



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:

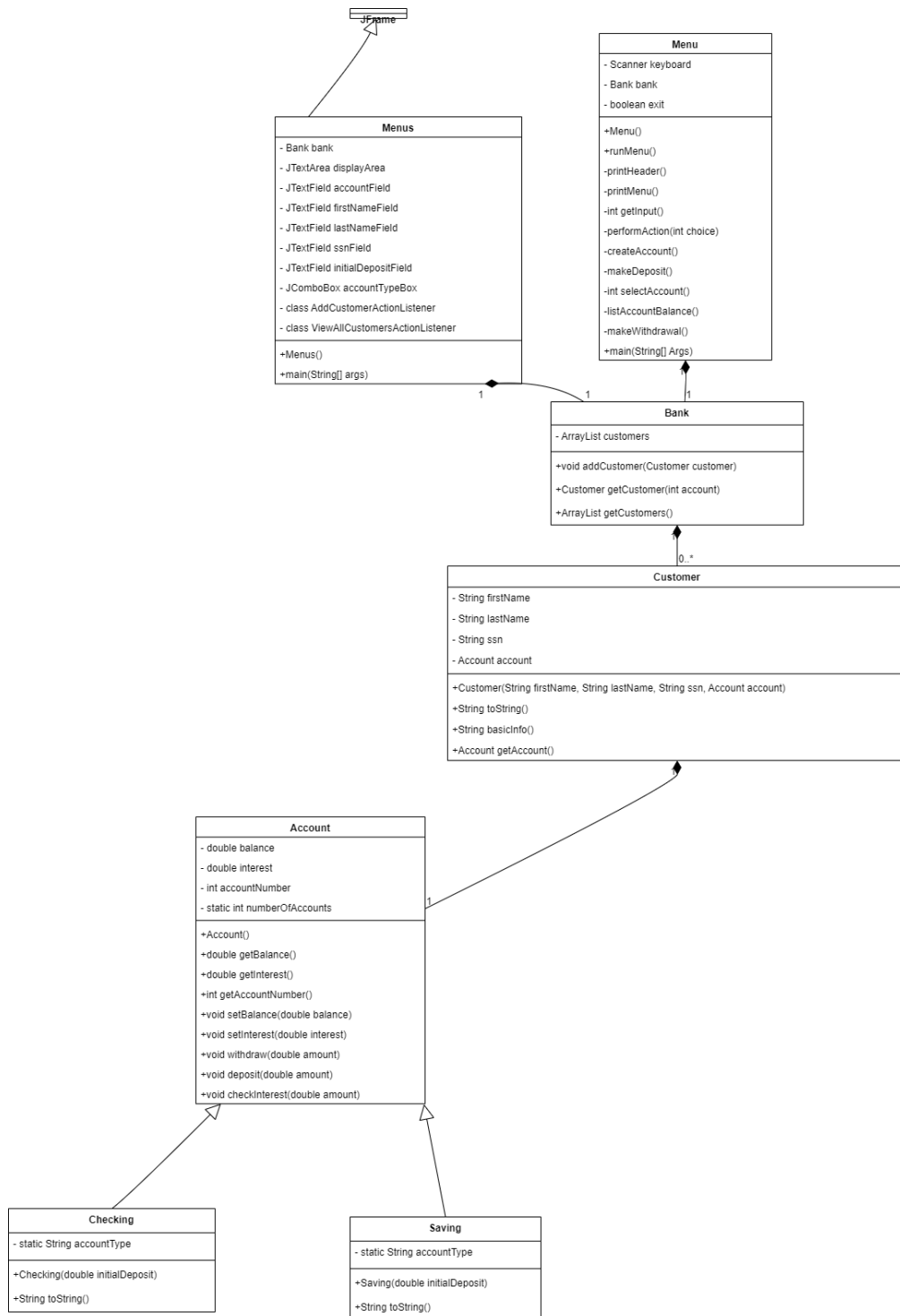
Since there is a version of my code that uses a GUI, I decided to use java swing since that is something that I have done before and it looks clean and simple to look at.

