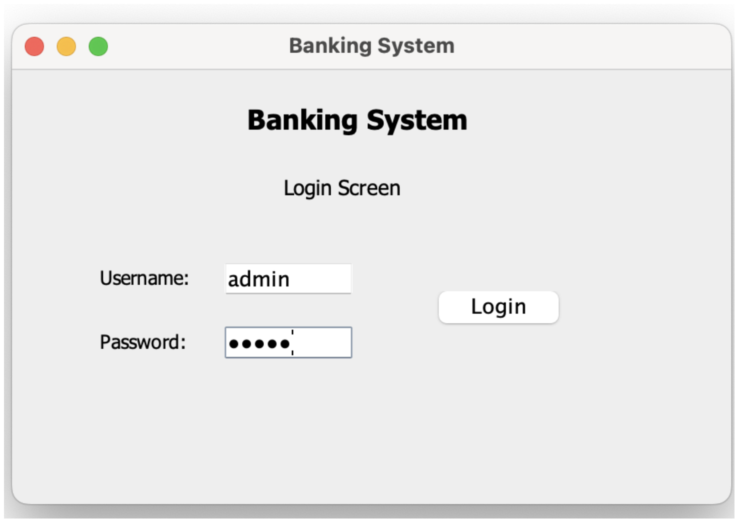
Responsibilities GitHub repository Link: gh repo clone ArturDisha/Ga-Programming

1. Class Evaluation

For my part in the project, I'll focus on three main tasks: Class Evaluation Testing, Integration Testing, and documenting the results.
I'll start by selecting one of the methods from the account management module likely something like getAccountDetails. Once I've picked the method, my job is to group its inputs into logical classes (e.g., Savings, Current, or Fixed Deposit account types). After that, I'll verify if the method handles each class properly and gives consistent, accurate outputs.

If needed, I'll use a decision table instead. This will map out all the possible input conditions and expected results for the method. The goal is to make sure the method produces the right output for every combination of inputs.

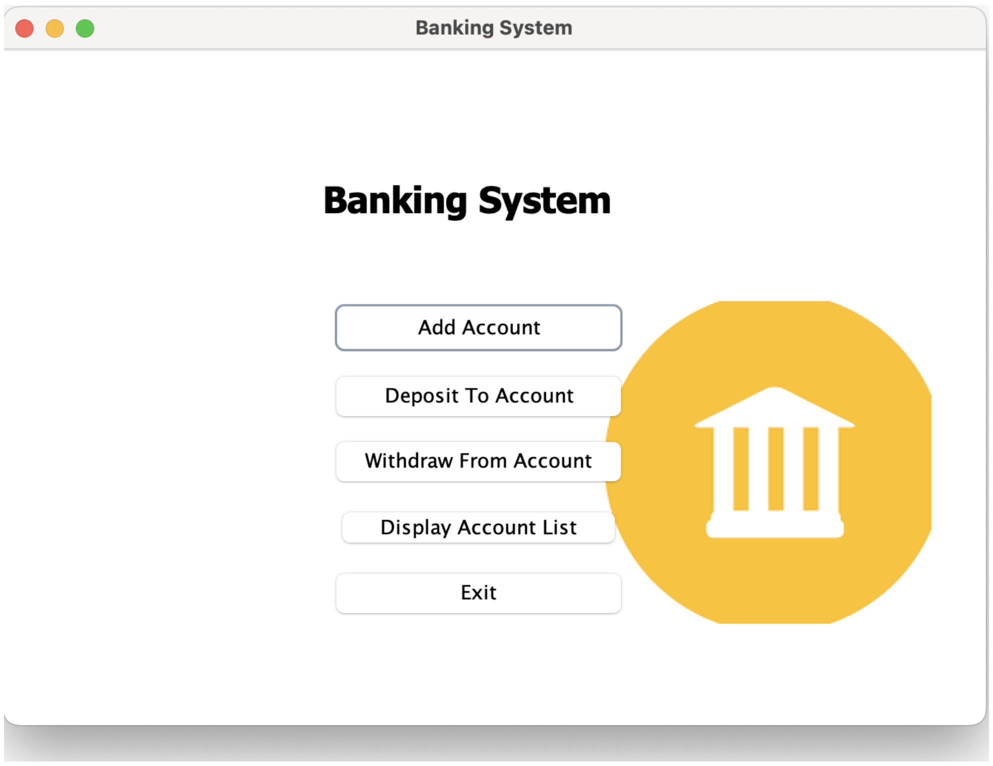


<u>1)Login Screen</u>

This is the entry point to the banking system, ensuring only authorized users can access its features. The screen requires the user to:

- **Enter their username:** This could be an admin username or another authorized user.
- **Enter their password:** For security, the password is masked (hidden as dots).
- **Click the Login button:** Validates the entered credentials against the system's database.

