

CECS 343 ATM System Cover Page

William Gusmanov, Isaiah Gudardo, Daniel Duong, and George Flores

CECS 343 ATM System Documentation

Instructor: Phuong Nguyen

MW: Lecture @ 5:30 -6:20 PM ECS 308

Lab @ 6:30 - 7:45 PM ECS 414

Due: May 8th, 2020

Team 1

Table Of Contents

Cover Page	1
Table Of Contents	2
Introduction	3
Problem Statement	3
The Initial Four Use Cases and Scenarios	3 - 9
Requirement Modeling	9 - 23
Low Level UML Diagram.....	10
High Level UML Diagram	11
Class Associations	12 - 13
Activity Diagrams	14 - 17
Sequence Diagrams	18 - 21
Collaboration Diagrams	22
State Diagrams	23
Component Design	24
User Interface Design	25 - 28
Deployment Design	28
Coding	22 - 33
Unit Testing	40 - 41
Runtime Output	42 - 44
Summary	44

Introduction

An “Automatic Teller Machine (ATM)” is a machine that allows customers to complete basic transactions without the aid of a representative. This project aims to create the software for an ATM machine while utilizing software development principles.

Problem Statement

Users often require the need to access information from their bank about their bank accounts.

These interactions often require the need of bank tellers. However, to reduce the need for such employees, an ATM machine allows customers to directly access their Bank account to be able to Display available funds, withdraw and deposit funds and the ability transfer funds between their accounts without the need of a bank, resulting in monetary return.

The Initial Use Cases

There are four use cases that are necessary and that is deposit funds, withdraw funds, transfer funds, and display funds.

USE CASE 1: Transfer Funds

USE CASE NAME: Transfer Funds	
SUMMARY	A Customer transfers money from an account to another account.
ACTOR	Customer
PRECONDITION	Customer enters their PIN number successfully and their bank account is successfully found. The user must select the “Transfer Funds” option.

SUMMARY	
1.	The ATM displays the list of accounts.
2.	The user selects a donor account. (Account to transfer from.)
3.	The ATM confirms the existence of donor account.
4.	The ATM asks for a transfer amount.
5.	ATM validates transfer amount.
6.	ATM asks for recipient account
7.	ATM confirms existence of recipient account.
8.	ATM asks for confirmation of transfer.
9.	Amount is transferred from donor to recipient.
POSTCONDITION	A transfer message is displayed and the program returns to the options menu.

Alternate Flow	
Insufficient Funds Flow 3	
1.	The system notifies the customer this is an invalid option.
2.	RESUME BACK TO 1.

Alternate Flow	
Invalid Account Flow 5	
1.	The system notifies the customer this is an invalid option.
2.	RESUME BACK TO 4.

Alternate Flow	
Invalid Account Flow 7	
1.	The system notifies the customer this account does not exist.
2.	EXIT TO MENU.

USE CASE 2: Display Funds

USE CASE NAME: Display Funds	
SUMMARY	The customer views the current balance in their checking or savings account.
ACTOR	Customer
PRECONDITION	The ATM pin must be entered correctly and the option selected must be "Display Balance".

SUMMARY	
1.	The system displays the list of accounts.
2.	The User selects an account to choose from.
3.	The user's balance is displayed for the selected account.
POSTCONDITION	The screen displays the Options menu.

Alternate Flow	
Invalid Account Flow	
1.	An exception is thrown if the account entered does not exist.

USE CASE 3: Deposit Funds

USE CASE NAME: Deposit Funds	
SUMMARY	The user deposits money into a selected account.
ACTOR	Customer
PRECONDITION	The PIN# must be correct prior to any transaction. The "Deposit Funds" option must be selected from the Options menu.

SUMMARY	
1.	The Screen displays the list of accounts.
2.	The customer will select one account from the list of accounts.
3.	The ATM prompts the user to input the amount to deposit.
4.	The ATM prints how much money was deposited as well which account it went into.
PostCondition	The funds are successfully added to the customer's checking or savings account. The Options menu is displayed.

Alternate Flow	
Invalid Account Flow	
2.	Displays account not found and returns to the options menu.

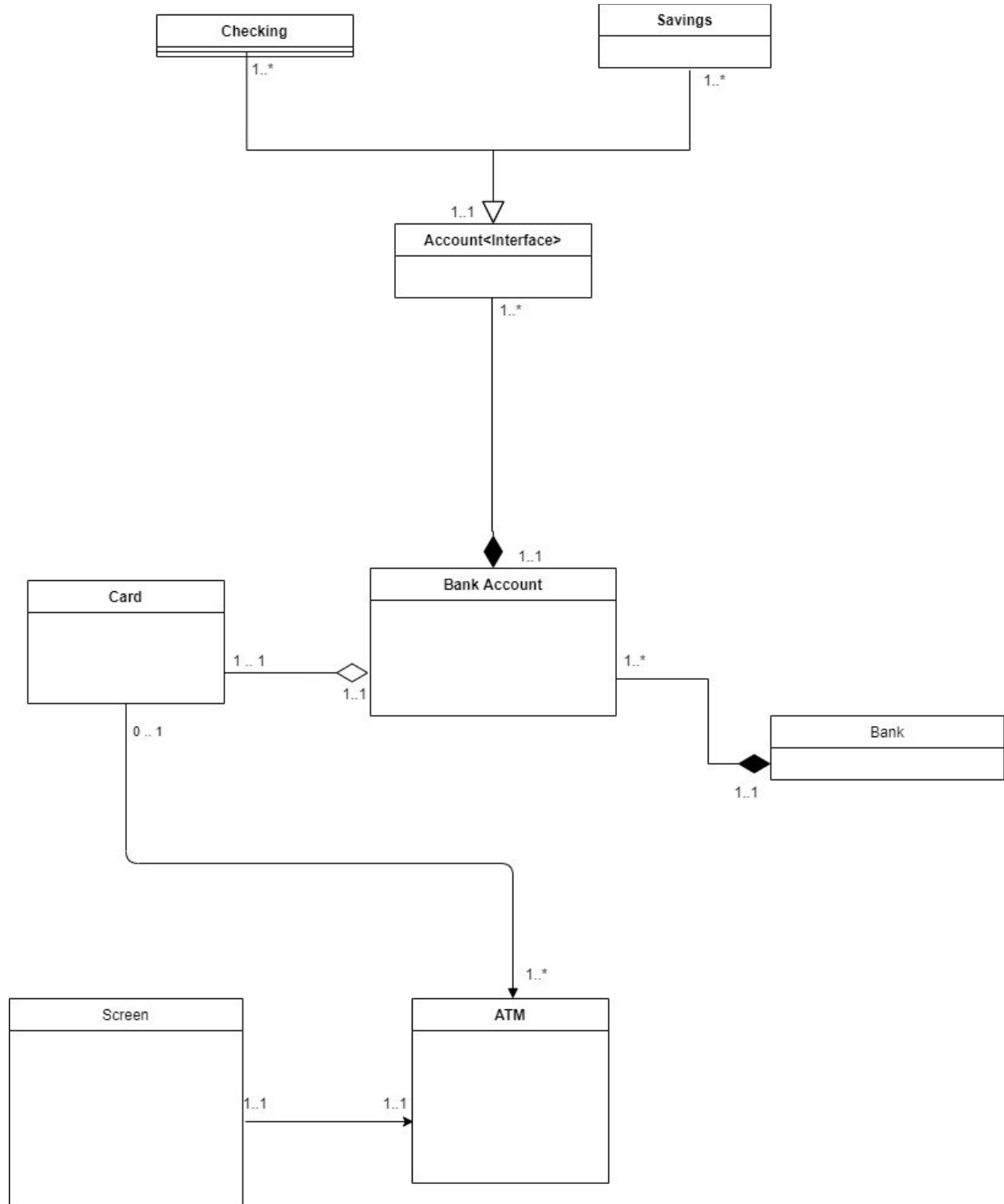
USE CASE 4: Withdraw Funds

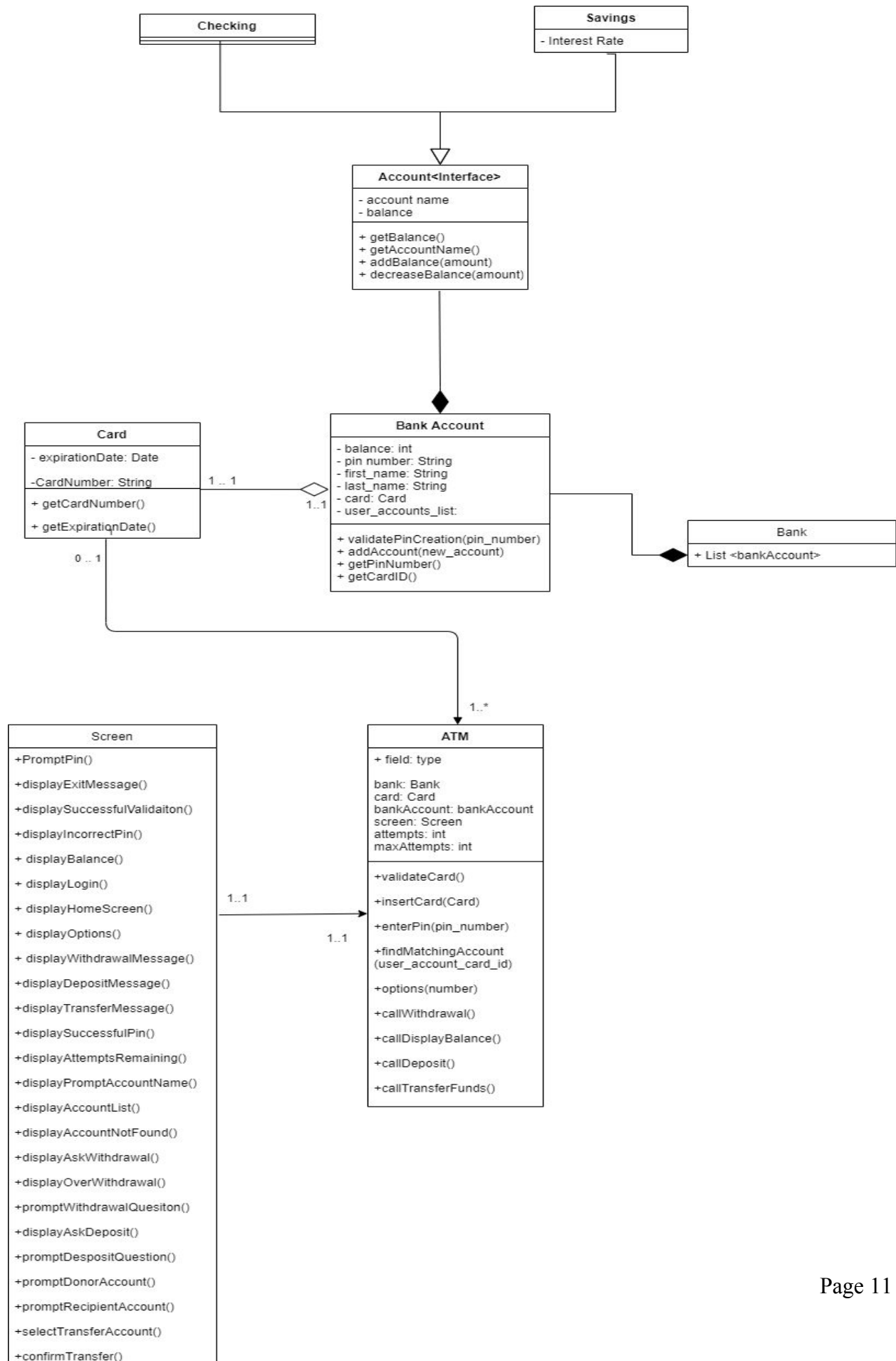
USE CASE NAME: Withdraw Funds	
SUMMARY	The customer withdraws funds from a selected account.
ACTOR	Customer
PRECONDITION	The PIN# must be correct prior to any transaction. The user must select "Withdraw Funds" in the options menu.