- Java Util:

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

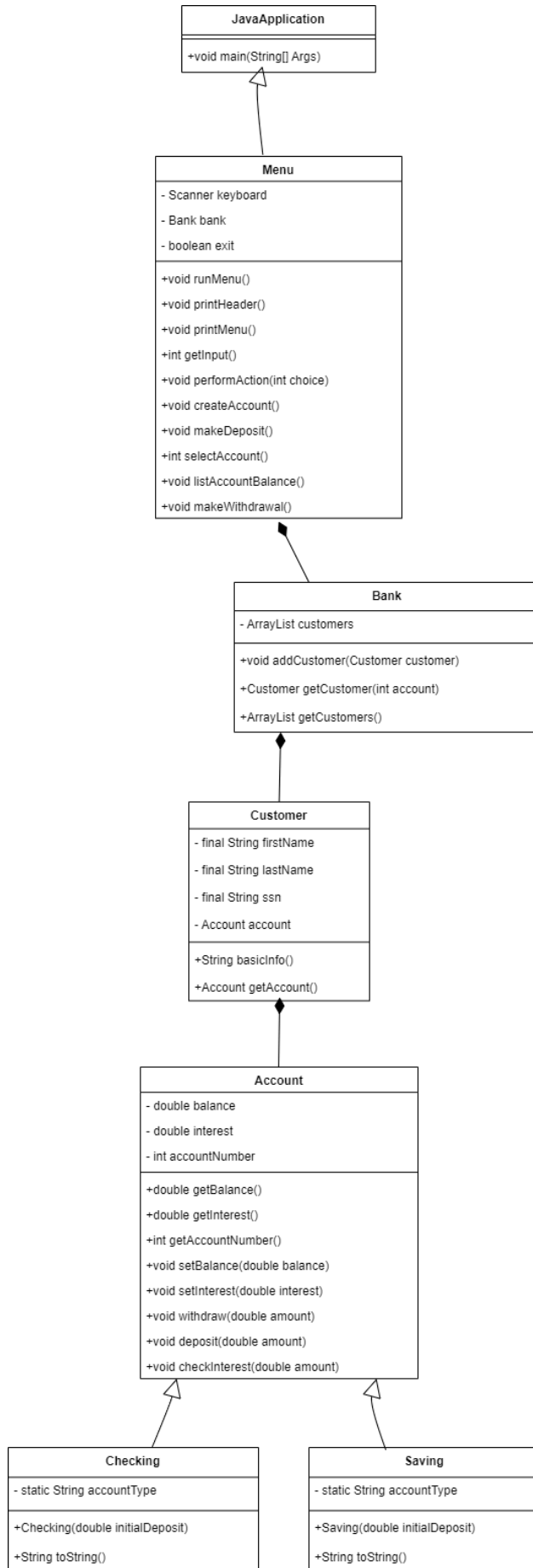
Solution Design:

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:



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

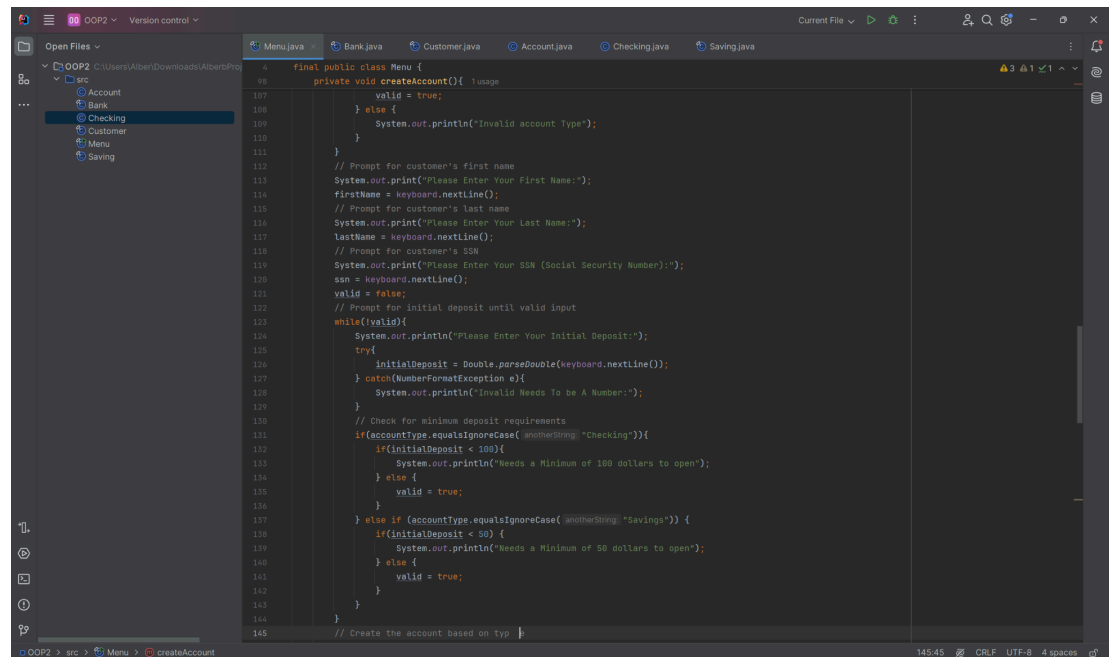
Explanation:

