BINUS UNIVERSITY

OOP FINAL PROJECT

PROJECT REPORT

Student Information :

Name: Albertus Santoso **NIM:** 2702334885 **Class:** L2AC

Course Information:

Course Code: COMP6699001 **Lecturer:** Jude Joseph Lamug Martinez, MCS

Project Specification:

1. **Background:**

   Everything is online these days from something as simple as applying online to something more complicated as ordering stuff online. So I wanted to make a management system since it is something that is interesting for me. The management system I eventually choose is a bank management system with the function of withdraw and deposit.

2. **Solution:**

   To solve this problem I made a bank management system. The System has a version with and without GUI. When the code is run there are options to create an account,deposit,withdraw,and see a list of accounts with balance. In the system if you have no account then you cannot withdraw or deposit since that means you have not opened an account for this bank. You need a minimum of 100 dollars for deposit and 50 to withdraw with fees for withdrawal and interest when deposit.

3. **Libraries used in the system:**

   - Java Swing:

     Since there is a version of my code that uses a GUI, I decided to use java swing since that is something that I have done before and it looks clean and simple to look at.
   - Java Util:

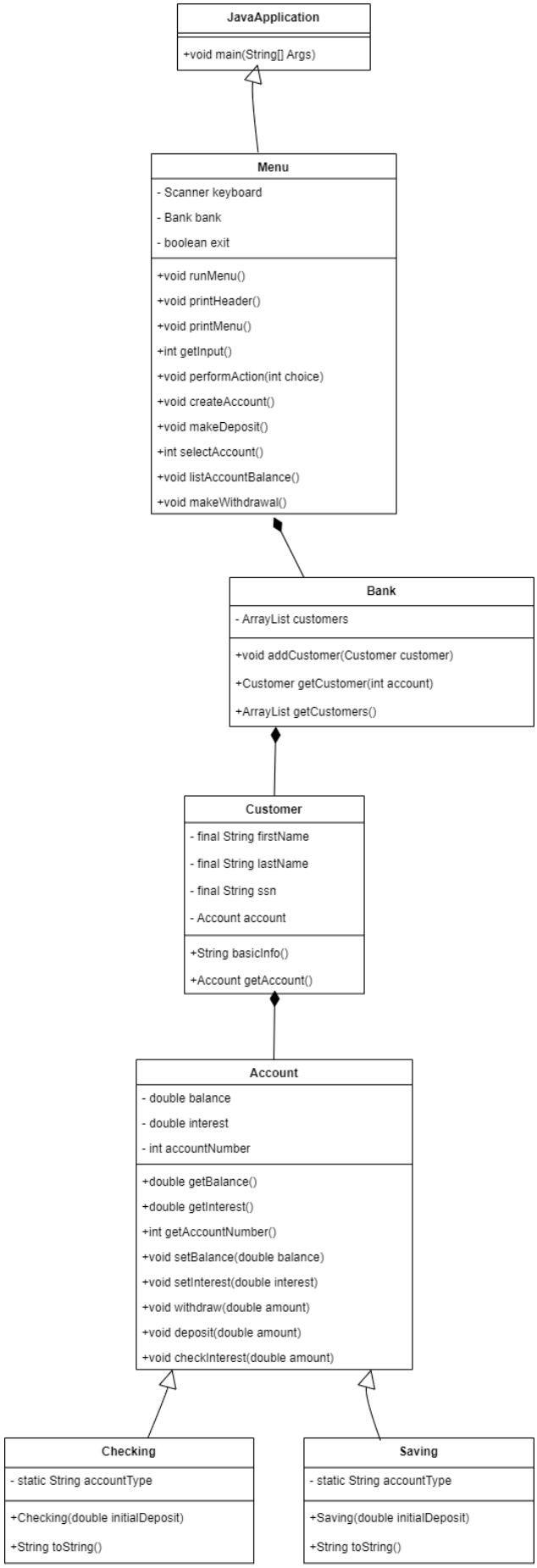
     Since I use an arraylist to store information I need to import java.util. Arraylist here is used to store account information.

Solution Design:

Class Diagram with GUI:

This class diagram represents the class diagram within my code which has several things that have composition and several normal associations.

