`

I Project Introduction

1.1 Overview

//Write a paragraph explain what is your project about

Bank Account Management System is a comprehensive platform that facilitates seamless financial transactions. It enables users to deposit and withdraw funds, as well as transfer money between accounts. Moreover, it efficiently manages the queue of clients, ensuring smooth and organized service delivery. This system also maintains a detailed record of past transactions, allowing for easy tracking and monitoring of financial activities.

1.2 Problem and solution

//Here the students write about the project problem and description of your solution.

The bank management system may encounter various issues, including technical glitches that hinder access to funds and transactional services. Data loss or system failures can lead to a loss of crucial financial information, causing inconvenience and potential security risks for customers. Inadequate client queue management may result in poor service delivery, leading to prolonged wait times and dissatisfied customers.

In order to address these problems we used data structures, in the transactional services field we used Singly Linked List to makes it easier to the user to create an account and preform financial transaction, and to prevent the transaction data loss we used Stack to record every transaction and to access or delete it easily. To enhance the service delivery and customer service every user must be in a queue that can be smoothly managed.

1.3 Solution justification

//write reasons of choosing the data structure you used.

We used the three data structures: Stack, Linked List, and Queue.

-Stack: were used to record the transactions that have been made, and can be accessed from the nearest transaction to the all past transactions, it allows the user to manage the transactions history.

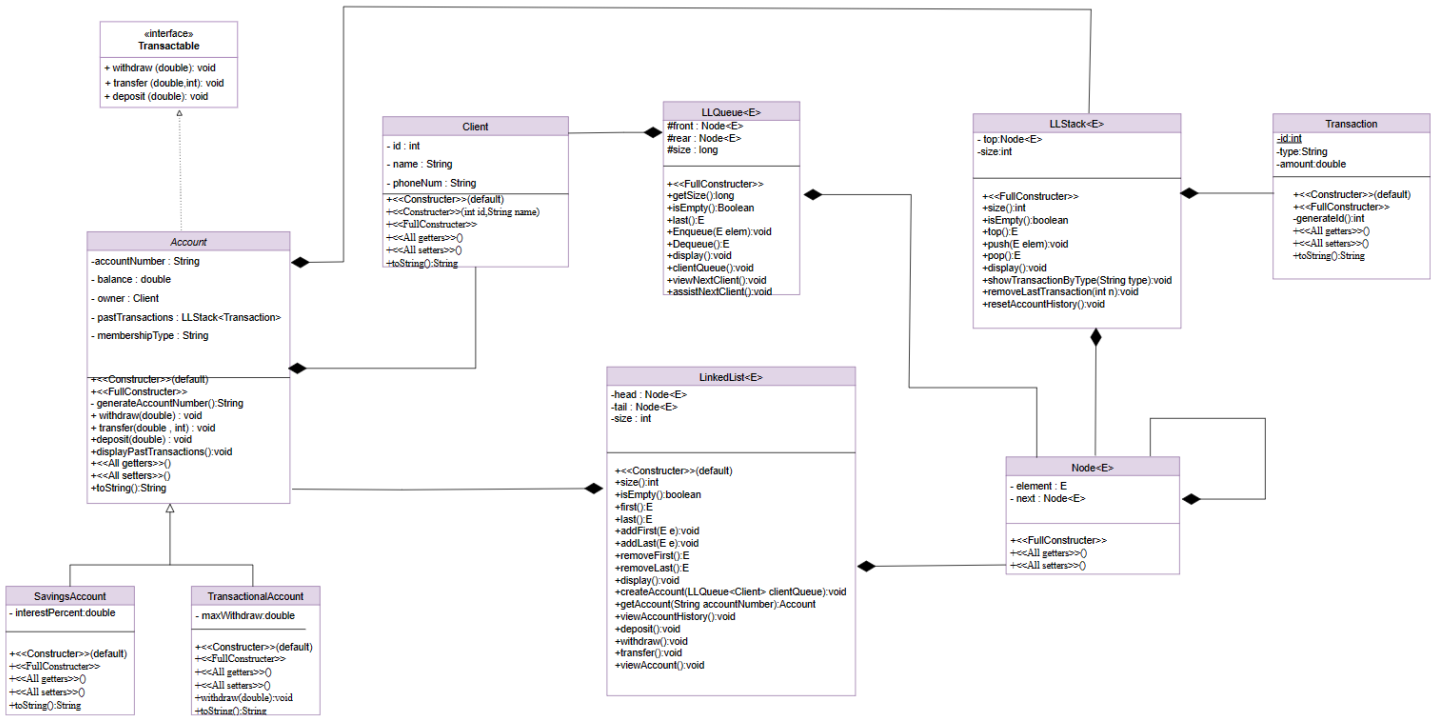
-Singly Linked List: were used to add the accounts that have been made, the reason for choosing this specific data structure that the order is not that important and it makes no difference.

