**Name:** Dhanashri D. Mahapatra **PRN:** 21070521120

**Department:** CSE **Division:** B

**Symbiosis Institute of Technology**

**Department of Applied Science**



**Flexi Credit Course (Generative AI)**

**Computer Science and Engineering**

**Batch 2021-2025**

**Semester - VII Course Code – F0003**

# Problem Statement

Generate a model in Python for representation of a bank account of type savings and

balance along with transactions of deposit and withdrawals and currently create a program to

generate 100 accounts with Random balance and transactions for no. of months and no. of

transactions with a seed value of amount. Print all 100 accounts with the last balance and

organize them by lowest to highest balance.

1. **System Design**

-Account Class:

- Attributes: `account\_id`, `balance`, `transactions[]`

- Functions:

- `deposit(amount)`: Increases balance, logs the deposit.

- `withdraw(amount)`: Decreases balance if funds are available, logs the withdrawal.

- `\_\_str\_\_()`: Returns account details (ID and balance).

- System Behavior:

- Generate 100 accounts with random initial balances.

- For each account, simulate monthly transactions (deposits/withdrawals).

- Sort accounts by balance at the end and display them from lowest to highest balance.

1. **Code**

import random

class SavingsAccount:

    def \_\_init\_\_(self, account\_id, initial\_balance):

        self.account\_id = account\_id

        self.balance = initial\_balance

        self.transactions = []

    def deposit(self, amount):

        self.balance += amount

        self.transactions.append(f"Deposit: +{amount}")

    def withdraw(self, amount):

        if self.balance >= amount:

            self.balance -= amount

            self.transactions.append(f"Withdraw: -{amount}")

        else:

            self.transactions.append(f"Failed Withdraw: -{amount} (Insufficient funds)")

    def \_\_str\_\_(self):

        return f"Account ID: {self.account\_id}, Balance: {self.balance}"

def generate\_random\_account(account\_id, initial\_balance, months, transactions\_per\_month, seed):

    random.seed(seed)

    account = SavingsAccount(account\_id, initial\_balance)

    for \_ in range(months):

        for \_ in range(transactions\_per\_month):

            amount = random.randint(1, 1000)

            if random.choice(['deposit', 'withdraw']) == 'deposit':

                account.deposit(amount)

            else:

                account.withdraw(amount)

    return account

def main():

    accounts = []

    num\_accounts = 100

    initial\_seed = 1000  # Seed for randomization

    # Generate 100 accounts

    for account\_id in range(1, num\_accounts + 1):

        initial\_balance = random.randint(1000, 10000)  # Random initial balance between 1000 and 10000

        months = random.randint(1, 12)  # Random number of months (1-12)

        transactions\_per\_month = random.randint(1, 10)  # Random number of transactions per month (1-10)

        account = generate\_random\_account(account\_id, initial\_balance, months, transactions\_per\_month, seed=initial\_seed)

        accounts.append(account)

        initial\_seed += 1  # Change seed for randomness

    # Sort accounts by balance

    accounts.sort(key=lambda acc: acc.balance)

    # Print sorted accounts

    for account in accounts:

