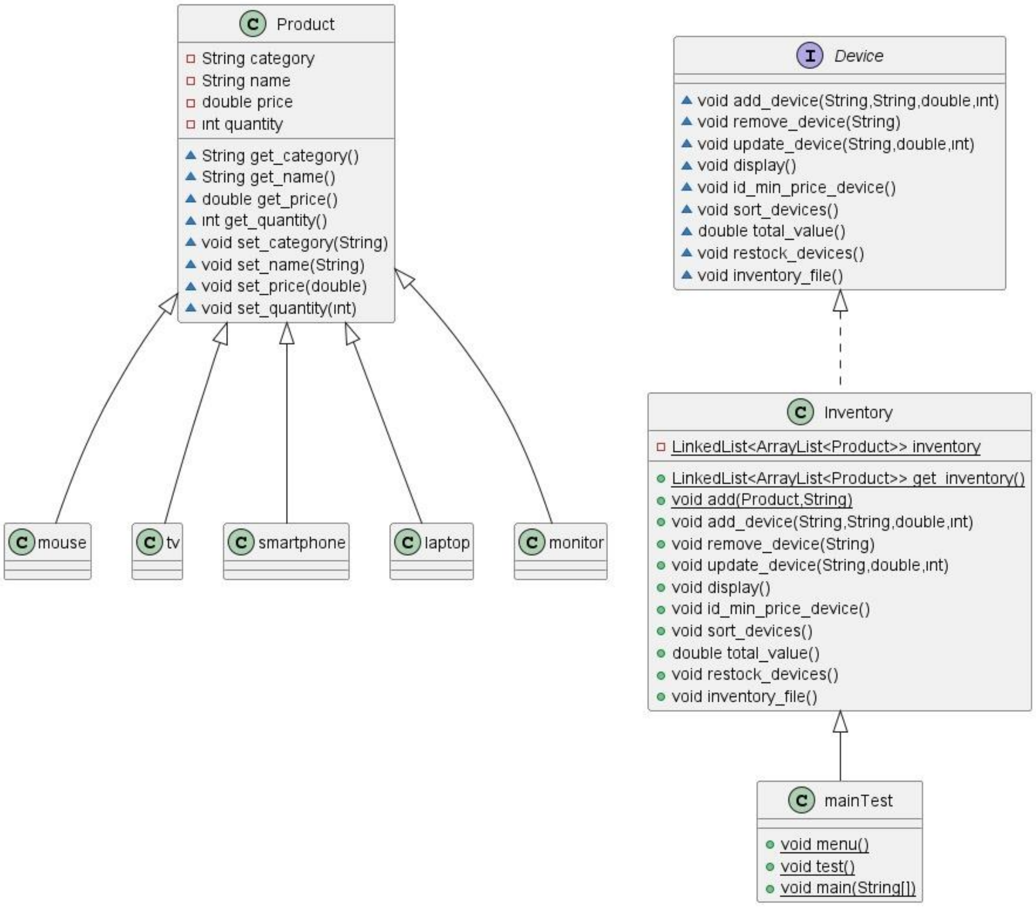
CSE 222 HW REPORT

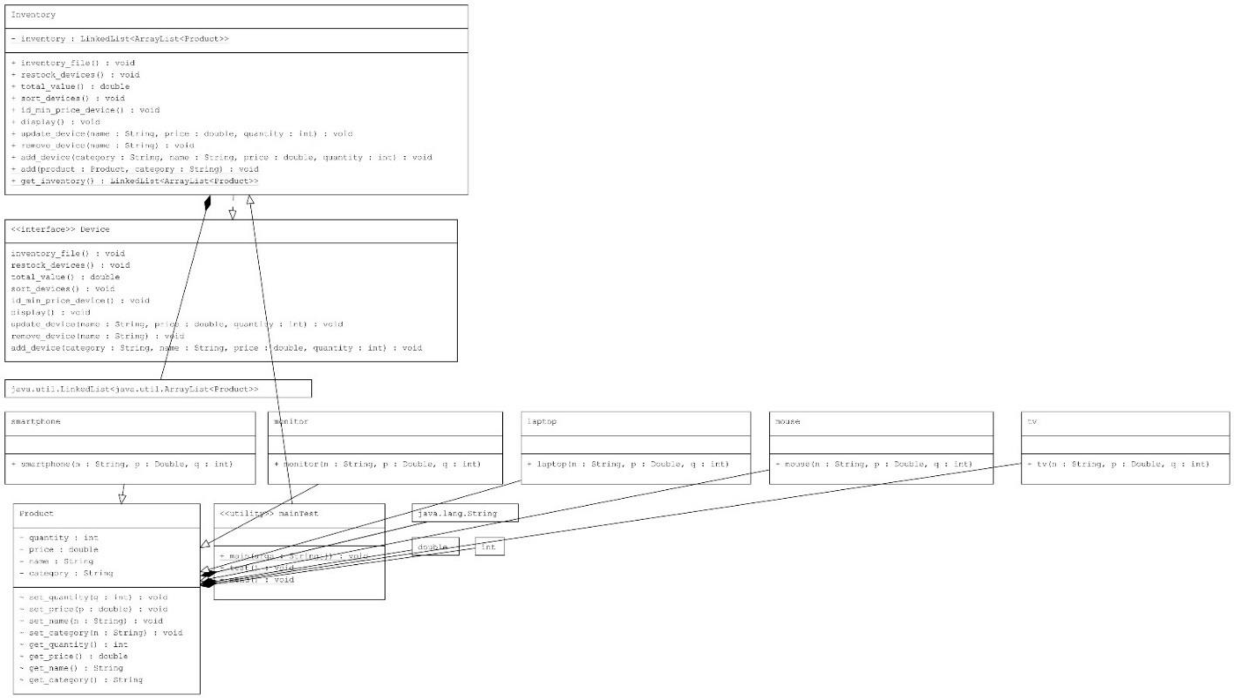
Hw3

MARCH 25, 2024

ELIFNUR KABALCI
1801042617

Class Diagrams:

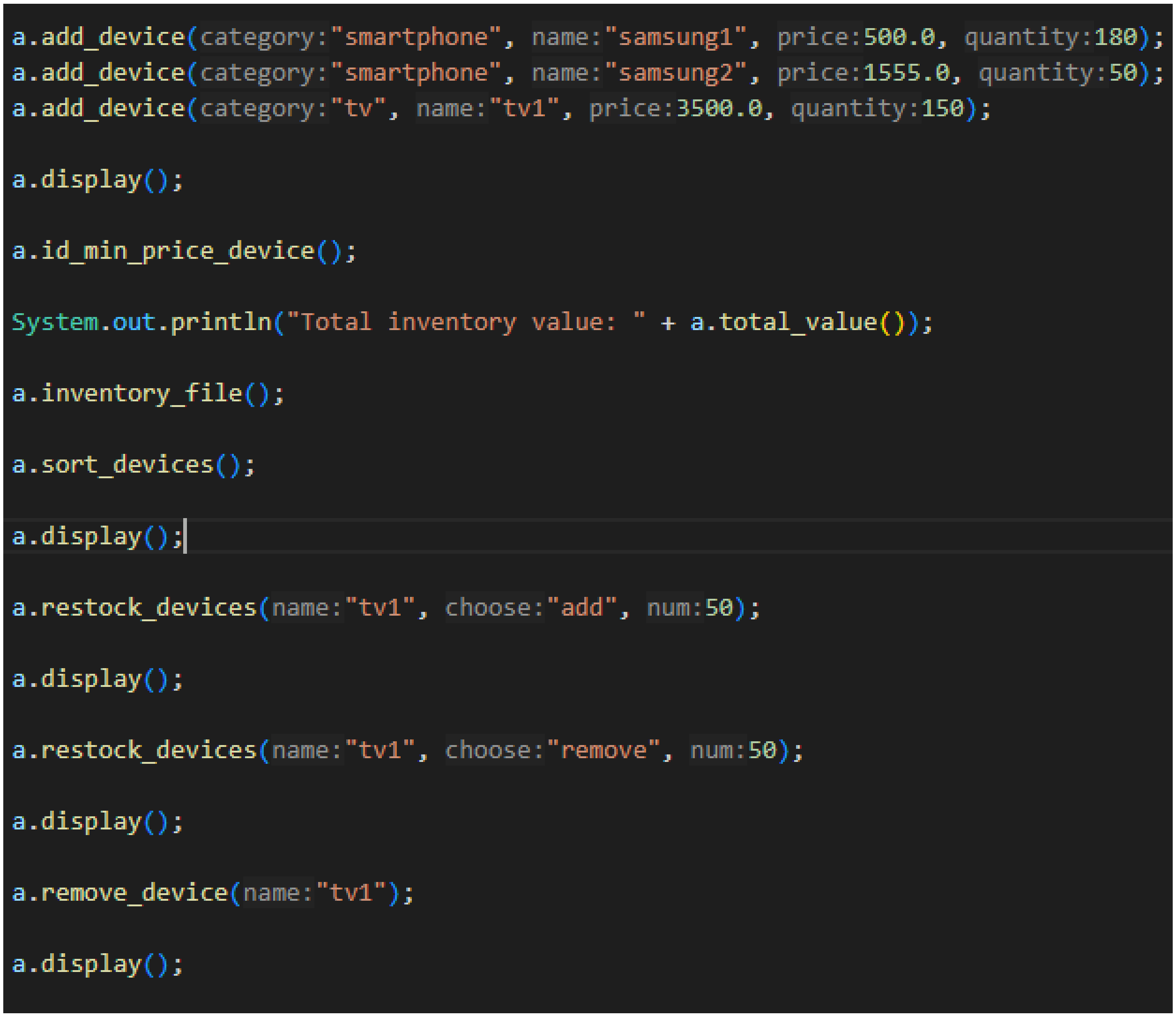




Inventory
- inventory : LinkedList<ArrayList<Product>>
+ inventory_file() : void
+ restock_devices() : void
+ total_value() : double
+ sort_devices() : void
+ id_min_price_device() : void
+ display() : void
+ update_device(name : String, price : double, quantity : int) : void
+ remove_device(name : String) : void
+ add_device(category : String, name : String, price : double, quantity : int) : void
+ add(product : Product, category : String) : void
+ get_inventory() : LinkedList<ArrayList<Product>>
<<interface>> Device
inventory_file() : void
restock_devices() : void
total_value() : double
sort_devices() : void
id_min_price_device() : void
display() : void
update_device(name : String, price : double, quantity : int) : void
remove_device(name : String) : void
add_device(category : String, name : String, price : double, quantity : int) : void
java.util.LinkedList<java.util.ArrayList<Product>>
smartphone
+ smartphone(n : String, p : Double, q : int)
monitor
+ monitor(n : String, p : Double, q : int)
laptop
+ laptop(n : String, p : Double, q : int)
mouse
+ mouse(n : String, p : Double, q : int)
tv
+ tv(n : String, p : Double, q : int)
Product
- quantity : int
- price : double
- name : String
- category : String
~ set_quantity(q : int) : void
~ set_price(p : double) : void
~ set_name(n : String) : void
~ set_category(n : String) : void
~ get_quantity() : int
~ get_price() : double
~ get_name() : String
~ get_category() : String
<<utility>> mainTest
+ main(args : String[]) : void
+ test() : void
+ menu() : void
java.lang.String
double
int

