Python Full Stack Development Training
OOP Mini-Project Assignment (Day 4)

Instructor: Giridhari Lal Gupta

Objective

Implement a simple banking system to practice Object-Oriented Programming concepts:

- Class construction and `__init__`
- Instance vs. class variables
- Encapsulation via name-mangling and `@property`
- Inheritance and method overriding
- Polymorphism and duck typing
- Operator overloading and special methods
- Composition (“has-a” relationships)

Requirements

1. Base Class: `BankAccount`

- **Constructor:**
  - `__init__(self, owner:str, balance:float=0.0)`
  - store `owner` publicly; store `balance` as name-mangled `__balance`
- **Class Variable:** `total_accounts` (int), incremented on each new account
- **Methods:**
  - `deposit(self, amt:float)` — validate `amt>0`, update `__balance`
- `withdraw(self, amt:float)` — validate `0<amt`$\leq balance, update$`__balance`
- **Property:**
- `@property def balance(self)`$\to float | returns$`__balance@balance.setter`$| validates non-negative assignment$
- **Special Methods:**
  - `__str__` — return `"BankAccount(owner=..., balance=...)"`
  - `__repr__` — detailed representation
  - `__add__(self, other)` — merge two accounts into a new one; owner names joined by ‘” ”‘; adjust `total_accounts`

2. Subclasses

`SavingsAccount`:
- adds instance variable `interest_rate`
- method `apply_interest(self)` — increases `balance` by `balance` × `interest_rate`

`CheckingAccount`:
- protected attribute `_overdraft_limit`
- override `withdraw(self, amt)` — allow up to `balance + overdraft_limit`
- `@property` and `@overdraft_limit.setter` for validating limit ≥ 0

3. Composition: `Customer`

- `__init__(self, name:str)` — initializes `accounts` list
- `add_account(self, account:BankAccount)` — attach account
- `total_balance(self)`$\to float | sum of all account balances$`transfer(self, from_acc, to_acc, amt)`$| withdraw then depo$

4. Duck Typing Utility

- Function `print_account_summary(obj)` — accesses `obj.owner` and `obj.balance` (or `get_balance()`) without checking type
- Demonstrate on both `BankAccount` and another custom class with similar interface

5. Demo Polymorphism

- Create instances of `SavingsAccount` and `CheckingAccount`
- Store in one list and invoke common methods (`withdraw`, `apply_interest`, etc.) to show polymorphic dispatch
- Merge two accounts via `+` operator

Deliverables

1. `banking.py` — all class definitions and utility function
2. `demo.py` — script that:
    - Creates and manipulates accounts as specified
    - Demonstrates deposits, withdrawals, overdraft, interest
    - Merges accounts using `+`
    - Creates a `Customer`, adds accounts, prints total
    - Calls `print_account_summary` on each object
    - Prints each object (using `__str__` / `__repr__`)
3. `README.md` — brief explanation:
    - Which Day 4 concept each part covers
    - Any challenges or design decisions

Grading

- **Correctness:** All methods behave as specified; validations raise appropriate exceptions.
- **Use of OOP:** Clear demonstration of each required concept.
- **Code quality:** PEP 8 naming, docstrings, clear structure.
- **Completeness:** Demo covers all points; README documents concepts.