**Problem Understanding:**

Effective inventory systems must quickly store, retrieve and update product details. As data grows, the efficiency of algorithm and data structure significantly affects performance.

**Design Explanation:**

The Single Responsibility Principle is applied by separating the product data model from the inventory operations. Interface Segregation and Dependency Inversion are maintained by creating interfaces and using them to program against abstractions. The HashMap is chosen for product storage to allow constant time operations.

**Implementation:**

**Product class**

public class Product {

private String productId;

private String productName;

private int quantity;

private double price;

public Product(String productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String getProductId() {

return productId;

}

public void setProductId(String productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ",Quantity: " + quantity +", Price: " + price;

}

}

**Inventory Manager interface**

public interface IInventoryManager {

void addProduct(Product product);

void updateProduct(Product product);

void deleteProduct(String productId);

Product getProduct(St`ring productId);

}

import java.util.HashMap;

import java.util.Map;

public class InventoryManager implements IInventoryManager {

private Map<String, Product> inventory = new HashMap<>();

public void addProduct(Product product) {

inventory.put(product.getProductId(), product);

}

public void updateProduct(Product product) {

inventory.put(product.getProductId(), product);

}

public void deleteProduct(String productId) {

inventory.remove(productId);

}

public Product getProduct(String productId) {

return inventory.get(productId);

}

public void listAllProducts() {

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

System.***out***.println(product);

}

}

}

**Main class**

public class Main {

public static void main(String[] args) {

IInventoryManager inventory = new InventoryManager();

Product p1 = new Product("P001", "Pen", 100, 10.0);

Product p2 = new Product("P002", "Notebook", 50, 30.0);

inventory.addProduct(p1);

inventory.addProduct(p2);

System.out.println("All products:");

((InventoryManager) inventory).listAllProducts();

System.out.println("\nGet Product P001:");

System.out.println(inventory.getProduct("P001"));

p1.setPrice(12.5);

inventory.updateProduct(p1);

System.out.println("\nAfter update:");

((InventoryManager) inventory).listAllProducts();

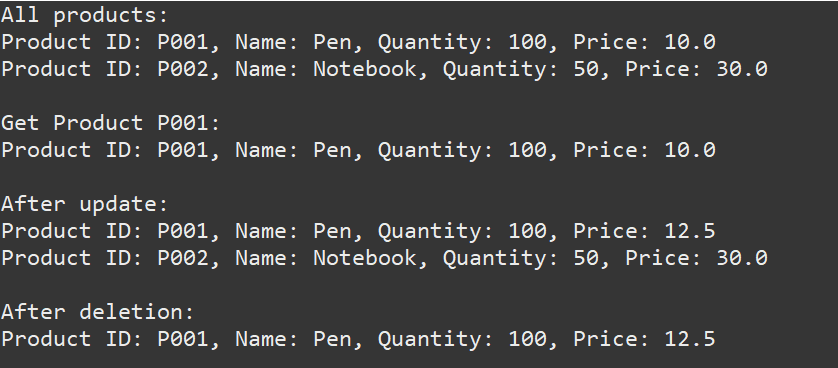
inventory.deleteProduct("P002");

System.out.println("\nAfter deletion:");

((InventoryManager) inventory).listAllProducts();

}

}

**Output:**

**Time Complexity Analysis:**

* Add: O(1)
* Update: O(1)
* Delete: O(1)
* Get: O(1)

HashMap provides average-case constant time performance. For larger systems, sharding or caching can be introduced.