-Queue: were used to manage the queue of clients, so that the first client in the queue will be served first and to achieve customer satisfaction.

`

II Project class diagram

//Here the students put the class diagram they have constructed for the project.



`

III Concepts covered

//Here the students have to put each topic covered with a screen shot that shows the part of code that have been used for the topic.

III.1 Data Structure: Queue – used for Clients

III.1.1 Method #1 : clientQueue( )

```
public void clientQueue() {
    if (isEmpty()) {
        System.out.println("Zero clients in the queue!");
    } else {
        int size = (int) getSize();
        System.out.print("[1st] ");
        for (int i = 0; i < size; i++) {
            Client client = (Client)Dequeue();
            System.out.print(client.getName() + " -> ");
             Enqueue((E)client);
        }
    }
}
```

III.1.2 Method #2 : viewNextClient( )

```
public void viewNextClient() {
    if (isEmpty()) {
        System.out.println("Zero clients in the queue.");
    } else {
        Client client = (Client)first();
        System.out.print("The next client in the queue is '" + client.getName() + "'");
    }
}
```

III.1.3 Method #3: assistsNextClient( )

```
public void assistNextClient() {
    if (isEmpty()) {
        System.out.println("Zero clients in the queue");
    } else {
        Client client = (Client)Dequeue();
        System.out.println("Banker is assisting '" + client.getName() + "' in the moment");
        viewNextClient();
    }
}
}
```

`

III.2 Data Structure: Singly Linked List – used for Account

III.2.1 Method #1: createAccount( )

```
public void createAccount(LLQueue<Client> clientQueue) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your id: ");
    int id = input.nextInt();
    input.nextLine();
    System.out.print("Enter your name: ");
    String name = input.nextLine();
    System.out.print("Enter your phone number: ");
    String phone = input.next();
    Client client = new Client(id, name, phone);
    clientQueue.Enqueue(client);
    Account account;
    int accountTypeOption;
    do {
        System.out.print("Select Account Type: \n"
                + "1-> Transactional Account\n"
                + "2-> Savings Account\n"
                + "--> ");
        accountTypeOption = input.nextInt();

    } while (accountTypeOption < 1 || accountTypeOption > 2);

    if (accountTypeOption == 1) {
        account = new TransactionalAccount(client);
    } else {
        account = new SavingsAccount(client);
    }
    addLast((E) account);
    System.out.println("Your account has been created successfully!");
    System.out.println(account);
}
```

III.2.2 Method #2: getAccount( )

```
public Account getAccount(String accountNumber) {
    Account account = null;
    int size = size();
    for (int i = 0; i < size; i++) {
        Account acc = (Account) removeFirst();
        if (acc.getAccountNumber().equals(accountNumber)) {
            account = acc;
        }
        addLast((E) acc);
    }
    return account;
}
```

`

III.2.3 Method #3: viewAccountHistory( )

```
public void viewAccountHistory() {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your account number: ");
    String accountNumber = input.next();
    Account account = getAccount(accountNumber);
    if (account != null) {
        account.displayPastTransactions();
    } else {
        System.out.println("Account with number " + accountNumber + " is not found!");
    }
}
```

III.2.4 Method #4: deposit( )

```
public void deposit() {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your account number: ");
    String accountNumber = input.next();
    Account account = getAccount(accountNumber);
    if (account != null) {
        System.out.print("Enter deposit amount: ");
        double amount = input.nextDouble();
        account.deposit(amount);
    } else {
        System.out.println("Account with number " + accountNumber + " is not found!");
    }
}
```

