FI Concession Examination DEV401I

Examiner: Masombuka KT

Instructions:

1. Answer all the questions
2. Submit the following for every question:
   2.1. A solution in pdf format.
   2.2. Test run and output screenshots.
   2.3. Corresponding .py files.

Total: 60

Appendix A shows a `Bank` and the `ATM` class. Study the two classes and answer the following questions:

1 Question 1(Classes, 10 Marks)

1.1 The `str` method of the `Bank` class returns a string containing the accounts in random order. Design and implement a change that cause the accounts to be placed in the string by order of name. (Hint: you will also have to define some methods in the `SavingsAccount` class.)

1.2 The ATM program allows a user an indefinite number of attempts to log in. Fix the program so that it displays a message that the police will be called after a user has had three successive failures. The program should also shut down the bank when this happens.

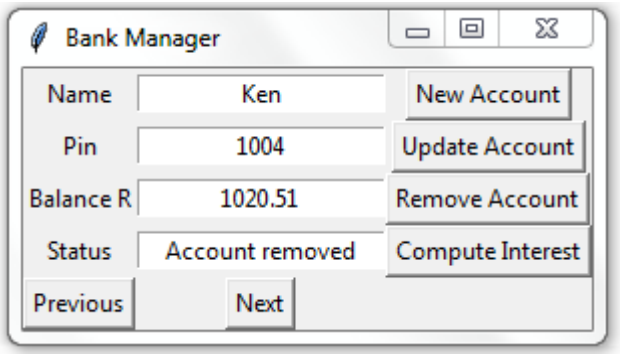
2 Question 2 (Client-server applications, 25 Marks)

Convert the ATM application presented in Appendix A to a networked application. The client manages the user interface, whereas the server handles transactions with the bank.

3 Question 3 (GUI, 25 Marks)

Write a GUI-based program that allows a bank manager to view and manipulate the accounts in a bank. The window should display the information for the currently selected account editable entry fields. Command buttons should allow the user to navigate to the next account and the previous account, assuming that the accounts are ordered by a PIN. Add a method `getPins()` to the `Bank` class. This method should build and return a sorted list of the PINs in the bank. The GUI should use this method to help locate the first account, next account, and previous account. Command buttons should also allow the user to remove an account, save an account's edited information, and add a new account. When a new account is added, the entry fields should be reset to default values, and the `Update Account` button should create the account.

Sample screen short



Appendix A

ATM Class

```
"""
File: atm.py

This module defines the ATM class and its application.

To test, launch from Idle and run

>>> createBank(5)
>>> main()

Can be modified to run as a script after a bank has been saved.
"""
from bank import Bank, SavingsAccount

class ATM(object):
    """This class represents terminal-based ATM transactions."""

    SECRET_CODE = "CloseItDown"

    def __init__(self, bank):
        self._account = None
        self._bank = bank
        self._methods = {}          # Jump table for commands
        self._methods["1"] = self._getBalance
        self._methods["2"] = self._deposit
        self._methods["3"] = self._withdraw
        self._methods["4"] = self._quit

    def run(self):
        """Logs in users and processes their accounts."""
        while True:
            name = input("Enter your name: ")
            if name == ATM.SECRET_CODE:
                break
            pin = input("Enter your PIN: ")
            self._account = self._bank.get(pin)
            if self._account == None:
                print("Error, unrecognized PIN")
            elif self._account.getName() != name:
                print("Error, unrecognized name")
                self._account = None
            else:
                self._processAccount()

    def _processAccount(self):
        """A menu-driven command processor for a user."""
        while True:
            print("1  View your balance")
            print("2  Make a deposit")
            print("3  Make a withdrawal")
            print("4  Quit\n")
            number = input("Enter a number: ")
```

```
            theMethod = self._methods.get(number, None)
            if theMethod == None:
                print("Unrecognized number")
            else:
                theMethod()
                if self._account == None:
                    break

    def _getBalance(self):
        print("Your balance is $", self._account.getBalance())

    def _deposit(self):
        amount = float(input("Enter the amount to deposit: "))
        self._account.deposit(amount)

    def _withdraw(self):
        amount = float(input("Enter the amount to withdraw: "))
        message = self._account.withdraw(amount)
        if message:
            print(message)

    def _quit(self):
        self._bank.save()
        self._account = None
        print("Have a nice day!")

# Top-level functions
def main():
    """Instantiate a Bank and an ATM and run it."""
    bank = Bank("bank.dat")
    atm = ATM(bank)
    atm.run()

def createBank(number = 0):
    """Saves a bank with the specified number of accounts.
    Used during testing."""
    bank = Bank()
    for i in range(number):
        bank.add(SavingsAccount('Name' + str(i + 1),
                                str(1000 + i),
                                100.00))
    bank.save("bank.dat")
```

