Project Document: Console Based Banking Operations System

Table of Contents

1. Introduction

1.1. Purpose of the Document

The purpose of this document is to provide an overview of the Bank Operation Project developed using Python. It outlines the system's requirements,
architecture, features, user interface, testing procedures.

1.2. Project Overview

The Banking Operation System is a Console-Based Application that allows users to perform various banking activities such as account management operations like Deposit, Withdrawal, Fund Transfers, Balance Enquiry.

1.3. Scope

The scope of the Online Banking System project includes the following functionalities:

- - Account management (Create Account, Deposit Amount, Withdraw Amount).

- Fund transfers between accounts.

- Balance Inquiries.

2. System Requirements

2.1. Functional Requirements

The Functional Requirements for this Python Project for Banking Operations to create a class hierarchy of bank accounts where we can deposit cash, obtain the balance and make a withdrawal and transfer amount.

Some of the accounts provide interest and others charge fees for withdrawals.

The various classes to be used as follows...

The BankAccount class

Consider the attributes and operations of a BankAccount Class

It's usually best to consider the operations first then provide attributes as needed to support these operations

1. Deposit cash,
2. Withdraw cash,
3. Check current balance and
4. Transfer funds to another account.

To support these operations we will need the attributes like the current_balance.

Basic Skeleton of Bank

Account Class public

class BankAccount

```
public BankAccount (double
    initialAmount) this.balance =
    initialAmount
    print("Account created with balance " , balance)
deposit(double amount)
    # Update the Balance
accordingly withdraw(double
```

amount)

Update the Balance accordingly with validity of operation chkBalance()# This is to check the balance transfer(amount, account)

The SavingsAccount class

Use inheritance to create savings account that adds 3% interest on every deposit

The CurrentAccount class

Again, Use inheritance to create current account that charges INR 200 for every withdrawal

 # Tester Class Activities

 # Test out the BankAccount

Create 2 BackAccounts with intial Balance 500 and 200 B1 = BankAccount(500); B2 = BankAccount(200);

