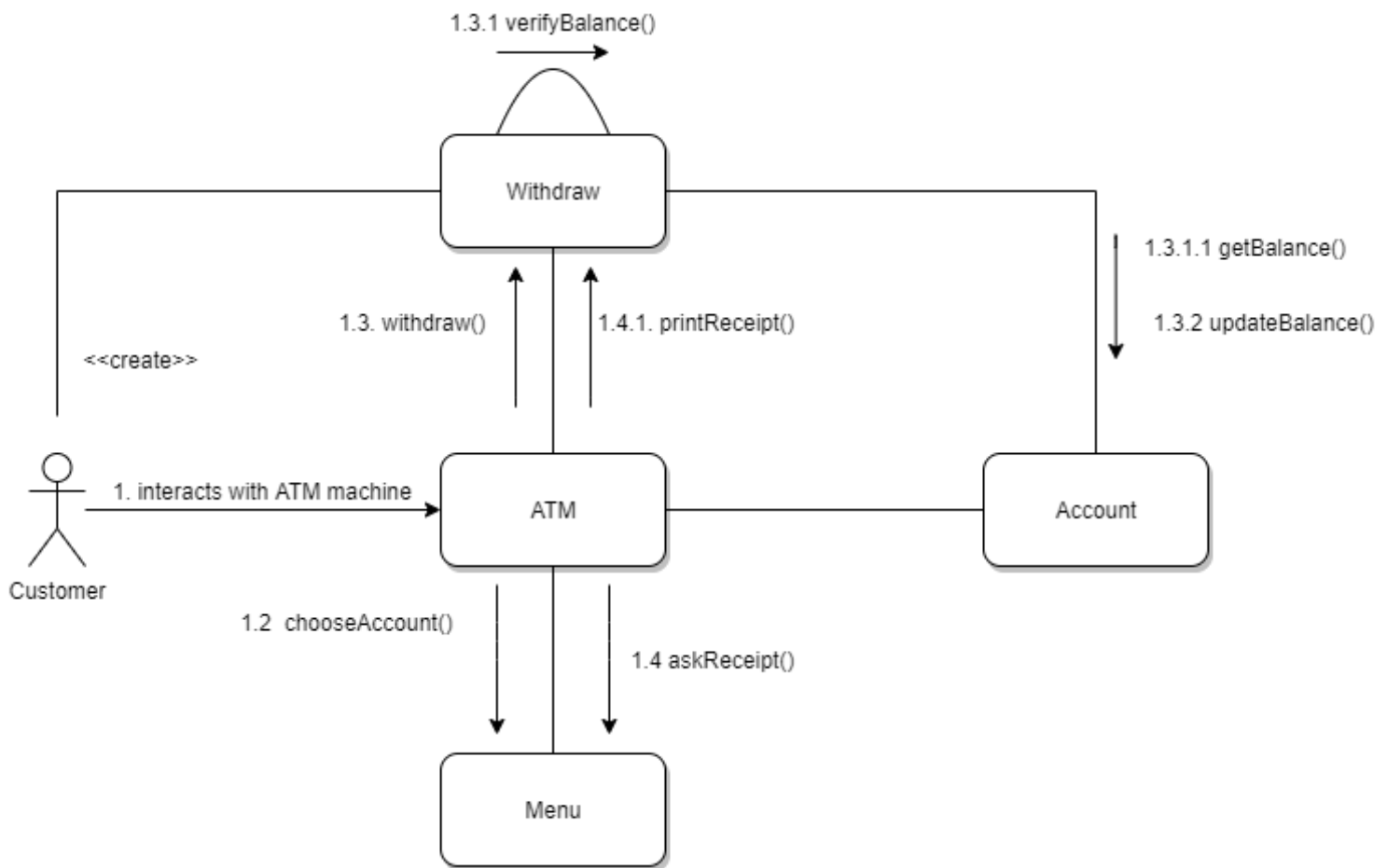
Assume CheckBalance already selected



f. Collaboration diagram

The collaboration diagram is a diagram that shows objects and their links which can contain simple class instances and class utility instances. In this collaboration diagram, our team focused on designing the withdrawal process collaboration diagram. In this diagram, it will provide a view of the interaction or structural relationships that occur between object and object like entities. For this example, the first thing that happens is the user interacts with the machine. The system will prompt the user to choose which account they want to withdraw from and the amount of money they'd like to withdraw. Then the ATM machine will check if the user has enough money to withdraw. If enough, it will withdraw, update balance, and then print the receipt.