Class Diagram with no GUI:


JavaApplication
+void main(String[] Args)
Menu
- Scanner keyboard
- Bank bank
- boolean exit
+void runMenu()
+void printHeader()
+void printMenu()
+int getInput()
+void performAction(int choice)
+void createAccount()
+void makeDeposit()
+int selectAccount()
+void listAccountBalance()
+void makeWithdrawal()
Bank
- ArrayList customers
+void addCustomer(Customer customer)
+Customer getCustomer(int account)
+ArrayList getCustomers()
Customer
- final String firstName
- final String lastName
- final String ssn
- Account account
+String basicInfo()
+Account getAccount()
Account
- double balance
- double interest
- int accountNumber
+double getBalance()
+double getInterest()
+int getAccountNumber()
+void setBalance(double balance)
+void setInterest(double interest)
+void withdraw(double amount)
+void deposit(double amount)
+void checkInterest(double amount)
Checking
- static String accountType
+Checking(double initialDeposit)
+String toString()
Saving
- static String accountType
+Saving(double initialDeposit)
+String toString()

This class diagram represents the class diagram within my code which has several things that have composition and several normal associations.

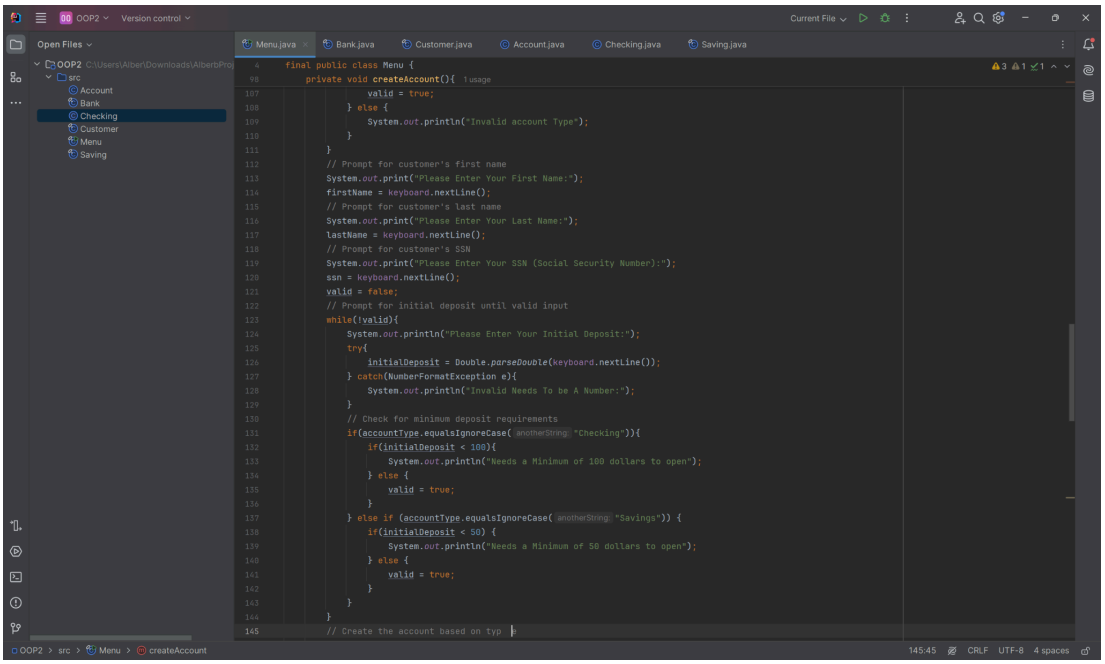
Explanation:

Algorithm:

Since both the GUI and non GUI is the same in the algorithm use I will Screenshot the algorithm use with the non GUI version

1. **Validation:**

   Validation algorithm is used to validate customer inputs and make it when an invalid input is used print statement.



