BANK SYSTEM

SERGIU

TABLE OF CONTENTS

1) INTRODUCTION.....2
2) INSTRUCTIONS TO RUN THE PROGRAM 2
3) ARCHITECTURE OF SOFTWARE.....3
4) CLASS DESCRIPTION 4
5) DATA STRUCTURE AND ALGORITMS 7
6) PROGRAM TESTS 8
7) USER INTERFACE.....17
8) CONCLUSION 18
9) REFERENCES 19

1) INTRODUCTION

The project aims to enhance understanding of data structures by developing custom structures for managing bank accounts and transactions. It involves creating a bank system where account data is stored in a custom structure, and transaction data is handled in a separate, purpose-built structure. This approach focuses on applying custom data structures in a real-world context.

2) INSTRUCTIONS TO RUN THE PROGRAM

Follow the instructions to run the program:

I. Download the ZIP folder from WebLearn.

II. Extract the contents:
- Right-click the downloaded ZIP folder and select "Extract All."
- Choose a destination folder and click "OK."

III. Open the IDE app:
- Click "File" in the menu bar.
- Select "Open Project" from the dropdown menu.

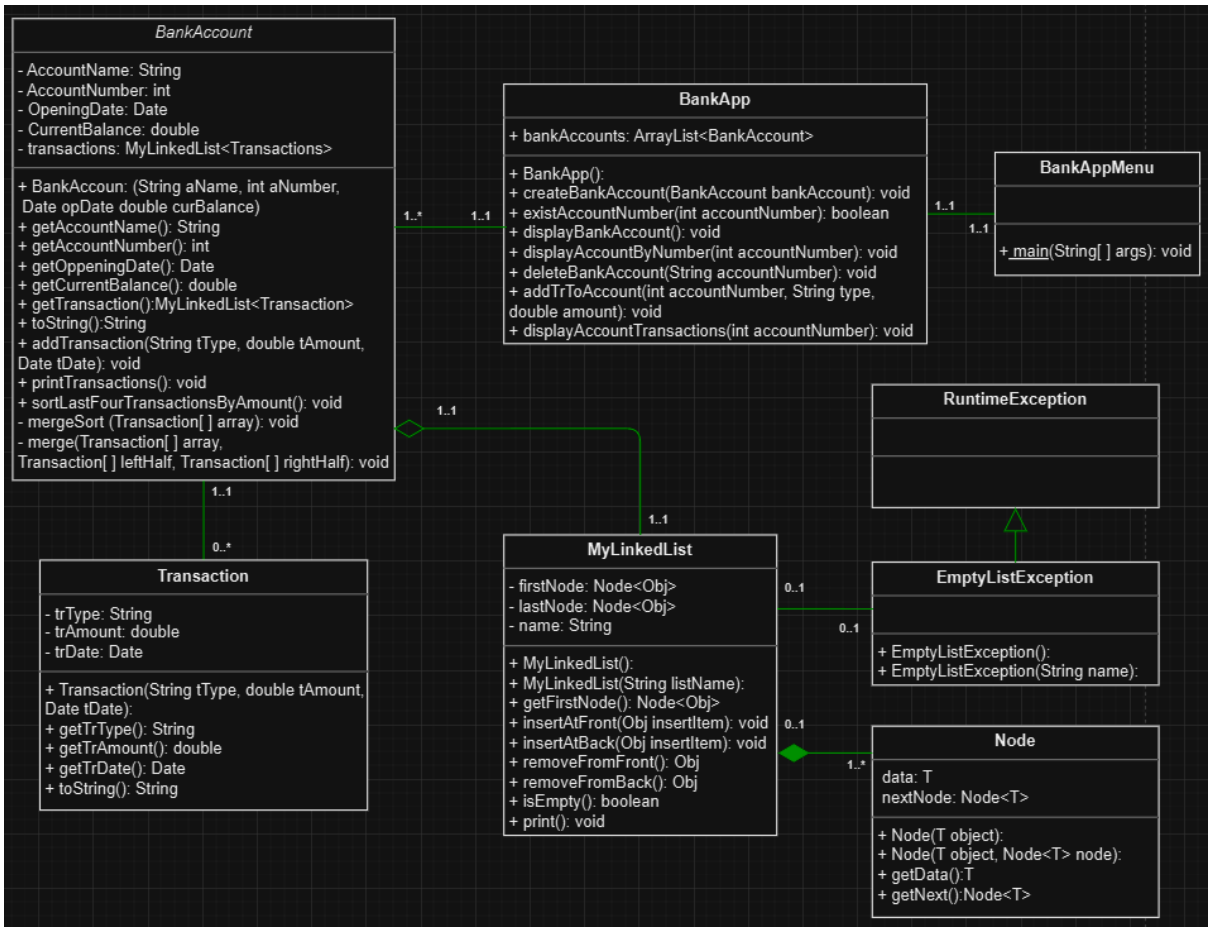
IV. Locate the extracted project folder and select it. Click "Open Project."