III.2.5 Method #5: withdraw( )

```
public void withdraw() {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your account number: ");
    String accountNumber = input.next();
    Account account = getAccount(accountNumber);
    if (account != null) {
        System.out.print("Enter withdrawal amount: ");
        double amount = input.nextDouble();
        account.withdraw(amount);
    } else {
        System.out.println("Account with number " + accountNumber + " is not found!");
    }
}
```

`

III.2.6 Method #6: transfer( )

```
public void transfer() {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your account number: ");
    String accountNumber = input.next();
    Account account = getAccount(accountNumber);

    if (account != null) {
        System.out.println("Enter Receiver  Name: ");
        String receiverName = input.next();
        System.out.println("Enter Receiver  ID: ");
        int receiverID = input.nextInt();
        Client receiver = new Client(receiverID, receiverName);
        System.out.print("Enter the amount you want to transfer: ");
        double amount = input.nextDouble();
        account.transfer(amount, receiverID);
    } else {
        System.out.println("Account with number " + accountNumber + " is not found!");
    }
}
```

III.2.7 Method #7: viewAccount( )

```
public void viewAccount() {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your account number: ");
    String accountNumber = input.next();
    if (isEmpty()) {
        System.out.println("No accounts!");
    } else {
        Account account = getAccount(accountNumber);
        if (account != null) {
            System.out.println(account);
        } else {
            System.out.println("Account with number " + accountNumber + " is not found!");
        }
    }

}
```

`

III.3 Data Structure: Stack – used for Transactions

III.3.1 Method #1: showTransactionsByType( )

```
public void showTransactionsByType(String type) {
    LLStack<E> tempStack = new LLStack<>();
    while (!isEmpty()) {
        E ele = pop();
        if (((Transaction) ele).getType().equalsIgnoreCase(type)) {
            System.out.println(ele);
            System.out.println("----------------------------------------");
        }
        tempStack.push(ele);
    }

    // Restoring the original stack
    while (!tempStack.isEmpty()) {
        push(tempStack.pop());
    }
}
```

III.3.2 Method #2: removeLastTransactions( )

```
public void removeLastTransactions(int n) {

    if (size() >= n) {
        for (int i = 0; i < n; i++) {
            pop();
        }
        System.out.println("Last " + n + " transactions have been removed!");
    } else {
        System.out.println("There is no " + n + " transactions in your account!");
    }

}
```

III.3.3 Method #3: resetAccountHistory( )

```
public void resetAccountHistory() {
    while (!isEmpty()) {
        pop();
    }
    System.out.print("Account history has been rest!");
}
```

`

IV Main Class

//The students have to explain each step of the main class And put a screen shot of the output.

The Main class import Scanner and contains 2 void methods: the main method + **DemoData()** (used only for presenting), it also contains 2 static objects 1: from **LLQueue** class an object of Client Queue, 2: from **LinkedList** class an object of Account List. They're used to call the methods from the ADTs.

In the main method we used switch to call the methods from the ADTs, the switch will continue to loop until the user enters 0 to leave, it contains 10 cases in addition to the default case.

- **Case 1:** Client Queue; it will call the method **clientQueue()** from class **LLQueue,** and it will print all the clients in the queue by Dequeue() + Enqueue() , and labeling the first in the queue.

- **Case 2:** View Next Client; it will call the method **viewNextClient ()** from class **LLQueue**, it will print the next client in the queue by using method **first()** in the **LLQueue**.

- **Case 3:** Assist Next Client; it will call the method **assistNextClient()** from class **LLQueue**, it will assist the next client in queue by **Dequeue()** , and printing this client and calls **viewNextClient ()** to print the next client in the queue.

- **Case 4:** Account Creation; it will call method **createAccount()** from class **LinkedList**, that will take the client queue as a parameter, the method will ask the user to enter the id, name, and phone number, we will create an object of type Client to add the user information, then we will **Enqueue()** the object int the queue of clients. Inside a do-while we asked the user to choose the account type whether Transactional or Savings Account, and according to the users choice we will create an object of the super type Account that stores the a new object of the sub class (Transactional or Savings Account) and takes as a parameter the object of Client we created before. Then we will **addLast()** in the list of accounts the object of Account we created before, lastly it will print a succession message and the information of the account and client.