**print(account)**

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output-

Account ID: 44, Balance: 3

Account ID: 43, Balance: 433

Account ID: 95, Balance: 522

Account ID: 48, Balance: 684

Account ID: 55, Balance: 733

Account ID: 16, Balance: 851

Account ID: 30, Balance: 1173

Account ID: 70, Balance: 1214

Account ID: 17, Balance: 1471

Account ID: 81, Balance: 1594

Account ID: 62, Balance: 1595

Account ID: 26, Balance: 1886

Account ID: 50, Balance: 1888

Account ID: 42, Balance: 1961

Account ID: 56, Balance: 2079

Account ID: 100, Balance: 2161

Account ID: 80, Balance: 2177

Account ID: 21, Balance: 2216

Account ID: 46, Balance: 2266

Account ID: 64, Balance: 2548

Account ID: 93, Balance: 2615

Account ID: 7, Balance: 2892

Account ID: 20, Balance: 2900

Account ID: 66, Balance: 2955

Account ID: 82, Balance: 2997

Account ID: 74, Balance: 3002

Account ID: 85, Balance: 3177

Account ID: 87, Balance: 3227

Account ID: 68, Balance: 3300

Account ID: 38, Balance: 3518

Account ID: 72, Balance: 3539

Account ID: 14, Balance: 3571

Account ID: 23, Balance: 3715

Account ID: 6, Balance: 3795

Account ID: 54, Balance: 3850

Account ID: 5, Balance: 3932

Account ID: 83, Balance: 4238

Account ID: 75, Balance: 4379

Account ID: 3, Balance: 4468

Account ID: 88, Balance: 4528

Account ID: 27, Balance: 4630

Account ID: 97, Balance: 4650

Account ID: 92, Balance: 4658

Account ID: 31, Balance: 4686

Account ID: 33, Balance: 4720

Account ID: 84, Balance: 4785

Account ID: 78, Balance: 4819

Account ID: 91, Balance: 4928

Account ID: 18, Balance: 5037

Account ID: 73, Balance: 5126

Account ID: 24, Balance: 5237

Account ID: 11, Balance: 5381

Account ID: 29, Balance: 5411

Account ID: 4, Balance: 5432

Account ID: 63, Balance: 5565

Account ID: 98, Balance: 5569

Account ID: 53, Balance: 5696

Account ID: 28, Balance: 5847

Account ID: 69, Balance: 5884

Account ID: 67, Balance: 6057

Account ID: 8, Balance: 6058

Account ID: 19, Balance: 6192

Account ID: 89, Balance: 6226

Account ID: 36, Balance: 6381

Account ID: 1, Balance: 6562

Account ID: 12, Balance: 6623

Account ID: 15, Balance: 6647

Account ID: 41, Balance: 6680

Account ID: 39, Balance: 6813

Account ID: 51, Balance: 7260

Account ID: 79, Balance: 7444

Account ID: 22, Balance: 7604

Account ID: 10, Balance: 8148

Account ID: 94, Balance: 8178

Account ID: 32, Balance: 8258

Account ID: 37, Balance: 8265

Account ID: 2, Balance: 8419

Account ID: 60, Balance: 8591

Account ID: 25, Balance: 8653

Account ID: 35, Balance: 9059

Account ID: 59, Balance: 9150

Account ID: 52, Balance: 9249

Account ID: 34, Balance: 9426

Account ID: 9, Balance: 9755

Account ID: 96, Balance: 9804

Account ID: 90, Balance: 10270

Account ID: 65, Balance: 10484

Account ID: 13, Balance: 11119

Account ID: 40, Balance: 11296

Account ID: 45, Balance: 11362

Account ID: 76, Balance: 11592

Account ID: 47, Balance: 11631

Account ID: 71, Balance: 11875

Account ID: 61, Balance: 13151

Account ID: 49, Balance: 13488

Account ID: 57, Balance: 13938

Account ID: 58, Balance: 15218

Account ID: 99, Balance: 15471

Account ID: 86, Balance: 15765

Account ID: 77, Balance: 19291

1. **Conclusion**

This system effectively models a basic savings account with functionalities for deposits and withdrawals, while handling common scenarios like insufficient funds. By simulating random transactions over multiple months for 100 accounts, it provides a snapshot of account balances and enables easy sorting from lowest to highest. The design is simple yet scalable, making it adaptable for more complex financial operations or integration into larger banking systems.

2.

1. **Problem Statement**

Generate a model to represent interest calculations of a Bank account where the process of

calculating interest for 6 months is a. Find minimum balance for each month b. Make a total of all minimum balances c. Calculate interest based on interest rate d. Divide interest by 12 to

find one-month interest e. Multiply interest by 6 to show interest in the account. Generate a

model to represent transactions and interest calculations for 6 months.

1. **System Design**

Bank Account Interest Calculation

- BankAccount Class:

- Attributes: `account\_id`, `balance`, `transactions[]`, `monthly\_min\_balances[]`, `interest\_rate`.

- Functions:

- `deposit(amount)`: Adds the deposit to the balance.

- `withdraw(amount)`: Subtracts the amount from the balance if sufficient funds.