Bank Class

```
"""
File: bank.py

This module defines the SavingsAccount and Bank classes.
"""
import pickle

class SavingsAccount(object)
    """This class represents a savings account
    with the owner's name, PIN, and balance."""
    RATE = 0.02

    def __init__(self, name, pin, balance = 0.0):
        self._name = name
        self._pin = pin
        self._balance = balance

    def __str__(self):
        result =  'Name:    ' + self._name + '\n'
        result += 'PIN:     ' + self._pin + '\n'
        result += 'Balance: ' + str(self._balance)
        return result

    def getBalance(self):
        return self._balance

    def getName(self):
        return self._name

    def getPin(self):
        return self._pin

    def deposit(self, amount):
        """Deposits the given amount."""
        self._balance += amount
        return self._balance

    def withdraw(self, amount):
        """Withdraws the given amount.
        Returns None if successful, or an
        error message if unsuccessful."""
        if amount < 0:
            return 'Amount must be >= 0'
        elif self._balance < amount:
            return 'Insufficient funds'
        else:
            self._balance -= amount
            return None

    def computeInterest(self):
        """Computes, deposits, and returns the interest."""
        interest = self._balance * SavingsAccount.RATE
        self.deposit(interest)
```

```
        return interest

class Bank(object):
    """This class represents a bank as a dictionary of
    accounts.  An optional file name is also associated
    with the bank, to allow transfer of accounts to and
    from permanent file storage."""

    def __init__(self, fileName = None):
        """Creates a new dictionary to hold the accounts.
        If a file name is provided, loads the accounts from
        a file of pickled accounts."""
        self._accounts = {}
        self.fileName = fileName
        if fileName != None:
            fileObj = open(fileName, 'rb')
            while True:
                try:
                    account = pickle.load(fileObj)
                    self.add(account)
                except(EOFError):
                    fileObj.close()
                    break

    def add(self, account):
        """Inserts an account using its PIN as a key."""
        self._accounts[account.getPin()] = account

    def remove(self, pin):
        return self._accounts.pop(pin, None)

    def get(self, pin):
        return self._accounts.get(pin, None)

    def computeInterest(self):
        """Computes interest for each account and
        returns the total."""
        total = 0
        for account in self._accounts.values():
            total += account.computeInterest()
        return total

    def __str__(self):
        """Return the string rep of the entire bank."""
        return '\n'.join(map(str, self._accounts.values()))

    def save(self, fileName = None):
        """Saves pickled accounts to a file.  The parameter
        allows the user to change file names."""
        if fileName != None:
            self.fileName = fileName
        elif self.fileName == None:
            return
        fileObj = open(self.fileName, 'wb')
        for account in self._accounts.values():
```

```
            pickle.dump(account, fileObj)
        fileObj.close()

def testBank(number = 0):
    """Returns a bank with the specified number of accounts and/or
    the accounts loaded from the specified file name."""
    bank = Bank()
    for i in range(number):
        bank.add(SavingsAccount('Name' + str(i + 1),
                                str(1000 + i),

        100.00))
    return bank
```