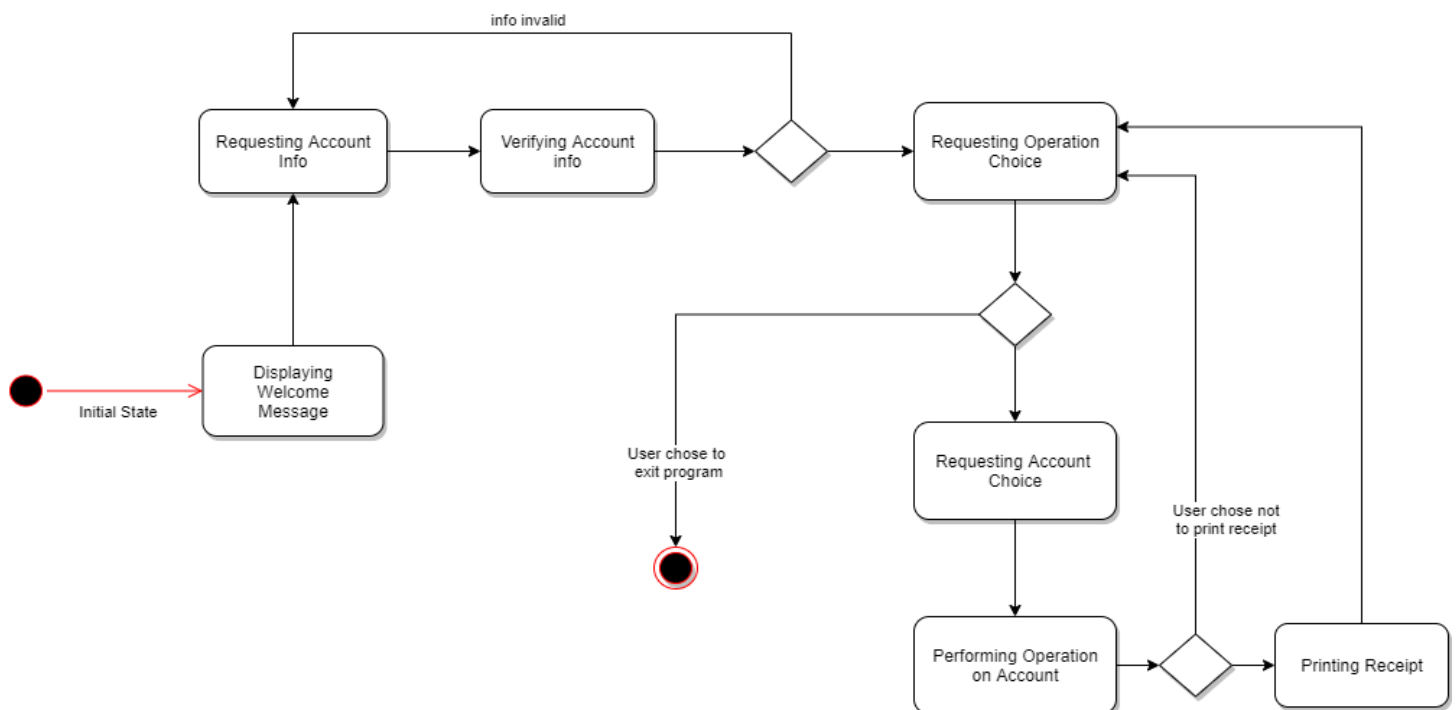
**Withdraw Collaboration Diagram**



g. State diagram

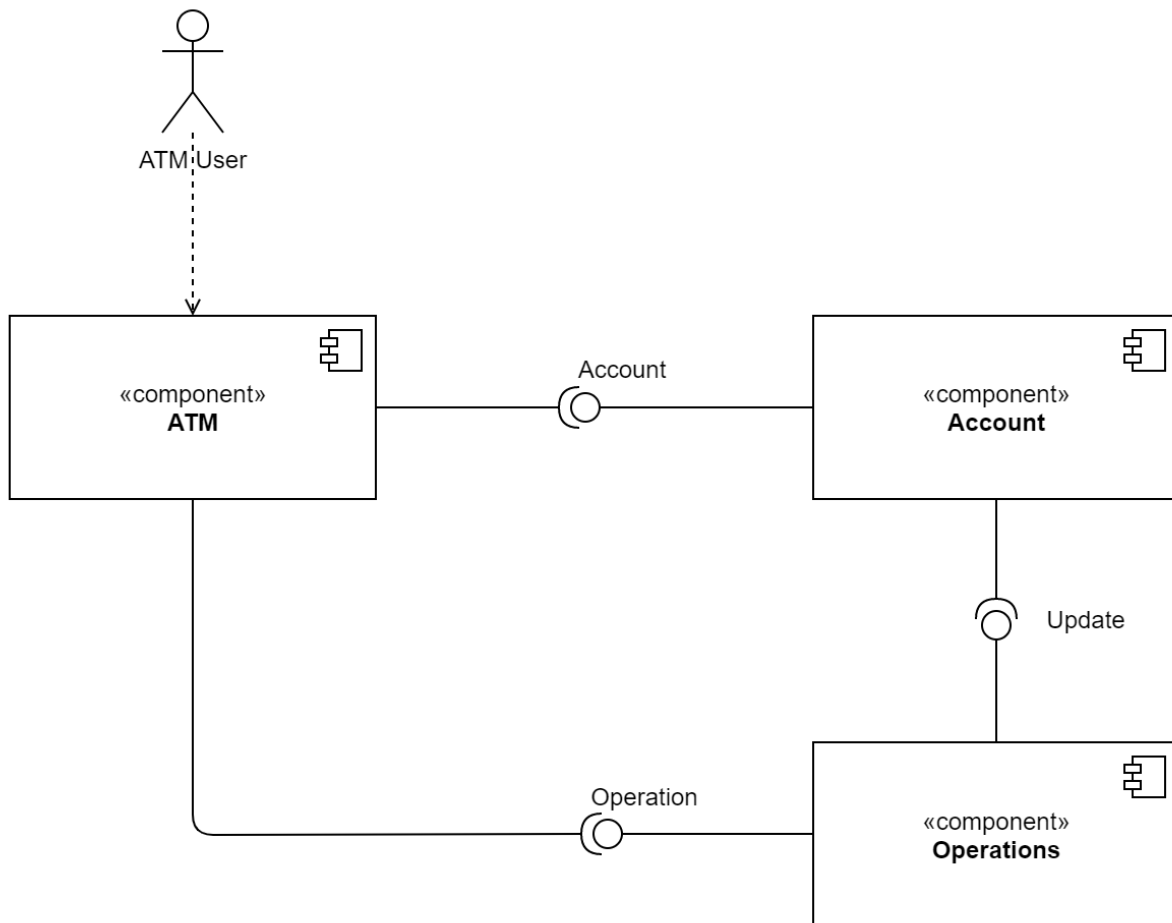
The state diagram is a diagram that describes different states of components in a system. The diagram describes the flow control from one state to another. As a group, we decided to do the state diagram on the ATM machine. We modeled the dynamic aspect of the system, described each state, and modeled the states of the object. During the initial state, the ATM machine displays a welcome message, then it requests account information. Once account information is entered, it will verify if the account information is valid. If invalid, it will ask the customer to re enter their information. The system will then ask the user to enter what operation that they want to be done and which account they want the operations to operate on. Once completed, the customer would be asked if they want a receipt.