SUMMARY	
1.	The Screen displays the list of accounts.
2.	The customer will select one account from the list of accounts.
3.	The ATM prompts the user how much to withdraw.
4.	The ATM prints how much money was withdrawn as well as from which account.
PostCondition	The customer successfully received the funds they withdrew from their checking or savings account. The Options Menu is displayed.

Alternate Flow	
Invalid Account Flow	
1.	Displays account not found and returns to the options menu.

2.	If the withdrawal amount is greater than their current balance, prompt the user again.
Non - Functional Requirements	Cannot take out more money than their current balance.





Class Associations

1. A Screen has one and only one ATM.
An ATM has one and only one Screen.
2. A card is read by one to many ATMs.
An ATM reads zero to one Card at a time.
3. A Bank has one to many Bank Accounts.
A Bank Account belongs to one and only one bank.
4. An account has one to many savings accounts.
A savings account has one and only one account.
5. An account has one to many checking accounts.
A checking account has one and only one account.
6. A Bank Account has one and only one card.
A card belongs to one and only one Bank Account.
7. A Bank Account can hold one to many accounts.
An Account belongs to only one and only one Bank Account.

Screen Class :

This is the user interface. It allows for user input and more convenient output to the user.

Checking Class:

This is a type of user owned account that uses the *Account* interface. These are meant for everyday transactions.

Savings Class:

This is a type of user owned account that uses the *Account* interface. These are meant for storing money and earning interest.

Bank Account Class:

This is the account a bank has on file for a user. It has all personal information to keep track of each individual customer of their bank.

Bank Class:

Representative of the bank, this class maintains a list of all user owned Bank Accounts belonging to their bank.

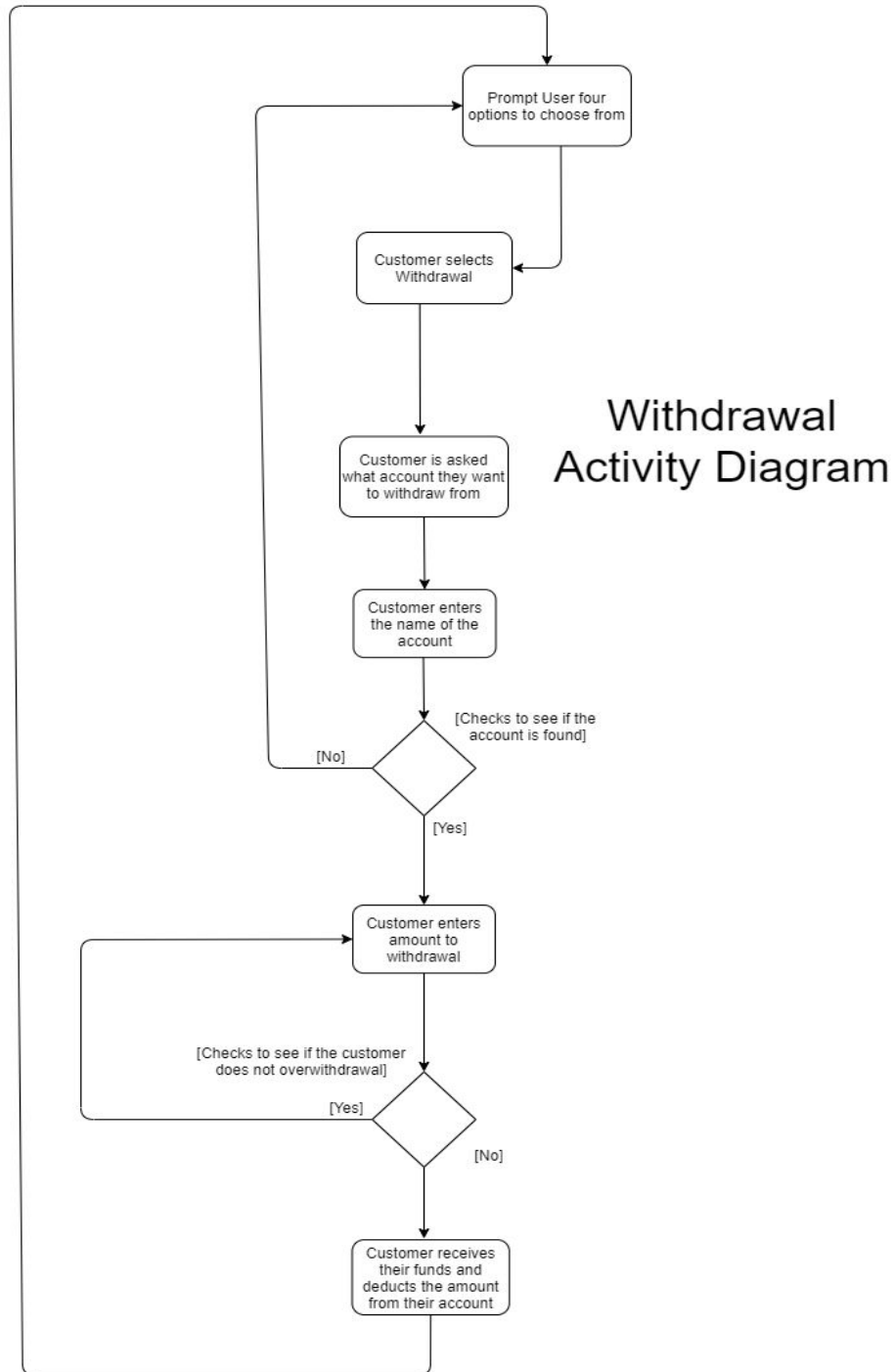
Card Class:

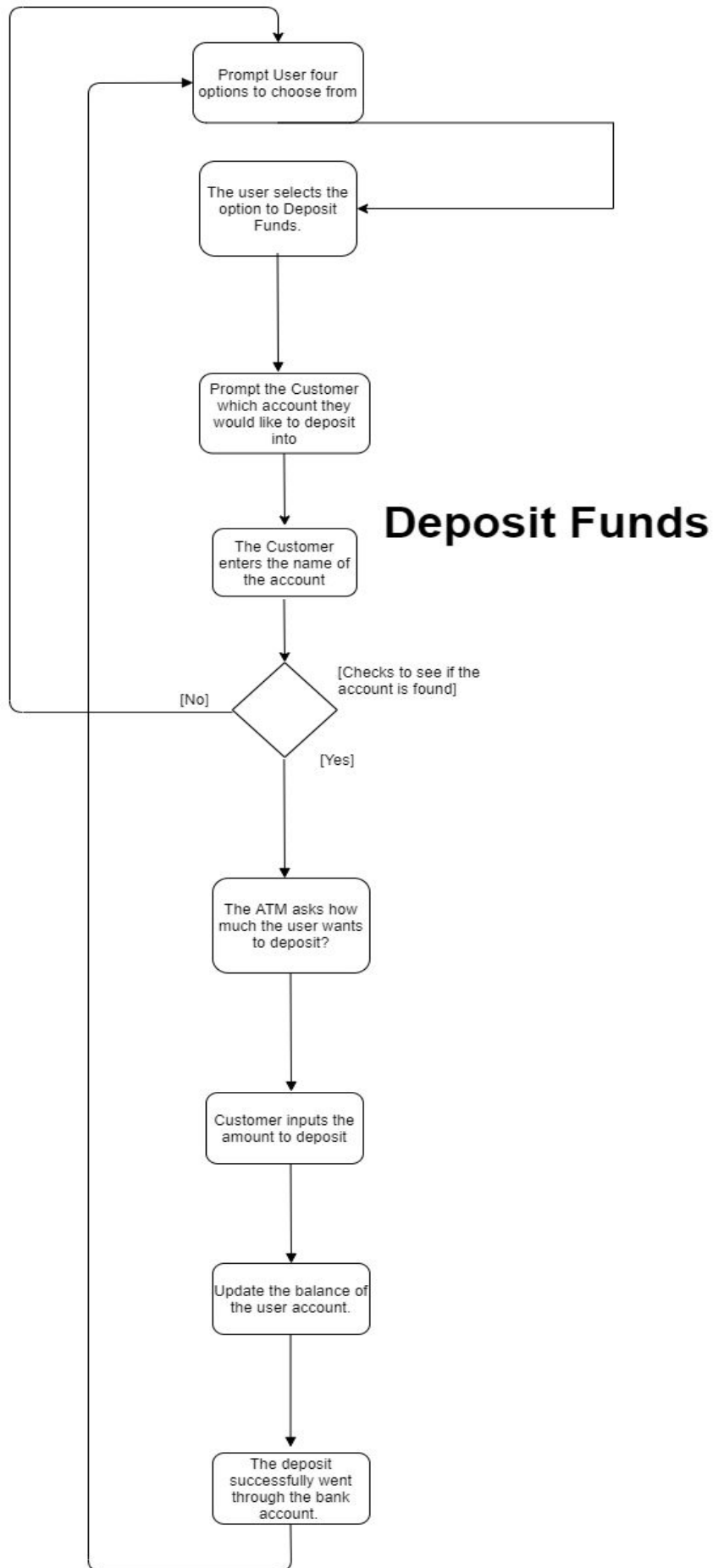
This class corresponds to a user's debit card. It allows a user to access their accounts at any ATM (provided they know the PIN). Users may only own 1 card at a time.