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.



```
117 final public class Menu {
118     private void createAccount(){ // usage
119         valid = true;
120         } else {
121             System.out.println("Invalid account Type");
122         }
123     }
124     // Prompt for customer's first name
125     System.out.print("Please Enter Your First Name:");
126     firstName = keyboard.nextLine();
127     // Prompt for customer's last name
128     System.out.print("Please Enter Your Last Name:");
129     lastName = keyboard.nextLine();
130     // Prompt for customer's SSN
131     System.out.print("Please Enter Your SSN (Social Security Number):");
132     ssn = keyboard.nextLine();
133     valid = false;
134     // Prompt for initial deposit until valid input
135     while(!valid){
136         System.out.println("Please Enter Your Initial Deposit:");
137         try{
138             initialDeposit = Double.parseDouble(keyboard.nextLine());
139         } catch(NumberFormatException e){
140             System.out.println("Invalid Needs To Be A Number:");
141         }
142     }
143     // Check for minimum deposit requirements
144     if(accountType.equalsIgnoreCase("Checking")){
145         if(initialDeposit < 100){
146             System.out.println("Needs a Minimum of 100 dollars to open");
147         } else {
148             valid = true;
149         }
150     } else if (accountType.equalsIgnoreCase("Savings")) {
151         if(initialDeposit < 50) {
152             System.out.println("Needs a Minimum of 50 dollars to open");
153         } else {
154             valid = true;
155         }
156     }
157 }
158 // Create the account based on typ
```

2. add:

This algorithm is used to add information such as customer information

```
final public class Menu {
    private void createAccount() { 1 usage
        // Create the account based on type
        Account account;
        if(accountType.equalsIgnoreCase("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

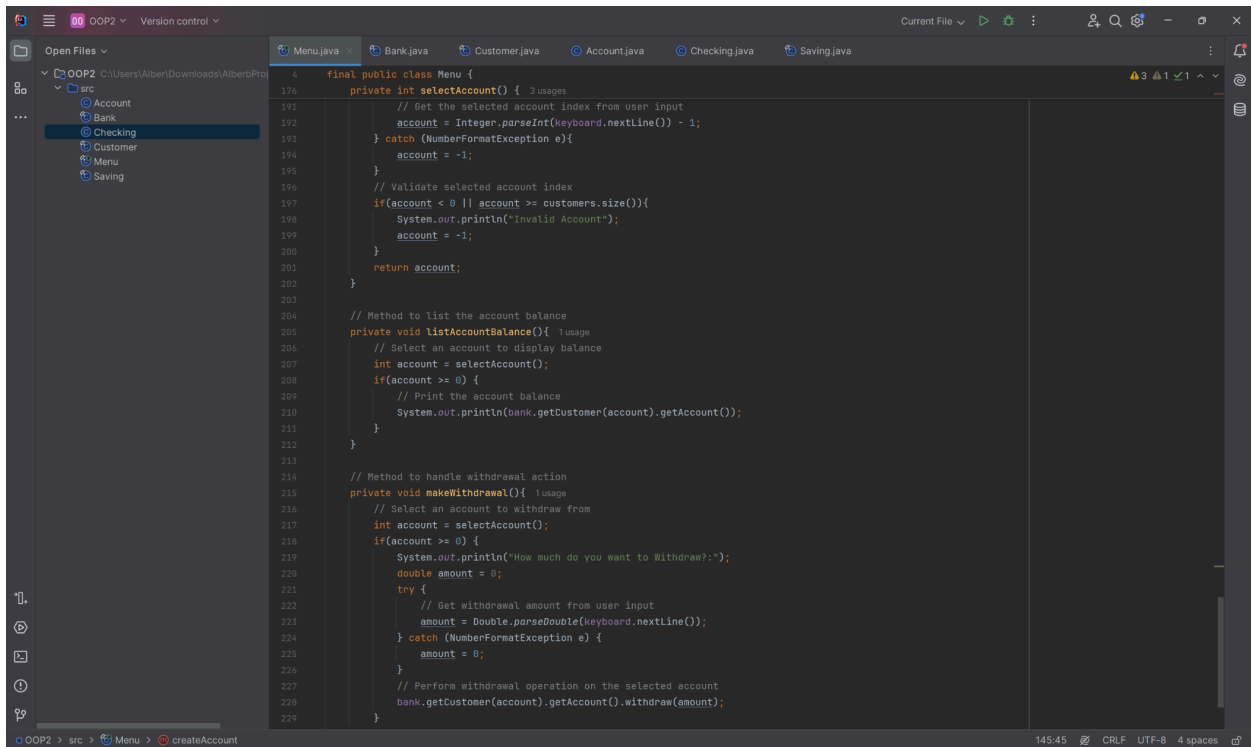
```
final public class Menu {
    private int selectAccount() { 3 usages
        account = -1;
        return account;
    }