withdraw 200 from 1st BankAccount

deposit 1000 in second BankAccount

transfer 500 from second to first

Bank Account # Then check balances

of both accounts

 # Test out the SavingsAccount

#Create a SavingsAccounts with intial

Balance 2000 #Deposit 500 in it

#Check its balance

 # Test out the CurrentAccount

#Create a CurrentAccounts with intial

Balance 2000 #Deposit 500 to it

#withdraw 100 from it

#Transfer 500 to First SavingsAccount

Then check balances of both accounts - CurrentAccount and First SavingsAccount

 # Finally transfer from current account to the saving account

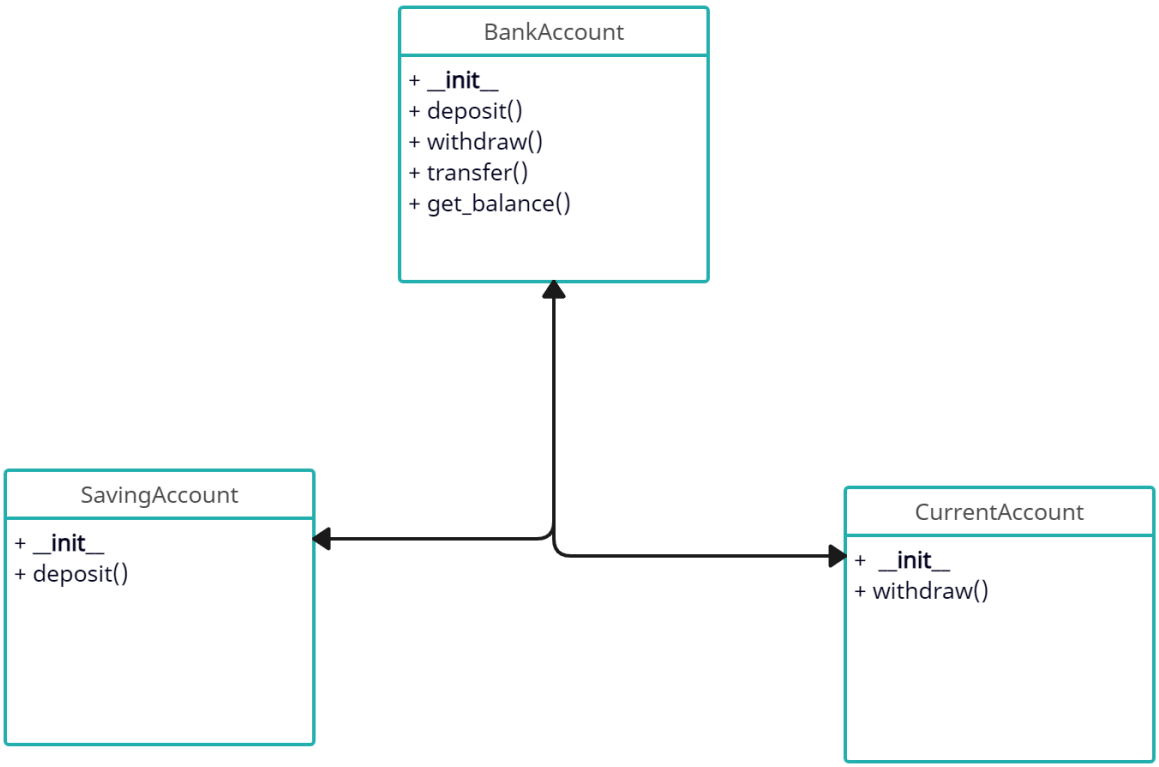
 # The current account should charge and the saving account should add interest