ATM State Diagram



## 5. Component Design

Component diagram is a diagram that provides a simplified high order view of a large system. In this diagram, we showed the structural relationships between the components of the ATM machine. The three components we have are the ATM, Account, and Operations.



ATM component: ATM and the Menu class.

Account component: Account, Savings, and Checking class

Operations component: Operations, TransferFunds, Deposit, Withdraw, and CheckBalance class

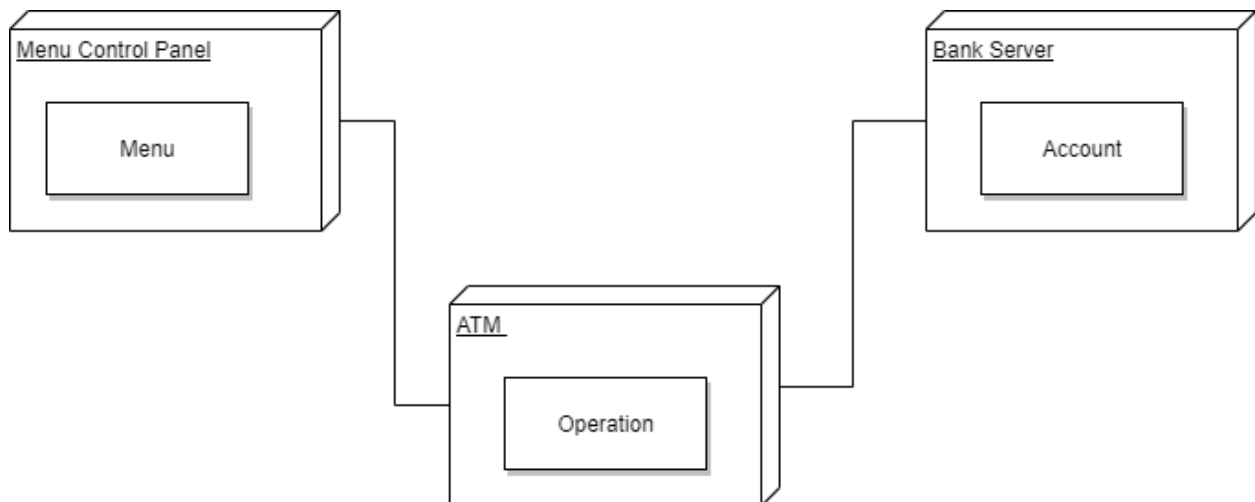
## 6. User Interface Design

In the ATM the Menu object is mainly used for user input and output. It displays messages requesting account information and user choices. The choices that the Menu asks for include which operations to perform, which account type to use, and if the user wants a receipt. The Menu will display an error message whenever the user enters invalid information. For modularity, each type of Operation has its own input/output messages. They may ask for an amount of funds to use in the operation. If the Operation received an invalid fund amount then they display an error message. The Operation is also in charge of displaying the receipt.

*For the images of the interface please go to Runtime Output.*

## 7. Deployment Design

The Menu Control Panel represents the interface that communicates between the user and the ATM. The Bank Server is the device that the ATM communicates with to access user accounts. The ATM itself uses the provided information to perform operations on the user's account.