- **Case 5:** View All Past Transaction; it will call the method **viewAccountHistory()** from class **LinkedList**, the method will ask the user to enter the account number, then uses a method defined in the same ADT which is method **getAccount () **** that will a String account number as a parameter, we defined an object of type Account and assigned it to null, and inside a for loop we defined an object of type Account that will use **removeFirst()** to get the first element and compare it with the account number, if it equals it will assign the object removed from the list to the object we assigned to null before, then it will add the object back in the list using **addLast()** and return the object of type Account.****** So the method **viewAccountHistory()** will call **getAccount ()** and uses the number the user entered, if the account doesn't equals null we will call method **displayPastTransactions()** from class **Account** that will call method display in the **LLStack** class that will **pop()** every transaction the user did that was pushed in the stack. Lastly if account is null it will print that the account with the account number is not found.

`

- **Case 6:** Deposit; it will call method **deposit()** from class **LinkedList**, the method will ask the user to enter the account number, then it will create an object of type Account and assign it to the call of method **getAccount ()** *defined in case 5* and uses the number the user entered as a parameter, if the account doesn't equals null, it will ask the user to enter the amount they want to deposit, then we will call method **deposit()** from class **Account** that takes a deposited amount as a parameter, the **deposit()** in class **Account** will check it the amount is less than or equal to zero it will print invalid amount, else it will sum the user account balance with the deposited amount and print a succession message + the new account balance, lastly we defined an object of class Transaction that takes the transaction String type and the amount then push it in the stack of **pastTransactions** . If the account in method **deposit()** in class **LinkedList** is null it will print that the account is not found.

- **Case 7:** Withdraw ; it will call method **withdraw ()** from class **LinkedList**, the method will ask the user to enter the account number, then it will create an object of type Account and assign it to the call of method **getAccount ()** *defined in case 5* and uses the number the user entered as a parameter, if the account doesn't equals null, it will ask the user to enter the amount they want to withdraw, then we will call method **withdraw()** from class **Account** that takes a withdrawal amount as a parameter, the **withdraw ()** in class **Account** will check it the amount is less than or equal to zero or the amount is greater than the account balance it will print invalid amount, else it will subtract the user account balance with the withdrawal amount and print a succession message + the new account balance, lastly we defined an object of class Transaction that takes the transaction String type and the amount then push it in the stack of **pastTransactions** . If the account in method **withdraw ()** in class **LinkedList** is null it will print that the account is not found.

- **Case 8**: Transfer; it will call method **transfer ()** from class **LinkedList**, the method will ask the user to enter the account number, then it will create an object of type Account and assign it to the call of method **getAccount ()** *defined in case 5* and uses the number the user entered as a parameter, if the account doesn't equals null, it will ask the user to enter the receiver name, id, and the amount they want to transfer, then we will call method **transfer ()** from class **Account** that takes a transfer amount and receiver id as parameters, the **transfer ()** in class **Account** will check it the amount is less than or equal to zero or the amount is greater than the account balance it will print invalid amount, else it will subtract the user account balance with the transfer amount and print a succession message + the new account balance, lastly we defined an object of class Transaction that takes the transaction String type and the amount then push it in the stack of **pastTransactions** . If the account in method **transfer ()** in class **LinkedList** is null it will print that the account is not found.

- **Case 9:** View Account; it will call method **viewAccount ()** from class **LinkedList**, the method will ask the user to enter the account number. In if statement we will call method **isEmpty()** to check if the list is empty or not, if not we will create an object of type Account and assign it to the call of method **getAccount ()** *defined in case 5* and uses the number the user entered as a parameter, if the account doesn't equals null, we will print the object of Account type. If it equals null we will print that the account is not found.