Outputs:

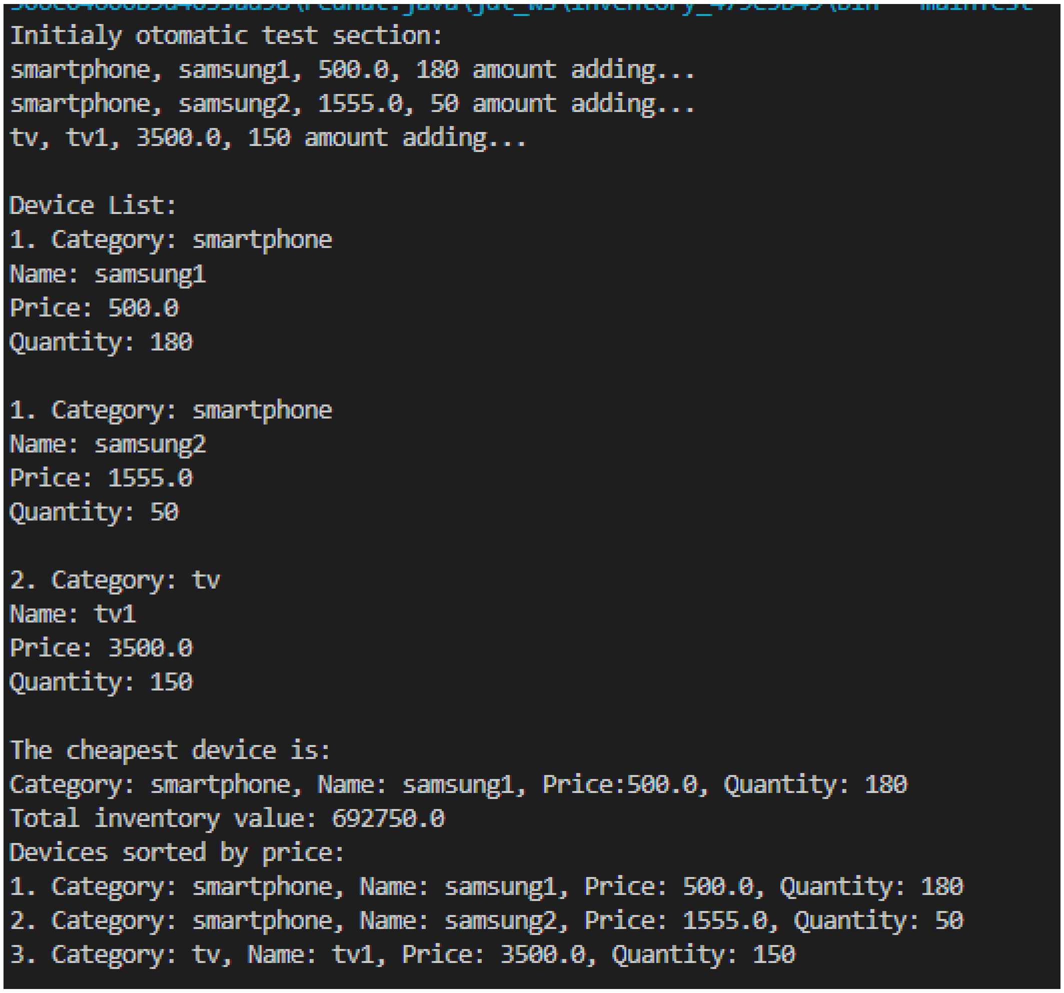
Tests:



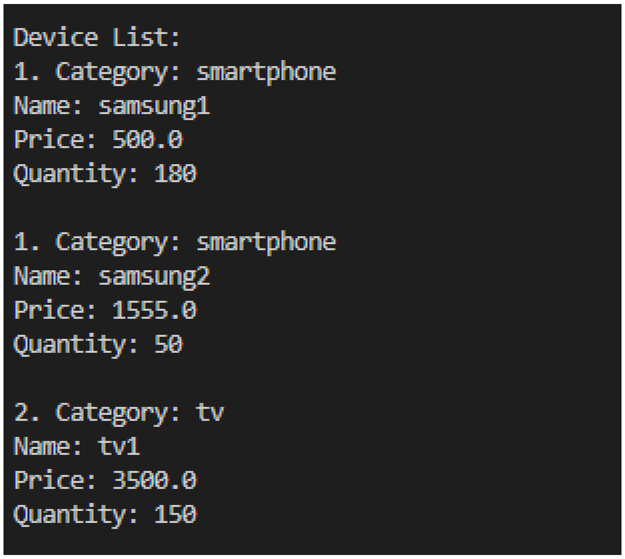
```
a.add_device(category:"smartphone", name:"samsung1", price:500.0, quantity:180);
a.add_device(category:"smartphone", name:"samsung2", price:1555.0, quantity:50);
a.add_device(category:"tv", name:"tv1", price:3500.0, quantity:150);

a.display();

a.id_min_price_device();

System.out.println("Total inventory value: " + a.total_value());

a.inventory_file();

a.sort_devices();

a.display();

a.restock_devices(name:"tv1", choose:"add", num:50);

a.display();

a.restock_devices(name:"tv1", choose:"remove", num:50);

a.display();

a.remove_device(name:"tv1");

a.display();
```

- ➔ I made these test it’s work or not.

Results of tests:

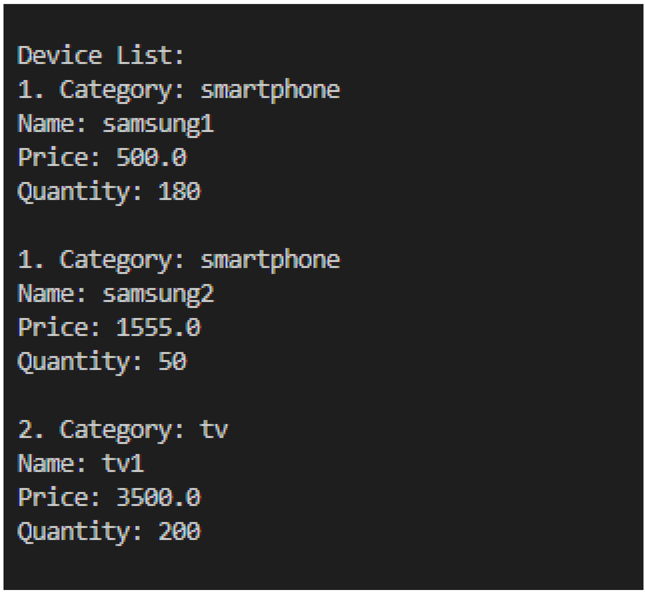


```
Initialy otomatic test section:
smartphone, samsung1, 500.0, 180 amount adding...
smartphone, samsung2, 1555.0, 50 amount adding...
tv, tv1, 3500.0, 150 amount adding...

Device List:
1. Category: smartphone
Name: samsung1
Price: 500.0
Quantity: 180

1. Category: smartphone
Name: samsung2
Price: 1555.0
Quantity: 50

2. Category: tv
Name: tv1
Price: 3500.0
Quantity: 150

The cheapest device is:
Category: smartphone, Name: samsung1, Price:500.0, Quantity: 180
Total inventory value: 692750.0
Devices sorted by price:
1. Category: smartphone, Name: samsung1, Price: 500.0, Quantity: 180
2. Category: smartphone, Name: samsung2, Price: 1555.0, Quantity: 50
3. Category: tv, Name: tv1, Price: 3500.0, Quantity: 150
```



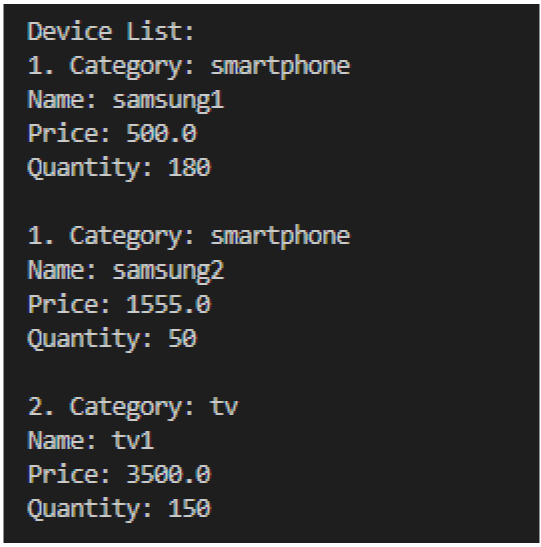
```
Device List:
1. Category: smartphone
Name: samsung1
Price: 500.0
Quantity: 180

1. Category: smartphone
Name: samsung2
Price: 1555.0
Quantity: 50

2. Category: tv
Name: tv1
Price: 3500.0
Quantity: 150
```

-> this shows the tv1 quantity 150 before adding



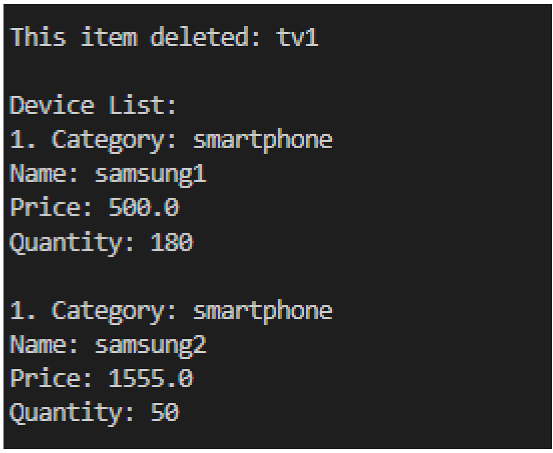
```
Device List:
1. Category: smartphone
Name: samsung1
Price: 500.0
Quantity: 180

1. Category: smartphone
Name: samsung2
Price: 1555.0
Quantity: 50

2. Category: tv
Name: tv1
Price: 3500.0
Quantity: 200
```

-> this shows the tv1 quantity is 200, after add



```
Device List:
1. Category: smartphone
Name: samsung1
Price: 500.0
Quantity: 180

1. Category: smartphone
Name: samsung2
Price: 1555.0
Quantity: 50

2. Category: tv
Name: tv1
Price: 3500.0
Quantity: 150
```