2. **add:**

   This algorithm is used to add information such as customer information

```java
final public class Menu {
    private void createAccount(){ 1 usage
            } else if (accountType.equalsIgnoreCase( anotherString: "Savings")) {
                if(initialDeposit < 50) {
                    System.out.println("Needs a Minimum of 50 dollars to open");
                } else {
                    valid = true;
                }
            }
        }
        // Create the account based on type
        Account account;
        if(accountType.equalsIgnoreCase( anotherString: "Checking")){
            account = new Checking(initialDeposit);
        } else {
            account = new Saving(initialDeposit);
        }
        // Create a new customer with the provided information and add to bank
        Customer customer = new Customer(firstName, lastName, ssn, account);
        bank.addCustomer(customer);
    }

    // Method to handle deposit action
    private void makeDeposit() { 1 usage
        // Select an account to deposit into
        int account = selectAccount();
        if(account >= 0) {
            System.out.println("How much do you want to deposit?:");
            double amount = 0;
            try {
                // Get deposit amount from user input
                amount = Double.parseDouble(keyboard.nextLine());
            } catch (NumberFormatException e) {
                amount = 0;
            }
            // Perform deposit operation on the selected account
            bank.getCustomer(account).getAccount().deposit(amount);
        }
    }

    // Method to select an account from the list of customers
```

3. get:

This algorithm is used to get information such as customer information

```java
final public class Menu {
    private int selectAccount() { 3 usages
        account = -1;
    }
    return account;
}

// Method to list the account balance
private void listAccountBalance(){ 1 usage
    // Select an account to display balance
    int account = selectAccount();
    if(account >= 0) {
        // Print the account balance
        System.out.println(bank.getCustomer(account).getAccount());
    }
}

// Method to handle withdrawal action
private void makeWithdrawal(){ 1 usage
    // Select an account to withdraw from
    int account = selectAccount();
    if(account >= 0) {
        System.out.println("How much do you want to Withdraw?:");
        double amount = 0;
        try {
            // Get withdrawal amount from user input
            amount = Double.parseDouble(keyboard.nextLine());
        } catch (NumberFormatException e) {
            amount = 0;
        }
        // Perform withdrawal operation on the selected account
        bank.getCustomer(account).getAccount().withdraw(amount);
    }
}
}
```

