OPEN UNIVERSITY – SRI LANKA

[MINI PROJECT]

[EEI3372]

::: 2023_EEI3372-WD-G05:::

4/11/2024

NAME: G.D.L MADHUMINI

S NO: S23010147

REG NO: 523598053

INTRODUCTION

This lab report contains the analysis and a Python command-line app that simulates a basic bank account management system. The app allow users to perform the following actions such as - Create a new account with a unique account number and initial deposit amount, Deposit money into an account using the account number , Withdraw money from an account using the account number, Check the account balance using the account number and Transfer money between two accounts using their account numbers.

This report provides a simple text-based menu for users to interact with the system. Store account information in memory without the need for persistent storage. Also the app can handle basic validation, such as - Check for non-negative deposit and withdrawal amounts, Ensure sufficient balance before withdrawal or transfer and Verify the existence of accounts before any transactions.

This command-line application built with Python simulates a basic bank account management system. Users can create new accounts, deposit and withdraw money, check their account balances, and transfer funds between accounts. The app has basic validation checks and stores all account information in memory without requiring a database.

A bank account management system refers to the strategic and operational processes involved in effectively overseeing and controlling a company's bank accounts. It includes all functions related to opening, maintaining, and optimizing bank accounts to achieve financial efficiency, security, and compliance.

ASSUMPTIONS

1- Account numbers are unique and do not contain spaces or special characters.

Account numbers serve as unique identifiers for financial accounts, and they follow specific rules to maintain consistency and security.

2- Users enter valid numeric amounts for deposits, withdrawals, and transfers.

Users are expected to input numeric amounts (positive or negative) accurately when performing deposits, withdrawals, or transfers within a financial system.

3- Account numbers are case-sensitive.

When dealing with account numbers, always consider the exact letter case specified to ensure accurate transactions and security.

4- The app does not persistently store account information.

The app does not retain account information beyond the current session, and there is no database connectivity for storing such data. In other words there is no database connectivity.

5- The app runs in a single session and does not handle multi-user scenarios.

The app running in an application where it maintains only one active session at a time is designed for one user at a time and does not handle multiple users interacting with the app simultaneously.

PROBLEMS THAT I FACED

1) How to start

Firstly I wrote down the conditions and steps I need to make the code.

- Creating a New Account
- Depositing Money
- Withdrawing Money
- Checking Account Balance
- Transferring Money
- Basic Validation -

❖ Non-negative deposit and withdrawal amounts are enforced.
❖ Sufficient balance is checked before withdrawal or transfer.
❖ Existence of accounts is verified before any transactions.
❖ Text-Based Menu for users to interact with the system.

- In Memory Storage

2) How to make a simple text-based menu

As we instructed, to make a simple text-based menu for users to interact with the system, I decided to make functions as choices such as,

1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit

```
def main():
        print("\n1. Create Account\n2. Deposit\n3. Withdraw\n4. Transfer\n5.
Check Balance\n6. Quit")
        choice = input("Enter your choice: ")
```

To choose the choice, I add a while loop to continue until condition is true.

```
def main():
    while True:
        print("\n1. Create Account\n2. Deposit\n3. Withdraw\n4. Transfer\n5.
Check Balance\n6. Quit")
        choice = input("Enter your choice: ")
        if choice == '1':
            create_account(accounts)
        elif choice == '2':
            deposit(accounts)
        elif choice == '3':
            withdraw(accounts)
        elif choice == '4':
            transfer(accounts)
        elif choice == '5':
            check_balance(accounts)
        elif choice == '6':
            break
        else:
            print("Invalid choice.")
```

3) How to make a basic structure for managing bank accounts

This is the basic structure with two parameters. we can assign values to instance variables using this.

```
class BankAccount:
    def __init__(self, account_number, initial_balance):
        self.account_number = account_number
        self.balance = initial_balance
```

following the above structure, we can add other functions as well.

4) How to define the functions.

I look at a example and make my code according to the given example and algorithms later in this report.