-> This shows the tv1 quantity, remove 50 element

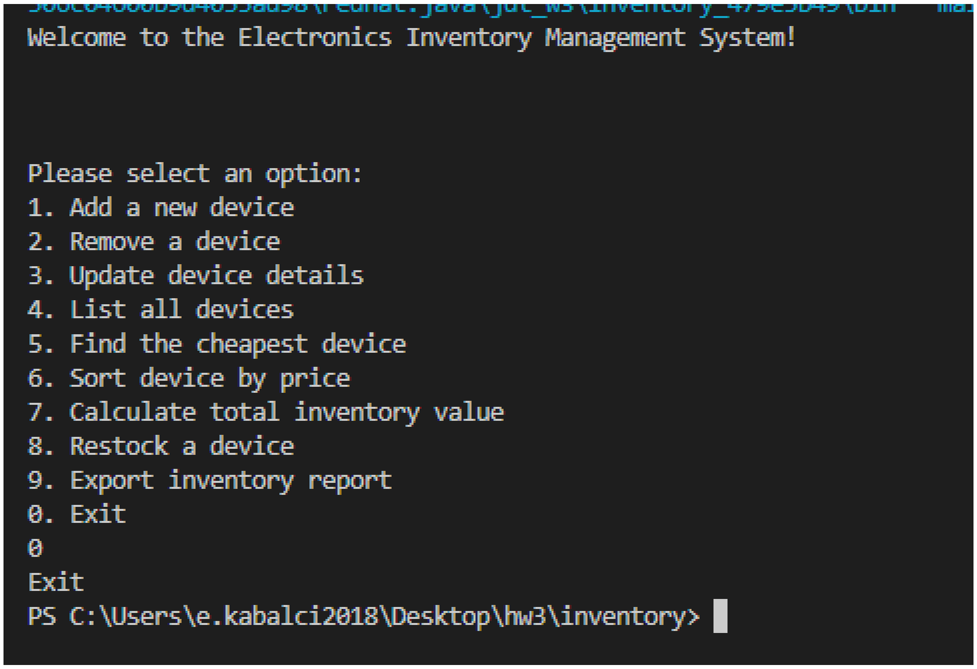


```
This item deleted: tv1

Device List:
1. Category: smartphone
Name: samsung1
Price: 500.0
Quantity: 180

1. Category: smartphone
Name: samsung2
Price: 1555.0
Quantity: 50
```

-> it shows delete the tv1 element

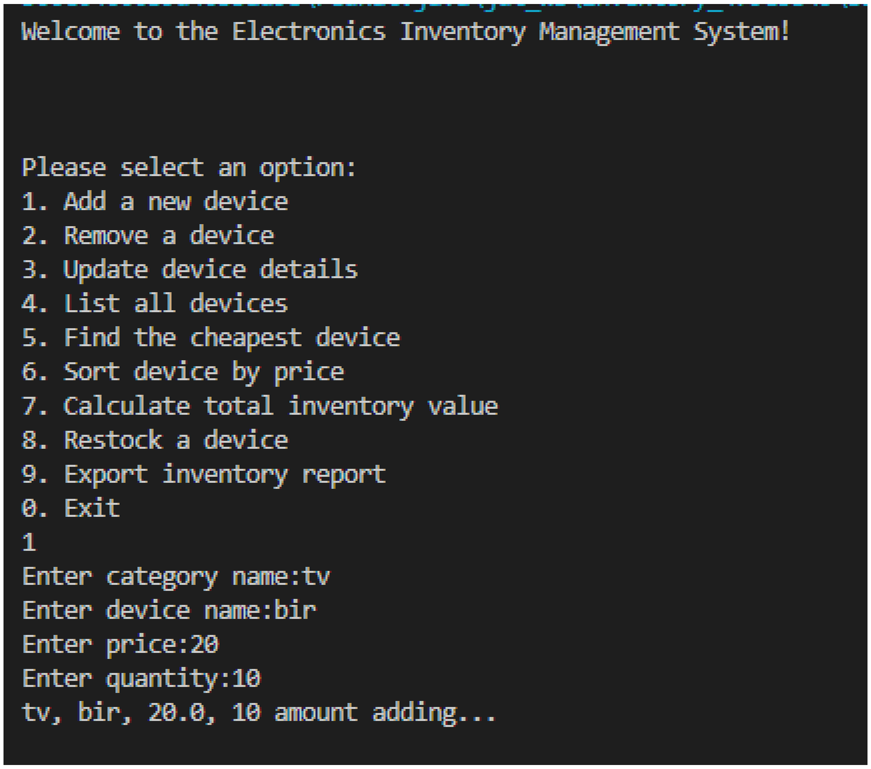
Terminal Tests:

Choose – 0:

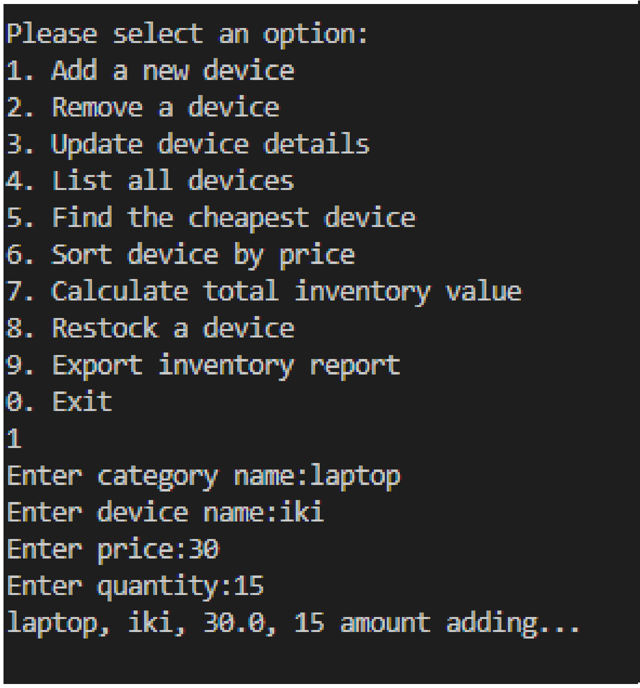


```
Welcome to the Electronics Inventory Management System!



Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
0
Exit
PS C:\Users\e.kabalci2018\Desktop\hw3\inventory> 
```

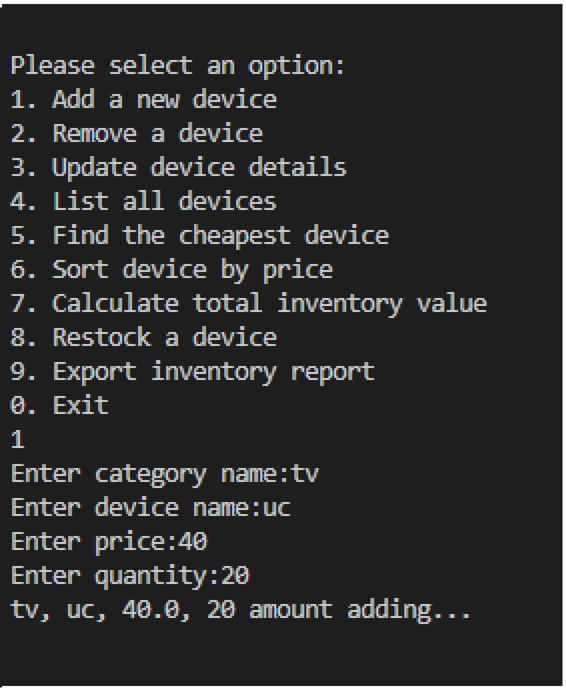
Choose - 1:



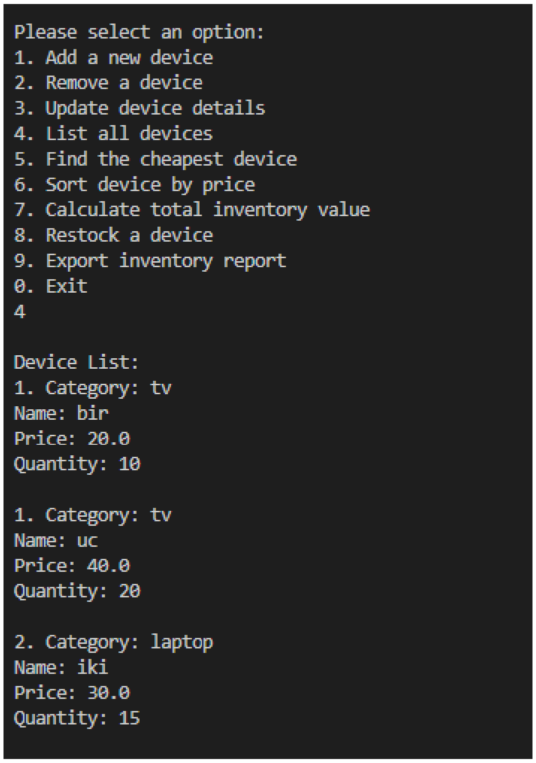
```
Welcome to the Electronics Inventory Management System!



Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
1
Enter category name:tv
Enter device name:bir
Enter price:20
Enter quantity:10
tv, bir, 20.0, 10 amount adding...
```



```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
1
Enter category name:laptop
Enter device name:iki
Enter price:30
Enter quantity:15
laptop, iki, 30.0, 15 amount adding...
```



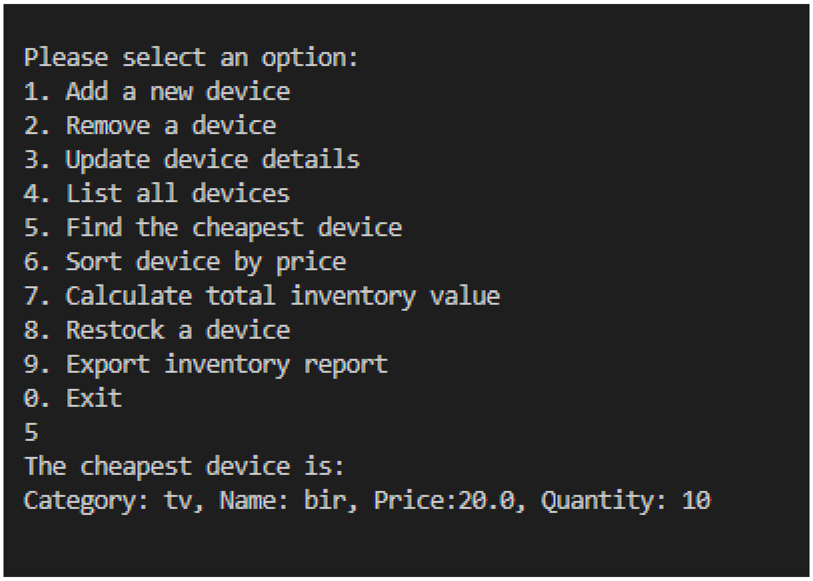
```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
1
Enter category name:tv
Enter device name:uc
Enter price:40
Enter quantity:20
tv, uc, 40.0, 20 amount adding...
```



```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
4

Device List:
1. Category: tv
Name: bir
Price: 20.0
Quantity: 10

1. Category: tv
Name: uc
Price: 40.0
Quantity: 20

2. Category: laptop
Name: iki
Price: 30.0
Quantity: 15
```