    // 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());
        }
    }

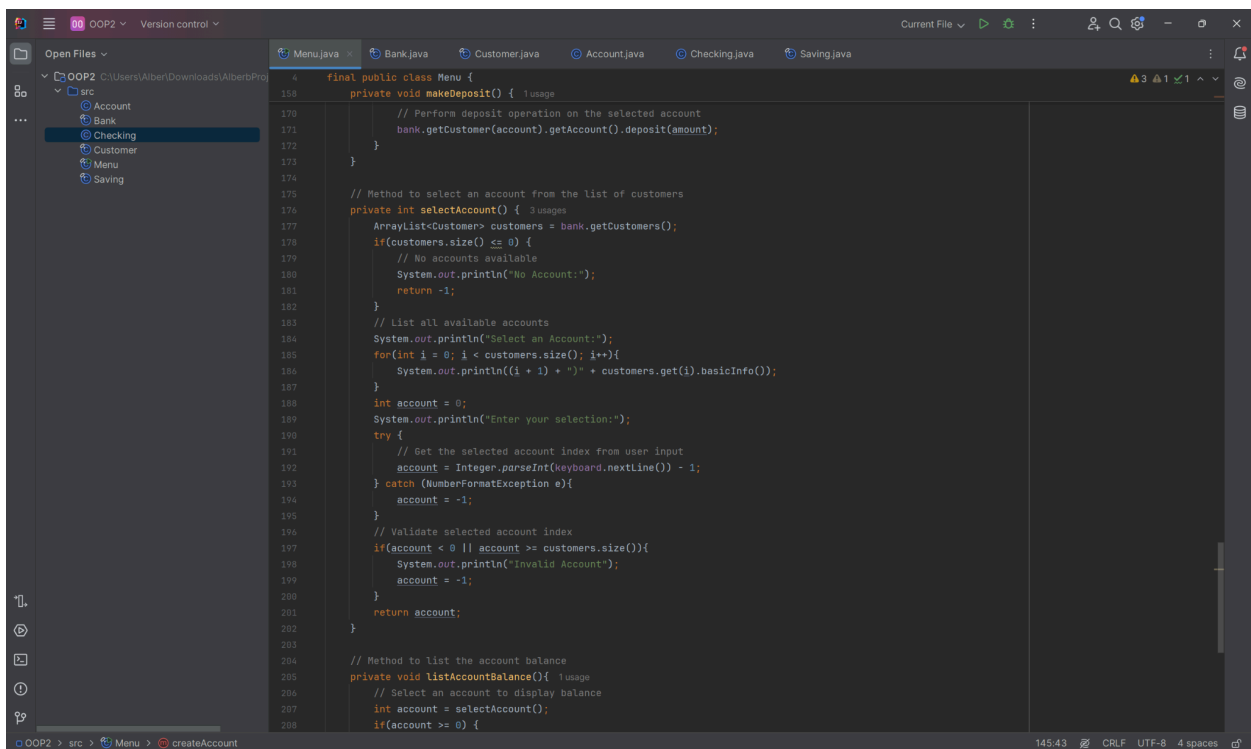
    // Method to handle withdrawal action
    private void makeWithdrawal() { 1 usage
        // Select an account to withdraw from
        int account = selectAccount();
        if(account >= 0) {
            System.out.println("How much do you want to Withdraw?:");
            double amount = 0;
            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).