V. In the Projects section on the left:
- Locate and expand the BankApp project by clicking the arrow next to it.
- Expand the "Source Packages" folder.
- Expand "com.mycompany.bankapp" to view the Java files.

VI. Run the program:
- Right-click on "BankAppMenu.java" from the list of Java files.
- Select "Run File."

3) ARCHITECTURE OF SOFTWARE

My bank account system consists of eight classes that are interconnected through various relationships, including association, aggregation, composition, and inheritance.



4) CLASS DESCRIPTION

1. The **BankAccount** class

Atributes	**Description**
private String AccountName; private int AccountNumber; private Date OpeningDate; private double CurrentBalance;	Define an account.
private MyLinkedList<Transaction> transactions;	Attribute is of type MyLinkedList to store transactions.
Methods	**Description**
public BankAccount(String aName, int aNumber, Date opDate, double curBalance)	Assign values to attributes when creating an instance.
public String getAccountNume(), public int getAccountNumber(), public Date getOpeningDate(), public double getCurrentBalance(), public MyLinkedList<Transaction> getTransaction()	Retrieve the values of the attributes.
@Override public String toString()	Method to display account information
public void addTransaction(String tType, double tAmount,Date tDate)	Create a new transaction
public void printTransactions()	Prints the transaction info
public void sortLastFourTransactionsByAmount(),private void merge(Transaction[] array, Transaction[] leftHalf, Transaction[] rightHalf), private void mergeSort(Transaction[] array)	Extracts last four transactions and sort them by amount.

2. The **Transaction** class, have association relationship with the **BankAccount** class.

Atributes	**Description**
private String trType; private double trAmount; private Date trDate;	Define transaction instances.
Methods	**Description**
public Transaction(String tType, double tAmount, Date tDate)	Assign the values when a transaction is created.
public String getTrType(), public double getTrAmount(), public Date getTrDate()	Retrieve values.
@Override public String toString()	Helps to print transaction information.

3. The **BankApp** class

Atributes	**Description**
public ArrayList<BankAccount> bankAccounts;	bankAccounts attribute of type ArrayList to store bank accounts
Methods	**Description**
public void createBankAccount(BankAccount bankAccount)	Creates a new bank account.
public boolean existAccountNumber(int accountNumber)	Check if an account already exists in the array list.
public void displayBankAccount() public void displayAccountByNumber(int accountNumber)	Displays accounts information
public void deleteBankAccount(String accountNumber	Deletes an account from the system
public void addTrToAccount(int accountNumber, String type, double amount)	Add a transaction to a specific account.
public void displayAccountTransactions(int accountNumber)	Display transactions of specific account.

4. The **BankAppMenu class** serves as the user interface, allowing interaction with the system. It primarily calls methods from the BankApp class to execute operations.

Methods	**Description**
public static void main(String[] args)	Run the bank system program

5. The **MyLinkedList** class is custom linked list used to store transaction objects.

Atributes	**Description**
private Node< Obj > firstNode; private Node< Obj > lastNode;	Store references to the first and last nodes.
private String name;	Attribute to store name of the list
Methods	**Description**
public MyLinkedList() public MyLinkedList(String listName)	Constructor methods to create the list
public void insertAtFront(Obj insertItem) public void insertAtBack(Obj insertItem)	Insert objects into the list
public Obj removeFromFront() public Obj removeFromBack()	Remove objects from the list
public boolean isEmpty()	Check if the list is empty
public void print()	Print the list

6. The **Node Class** facilitates the creation of individual node objects for the MyLinkedList class.

Atributes	**Description**
T data;	Holds data
Node< T > nextNode;	Reference to next node
Methods	**Description**
public Node(T object) public Node(T object, Node<T> node)	Creates a node and sets the next node
public T getData()	Return node data
public Node< T > getNext(){	Return next node reference

7. The **EmptyListException** class is a subclass of Java's RuntimeException, used to handle cases where a linked list is empty, preventing program crashes.

8. **RuntimeException** is part of the Java standard library, it doesn't require an explicit import. The inheritance relationship between EmptyListException and RuntimeException is shown in the class diagram for clarity.