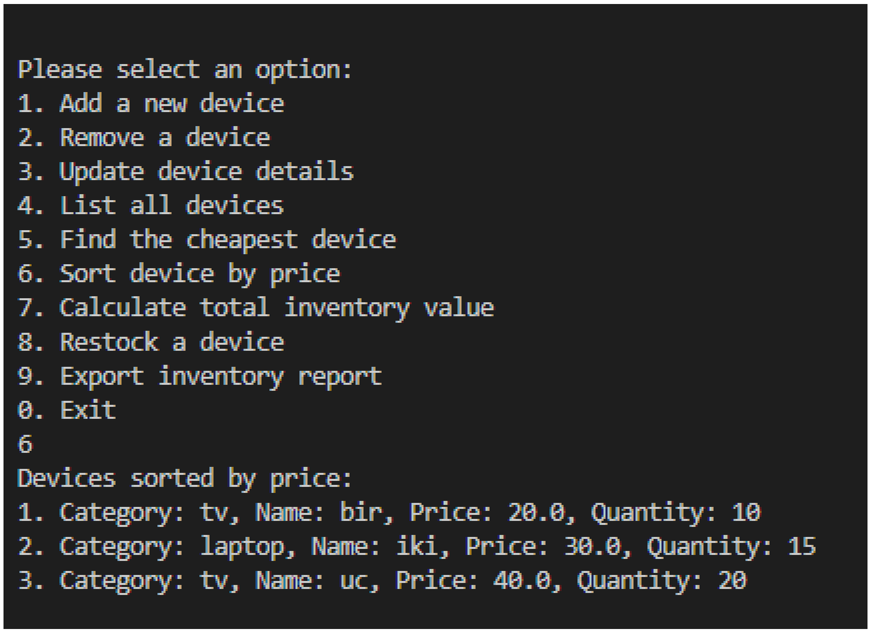
-> it shows, adding is succesfuly, and 4 is working well

Choose – 5:



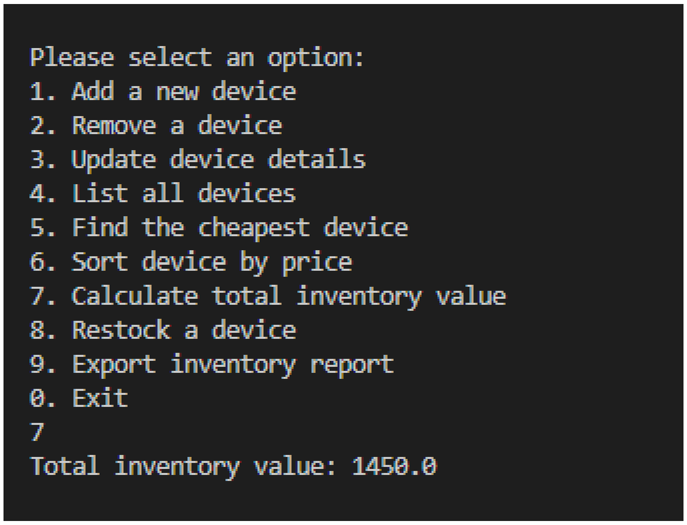
```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
5
The cheapest device is:
Category: tv, Name: bir, Price:20.0, Quantity: 10
```

Choose - 6:



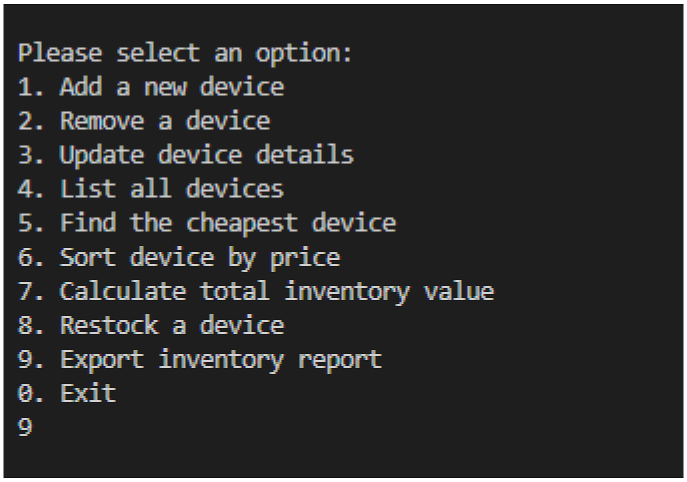
```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
6
Devices sorted by price:
1. Category: tv, Name: bir, Price: 20.0, Quantity: 10
2. Category: laptop, Name: iki, Price: 30.0, Quantity: 15
3. Category: tv, Name: uc, Price: 40.0, Quantity: 20
```

Choose – 7:

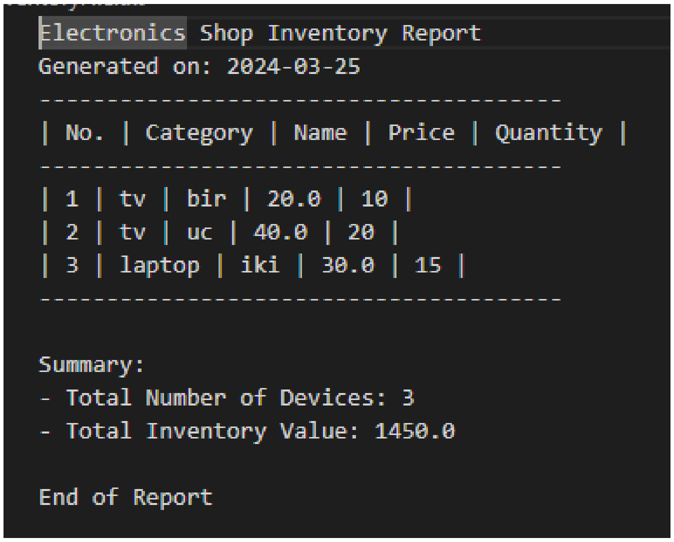


```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
7
Total inventory value: 1450.0
```

Choose – 9:



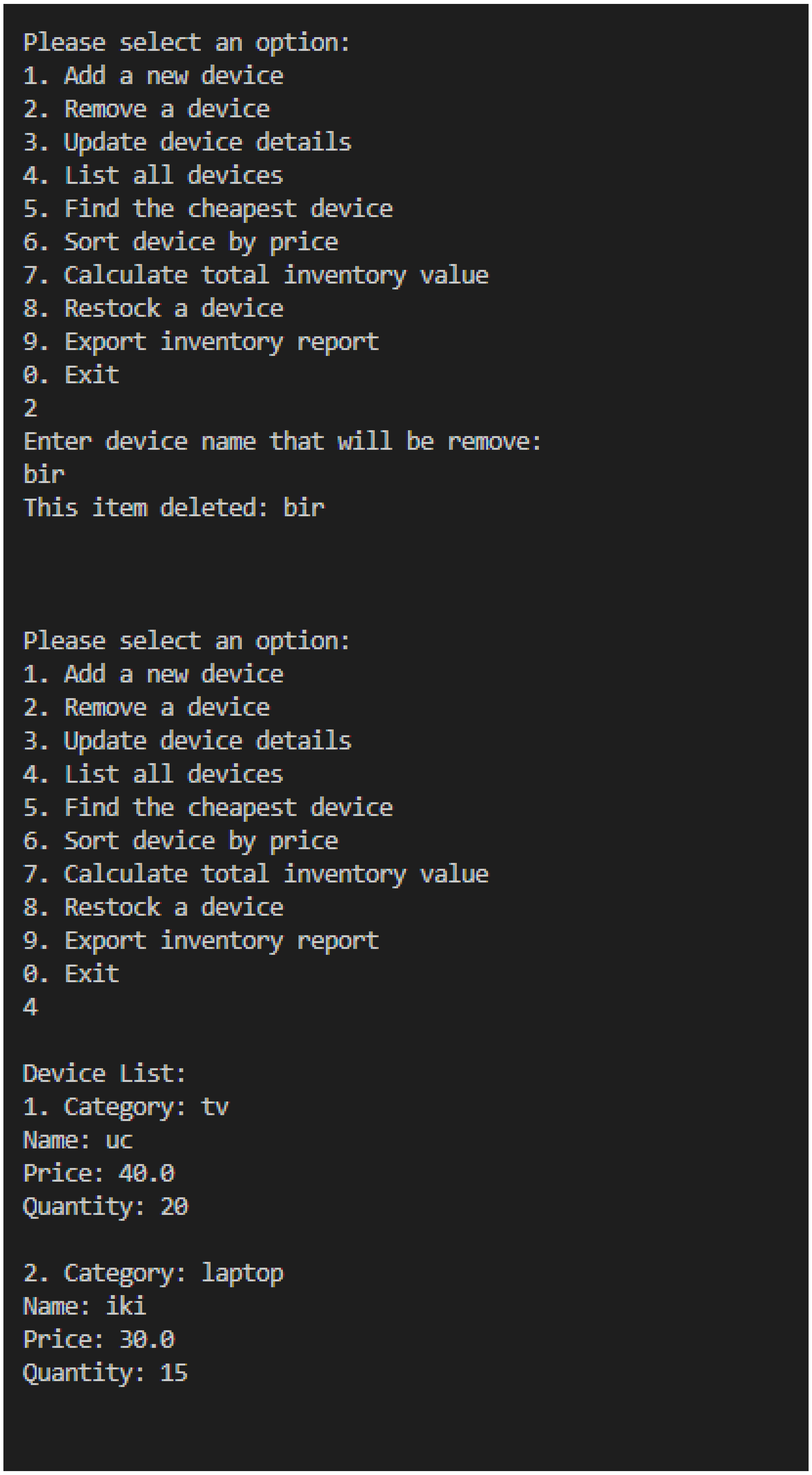
```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
9
```



```
Electronics Shop Inventory Report
Generated on: 2024-03-25
---------------------------------------
| No. | Category | Name | Price | Quantity |
---------------------------------------
| 1 | tv | bir | 20.0 | 10 |
| 2 | tv | uc | 40.0 | 20 |
| 3 | laptop | iki | 30.0 | 15 |
---------------------------------------

Summary:
- Total Number of Devices: 3
- Total Inventory Value: 1450.0

End of Report
```

