CET2011 - Programming Paradigms: Advanced Python - Practicum 01

Important Information

- To get familiarize with the object oriented principles and their implementation in Python.
- Programming using Python scripts with VS Code or your preferred IDE.
- This practicum is worth 2 practicums therefore it will have an overall mark of 200.

Allowable libraries:

- Python `random` library
- Python `datatime` library
- Python `abc` library

Deliverables

- A zip file of all completed scripts using the given filenames as defined in this document the format `CET2011_P01_<student_name>.zip`. (e.g. `CET2011_P01_John_Doe.zip`)
- Your assessment will depend on the proper functioning of your classes as a whole program. **Failure to do so will be graded ZERO.**

Background

In this practicum, we will be **simulating of a scaled down version of an Automated Teller Machine (ATM)** implemented using Object Orientation Principles. To scale down the complexities of a real world ATM, the practicum will be limited to this scenario:

> - Bank XYZ maintains many ATMs. Each ATM can only perform 4 operations: Check balance, Withdrawals, Transfers and Quit.
> - ATM transactions can only start when the user's PIN is entered correctly. If the PIN is invalid, the card is returned.
> - Each bank can has only 1 type of ATM card and each ATM card has access to both a customer's Savings and Current account (if any).
> - Each customer of the bank by default has a savings account.
> - Customers of Bank XYZ are allowed to own only 1 ATM card.
> - Withdrawals operations are done either from the customer's Savings or Current account.
> - Transfers operations are done either from Savings to Current account and vice versa from the same customer or from an account of Customer A to an account of Customer B (both customers must be account holders of Bank XYZ).
> - Transfer transactions must not go through if the transfer account number is not existent or transferee has insufficient funds for transfer.

Assumptions For Testing

1. Each customer object has either a Savings or Current account or both from when they personally opened the accounts at the physical bank therefore do not create any customer objects with more than 2 account types.
2. There is only 1 bank and 1 Automated Teller Machine for testing to keep the scale small. All other objects such as customer, ATM cards and accounts are allowed to have multiples.
3. Each customer can only own 1 ATM card.

Implementation

Please read the following instructions very carefully as your task is to implement the simulation of this scaled down version of an ATM. To properly simulate the ATM, there will be **8 Python scripts** that you will need to create: Account, ATM, ATM Transaction, Bank, Customer, ATM Card, Main App and Additional Exceptions. You are to implement each script based on the instructions defined for each script file. **The script file names are the headings of each section.**

additional_exceptions.py (10 marks)

This script will house all the special exceptions that is created for this simulation. This script will have 5 classes: `InvalidAccount`, `AccountNotFound`, `InsufficientFunds`, `InvalidATMCard` and `InvalidPinNumber`. All are children of the `Exception` class.

customer.py (10 marks)

This script represents the `Customer` object therefore it has class called `Customer` and its properties are:
Attributes:
They are all private variables

- name - customer's name
- address - customer's address
- date of birth - customer's date of birth in `datetime` format (DD-MMM-YYYY)
- list of accounts - a list of accounts a customer can have, maximum is 2. Savings and/or Current account/s.

Methods:

- `__init()__` - constructor that initializes the customer's name, address and date of birth from input parameters and leave the lists as empty lists.
- `owns()` - accepts an `Account` object and adds it to the list.
- `__str__()` - returns string representation of the `Customer` object detailing the name, address and date of birth.
- getters for name and list of accounts

account.py (30 marks)

This script is the account object of each customer of the bank. It has 3 classes in it: `Account`, `Savings_Account` and `Current_Account`. `Account` the is parent class of both `Savings_Account` and `Current_Account` classes.

The `Account` class has the following properties:
Attributes:

- account type - protected variable. Values are either `'savings'` or `'current'`
- owner - private variable. A `Customer` object
- balance - private variable. Account balance

Methods:

- `__init()__` - constructor that initializes the account type and owner from the child classes and sets the account balance to zero.
- `check_balance()` - returns the current balance of the account. This doubles up as a getter for the balance attribute.
- 1 setter for the balance attribute and 2 getters for owner and account type
- `debit()` - accepts an amount, checks it then subtracts it from the balance. Returns `True` if there is enough funds for the debit otherwise, raise an `InsufficientFunds` exception (the balance remains unchanged).
- `credit()` - accepts an amount, checks it then adds it to the balance.
- `validate_amount()` - private method, accepts a monetary amount (be it of type `integer` or `float` or `string`), converts it to type `float` . Raise a `ValueError` if the monetary amount is <= 0 or the amount has more than 2 decimal places. Return `True` otherwise

The `Savings_Account` and `Current_Account` classes inherit the parent class `Account`, and both of them have the following properties:
Attributes:

- account number - private variable. The account number of the respective account
- owner - a `Customer` object
- account type - value is `'savings'` for `Savings_Account` account or `'current'` for `Current_Account` account.