5) DATA STRUCTURE AND ALGORITMS

In the Bank App System, I used an **ArrayList** to store and manage bank accounts in the BankApp class. The ArrayList offers easy and efficient access to account data, allowing for quick retrieval by index, which is ideal for displaying account details. Its dynamic nature also provides flexibility for adding or removing accounts without needing to manage memory allocation manually. This is an imported from Java library data structure.

For the second data structure, which required implementing a custom data structure, I used a **LinkedList**. The LinkedList's structure allows for easy insertion and removal of objects from both the front and back, and it can be easily transformed into a queue or stack. This structure was used in BankAccount class to store the transactions. To implement the classes in my custom package, I followed the example provided in the book by (Deitel & Deitel, 2017).

I chose the **Merge Sort** algorithm to sort the transactions, despite the small size of the list, because of its efficiency and recursive approach, which I found both challenging and rewarding to implement the development was inspired by the work of (John, 2021)

6) PROGRAM TESTS

1) Create an account:

Insert all the values right

```
===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 1

Enter Account Name: sam
Enter Account Number: 1234
Enter Opening Balance: 0
New Bank Account created successfully.

Bank Account Details:
Account Name: SAM
Account Number: 1234
Opening Date: 05-Dec-2024
Your Current Balance: 0.0 £
```

Insert different type for the name

```
Enter Account Name: 24243
Account Name should contain only letters. Please try again.
```

Insert initial balance smaller than 0 or a different type

```
Enter Account Name: John
Enter Account Number: 1235
Enter Opening Balance: -6
Opening Balance must be greater than 0. Please try again.
Enter Opening Balance: fdsgsdg
Please enter a number greater than '0'
Enter Opening Balance:
```

2) Add Transaction to account:

```
Bank Account Details:
Account Name: SAM
Account Number: 1234
Opening Date: 05-Dec-2024
Your Current Balance: 0.0 £
```

Initial balance 0.0£

```
===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 5

Enter Account Number: 1234
Enter Transaction Type (Deposit/Withdraw): deposit
Enter Transaction Amount: 19
Your current balance is: 19.0 £
Transaction added successfully.
```

Current balance updated to 19£

```
Enter your choice: 5

Enter Account Number: 1234
Enter Transaction Type (Deposit/Withdraw): Withdraw
Enter Transaction Amount: 25
Your current balance is: -6.0 £
Transaction added successfully.
```

Call the addTransation method with “Withdraw” greater than “currentBalance” and it works which is not acceptable.

```
public void addTransaction(String tType, double tAmount,Date tDate){
    if (tType.equals("Withdraw") && tAmount > CurrentBalance){
        System.out.println("Insuficient Balance");
        return;
    }

    if (tType.equals("deposit")){
        CurrentBalance += tAmount;
        System.out.println("Your current balance is: " + getCurrentBalance() + " £" );
    }
    else if (tType.equals("withdraw")){
        CurrentBalance -= tAmount;
        System.out.println("Your current balance is: " + CurrentBalance + " £" );
    }
    else{
        System.out.println("Unknown type of trasaction please choose from 'Deposit' and 'Withdraw'");
        return;
    }

    transactions.insertAtFront(new Transaction(tType,tAmount,tDate));
    System.out.println("Transaction added successfully.");
}
```

The add transaction method in the BankAccount class

```
case 5: //Add Transaction to Account
    while(true) {
        try{
            System.out.print("Enter Account Number: ");
            int accNumberForTransaction = scanner.nextInt();
            System.out.print("Enter Transaction Type (Deposit/Withdraw): ");
            String transactionType = scanner.next();
            transactionType = transactionType.toLowerCase();
            System.out.print("Enter Transaction Amount: ");
            double transactionAmount = scanner.nextDouble();
            bankApp.addTrToAccount(accNumberForTransaction, transactionType,  transactionAmount);
            break;
        }
        catch(Exception e){
            System.out.println("Plese use digits to insert the amount");
            scanner.next();
        }
    }
    break;
case 6: //Display Transactions for an Account
```

The bug occurred because I added a method to convert the inputted value to lowercase but forgot to update the addTransaction method in the BankAccount class to account for this change.

```
Enter your choice: 5

Enter Account Number: 1234
Enter Transaction Type (Deposit/Withdraw): withdraw
Enter Transaction Amount: 25
Insuficient Balance
```

```
public void addTransaction(String tType, double tAmount,Date tDate){
    if (tType.equals("withdraw") && tAmount > CurrentBalance){
        System.out.println("Insuficient Balance");
        return;
    }
```

Now that I changed from “Withdraw” to “withdraw” it perfectly works.