- `calculate\_min\_balance\_for\_month()`: Simulates transactions and stores the monthly minimum balance.

- `calculate\_interest\_for\_6\_months()`: Calculates interest based on the 6-month minimum balances.

- System Behavior:

- Simulate 6 months of transactions.

- Calculate interest based on the minimum balances for those months.

1. **Code**

import random

class BankAccount:

    def \_\_init\_\_(self, account\_id, initial\_balance, interest\_rate):

        self.account\_id = account\_id

        self.balance = initial\_balance

        self.transactions = []  # Log of transactions

        self.monthly\_min\_balances = []  # Store minimum balance for each month

        self.interest\_rate = interest\_rate  # Annual interest rate

    def deposit(self, amount):

        self.balance += amount

        self.transactions.append(f"Deposit: +{amount}")

    def withdraw(self, amount):

        if self.balance >= amount:

            self.balance -= amount

            self.transactions.append(f"Withdraw: -{amount}")

        else:

            self.transactions.append(f"Failed Withdraw: -{amount} (Insufficient funds)")

    def calculate\_min\_balance\_for\_month(self):

        """ Simulate monthly transactions and store the minimum balance for that month """

        min\_balance = self.balance

        for \_ in range(random.randint(1, 10)):  # Random number of transactions in a month

            transaction\_amount = random.randint(1, 500)

            if random.choice([True, False]):

                self.deposit(transaction\_amount)

            else:

                self.withdraw(transaction\_amount)

            min\_balance = min(min\_balance, self.balance)

        self.monthly\_min\_balances.append(min\_balance)

    def calculate\_interest\_for\_6\_months(self):

        """ Calculate the interest for 6 months based on the minimum balances """

        total\_min\_balance = sum(self.monthly\_min\_balances)

        annual\_interest = (total\_min\_balance \* self.interest\_rate) / 100

        monthly\_interest = annual\_interest / 12

        interest\_for\_6\_months = monthly\_interest \* 6

        return interest\_for\_6\_months

    def \_\_str\_\_(self):

        return (f"Account ID: {self.account\_id}, Current Balance: {self.balance}, "

                f"Minimum Balances (6 months): {self.monthly\_min\_balances}")

def simulate\_bank\_accounts(num\_accounts=5, interest\_rate=5):

    accounts = []

    for account\_id in range(1, num\_accounts + 1):

        initial\_balance = random.randint(1000, 10000)

        account = BankAccount(account\_id, initial\_balance, interest\_rate)

        # Simulate 6 months of transactions and calculate interest

        for \_ in range(6):

            account.calculate\_min\_balance\_for\_month()

        interest = account.calculate\_interest\_for\_6\_months()

        accounts.append((account, interest))

    # Display account details and interest calculations

    for account, interest in accounts:

        print(account)

        print(f"Interest earned over 6 months: {interest:.2f}\n")

# Run the simulation for 5 accounts

simulate\_bank\_accounts(5, interest\_rate=3.5)

Output –

Account ID: 1, Current Balance: 1039, Minimum Balances (6 months): [1288, 2073, 2255, 1378, 1030, 1030]

Interest earned over 6 months: 158.44

Account ID: 2, Current Balance: 7604, Minimum Balances (6 months): [7601, 8064, 7891, 8112, 7657, 7192]

Interest earned over 6 months: 814.05

Account ID: 3, Current Balance: 4941, Minimum Balances (6 months): [3566, 4232, 4228, 4044, 4044, 4177]

Interest earned over 6 months: 425.09

Account ID: 4, Current Balance: 8722, Minimum Balances (6 months): [5905, 5733, 5937, 7870, 8996, 8722]

Interest earned over 6 months: 755.35

Account ID: 5, Current Balance: 7903, Minimum Balances (6 months): [4076, 3752, 3943, 4651, 6060, 6789]

Interest earned over 6 months: 512.24

1. **Conclusion**

The Interest Calculation System ensures accurate calculation of interest based on the minimum monthly balances and can handle multiple accounts. The system efficiently records transactions and calculates interest for any given period, following the bank's interest policies.