If the credentials are correct, the user gains access to the main menu. Otherwise, an error message will be shown. This screen is crucial for maintaining security and preventing unauthorized access to sensitive account data.

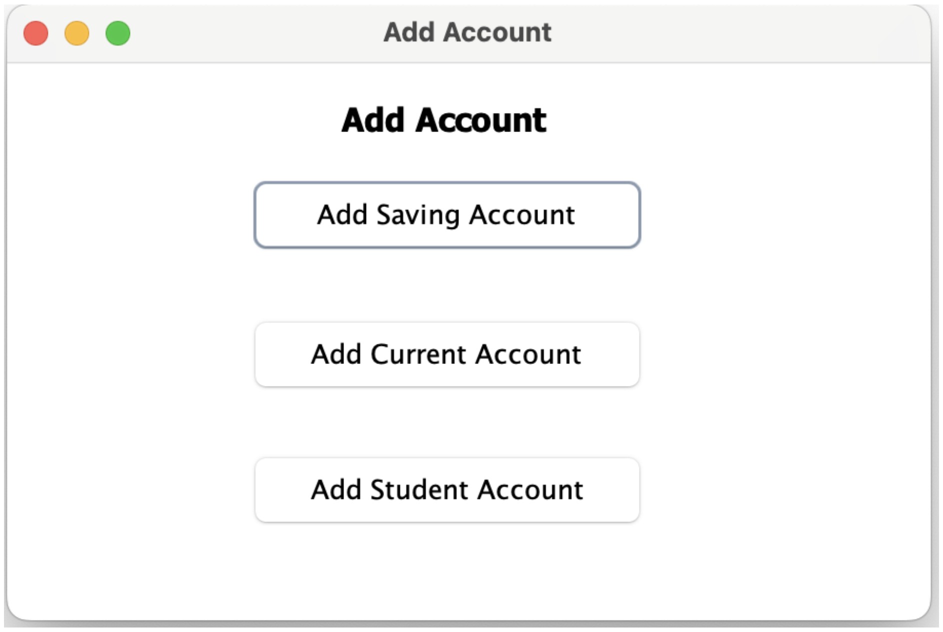


2)Main Menu - Banking System

This is the central hub of the application, offering quick access to all major functionalities:

- **Add Account:** Directs the user to the account creation interface.
- **Deposit To Account:** Opens the deposit screen where users can add funds to any account by entering the account number and amount.
- **Withdraw From Account:** Redirects to the withdrawal screen for processing withdrawals.
- **Display Account List:** Opens the account list view, displaying all accounts and their details.
- **Exit:** Safely closes the application.

The design of this menu ensures simplicity, making it easy for users to navigate between features without confusion.

