

***Software Construction & Development***

***Project: Bank ATM Account***

***Prepared By:***

***Talha Jabbar 196***

***Ameer Hamza 060***

***Zohaib Shakeel 090***

***Ayesha Gulraiz 058***

***Irmana Ahmed 002***

***Submitted To: Sir Ali Haider***

***Section: B***

***Session: Fall-2020***

***Features***

* *View Current Account Balance*
* *Current Account Withdraw*
* *Current Account Deposit*
* *Transfer funds from Saving to Current Account*
* *View Saving Account Balance*
* *Saving Account Withdraw*
* *Saving Account Deposit*
* *Transfer funds from Current to Saving Account*
* *Exit*

***Modules and Tools***

* *Import python module “time”*
* *Import python module “random”*
* *Import “ATM” file in “main” file*
* *Classes and Objects*
* *Functions*
* *While-Loop & If-else statements*
* *Unit Testing*
* *Exception Handling*

***Functions***

*1 - View Current Balance 2 - Withdraw Current*

*3 - Deposit Current 4 - Transfer from Saving to Current Account*

*5 - View Savings Balance 6 - Withdraw Savings*

*7 - Deposit Savings 8 - Transfer from Current to Saving Account*

*9 - Exit*

***Analysis***

* The project contains 2 files, one with the name ATM in which we created class named Account and defined all function needed in it.
* The other is the main file in which we have perform operations on the pre-defined functions.
* The ATM file is imported in the main file and objects are created in this file to access methods of functions.
* Test Cases and exception handling are performed on the functions.

***Implementation of Algorithm***

***ATM.py File:***

class Account:

    # Construct an Account object

    def \_\_init\_\_(self, id, currentBalance=0, savingsBalance=0, annualInterestRateSavings=3.4):

        self.id = id

        self.currentBalance = currentBalance

        self.savingsBalance = savingsBalance

        self.annualInterestRateSavings = annualInterestRateSavings

    def getId(self):

        return self.id

    def currentAccountBalance(self):

        return self.currentBalance

    def withdrawCurrentAccount(self, amount):

        self.currentBalance -= amount

    def depositCurrentAccount(self, amount):

        self.currentBalance += amount

    def transferCurrentAccount(self, amount):

        self.currentBalance += amount

        self.savingsBalance -= amount

    def savingsAccountBalance(self):

        return self.savingsBalance

    def withdrawSavingsAccount(self, amount):

        self.savingsBalance -= amount

    def depositSavingsAccount(self, amount):

        self.savingsBalance += amount

    def transferSavingsAccount(self, amount):

        self.savingsBalance += amount

        self.currentBalance -= amount

    def savingsAccountMonthlyInterest(self):

        return self.savingsBalance \* self.savingsAccountMonthlyInterest()

    def savingsAccountAnnualInterestRate(self):

        return self.annualInterestRateSavings

    def savingsAccountMonthlyInterestRate(self):

        return self.annualInterestRateSavings / 12

***main.py File***

import random

import time

from ATM import Account

def main():

    # Creating accounts

    accounts = []

    for i in range(1000, 9999):

        account = Account(i, 0)

        accounts.append(account)

    while True:

        # Reading id from user

        print("Welcome to International Bank.")

        id = int(input("\nEnter 4-digit account pin: "))

        # Loop till id is valid

        while id < 1000 or id > 9999:

           id = int(input("\nInvalid Id.. Re-enter: "))

        # Iterating over interface

        while True:

            # Printing menu

            print("\nHow can we help you today?")

            print("""\n1 - View Current Balance \t\t2 - Withdraw Current \n3 - Deposit Current \t\t\t4 - Transfer from Saving to Current Account

                    \n5 - View Savings Balance \t\t6 - Withdraw Savings \n7 - Deposit Savings \t\t\t8 - Transfer from Curren to Saving Account

                    \n9 - Exit """)

            # Reading selection

            selection = int(input("\nEnter your numerical selection: "))

            # Getting account object

            for acc in accounts:

                # Comparing account id

                if acc.getId() == id:

                    accountObj = acc

                    break

            # View Current Balance

            if selection == 1:

                # Printing balance

                print(

                    f"\nCurrent account balance: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                time.sleep(2)

            # Current Withdraw

            elif selection == 2:

                # Reading amount

                amt = float(

                    input("\nEnter amount to withdraw from current: "))

                ver\_current\_withdraw = input(

                    f"Is ${format(amt, '.2f')} the correct amount to withdraw, Yes or No ? ")

                ver\_current\_withdraw = ver\_current\_withdraw.upper()

                if ver\_current\_withdraw == "YES":

                    print("\nVerified current account withdraw.")