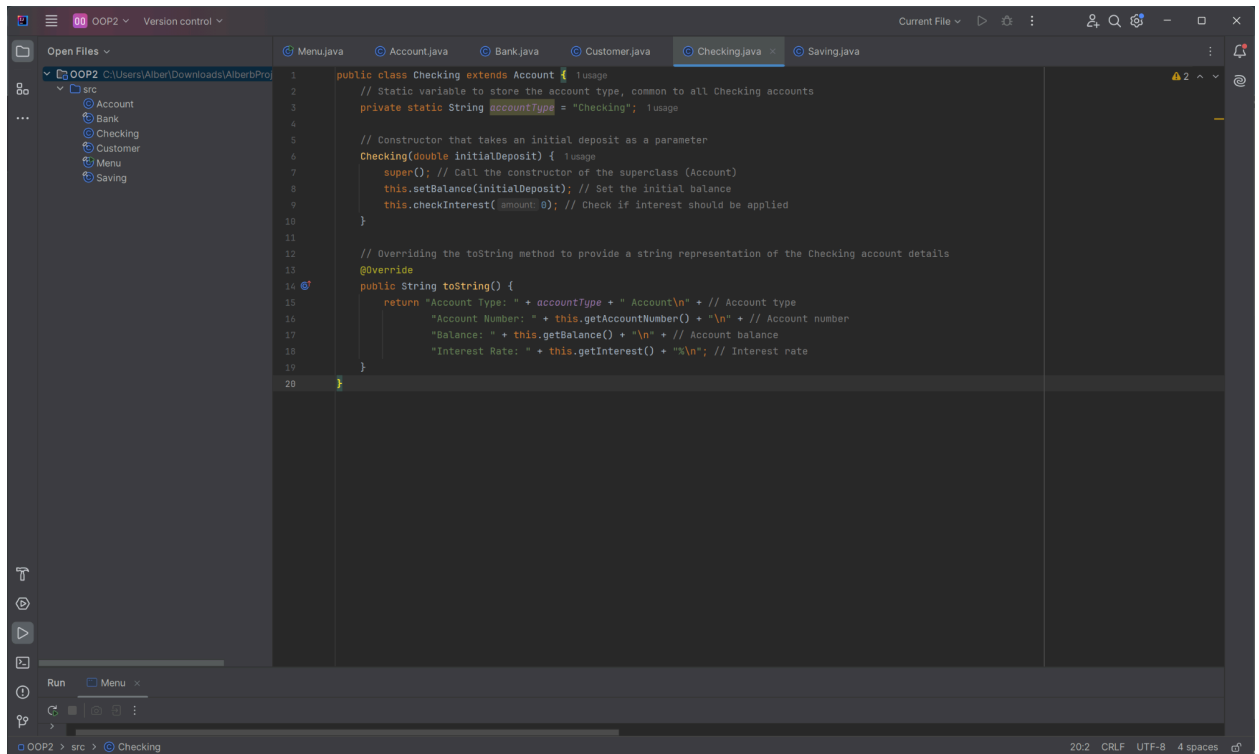
```
4 final public class Menu {
176 private int selectAccount() { 3 usages
191 // Get the selected account index from user input
192 account = Integer.parseInt(keyboard.nextLine()) - 1;
193 } catch (NumberFormatException e){
194 account = -1;
195 }
196 // Validate selected account index
197 if(account < 0 || account >= customers.size()){
198 System.out.println("Invalid Account");
199 account = -1;
200 }
201 return account;
202 }
203
204 // Method to list the account balance
205 private void listAccountBalance(){ 1 usage
206 // Select an account to display balance
207 int account = selectAccount();
208 if(account >= 0) {
209 // Print the account balance
210 System.out.println(bank.getCustomer(account).getAccount());
211 }
212 }
213
214 // Method to handle withdrawal action
215 private void makeWithdrawal(){ 1 usage
216 // Select an account to withdraw from
217 int account = selectAccount();
218 if(account >= 0) {
219 System.out.println("How much do you want to Withdraw?:");
220 double amount = 0;
221 try {
222 // Get withdrawal amount from user input
223 amount = Double.parseDouble(keyboard.nextLine());
224 } catch (NumberFormatException e) {
225 amount = 0;
226 }
227 // Perform withdrawal operation on the selected account
228 bank.getCustomer(account).getAccount().withdraw(amount);
229 }
```



```
4 final public class Menu {
158 private void makeDeposit() { 1 usage
170 // Perform deposit operation on the selected account
171 bank.getCustomer(account).getAccount().deposit(amount);
172 }
173 }
174
175 // Method to select an account from the list of customers
176 private int selectAccount() { 3 usages
177 ArrayList<Customer> customers = bank.getCustomers();
178 if(customers.size() <= 0) {
179 // No accounts available
180 System.out.println("No Account:");
181 return -1;
182 }
183 // List all available accounts
184 System.out.println("Select an Account:");
185 for(int i = 0; i < customers.size(); i++){
186 System.out.println((i + 1) + " " + customers.get(i).basicInfo());
187 }
188 int account = 0;
189 System.out.println("Enter your selection:");
190 try {
191 // Get the selected account index from user input
192 account = Integer.parseInt(keyboard.nextLine()) - 1;
193 } catch (NumberFormatException e){
194 account = -1;
195 }
196 // Validate selected account index
197 if(account < 0 || account >= customers.size()){
198 System.out.println("Invalid Account");
199 account = -1;
200 }
201 return account;
202 }
203
204 // Method to list the account balance
205 private void listAccountBalance(){ 1 usage
206 // Select an account to display balance
207 int account = selectAccount();
208 if(account >= 0) {
```

