**Exercise 1: Inventory Management System**

**1. Understanding the Problem**

**Why Data Structures and Algorithms are Essential:**

In an inventory management system, handling large inventories efficiently is critical for performance and reliability. Data structures and algorithms plays a crucial role in this:

* **Data Structures**: it is used to organize and store inventory data in a way that makes it easy to access and manipulate. The choice of data structure affects the performance of operations like adding, updating, and deleting products. Efficient data structures can improve query performance and reduce the time complexity of these operations.
* **Algorithms**: They are used to implement operations on the data structures, such as searching, inserting, and deleting.

**Types of Data Structures Suitable for the Problem:**

* **HashMap**: This is the data structure chosen in the provided code. It maps product IDs (unique identifiers) to Product objects. HashMap provides average-case constant-time complexity O(1) for operations like add, update, and delete, making it highly efficient for these tasks.
* **ArrayList**: ArrayList can be used to store products, it would be less efficient for searching and deleting operations since it requires linear time (O(n)) to find a product by ID. However, it could be suitable if you need ordered storage or frequent access by index.

**4. Analysis**

**Time Complexity Analysis of Operations:**

* **Add Product**: In the HashMap, adding a product has an average time complexity of O(1). This is because inserting a new entry in a HashMap involves computing a hash and placing the entry in the appropriate bucket.
* **Update Product**: Updating a product also has an average time complexity of O(1). Since HashMap allows direct access to the value via its key, the update operation involves simply replacing the value associated with the given key.
* **Delete Product**: Deleting a product from a HashMap has an average time complexity of O(1). Removing an entry involves locating the bucket for the given key and removing the entry from that bucket.

**Optimizing Operations:**

**HashMap**: Ensuring that the hash function distributes keys uniformly to avoid collisions, which can degrade performance. Properly managing the load factor and resizing the hash table will maintain O(1) performance.