                    time.sleep(2)

                else:

                    break

                if amt < accountObj.currentAccountBalance():

                    # Calling withdraw method

                    accountObj.withdrawCurrentAccount(amt)

                    # Printing updated balance

                    print(

                        f"Updated current account balance: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    time.sleep(2)

                else:

                    print(

                        f"\nYour current account balance is less than withdrawl amount: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    time.sleep(2)

            # Deposit

            elif selection == 3:

                # Reading amount

                amt = float(input("\nEnter amount to deposit in current: "))

                ver\_current\_deposit = input(

                    f"Is ${format(amt, '.2f')} the correct amount to deposit, Yes or No ? ")

                ver\_current\_deposit = ver\_current\_deposit.upper()

                if ver\_current\_deposit == "YES":

                    # Calling deposit method

                    print("\nVerified current account deposit.")

                    time.sleep(2)

                    accountObj.depositCurrentAccount(amt)

                    # Printing updated balance

                    print(

                        f"\nUpdated current account balance: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    time.sleep(2)

                else:

                    break

            #Transfer Savings Account to Current Account

            elif selection == 4:

                amt = float(

                    input("\nEnter amount to transfer from savings account to current: "))

                ver\_current\_transfer = input(

                    f"Is ${format(amt, '.2f')} the correct amount to transfer, Yes or No ? ")

                ver\_current\_transfer = ver\_current\_transfer.upper()

                if ver\_current\_transfer == "YES":

                    print("Verified savings account transfer to current account.")

                    time.sleep(2)

                else:

                    time.sleep(2)

                    break

                if amt < accountObj.savingsAccountBalance():

                    # Calling transfer method

                    accountObj.transferCurrentAccount(amt)

                    # Printing updated balance

                    print(

                        f"Updated current account balance: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    print(

                        f"Updated savings account balance: ${format(accountObj.savingsAccountBalance(), '.2f')} ")

                    time.sleep(2)

                else:

                    print(

                        f"\nYour savings account balance is less than transfer amount: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    time.sleep(2)

            # View Saving Balance

            elif selection == 5:

                # Printing balance

                print(

                    f"Savings account balance: ${format(accountObj.savingsAccountBalance(), '.2f')} ")

                time.sleep(2)

            # Savings Withdraw

            elif selection == 6:

                # Reading amount

                amt = float(input("\nEnter amount to withdraw from savings: "))

                ver\_savings\_withdraw = input(

                    f"Is ${format(amt, '.2f')} the correct amount to withdraw, Yes or No ? ")

                ver\_savings\_withdraw = ver\_savings\_withdraw.upper()

                if ver\_savings\_withdraw == "YES":

                    print("\nVerified savings account withdraw.")

                    time.sleep(2)

                else:

                    time.sleep(2)

                    break

                if amt < accountObj.savingsAccountBalance():

                    # Calling withdraw method

                    accountObj.withdrawSavingsAccount(amt)

                    # Printing updated balance

                    print(

                        f"Updated current account balance: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    print(

                        f"Updated savings account balance: ${format(accountObj.savingsAccountBalance(), '.2f')} ")

                    time.sleep(2)

                else:

                    print(

                        f"\nYour savings account balance is less than withdrawl amount: ${format(accountObj.savingsAccountBalance(), '.2f')} ")

                    time.sleep(2)

            # Savings Account Deposit

            elif selection == 7:

                # Reading amount

                amt = float(input("\nEnter amount to deposit in savings: "))

                ver\_savings\_deposit = input(

                    f"Is ${format(amt, '.2f')} the correct amount to deposit, Yes or No ? ")

                ver\_savings\_deposit = ver\_savings\_deposit.upper()

                if ver\_savings\_deposit == "YES":

                    print("\nVerified savings account deposit.")

                    time.sleep(2)

                    # Calling deposit method

                    accountObj.depositSavingsAccount(amt)

                    # Printing updated balance

                    print(

                        f"Updated savings account balance: ${format(accountObj.savingsAccountBalance(), '.2f')} ")

                    time.sleep(2)

                else:

                    break

                    time.sleep(2)

            #Transfer Current Account to Savings Account

            elif selection == 8:

                amt = float(

                    input("\nEnter amount to transfer from current account to savings account: "))

                ver\_savings\_transfer = input(

                    f"Is ${format(amt, '.2f')} the correct amount to transfer, Yes or No ? ")

                ver\_savings\_transfer = ver\_savings\_transfer.upper()

                if ver\_savings\_transfer == "YES":

                    print("Verified current account transfer to savings account.")

                    time.sleep(2)

                else:

                    break

                if amt < accountObj.checkingAccountBalance():