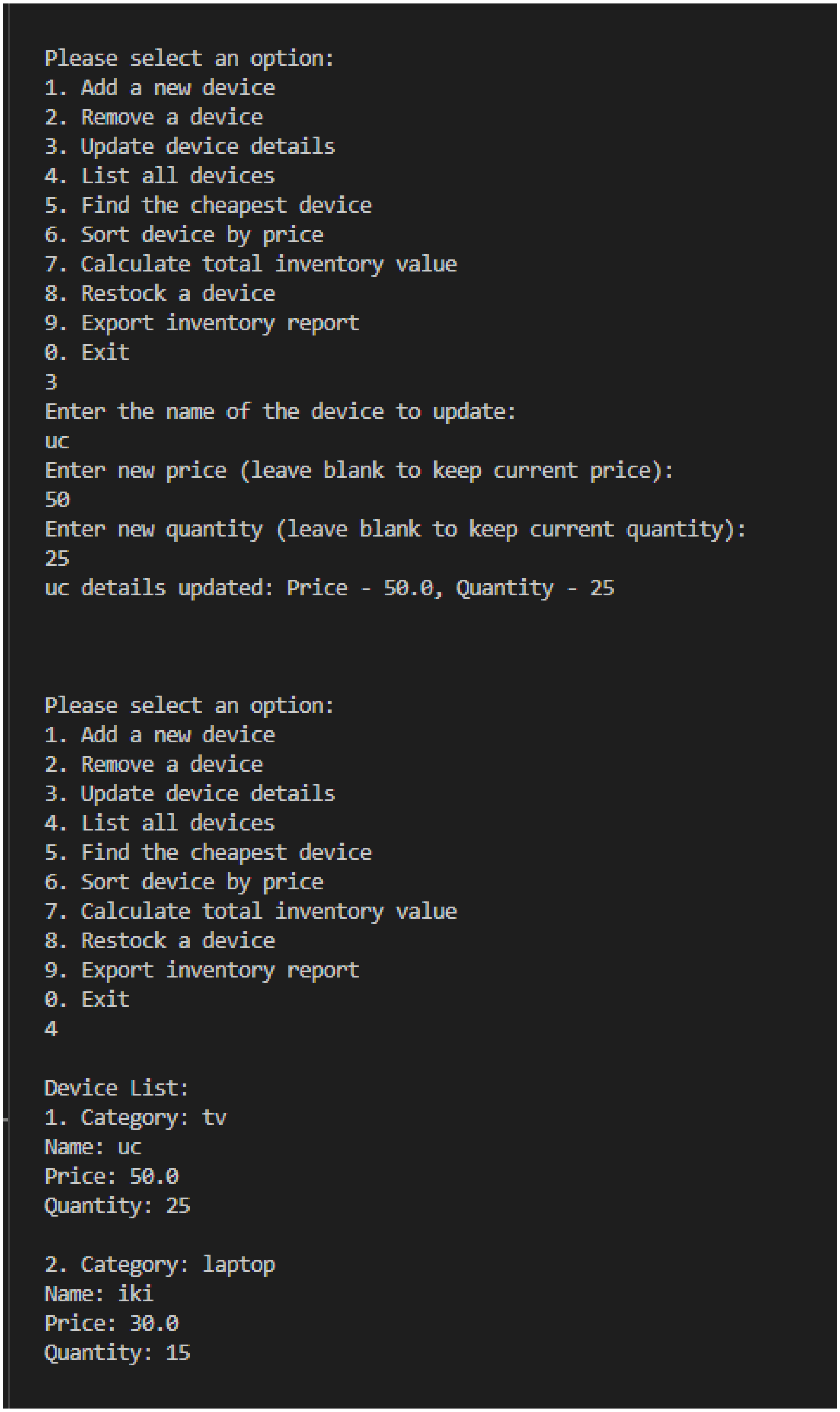
-> File

Choose – 2:



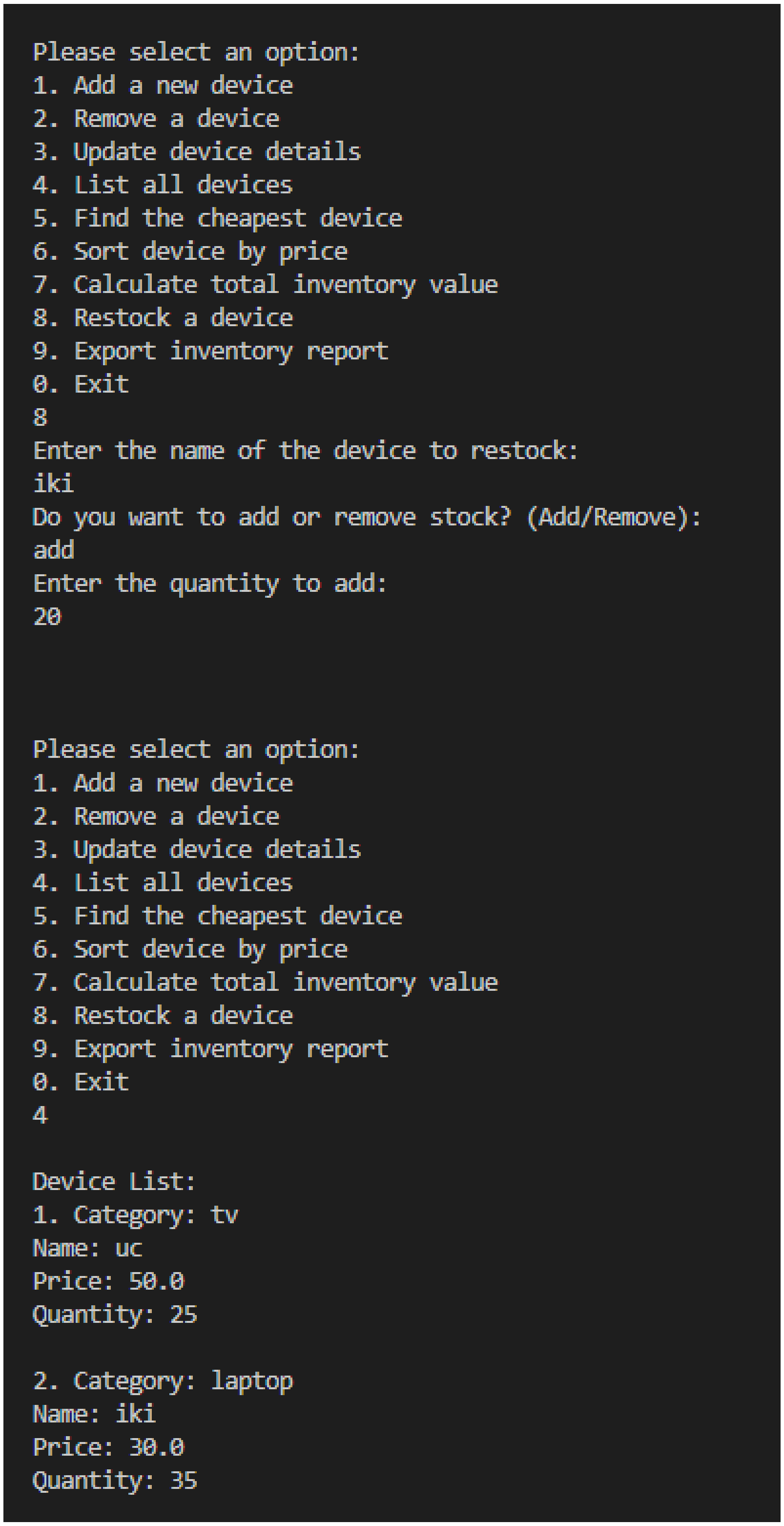
```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
2
Enter device name that will be remove:
bir
This item deleted: bir



Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
4

Device List:
1. Category: tv
Name: uc
Price: 40.0
Quantity: 20

2. Category: laptop
Name: iki
Price: 30.0
Quantity: 15
```

Choose - 3:



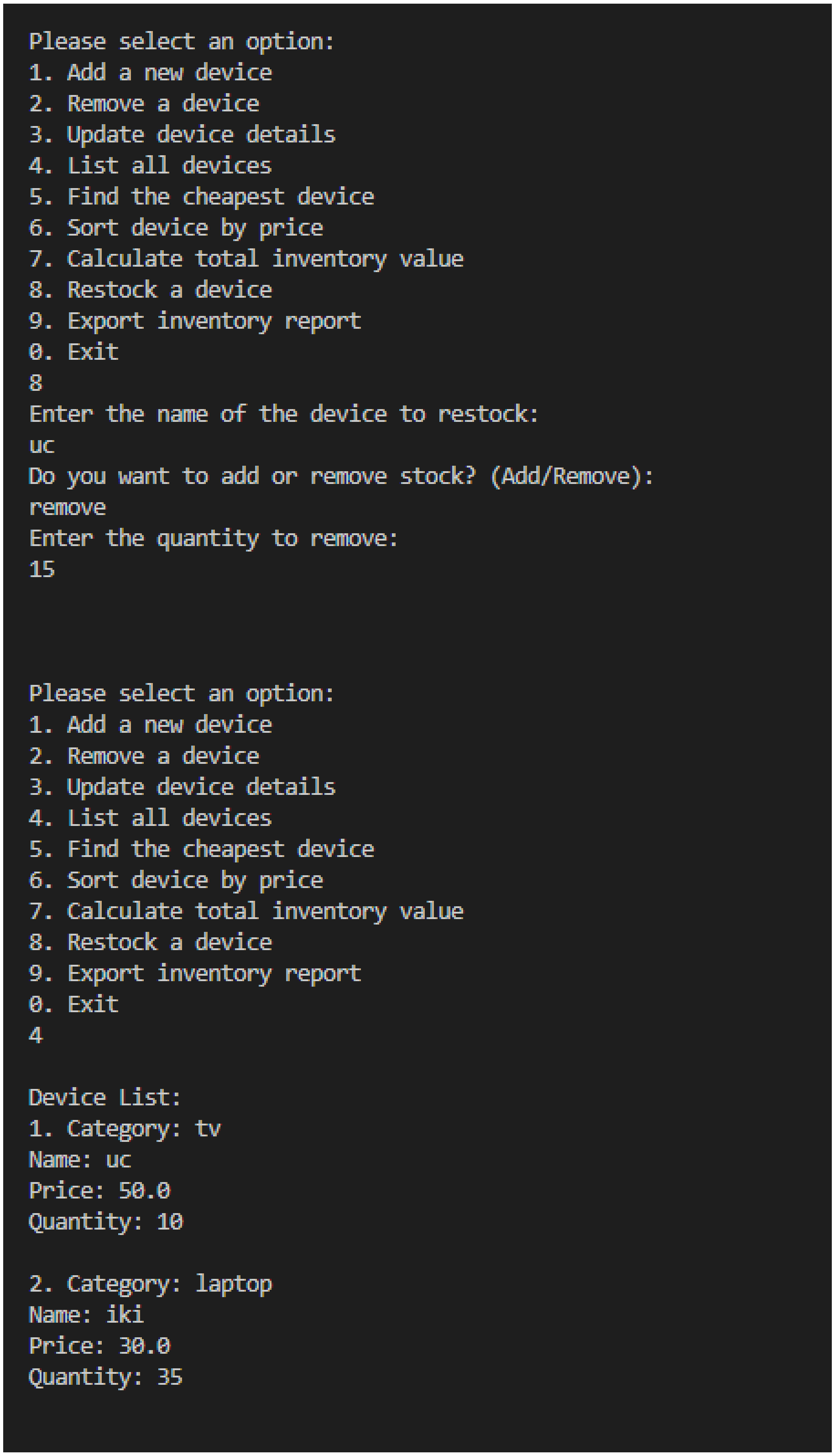
```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
3
Enter the name of the device to update:
uc
Enter new price (leave blank to keep current price):
50
Enter new quantity (leave blank to keep current quantity):
25
uc details updated: Price - 50.0, Quantity - 25



Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
4

Device List:
1. Category: tv
Name: uc
Price: 50.0
Quantity: 25

2. Category: laptop
Name: iki
Price: 30.0
Quantity: 15
```