Methods:

- `__init()__` - constructor that initializes the account number and owner from input parameters and initializes the `Account` parent class with the respective account type (either `'savings'` or `'current'`) and owner (`Customer` object).
- getters for the account number variable

bank.py (25 marks)

This represents the bank therefore it has a class called `Bank`. Similar to the real world, the bank will know the customer's details, which customer has an ATM card and ATMs are currently active. The bank must meet the following properties:
Attributes:

- code - private variable, a bank code
- address - private variable, bank's address
- list of ATM - a list of `ATM` objects
- list of customers - a list of `Customer` records
- list of atm cards - a list of `ATM_card` objects

Methods:

- `__init()__` - constructor that initializes the bank's code and address from input parameters and initializes all the lists as empty lists.
- `add_customer()` - the bank must be able to add a `Customer` object and their pin number. Each customer's record is stored as a `dictionary` consisting of the keys listed below. Each customer's record must be stored in the list of customers. How you name your keys is up to you.
    - a **unique** customer id that is a combination of the customer's name and a **random** string of 4 digit numbers. Eg: Tom4851

    - the customer object
    - the customer pin number
- `manages()` - accepts an `atm_card` object and stores it in the list of atm cards.
- `maintains()` - accepts an `ATM` object and stores it in the list of ATMs.
- `authorize_pin()` - accepts a customer's object and pin number as input parameters, checks that the customer's pin number is valid. Returns a `True` if the pin number is valid otherwise raises a `InvalidPinNumber` exception.
- `get_acct()` - accepts an account number as input parameter and check if the account number is valid. Returns an `Account` object if the account number is valid otherwise raises a `AccountNotFound` exception.

atm_card.py (20 marks)

This represents the ATM card that every customer can own therefore it has a class called `ATM_Card`. Like in the real world, a customer can only own one ATM card per bank. The ATM card is used by the ATM to access the customer's accounts and it has these properties:
Attributes:

- card number - private variable, the ATM card's number
- owned by - private variable, a `Customer` object

Methods:

- `__init()__` - constructor that initializes the ATM card's number and owner via input parameters.
- `get_acct_types()` - returns a list of the account types available for the customer.
- `access()` - accepts an account type and returns the `Account` object of the customer.
- `__str__()` - returns a string representation of the ATM card detailing the card's number and owner's name.
- getters for the private variables card number and owned by

atm_transaction.py (35 marks)

This represents the transactions that the ATM can perform therefore it has 3 classes called `ATM_Transaction`, `Withdrawal` and `Transfer`. `ATM_Transaction` is the parent class of `Withdrawal` and `Transfer`.

The `ATM_Transaction` class is an abstract class and has the following properties:
Attributes:

- transaction id - to keep track of all transactions that has taken place
- date - stores the date and time of current transaction, type `datetime`.
- transaction type - protected variable, type of transaction, either `'withdrawal'` or `'transfer'`.
- amount - protected variable, the monetary amount used for the transaction.

Methods:

- `__init()__` - constructor that initializes the transaction's date & time (timestamp), transaction type and amount (default value of 0) via input parameters. In addition, increment the transaction id by 1 (Hint: declare transaction id as a static variable that initially starts from 0).
- `update()` - accepts the customer's `Account` object, amount (monetary value) and a transfer account object (default value is `None`) via input parameters. Implementation to be done by

the child classes. (Hint: @abstractmethod)

The `Withdrawal` class has the following properties:
Attributes:

- None

Methods:

- `__init()__` - constructor that initializes the transaction's date & time (timestamp) by using one of the `datatime` functions and transaction type set to `'withdrawal'`. The attribute amount is initialized via input parameters.
- `withdraw()` - accepts an `Account` object and calls the `update()` function with the appropriate parameters. Returns the status of the `update()` function.
- `update()` - function that updates the customer's `Account` object with the funds to be withdrawn. Returns the status of the transaction.

