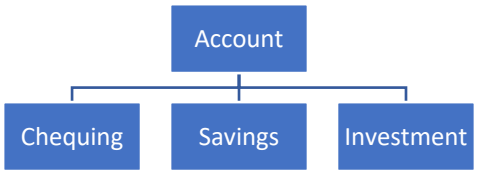
PYTHON II - Assignment 01

Overview:

In this assignment you will be creating an object-oriented system to manage various financial accounts for users. Accounts are of three types: 1) Chequing, 2) Savings, and 3) Investment.



Accounts in our system have to be able to handle various currency and conversions.

Instructions:

In this assignment you will be creating the following Python files:

1. Account.py
2. Chequing.py
3. Savings.py
4. Investment.py
5. AccountTester.py (Program Driver/Tester)

Account.py

All accounts in your system will have an `owner`, `currency`, and `balance`. `owner` is the name of the account owner, `currency` is the abbreviated string representing the currency of the account (i.e. "EUR" for Euro) and `balance` is the value held in the account.

Account should also have a dictionary attribute to store the conversion between the following currencies and the Canadian dollar:

1. USD → [1.34607, $]
2. EUR → [1.51746, €]
3. GBP → [1.70233, £]
4. CNY → [0.0189917, ¥]
5. INR → [0.0178035, ₹]
6. CAD → [1, $]

In terms of the initializer (`__init__`) at minimum when creating an `Account` object, `owner` must be provided. Both `currency` and `balance` are optional, but if not provided should default to `"CAD"` and `0`.

- **Note:** If a currency outside the dictionary attribute is provided display the error message: `"ERROR: Unsupported currency type"`
- `balance` has no restrictions as a negative value indicates an account in overdraft

You also need a `deposit` and `withdraw` method that updates the appropriate attribute of the account.

- Note both methods should only accept positive double values as arguments. If a negative value is provided display the error message:
  `"ERROR: Value for deposit/withdraw is restricted to positive values"`

Add a setter and getter for the `currency` attribute. If an attempt is made to set the currency to an unsupported value display the message: `"ERROR: Unsupported currency type"`

- **IMPORTANT**: If a change in `currency` is made, you need to update `balance` appropriately

In terms of the `__str__` method you should return a string with the following format:

```
Owner: owner
Balance: <Currency Symbol>balance
```

For example, an Account owned by Alex Smith with a balance of 100.32 euros:

```
Owner: Alex Smith
Balance: €100.32
```

- **NOTE:** Ensure balance is displayed only two decimal places

In terms of the `__eq__` equals method two Accounts are equal if their balances are equal in terms of the same currency.

- **HINT:** Convert the two currencies to CAD and then compare for equality

Lastly, you need to implement `__add__` which takes two an additional `Account` object and returns a new `Account` Object with:

- `owner` → `owner` of the calling object
- `balance` → Sum of calling object `balance` and account balance provided as argument
- `currency` → `currency` of calling object

For example:

a1 = Account("Aaron Sarson", "CAD", 1000)
a2 = Account("Jim Cooper", "USD", 1200)
a3 = a1 + a2 #owner = "Aaron Sarson", currency = "CAD", balance=2615.284

Chequing.py

A `Chequing` account has two additional properties to that of `Account`. A `fee` for performing a `withdraw` from the account and an `overdraftLimit`.

In terms of the initializer (`__init__`) at minimum when creating the account `owner, fee,` and `overdraftLimit` must be provided. Both `currency` and `balance` are optional, but if not provided should default to `"CAD"` and `0`.

- **Note:** If a currency outside the dictionary attribute is provided display the error message: `"ERROR: Unsupported currency type"`
- `OverdraftLimit` should be represented as a positive value
- `balance` cannot exceed `overdraftLimit.` If this happens display the message `"ERROR: Overdarft limit has been exceed"`

You must also override the `withdraw` method so that when a withdraw is completed the `balance` is decreased by the amount of the withdraw and the `fee` associated with the `Chequing` account.

In terms of the `__str__` method you should return a string with the following format:

```
Owner: owner
Balance: <Currency Symbol>balance
Fee: <Currency Symbol>fee
Overdraft Limit: <Currency Symbol>overdraftLimit
```

- **Note:** For full marks make use of Account's `__str__` method

Lastly, add setters and getters for `fee` and `overdraftLimit`.