## 8. Coding

### //ATM Main

```
//-----  
  
package project343;  
  
public class ATM_Main {  
  
    public static void main(String[] args) {  
  
        ATM atm = new ATM();  
  
        atm.runATM();  
  
    }  
  
}
```

### //ATM

```
//-----  
  
package project343;  
  
import java.util.ArrayList;  
  
  
public class ATM {  
  
  
  
    private static ArrayList<Checking> checkingAccountList;  
  
    private static ArrayList<Savings> savingsAccountList;  
  
    private static Checking currentCheckingAcct;  
  
    private static Savings currentSavingsAcct;  
  
    private static String currentPIN;
```



```

private static Menu menu = new Menu();

ATM(){

    checkingAccountList = new ArrayList<Checking>();

    savingsAccountList = new ArrayList<Savings>();

    currentCheckingAcct = null;

    currentSavingsAcct = null;

}

void runATM() {

    /***creating testing accounts-----

    checkingAccountList.add(new Checking("1111", "Doe", "John", "12345678",
"Checking"));

    savingsAccountList.add(new Savings("1111", "Doe", "John", "12345678",
"Savings"));

    checkingAccountList.get(0).updateBalance(1200);

    savingsAccountList.get(0).updateBalance(2400);

    checkingAccountList.add(new Checking("2222", "Vera", "Alexis", "11112222",
"Checking"));

    savingsAccountList.add(new Savings("2222", "Vera", "Alexis", "11112222",
"Savings"));

```

```

        checkingAccountList.get(1).updateBalance(3500);

        savingsAccountList.get(1).updateBalance(1600);


        checkingAccountList.add(new Checking("3333", "Shunner", "Ken", "87654321",
"Checking"));

        savingsAccountList.add(new Savings("3333", "Shunner", "Ken", "87654321",
"Savings"));

        checkingAccountList.get(2).updateBalance(600);

        savingsAccountList.get(2).updateBalance(1500);

        /**/


        menu.displayWelcome();


//LOOKING FOR ACCOUNT-----

        while(!findAccount()) {

                menu.accountNotFound();

        }


        while(!verifyPIN()) {

                menu.invalidPIN();

        }


//STARTING OPERATIONS-----

        int choice = 0;

```

```

while(choice != 5) {

    //Getting operation choice from user-----

    choice = menu.chooseOperation();

    int accountChoice;

    //this is used by every operationg to show balance at end-----

    CheckBalance check = new CheckBalance();

    switch(choice) {

        case 1: //Transfer Funds -----
        -----

            //Requesting user to choose account to tranfers from---

            accountChoice =
menu.transferFromTo(currentCheckingAcct.getAccountName(),
currentSavingsAcct.getAccountName());

            //Creating operation---

            TransferFunds transfer = new TransferFunds();

            if(accountChoice == 1)

            {

                //Transferring from Checking to Savings---

```

```

currentSavingsAcct);

transfer.transfer(currentCheckingAcct,

check.ReturnBalance(currentCheckingAcct);

check.ReturnBalance(currentSavingsAcct);


// Ask for receipt

choice = menu.askReceipt();

if(choice == 1)

{

    transfer.Receipt(currentCheckingAcct);

}

}

else

{

    //Transferring from Savings to Checking---

transfer.transfer(currentSavingsAcct,

currentCheckingAcct);

check.ReturnBalance(currentCheckingAcct);

check.ReturnBalance(currentSavingsAcct);


// Ask for receipt

choice = menu.askReceipt();

if(choice == 1)

{

```

```

        transfer.Receipt(currentSavingsAcct);

    }

}

System.out.println("\nRetuirning to menu...");

break;

case 2: //Deposit -----
-----

    //Requesting user to choose account to perform operation on---

    accountChoice =
menu.chooseAccount(currentCheckingAcct.getAccountName(),
currentSavingsAcct.getAccountName());

    //Creating operation---

    Deposit depo = new Deposit();

    if(accountChoice == 1)

    {

        //Depositing into Checking--

        depo.deposit(currentCheckingAcct);

        check.ReturnBalance(currentCheckingAcct);

        // Ask for receipt

        choice = menu.askReceipt();

        if(choice == 1)

```

```

        {
            depo.Receipt(currentCheckingAcct);
        }
    }
else
{
    //Depositing into Savings--
    depo.deposit(currentSavingsAcct);
    check.ReturnBalance(currentSavingsAcct);
    //Ask for receipt
    choice = menu.askReceipt();
    if(choice == 1)
    {
        depo.Receipt(currentSavingsAcct);
    }
}

System.out.println("\nRetuirning to menu...");

break;

```

case 3: //Withdraw -----

-----  
 //Requesting user to choose account to perform operation on---

```

        accountChoice =
menu.chooseAccount(currentCheckingAcct.getAccountName(),
currentSavingsAcct.getAccountName());

        //Creating operation---

Withdraw with = new Withdraw();

if(accountChoice == 1)

{

        //Withdrawing from Checking---

with.withdraw(currentCheckingAcct);

check.ReturnBalance(currentCheckingAcct);

        // Ask for receipt

choice = menu.askReceipt();

if(choice == 1)

{

                with.Receipt(currentCheckingAcct);

        }

}

else

{

        //Withdrawing from Savings--

with.withdraw(currentSavingsAcct);

check.ReturnBalance(currentSavingsAcct);

        // Ask for receipt

```

```

        choice = menu.askReceipt();

        if(choice == 1)
        {
            with.Receipt(currentSavingsAcct);
        }
    }

    System.out.println("\nRetuirning to menu...");

    break;

case 4: //CheckBalance -----
-----

    //Requesting user to choose account to perform operation on---

    accountChoice =
menu.chooseAccount(currentCheckingAcct.getAccountName(),
currentSavingsAcct.getAccountName());

    if(accountChoice == 1)
    {

        //Showing Checking balance and info--

        check.ReturnBalance(currentCheckingAcct);

        // Ask for receipt

        choice = menu.askReceipt();

        if(choice == 1)

