**Exercise 1: Inventory Management System**

**Understanding the Problem:**  
Efficient handling of large inventories in warehouses is critical. Operations like adding new products, updating stock, or deleting obsolete items must be fast. If done inefficiently, system performance can degrade. Using the right data structure plays a major role in ensuring this efficiency.

**Setup and Implementation:**We define a Product class with attributes:

* productId (int)
* productName (String)
* quantity (int)
* price (double)

We use a HashMap<Integer, Product> to store the products. This allows quick access via productId.

**Java Code:**

import java.util.\*;

class Product {

int productId;

String productName;

int quantity;

double price;

Product(int id, String name, int qty, double price) {

this.productId = id;

this.productName = name;

this.quantity = qty;

this.price = price;

}

}

public class InventoryManagementSystem {

static HashMap<Integer, Product> inventory = new HashMap<>();

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("1. Add 2. Update 3. Delete 4. View 5. Exit");

int ch = sc.nextInt();

if (ch == 1) {

System.out.print("Enter id name quantity price: ");

int id = sc.nextInt();

String name = sc.next();

int qty = sc.nextInt();

double price = sc.nextDouble();

Product p = new Product(id, name, qty, price);

inventory.put(id, p);

} else if (ch == 2) {

System.out.print("Enter id to update: ");

int id = sc.nextInt();

if (inventory.containsKey(id)) {

System.out.print("Enter new quantity and price: ");

int qty = sc.nextInt();

double price = sc.nextDouble();

Product p = inventory.get(id);

p.quantity = qty;

p.price = price;

}

} else if (ch == 3) {

System.out.print("Enter id to delete: ");

int id = sc.nextInt();

inventory.remove(id);

} else if (ch == 4) {

for (Product p : inventory.values()) {

System.out.println(p.productId + " " + p.productName + " " + p.quantity + " " + p.price);

}

} else {

break;

}

}

}

}

**Time Complexity Analysis:**

* Add: O(1) on average (HashMap put operation)
* Update: O(1) (direct key access)
* Delete: O(1) (key-based removal)
* View: O(n) to iterate all products

**Optimization Discussion:**  
Using a HashMap is optimal for this problem because it offers constant-time operations for insert, update, and delete based on the productId key. If we used an ArrayList, all operations would require linear time (O(n)) because we would need to search the list for a specific product. With thousands of inventory items, HashMap significantly improves performance.