1-deposit

```
def deposit(account, amount):
    if amount > 0:
        account.balance += amount
        return True
    else:
        return False
```

The deposit method allows users to add money to their bank account. It takes an amount as input. If the amount is non-negative, it adds it to the account balance. If the amount is negative, it prints an error message and returns False.

```
def deposit(self, amount):
    if amount >= 0:
        self.balance += amount
        return True
    else:
        print("Deposit amount must be non-negative.")
        return False
```

2-withdraw

```
def withdraw(account, amount):
    if amount > 0 and
account.balance >= amount:
        account.balance -= amount
        return True
    else:
        return False
```

The withdrawal method allows users to take money out of their bank account. If the amount is non-negative, it checks if the account has sufficient balance. If the balance is sufficient, it deducts the withdrawal amount from the account balance.If the balance is insufficient or the amount is negative, it prints an error message and returns False

```
def withdraw(self, amount):
    if amount >= 0:
        if self.balance >= amount:
            self.balance -= amount
            return True
        else:
            print("Insufficient balance.")
            return False
    else:
        print("Withdrawal amount must be non-negative.")
        return False
```

3-transfer

```
def transfer(source_account,
target_account, amount):
    if amount > 0 and
source_account.balance >= amount:
        source_account.balance -= amount
        target_account.balance += amount
        return True
    else:
        return False
```

The transfer method allows users to transfer money from one account to another. If the amount is positive, it checks if the source account has sufficient balance. If the balance is sufficient, it deducts the amount from the source account and adds it to the target account. If the balance is insufficient or the amount is negative, it prints an error message and returns False.

```
def transfer(self, other_account,
amount):
    if amount >= 0:
        if self.balance >= amount:
            self.balance -= amount
            other_account.balance +=
amount
            return True
        else:
            print("Insufficient balance.")
            return False
    else:
        print("Transfer amount must
be positive.")
        return False
```

4-check balance

```
def get_balance(account):
    return account.balance
```

The check balance method allows users to retrieve the current balance of their bank account. It simply returns the current balance stored in the self. balance attribute.

```
def check_balance(self):
    return self.balance
```

5) How to Store account information in memory without the need for persistent storage

The account information is stored in memory using Python dictionaries within the Bank class. Each account number serves as the key, and the corresponding BankAccount object serves as the value. This approach allows account information to be stored and accessed efficiently during the program's execution without the need for persistent storage such as a database

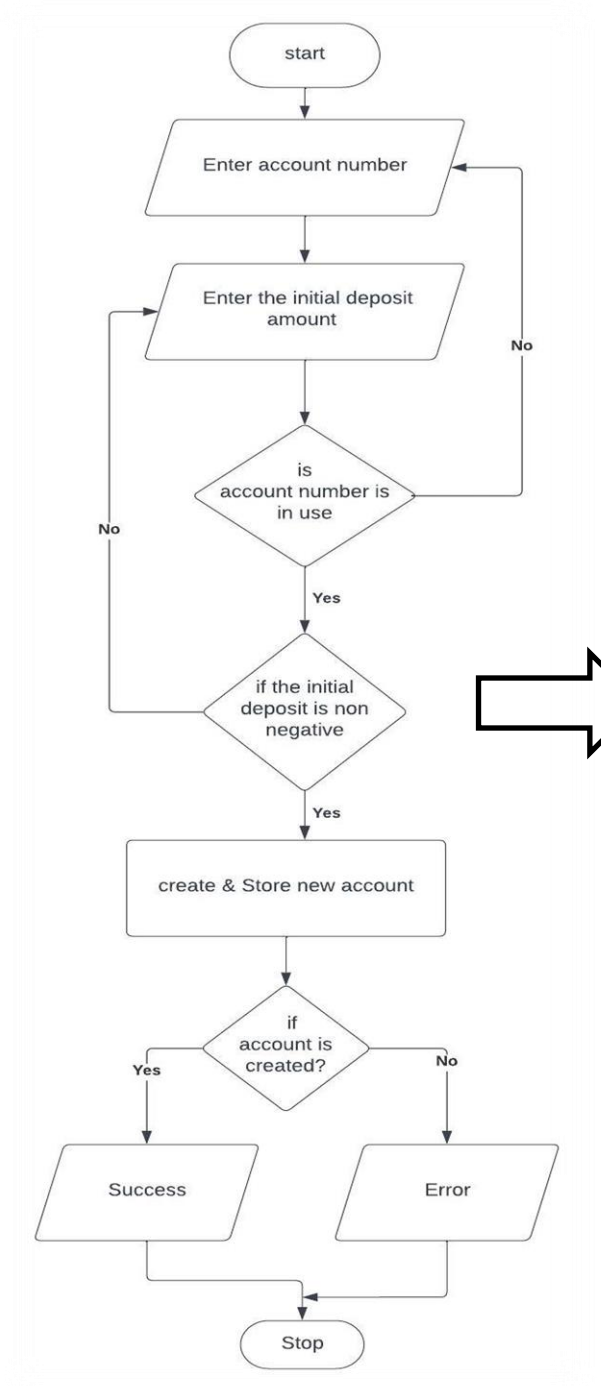
```
class Bank:

    def __init__(self):
        self.accounts = {}
```

This is how the account information is stored in memory within the Bank class.