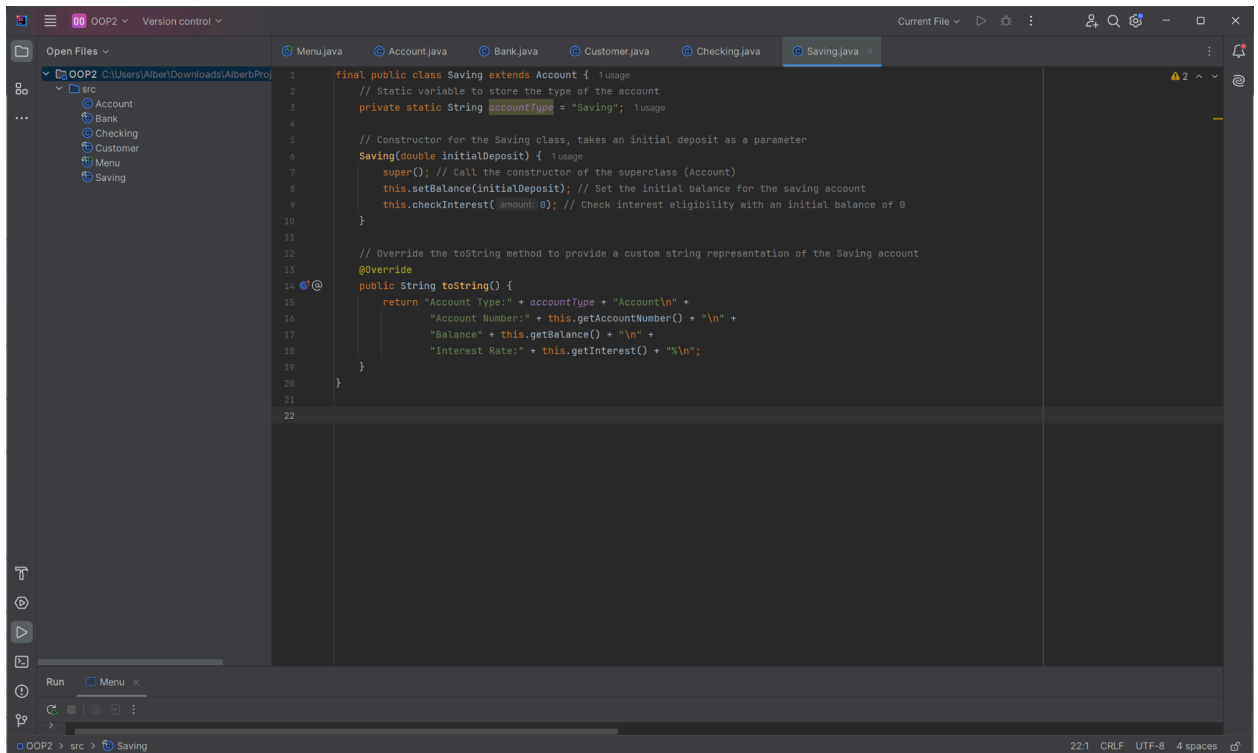
5. Override:

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.



