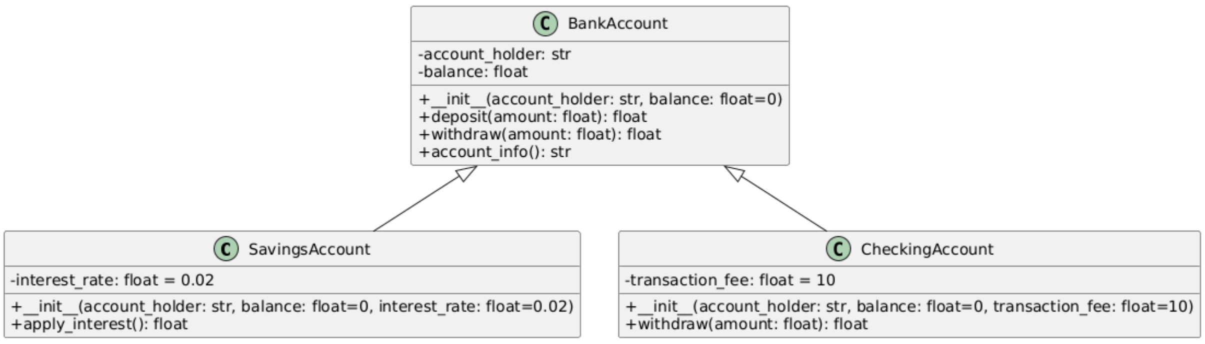
1. Introduction

In this report I was tasked with designing and implementing a class structure using class inheritance. The goal was to create one base class and two derived classes that inherit from the base class while keeping its own specific functionality.

The class structure is supposed to resemble a simple banking system consisting of 3 different account types. The base class is a Bank Account. This account should hold account holder name and balance, while including methods for depositing and withdrawing amount to the balance and printing information about the account. The derived classes should inherit the deposit, withdrawal and account info methods from the base class but have their own functionality as well. For Checking Account, this includes a transaction fee. For every withdrawal made from the balance, there should be included a transaction fee. The Savings Account should apply interest to the balance in the account.



C BankAccount
-account_holder: str
-balance: float
+__init__(account_holder: str, balance: float=0)
+deposit(amount: float): float
+withdraw(amount: float): float
+account_info(): str
C SavingsAccount
-interest_rate: float = 0.02
+__init__(account_holder: str, balance: float=0, interest_rate: float=0.02)
+apply_interest(): float
C CheckingAccount
-transaction_fee: float = 10
+__init__(account_holder: str, balance: float=0, transaction_fee: float=10)
+withdraw(amount: float): float

Image 1. Created using PlantUML

2. Design and Implementation

The concept of designing and implementing the banking system using class inheritance meant the base class should cover most of the use cases for every account. Core attributes in this case were account holder information (just a name) and the balance of the account. Methods implemented were depositing and withdrawing an amount and printing account info. There are some precautions taken in the methods to ensure each class act more like a real bank account. In the deposit method a simple IF statement makes sure the amount passed is greater than 0, as it is impossible to deposit negative numbers, that would be withdrawing. The withdrawal method uses the same logic of requiring a positive number, while also checking the balance to ensure the withdrawal amount does not exceed the balance.

The derived classes, both Savings Account and Checking Account will always use the methods from the base class Bank Account. They just have specializations in addition to the base class. This inheritance of methods and attributes allows for easier debugging and less redundant code.

Savings account calls on the base class to create an account holding account holder and balance but has another attribute – interest rate. In my notebook this is default to 0.02 but can be any number higher than or equal to 0. This can be set by passing a third parameter in the function or letting it be the default. The method for this class is interesting (pun intended). It multiplies the balance by the interest rate and adds interest to the balance before returning a new balance.

Checking Account does the exact same thing as the Savings Account. Instead of interest rate, it takes a transaction fee attribute. To make sure the transaction fee is added to the action of withdrawing from the balance, and making sure to keep redundancy at a minimum, I simply return the withdrawal amount call the parent class withdrawal and add the amount and transaction fee in the passing of the parameter. This makes sure the transaction fee is added, and that there is a sufficient balance, and positive withdrawal amount all without repeating code.

3. Discussion & Reflection

This assignment was fun and presented some challenges. The main challenge was logic, as the code itself isn't necessarily complicated. The classes are either subtracting or adding a specific value to a balance. I figured out quite quickly that I would make one deposit method and reuse them in the child classes. The withdraw method in the checking account started out more complex than it needed to be, and I did improve upon it until it just returned a super() call on the base class which was a fun realization. A great way to make sure an account method overrides the base class method, is to use the base class method.

A fun improvement/implementation would be to simulate an ATM, introduce some more logic in the classes, like having the account holder be a variable that holds multiple accounts. And then actions done in each class use this inheritance. There are surely more tests that could be done.

I haven't used inheritance before this, but it will be useful for future project, and the learning curve was not steep and fun to do.

4. Conclusion

The repository contains everything used in this assignment and can be found on (repo link). The tasks were completed in line with the assignment, and the results prove this. The notebook is well organized, comments are sufficiently made through the code and the report used to discuss my work.