Project Document: Console Based Banking Operations System

Table of Content

[1. Introduction 2](#_Toc28230)

[1.1. Purpose of the Document 2](#_Toc18873)

[1.2. Project Overview 2](#_Toc7243)

[1.3. Scope 2](#_Toc16326)

[2. System Requirements 3](#_Toc26094)

[2.1. Functional Requirements 3](#_Toc23328)

[The BankAccount class 3](#_Toc14398)

[Basic Skeleton of Bank Account Class 3](#_Toc20407)

[The CurrentAccount class 4](#_Toc3586)

[# Tester Class Activities 5](#_Toc20840)

[# Test out the SavingsAccount 6](#_Toc10003)

[# Test out the CurrentAccount 6](#_Toc13936)

[# Finally transfer from current account to the saving account 6](#_Toc18286)

[3. Architecture 7](#_Toc18085)

[3.1. High-Level Architecture 7](#_Toc9244)

[3.2. Class Diagram 8](#_Toc25935)

[3.3. Sequence Diagrams 9](#_Toc5156)

[4. User Interface 10](#_Toc29385)

[5. Technologies Used 10](#_Toc3608)

[6. Testing 11](#_Toc27750)

[6.1. Test Cases 11](#_Toc32508)

[6.2. Unit Testing 11](#_Toc17443)

[Unit Test Cases 11](#_Toc11687)

[7. Conclusion 12](#_Toc30500)

[8. References 12](#_Toc13759)

# Introduction

## 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.

## 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.

## 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.

# System Requirements

## 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

from BankAccount import BankAccount

from SavingAccount import SavingAccount

from CurrentAccount import CurrentAccount

print("Saving Account")

print()

s1=SavingAccount(2000)

s1.DepoistAmount(500)

print("Current Account")

print()

c1=CurrentAccount(2000)

c1.DepoistAmount(500)

c1.WithdrawAmount(100)

c1.TransferFund(s1,500)

print("Current Balance of Saving Account :",s1.getCurrentBalance())

print("Current Balance of Current Account:",c1.getCurrentBalance())

**The SavingsAccount class**

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

**Basic Skeleton of Saving Account Class**

from BankAccount import BankAccount

class SavingAccount(BankAccount):

def DepoistAmount(self, amount):

if(amount<=0):

print("For Depoist : Enter Amount must be greater than 0")

print()

else:

interest=(amount)\*0.03

super().DepoistAmount(amount+interest)

print("Interest :",interest)

print("Amount Deposit :",amount)

print("Total Amount added in Saving Account:",amount+interest)

print()

# The CurrentAccount class

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

**Basic Skeleton of Bank Account Class**

from BankAccount import BankAccount

class CurrentAccount(BankAccount):

def WithdrawAmount(self, amount):

if(self.getCurrentBalance()>0 and self.getCurrentBalance()>=amount+200):

charge=200

super().WithdrawAmount(amount+charge)

print("Interest Charged:",charge)

print(amount," Withdraw from the Current Account")

print("Total Amount deduced from Current Account :",amount + charge)

print()

else:

print("Do not have sufficient balance for withdraw in Current Account")

print()