The `Transfer` class has the following properties:
Attributes:

- None

Methods:

- `__init()__` - constructor that initializes the transaction's date & time (timestamp) by using one of the `datatime` functions and transaction type set to `'transfer'`. The attribute amount via input parameters.
- `update()` - function that updates the customer's `Account` object with funds for the transfer process. Returns `True` if successful.

atm.py (35 marks)

This represents the Automated Teller Machine itself and has a class called `ATM`. This class will need access to the `Withdrawal` and `Transfer` classes from the `atm_transction.py`. Its properties are:
Attributes:

- location - location/address of the ATM
- managed by - the bank that manages the ATM
- current card - private variable, the current ATM card that is "inserted" by the user of the ATM.

Methods:

- `__init()__` - constructor that initializes the ATM's location and managed by attributes via input parameters. The attribute current card is set to `None`.
- accessor methods for current card attribute.
- `transactions()` - accepts transaction type, amount, account type and transfer account number which has a default value of `None` via input parameters. Process the requested transaction according to the transaction type. Returns the status of the respective transactions or raises an exception indicating whether or not the transactions were successful.
    - for `Transfer` transactions, if the sender and receiver are the same account numbers, raise a `InvalidAccount` exception.
- `check_accts()` - checks if the user has 1 or 2 accounts. Returns `True` if there is 2 otherwise returns `False`.

- `check_pin()` - accepts a user's pin number and checks with the bank if it is valid. Returns the status of the check from the bank.
- `check_card()` - accepts a ATM card number and checks with the bank if the card is a valid card. If it is valid, the `current card` attribute is set to this ATM card object and returns `True` but if the card is invalid, raise an `InvalidATMCard` exception.
- `show_balance()` - accepts an `Account` type and returns a string message detailing the current balance of the requested account of that customer.

main_app.py (35 marks)

This represents the ATM and the menu system of your ATM. You are to use 2 or more functions to simulate your ATM and the ATM menu system. **All exceptions are to be handled by the functions within this script.**

Data used for testing your code is to be stored in another Python script. You not required to submit your test data.

Part 1 - The ATM itself (simulation menu)
Like in the real world, the ATM is *constantly on* so long as there is power supplied to the machine. Therefore you are to create a function called `atm_app` that accepts an ATM object and has the following 2 options:

- **Insert ATM card** - allows the user to enter an ATM card number. If the ATM card number and pin number is valid, it moves on to the ATM Menu (part 2) otherwise it must show a message indicating which is invalid (either the card or the pin).
- **Quit Simulation** - Exits the program and ends the simulation.

Part 2 - The ATM Menu
Create another function called `atm_menu_sys` that handles the actual ATM functionalities. This function accepts the same ATM object as in part 1. This menu system must be able to perform the following 4 functions:

- **Check balance** - allows the user to check their `Account` balance. The ATM is to show a choice to the user if the user has 2 accounts otherwise savings account is the default
    - after a valid option is chosen and balance shown, the user it able to continue with the other functions available from the ATM menu.
    - if the option is invalid, it is to exit the current choice selection and return to the ATM menu.
- **Withdraw Funds** - allows the user to withdraw funds from their `Account`, show the account balance then "returns" card to the user and "exits" to the simulation menu.
    - a maximum of 3 tries is given to the user to enter a valid withdrawal amount, after this limit is hit, return to the ATM menu.
    - if the account has insufficient funds, a message is shown and "returns" to the ATM menu.
- **Transfer Funds** - allows the user to transfer funds either within their `Savings` to `Current` accounts and vice versa or transfer funds to another customer of Bank XZY.
    - If the transfer is successful, show the account balance of the user then "return" their card and "exits" to the simulation menu.
    - If the transfer is unsuccessful because of invalid account number or mistake when entering the values, the user has a maximum of 3 tries before the function "exits" to the ATM menu.

    - if the transfer is unsuccessful because of insufficient funds, show a message then "exits" to the ATM menu.
    - you will need the receiver's account number and amount of funds to begin the processing of the transfer transaction.
    - proper messages must be shown to the user why the transfer has failed
- **Return card** - "Returns" the card to the user and "exits" to the simulation menu in part 1.