```



```

        {
            check.Receipt(currentCheckingAcct);
        }
    }
else
{
    //Showing Savings balance and info--
    check.ReturnBalance(currentSavingsAcct);

    // Ask for receipt
    choice = menu.askReceipt();
    if(choice == 1)
    {
        check.Receipt(currentSavingsAcct);
    }
}

System.out.println("\nRetuirning to menu...");

break;

case 5:

    //Exiting program---
    break;

```

```

        default:

            //User has entered an invalid choice---

            System.out.println("  Invalid Input!");

        }

    }

}

```

```

static Boolean findAccount() {

    boolean accountFound = false;

    String accountNum = menu.getAccountNumber();

    //Searching list of checking accounts----

    for (Account a: checkingAccountList) {

        if (a.getAccountNum().equals(accountNum)) {

            currentCheckingAcct = (Checking) a;

            accountFound = true;

            break;

        }

    }
}

```

```

    }

    if(!accountFound) {

        //account wasnt found with account number the user entered---

        return accountFound;

    }

    //Searching list of saving accounts----

    for (Account a: savingsAccountList) {

        if (a.getAccountNum().equals(accountNum)) {

            currentSavingsAcct = (Savings) a;

            break;

        }

    }

    return accountFound;

}

static Boolean verifyPIN() {

    currentPIN = menu.enterPIN();

    return(currentCheckingAcct.getPin().equals(currentPIN));

}

}

```

//Menu

//-----

package project343;

import java.util.Scanner;

class Menu {

    Scanner input = new Scanner(System.in);

    public void displayWelcome() {

        System.out.println("Welcome! Press Enter to begin.");

        input.nextLine();

    }

    public String getAccountNumber() {

        System.out.print("Enter Account Number: ");

        String number = input.next();

        return number;

```
}
```

```
public void accountNotFound(){  
    System.out.println("    The account number does not match any records!");  
}
```

```
public String enterPIN() {  
  
    System.out.print("Enter your pin. PIN: ");  
    String pin = input.next();  
    return pin;  
}
```

```
public void invalidPIN(){  
    System.out.println(" PIN does not match!");  
}
```

```
public double getFundAmount() {  
    System.out.print("Enter an amount: ");  
    double amount = input.nextDouble();  
    return amount;  
}
```

```

public int chooseOperation() {

    int selection;

    System.out.println("\n---Select an Option---");

    System.out.println("1. Transfer");

    System.out.println("2. Deposit");

    System.out.println("3. Withdraw");

    System.out.println("4. Check Balance");

    System.out.println("5. Exit");

    System.out.print("Choice: ");

    selection = input.nextInt();

    return selection;

}

```

```

public int chooseAccount(String a1, String a2) {

    System.out.println("\n--Choose an account--");

    System.out.println("1. " + a1);

    System.out.println("2. " + a2);

```

```

        System.out.print("Choice: ");

        int selection = input.nextInt();

        return selection;
    }

    public void displayBalance(String accountName, double balance) {
        System.out.println("\nAccount: " + accountName);
        System.out.println("Balance: " + balance);
    }

    public int transferFromTo(String a1, String a2) {

        int selection;

        System.out.println("\n--Choose Account to transfer from--");
        System.out.println("1. " + a1);
        System.out.println("2. " + a2);

        System.out.print("Choice: ");

```

```
        selection = input.nextInt();

        return selection;
    }

    public int askReceipt()
    {
        int selection;

        System.out.println("\nWould you like a receipt for this transaction?");
        System.out.println("Enter 1 for YES and 2 for NO");
        System.out.print("Choice: \n");

        selection = input.nextInt();

        return selection;
    }
}
```



//Account

//-----

package project343;

abstract class Account

{

    private String pin;

    private String lastName;

    private String firstName;

    private String accountNumber;

    Account()

    {

        pin = "";

        lastName = "";

        firstName = "";

        accountNumber = "";

```
}
```

```
Account(String p, String ln, String fn, String an)
```

```
{
```

```
    pin = p;
```

```
    lastName = ln;
```

```
    firstName = fn;
```

```
    accountNumber = an;
```

```
}
```

```
public String getPin(){
```

```
    return pin;
```

```
}
```

```
public void setPin(String p){
```

```
    pin = p;
```

```
}
```

```
public String getAccountNum(){
```

```
    return accountNumber;
```

```
}
```

```
public abstract double getBalance();
```

```
public abstract void updateBalance(double difference);

public abstract String getAccountName();

}
```

**//Savings**

//-----

```
package project343;
```

```
public class Savings extends Account{
```

```
    private double balance;
```

```
    private String accountName;
```

```
    Savings(String p, String ln, String fn, String an, String name){
```

```
        super(p, ln, fn, an);
```

```
        accountName = name;
```

```
        balance = 0;
    }

    public double getBalance(){
        return balance;
    }

    public void updateBalance(double b){
        balance += b;
    }

    public String getAccountName(){
        return accountName;
    }
}
```

### **//Checking**

//-----

```
package project343;
```

```
class Checking extends Account{
```

```
private double balance;

private String accountName;

Checking(String p, String ln, String fn, String an, String name){

    super(p, ln, fn, an);

    accountName = name;

    balance = 0;

}

public double getBalance(){

    return balance;

}

public void updateBalance(double b){

    balance += b;

}

public String getAccountName(){

    return accountName;

}
```

```
}
```