# # Tester Class Activities

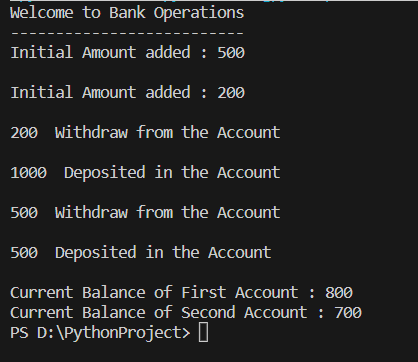
**# Test out the BankAccount**

# 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

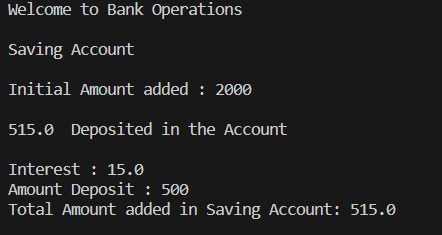
#withdraw 100 from it

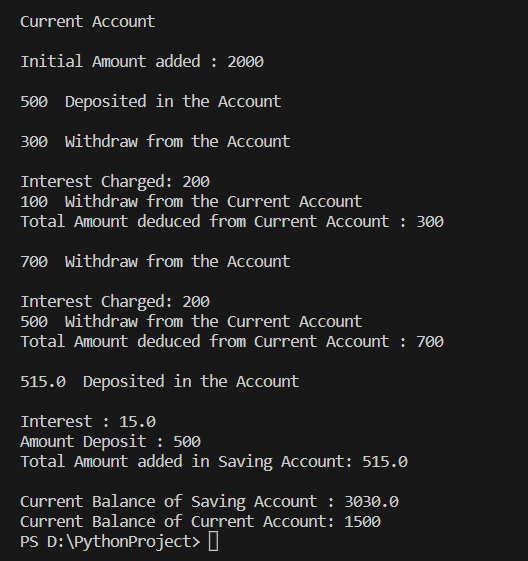
#Transfer 500 to First SavingsAccount

# 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**



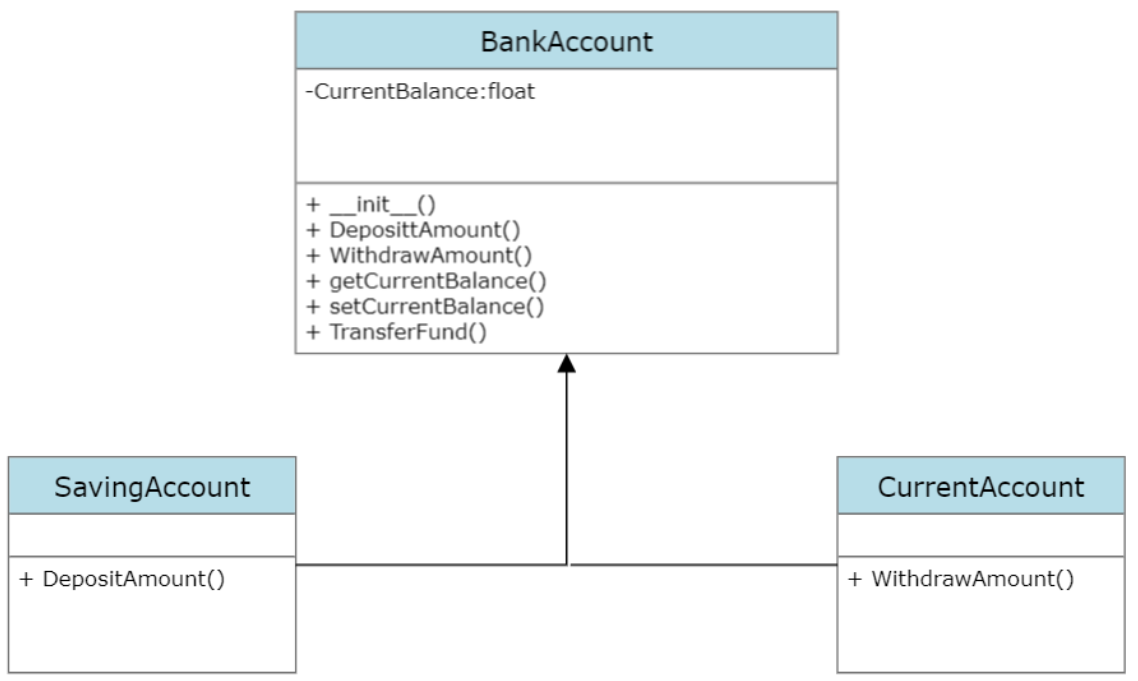


# Architecture

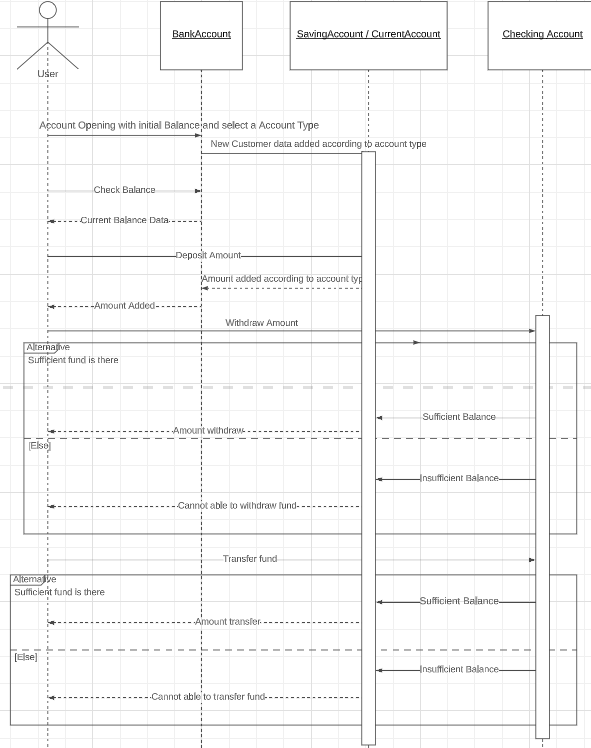
## High-Level Architecture

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.

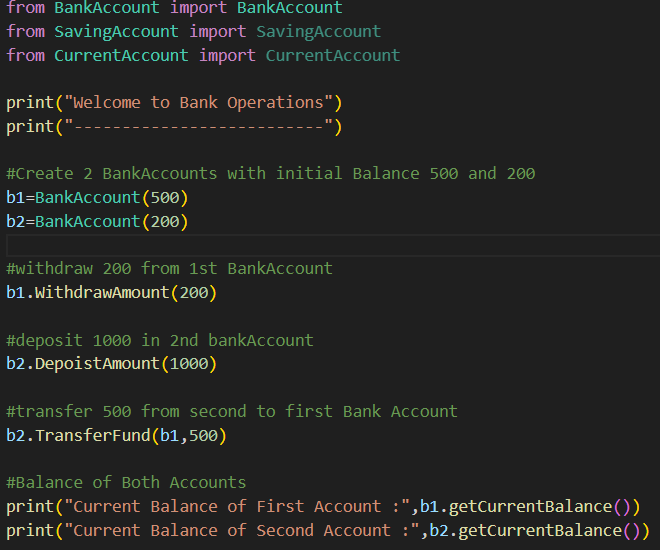
## Class Diagram



## Sequence Diagrams



# User Interface



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

# Technologies Used

Programming Language Used : Python

Platform : Pycharm

Operating System : Windows, Linux or any other

# Testing

## Test Cases

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

## 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 |

# Conclusion

This project was effective in putting in place a user-friendly system for bank operations, including balance enquiries, fund transfers, withdrawals, and its Security, effectiveness, and consistency were given top priority during the system design. Future changes will be necessary to adjust to changing user needs and technological advancements. Overall, the project represents an important step towards improving contemporary banking experiences through technological innovations.

# References

* <https://www.geeksforgeeks.org/python-programming-language/>
* <https://www.w3schools.com/python/>