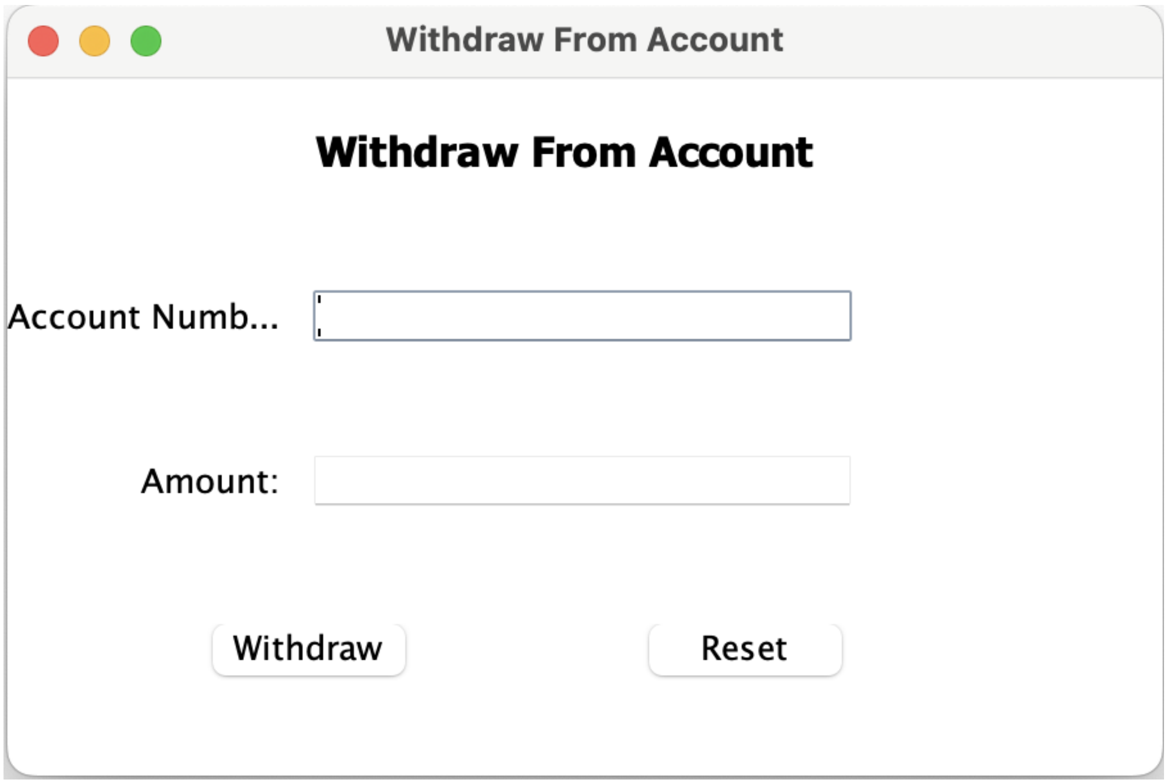


3)Add Account

This screen provides a simple interface for adding new accounts to the system. Users are offered three options to categorize the type of account they want to create:

- **Savings Account:** For regular customers who want to save money.
- **Current Account:** Often used by businesses for day-to-day transactions.
- **Student Account:** A specialized account for students with unique features like lower minimum balance requirements.

By selecting the appropriate button, the user is directed to another form to input details specific to the chosen account type.

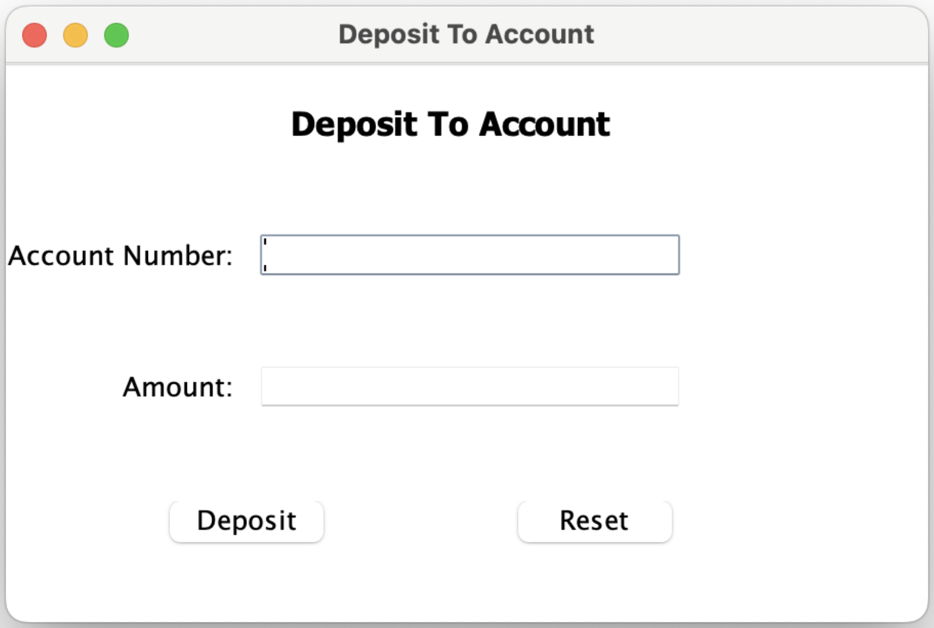


4)Withdraw From Account

This screen facilitates the process of withdrawing money from a specific account. The user can:

- **Input the account number:** This ensures the transaction is applied to the correct account.
- **Specify the withdrawal amount:** The amount to be deducted from the balance.
- **Withdraw button:** Processes the withdrawal request. If there are issues (e.g., insufficient balance), the system will alert the user.
- **Reset button:** Clears the fields, allowing the user to re-enter the details if there's an error.

