CS304 Assignment 01: Python classes and inheritance

Due: Wednesday February 2nd at 11:59 PM. See syllabus for late penalties.

Submission: upload your source code to moodle.

Groups: work must be done *individually* (no teamwork is allowed, do not share source code with classmates. Duplicate submissions will receive a grade of 0%).

Background: A *class* serves as the primary means for abstraction in object-oriented programming. In Python, every piece of data is an *instance* of some class. A class provides a set of behaviors in the form of *member functions* (also known as *methods*) with implementations that are common to all instances of that class. A class serves as a *blueprint* for its instances, determining the way that state information for each instance is represented in the form of *attributes* (also known as *fields, instance variables,* or *data members*). In object-oriented programming, *inheritance* allows a new class to be defined based upon an existing class as the starting point. The base class is also known as *parent* or *super-class,* while the newly defined class is known as the *subclass* or *child.*

In this assignment, we will work with classes and inheritance in python, and do some simple testing on our class to ensure correctness of our methods.

Your job is to create three classes:

1. `BankAccount`
2. `ChequingAccount`
3. `SavingsAccount`

`BankAccount` must be an 'abstract base class' (see section 2.4.3 of LECTURE_02_NOTES). `ChequingAccount` and `SavingsAccount` must both *inherit* from `BankAccount` (see section 2.4.1 of LECTURE_02_NOTES).

`BankAccount` must be constructed with 3 arguments to `__init__` which are:

1. Name of the person who owns the bank account (string)
2. Name of the bank (string)
3. Initial balance (float)

`BankAccount` must implement the following *methods*

1. `deposit(amount)` – adds `amount` to `balance`
2. `withdraw(amount)` – attempts to subtract `amount` from `balance`.
    a. Returns `True` if `balance - amount > 0`, and `False` otherwise

`ChequingAccount`, which inherits from `BankAccount`, must take an additional argument to `__init__` which is `transaction_fee`.

`ChequingAccount` must also implement the method `make_purchase(amount)` which will call the `withdraw` function of the super class, and apply `transaction_fee` *after* `withdraw` returns.

SavingsAccount, which also inherits from BankAccount, must take an additional argument to __init__ which is interest_rate (a float between 0 and 0.025).

SavingsAccount must implement the method accrue_interest which will calculate the interest on balance and increase balance by the amount: interest_rate * balance.

Tests:

In __main__ (see 'testing the class' in section 2.3 of LECTURE_02_NOTES) provide method coverage and statement coverage for your ChequingAccount and SavingsAccount classes. Method coverage means your tests should call every method of the class at least once, and statement coverage means your tests should result in the execution of every statement in the class at least once (so method coverage is actually a subset of statement coverage).

Operator overloading:

In table 2.1 of LECTURE_02_NOTES, overloaded operations are shown along with Python's corresponding special methods. Implement the +=, −=, <, and > operators for your BankAccount class by defining the corresponding method (for example, the corresponding method to '+=' is __iadd__. The operator overloading methods should have the following behavior:

+= and −= should add to or subtract from the bank account's balance.

< and > should compare two bank account balances. For example, if a.__lt__(b) is called, it will return True if the balance field of BankAccount a is less than the balance field of BankAccount b.

Be sure to include tests for your overloaded operator methods!

Include 3 source files in your moodle submission:

1) BankAccount (no tests needed because it is an abstract class
2) ChequingAccount (include the tests at the bottom)
3) SavingsAccount (include the tests at the bottom)