4. List:

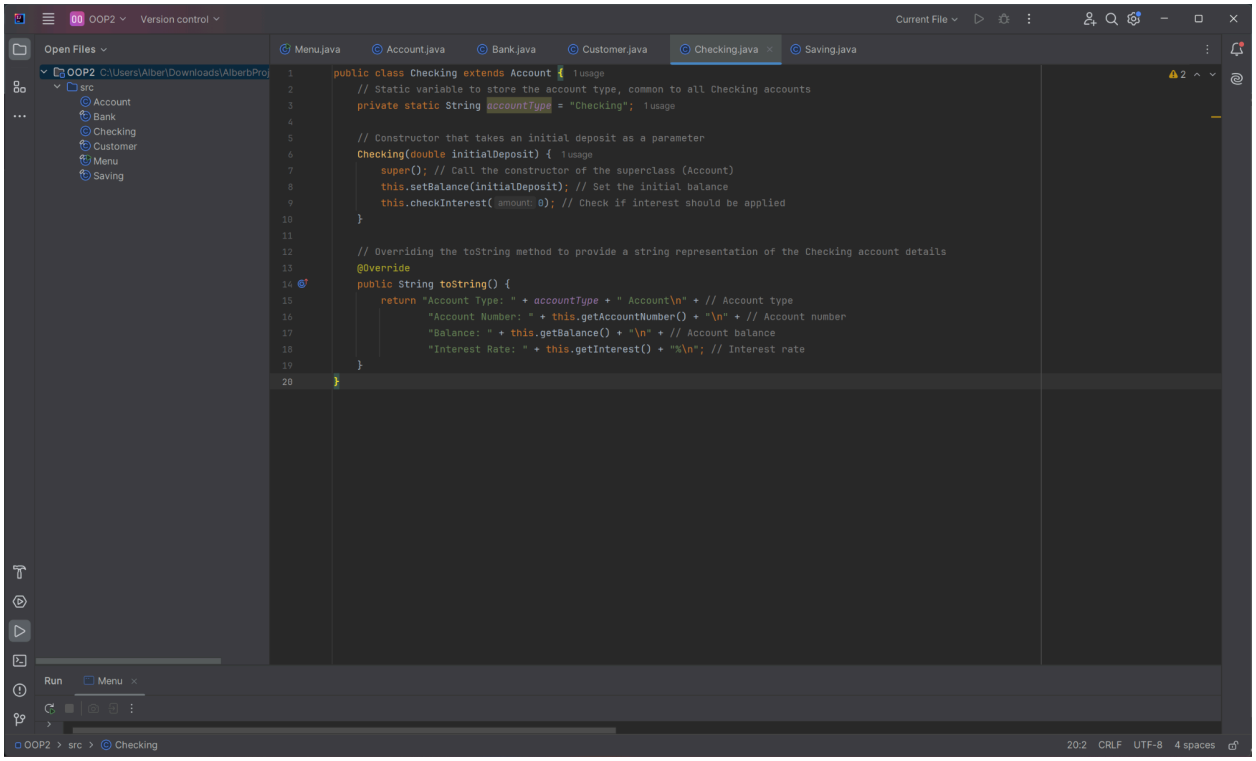
There is an algorithm used here to list down information of an account with balance (4th case after creating account), and without when after creating an account and wants to deposit or withdraw (list of accounts to choose).

```java
final public class Menu {
    private int selectAccount() {
            // Get the selected account index from user input
            account = Integer.parseInt(keyboard.nextLine()) - 1;
        } catch (NumberFormatException e){
            account = -1;
        }
        // Validate selected account index
        if(account < 0 || account >= customers.size()){
            System.out.println("Invalid Account");
            account = -1;
        }
        return account;
    }

    // Method to list the account balance
    private void listAccountBalance(){
        // Select an account to display balance
        int account = selectAccount();
        if(account >= 0) {
            // Print the account balance
            System.out.println(bank.getCustomer(account).getAccount());
        }
    }

    // Method to handle withdrawal action
    private void makeWithdrawal(){
        // Select an account to withdraw from
        int account = selectAccount();
        if(account >= 0) {
            System.out.println("How much do you want to Withdraw?:");
            double amount = 0;
            try {
                // Get withdrawal amount from user input
                amount = Double.parseDouble(keyboard.nextLine());
            } catch (NumberFormatException e) {
                amount = 0;
            }
            // Perform withdrawal operation on the selected account
            bank.getCustomer(account).getAccount().withdraw(amount);
        }
```

```java
final public class Menu {
    private void makeDeposit() {
            // Perform deposit operation on the selected account
            bank.getCustomer(account).getAccount().deposit(amount);
        }
    }

    // Method to select an account from the list of customers
    private int selectAccount() {
        ArrayList<Customer> customers = bank.getCustomers();
        if(customers.size() <= 0) {
            // No accounts available
            System.out.println("No Account:");
            return -1;
        }
        // List all available accounts
        System.out.println("Select an Account:");
        for(int i = 0; i < customers.size(); i++){
            System.out.println((i + 1) + ")" + customers.get(i).basicInfo());
        }
        int account = 0;
        System.out.println("Enter your selection:");
        try {
            // Get the selected account index from user input
            account = Integer.parseInt(keyboard.nextLine()) - 1;
        } catch (NumberFormatException e){
            account = -1;
        }
        // Validate selected account index
        if(account < 0 || account >= customers.size()){
            System.out.println("Invalid Account");
            account = -1;
        }
        return account;
    }

    // Method to list the account balance
    private void listAccountBalance(){
        // Select an account to display balance
        int account = selectAccount();
        if(account >= 0) {
```

5. **Override:**

This algorithm is used several times to override the String toString methods. So it is used to modify the data from the superclass. It is used in the checking and saving to return information account detail when specified. It is also used in the customer file where it is used to overwrite the details on the customer information data.



