ACIT4420-1 25H Problem-solving with scripting
Assignment 1

Azza H. Ahmed

September 21, 2025

1 Objective

In this assignment, you will design and implement a class structure using **class inheritance**. The goal is to create a base class and multiple derived classes, each with its own specific functionality. You will also be required to debug and optimize the code, focusing on the proper use of inheritance to avoid redundancy and to improve code maintainability.

2 Problem Description

You are tasked to develop a class structure for a simple **banking system**. The system must have the following types of bank accounts:

1. **Standard Bank Account**
2. **Savings Account** (inherits from Standard Bank Account)
3. **Checking Account** (inherits from Standard Bank Account)

Each type of account has different rules for handling deposits, withdrawals, and interest.

3 Requirements

1. **Base Class:** BankAccount
   - **Attributes**:
     account_holder: Name of the account holder.
     balance: The current balance (initialized to 0 by default).
   - **Methods:**
     deposit(amount): Adds the specified amount to the account balance.
     withdraw(amount): Subtracts the specified amount from the balance, if sufficient funds exist. If not, print a message indicating insufficient funds.
     account_info(): Returns the account holder's name and current balance.

2. **Derived Class:** SavingsAccount
   - **Attributes:**
     interest_rate: A fixed interest rate (e.g., 2% annually).
   - **Methods:**
     apply_interest(): Applies the interest to the balance. (e.g., increases the balance by multiplying it by (1 + interest_rate)).

3. **Derived Class:** CheckingAccount
   - **Attributes:**
     transaction_fee: A fixed fee (e.g., $1) that is charged for every withdrawal.
   - **Methods:**
     withdraw(amount): This method overrides the base class method to subtract the transaction fee in addition to the withdrawn amount.

4 Tasks

1. Design and Implement the Class Structure:
   - Create a base class BankAccount with the attributes and methods specified above.
   - Create derived classes SavingsAccount and CheckingAccount that inherit from BankAccount and implement additional functionality specific to each type of account.

2. Inheritance and Method Overriding:
   - Ensure that the withdraw() method in CheckingAccount correctly overrides the withdraw() method from BankAccount.
   - Ensure that SavingsAccount has a method apply_interest() that applies the interest rate to the balance.

3. Debugging:
   - Test the classes with various inputs to ensure correct behavior (e.g., deposit 100, withdraw 50, try over-withdrawing, apply interest, etc).
   - Ensure that withdrawing funds from a CheckingAccount deducts both the withdrawal amount and the transaction fee.
   - Ensure that the interest is correctly applied to the balance in SavingsAccount.

4. Optimization:
   - Avoid redundancy by using inheritance to reuse common functionality.
   - Ensure that methods are correctly overridden where needed, and avoid duplicating code between the derived classes.

5 Deliverables

1. Python code with the complete class hierarchy, including base and derived classes.
2. A report (maximum 2 pages) explaining:
   - How inheritance was used to avoid redundancy.
   - How the withdraw() method was overridden in CheckingAccount.
   - A brief explanation of any optimizations or design decisions.
3. A set of test cases demonstrating the functionality of the classes.

6 Evaluation Criteria

1. **Code Implementation (50%)**
   - Correct use of inheritance and class hierarchy.
   - Proper implementation of all required attributes and methods.
   - Method overriding works as specified (e.g., withdraw() in CheckingAccount).
   - Code runs without errors.

2. **Testing (20%)**
   - Test cases cover deposits, withdrawals, interest application, and insufficient funds.
   - Demonstrates edge cases (e.g., over-withdrawal, zero balance).

3. **Report (20%)**
   - Explains how inheritance was used to avoid redundancy.
   - Describes method overriding in CheckingAccount.
   - Discusses any optimizations or design decisions.

4. **Code Quality & Style (10%)**
   - Readable, well-commented code.
   - Consistent naming conventions.
   - Avoids unnecessary duplication.