However, in this process, I learned that If the program is terminated, all data stored in memory will be lost. If you need to persist data between program executions, you would need to implement additional functionality to save and load data from external storage, such as a file or a database.

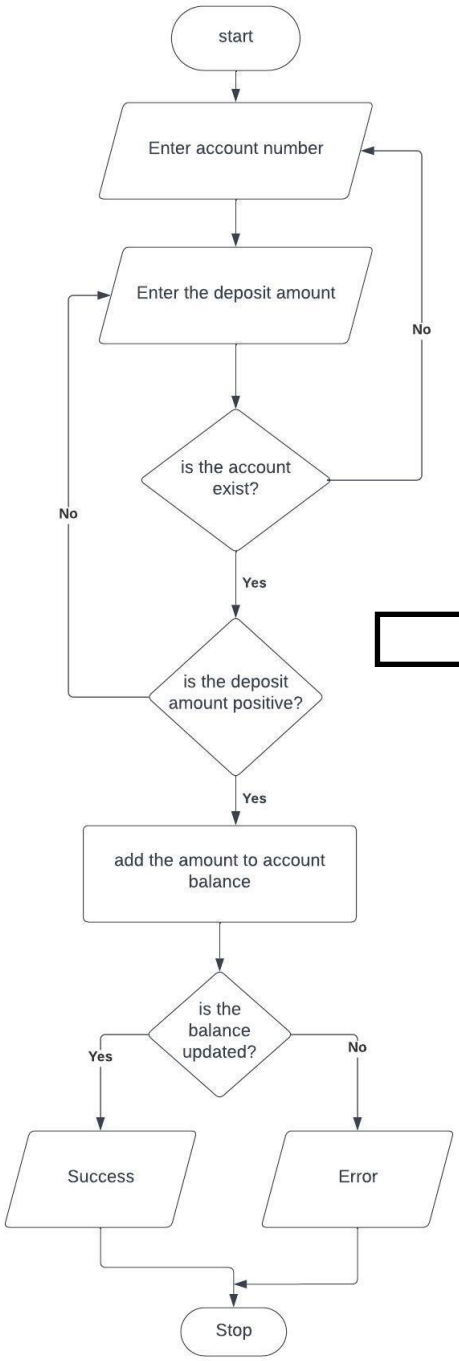
FUNCTIONAL REQUIREMENTS

1. Creating a new account



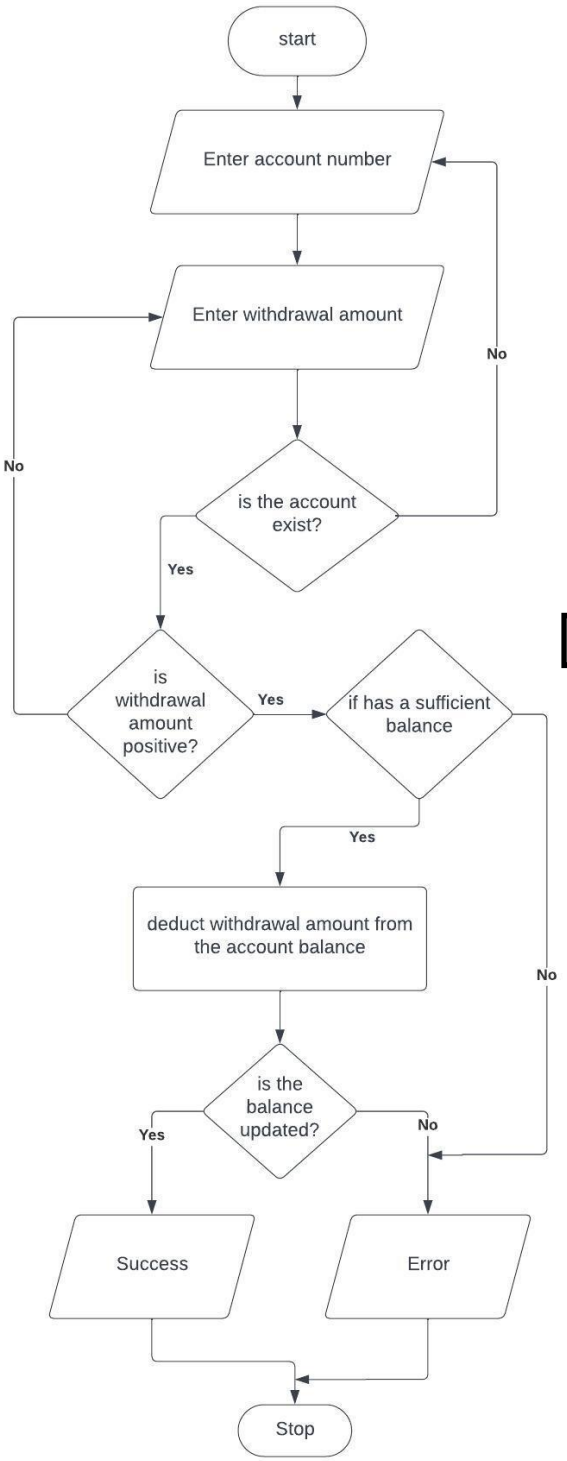

```
def create_account(accounts):
    account_number = input("Enter account number: ")
    if account_number in accounts:
        print("Account already exists.")
        return
    initial_balance = float(input("Enter initial balance: "))
    accounts[account_number] = BankAccount(account_number, initial_balance)
    print("Account created successfully.")
```