Checking

```java
final public class Saving extends Account {
    // Static variable to store the type of the account
    private static String accountType = "Saving";

    // Constructor for the Saving class, takes an initial deposit as a parameter
    Saving(double initialDeposit) {
        super(); // Call the constructor of the superclass (Account)
        this.setBalance(initialDeposit); // Set the initial balance for the saving account
        this.checkInterest( amount: 0); // Check interest eligibility with an initial balance of 0
    }

    // Override the toString method to provide a custom string representation of the Saving account
    @Override
    public String toString() {
        return "Account Type:" + accountType + "Account\n" +
                "Account Number:" + this.getAccountNumber() + "\n" +
                "Balance" + this.getBalance() + "\n" +
                "Interest Rate:" + this.getInterest() + "%\n";
    }
}
```

Saving

```java
final public class Customer {
    Customer(String firstName, String lastName, String ssn, Account account) {
        this.firstName = firstName; // Initialize the first name
        this.lastName = lastName; // Initialize the last name
        this.ssn = ssn; // Initialize the social security number
        this.account = account; // Initialize the account
    }

    // Override the toString method to provide a custom string representation of the Customer
    @Override
    public String toString() {
        return "\nCustomer Information\n" + // Header for customer information
                "First Name: " + firstName + "\n" + // Provide first name
                "Last Name: " + lastName + "\n" + // Provide last name
                "SSN: " + ssn + "\n" + // Provide social security number
                account; // Provide account information by calling its toString method
    }

    // Method to return basic customer information
    public String basicInfo() {
        return "First Name: " + firstName + // Provide first name
                " Last Name: " + lastName + // Provide last name
                " SSN: " + ssn + // Provide social security number
                " Account Number: " + account.getAccountNumber(); // Provide account number
    }

    // Getter method for the account
    Account getAccount() {
        return account; // Return the account object
    }
}
```

Customer

Solution Scheme:

1. **Account:**
   This class contains information of what to do when a customer wants to withdraw and deposit. It is mainly used to print that sort of information.
2. **Customer:**
   This class contains information on the customer from its constructors to the main code to display basic customer information (without balance).
3. **Menu:**
   This class holds information on the first time when the code is run. It contains the code for the display of options with how when a number from 1 to 4 is pressed it will continue to perform the action and what will happen when the number is outside that range and takes it as invalid.
4. **Bank:**
   This class is used to store information on the customer, alongside the methods on the retrieval of the customer data.
5. **Checking:**
   This class is an extension of the class account in which this is used to store the information when the customer creates a checking account.

6. **Saving:**
   This class is an extension of the class account in which this is used to store the information when the customer creates a saving account.

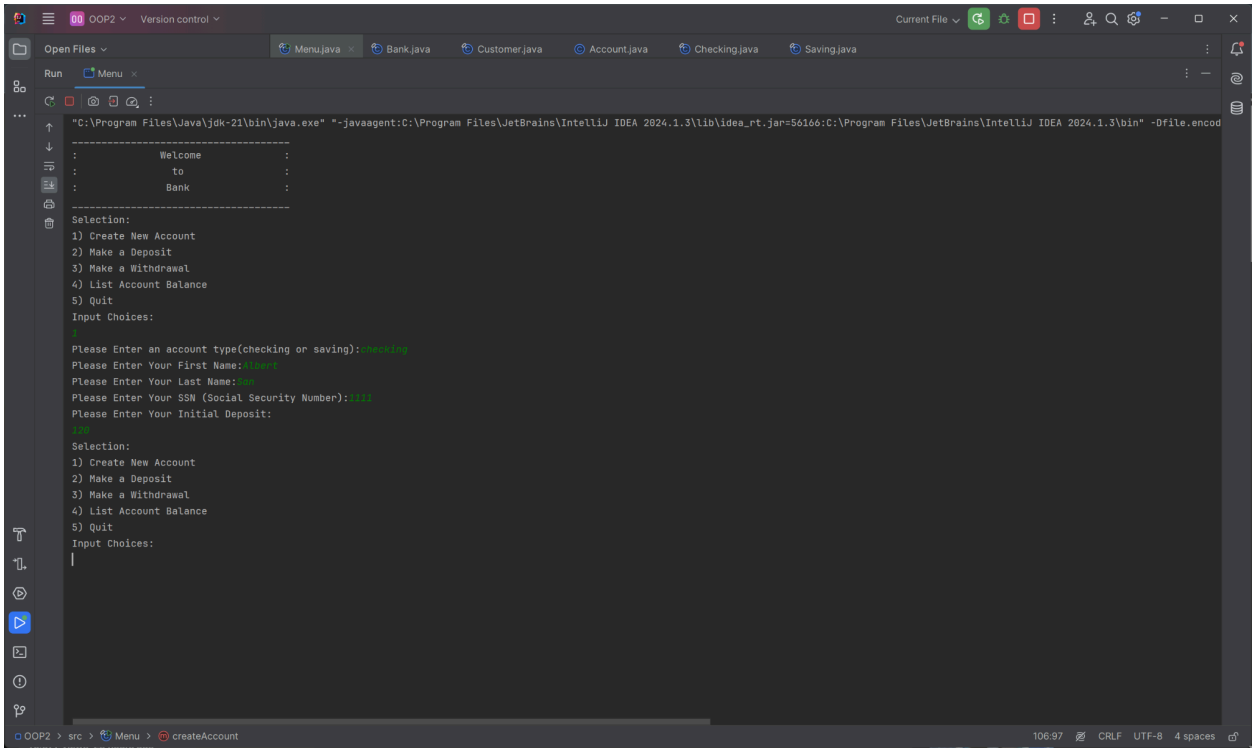
7. **Menus(GUI):**
   This class contains the code to run the GUI that uses the java swing

