Low-Level Design (LLD) – Pandas Student Performance Tracker

Total Marks: 25

Design Format: 1 Class with 5 Independent Methods | 5 Visible Test Cases

Summary of Design Requirements

Implement a class StudentTracker using Python and the Pandas library.

The class should:

- Maintain a database of student grades and subjects.
- Perform analytical queries using Pandas methods.

Each function must be independent:

- Each method must work based on passed or internal data only.

Use Pandas DataFrame for core storage.

Avoid complex loop structures; use Pandas built-in functions.

Concepts Tested

- Class initialization and state (`self.df`)
- DataFrame creation and adding data (`append` or `concat` or `loc`)
- Basic aggregation (`mean`, `count`)
- Boolean indexing for filtering
- Exporting/Checking DataFrame status

Problem Statement

Design a Python class StudentTracker to manage academic records. You will use a Pandas DataFrame as the internal structure to store data about students, their scores, and the subjects they are enrolled in.

The DataFrame should have the following columns:

- 'StudentName' (string)
- 'Score' (float/int)
- 'Subject' (string)

Operations (**Methods**)

1. Initialize Tracker (**Easy**)

Create an empty Pandas DataFrame with columns: ['StudentName', 'Score', 'Subject'].

def __init__(**self**)**:**

self.data → Empty pd.DataFrame

2. Record Grade (**Easy**)

Add a student's performance record to the internal DataFrame.

def record_grade(**self, name: str, score: float, subject: str**) -> **None:**

- Add a new row to self.data with the provided values.
- No return value.

3. Calculate Subject Average (**Easy**)

Find the average score for all students in a specific subject.

def get_subject_avg(**self, subject: str**) -> **float:**

- Filter DataFrame by the given subject.
- Calculate and return the mean of the 'Score' column.
- If subject doesn't exist, return 0.0.

4. Find Top Scorers (**Medium**)

Return a list of student names who scored above a certain threshold.

def get_top_scorers(**self, threshold: float**) -> **list:**

- Filter DataFrame where 'Score' is greater than (>) the threshold.
- Return a unique list of 'StudentName' values.
- Return an empty list if no one qualifies.

5. Data Summary (**Medium**)

Return the total count of unique students and the total records stored.

def get_summary_stats(self) -> dict:

- Calculate total number of rows.
- Calculate total unique student names.
- Return as: {'total_records': int, 'unique_students': int}

Test Cases & Marks Allocation

Test Case ID	**Description**	**Method**	**marks**
TC1	Initialize DataFrame structure	__init__()	5
TC2	Add rows to the DataFrame	record_grade()	5
TC3	Get average for a specific subject	get_subject_avg()	5
TC4	List names based on score threshold	get_top_scorers()	5
TC5	Get record and uniqueness counts	get_summary_stats()	5
	Total		**25**

-

Visible Test Case Descriptions

TC1: Instantiating StudentTracker() initializes self.data with correct column headers.

TC2: record_grade("Alice", 85, "Math") adds one record to the DataFrame.

TC3: get_subject_avg("Math") returns 85.0 if Alice is the only record for Math.

TC4: get_top_scorers(80) returns ["Alice"] if Alice has 85 and others have less.

TC5: get_summary_stats() returns correctly formatted counts of rows and unique names.