2. Depositing money



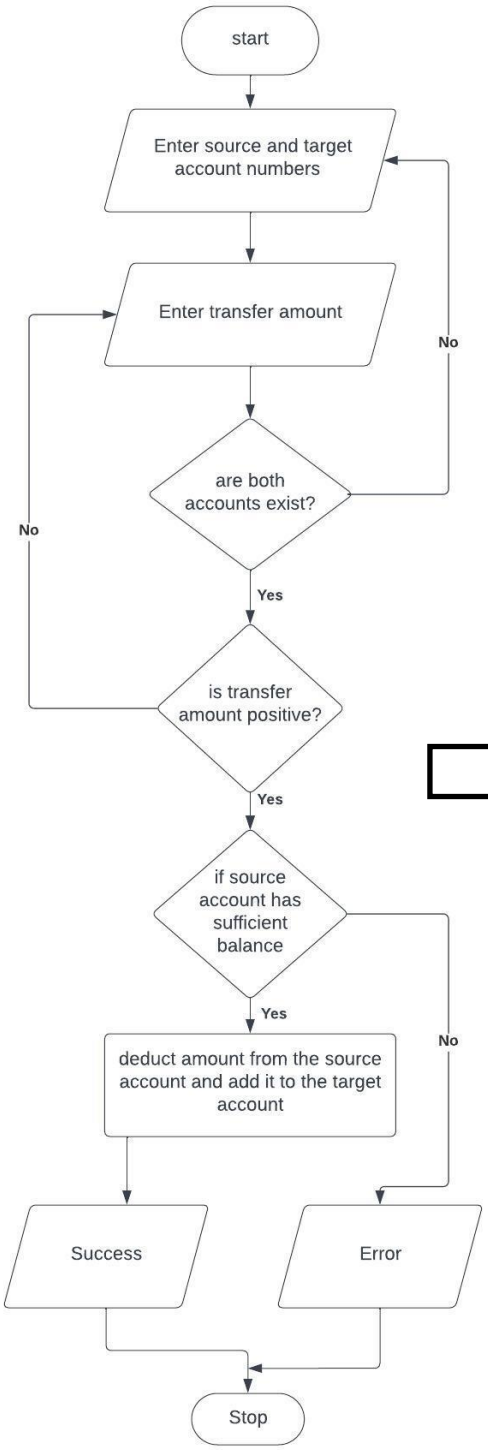

```
def deposit(accounts):
    account_number = input("Enter account
number: ")
    if account_number not in accounts:
        print("Account does not exist.")
        return
    amount = float(input("Enter deposit
amount: "))
    accounts[account_number].deposit(amount)
```