- **Case 10**: Reset Account History; it will ask the user to enter the account number and we will create an object of type Account and assign it to the call of method **getAccount ()** *defined in case 5* and uses the number the user entered as a parameter, if account is not null, we will define a stack to call

`

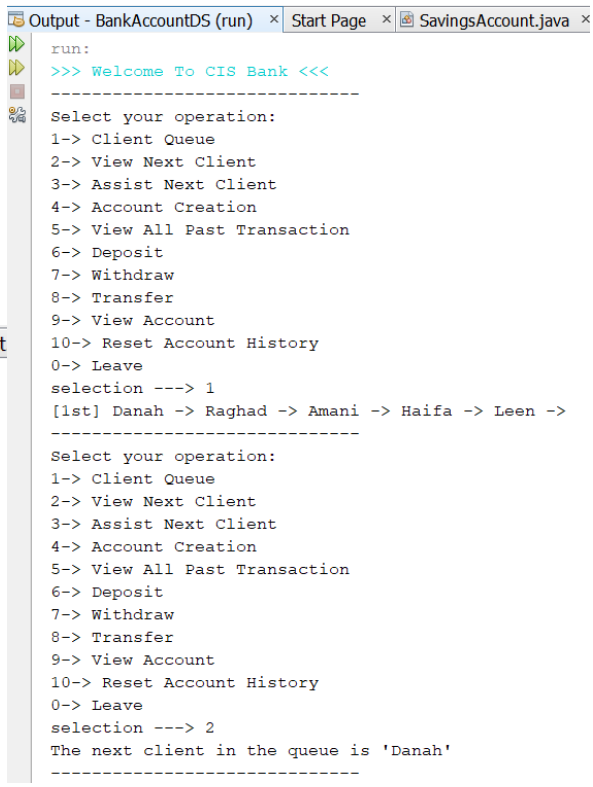
the **pastTransactions,** then by this stack we will call method **resetAccountHistory()** from class **LLStack**, that will **pop()** all the elements while the stack is not empty.

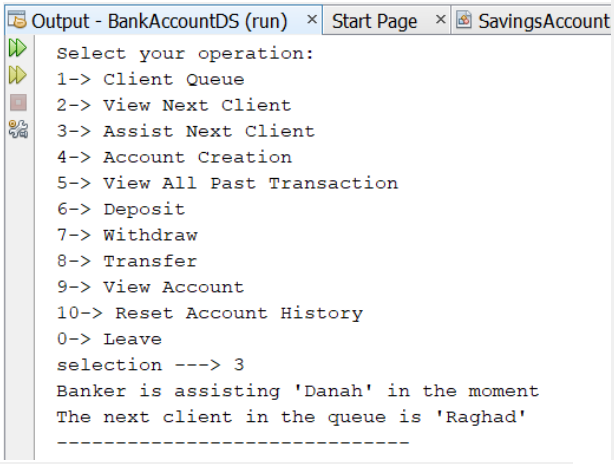
- **Case 0:** Leave; the user will enter zero to stop the loop and leave the system, we will print "thank you" statement.

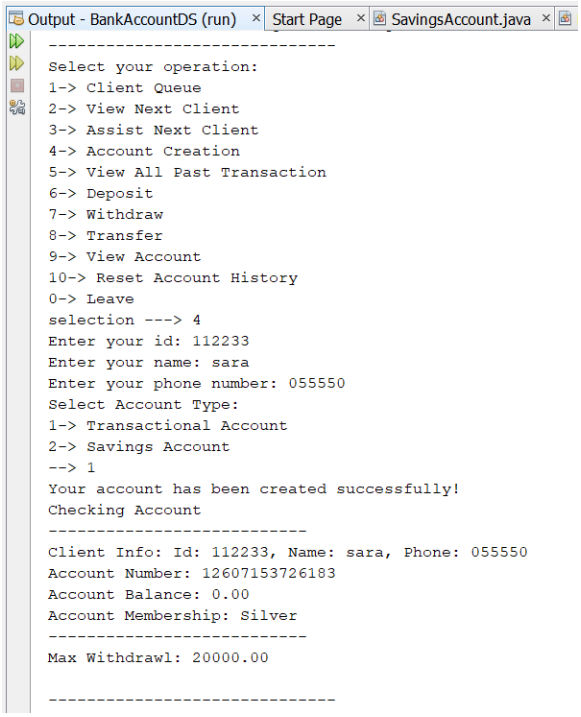
- **Default:** it will print "invalid selection".