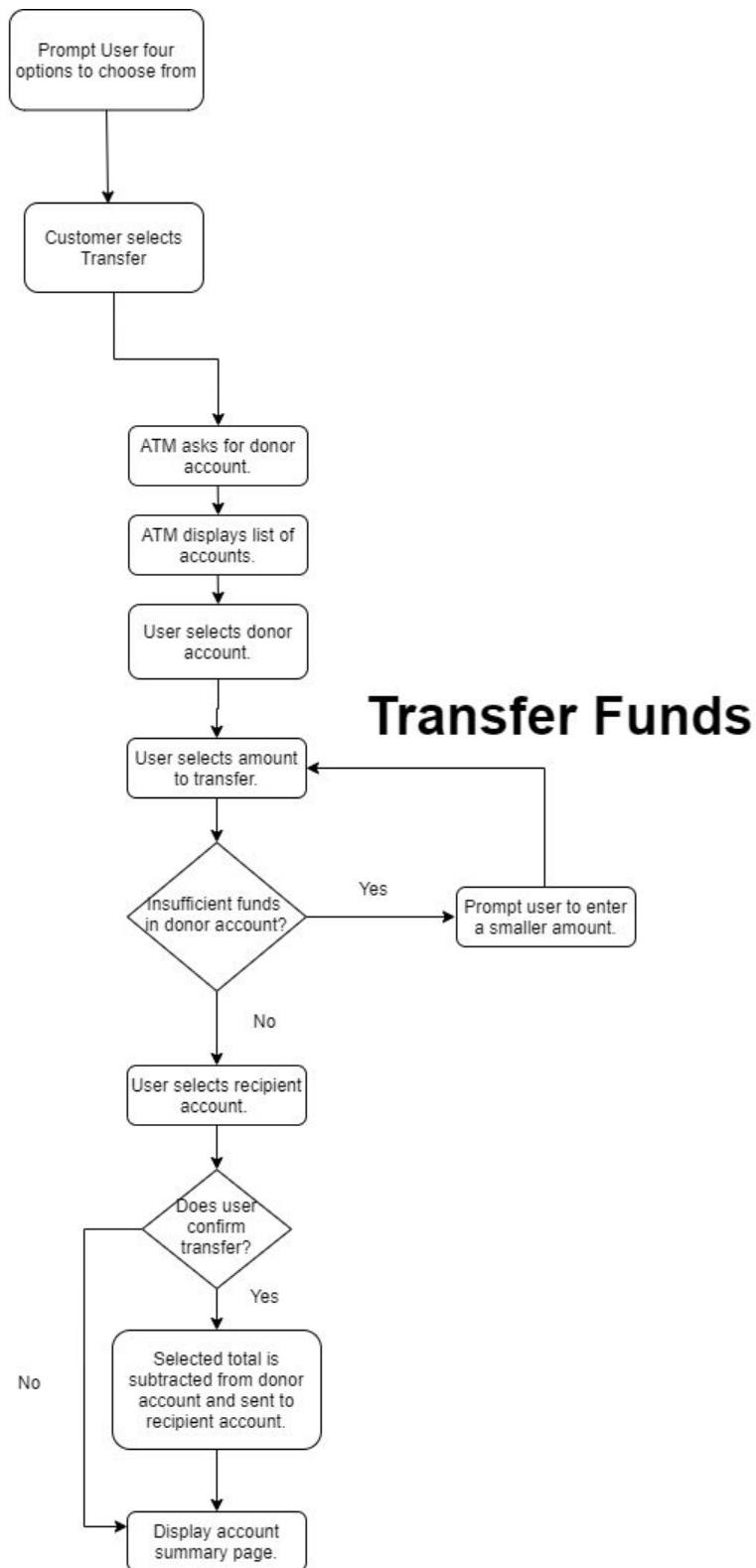
ATM Class:

This class represents the software and running of the ATM machine. It allows the user to perform basic transactions without the aid of a branch representative or teller.