This screen ensures that transactions are secure, targeted to the correct account, and well-documented.

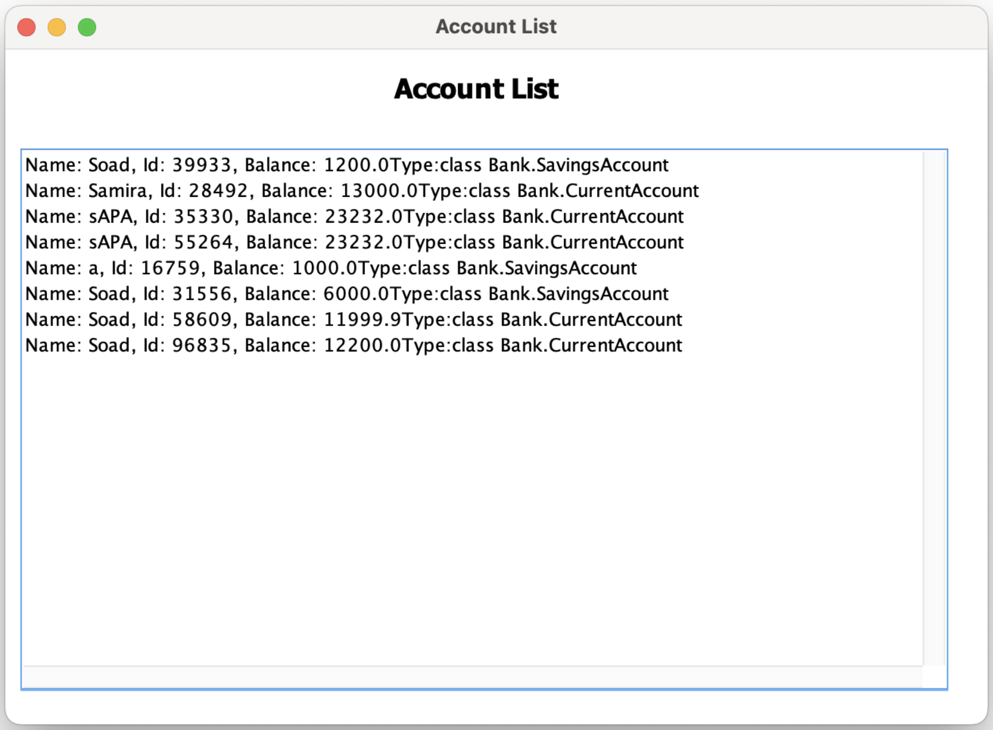


5)Deposit to Account Screen

This screen is designed to allow users to deposit money into a specific account within the system. Here's what each part does:

- **Account Number Field:**Users enter the unique account number to ensure the deposit is applied to the correct account.
- **Amount Field:**Here, users type in the amount of money they want to deposit into the account.
- **Deposit Button:**When clicked, the system processes the deposit, adding the specified amount to the account balance. If the account number or amount is invalid, the system will show an error message.
- **Reset Button:**This button clears both the account number and amount fields, letting the user start over if they've made a mistake.

This screen helps ensure accurate and secure deposits while keeping the process straightforward for users.



```
Name: Soad, Id: 39933, Balance: 1200.0Type:class Bank.SavingsAccount
Name: Samira, Id: 28492, Balance: 13000.0Type:class Bank.CurrentAccount
Name: sAPA, Id: 35330, Balance: 23232.0Type:class Bank.CurrentAccount
Name: sAPA, Id: 55264, Balance: 23232.0Type:class Bank.CurrentAccount
Name: a, Id: 16759, Balance: 1000.0Type:class Bank.SavingsAccount
Name: Soad, Id: 31556, Balance: 6000.0Type:class Bank.SavingsAccount
Name: Soad, Id: 58609, Balance: 11999.9Type:class Bank.CurrentAccount
Name: Soad, Id: 96835, Balance: 12200.0Type:class Bank.CurrentAccount
```

