Low-Level Design (LLD) – Library Management System (OOPs in Python)

Difficulty Level: Easy | **Total Marks**: 20
Design Format: 1 Class with 6 Independent Methods | 6 Visible Test Cases

□ Summary of Design Requirements

- Implement a class `Library` using Object-Oriented Programming (OOP).
- The class should:
  - Track books and their availability.
  - Track transaction history.
- Each function must be **independent**:
  - Do **not rely on another method** to setup data.
  - 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 encapsulation using `self`
- Dictionary-based state tracking
- List manipulation and string formatting
- Pure function behavior with no cross-dependencies

□ Problem Statement

Design a Python class `Library` to manage a basic book tracking system with the following features:

- Add, borrow, and return books.
- Search for a book by title.
- View the full transaction history.

Each book has:

- Title (unique)
- Author
- Availability status (True/False)

Each transaction (add, borrow, return) should be recorded in a history list.

□ Operations (Methods)

1. Initialize Library

Create an empty dictionary of books and an empty list for transaction history.

```
def __init__(self):
```

- `self.books` → `{title: {'author': str, 'available': bool}}`
- `self.history` → `list[str]`

2. Add Book

Add a new book with title and author. Set its availability to `True`.

```
def add_book(self, title: str, author: str) -> None:
```

- Update `self.books`
- Append `"Added: 'title' by author"` to `self.history`

3. Borrow Book

Borrow a book by title if it is available.

```
def borrow_book(self, title: str, user: str) -> str:
```

- If available: set `available = False`, log `"Borrowed: 'title' by user"`, return that string.
- Else: return `"Not Available"`

4. Return Book

Return a borrowed book by title.

```
def return_book(self, title: str, user: str) -> str:
```

- If exists: set `available = True`, log `"Returned: 'title' by user"`, return that string.
- Else: return `"Book not found"`

5. Search Book

Check if a book exists.

```
def search_book(self, title: str) -> bool:
```

- Return `True` if the book is in `self.books`, else `False`.

6. View History

Return the full transaction history as a list of strings.

```
def view_history(self) -> list:
```

- Return `self.history`
- If empty, return list like `["No transactions yet."]`

□ Test Cases & Marks Allocation

Test Case ID	Description	Method	Marks
TC1	Initialize library	`__init__()`	□ 3
TC2	Add book	`add_book()`	□ 3
TC3	Borrow book	`borrow_book()`	□ 3
TC4	Return book	`return_book()`	□ 3
TC5	Search book	`search_book()`	□ 4
TC6	View full transaction history	`view_history()`	□ 4

Total Marks: 20

□ Visible Test Case Descriptions

- □ **TC1**: Instantiating `Library()` should initialize `books` as a dict and `history` as a list.
- □ **TC2**: `add_book("Python 101", "Guido")` should update `books` and log the transaction.
- □ **TC3**: `borrow_book("Python 101", "Alice")` should change availability and return confirmation.
- □ **TC4**: `return_book("Python 101", "Alice")` should restore availability and log return.
- □ **TC5**: `search_book("Python 101")` returns `True`, `search_book("ABC")` returns `False`.

- □ **TC6**: `view_history()` returns all transaction messages in order.