Data Structure used:

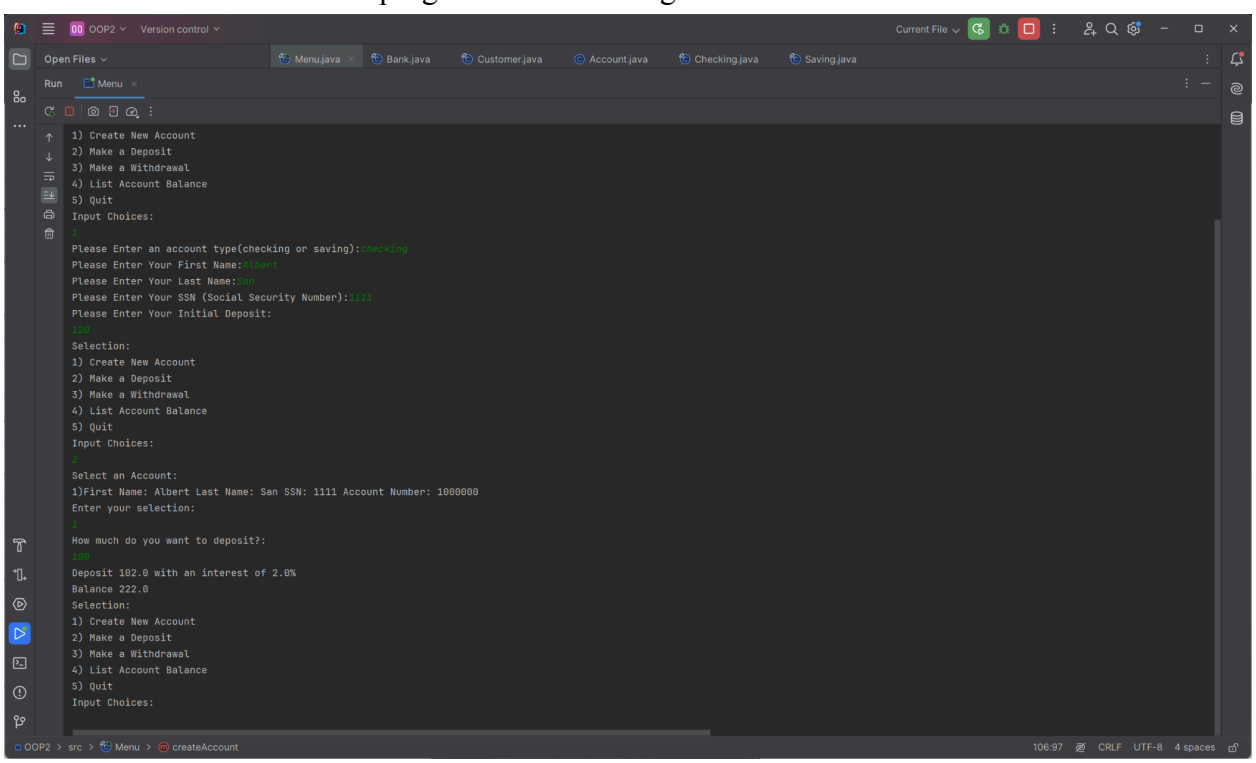
Arraylist: The system uses arraylist to store data on accounts(the customer) since you can list the account and see if your account is in it.