6)Account List Screen

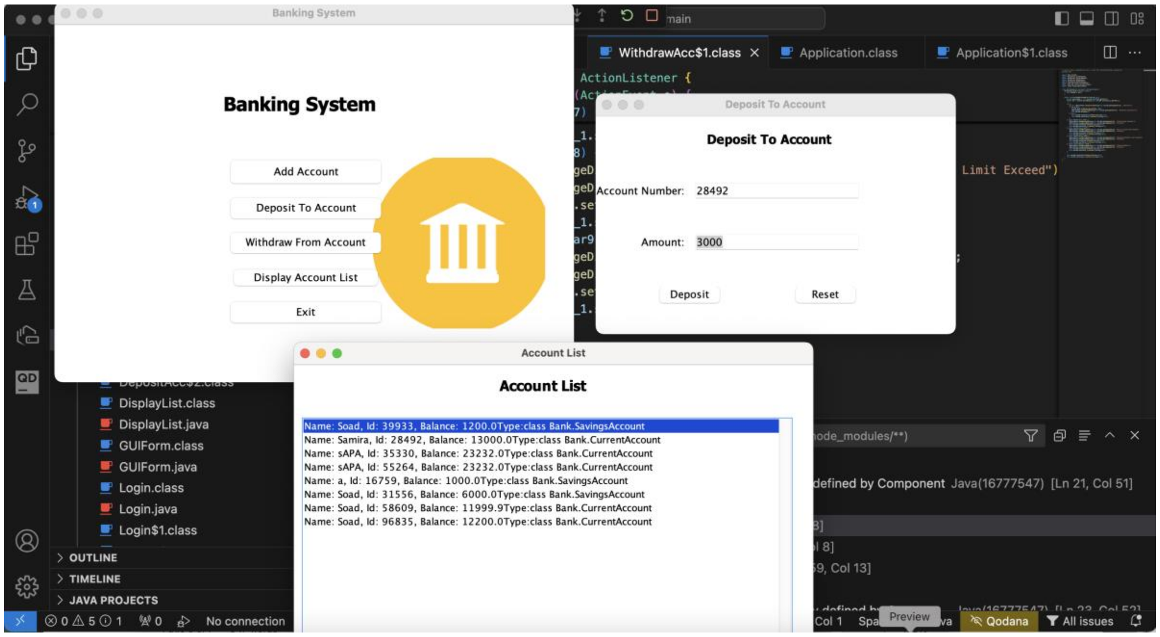
This screen displays a comprehensive list of all the accounts currently in the banking system. Each account is detailed with the following information:

- **Name:** The name of the account holder (e.g., "Soad," "Samira").
- **Account ID:** A unique identifier for each account (e.g., 39933, 28492).
- **Balance:** The current balance in the account (e.g., 1200.0, 13000.0).
- **Account Type:** The type of account, such as Savings or Current Account.

This view is particularly useful for administrators to monitor all accounts at a glance, ensuring transparency and quick access to account details. For instance, they can spot accounts with low balances or duplicate names that might need investigation.

2. Integration Testing

For integration testing, I'll focus on how the system's modules work together, especially the account module, transaction module. I'll run test cases to check things like:
- When a user makes a deposit or withdrawal, does their account balance update correctly?