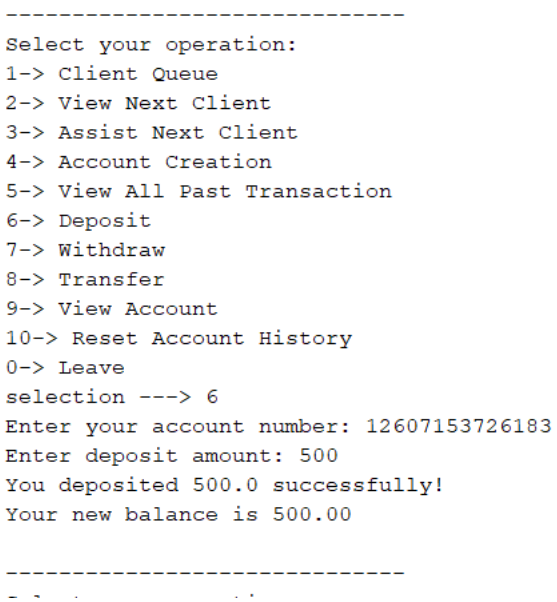
-All the cases are in try, catch(**InputMismatchException exception**) and if invalid input is entered it will print a message and asks the user the re input.

`


Output - BankAccountDS (run)
Start Page
SavingsAccount.java
run:
>>> Welcome To CIS Bank <<<
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 1
[1st] Danah -> Raghad -> Amani -> Haifa -> Leen ->
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 2
The next client in the queue is 'Danah'


Output - BankAccountDS (run)
Start Page
SavingsAccount
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 3
Banker is assisting 'Danah' in the moment
The next client in the queue is 'Raghad'


Output - BankAccountDS (run)
Start Page
SavingsAccount.java
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 4
Enter your id: 112233
Enter your name: sara
Enter your phone number: 055550
Select Account Type:
1-> Transactional Account
2-> Savings Account
--> 1
Your account has been created successfully!
Checking Account
Client Info: Id: 112233, Name: sara, Phone: 055550
Account Number: 12607153726183
Account Balance: 0.00
Account Membership: Silver
Max Withdrawl: 20000.00


Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 6
Enter your account number: 12607153726183
Enter deposit amount: 500
You deposited 500.0 successfully!
Your new balance is 500.00

`

```
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 7
Enter your account number: 12607153726183
Enter withdrawal amount: 100
You withdrew 100.0 successfully!
Your new balance is 400.00

------------------------------
```

```
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 8
Enter your account number: 59273697634174
Enter Receiver  Name:
danah
Enter Receiver  ID:
22
Enter the amount you want to transfer: 200
You transferd to 22, 200.0 successfully!
Your new balance is 300.00

------------------------------
```

```
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 9
Enter your account number: 12607153726183
Checking Account
----------------------------
Client Info: Id: 112233, Name: sara, Phone: 055550
Account Number: 12607153726183
Account Balance: 200.00
Account Membership: Silver
----------------------------
Max Withdrawl: 20000.00

------------------------------
```

```
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 5
Enter your account number: 12607153726183
Account Transaction History:
Id: 713, Type: Transfered, Amount: 200.00
----------------------------------------
Id: 422, Type: Withdrawal, Amount: 100.00
----------------------------------------
Id: 590, Type: Deposit, Amount: 500.00
----------------------------------------

------------------------------
```

`

```
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 10
Enter your account number: 12607153726183
Account history has been rest!
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 5
Enter your account number: 12607153726183
Account Transaction History:
No Transactions, Stack is empty!

------------------------------
```

```
------------------------------
Select your operation:
1-> Client Queue
2-> View Next Client
3-> Assist Next Client
4-> Account Creation
5-> View All Past Transaction
6-> Deposit
7-> Withdraw
8-> Transfer
9-> View Account
10-> Reset Account History
0-> Leave
selection ---> 0
Thank you!
BUILD SUCCESSFUL (total time: 1 minute 38 seconds)
```