Evidence of working program:

Without GUI:



This is the Screenshot of the program when creating an account



This is the Screenshot when making a deposit

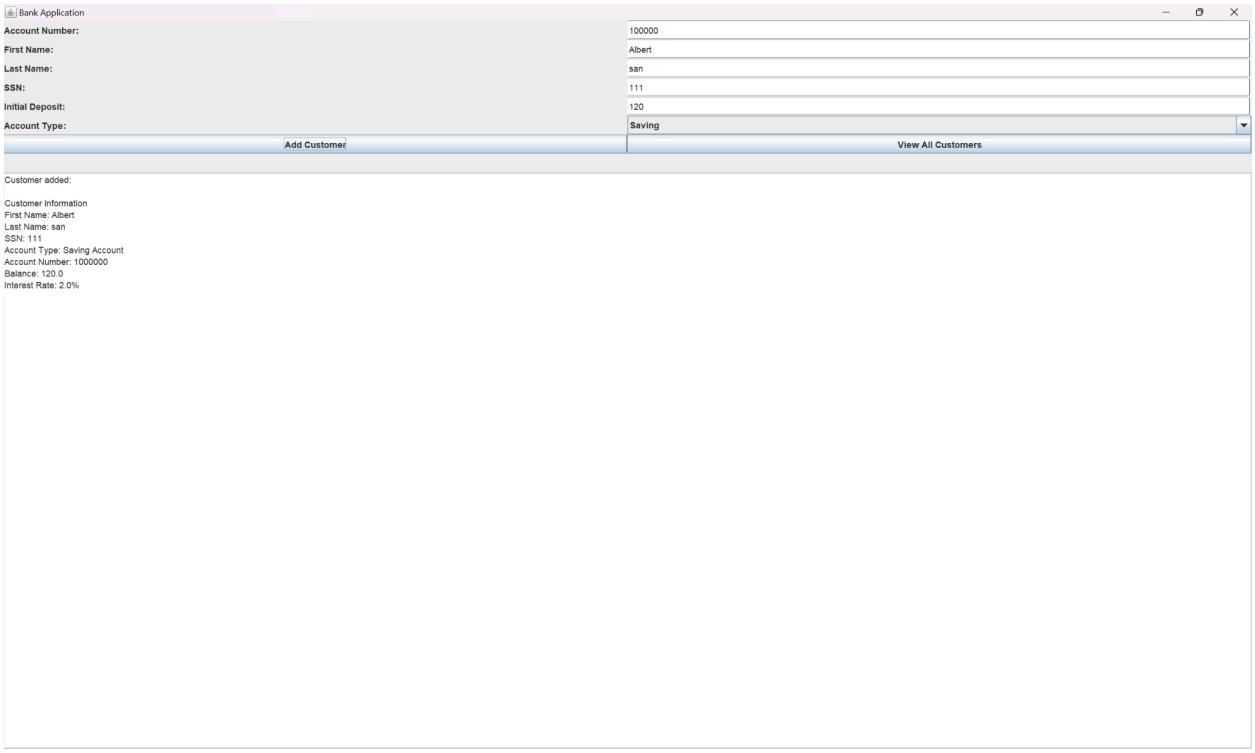
```
3) Make a Withdrawal
4) List Account Balance
5) Quit
Input Choices:
2
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
1
How much do you want to deposit?:
100
Deposit 102.0 with an interest of 2.0%
Balance 222.0
Selection:
1) Create New Account
2) Make a Deposit
3) Make a Withdrawal
4) List Account Balance
5) Quit
Input Choices:
3
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
1
How much do you want to Withdraw?:
50
Withdrawn 50.0 with a fee of 5 dollars
Balance 167.0
Selection:
1) Create New Account
2) Make a Deposit
3) Make a Withdrawal
4) List Account Balance
5) Quit
Input Choices:
```