3) Add more than 5 transactions to an account then display only last 4 sorted transactions

```
Enter Account Number to sort transactions: 1234
Last transactions sorted by amount:
Type: withdraw, Amount: 8.0, Date: 05-12-2024
Type: withdraw, Amount: 16.0, Date: 05-12-2024
Type: deposit, Amount: 33.0, Date: 05-12-2024
Type: deposit, Amount: 86.0, Date: 05-12-2024
----------------------------------------
Transactions for account 1234:
Type: withdraw, Amount: 8.0, Date: 05-12-2024
Type: deposit, Amount: 33.0, Date: 05-12-2024
Type: withdraw, Amount: 16.0, Date: 05-12-2024
Type: deposit, Amount: 86.0, Date: 05-12-2024
Type: deposit, Amount: 29.0, Date: 05-12-2024
Type: deposit, Amount: 12.0, Date: 05-12-2024
```

All the transaction in the linked list is introduced from front so the last introduced transaction is displayed as first in the list.

4) Attempting to sort the transactions of a non-existent account

```
===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 7

Enter Account Number to sort transactions: 55698
Account not found.
```

Sort transactions of an account which doesn't have any transactions:

```
Enter Account Name: Tim
Enter Account Number: 1596
Enter Opening Balance: 15
New Bank Account created successfully.

Bank Account Details:
Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £


===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 7

Enter Account Number to sort transactions: 1596
No transactions available to sort.
----------------------------------------
No transactions available.
```

5) Create a new account with the same number of an existent account:

```
Enter Account Name: Tim
Enter Account Number: 1596
Enter Opening Balance: 15
New Bank Account created successfully.

Bank Account Details:
Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £
```

Tim 1596 already existing account

New account with the same account number 1596 displays a message

```
===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 1

Enter Account Name: dan
Enter Account Number: 1596
Enter Opening Balance: 44

An  account number alredy exist. Please use a differntnt number
Bank Account Details:
Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £
```

This can be demonstrated by observing two accounts in the system, each with a unique account number.

```
Enter your choice: 1

Enter Account Name: tim
Enter Account Number: 1596
Enter Opening Balance: 15
New Bank Account created successfully.

Bank Account Details:
Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £


===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 1

Enter Account Name: dan
Enter Account Number: 1596
Enter Opening Balance: 44

ERROR: An account number alredy exist. Please use a differntnt number /n
Bank Account Details:
Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £


===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 3

In the Bank System are: 2 accounts

Account Name: SADASD
Account Number: 1234
Opening Date: 05-Dec-2024
Your Current Balance: 12.0 £

Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £


===== Bank App Menu =====
```

6) Attempt to insert as many accounts as possible.

```
In the Bank System are: 9 accounts

Account Name: SADASD
Account Number: 1234
Opening Date: 05-Dec-2024
Your Current Balance: 12.0 £

Account Name: TIM
Account Number: 1596
Opening Date: 05-Dec-2024
Your Current Balance: 15.0 £

Account Name: SAM
Account Number: 1753
Opening Date: 05-Dec-2024
Your Current Balance: 12.0 £

Account Name: TOM
Account Number: 1675
Opening Date: 05-Dec-2024
Your Current Balance: 66.0 £

Account Name: VICTOR
Account Number: 8854
Opening Date: 05-Dec-2024
Your Current Balance: 48.0 £

Account Name: JOHN
Account Number: 6636
Opening Date: 05-Dec-2024
```