Choose – 8 - Add:



```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
8
Enter the name of the device to restock:
iki
Do you want to add or remove stock? (Add/Remove):
add
Enter the quantity to add:
20



Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
4

Device List:
1. Category: tv
Name: uc
Price: 50.0
Quantity: 25

2. Category: laptop
Name: iki
Price: 30.0
Quantity: 35
```

Choose – 8 - Remove:



```
Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
8
Enter the name of the device to restock:
uc
Do you want to add or remove stock? (Add/Remove):
remove
Enter the quantity to remove:
15



Please select an option:
1. Add a new device
2. Remove a device
3. Update device details
4. List all devices
5. Find the cheapest device
6. Sort device by price
7. Calculate total inventory value
8. Restock a device
9. Export inventory report
0. Exit
4

Device List:
1. Category: tv
Name: uc
Price: 50.0
Quantity: 10

2. Category: laptop
Name: iki
Price: 30.0
Quantity: 35
```

Complexity Analysis:

- Product.java
- Tv.java
- Laptop.java
- Monitor.java
- Mouse.java
- Smartphone.java

This classes have only getter setter methods, So all of them time complexity is constant. **O(1).**

- mainTest.java

This class using for menu, The menu work until you closed. IF we look at the topic about worst case it’s time complexity is infinity**. O(infinity).**

- Device.java (interface)

This interface contains 9 method initial declarations. So it depends the implementation part.

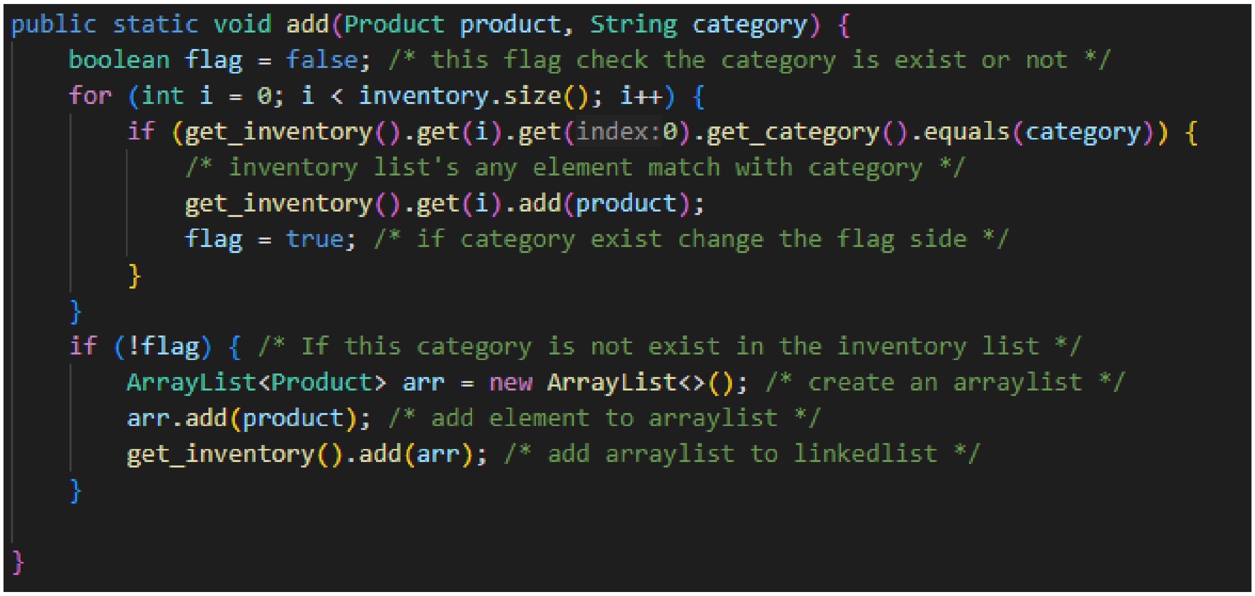
- Inventory.java

-getter inventory method is constant **– O(1)**

LinkedList get()	O(n)
Arraylist get()	O(1)
Arraylist add()	O(1)
Arraylist remove()	O(n)
Buff.write()	O(n)
Collections.sort()	O(nlogn)

Add_device()

- Initially look at the add(Product, category) method



```java
public static void add(Product product, String category) {
    boolean flag = false; /* this flag check the category is exist or not */
    for (int i = 0; i < inventory.size(); i++) {
        if (get_inventory().get(i).get(index:0).get_category().equals(category)) {
            /* inventory list's any element match with category */
            get_inventory().get(i).add(product);
            flag = true; /* if category exist change the flag side */
        }
    }
    if (!flag) { /* If this category is not exist in the inventory list */
        ArrayList<Product> arr = new ArrayList<>(); /* create an arraylist */
        arr.add(product); /* add element to arraylist */
        get_inventory().add(arr); /* add arraylist to linkedlist */
    }

}
```

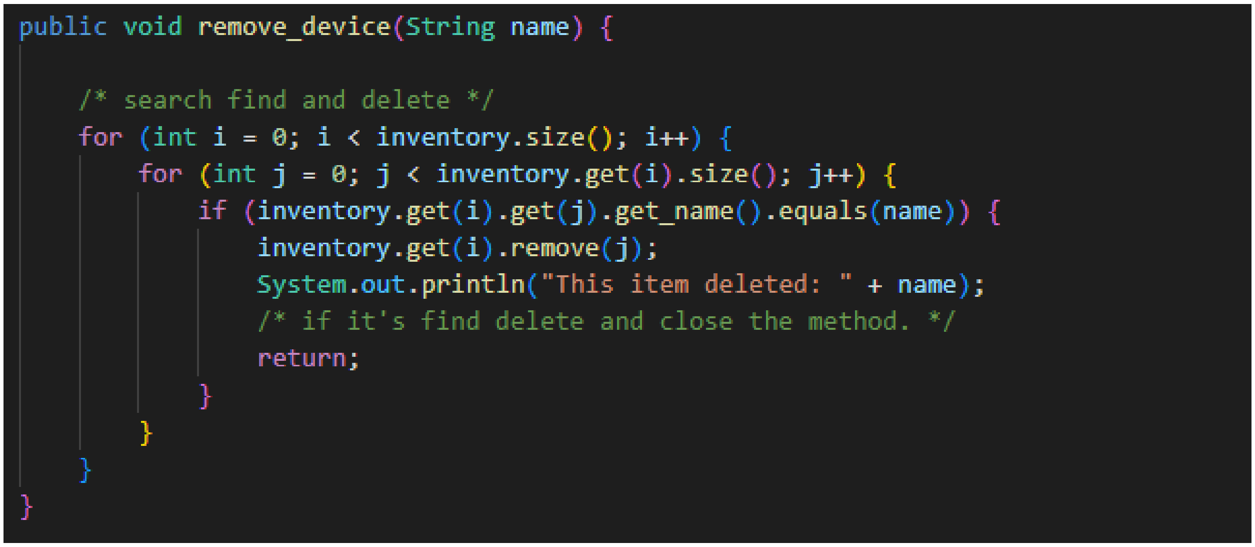
For – n times

If(Linkedlist get -n ,Arraylist get -1, equals -1){ Arraylist add -1} => This part – O(n)

If(flag) new Arraylist-1, arr.add -1, linkedlist add- n => This part- O(n)

- ⇨ Add method's time complexity is O(n) + O(n) ->O(n)
- ⇨ Add_device method call one time add method on every calling itself. So it also -**O(n)**

Remove_device()

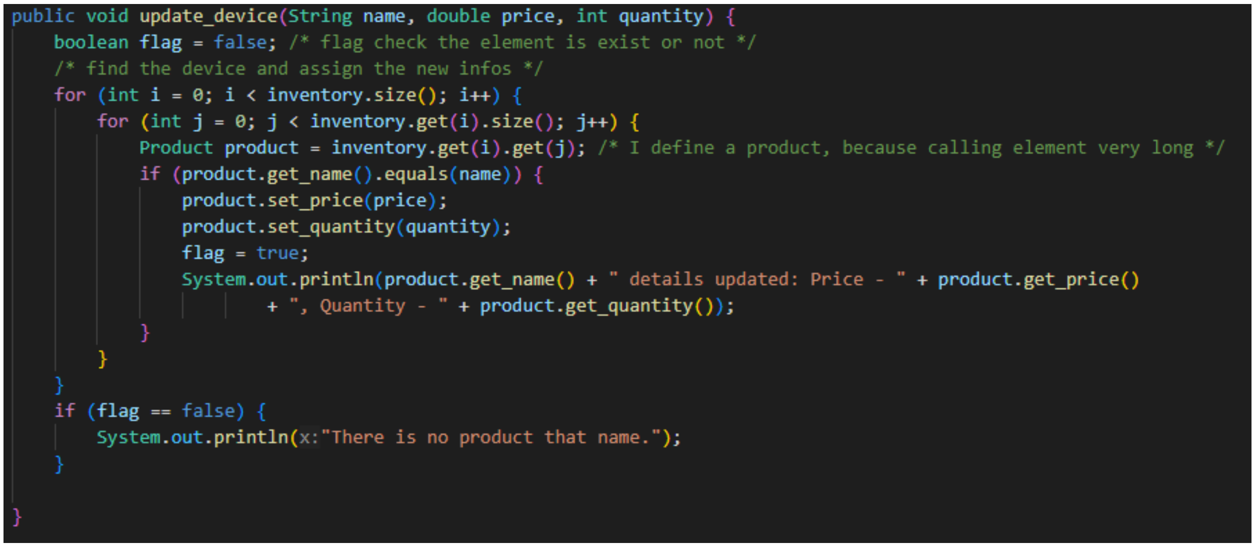


```java
public void remove_device(String name) {

    /* search find and delete */
    for (int i = 0; i < inventory.size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            if (inventory.get(i).get(j).get_name().equals(name)) {
                inventory.get(i).remove(j);
                System.out.println("This item deleted: " + name);
                /* if it's find delete and close the method. */
                return;
            }
        }
    }
}
```

- Remove method has O(n) complexity.