                    # Calling withdraw method

                    accountObj.transferSavingsAccount(amt)

                    # Printing updated balance

                    print(

                        f"Updated current account balance: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    print(

                        f"Updated savings account balance: ${format(accountObj.savingsAccountBalance(), '.2f')} ")

                    time.sleep(2)

                else:

                    print(

                        f"\nYour current account balance is less than transfer amount: ${format(accountObj.currentAccountBalance(), '.2f')} ")

                    time.sleep(2)

            elif selection == 9:

                print("\nTransaction is now complete.")

                print("Transaction number: ", random.randint(10000, 1000000))

                print("Current Interest Rate: ",

                      accountObj.annualInterestRateSavings)

                print("Monthly Interest Rate: ",

                      accountObj.annualInterestRateSavings / 12)

                print("Thanks for choosing us as your bank")

                exit()

            # Any other choice

            else:

                print("\nThat's an invalid choice.")

                break

# Main function

main()

***Unit Testing***

import unittest

from ATM import Account

class TestAccount(unittest.TestCase):

    def setUp(self):

        self.account = Account(1, currentBalance=100, savingsBalance=200, annualInterestRateSavings=3.4)

    def test\_getId(self):

        self.assertEqual(self.account.getId(), 1)

    def test\_currentAccountBalance(self):

        self.assertEqual(self.account.currentAccountBalance(), 100)

    def test\_withdrawCurrentAccount(self):

        self.account.withdrawCurrentAccount(50)

        self.assertEqual(self.account.currentAccountBalance(), 50)

    def test\_depositCurrentAccount(self):

        self.account.depositCurrentAccount(50)

        self.assertEqual(self.account.currentAccountBalance(), 150)

    def test\_transferCurrentAccount(self):

        self.account.transferCurrentAccount(50)

        self.assertEqual(self.account.currentAccountBalance(), 150)

        self.assertEqual(self.account.savingsAccountBalance(), 150)

    def test\_savingsAccountBalance(self):

        self.assertEqual(self.account.savingsAccountBalance(), 200)

    def test\_withdrawSavingsAccount(self):

        self.account.withdrawSavingsAccount(50)

        self.assertEqual(self.account.savingsAccountBalance(), 150)

    def test\_depositSavingsAccount(self):

        self.account.depositSavingsAccount(50)

        self.assertEqual(self.account.savingsAccountBalance(), 250)

    def test\_transferSavingsAccount(self):

        self.account.transferSavingsAccount(50)

        self.assertEqual(self.account.savingsAccountBalance(), 250)

        self.assertEqual(self.account.currentAccountBalance(), 150)

    def test\_savingsAccountMonthlyInterest(self):

        self.assertAlmostEqual(self.account.savingsAccountMonthlyInterest(), 6.876712328767123, places=14)

    def test\_savingsAccountAnnualInterestRate(self):

        self.assertEqual(self.account.savingsAccountAnnualInterestRate(), 3.4)

    def test\_savingsAccountMonthlyInterestRate(self):

        self.assertAlmostEqual(self.account.savingsAccountMonthlyInterestRate(), 0.2833333333333333, places=14)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

***Exception Handling***

class Account:

    # Construct an Account object

    def \_\_init\_\_(self, id, currentBalance=0, savingsBalance=0, annualInterestRateSavings=3.4):

        self.id = id

        self.currentBalance = currentBalance

        self.savingsBalance = savingsBalance

        self.annualInterestRateSavings = annualInterestRateSavings

    def getId(self):

        return self.id

    def currentAccountBalance(self):

        return self.currentBalance

    def withdrawCurrentAccount(self, amount):

        if amount > self.currentBalance:

            raise ValueError("Insufficient balance in the current account")

        self.currentBalance -= amount

    def depositCurrentAccount(self, amount):

        if amount <= 0:

            raise ValueError("Amount to deposit must be positive")

        self.currentBalance += amount

    def transferCurrentAccount(self, amount):

        if amount > self.savingsBalance:

            raise ValueError("Insufficient balance in the savings account")

        self.currentBalance += amount

        self.savingsBalance -= amount

    def savingsAccountBalance(self):

        return self.savingsBalance

    def withdrawSavingsAccount(self, amount):

        if amount > self.savingsBalance:

            raise ValueError("Insufficient balance in the savings account")

        self.savingsBalance -= amount

    def depositSavingsAccount(self, amount):

        if amount <= 0:

            raise ValueError("Amount to deposit must be positive")

        self.savingsBalance += amount

    def transferSavingsAccount(self, amount):

        if amount > self.currentBalance:

            raise ValueError("Insufficient balance in the current account")

        self.savingsBalance += amount

        self.currentBalance -= amount

    def savingsAccountMonthlyInterest(self):

        return self.savingsBalance \* self.savingsAccountMonthlyInterest()