Deposit To Account

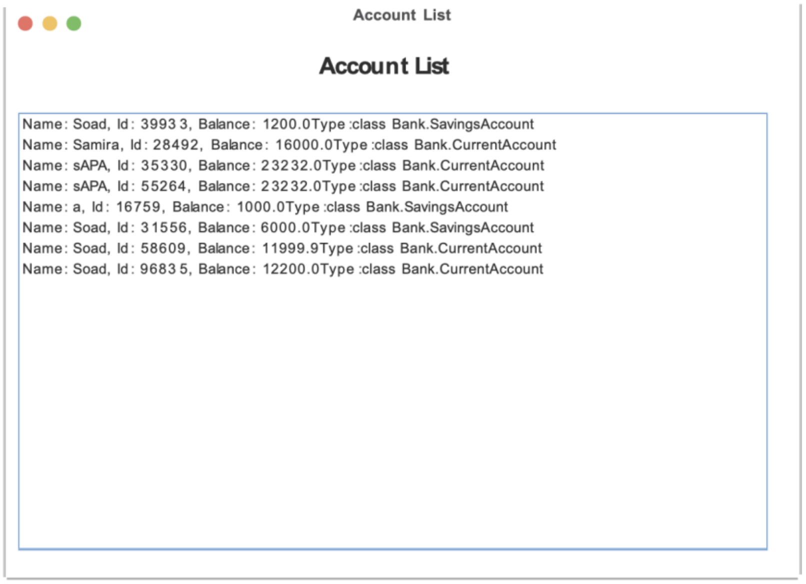
Account Number: 28492

Amount: 3000

Account List

```
Name: Soad, Id: 39933, Balance: 1200.0Type:class Bank.SavingsAccount
Name: Samira, Id: 28492, Balance: 13000.0Type:class Bank.CurrentAccount
Name: sAPA, Id: 35330, Balance: 23232.0Type:class Bank.CurrentAccount
Name: sAPA, Id: 55264, Balance: 23232.0Type:class Bank.CurrentAccount
Name: a, Id: 16759, Balance: 1000.0Type:class Bank.SavingsAccount
Name: Soad, Id: 31556, Balance: 6000.0Type:class Bank.SavingsAccount
Name: Soad, Id: 58609, Balance: 11999.9Type:class Bank.CurrentAccount
Name: Soad, Id: 96835, Balance: 12200.0Type:class Bank.CurrentAccount
```

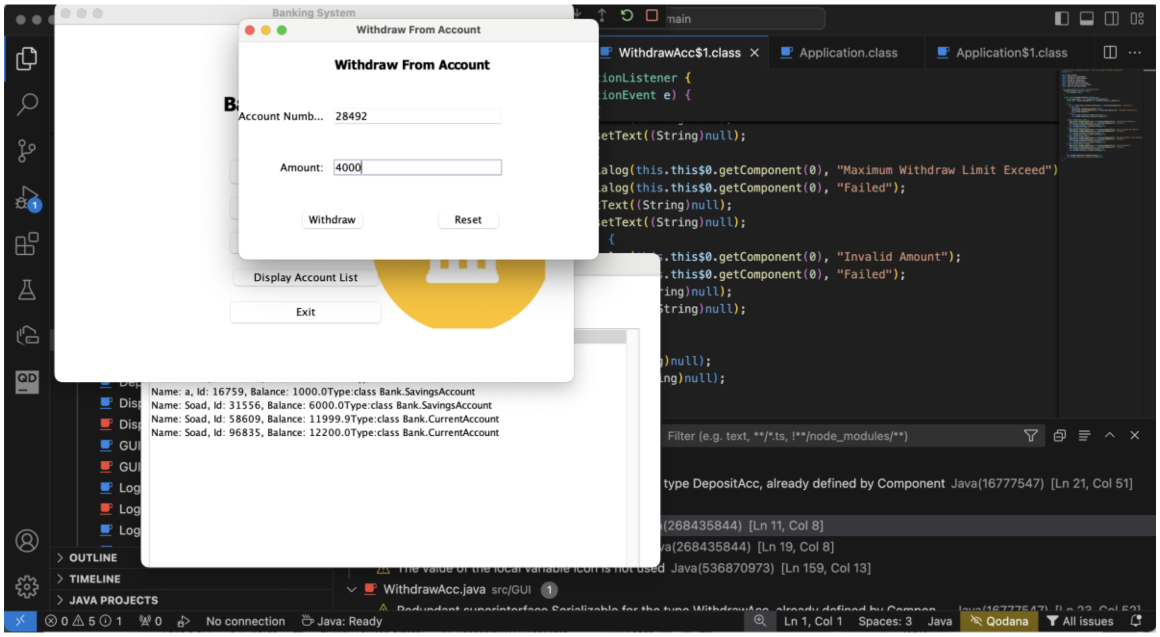
1-First of all we take a account from the list and than we deposit 3000 amount to the account and than we check again if the account list is updated.



Account List

```
Name: Soad, Id: 39933, Balance: 1200.0Type :class Bank.SavingsAccount
Name: Samira, Id: 28492, Balance: 16000.0Type :class Bank.CurrentAccount
Name: sAPA, Id: 35330, Balance: 23232.0Type :class Bank.CurrentAccount
Name: sAPA, Id: 55264, Balance: 23232.0Type :class Bank.CurrentAccount
Name: a, Id: 16759, Balance: 1000.0Type :class Bank.SavingsAccount
Name: Soad, Id: 31556, Balance: 6000.0Type :class Bank.SavingsAccount
Name: Soad, Id: 58609, Balance: 11999.9Type :class Bank.CurrentAccount
Name: Soad, Id: 96835, Balance: 12200.0Type :class Bank.CurrentAccount
```

2-After that will proceed with the withdrawal class and we will se to the same account if the amount that will be withdrawal will be updated to the account list.