7) Remove an account from the bank system

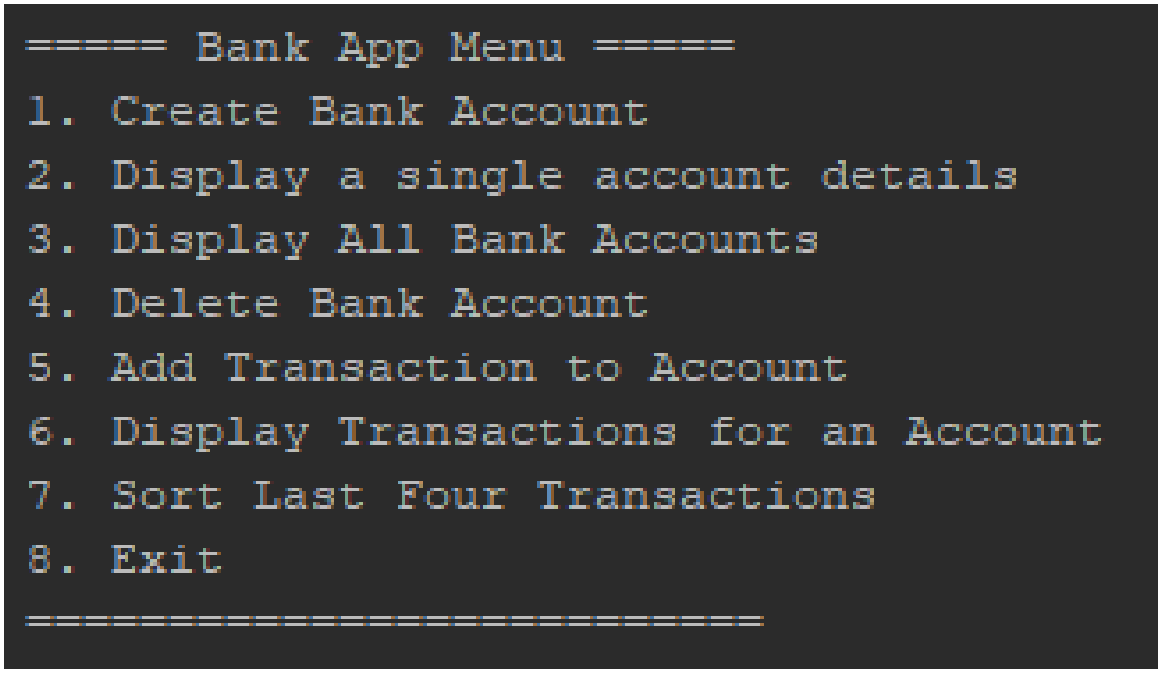
```
===== Bank App Menu =====
1. Create Bank Account
2. Display a single account details
3. Display All Bank Accounts
4. Delete Bank Account
5. Add Transaction to Account
6. Display Transactions for an Account
7. Sort Last Four Transactions
8. Exit
Enter your choice: 4

Enter Account Number to delete: 1596
ACCOUNT with account number: 1596 has been succesefuly removed
```

Previously in the bank system, there were 9 accounts, but now there are 8. Tim, with account number 1596, is no longer on the list.

```
In the Bank System are: 8 accounts

Account Name: SADASD
Account Number: 1234
Opening Date: 05-Dec-2024
Your Current Balance: 12.0 £

Account Name: SAM
Account Number: 1753
Opening Date: 05-Dec-2024
Your Current Balance: 12.0 £

Account Name: TOM
Account Number: 1675
Opening Date: 05-Dec-2024
Your Current Balance: 66.0 £

Account Name: VICTOR
Account Number: 8854
Opening Date: 05-Dec-2024
Your Current Balance: 48.0 £
```

7) USER INTERFACE

The user interface is a text-based menu for a bank application, offering eight options for user interaction. It is designed for easy navigation and efficient operation of the banking system.



To prevent errors when inserting text into the user interface, I used a while loop with a try-catch block. This ensures invalid input is handled smoothly, allowing the user to correct it without crashing the program.

8) CONCLUSION

When I started this coursework, I felt overwhelmed and unsure where to begin. While creating the BankAccount, Transaction, and BankApp classes was straightforward, the real challenge came with implementing data structures to store the data, which was a new concept for me. Through attending lectures, watching tutorials, and practicing, I gradually learned how to implement data structures in my project. My first obstacle was resolving syntactic errors, for which I used resources like (w3schools, 1999 - 2024) (Oracle, 1993 - 2024), and AI tools. For more complex bugs, I experimented with synchronization algorithms and sought inspiration from online videos (John, 2021). These efforts helped me successfully complete the project, and I believe the final product meets all required functionalities.

9) REFERENCES

Deitel, P. J. & Deitel, H. M., 2017. *Java: how to program early objects.* Edition 11 ed. New York: Pearson Education, Limited.

John, C. w., 2021. *Merge Sort Algorithm in Java - Full Tutorial with Source.* [Online] Available at: https://www.youtube.com/watch?v=bOk35XmHPKs&ab_channel=CodingwithJohn [Accessed 18 10 2024].

Oracle, 1993 - 2024. *Java™ Platform, Standard Edition 8.* [Online] Available at: https://docs.oracle.com/javase/8/docs/api/overview-summary.html [Accessed Oct - Dec 2024].

w3schools, 1999 - 2024. *TUTORIAL Learn JAVA.* [Online] Available at: https://www.w3schools.com/java/java_conditions_else.asp?goalId=7fd10b8f-82de-4f93-a382-5a8d30fff1c8 [Accessed Oct - Dec 2024].