4. Analysis:

Data Structure Used: HashMap<String, Product>

Time Complexities:

- addProduct(product): O(1) average case — inserting a product into a HashMap using a key.

- updateProduct(productId): O(1) average case — retrieving and modifying a product using its key.

- deleteProduct(productId): O(1) average case — removing a product by key.

- showInventory(): O(n) — iterates over all products.

Why HashMap is Chosen:

- Allows for direct access using a unique key (productId).

- Offers excellent performance for frequent read/write operations.

Potential Bottlenecks:

- Hash collisions in poorly implemented hash functions can degrade performance to O(n).

- Does not maintain any ordering of elements.

Optimizations and Extensions:

1. Thread-Safety:

- Use ConcurrentHashMap instead of HashMap for multithreaded environments.

2. Sorting Needs:

- Use TreeMap if sorted display by productId or name is necessary.

3. Scalability:

- Persist the inventory in a database for durability and large-scale usage.

- Use indexing and caching strategies for performance in persistent storage.

4. Search by Other Attributes:

- If you need to frequently search by name or price, consider maintaining auxiliary maps or indexes.

5. Exception Handling and Validation:

- Add input validation and exception handling for robustness.

6. Separation of Concerns:

- Introduce a service layer between UI and InventoryManager.

- Later integrate with a GUI or web interface for usability.

Summary:

- The HashMap-based approach is optimal for small to medium-sized in-memory inventory management systems.

- For production-grade applications, use layered architecture with persistent storage and optimized indexing.