Python OOP Practice Problem — Employee Asset Management System

Problem Overview

Design and implement a **Python program** to manage employees and their allocated assets in a company.
The system should calculate salaries, validate employee and asset data, handle invalid cases gracefully, and generate reports.
Your solution must demonstrate:

✓ Class design

✓ Inheritance

✓ Encapsulation (getters/setters)

✓ Static/class variables

✓ Exception handling

✓ Polymorphism

✓ Regular expressions

✓ Aggregation (employees having assets)

Class Specifications

1. Asset Class

Constructor:
Asset(asset_id: str, asset_name: str, asset_expiry: str)
Description:
Initializes the instance variables with the given values.
Validations:

- The asset_id must match the following pattern:
  Should start with either "DSK" or "LTP" or "IPH"
- followed by a hyphen (-),
- followed by exactly 6 digits,
- and ending with a character 'H' or 'L' (case-insensitive).

Example of valid IDs:

- DSK-123456H
- LTP-654321l
- IPH-777777L

If invalid, raise an InvalidAssetsException with the message:
"Invalid Asset Id: <asset_id>"

-

Methods to Implement:

- set_asset_id(asset_id)
- get_asset_id()
- get_asset_name()
- get_asset_expiry()

2. Resources Class

Static Method:

get_month(month: str) -> int

Description:

Converts a three-letter month abbreviation (e.g., "Jan", "Feb", "Mar") to its corresponding month number (1–12).

Rules:

- The input month string must be **3 characters** long and start with an **uppercase letter**.
- Return 0 for invalid inputs.

Example:

- get_month("Sep") → 9
- get_month("sep") → 0
- get_month("Abc") → 0

3. InvalidAssetsException Class

Constructor:

InvalidAssetsException(message: str)

Stores the message passed during object creation.

4. InvalidExperienceException Class

Constructor:

InvalidExperienceException(message: str)

Stores the message passed during object creation.

5. Employee (Base Class)

Static Variables:

- contract_id_counter = 10000
- permanent_id_counter = 10000

These counters are used to auto-generate employee IDs.

Constructor:

Employee(employee_name: str)

- Validates and sets the employee name.
- Employee name is valid if:
  - It contains **only alphabets and spaces**.
  - There are **at least 2 words**.
  - Each word starts with an **uppercase letter** and has at least 2 characters.

ID Generation Rules:

- For contract employees → IDs start with "C10001", "C10002", etc.
- For permanent employees → IDs start with "E10001", "E10002", etc.

Methods to Implement:

- set_employee_name(name)
- get_employee_name()
- set_salary(salary)
- get_salary()
- get_employee_id()

Invalid name should raise:

ValueError("Invalid Employee Name: <name>")

6. ContractEmployee (Derived from Employee)

Constructor:

ContractEmployee(employee_name: str, wage_per_hour: float)

- Initializes inherited variables using super().
- Assigns the employee ID automatically.

Method:

calculate_salary(hours_worked: float)

Logic:

salary = wage_per_hour * hours_worked

If hours worked < 190:

Deduction = (0.5 * wage_per_hour) * (190 - hours_worked)

salary = (wage_per_hour * hours_worked) - deduction

Round the salary to the nearest integer.

7. PermanentEmployee (Derived from Employee)

Constructor:

PermanentEmployee(employee_name: str, basic_pay: float, salary_components: list[str], assets: list[Asset])

Example:

salary_components = ["DA-50", "HRA-40"]

Here, DA = 50% of basic pay, HRA = 40% of basic pay.

Methods:

1. calculate_bonus(experience: float)
    - Bonus based on experience:

◦ Experience	◦ Bonus % of basic pay
◦ < 2.5 years	◦ ❌ InvalidExperienceException
◦ 2.5–5 years	◦ 10%
◦ 5–10 years	◦ 20%
◦ > 10 years	◦ 35%

    - Raise: InvalidExperienceException("A minimum of 2.5 years is required for bonus!")
2. calculate_salary(experience: float)
    - Formula: salary = basic_pay + DA + HRA + bonus
    - Handle exception: If InvalidExperienceException is raised → set bonus = 0
    - Round the salary to the nearest integer.
3. get_assets_by_date(last_date: str)
    - Return all assets whose expiry ≤ last_date.
    - Date format: "YYYY-MON-DD", e.g., "2021-Dec-31"
    - Use Resources.get_month() for month validation.
    - Raise InvalidAssetsException("No assets found for the given criteria!") if none match.

8. Admin Class

Implements company-level operations.

Methods:

a) generate_salary_slip(employees: list[Employee], salary_factor: list[float])

- Calls each employee's calculate_salary() with the corresponding factor.
- The factor is:
    - hours_worked → for contract employees
    - experience → for permanent employees

b) generate_assets_report(employees: list[Employee], last_date: str) -> int

- Return total count of assets expiring on or before last_date for all permanent employees.
- If InvalidAssetsException occurs for any employee, return -1.

c) generate_assets_report(employees: list[Employee], asset_category: char) -> list[str]

- Return asset IDs starting with the given category character (case-insensitive).
- Length of the returned list = 3 × number of employees.

9. Tester Class (Main)

Create objects for:

- Assets (valid and invalid)
- Permanent and contract employees
- Admin to generate:
  - Salary slips
  - Asset expiry reports
  - Asset category reports

Ensure to test:

- Invalid asset IDs
- Invalid employee names
- Invalid experience (< 2.5 years)
- Salary calculation for both employee types
- Exception handling in asset reports