3. Withdrawing money



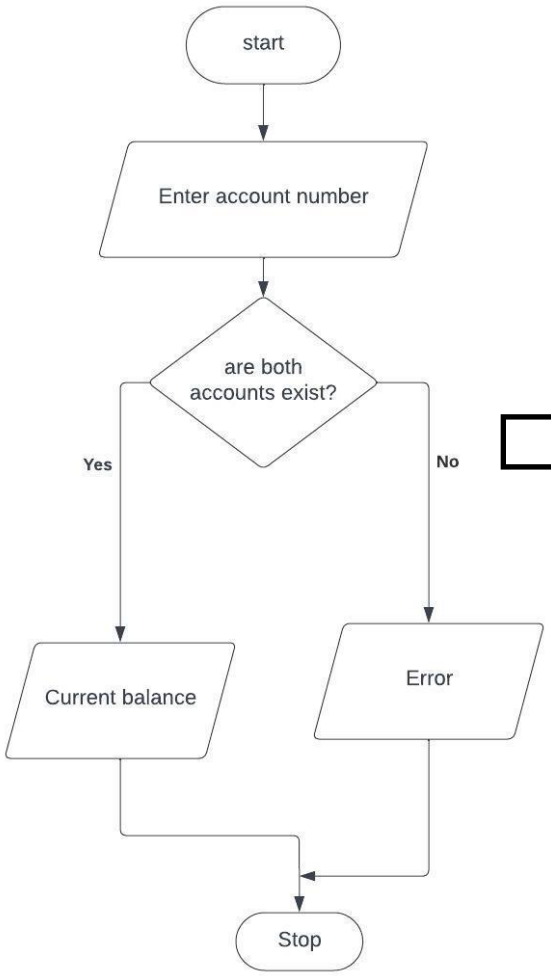

```
def withdraw(accounts):
    account_number = input("Enter account number: ")
    if account_number not in accounts:
        print("Account does not exist.")
        return
    amount = float(input("Enter withdrawal amount: "))
    accounts[account_number].withdraw(amount)
```

4. Transferring money




```
def transfer(accounts):
    from_account = input("Enter source
account number: ")
    to_account = input("Enter target account
number: ")
    if from_account not in accounts or
to_account not in accounts:
        print("One or both accounts do not
exist.")
        return
    amount = float(input("Enter transfer
amount: "))
    accounts[from_account].transfer(accounts[
to_account], amount)
```

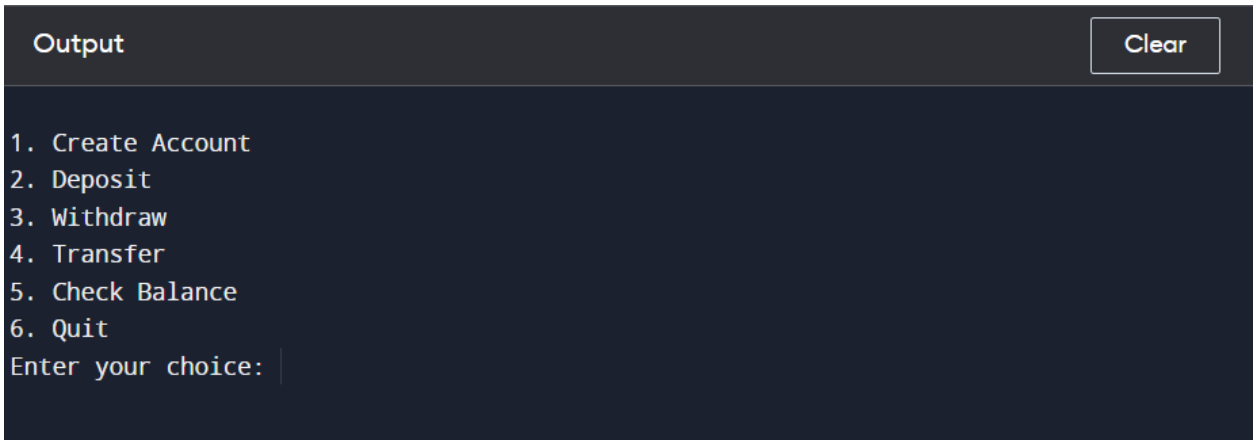
5. Checking Account Balance




```python
def check_balance(accounts):
    account_number = input("Enter account number: ")
    if account_number not in accounts:
        print("Account does not exist.")
        return
    balance = accounts[account_number].check_balance()
    print("Current balance:", balance)
```

Here is the completed python code and outputs:

```
 90 def main():
 91     accounts = {}
 92     while True:
 93         print("\n1. Create Account\n2. Deposit\n3. Withdraw\n4. Transfer\n5.
            Check Balance\n6. Quit")
 94         choice = input("Enter your choice: ")
 95         if choice == '1':
 96             create_account(accounts)
 97         elif choice == '2':
 98             deposit(accounts)
 99         elif choice == '3':
100             withdraw(accounts)
101         elif choice == '4':
102             transfer(accounts)
103         elif choice == '5':
104             check_balance(accounts)
105         elif choice == '6':
106             break
107         else:
108             print("Invalid choice. Please try again.")
109
110
```



