Fall 2025 - CS 122

Homework 3

NOTE: Some of the concepts here will be dependant on the topics covered on Sep. 23rd.

Due: Sep. 28th , 11:59 PM
Topics: Object-Oriented Programming (OOP), Inheritance, Custom Exceptions, File I/O
Total Points: 100

Overview
Homework Setup and Submission
1. File Structure:
You must create the following directory structure, replacing SJSUID with your own 9-digit San Jose State University ID number. The banking folder is a Python package.

```
hw3_SJSUID/
├── banking/
│   ├── __init__.py      (can be empty)
│   ├── exceptions.py
│   ├── accounts.py
│   └── persistence.py
├── main.py
└── test_homework.py
```

2. Submission:
Compress the entire hw3_SJSUID directory into a single file named hw3_SJSUID.zip and submit it.

3. Testing Your Work:
A file named test_homework.py is provided. Place it in the root of your hw3_SJSUID directory as shown. This script will help you check if your implementation is correct.
How to Run the Tests:
Open your terminal, navigate into your hw3_SJSUID directory, and run the command:
python test_homework.py
Your goal is to make all tests pass.

Grading Rubric

- **Automated Tests (60 points):** Your submission must pass all tests in test_homework.py.
- **Instructor Tests (20 points):** Your code will be tested against a private set of tests with different inputs.
- **Code Quality (20 points):** Your code must be clean, well-commented, and logically structured.

Task Breakdown by File

1. banking/exceptions.py

In this file, you will define two custom exception classes that inherit from Python's base Exception class.

- InsufficientFundsError: To be raised when a withdrawal is attempted for an amount greater than the available balance (including fees).
- InvalidAmountError: To be raised when a deposit or withdrawal is attempted with a negative amount or zero.

2. banking/accounts.py

This file will contain the class definitions for the bank accounts.

- **BankAccount (Base Class):**
  - An __init__(self, owner_name, initial_balance=0.0) method to set the account owner's name and initial balance.
  - A deposit(self, amount) method. It should raise an InvalidAmountError if the amount is not positive. Otherwise, it adds the amount to the balance.
  - A withdraw(self, amount) method. This method in the base class should simply raise a NotImplementedError. This forces the subclasses to provide their own implementation.
  - A __str__(self) method that returns a formatted string, e.g., "Account Owner: Alice, Balance: $100.00". The balance should be formatted to two decimal places.
- **SavingAccount (Subclass of BankAccount):**
  - It should have a withdrawal fee of **$2.00**.
  - Override the withdraw(self, amount) method. It must:
    1. Raise InvalidAmountError if the amount is not positive.
    2. Raise InsufficientFundsError if amount + fee is greater than the current balance.
    3. If successful, subtract both the amount and the fee from the balance.
- **CheckingAccount (Subclass of BankAccount):**
  - It should have a withdrawal fee of **$1.00**.
  - Override the withdraw(self, amount) method with the same logic as the SavingAccount, but using its specific fee.

3. banking/persistence.py

This file will manage all accounts and handle saving/loading data.

- **Bank Class:**
  - An __init__(self, db_path) method. It should initialize an empty dictionary self.accounts to hold account objects and store the db_path for the shelve file.
  - A create_account(self, account_type, owner_name, initial_balance=0.0) method. It checks if an account for owner_name already exists. If it does, an error message should be printed, and the method should return False. If not, it creates a SavingAccount or CheckingAccount object and adds it to the self.accounts dictionary, with the owner's name as the key. The method should return True on successful creation.
  - A get_account(self, owner_name) method that returns the account object for a given name, or None if not found.
  - A save_data(self) method that uses the shelve module to save the entire self.accounts dictionary to the file specified by db_path.

- A load_data(self) method that uses shelve to load the accounts dictionary from the file. This should handle the case where the file doesn't exist yet (e.g., the first time the program runs). If the shelve file does not exist when this method is called, it should not raise an error. Instead, the self.accounts dictionary should simply remain empty.

4. main.py

This is the user-facing part of your application. It should contain the command-line interface (CLI) for the user to interact with the bank. **You are not required to implement the full interactive menu for this assignment.** You only need to ensure your classes and methods from the banking package work correctly as per the tests. You may optionally build the main function for your own testing purposes, main.py is not graded.