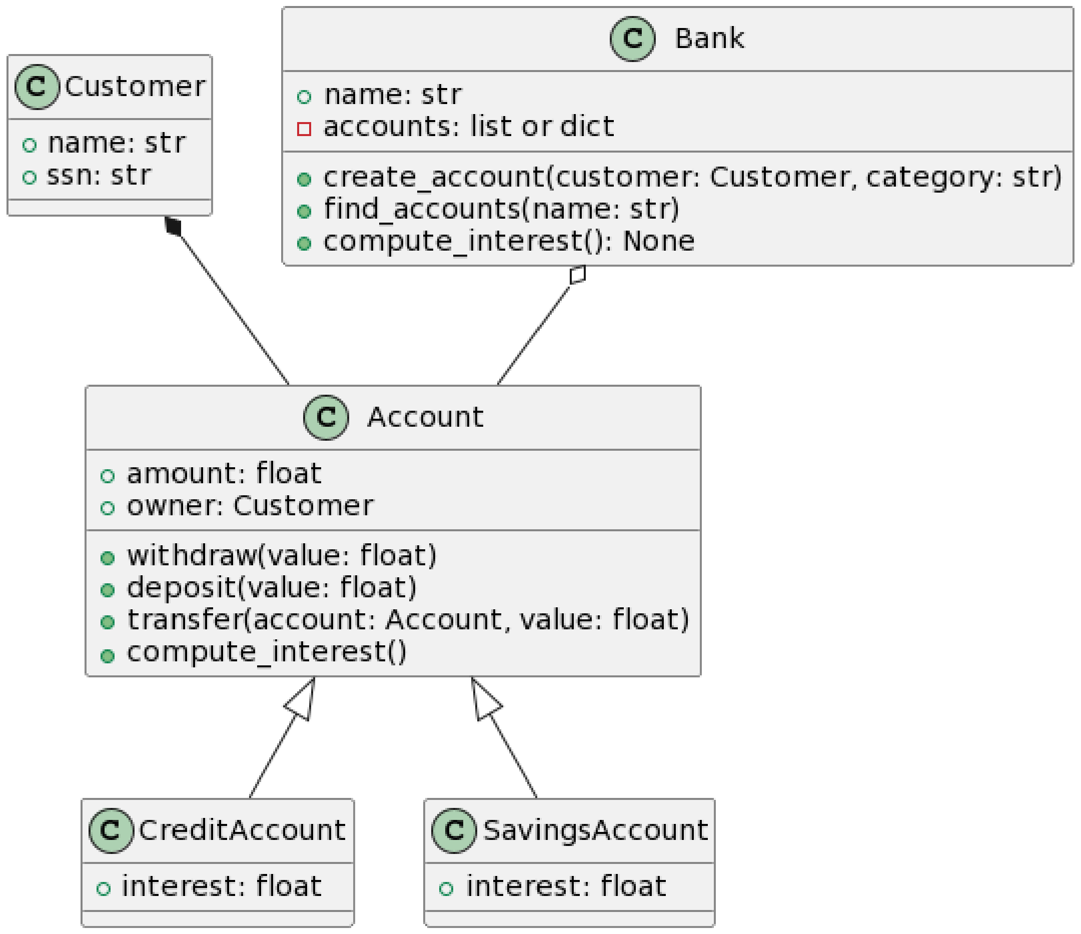
ACIT2515

Lab: bank and bank accounts

Class diagram



1. Use the `Customer` class

A customer has:

- a name
- a SSN (Social Security Number)

The code for this class is already provided.

2. Build the `Account` class

A bank account has:

- "owner": a `Customer` instance
- an amount (float, default value: 0)

- a method `deposit`
    - it receives a `float` argument: the amount to be deposited. If the argument is negative, raise a `AttributeError` exception.
    - it adds the deposited amount to the account amount
- a method `withdraw`
    - it receives a `float` argument: the amount to be withdrawn. If the argument is negative, raise a `AttributeError` exception.
    - it removes the amount provided from the account
- a method `transfer`, to make transfers between accounts
    - it has two arguments: `account` and `amount`
    - it must raise a `TypeError` exception if the account is not an instance of `Account`
    - the method withdraws `amount` from the current instance and deposits it into the `account`
- a method `compute_interest`, used to compute the interest over the account
    - this method does nothing on regular accounts

3. Build child classes

3.1. `CreditAccount`

- The credit account inherits from `Account`. Its amount is usually negative.
- Its constructor receives an additional argument `interest_rate` (a number between 0 and 100).
- This is the interest rate in %. Store it in the `interest` attribute of the class.
- The `compute_interest` method, **if the amount is negative**:
    - charges the interest to the account: `amount = amount * (100 + interest_rate) / 100`
    - then charges $10 to the account (administration fees)

3.3. `SavingsAccount`

- The savings account inherits from `Account`.
- Its constructor receives an additional argument `interest_rate` (a number between 0 and 100).
- This is the interest rate in %. Store it in the `interest` attribute of the class.
- The `compute_interest` method adds the interest to the account: `amount = amount * (100 + interest_rate) / 100`
- The `withdraw` method must raise an `UserWarning` exception if someone tries to withdraw more money than available on the account.

4. Build the `Bank`

- A bank has a name (received by the constructor)

- `create_account`: creates an account in the bank. Receives two mandatory arguments

    - `category`: can be either "account", "credit", or "savings"
    - `owner`: a `Customer` instance
    - and an optional argument: `interest_rate` (default value = 0)

        - credit and savings accounts must use this value for the interest rate
    - this method creates an account of the specified type, associates it with the provided owner
    - the method **returns** the account created, but you need to make sure your bank keeps track of the accounts (you could use a dictionary, or a list)

- `compute_interest()`: computes interest on all accounts of the bank.

- `find_accounts()`: receives one argument

    - first argument is a string: the name of a customer
    - the method returns the list of account(s) associated with the given customer

Grading rubric

- 1 mark for each test that passes
- 3 marks for PEP8 syntax, docstrings, and encapsulation (use of `property` and `setter`)

TOTAL = 18 marks