### //Operations

```
//-----
```

```
package project343;
```

```
import java.text.DateFormat;
```

```
import java.text.SimpleDateFormat;
```

```
import java.util.Date;
```

```
public class Operations {
```

```
    void Receipt(Account acc)
```

```
    {
```

```
        System.out.println("\nBEACH BANK");
```

```
        System.out.println("-----");
```

```
        DateFormat date_display = new SimpleDateFormat("MM/dd/yyyy");
```

```
        DateFormat time_display = new SimpleDateFormat("hh:mm a");
```

```
        Date date = new Date();
```

```
        Date time = new Date();
```

```
        System.out.println("Date: " + date_display.format(date));
```

```
        System.out.println("Time: " + time_display.format(time));
```

```
        String lastFour = acc.getAccountNum().substring(acc.getAccountNum().length() -  
4);
```

```
        System.out.println("Account Number: " + "XXXX" + lastFour);
```

```
        int min = 100000;
```

```

        int max = 999999;

        int range = (max - min) + 1;

        int transNum = (int)(Math.random() * range) + min;

        System.out.println("Transaction Num: " + transNum);

        System.out.println("Account Balance: $" + acc.getBalance());

    }

    boolean verifyBalance(Account acc, double balance)
    {

        if(acc.getBalance() >= balance)
        {

            return true;

        }

        return false;

    }

}

//CheckBalance

//-----

```

```
package project343;
```

```
public class CheckBalance extends Operations
```

```
{
```

```
    void ReturnBalance(Account acc)
```

```
    {
```

```
        System.out.println("\nAccount: " + acc.getAccountName());
```

```
        System.out.println("Current balance in the account: $" + acc.getBalance());
```

```
    }
```

```
    void Receipt(Account acc)
```

```
    {
```

```
        super.Receipt(acc);
```

```
        System.out.println("Transaction Check Balance From: " +  
acc.getAccountName());
```

```
    }
```

```
}
```

```
//Deposit
```

```
//-----
```

```
package project343;
```

```
import java.util.Scanner;
```



```

public class Deposit extends Operations
{
    void deposit(Account acc)
    {
        Scanner input = new Scanner(System.in);

        int amountToDeposit;

        System.out.print("Enter the amount of money you want to deposit: $");

        amountToDeposit = input.nextInt();

        acc.updateBalance(amountToDeposit);

        System.out.println("\n$" + amountToDeposit + " deposited into account number "
+ acc.getAccountNum());
    }

    void Receipt(Account acc)
    {
        super.Receipt(acc);

        System.out.println("Transaction Deposit From: " + acc.getAccountName());
    }
}

```

**//TransferFunds**

//-----

```

package project343;

import java.util.Scanner;

public class TransferFunds extends Operations
{
    void transfer(Account acc1, Account acc2)
    {
        Scanner input = new Scanner(System.in);

        int amountToTransfer;

        System.out.print("Enter the amount of money you want to transfer: $");

        amountToTransfer = input.nextInt();

        if(verifyBalance(acc1, amountToTransfer))
        {
            acc2.updateBalance(amountToTransfer); //Adds the balance from
account1 to account2 balance

            acc1.updateBalance(-amountToTransfer); //Subtracts the balance from
account1

        }
        else
        {
            System.out.println("Invalid balance. Not enough funds!");
        }
    }
}

```

```

        }

    }

    void Receipt(Account acc)
    {
        super.Receipt(acc);
        System.out.println("Transaction Transfer From: " + acc.getAccountName());
    }
}

```

### **//Withdraw**

```

//-----

package project343;

import java.util.Scanner;

public class Withdraw extends Operations
{
    void withdraw(Account acc)
    {
        Scanner input = new Scanner(System.in);
        int amountToWithdraw;
    }
}