Note that throughout the function of your ATM simulation, properly formatted messages must be shown to the users indicating whether or not their transactions are successful or not. Failure to do so will result in a penalty.

Test Cases

Below are **some** test cases that demonstrates the functions of the ATM simulation. You may consider using the following test codes to verify your .py files.

```
from bank import Bank  # from bank.py import Bank class
from atm import ATM
from account import Savings_Account, Current_Account
from atm_card import ATM_Card
from customer import Customer
from main_app import atm_app # from main_app.py import atm_app function

# set up the bank objects.
bank_01 = Bank("POSB01", "51 Sunset lane")
# set up atm object
atm_01 = ATM("tannery lane 78", bank_01)
# bank_01 has 1 atm
bank_01.maintains(atm_01)

# set up customer object.
# customer 1 has 1 savings acct with 8k and 1 current acct with 5k with bank_01
cust_01 = Customer('Tom', 'abc lane', '15-may-1987')
save_01 = Savings_Account("458-96252", cust_01)
curr_01 = Current_Account("124-87592", cust_01)
cust_01.owns(save_01)
cust_01.owns(curr_01)
# adding money to acct
save_01.credit(8000)
curr_01.credit(5000)
# ATM card
dc_01 = ATM_Card("589-5955-874-6941", cust_01)
bank_01.manages(dc_01)
bank_01.add_customer(cust_01, "1234")


# customer 2 has 1 savings acct with 6k with bank_01
cust_02 = Customer('Kim', 'blank lane', '05-dec-1999')
save_02 = Savings_Account("874-15268", cust_02)
cust_02.owns(save_02)
# adding money to acct
save_02.credit(6000)
# ATM card
dc_02 = ATM_Card("847-9521-248-8451", cust_02)
```

```
bank_01.manages(dc_02)
bank_01.add_customer(cust_02, "8745")

#print customer info
print(cust_01)
print(dc_01)
print()
print(cust_02)
print(dc_02)

#start running the ATM using the atm_app function you created in main_app.py
atm_app(atm_01)
```

Test Case 1: Bad debit card

```
Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM card number:54813-84545
Invalid Card. Please take your card.
```

Test Case 2: Checking of Account Balance

```
Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM card number:589-5955-874-6941
Enter a Pin Number: 1234

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 1

Choose Account:
1. Current Account
2. Savings Account
Enter an account option: 2
Your savings account has a balance of $8000.00.
```

Test Case 3: Withdrawing Funds

```
Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 2

Choose Account:
1. Current Account
2. Savings Account
Enter an account option: 2
Enter an amount to withdraw: 2000
Card Returned
Your savings account has a balance of $6000.00.
```

Test Case 4: Transfer of funds between 2 accounts from the same user

```
Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM card number:589-5955-874-6941
Enter a Pin Number: 1234

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 3

Choose Account:
1. Current Account
2. Savings Account
Enter an account option: 1
Enter the account number to transfer funds to: 458-96252
Enter the amount to transfer: 500
Card Returned
Your current account has a balance of $4500.00.

Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM card number:589-5955-874-6941
Enter a Pin Number: 1234

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
```

```
Enter a transaction option: 1

Choose Account:
1. Current Account
2. Savings Account
Enter an account option: 1
Your current account has a balance of $4500.00.

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 1

Choose Account:
1. Current Account
2. Savings Account
Enter an account option: 2
Your savings account has a balance of $8500.00.
```

Test Case 5: Transfer of funds between 2 accounts from different users

```
Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM number:847-9521-248-8451
Enter a Pin Number: 8745

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 1
Your savings account has a balance of $6000.00.

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 4

Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM number:589-5955-874-6941
Enter a Pin Number: 1234

Avaliable options:
1. Check balance
2. Withdraw Funds
```

```
3. Transfer Funds
4. Return Card
Enter a transaction option: 3

Choose Account:
1. Current Account
2. Savings Account
Enter an account option: 1
Enter the account number to transfer funds to: 874-15268
Enter the amount to transfer: 1500
Card Returned
Your current account has a balance of $3500.00.

Avaliable options:
1. Insert ATM card
2. Quit Simulation
Enter an option:1
Enter a ATM number:847-9521-248-8451
Enter a Pin Number: 8745

Avaliable options:
1. Check balance
2. Withdraw Funds
3. Transfer Funds
4. Return Card
Enter a transaction option: 1
Your savings account has a balance of $7500.00.
```