**INVENTORY MANAGEMENT SYSTEM**

1. **WHY DATASTRUCTURES ARE IMPORTANT IN HANDLING LARGE INVENTORIES:**

DataStructures and algorithms are used because, inventory management must handle thousands of items quickly.

DataStructures allow:

* Fast Searching and lookups
* Efficient updates
* Quick inserting and deletion
* Minimal use with optimal performance

1. **DATASTRUCTURES SUITABLE:**

* **ArrayList-**For Ordered Access
* **HashMap-**Fast access by key

**PROGRAM:**

import java.util.\*;

public class Product{

List<Object> products=new ArrayList<>();

HashMap<Integer,List<Object>> inventory=new HashMap<>();

public void addproducts(int id,String name,int q,float p){

products=new ArrayList<>();

products.add(name);

products.add(q);

products.add(p);

inventory.put(id,products);

}

public void update(int id,int q,float p){

products=inventory.get(id);

products.remove(1);

products.remove(1);

products.add(q);

products.add(p);

inventory.put(id,products);

}

public void delete(int id){

inventory.remove(id);

}

public void display(){

System.out.println(inventory);

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

Product p=new Product();

int productid,quantity;

float price;

String productname;

System.out.println("INVENTORY MANAGEMENT\n");

System.out.println("CHOOSE 1 TO 'ADD' PRODUCT, 2 TO 'UPDATE' PRODUCT, 3 TO 'DELETE' PRODUCT, 4 TO EXIT:");

while(true){

System.out.println("Enter choice:\n");

int choice=sc.nextInt();

if(choice==1){

productid=sc.nextInt();

quantity=sc.nextInt();

price=sc.nextFloat();

productname=sc.next();

p.addproducts(productid,productname,quantity,price);

}

else if(choice==2){

System.out.println("Enter productid to be updated:");

productid=sc.nextInt();

System.out.println("Enter quantity to be updated:");

quantity=sc.nextInt();

System.out.println("Enter price to be updated:");

price=sc.nextFloat();

p.update(productid,quantity,price);

}

else if(choice==3){

System.out.println("Enter productid to be removed:");

productid=sc.nextInt();

p.delete(productid);

}

else{

break;

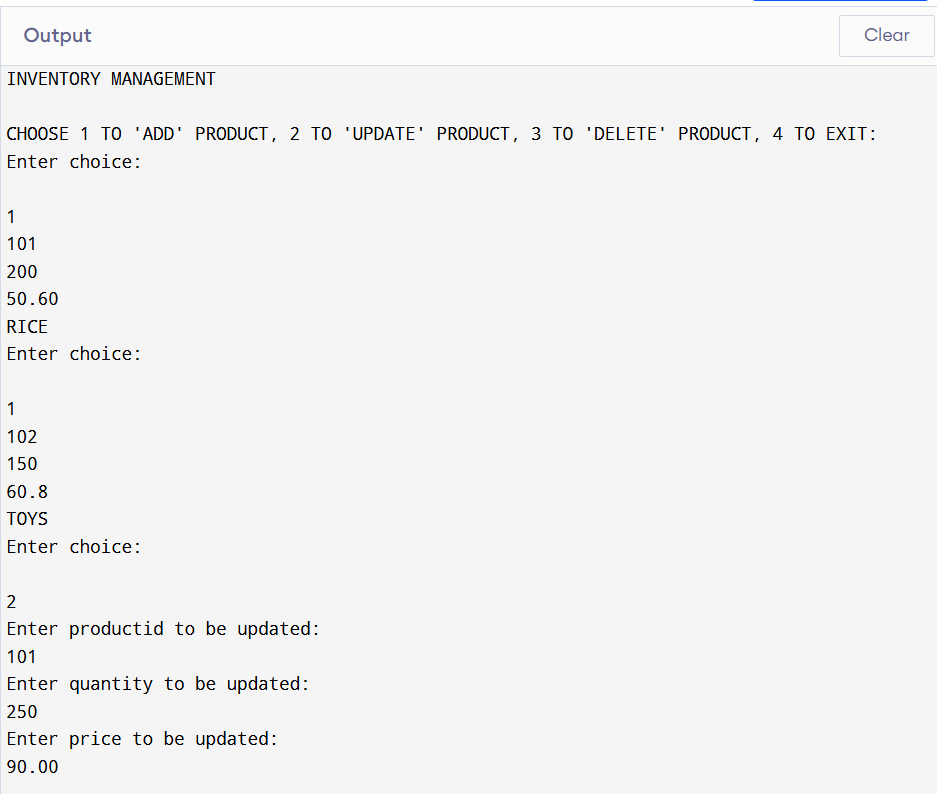
}

}

p.display();

}

}





1. **TIME COMPLEXITY:**

Since HashMap and ArrayList is used for add, update, and delete:

* **ADD: O(1)**
* **UPDATE:O(1)**
* **DELETE-O(1)**
* **DISPLAY-O(1)**

1. **OPTIMIZATION:**

* BEFORE INSERTING PRICE, VALIDATE WHETHER PRICE > 0.
* SCALABILITY SHOULD BE CONSIDERED.