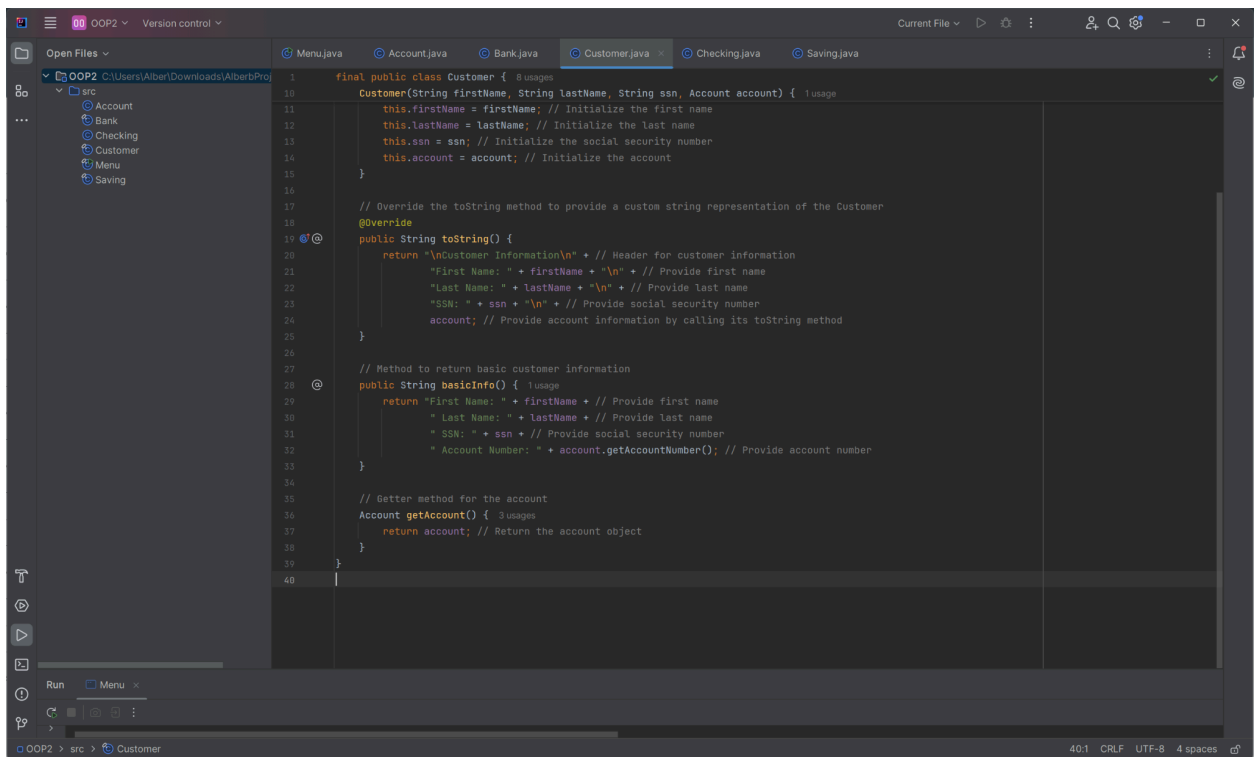
```
1 public class Checking extends Account { // Usage
2     // Static variable to store the account type, common to all Checking accounts
3     private static String accountType = "Checking"; // Usage
4
5     // Constructor that takes an initial deposit as a parameter
6     Checking(double initialDeposit) { // Usage
7         super(); // Call the constructor of the superclass (Account)
8         this.setBalance(initialDeposit); // Set the initial balance
9         this.checkInterest(0); // Check if interest should be applied
10    }
11
12    // Overriding the toString method to provide a string representation of the Checking account details
13    @Override
14    public String toString() {
15        return "Account Type: " + accountType + " Account\n" + // Account type
16            "Account Number: " + this.getAccountNumber() + "\n" + // Account number
17            "Balance: " + this.getBalance() + "\n" + // Account balance
18            "Interest Rate: " + this.getInterest() + "%\n"; // Interest rate
19    }
20 }
```

Checking



```
1 final public class Saving extends Account { 1 usage
2 // Static variable to store the type of the account
3 private static String accountType = "Saving"; 1 usage
4
5 // Constructor for the Saving class, takes an initial deposit as a parameter
6 Saving(double initialDeposit) { 1 usage
7     super(); // Call the constructor of the superclass (Account)
8     this.setBalance(initialDeposit); // Set the initial balance for the saving account
9     this.checkInterest(0); // Check interest eligibility with an initial balance of 0
10 }
11
12 // Override the toString method to provide a custom string representation of the Saving account
13 @Override
14 @
15 public String toString() {
16     return "Account Type: " + accountType + "Account\n" +
17         "Account Number: " + this.getAccountNumber() + "\n" +
18         "Balance: " + this.getBalance() + "\n" +
19         "Interest Rate: " + this.getInterest() + "%\n";
20 }
21
22
```