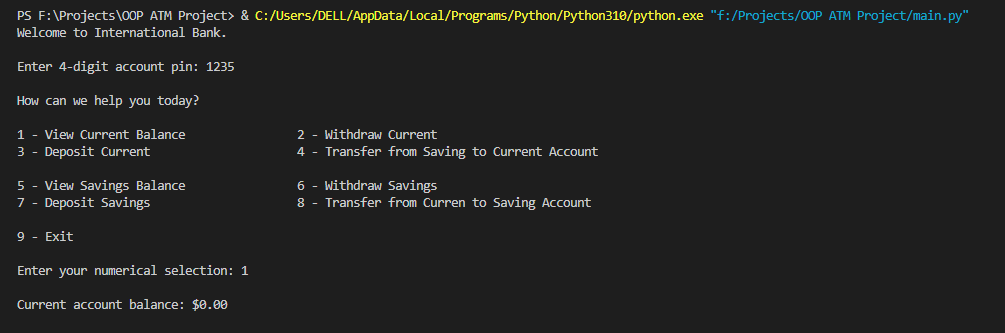
    def savingsAccountAnnualInterestRate(self):

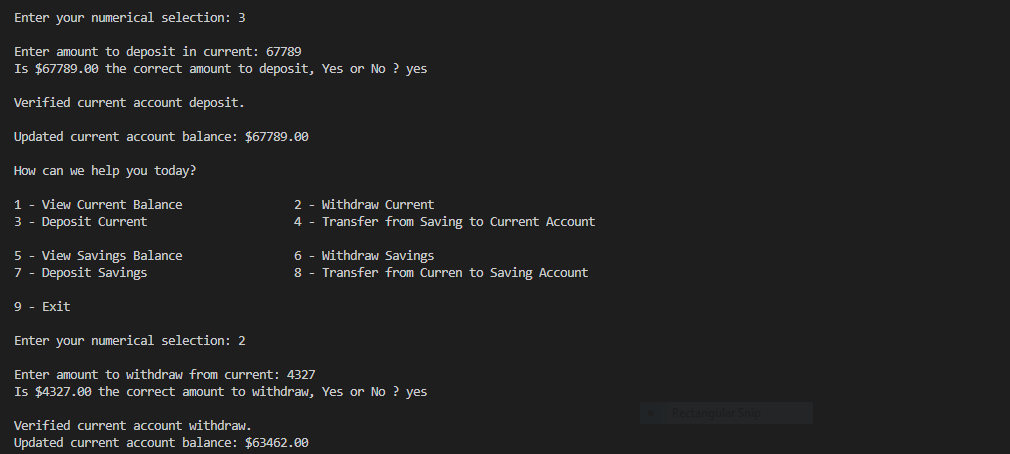
        return self.annualInterestRateSavings

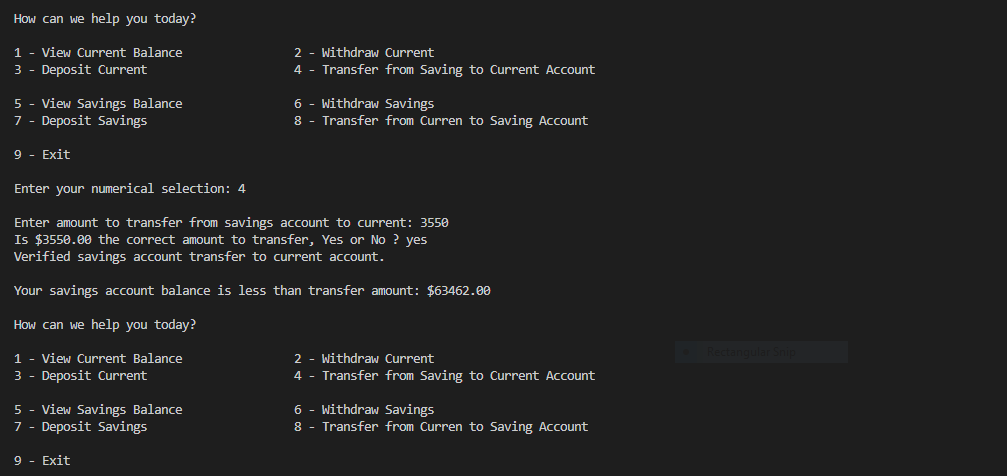
    def savingsAccountMonthlyInterestRate(self):

        return self.annualInterestRateSavings / 12

***Output:***







\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_