Low-Level Design (LLD) – Retail Inventory System (OOPs in Python)

Difficulty Level: Easy - Intermediate | Total Marks: 20
Design Format: 1 Class with 5 Independent Methods | 5 Visible Test Cases

Summary of Design Requirements

Implement a class RetailStore using Object-Oriented Programming (OOP).
The class should:
- Track products, their prices, and stock levels.
- Perform sales transactions and update inventory.

Each function must be independent:
- Each method must work based on passed or internal data only.

Use simple Python data structures (dict, list).
Avoid external libraries.

Concepts Tested

- Python class and instance methods
- Data management using self
- Dictionary-based state tracking (Nested structures)
- Basic arithmetic logic and data validation
- Error handling through conditional returns

Problem Statement

Design a Python class RetailStore to manage a retail shop's inventory with the following features:
- Add products and set their prices/stock.
- Update stock manually for existing products.
- Process customer sales by checking availability.

- Generate a summary report of the inventory value.

Each product has:
- Name (unique string)
- Price (float)
- Stock (int)

Operations (Methods)

1. Initialize Store (Easy)
Create an internal data structure to store products.
def __init__(self):
  self.inventory → {product_name: {'price': float, 'stock': int}}

2. Register Product (Easy)
Add a new product with its price and initial stock level.
def register_product(self, name: str, price: float, stock: int) -> str:
  - Add to self.inventory
  - Return string: "Product [name] registered."

3. Restock Item (Easy)
Increase the stock count for an existing product.
def restock_item(self, name: str, quantity: int) -> str:
  - If product exists: add quantity to current stock, return "Restocked [name]. Now: [new_stock]".
  - Else: return "Product not found."

4. Complete Sale (Medium)
Deduct stock when a sale occurs and return the total billing amount.
def complete_sale(self, name: str, quantity: int) -> str:

- If product exists AND stock >= quantity:
  - Deduct quantity from stock.
  - Calculate total (price * quantity).
  - Return formatted string: "Bill: $[total]".
- If product exists but stock < quantity: return "Insufficient stock."
- Else: return "Out of catalog."

5. Inventory Report (**Medium**)

Return a summary of the total monetary value of all stock in the store.

def inventory_report(self) -> float:

- Calculate sum of (price * stock) for all products in self.inventory.
- Return the total value as a float.

Test Cases & Marks Allocation:

Test Case ID	**Description**	**Method**	**marks**
TC1	Initialize retail store structure	__init__()	5
TC2	Register a new product	register_product()	5
TC3	Update stock for existing item	restock_item()	5
TC4	Process a sale with stock check	complete_sale()	5
TC5	Calculate total inventory value	inventory_report()	5
	Total		**25**

Visible Test Case Descriptions

TC1: Instantiating RetailStore() should initialize inventory as an empty dictionary.

TC2: register_product("Laptop", 1200.0, 10) updates dictionary and returns confirmation.

TC3: restock_item("Laptop", 5) increases stock to 15 and returns the status string.

TC4: complete_sale("Laptop", 2) returns "Bill: $2400.0" and reduces stock to 13.

TC5: inventory_report() returns the sum of all stored (price * stock) values.