ACIT4420 Report

Anthony Berg

20th September 2025

Inheritance

Making sure that the use of inheritance can be future-proof, maintainable, and readable, required usually requires some sort of planning. It is usually down to what the children classes are supposed to do, as at times, the parent class would either contain most of the logic, and/or the parent class should be a basis of standardizing the child classes.

In this case, it was both to standardize and avoid repeated code in the child classes. `BankAccount` holds most of the basic logic that the child classes would require, yet also a standard for the types of variables the other types of accounts the bank would offer.

The main design ethos with `BankAccount` was so that it would hold the most of the logic that could be reused by the children classes. For example `BankAccount._check_valid()` is used by the base class and every child to check if the value is positive, otherwise it raises a `ValueError` with the message depending on if it is zero or negative. Therefore, if something was implemented incorrectly, it would only require one function to be changed, rather than three functions in three separate classes.

Method Overriding in `CheckingAccount`

In `CheckingAccount`, the `withdraw()` method overrides the one from `BankAccount` when it is called from a `CheckingAccount` object, but still uses `BankAccount.withdraw()` by withdrawing the requested amount plus the transaction fee by calling `super().withdraw()`. However, at the end it will only return the amount requested.

The reason for using `super().withdraw()` is in case the logic for withdrawals may change in the future, and it removes repetitive code, such as running checks if the amount requested being invalid.

Design Decisions

One design decision was using a data class to represent account information. This was mainly done to improve maintainability as Language Server Protocols (LSPs) knows what variables, hence autocompletion and type hints would be available.

The classes were also built to be very basic, as it was assumed that these classes would be a part of the basis of the back-end. Therefore, something higher up in the back-end could run checks on the inputted/returned values, improve logging, or have another class for a person that holds multiple types of accounts.

A major consideration for the balance was to use integers instead of a float, as floats are inaccurate, and could cause people's balances to be higher or lower than expected. Therefore, the decimal point should be added elsewhere.

However, there are not a lot of performance optimizations in the code, other than using variables to prevent repetitive calculations. The main focus was on optimizing the code for maintainability, as there are a few extra calls to other functions in the children classes, to the parent class, when the code could otherwise just be a few extra lines for better performance instead.