1. Objective

The goal of Task 3 was to implement a **Stack** data structure to manage book return requests in a Library Management System. The Stack follows the **LIFO (Last-In-First-Out)** principle, ensuring that the most recently returned books are processed first.

---

2. Implementation

Classes Developed

1. **ReturnRequest**
   - **Attributes**:
     - bookId (int): Unique identifier of the book.
     - returnTime (long): Timestamp of when the book was returned.
   - **Methods**:
     - Constructor, getters, and toString() for displaying request details.

2. **ReturnStack**
   - **Data Structure**: Built using a **LinkedList** for efficient insertion/deletion.

- **Methods**:
  - push(ReturnRequest request): Adds a request to the top of the stack (**O(1)**).
  - pop(): Removes and returns the top request (**O(1)**).
  - peek(): Returns the top request without removal (**O(1)**).
  - printStack(): Displays all pending return requests.

3. **Test Scenario (Main.java)**
   - Added **5 return requests** to the stack.
   - Processed (**popped**) **2 requests**.
   - Printed the remaining **3 requests**.

---

3. Test Output

text

```
Adding 5 return requests...
--- Return Stack (LIFO) ---
Book ID: 5 | Return Time: 1620000000000
Book ID: 4 | Return Time: 1620000000000
Book ID: 3 | Return Time: 1620000000000
Book ID: 2 | Return Time: 1620000000000
Book ID: 1 | Return Time: 1620000000000

Processing 2 return requests...
Popped: Book ID: 5 | Return Time: 1620000000000
Popped: Book ID: 4 | Return Time: 1620000000000

Remaining stack:
--- Return Stack (LIFO) ---
Book ID: 3 | Return Time: 1620000000000
Book ID: 2 | Return Time: 1620000000000
Book ID: 1 | Return Time: 1620000000000
```

---

4. Analysis

Why is a Stack Suitable for Return Requests?

1. **LIFO Efficiency**:
   - The most recent returns are processed first, mimicking real-world library workflows (e.g., staff handling the latest books on top of a pile).

2. **O(1) Operations**:
   - push(), pop(), and peek() are **constant-time** operations, making the Stack highly efficient for frequent insertions/deletions.

What If a Queue Were Used Instead?

1. **FIFO Inefficiency**:
   - A Queue would process the **oldest** return first, delaying urgent recent returns.
2. **Same Big O, Wrong Logic**:
   - While Queues also offer **O(1)** operations, their FIFO behavior is **ill-suited** for return management.

---

5. Conclusion

- **Stack is optimal** for return requests due to its **LIFO behavior** and **O(1) performance**.
- A Queue would introduce **logical inefficiencies** despite similar time complexity.
- The implementation successfully demonstrates how a Stack streamlines return processing in a library system.