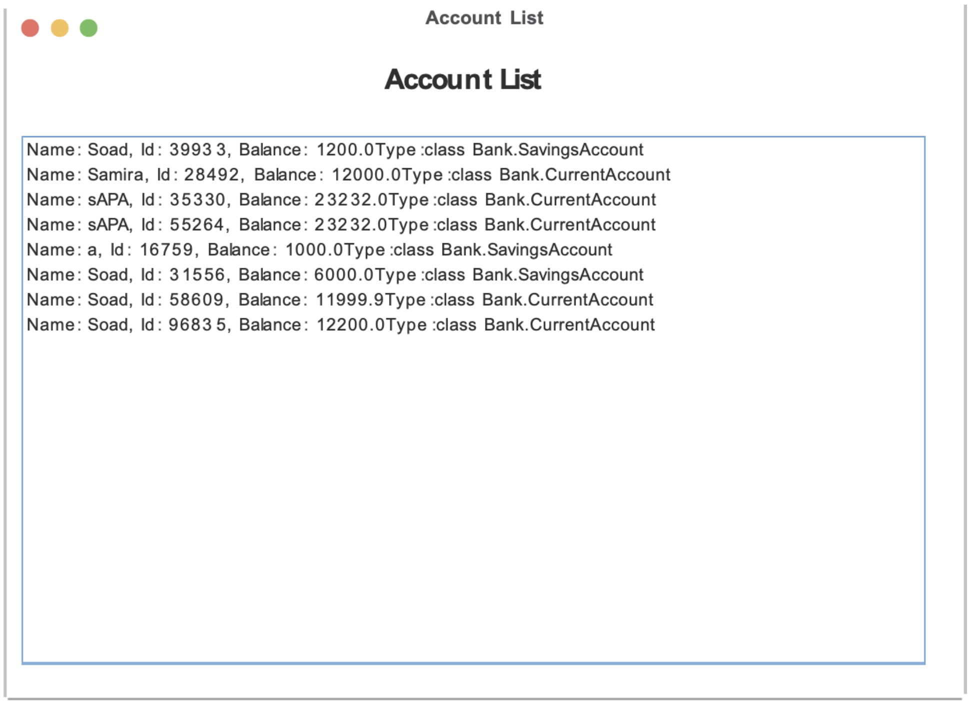


```
Name: a, Id: 16759, Balance: 1000.0Type:class Bank.SavingsAccount
Name: Soad, Id: 31556, Balance: 6000.0Type:class Bank.SavingsAccount
Name: Soad, Id: 58609, Balance: 11999.9Type:class Bank.CurrentAccount
Name: Soad, Id: 96835, Balance: 12200.0Type:class Bank.CurrentAccount
```

```
ionListener {
ionEvent e) {
etText((String)null);
alog(this.this$0.getComponent(0), "Maximum Withdraw Limit Exceed")
alog(this.this$0.getComponent(0), "Failed");
Text((String)null);
etText((String)null);
 {
s.this$0.getComponent(0), "Invalid Amount");
.this$0.getComponent(0), "Failed");
ing)null);
String)null);
)null);
ng)null);
```

```
type DepositAcc, already defined by Component Java(16777547) [Ln 21, Col 51]
(268435844) [Ln 11, Col 8]
a(268435844) [Ln 19, Col 8]
Java(536870973) [Ln 159, Col 13]
```

3-Now check the account list.



Account List

```
Name: Soad, Id: 39933, Balance: 1200.0Type :class Bank.SavingsAccount
Name: Samira, Id: 28492, Balance: 12000.0Type :class Bank.CurrentAccount
Name: sAPA, Id: 35330, Balance: 23232.0Type :class Bank.CurrentAccount
Name: sAPA, Id: 55264, Balance: 23232.0Type :class Bank.CurrentAccount
Name: a, Id: 16759, Balance: 1000.0Type :class Bank.SavingsAccount
Name: Soad, Id: 31556, Balance: 6000.0Type :class Bank.SavingsAccount
Name: Soad, Id: 58609, Balance: 11999.9Type :class Bank.CurrentAccount
Name: Soad, Id: 96835, Balance: 12200.0Type :class Bank.CurrentAccount
```

4-During the process all the calculations are performed correctly so the program works fine