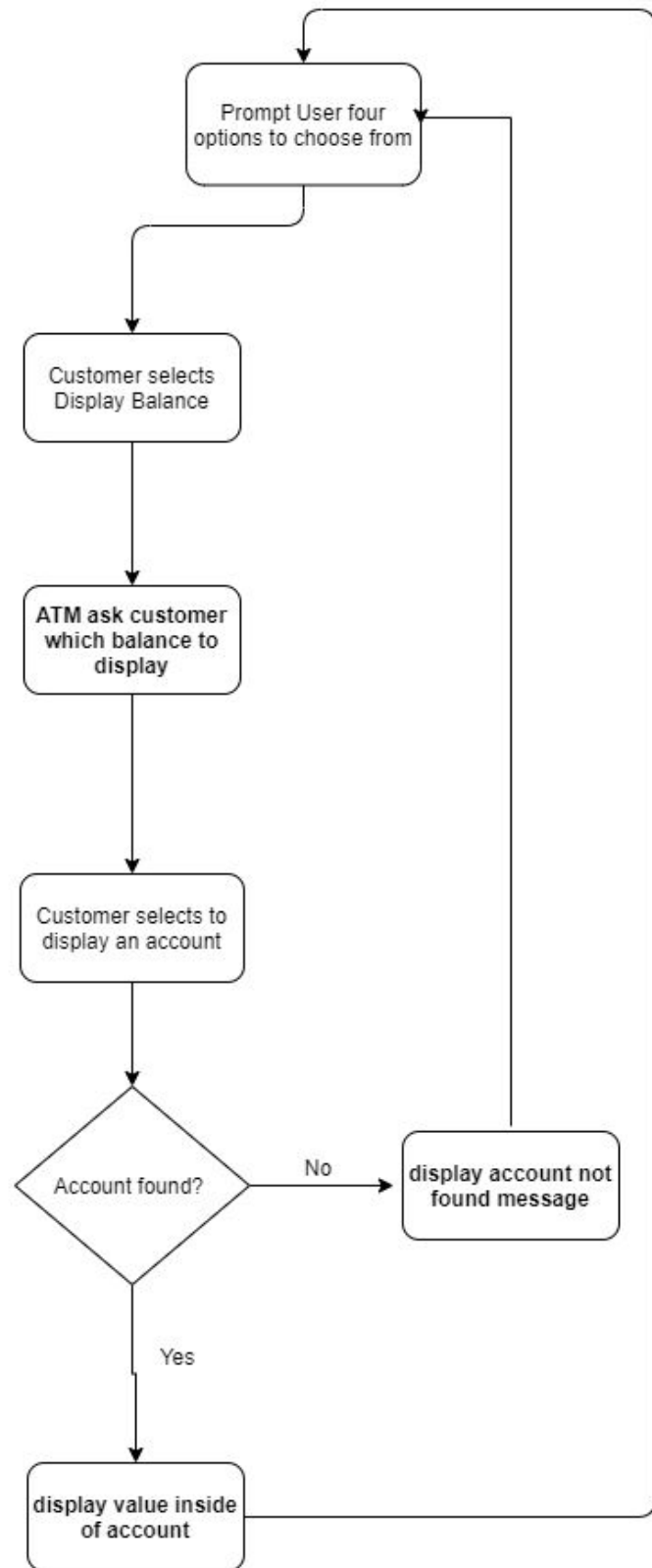
Activity Diagrams





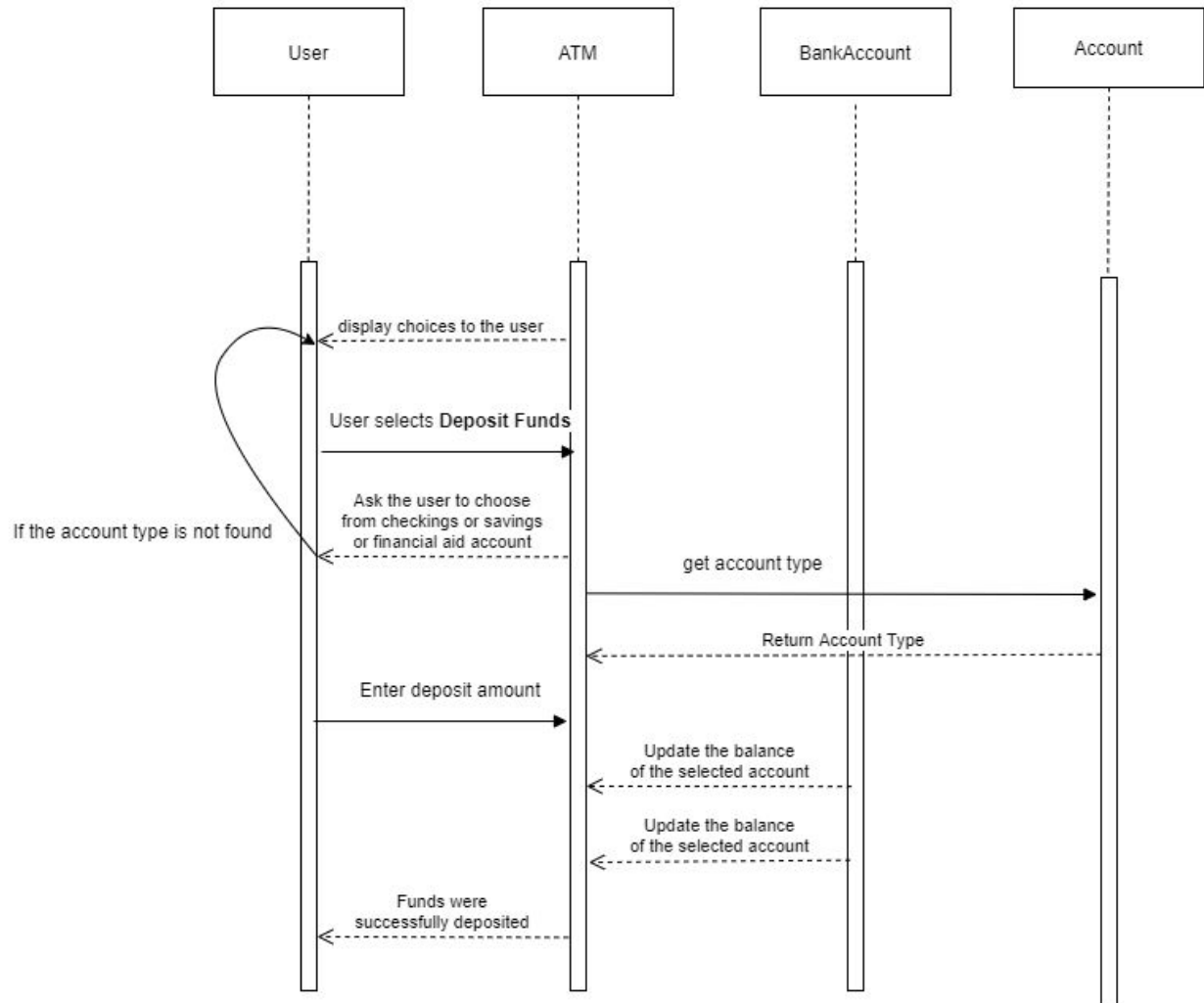


Display Funds

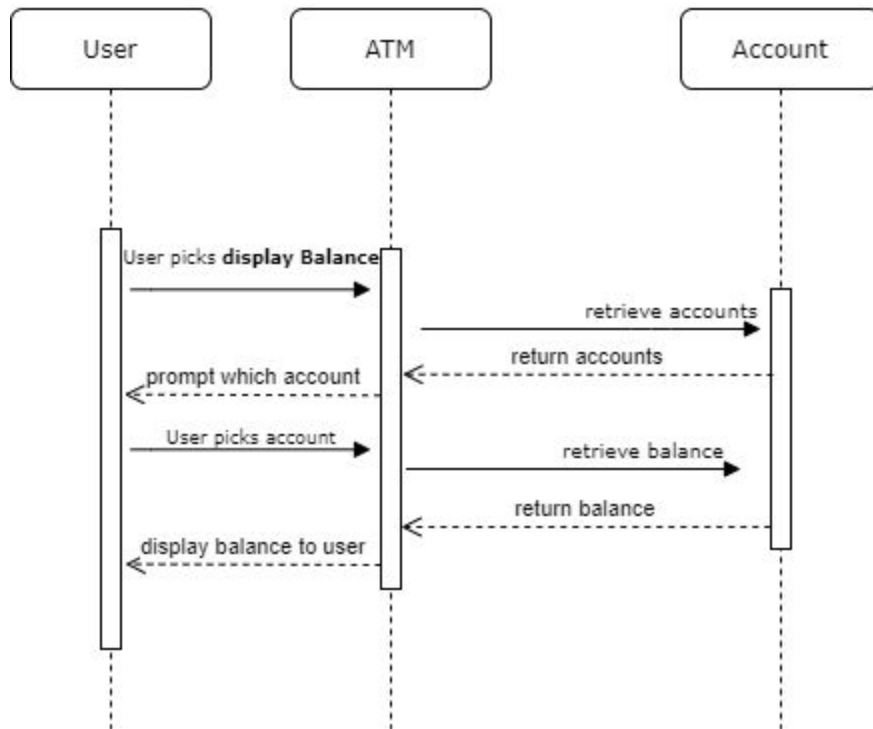


Sequence Diagrams

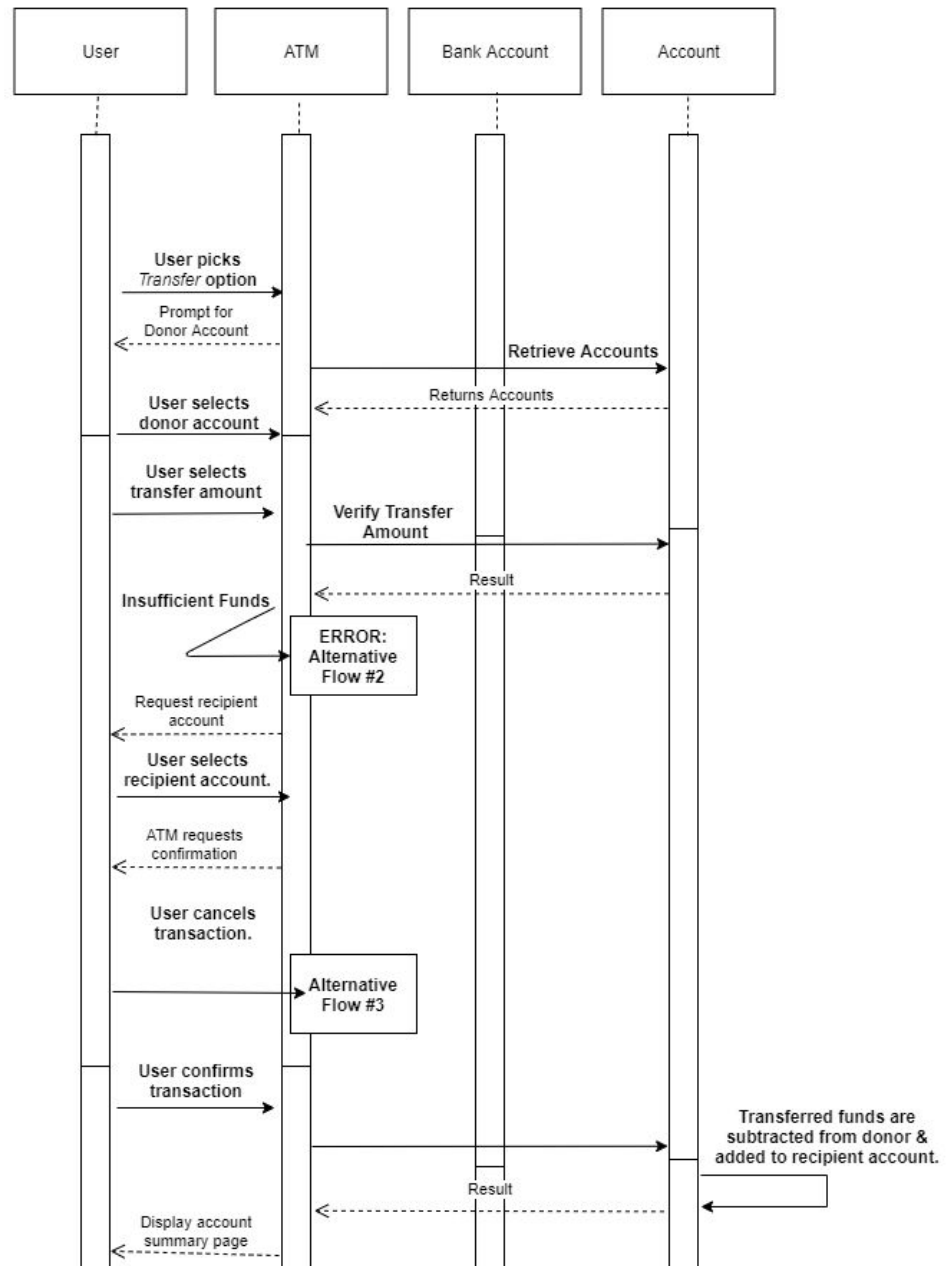
Deposit Funds Sequence Diagram



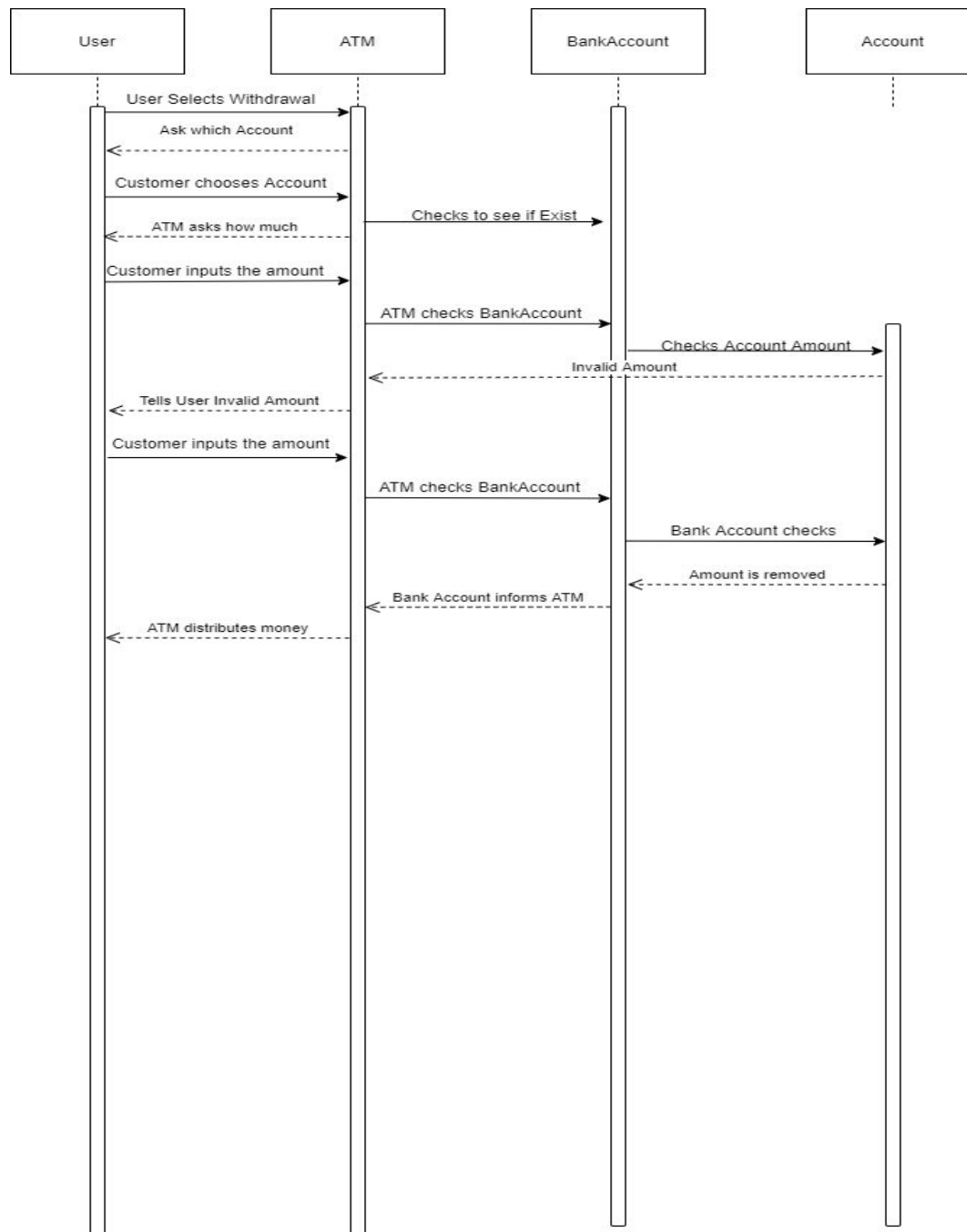
DISPLAY BALANCE SEQUENCE DIAGRAM



TRANSFER FUNDS SEQUENCE DIAGRAM

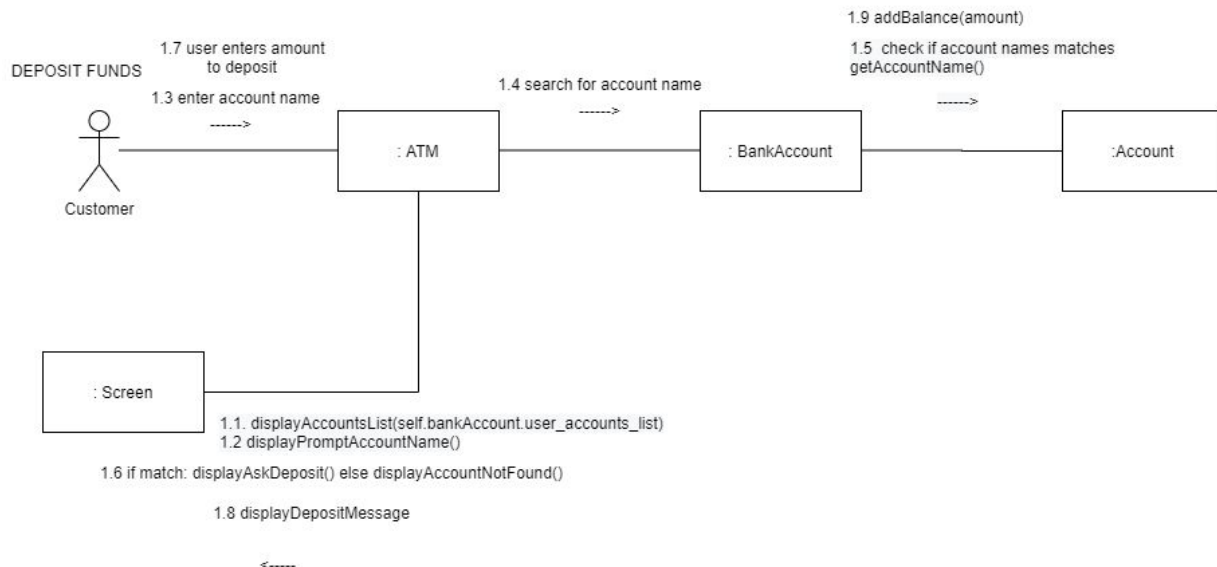
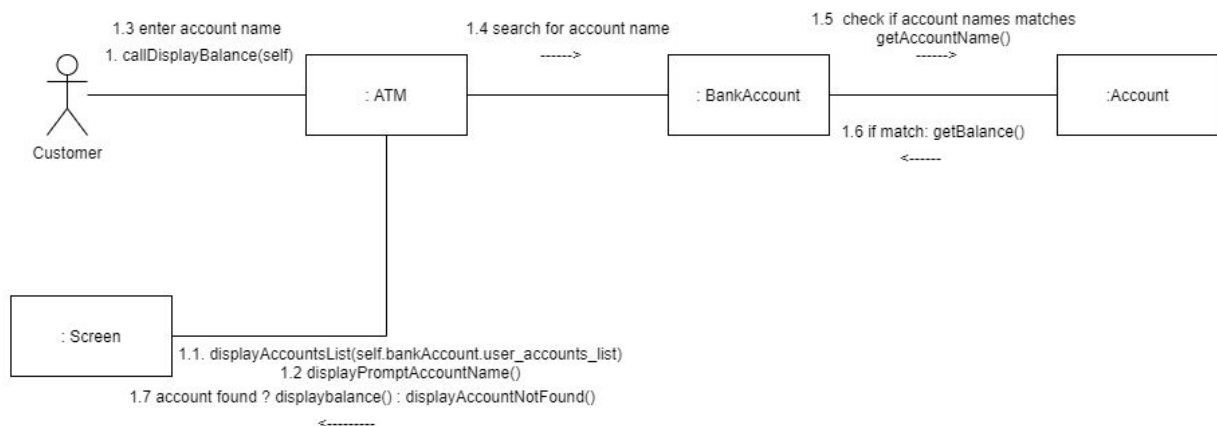