- Two times for loop. O(n^2) complexity. If we add the ready method's complexities also, Out total complexity become **O(n^3).** But, if we regard this, we take only for's so become **O(n^2)**

Update_device()

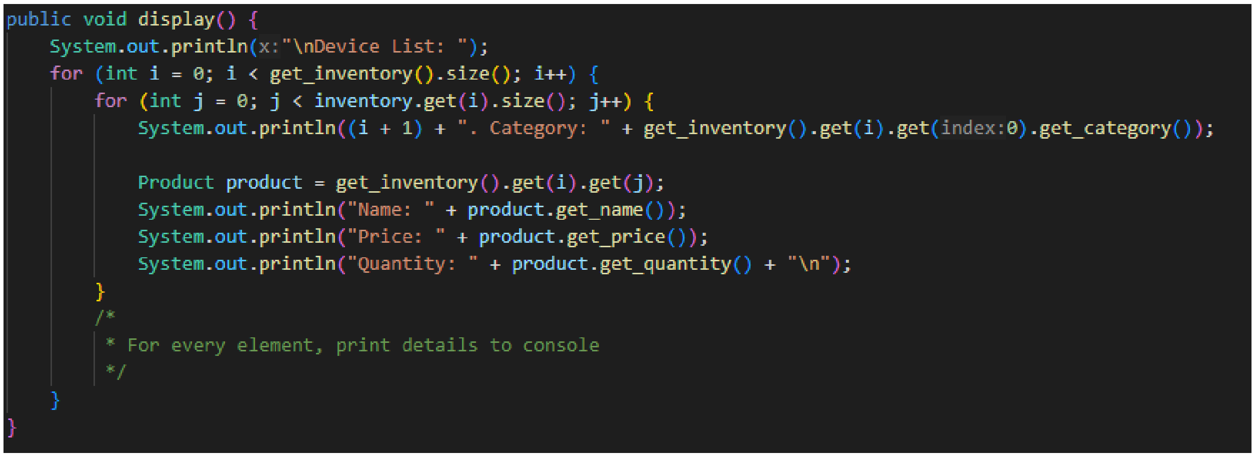


```java
public void update_device(String name, double price, int quantity) {
    boolean flag = false; /* flag check the element is exist or not */
    /* find the device and assign the new infos */
    for (int i = 0; i < inventory.size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            Product product = inventory.get(i).get(j); /* I define a product, because calling element very long */
            if (product.get_name().equals(name)) {
                product.set_price(price);
                product.set_quantity(quantity);
                flag = true;
                System.out.println(product.get_name() + " details updated: Price - " + product.get_price()
                        + ", Quantity - " + product.get_quantity());
            }
        }
    }
    if (flag == false) {
        System.out.println(x:"There is no product that name.");
    }

}
```

-getter setter calling, comparisons, println all of them have O(1) time complexity.

- This method have **O(n^2),** because of 2 layer for.

Display()

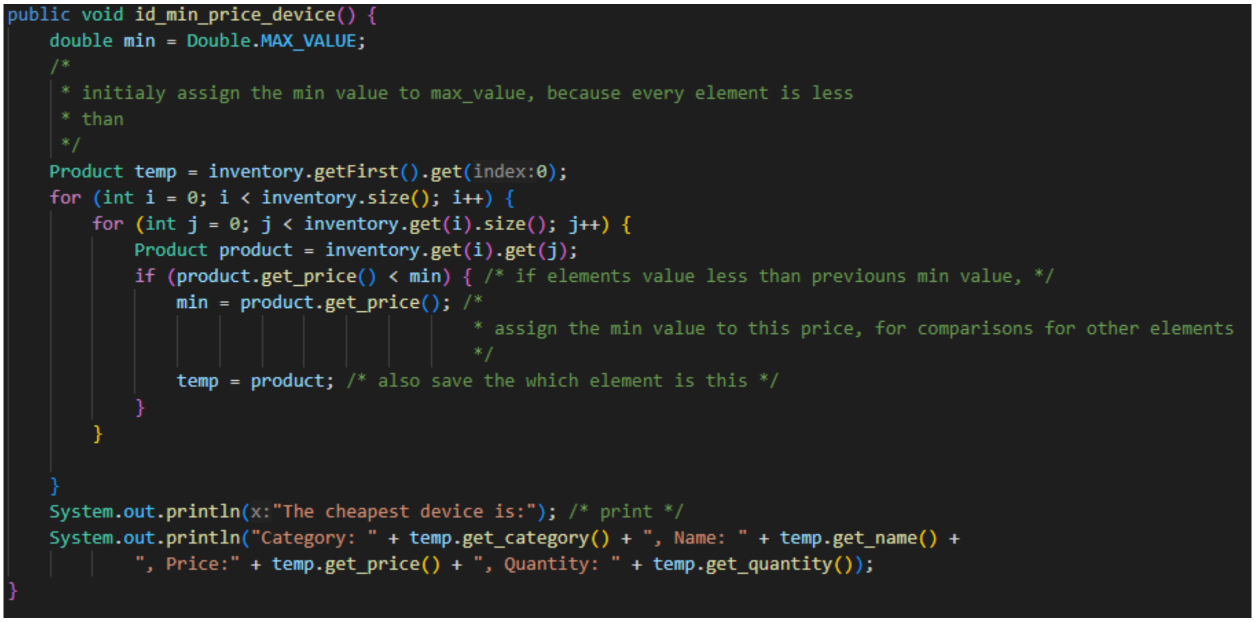


```java
public void display() {
    System.out.println(x:"\nDevice List: ");
    for (int i = 0; i < get_inventory().size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            System.out.println((i + 1) + ". Category: " + get_inventory().get(i).get(index:0).get_category());

            Product product = get_inventory().get(i).get(j);
            System.out.println("Name: " + product.get_name());
            System.out.println("Price: " + product.get_price());
            System.out.println("Quantity: " + product.get_quantity() + "\n");
        }
        /*
         * For every element, print details to console
         */
    }
}
```

- println's have O(1) complexity.

- This method have **O(n^2)** complexity, because of 2 layer for.

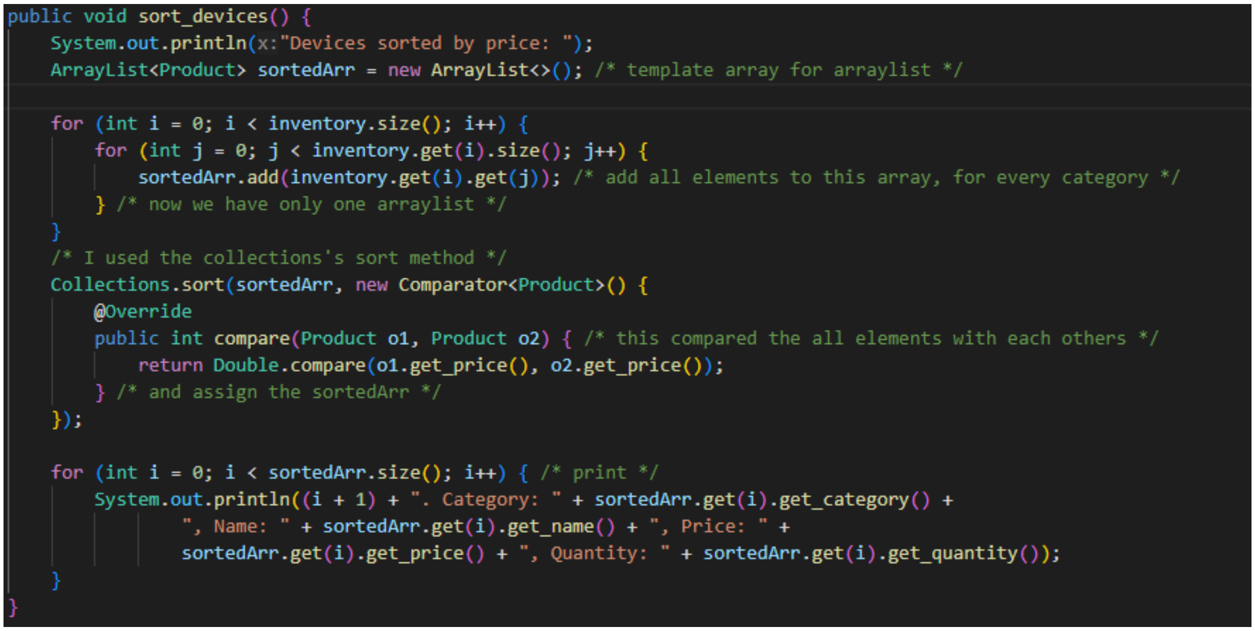
Id_min_price_device()



```
public void id_min_price_device() {
    double min = Double.MAX_VALUE;
    /*
     * initialy assign the min value to max_value, because every element is less
     * than
     */
    Product temp = inventory.getFirst().get(index:0);
    for (int i = 0; i < inventory.size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            Product product = inventory.get(i).get(j);
            if (product.get_price() < min) { /* if elements value less than previouns min value, */
                min = product.get_price(); /*
                                            * assign the min value to this price, for comparisons for other elements
                                            */
                temp = product; /* also save the which element is this */
            }
        }

    }
    System.out.println(x:"The cheapest device is:"); /* print */
    System.out.println("Category: " + temp.get_category() + ", Name: " + temp.get_name() +
            ", Price:" + temp.get_price() + ", Quantity: " + temp.get_quantity());
}
```

- In this method, bigger complexity part is for. Others have O(1).
- Time complexity is **O(n^2)** because of for loops.