This is the Screenshot when making a Withdrawal

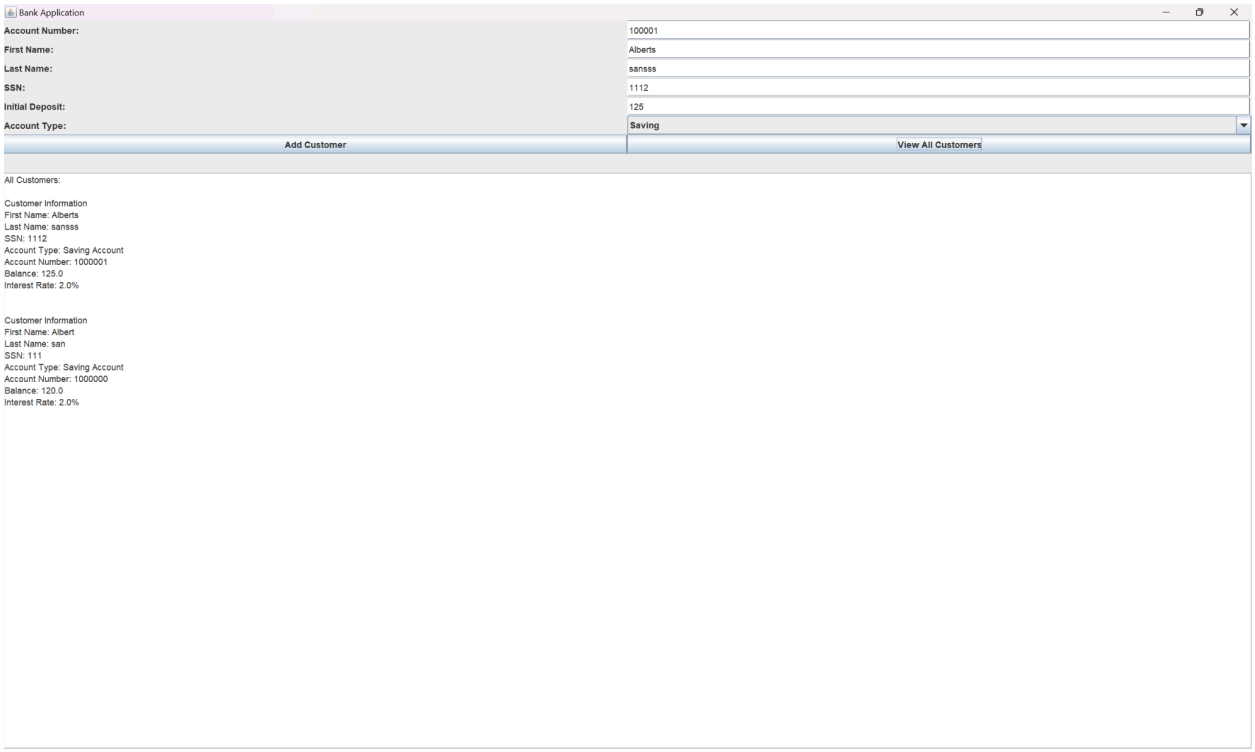
```
4) List Account Balance
5) Quit
Input Choices:
4
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
1
How much do you want to Withdraw?:
50
Withdrawn 50.0 with a fee of 5 dollars
Balance 167.0
Selection:
1) Create New Account
2) Make a Deposit
3) Make a Withdrawal
4) List Account Balance
5) Quit
Input Choices:
4
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
1
Account Type:CheckingAccount
Account Number:1000000
Balance167.0
Interest Rate:2.0%

Selection:
1) Create New Account
2) Make a Deposit
3) Make a Withdrawal
4) List Account Balance
5) Quit
Input Choices:
```

This is the Screenshot of the program when listing account with balance

WITH GUI:



This is the GUI when adding customer accounts



This is the GUI when viewing all customer accounts and it can see which type the customer chooses to create.

Resources:

- Github link: https://github.com/Hoyoddiction/OOP-Final