- **Important:** `overdraftLimit` can only be updated as long as it does not cause the `balance` to go into overdraft. If it does display the message following message and do not perform the update: `"ERROR: Overdarft limit will be exceed. Update abandoned!"`

Savings.py

A `Savings` account has one additional property to that of `Account`. An annual `interestRate` on the account value. Interest is incurred monthly on account balance.

In terms of the initializer (`__init__`) at minimum when creating the account `owner,` and `interestRate` must be provided. Both `currency` and `balance` are optional, but if not provided should default to `"CAD"` and `0`.

- **Note:** If a currency outside the dictionary attribute is provided display the error message: `"ERROR: Unsupported currency type"`
- `balance` has no restrictions as a negative value indicates an account in overdraft
- `interestRate` (i.e. A value 2 represents 2% interest)

Next add a method called `applyInterest()` that applies the incurred interest based on the account `balance` and `interestRate`.

Finally, ensure a setter and getter exists for `interestRate`.

Investment.py

An investment account has two additional attributes to that of `Account`: double `cash` and a dictionary of `stockHoldings`. Additionally, `Investment` should have a dictionary attribute called `stockList` with the following initial properties:

1. SHOP → [994.70, CAD]
2. IBM → [129.87, USD]
3. OTEX → [58.44, CAD]
4. JD → [60.70, USD]
5. MSFT → [196.84, USD]

The above stocks are the only stocks your investment account can hold. And the values represent the cost of one stock expressed in currency specified.

`stockHoldings` should be a dictionary with the initial values:

1. SHOP → 0
2. IBM → 0
3. OTEX → 0
4. JD → 0
5. MSFT → 0

This attribute maintains the number of each stock you hold in your `Investment` account.

In terms of the initializer (`__init__`) at minimum when creating the account `owner` must be provided. Both `currency` and `balance` are optional, but if not provided should default to `"CAD"` and `0`.

You must also override the `deposit` and `withdraw` methods as the values provided should increase the value of both `cash` and `balance` by the specified amount.

Additionally, you need a updateStockPrice(`tickerSymbol, value)` that can be used to set a stock price in `stockList`.

- `tickerSymbol` is the String representing stock to update(i.e. "JD")
- `value` is the new dollar value of the stock specified
- If enough and `ticketSymbol` is valid and value is greater than or equal to zero update the stock price in `stockList`; otherwise, display the error message: `"ERROR: Stock price could not be updated!"`
- **IMPORTANT:** When a stock price is updated remember to update the balance of the account accordingly.

Stocks can be purchased from `cash` via the `buy(tickerSymbol, amt)` method.

- `tickerSymbol` is the String representing stock to purchase (i.e. "JD")
- `amt` is the number of stocks you want to purchase (integer number)
- If enough `cash` is available and `ticketSymbol` is valid purchase the specified volume of stocks; otherwise, display the error message: `"ERROR: Stock purchase could not be completed!"`
- **Remember:** To keep in mind the currency of the stock vs the `currency` of this account

Stocks can be sold from `stockHoldings` and converted back to `cash` via `sell(tickerSymbol, amt)` method.

- `tickerSymbol` is the String representing stock to sell (i.e. "JD")
- `amt` is the number of stocks you wish to sell(integer number)
- If an adequate amount of stock is available in `stockHoldings` and `ticketSymbol` is valid sell the specified volume of stocks; otherwise, display the error message:
- `"ERROR: Stock sale could not be completed!"`
- **Remember:** To keep in mind the currency of the stock vs the `currency` of this account

In terms of the `__str__` method you should return a string with the following format:

```
Owner: owner
Balance: <Currency Symbol>balance
Stock Holdings:
SHOP – amt @ <Currency Symbol>value currency
IBM – amt @ <Currency Symbol>value currency
OTEX – amt @ <Currency Symbol>value currency
JD – amt @ <Currency Symbol>value currency
MSFT – amt @ <Currency Symbol>value currency
```

- **Note:** For full marks make use of Account's `__str__` method
- An example stock line in __str__ would look like:

```
IBM – 3 @ $129.87 USD
```

AccountTester.py

This class must be included, but what it contains is up to you. I want you to use this class to test the your system's functionality as you move throughout the assignment.