Withdrawal Sequence Diagram

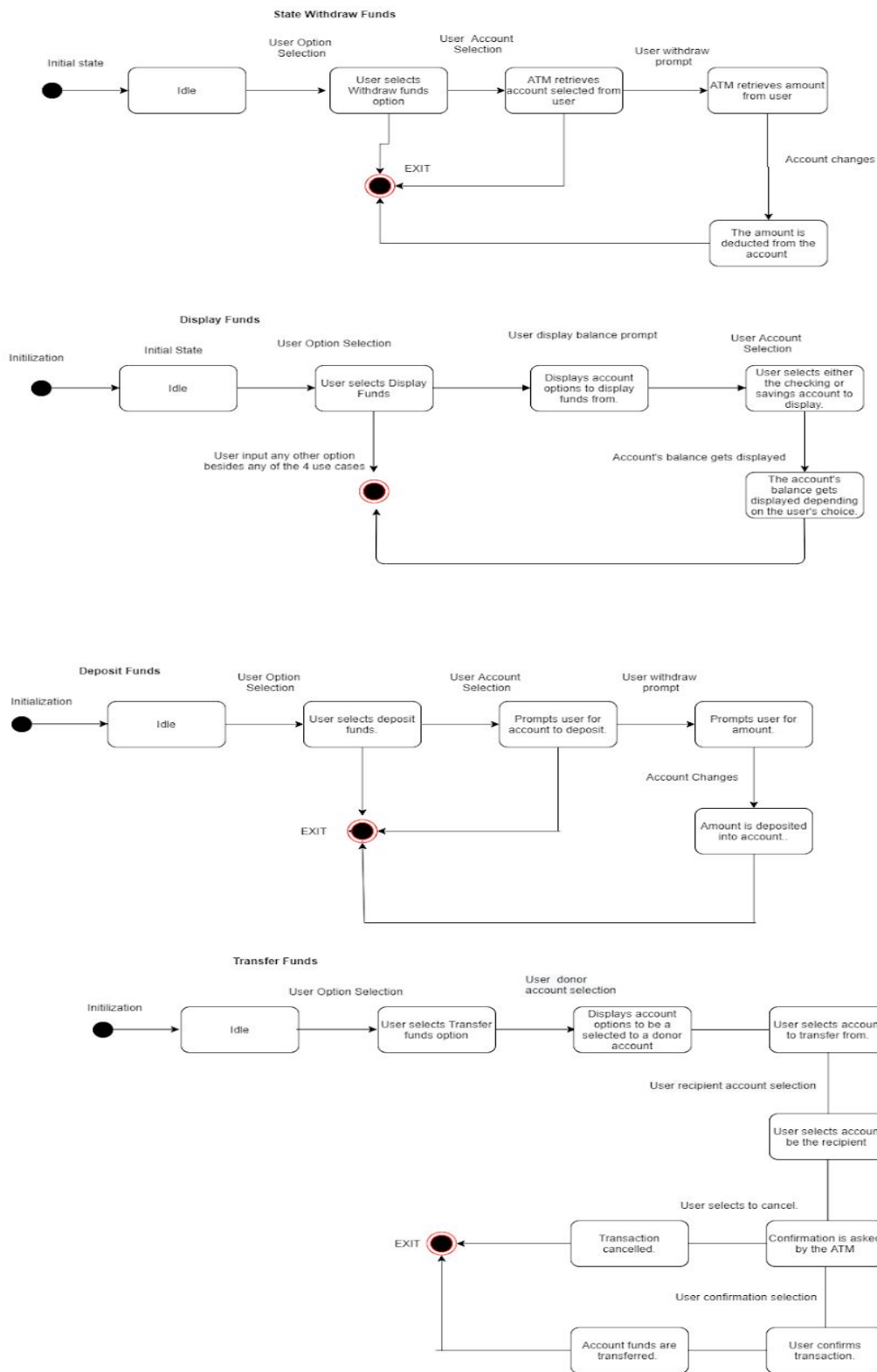


Collaboration Diagram

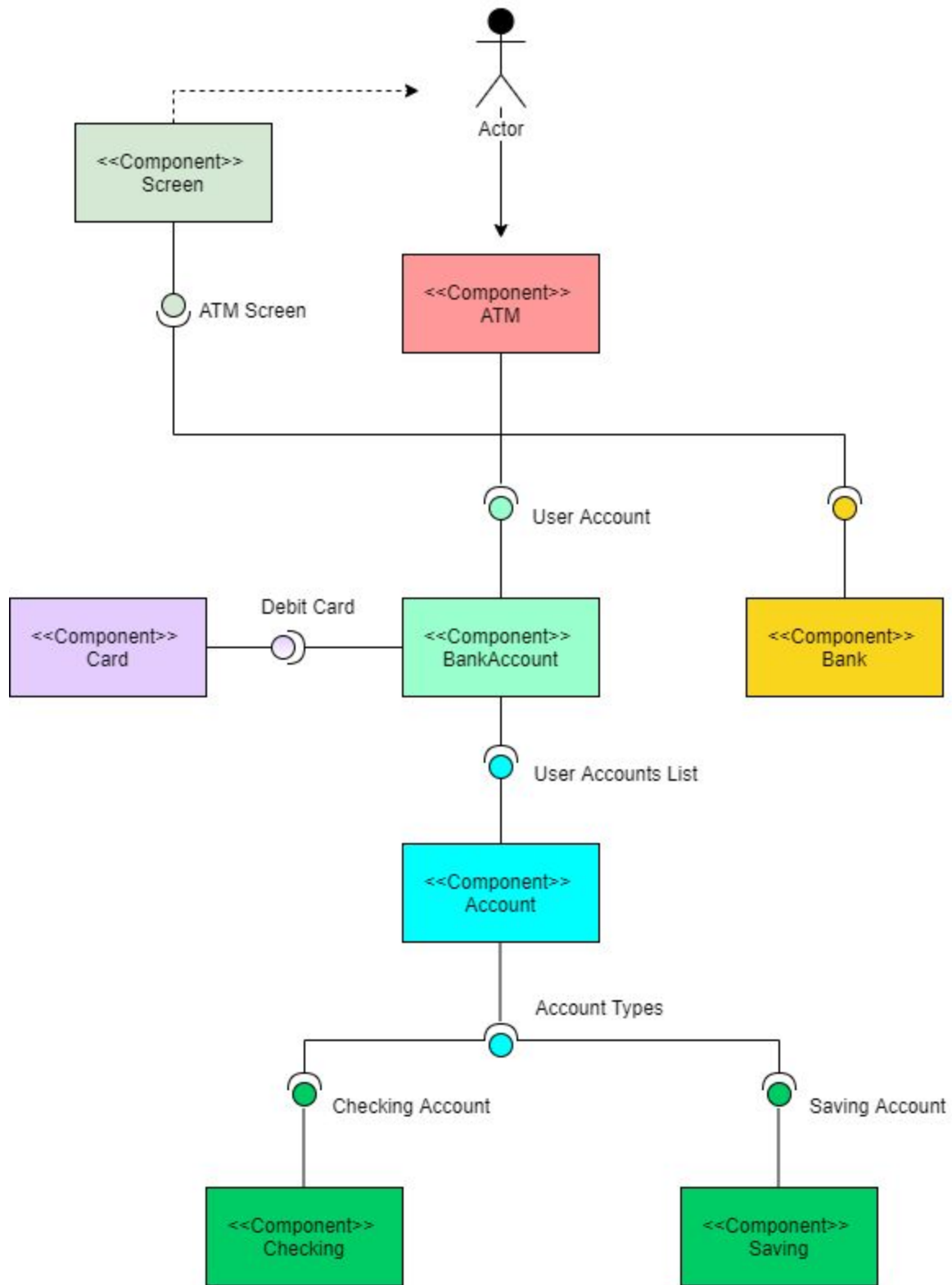
DISPLAY FUNDS



STATE DIAGRAMS




ATM Component Diagram



User Interface Design (UI)

Terminal:

Verify Card:

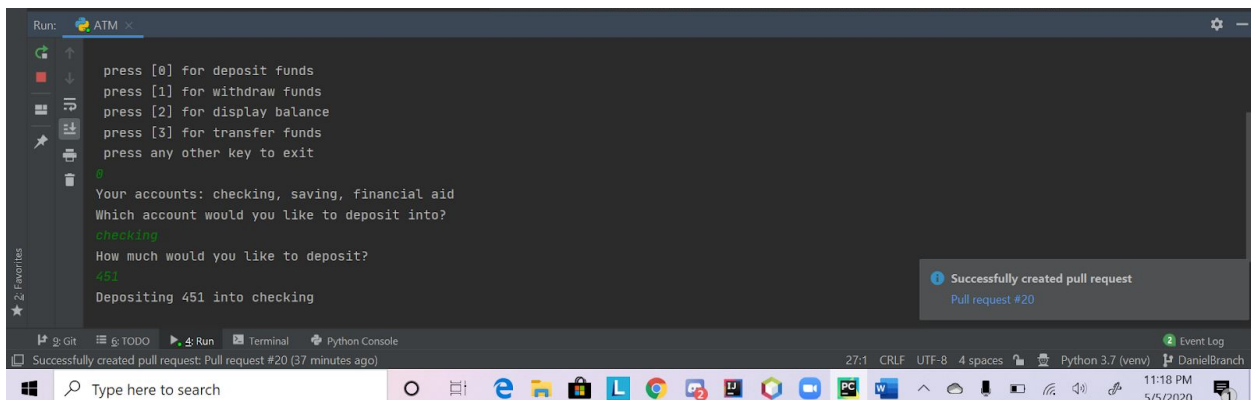


```
Welcome to Bank of America!
Please enter your card.
no errors found.
Please enter your Pin Number: |
9123
correct pin.

press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit
```

The user enters the pin number. If the pin number they entered is incorrect, the ATM will then display a message to the user that they have one less try each time they enter the incorrect pin number until the user enters the incorrect pin number three times consecutively. Otherwise, the ATM will display the options menu.

