**Use Case for: Bank Account Management System**

**Description:**

The bank account management system allows customers to perform deposit and withdrawal operations on their bank accounts. It provides an interface for users to interact with their accounts and maintains a list of bank account numbers

**Actors:**

- User

- Bank

**Preconditions:**

- The Bank Account Management System must be operational.

**Flow of Events:**

1. User launches the Bank Account Management System.

2. User is prompted to enter their bank account number.

3. User enters their bank account number.

4. The system creates an instance of the BankAccount class with the provided account number and initializes the account balance to 0.

5. The system adds the account number to the list of bank accounts.

6. The system displays a success message indicating that the account has been created.

7. The system presents the user with the available actions:

- Deposit

- Withdraw

8. User selects an action by entering the corresponding choice:

- If the user selects "Deposit":

- The system prompts the user to enter the deposit amount.

- User enters the amount to deposit.

- The system calls the `deposit()` method of the BankAccount instance, passing the deposit amount.

- The system updates the account balance accordingly.

- The system displays a success message with the updated balance.

- If the user selects "Withdraw":

- The system prompts the user to enter the withdrawal amount.

- User enters the amount to withdraw.

- The system calls the `withdraw()` method of the BankAccount instance, passing the withdrawal amount.

- The system checks if the account has sufficient funds for the withdrawal:

- If the account balance is greater than or equal to the withdrawal amount:

- The system deducts the withdrawal amount from the account balance.

- The system displays a success message with the updated balance.

- If the account balance is less than the withdrawal amount:

- The system displays an error message indicating insufficient funds.

9. The system retrieves the current balance of the account using the `get\_balance()` method.

10. The system displays the account number and the current balance.

11. The system terminates.

**Postconditions:**

- The user can create a bank account and perform deposit or withdrawal operations on that account.

- The Bank Account Management System maintains a list of bank account numbers for reference.

- The system provides the user with the current balance of the account.

This use case outlines the basic flow of events for the Bank Account Management System. It covers the process of creating a bank account, performing deposit and withdrawal operations, and retrieving the account balance. The system also maintains a list of bank account numbers for reference. Additional features like account balance inquiries, transaction history, and user authentication could be added to enhance the functionality of the system.

**Problem Statement for: Bank Account Management System**

You are tasked with implementing a Bank Account Management System using the Python programming language. The system should allow users to perform various operations on their bank accounts, including depositing and withdrawing funds, as well as checking the current balance.

**Functional Requirements:**

1. Create BankAccount class:

* The BankAccount class should have the following attributes:
* account\_number (integer): A unique identifier for the bank account.
* balance (float): The current balance in the account.
* The BankAccount class should have the following methods:
* \_\_init\_\_(self, account\_number): Initializes a new bank account with the given account number and a balance of 0.
* deposit(self, amount): Deposits the specified amount into the account and updates the balance accordingly.
* withdraw(self, amount): Withdraws the specified amount from the account, if the account has sufficient funds, and updates the balance accordingly.
* get\_balance(self): Returns the current balance in the account.

1. Create an empty list bankaccount to store bank account numbers.
2. Prompt the user to enter their account number and store it in the account\_number variable.
3. Create an instance of the BankAccount class with the provided account number.
4. Add the account number to the bankaccount list.
5. Present the user with the following options:

* Deposit: Prompt the user to enter the amount to deposit. Call the deposit() method of the BankAccount instance with the deposit amount.
* Withdraw: Prompt the user to enter the amount to withdraw. Call the withdraw() method of the BankAccount instance with the withdrawal amount.
* Cancel: Display a message indicating that the transaction has been canceled.
* Invalid choice: Display a message indicating that the user has made an invalid choice.

1. Display the account number and the current balance using the get\_balance() method.

**Non-functional Requirements:**

* The account number should be an integer.
* The balance should be a float and should not be allowed to go below 0.
* The program should handle invalid inputs, such as non-numeric values or negative amounts, and provide appropriate error messages to the user.
* The program should gracefully handle exceptions and prevent any unexpected crashes.
* The program should be easy to understand, with clear variable and method names and appropriate comments.