Choice 1 – creating the account

```
43 def create_account(accounts):
44     account_number = input("Enter account number: ")
45     if account_number in accounts:
46         print("Account already exists.")
47         return
48     initial_balance = float(input("Enter initial balance: "))
49     accounts[account_number] = BankAccount(account_number, initial_balance)
50     print("Account created successfully.")
51
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 1
Enter account number: 123456
Enter initial balance: 20000
Account created successfully
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 1
Enter account number: 654321
Enter initial balance: 300000
Account created successfully
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 1
Enter account number: 123456
Account already exists
```

Choice 2 – depositing money

```
    def deposit(self, amount):
        if amount >= 0:
            self.balance += amount
            return True
        else:
            print("Deposit amount must be non-negative.")
            return False

```

```

def deposit(accounts):
    account_number = input("Enter account number: ")
    if account_number not in accounts:
        print("Account does not exist.")
        return
    amount = float(input("Enter deposit amount: "))
    accounts[account_number].deposit(amount)


```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 2
Enter account number: 123456
Enter deposit amount: 20000
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 2
Enter account number: 123456
Enter deposit amount: -20000
Deposit amount must be non-negative
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 2
Enter account number: 654321
Account does not exist
```

Choice 3 – withdrawing the money

```
13
14      def withdraw(self, amount):
15          if amount >= 0:
16              if self.balance >= amount:
17                  self.balance -= amount
18                  return True
19              else:
20                  print("Insufficient balance.")
21                  return False
22          else:
23              print("Withdrawal amount must be non-negative.")
24              return False
```

```
62  def withdraw(accounts):
63      account_number = input("Enter account number: ")
64      if account_number not in accounts:
65          print("Account does not exist.")
66          return
67      amount = float(input("Enter withdrawal amount: "))
68      accounts[account_number].withdraw(amount)
69
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 3
Enter account number: 123456
Enter withdrawal amount: 10000
```

- If initial balance=0,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 3
Enter account number: 123456
Enter withdrawal amount: 10000
Insufficient balance
```

- If account doesn't exist,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 3
Enter account number: 654512
Account does not exist
```

- If withdrawal amount<0,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 3
Enter account number: 123456
Enter withdrawal amount: -20000
Withdrawal amount must be non-negative
```

Choice 4 – Transfering the money

```
25
26      def transfer(self, other_account, amount):
27          if amount >= 0:
28              if self.balance >= amount:
29                  self.balance -= amount
30                  other_account.balance += amount
31                  return True
32              else:
33                  print("Insufficient balance.")
34                  return False
35          else:
36              print("Transfer amount must be positive.")
37              return False
38
39      def check_balance(self):
40          return self.balance
41
42
```

```
70
71  def transfer(accounts):
72      from_account = input("Enter source account number: ")
73      to_account = input("Enter target account number: ")
74      if from_account not in accounts or to_account not in accounts:
75          print("One or both accounts do not exist.")
76          return
77      amount = float(input("Enter transfer amount: "))
78      accounts[from_account].transfer(accounts[to_account], amount)
79
80
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 4
Enter source account number: 123456
Enter target account number: 654321
Enter transfer amount: 10000
```

- If initial balance=0,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 4
Enter source account number: 123456
Enter target account number: 654321
Enter transfer amount: 20000
Insufficient balance
```

- If account doesn't exist,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 4
Enter source account number: 123456
Enter target account number: 655432
One or both accounts do not exist
```

- If transfer amount<0,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 4
Enter source account number: 123456
Enter target account number: 654321
Enter transfer amount: -10000
Transfer amount must be non-negative
```

Choice 5 – Checking the balance

```
38
39      def check_balance(self):
40          return self.balance
41
42
```

```
81  def check_balance(accounts):
82      account_number = input("Enter account number: ")
83      if account_number not in accounts:
84          print("Account does not exist.")
85          return
86      balance = accounts[account_number].check_balance()
87      print("Current balance:", balance)
88
89
```

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 5
Enter account number: 123456
Current balance:  200000.0
```

- If account doesn't exist,

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 5
Enter account number: 123456
Account does not exist
```

Choice 6 – Quit

```
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Check Balance
6. Quit
Enter your choice: 6

=== Code Execution Successful ===
```

Thank You!