Deposit Funds:

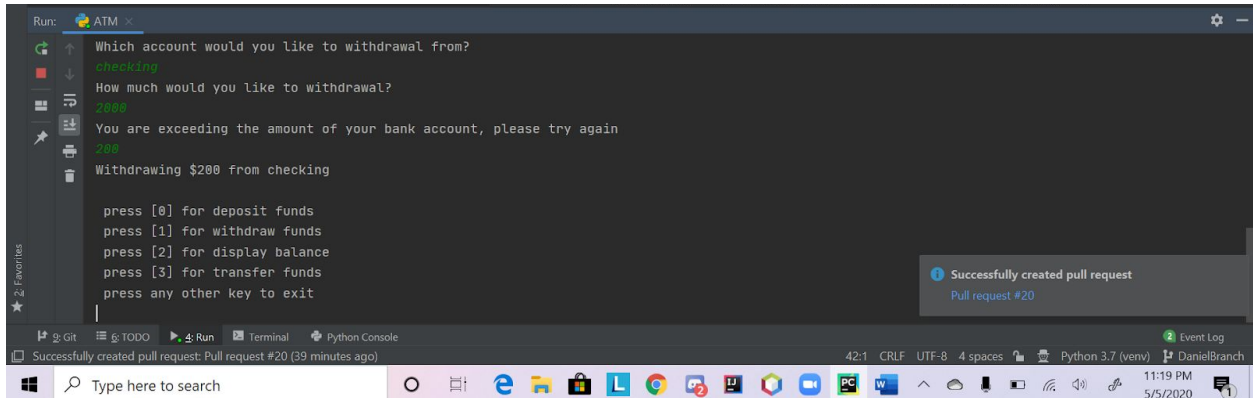


```
press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit
0
Your accounts: checking, saving, financial aid
Which account would you like to deposit into?
checking
How much would you like to deposit?
451
Depositing 451 into checking
```

When the user selects deposit funds, the ATM asks the user to ask which account to deposit into. If the user tries to deposit into an account that does not exist, the ATM tells the user that that account is not found and then takes the user back to the options menu. If the user enters an

account that exists, the ATM will prompt the user how much to deposit. Once the user enters an amount to deposit, the ATM displays how much money is being deposited as well as to indicate which account it is going into, then goes back to the options menu.

Withdraw Funds:



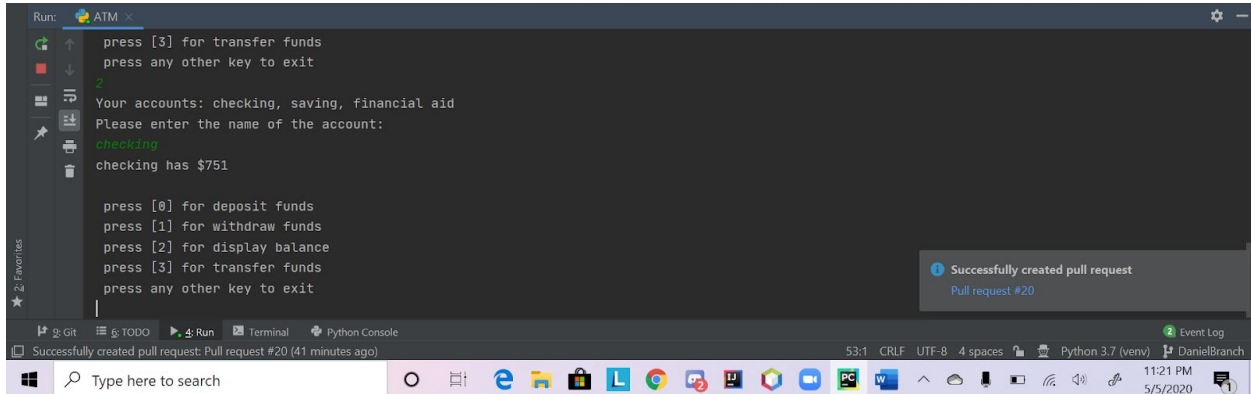
```
Run: ATM
Which account would you like to withdrawal from?
checking
How much would you like to withdrawal?
2000
You are exceeding the amount of your bank account, please try again
200
Withdrawing $200 from checking

press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit

Successfully created pull request
Pull request #20
```

When the user selects withdraw funds as the use case, the ATM asks the user which account to withdraw from. If the user were to withdraw from an account that does not exist, then it tells the user that the account is not found and the ATM takes the user back to the options menu. If the user enters an account that exists, then the ATM will prompt the user how much to withdraw. If the user enters an amount that is greater than the current balance of the account, the ATM will prompt the user “the amount that you want to withdraw is greater than the current balance of your account, please try again”. When the user enters an amount to withdraw that is less than or equal to the current balance of the account, the amount is deducted from the account and the user is returned to the options menu.

DISPLAY BALANCE:



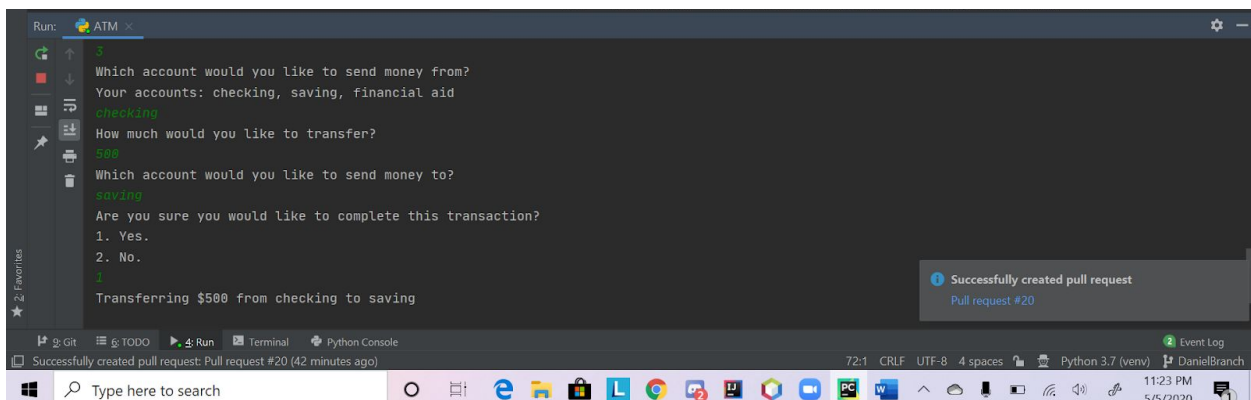
```
Run: ATM
press [3] for transfer funds
press any other key to exit
2
Your accounts: checking, saving, financial aid
Please enter the name of the account:
checking
checking has $751

press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit
```

Successfully created pull request
Pull request #20

When the user selects display balance, the ATM asks the user which account to display the balance for. If the user enters an account that does not exist, then it tells the user that that account is not found and the ATM takes the user back to the options menu. If the account is found, then it displays the account's account name as well as the balance of the account, then goes back to the options menu.

TRANSFER FUNDS:



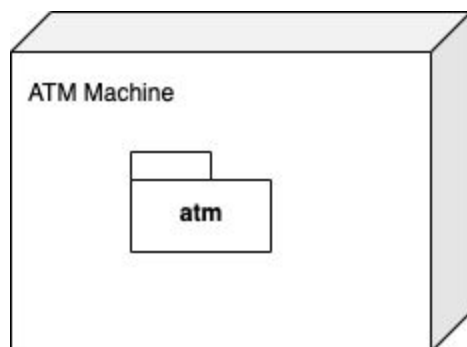
```
Run: ATM
3
Which account would you like to send money from?
Your accounts: checking, saving, financial aid
checking
How much would you like to transfer?
500
Which account would you like to send money to?
saving
Are you sure you would like to complete this transaction?
1. Yes.
2. No.
1
Transferring $500 from checking to saving
```

Successfully created pull request
Pull request #20

When the user selects transfer funds, the ATM asks which account to send money from (donor account). If the particular donor account is not found, then it tells the user that the particular

donor account is not found and then goes back to the options menu. If the particular donor account is found, then it prompts how much to transfer. If the amount to transfer is greater than or equal to the donor's account current balance, it will prompt the user enters the amount to withdraw that is greater than the current balance of the account, then the ATM will keep prompting the user "the amount that you want to withdraw is greater than the current balance of the donor account, please try again" until the user enters an amount to withdraw is that is less than or equal to the current balance of the donor account. Once the user inputs a balance to withdraw that is less than or equal to the current balance of the donor account, the ATM prompts the user for the recipient account. If the particular recipient account is not found, then it displays a message to the user saying the recipient account is found, as well as to go back to the options menu. If the particular recipient account is found, then the ATM asks the user to confirm this transaction. If the user selects one, then the transaction is successfully completed and it displays the donor account, the amount of money being transferred as well as the recipient account as well as to return to the options menu. If the user selects two, then the transaction is cancelled as well to go back to the options menu.

Deployment Diagram:



Python Code

1. ATM.py:

```
import datetime

from atm.Bank import Bank
from atm.BankAccount import BankAccount
from atm.Card import Card
from atm.Checking import Checking
from atm.Savings import Savings
from atm.Screen import Screen

class ATM:
    def __init__(self, bank):
        self.bank = bank # initialize bank
        self.card = None
        self.bankAccount = None
        self.screen = Screen()
        self.attempts = 0
        self.maxAttempts = 2

    def validateCard(self):
        found_account = self.findMatchingAccount(self.card.getCardNumber())

        # check to see if the account has been found
        if found_account is not None:
            self.bankAccount = found_account
        else:
            raise Exception('There is no account associated with debit card.')

    def insertCard(self, inserted_card):
        self.card = inserted_card
        self.attempts = 0

    """ this method searches through bank accounts and returns the account if found """

    def enterPin(self, pin_number):
        if self.attempts < self.maxAttempts:
            if self.bankAccount.getPinNumber() == pin_number:
```

```

        return True
    else:
        self.screen.displayAttemptsRemaining(str(self.maxAttempts - self.attempts))
        self.attempts = self.attempts + 1
        return False
    raise Exception("You have exceeded the maximum number of attempts.")

def findMatchingAccount(self, user_account_card_id):
    for account in self.bank.bankAccounts:
        if account.getCardID() == user_account_card_id:
            return account

def options(self, number):
    if number == 0:
        self.callDeposit()
        return True
    elif number == 1:
        self.callWithdrawal()
        return True
    elif number == 2:
        self.callDisplayBalance()
        return True
    elif number == 3:
        self.callTransferFunds()
        return True
    else:
        self.screen.displayExitMessage()
        return False

def callWithdrawal(self):
    self.screen.displayAccountsList(self.bankAccount.user_accounts_list)
    self.screen.promptWithdrawQuestion()
    answer = input().lower()

    for account in self.bankAccount.user_accounts_list:
        if account.getAccountName() == answer:
            self.screen.displayAskWithdrawal()
            decreaseBal = input()

            while not decreaseBal.isnumeric():
                self.screen.displayAskWithdrawal()
                decreaseBal = input()

```

```

        while int(decreaseBal) > account.getBalance():
            self.screen.displayOverWithdrawal()
            decreaseBal = input()
        self.screen.displayWithdrawalMessage(decreaseBal)
        account.decreaseBalance(int(decreaseBal))
        return
    self.screen.displayAccountNotFound(answer)

def callDisplayBalance(self):
    self.screen.displayAccountsList(self.bankAccount.user_accounts_list)
    self.screen.displayPromptAccountName()
    account_name = input().lower()
    account_name = account_name.lower()

    for account in self.bankAccount.user_accounts_list:
        if account.getAccountName() == account_name:
            amount = account.getBalance()
            message = "" + account_name + " has $" + str(amount)
            self.screen.displayBalance(message)
            return
    self.screen.displayAccountNotFound(account_name)

def callDeposit(self):
    self.screen.displayAccountsList(self.bankAccount.user_accounts_list)
    self.screen.promptDepositQuestion()
    answer = input().lower()

    for account in self.bankAccount.user_accounts_list:
        if account.getAccountName() == answer:
            self.screen.displayAskDeposit()
            increaseBal = input()

            while not increaseBal.isnumeric():
                self.screen.displayAskDeposit()
                increaseBal = input()

            self.screen.displayDepositMessage(increaseBal, account.getAccountName())
            account.addBalance(int(increaseBal))
            return
    self.screen.displayAccountNotFound(answer)
def callTransferFunds(self):

    # Ask for donor account.