```

```

        System.out.print("Enter the amount of money you want to withdraw: $");

        amountToWithdraw = input.nextInt();

        if(verifyBalance(acc, amountToWithdraw))
        {

            acc.updateBalance(-amountToWithdraw);

            System.out.println("\n$" + amountToWithdraw + " withdraws from
account number " + acc.getAccountNum());

        }
        else
        {

            System.out.println("Not enough funds in the account!");

        }
    }

    void Receipt(Account acc)
    {

        super.Receipt(acc);

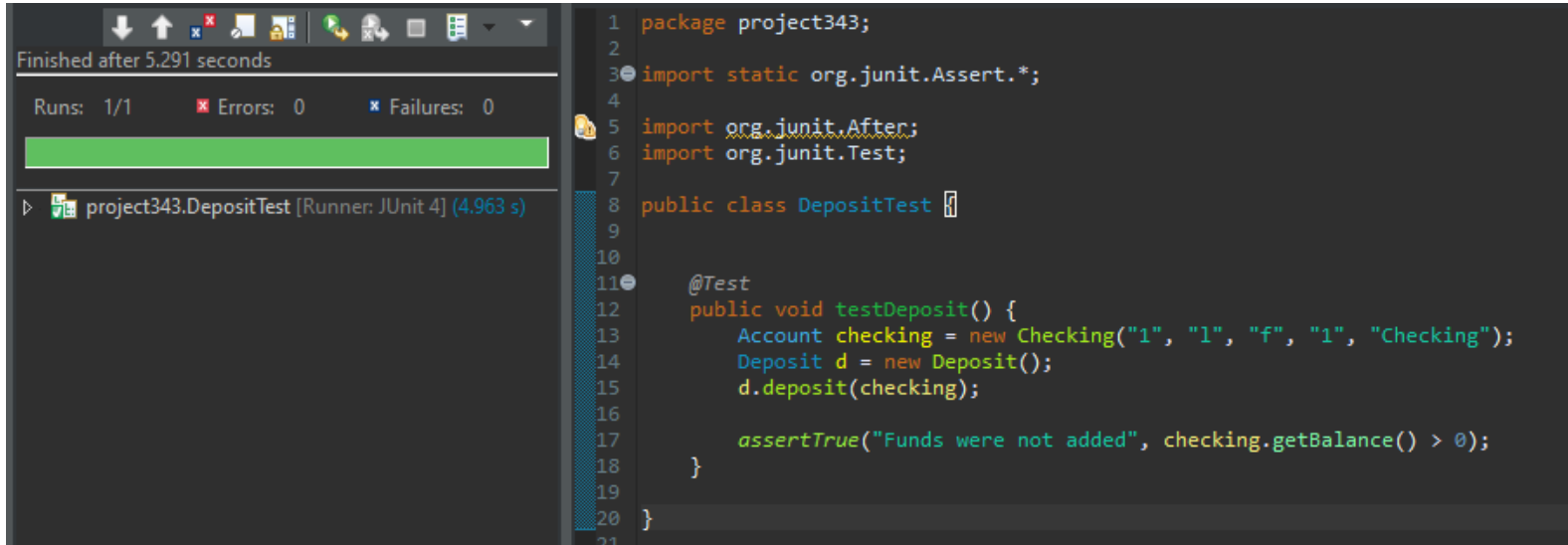
        System.out.println("Transaction Withdraw From: " + acc.getAccountName());

    }
}

```

## 9. Unit Testing

## Deposit Test



```
1 package project343;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.After;
6 import org.junit.Test;
7
8 public class DepositTest {
9
10
11     @Test
12     public void testDeposit() {
13         Account checking = new Checking("1", "1", "f", "1", "Checking");
14         Deposit d = new Deposit();
15         d.deposit(checking);
16
17         assertTrue("Funds were not added", checking.getBalance() > 0);
18     }
19
20 }
```

Finished after 5.291 seconds

Runs: 1/1   Errors: 0   Failures: 0

project343.DepositTest [Runner: JUnit 4] (4.963 s)

## Withdraw Test



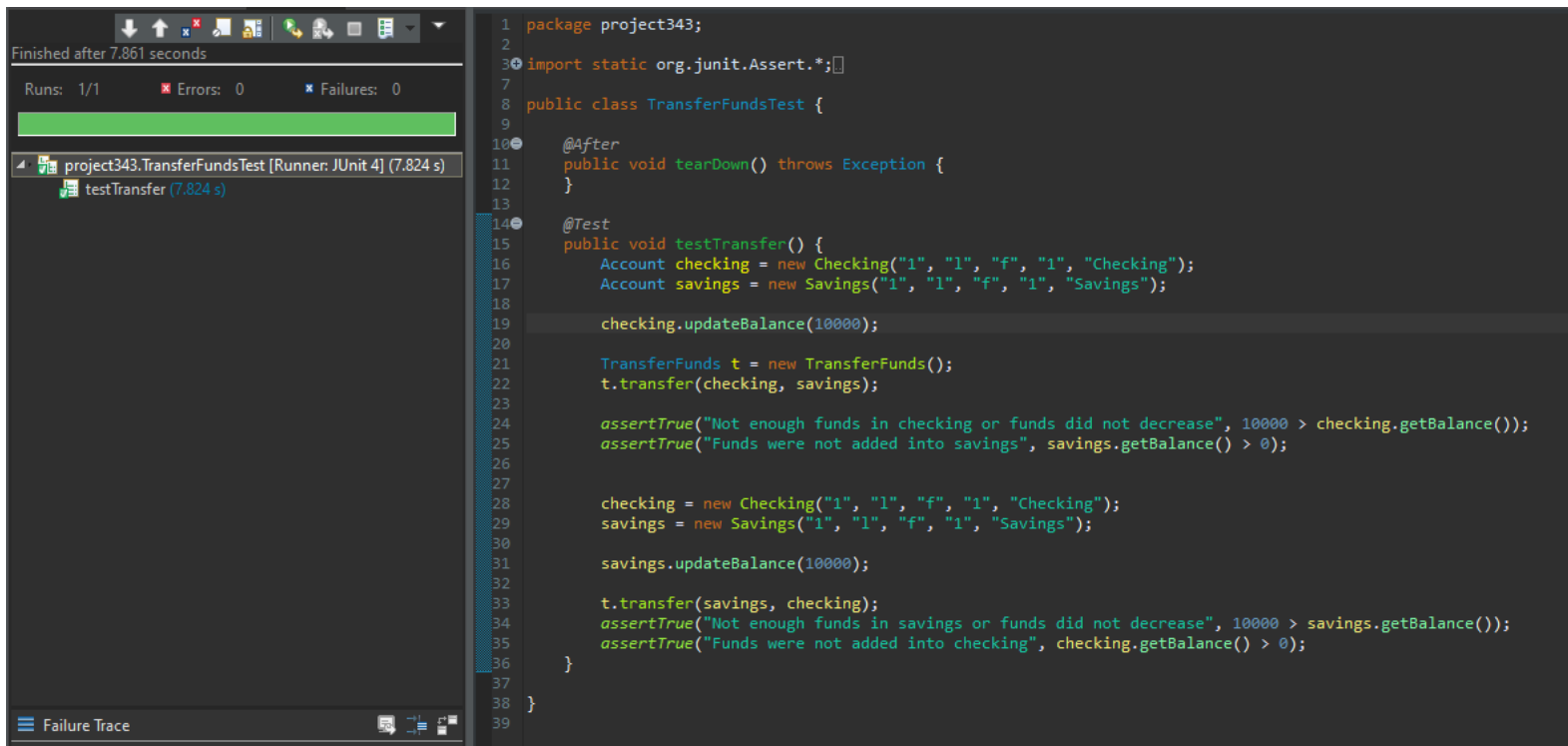
```
1 package project343;
2
3 import static org.junit.Assert.*;
4
5 public class WithdrawTest {
6
7
8     @Test
9     public void testWithdraw() {
10         Account checking = new Checking("1", "1", "f", "1", "Checking");
11         checking.updateBalance(10000);
12         Withdraw w = new Withdraw();
13         w.withdraw(checking);
14
15         assertTrue("Not enough funds in balance or funds did not decrease", 10000 > checking.getBalance());
16     }
17
18 }
```