**PROGRAM FOR Bank Account Management**

print()

class BankAccount :

# account\_number=int()

# balance=float(0.0)

def \_\_init\_\_(self,account\_number):

self.account\_number =account\_number

self.balance=0.0

def deposit(self,amount):

self.balance +=amount

if self.deposit:

print(f"Amount deposited successfully to - {self.account\_number}",f"balance : {self.balance}")

else:

print("You have canceld Thank You!")

def withdraw(self,amount):

if self.balance >= amount:

self.balance -= amount

print(f"Amount withdraw successfully from -{self.account\_number} ")

else :

print(f"Insaficient found in Account: {self.account\_number}")

def get\_balance(self):

return self.balance

num = int(input("Enter account numbers: "))

account\_numbers=1000

bankaccount = []

for i in range(num):

accNum=account\_numbers + (i+1)

num1=BankAccount(accNum)

bankaccount.append(num1)

print(f" \* Account created successfully:{accNum}")

print()

opt = input("Enter choice (1-deposit, 2-withdraw,3-check balance 9-cancel): ")

if opt == "1":

deposit\_amount = float(input("Enter the amount to deposit: "))

num1.deposit(deposit\_amount)

elif opt == "2":

withdrawal\_amount = int(input("Enter the amount to withdraw: "))

num1.withdraw(withdrawal\_amount)

elif opt == "3":

print(f"Your balance is : {num1.get\_balance()}")

elif opt == "9":

print("Transaction canceled")

else:

print("Invalid choice")

break

print(f"Account number:{num1.account\_number} ",f"Current balance:{num1.get\_balance()}")

print()

**Explanation of code :**

1. The BankAccount class is defined, which will represent a bank account. It has two attributes, account\_number and balance. The account\_number is initialized with the value passed to the constructor, and the balance is set to 0.0 by default.
2. The \_\_init\_\_ method is the constructor of the BankAccount class. It takes an account\_number parameter and initializes the account\_number and balance attributes accordingly.
3. The deposit method is used to deposit an amount into the account. It takes an amount parameter, adds the amount to the current balance, and prints a success message along with the updated balance.
4. The withdraw method is used to withdraw an amount from the account. It takes an amount parameter, checks if the account has sufficient balance, subtracts the amount from the balance if possible, and prints a success message. If the balance is insufficient, it prints a corresponding message.
5. The get\_balance method returns the current balance of the account.
6. The code prompts the user to enter the number of accounts they want to create (num). It also initializes the account\_numbers variable with a starting value of 1000.
7. A loop is executed num times to create num bank accounts. Inside the loop, an account number is generated by adding the current value of account\_numbers with the loop index (i+1). An instance of BankAccount is created with the generated account number, and it is appended to the bankaccount list.
8. After creating an account, the user is prompted to enter their choice: 1 for deposit, 2 for withdrawal, 3 for checking the balance, or 9 to cancel the transaction.
9. Depending on the user's choice, the corresponding action is performed on the num1 account (the current account being processed in the loop). If the choice is 1, the user is prompted to enter the amount to deposit, and the deposit method of the account is called. If the choice is 2, the user is prompted to enter the amount to withdraw, and the withdraw method of the account is called. If the choice is 3, the get\_balance method is called and the current balance is printed. If the choice is 9, a transaction canceled message is printed.
10. Finally, after performing the user's selected action, the account number and current balance of the num1 account are printed.

That's the explanation of the given code. It provides a basic framework for creating multiple bank accounts, depositing and withdrawing funds, and checking the balances.

**Output for Bank Account Management Code:**

Enter account numbers: 2

\* Account created successfully:1001

Enter choice (1-deposit, 2-withdraw,3-check balance 9-cancel): 1

Enter the amount to deposit: 3000

Amount deposited successfully to - 1001 balance : 3000.0

Account number:1001 Current balance:3000.0

\* Account created successfully:1002

Enter choice (1-deposit, 2-withdraw,3-check balance 9-cancel): 1

Enter the amount to deposit: 333

Amount deposited successfully to - 1002 balance : 333.0

Account number:1002 Current balance:333.0