```

```

self.screen.promptDonorAccount()
self.screen.displayAccountsList(self.bankAccount.user_accounts_list)
donor = input().lower()
transferAmount = 0

# If account exists, select and validate transfer amount.
for donorAccount in self.bankAccount.user_accounts_list:
    if donorAccount.getAccountName() == donor:
        self.screen.selectTransferAmount()
        transferAmount = input()

        while not transferAmount.isnumeric():
            self.screen.selectTransferAmount()
            transferAmount = input()

        while int(transferAmount) > donorAccount.getBalance():
            self.screen.displayOverWithdrawal()
            transferAmount = input()

# Select recipient account.
self.screen.promptRecipientAccount()
recipient = input().lower()

# Find recipient account, confirm, then transfer.
for recipientAccount in self.bankAccount.user_accounts_list:
    if recipientAccount.getAccountName() == recipient:
        verify = self.screen.confirmTransfer()
        # input validation & break in case "no" goes here.
        if int(verify) == 1:
            donorAccount.decreaseBalance(int(transferAmount))
            recipientAccount.addBalance(int(transferAmount))
            self.screen.displayTransferMessage(transferAmount, donor, recipient)
            return
        self.screen.displayAccountNotFound(recipient)
        return
self.screen.displayAccountNotFound(donor)

def main():
    """initialize initial values"""
    # initialize bank
    bank = Bank('Bank of America')

```



```

# initialize users and cards

# initialize Williams BankAccount
williams_debit_card = Card('111231314', datetime.datetime(2020, 5, 17)) # year, month, day
williams_bank_account = BankAccount('William', 'Gusmanov', williams_debit_card, '0123')

williams_checking = Checking("checking", 500)
williams_saving1 = Savings("saving", 1000, 0.01)
williams_saving2 = Savings("financial aid", 3000, 0.02)

williams_bank_account.addAccount(williams_checking)
williams_bank_account.addAccount(williams_saving1)
williams_bank_account.addAccount(williams_saving2)

bank.addNewBankAccount(williams_bank_account)

# initialize the atm machine
atm = ATM(bank)

"""begin atm transaction"""
atm.screen.displayHomeScreen('Bank of America')
atm.screen.displayLogin()

atm.insertCard(williams_debit_card)
atm.validateCard()
atm.screen.displaySuccessfulValidation()

# we need a scanner here
correct_pin = False
# will throw exception once number of tries exceed limit
# here we enter pin and verify it.
while not correct_pin:
    atm.screen.promptPin()
    pin = input()
    correct_pin = atm.enterPin(pin)
    atm.screen.displaySuccessfulPin() if correct_pin else atm.screen.displayIncorrectPin()

continue_transaction = True
while continue_transaction:
    message = '\n press [0] for deposit funds \n ' \
        'press [1] for withdraw funds \n press [2] for display balance \n ' \
        'press [3] for transfer funds \n press any other key to exit'
    atm.screen.displayOptions(message) # display options

```

```
input_number = int(input())
continue_transaction = atm.options(input_number)
```

```
if __name__ == '__main__':
    main()
```

2. Account.py:

```
class Account:
    def __init__(self, account_name, initial_balance):
        self.account_name = account_name
        self.balance = initial_balance

    def getBalance(self):
        return self.balance

    def getAccountName(self):
        return self.account_name

    def addBalance(self, amount):
        self.balance = self.balance + amount

    def decreaseBalance(self, amount):
        self.addBalance(-amount)
```

3. Bank.py:

```
class Bank:
    bankAccounts = []
    bankName = ""

    def __init__(self, name):
        self.bankName = name

    """add a new bank account to bankAccounts"""
    def addNewBankAccount(self, user_account):
        self.bankAccounts.append(user_account)
```

4. BankAccount.py:

```
class BankAccount:
    def __init__(self, first_name, last_name, card, pin_number):
        if self.validatePinCreation(pin_number):
            self.pin_number = pin_number
        self.first_name = first_name
        self.last_name = last_name
        self.card = card
        self.user_accounts_list = [] # a list of accounts like checkings/savings

    @staticmethod
    def validatePinCreation(pin_number):
        pin_size_val = False
        pin_type_val = False
        if len(pin_number) == 4:
            pin_size_val = True
        for character in pin_number:
            if str.isdigit(character):
                pin_type_val = True
        if pin_type_val and pin_size_val:
            return True
        else:
            return False

    # add a new user account (checking or savings) to accounts
    def addAccount(self, new_account):
        self.user_accounts_list.append(new_account)

    def getPinNumber(self):
        if self.pin_number is not None:
            return self.pin_number
        else:
            raise Exception('there is no pin number.')

    def getCardID(self):
        return self.card.getCardNumber()
```

5. Card.py:

```
class Card:
```

```

def __init__(self, card_number, expiration_date):
    self.__expirationDate = expiration_date
    self.__card_number = card_number

def getCardNumber(self):
    return self.__card_number

def getExpirationDate(self):
    return self.__expirationDate

```

6. Checking.py:

```

from atm.Account import Account

""" checking is different from savings.
    There can only be one checkings account - will be satisfied by using Singleton Pattern."""

class Checking(Account):
    _instance = None

    def __init__(self, account_name, initial_balance):
        super().__init__(account_name, initial_balance)
        self.account_name = account_name
        self.balance = initial_balance
        if Checking._instance is not None:
            raise Exception('there can only be one checking account.')
        else:
            Checking._instance = self

```

7. Savings.py:

```

from atm.Account import Account

"""Saving Accounts have interest rates. There can be multiple saving accounts. """

class Savings(Account):
    def __init__(self, account_name, initial_balance, interest_rate):
        super().__init__(account_name, initial_balance)
        self.account_name = account_name
        self.balance = initial_balance
        self.interest_rate = interest_rate

```

8. Screen.py:

```
class Screen:
```

```
    @staticmethod
```

```
    def promptPin():
```

```
        print('Please enter your Pin Number: ')
```

```
    @staticmethod
```

```
    def displayLogin():
```

```
        print('Please enter your card.')
```

```
    @staticmethod
```

```
    def displayHomeScreen(bank_name):
```

```
        print('Welcome to ' + bank_name + '!')
```

```
    @staticmethod
```

```
    def displayOptions(message):
```

```
        print(message)
```

```
    @staticmethod
```

```
    def displayWithdrawalMessage(amount, account_name):
```

```
        print("Withdrawing $" + amount + ' from ' + account_name)
```

```
    @staticmethod
```

```
    def displayDepositMessage(amount, account_name):
```

```
        print('Depositing ' + amount + ' into ' + account_name)
```

```
    @staticmethod
```

```
    def displayTransferMessage(amount, account_name1, account_name2):
```

```
        print("Transferring $" + amount + " from " + account_name1 + " to " + account_name2)
```

```
    @staticmethod
```

```
    def displayExitMessage():
```

```
        print("Thank you, Goodbye!")
```

```
    @staticmethod
```

```
    def displaySuccessfulValidation():
```

```
        print('no errors found.')
```

```
    @staticmethod
```

```

def displayIncorrectPin():
    print("Incorrect PIN entered.")

    @staticmethod
    def displaySuccessfulPin():
        print("correct pin.")

    @staticmethod
    def displayAttemptsRemaining(attempts):
        print("You have " + attempts + " attempts remaining")

    @staticmethod
    def displayPromptAccountName():
        print("Please enter the name of the account: ")

    @staticmethod
    def displayBalance(message):
        print(message)

    @staticmethod
    def displayAccountsList(account_list):
        print("Your accounts: ", end = "")
        message = list("")
        for account in account_list:
            message.append(account.getAccountName())
            message.append(", ")
        message[len(message) - 1] = ""
        string = "".join(message)
        print(string)

    @staticmethod
    def displayAccountNotFound(account_name):
        print(account_name + ' not found.')

# Withdraw Functions

    @staticmethod
    def displayAskWithdrawal():
        print("How much would you like to withdrawal?")

    @staticmethod
    def displayOverWithdrawal():
        print("You are exceeding the amount of your bank account, please try again")

```

```

@staticmethod
def promptWithdrawQuestion():
    print("Which account would you like to withdrawal from?")

# Deposit Functions
@staticmethod
def displayAskDeposit():
    print("How much would you like to deposit?")

@staticmethod
def promptDepositQuestion():
    print("Which account would you like to deposit into?")

# Transferring Funds

@staticmethod
def promptDonorAccount():
    print("Which account would you like to send money from?")

@staticmethod
def promptRecipientAccount():
    print("Which account would you like to send money to?")

@staticmethod
def selectTransferAmount():
    print("How much would you like to transfer?")

@staticmethod
def confirmTransfer():
    print("Are you sure you would like to complete this transaction?")
    print("1. Yes.")
    print("2. No.")
    answer = input()

    if int(answer) == 1:
        return True
    else:
        return False

```

Narrative for Unit Testing:

Essentially, we decided to do a unit test for two use cases, depositing funds as well as displaying funds. By using python, we had to import the module unittest, and to pass in unittest.TestCase as a parameter to the function. The way that we did our unit testing for depositing balance, was to have an account be initialized to have \$500. We used the addBalance function to add \$500 dollars to the account. We used the getBalance function to see how much was in the account after adding a certain amount to be deposited. We used python's unit testing function self.assertEqual to check if the balance of the bank was equal to \$1000, and with that it actually passed the test. The way that we did our unit testing for display balance, was to have an account to be initialized to have \$500. We used the getBalance function to see how much was in the account. We used self.assertsEqual to verify that when we display the balance of the account, it was still \$500, thus passing the test.