Finished after 5.724 seconds

Runs: 1/1   Errors: 0   Failures: 0

project343.WithdrawTest [Runner: JUnit 4] (5.694 s)

## Transfer Test



## 10. Runtime output

### Correct Account Number and PIN

```

Welcome! Press Enter to begin.

Enter Account Number: 12345678
Enter your pin. PIN: 1111
|
---Select an Option---
1. Transfer
2. Deposit
3. Withdraw
4. Check Balance
5. Exit
Choice:

```

### Wrong Account Number or PIN

```
Welcome! Press Enter to begin.  
  
Enter Account Number: 1234  
|       The account number does not match any records!  
Enter Account Number:
```

```
Welcome! Press Enter to begin.  
  
Enter Account Number: 12345678  
Enter your pin. PIN: 1234  
|       PIN does not match!  
Enter your pin. PIN:
```

## Transfer

```
---Select an Option---  
1. Transfer  
2. Deposit  
3. Withdraw  
4. Check Balance  
5. Exit  
Choice: 1  
  
--Choose Account to transfer from--  
1. Checking  
2. Savings  
Choice: 1  
Enter the amount of money you want to transfer: $250  
  
Account: Checking  
Current balance in the account: $1200.0  
  
Account: Savings  
Current balance in the account: $2650.0  
  
Would you like a receipt for this transaction?  
Enter 1 for YES and 2 for NO  
Choice:
```

## Deposit

```
---Select an Option---
1. Transfer
2. Deposit
3. Withdraw
4. Check Balance
5. Exit
Choice: 2

--Choose an account--
1. Checking
2. Savings
Choice: 1
Enter the amount of money you want to deposit: $250

$250 deposited into account number 12345678

Account: Checking
Current balance in the account: $1450.0

Would you like a receipt for this transaction?
Enter 1 for YES and 2 for NO
Choice:
```

## Withdraw



```
---Select an Option---
1. Transfer
2. Deposit
3. Withdraw
4. Check Balance
5. Exit
Choice: 3

--Choose an account--
1. Checking
2. Savings
Choice: 2
Enter the amount of money you want to withdraw: $250
|
$250 withdraws from account number 12345678

Account: Savings
Current balance in the account: $2400.0

Would you like a receipt for this transaction?
Enter 1 for YES and 2 for NO
Choice:
```

## CheckBalance

```
---Select an Option---
1. Transfer
2. Deposit
3. Withdraw
4. Check Balance
5. Exit
Choice:
4

--Choose an account--
1. Checking
2. Savings
Choice: 1
|
Account: Checking
Current balance in the account: $1200.0

Would you like a receipt for this transaction?
Enter 1 for YES and 2 for NO
Choice:
```

## Receipt

```
Would you like a receipt for this transaction?
Enter 1 for YES and 2 for NO
Choice:
1

BEACH BANK
-----
Date: 05/02/2020
Time: 10:50 PM
Account Number: XXXX5678
Transaction Num: 561977
Account Balance: $1200.0
Transaction Check Balance From: Checking
```

## 11. Summary

Our team created an ATM machine that did four different operations using Java. The operations included check balance, transfer funds, deposit, and withdraw. The ATM machine would allow the customer to choose an operation and based on that operation the system would do system checks and execute. The first thing we created was the problem statement. Afterwards, we transitioned into creating use cases to help identify the actors, precondition, postcondition, description, alternative flow, and nonfunctional requirements. Once we understood the flow of how the ATM machine would work, we created our class diagrams and detailed class diagrams. Our class diagrams helped us understand what objects and variables we needed to declare to convert the ideas into coding. Through the use of different software engineer diagrams, we were able to get an accurate representation of how the ATM machine would flow. We then proceeded to code the program by using the diagrams we created.