Saving



```
1 final public class Customer { 8 usages
10 Customer(String firstName, String lastName, String ssn, Account account) { 1 usage
11     this.firstName = firstName; // Initialize the first name
12     this.lastName = lastName; // Initialize the last name
13     this.ssn = ssn; // Initialize the social security number
14     this.account = account; // Initialize the account
15 }
16
17 // Override the toString method to provide a custom string representation of the Customer
18 @Override
19 @
20 public String toString() {
21     return "\nCustomer Information\n" + // Header for customer information
22         "First Name: " + firstName + "\n" + // Provide first name
23         "Last Name: " + lastName + "\n" + // Provide last name
24         "SSN: " + ssn + "\n" + // Provide social security number
25         account; // Provide account information by calling its toString method
26 }
27
28 // Method to return basic customer information
29 @
30 public String basicInfo() { 1 usage
31     return "First Name: " + firstName + // Provide first name
32         "Last Name: " + lastName + // Provide last name
33         "SSN: " + ssn + // Provide social security number
34         "Account Number: " + account.getAccountNumber(); // Provide account number
35 }
36
37 // Getter method for the account
38 Account getAccount() { 3 usages
39     return account; // Return the account object
40 }
41
```

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:

The screenshot shows a Java IDE with a project named 'OOP2'. The 'Run' console displays the output of a Java program. The program starts with a welcome message and a menu of options: 1) Create New Account, 2) Make a Deposit, 3) Make a Withdrawal, 4) List Account Balance, and 5) Quit. The user has selected option 1, and the program prompts for account details: 'Please Enter an account type(checking or saving):', 'Please Enter Your First Name:', 'Please Enter Your Last Name:', 'Please Enter Your SSN (Social Security Number):', and 'Please Enter Your Initial Deposit:'. The user has entered 'checking', 'Albert', 'San', '1111', and '100' respectively. The program then displays the same menu again.

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.1.3\lib\idea_rt.jar=56166:C:\Program Files\JetBrains\IntelliJ IDEA 2024.1.3\bin" -Dfile.encoding=UTF-8

-----
Welcome
to
Bank
-----

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

Please Enter an account type(checking or saving): checking
Please Enter Your First Name: Albert
Please Enter Your Last Name: San
Please Enter Your SSN (Social Security Number): 1111
Please Enter Your Initial Deposit: 100

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

This is the Screenshot of the program when creating an account

The screenshot shows the same Java IDE with the 'Run' console displaying the next steps of the program. The user has selected option 2, 'Make a Deposit'. The program prompts for the deposit amount: 'How much do you want to deposit?:'. The user has entered '100'. The program then displays the account details: '1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000'. The user has selected option 1, 'Create New Account'. The program then displays the same menu again.

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

Please Enter an account type(checking or saving): checking
Please Enter Your First Name: Albert
Please Enter Your Last Name: San
Please Enter Your SSN (Social Security Number): 1111
Please Enter Your Initial Deposit: 100

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

How much do you want to deposit?:
100

Deposit 100.0 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:
1

1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
1

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

This is the Screenshot when making a deposit

```
OOP2 - Version control
Current File
Menu.java Bank.java Customer.java Account.java Checking.java Saving.java

Run Menu
3) Make a Withdrawal
4) List Account Balance
5) Quit
Input Choices:
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
How much do you want to deposit?:
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:
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
How much do you want to Withdraw?:
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

```
OOP2 - Version control
Current File
Menu.java Bank.java Customer.java Account.java Checking.java Saving.java

Run Menu
4) List Account Balance
5) Quit
Input Choices:
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
How much do you want to Withdraw?:
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:
Select an Account:
1)First Name: Albert Last Name: San SSN: 1111 Account Number: 1000000
Enter your selection:
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:

Bank Application

Account Number: 100000

First Name: Albert

Last Name: san

SSN: 111

Initial Deposit: 120

Account Type: Saving

Add Customer View All Customers

Customer added:

Customer Information

First Name: Albert

Last Name: san

SSN: 111

Account Type: Saving Account

Account Number: 1000000

Balance: 120.0

Interest Rate: 2.0%

This is the GUI when adding customer accounts

Bank Application

Account Number: 100001

First Name: Alberts

Last Name: sansss

SSN: 1112

Initial Deposit: 125

Account Type: Saving

Add Customer View All Customers

All Customers:

Customer Information

First Name: Alberts

Last Name: sansss

SSN: 1112

Account Type: Saving Account

Account Number: 1000001

Balance: 125.0

Interest Rate: 2.0%

Customer Information

First Name: Albert

Last Name: san

SSN: 111

Account Type: Saving Account

Account Number: 1000000

Balance: 120.0

Interest Rate: 2.0%

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>