Unit Testing Classes:

9. TestDepositBalance.py:

```
import unittest
from atm.BankAccount import BankAccount
from atm.Account import Account

class TestDepositBalance(unittest.TestCase):
    def test_depositBalance(self):
        account = Account("test", 500)
        account.addBalance(500)
        results = account.getBalance()
        self.assertEqual(results, 1000)

if __name__ == '__main__':
    unittest.main()
```


10. test_displayFunds.py:

```
import unittest
from atm.BankAccount import BankAccount
from atm.Account import Account
```

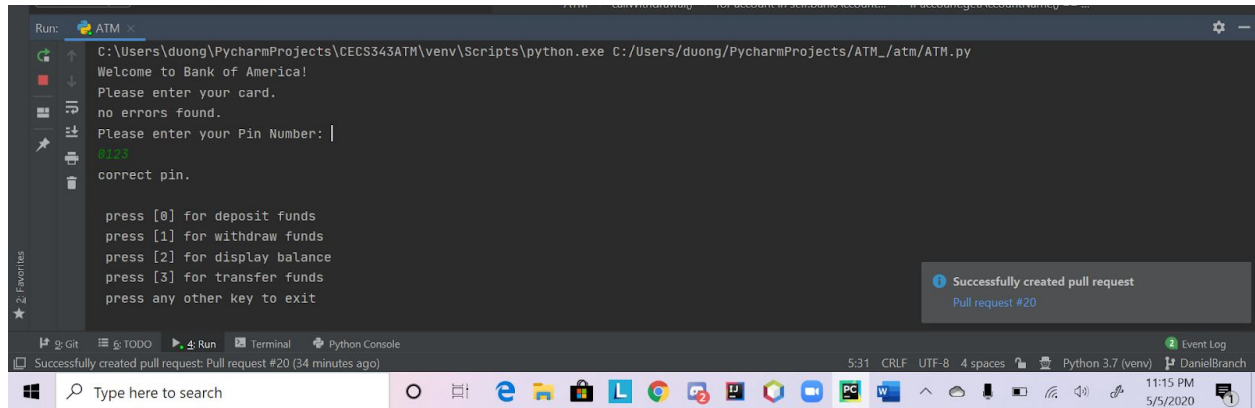
```
class TestGetBalance(unittest.TestCase):
    def test_getBalance(self):
        account = Account("test", 500)
        result = account.getBalance()
        self.assertEqual(result, 500)

if __name__ == '__main__':
    unittest.main()
```

11. __init__.py:

RUNTIME OUTPUT

Verify Card:

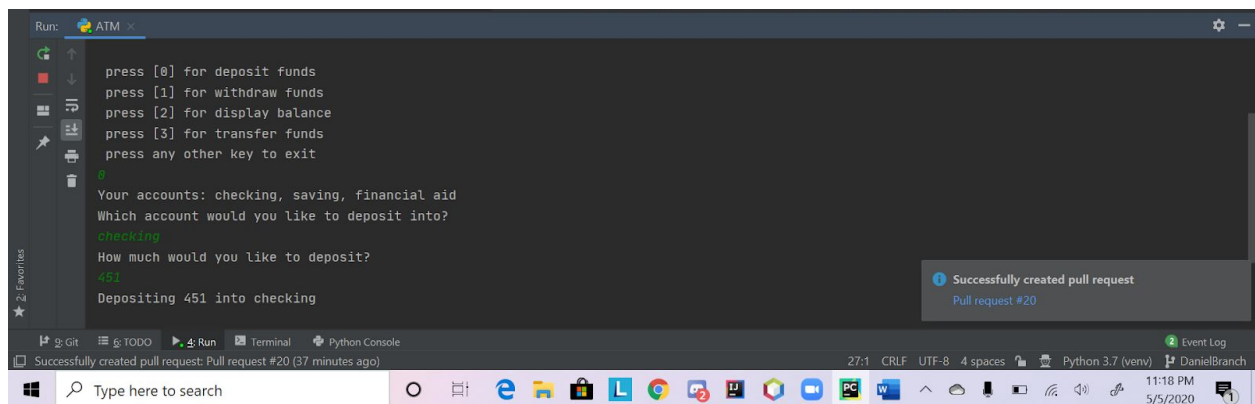


```
Run: ATM
C:\Users\duong\PycharmProjects\CECS343ATM\venv\Scripts\python.exe C:/Users/duong/PycharmProjects/ATM_/atm/ATM.py
Welcome to Bank of America!
Please enter your card.
no errors found.
Please enter your Pin Number: |
0123
correct pin.

press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit
```

Successfully created pull request
Pull request #20

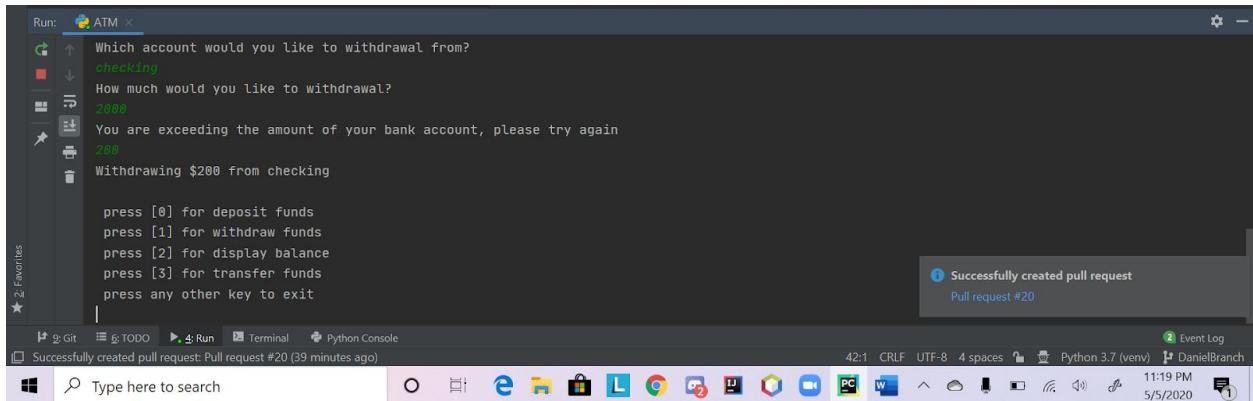
Deposit Funds:



```
Run: ATM
press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit
0
Your accounts: checking, saving, financial aid
Which account would you like to deposit into?
checking
How much would you like to deposit?
451
Depositing 451 into checking
```

Successfully created pull request
Pull request #20

WITHDRAW FUNDS:

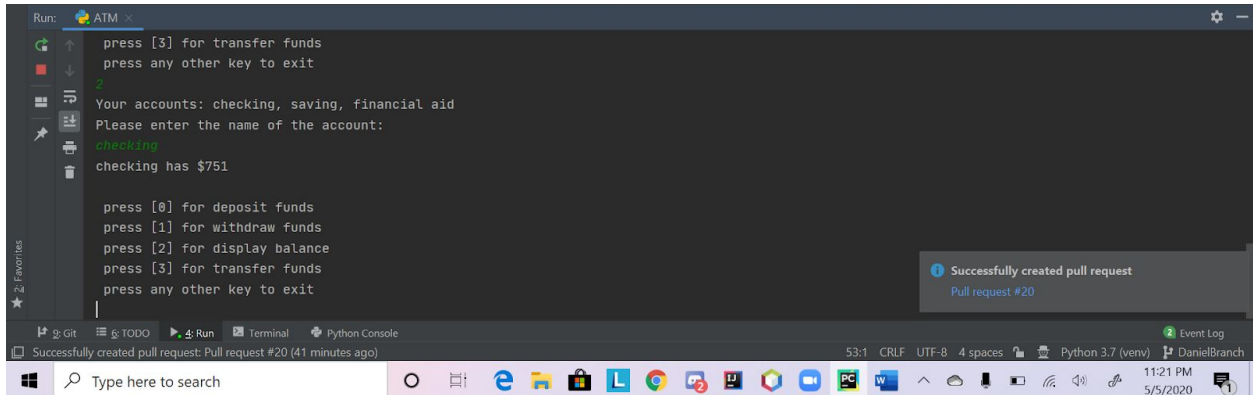


```
Run: ATM
Which account would you like to withdrawal from?
checking
How much would you like to withdrawal?
2000
You are exceeding the amount of your bank account, please try again
200
Withdrawing $200 from checking

press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit

Successfully created pull request
Pull request #20
```

DISPLAY BALANCE:

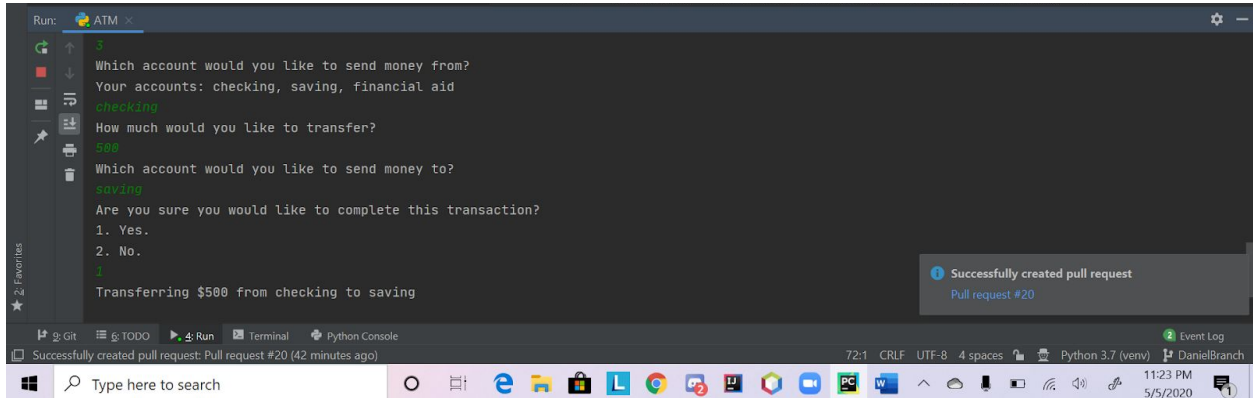


```
Run: ATM
press [3] for transfer funds
press any other key to exit
2
Your accounts: checking, saving, financial aid
Please enter the name of the account:
checking
checking has $751

press [0] for deposit funds
press [1] for withdraw funds
press [2] for display balance
press [3] for transfer funds
press any other key to exit

Successfully created pull request
Pull request #20
```

TRANSFER FUNDS:



```
Run: ATM
$
Which account would you like to send money from?
Your accounts: checking, saving, financial aid
checking
How much would you like to transfer?
500
Which account would you like to send money to?
saving
Are you sure you would like to complete this transaction?
1. Yes.
2. No.
1
Transferring $500 from checking to saving

Successfully created pull request
Pull request #20
```

SUMMARY

The ATM machine allows a user to communicate with their bank account and allow for various functions the user can choose from to manage their accounts. After determining the requirements for the stakeholder and implementing them via Python, we have implemented an ATM system that involves the four use cases of Depositing Funds, Withdrawing Funds, Displaying Funds and Transferring Funds. We constructed our code based off of our requirement modeling diagrams, along with a UML diagram with associations between classes. We created unit tests to verify the correctness of our program.