3. Architecture

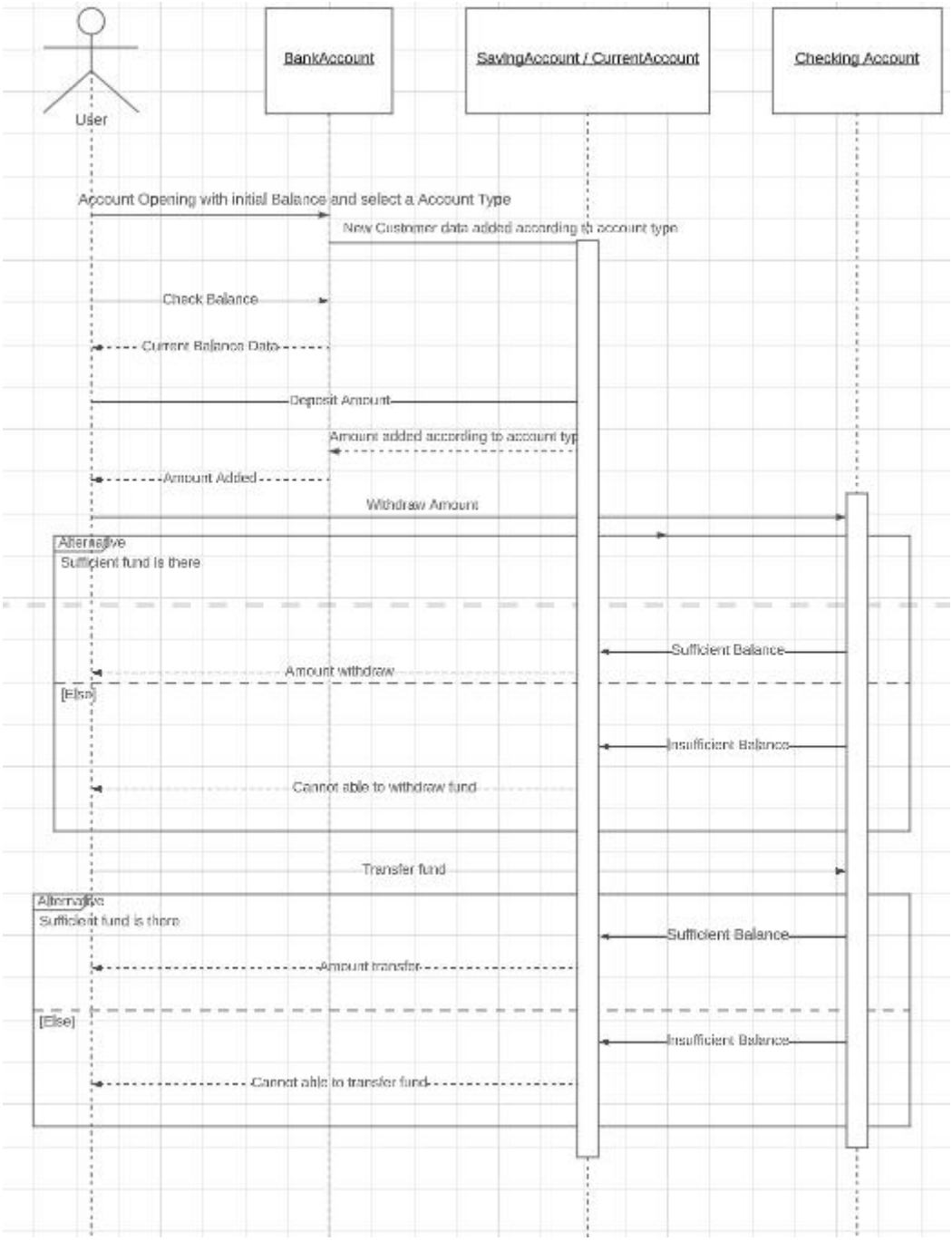
3.1. High-Level Architecture

The Online Banking System will follow a client-server architecture. The client- side will consist of a web-based user interface, while the server-side will handle the application logic, data processing, and database interactions.

3.2. Class Diagram



3.3. Sequence Diagrams



User Interface

```
1. Create savings account
2. Create Current account
3. Deposit
4. Withdraw
5. Transfer
6. Check Balance
7. Exit
Enter your choice:2
Enter the account number: 123456789
Enter account holder name: Business
```

Since this is a console based application we are providing the values by yourself only

4. Technologies Used

Programming Language Used : Python

Platform : Pycharm

Operating System : Windows, Linux or any other

5. Testing

5.1. Test Cases

A variety of test cases were developed to cover both standard and extreme circumstances for the deposit, withdraw, and transfer functionalities.

5.2. Unit Testing

Isolating the Functions: To identify specific problems, each function was independently tested.

Test Data: A variety of test data that mimicked actual scenarios, including both valid and incorrect inputs.

Execution: Tests were carried out, and actual and anticipated outcomes were compared.

Extreme circumstances, such as going over account limitations, were examined as edge cases.

Documentation: Test cases were meticulously documented.

This strategy ensured the validity, dependability, and security of these vital

Unit Test Cases

Test Cases for Initial Amount

Test Cases	Amount	Expected value	Actual Value	Remark
T1	-5	Invalid	Invalid	Pass
T2	0	Invalid	Invalid	Pass
T3	2000	Valid	Valid	Pass

Test Cases for Deposit Amount

Test Cases	Amount	Expected value	Actual Value	Remark
T1	-7	Invalid	Invalid	Pass
T2	0	Invalid	Invalid	Pass
T3	1000	Valid	Valid	Pass

Test Cases for Withdraw Amount

Test Cases	Amount	Expected value	Actual Value	Remark
T1	-10	Invalid	Invalid	Pass
T2	0	Invalid	Invalid	Pass
T3	500	Valid	Valid	Pass

6. Conclusion

The Console-based banking application offers a suite of functionalities to manage accounts. Users can use “Deposit” function to add funds . The “Withdrawal” function lets user deduct the funds within set limits, ensuring balance sufficiency and transaction records. The “transfer” function allows seamless movement of funds between accounts , adhering to account type constraints and logging transfers. The “Check_Balance” function provides real-time account balance information. These functions collectively empower users to perform financial tasks securely and conveniently through the application’s text-based interface.

7. References

https://www.javatpoint.com/python-tutorial

https://www.geeksforgeeks.org/python-programming-language/