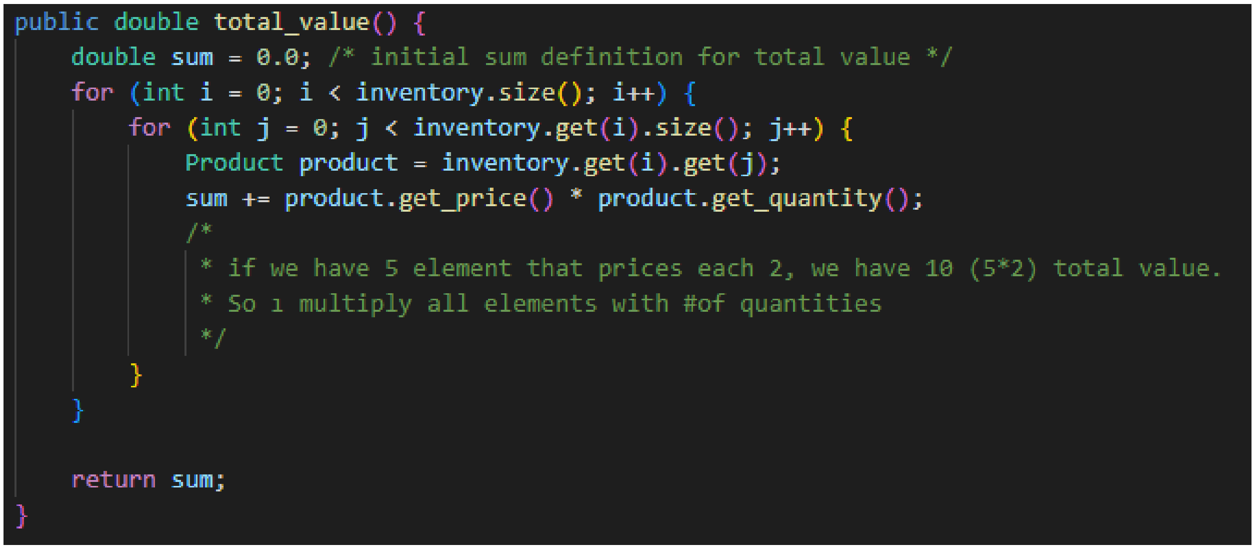
Sort_device()



```
public void sort_devices() {
    System.out.println(x:"Devices sorted by price: ");
    ArrayList<Product> sortedArr = new ArrayList<>(); /* template array for arraylist */

    for (int i = 0; i < inventory.size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            sortedArr.add(inventory.get(i).get(j)); /* add all elements to this array, for every category */
        } /* now we have only one arraylist */
    }
    /* I used the collections's sort method */
    Collections.sort(sortedArr, new Comparator<Product>() {
        @Override
        public int compare(Product o1, Product o2) { /* this compared the all elements with each others */
            return Double.compare(o1.get_price(), o2.get_price());
        } /* and assign the sortedArr */
    });

    for (int i = 0; i < sortedArr.size(); i++) { /* print */
        System.out.println((i + 1) + ". Category: " + sortedArr.get(i).get_category() +
                ", Name: " + sortedArr.get(i).get_name() + ", Price: " +
                sortedArr.get(i).get_price() + ", Quantity: " + sortedArr.get(i).get_quantity());
    }
}
```

- First for loops have O(n^2)
- Collections.sort have O(nlogn)
- Second for loop have O(n)
- O(n^2) + O(nlogn) + O(n) -> **O(n^2)**

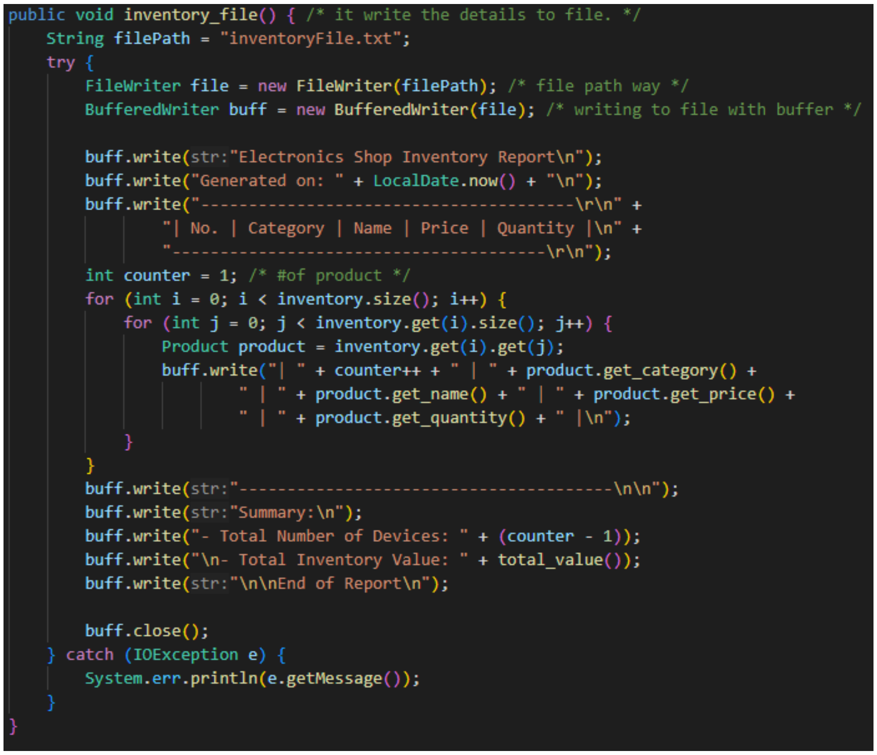
Total_value()



```java
public double total_value() {
    double sum = 0.0; /* initial sum definition for total value */
    for (int i = 0; i < inventory.size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            Product product = inventory.get(i).get(j);
            sum += product.get_price() * product.get_quantity();
            /*
             * if we have 5 element that prices each 2, we have 10 (5*2) total value.
             * So ı multiply all elements with #of quantities
             */
        }
    }

    return sum;
}
```

- ➔ Summation and multiplications are O(1)
- ➔ This method’s time complexity is **O(n^2)** because of for’s.

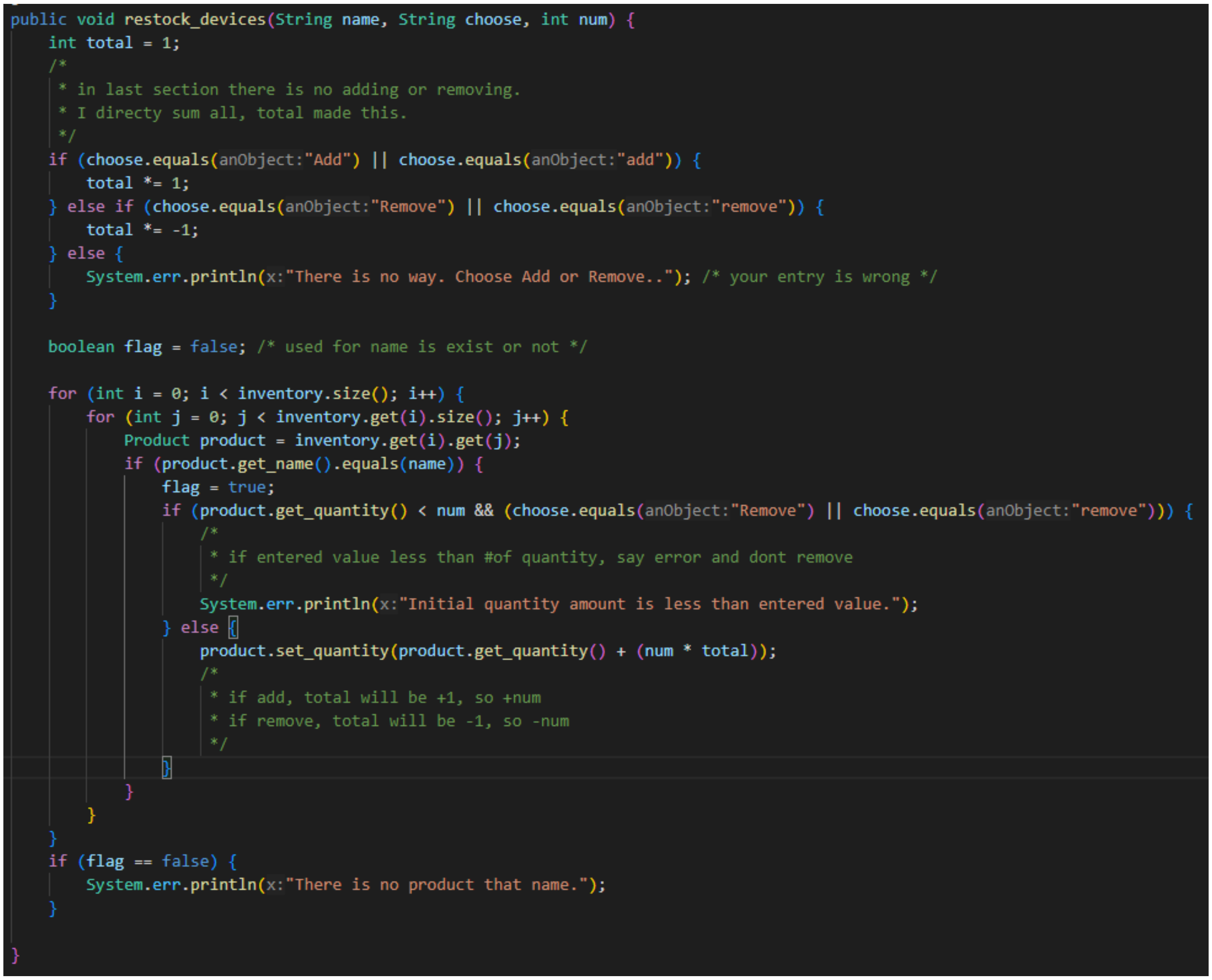
Inventory_file()



```java
public void inventory_file() { /* it write the details to file. */
    String filePath = "inventoryFile.txt";
    try {
        FileWriter file = new FileWriter(filePath); /* file path way */
        BufferedWriter buff = new BufferedWriter(file); /* writing to file with buffer */

        buff.write(str:"Electronics Shop Inventory Report\n");
        buff.write("Generated on: " + LocalDate.now() + "\n");
        buff.write("--------------------------------------\r\n" +
                "| No. | Category | Name | Price | Quantity |\n" +
                "--------------------------------------\r\n");
        int counter = 1; /* #of product */
        for (int i = 0; i < inventory.size(); i++) {
            for (int j = 0; j < inventory.get(i).size(); j++) {
                Product product = inventory.get(i).get(j);
                buff.write("| " + counter++ + " | " + product.get_category() +
                        " | " + product.get_name() + " | " + product.get_price() +
                        " | " + product.get_quantity() + " |\n");
            }
        }
        buff.write(str:"--------------------------------------\n\n");
        buff.write(str:"Summary:\n");
        buff.write("- Total Number of Devices: " + (counter - 1));
        buff.write("\n- Total Inventory Value: " + total_value());
        buff.write(str:"\n\nEnd of Report\n");

        buff.close();
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}
```

- File Writer has O(1), BufferedWriter has O(1), Buff.writer has O(n),
- For part has 2 layer for, if we add the buffer writer complexity, it will become **O(n^3),** if it is not become **O(n^2).**

Restock_devices()



```
public void restock_devices(String name, String choose, int num) {
    int total = 1;
    /*
     * in last section there is no adding or removing.
     * I directy sum all, total made this.
     */
    if (choose.equals(anObject:"Add") || choose.equals(anObject:"add")) {
        total *= 1;
    } else if (choose.equals(anObject:"Remove") || choose.equals(anObject:"remove")) {
        total *= -1;
    } else {
        System.err.println(x:"There is no way. Choose Add or Remove.."); /* your entry is wrong */
    }

    boolean flag = false; /* used for name is exist or not */

    for (int i = 0; i < inventory.size(); i++) {
        for (int j = 0; j < inventory.get(i).size(); j++) {
            Product product = inventory.get(i).get(j);
            if (product.get_name().equals(name)) {
                flag = true;
                if (product.get_quantity() < num && (choose.equals(anObject:"Remove") || choose.equals(anObject:"remove"))) {
                    /*
                     * if entered value less than #of quantity, say error and dont remove
                     */
                    System.err.println(x:"Initial quantity amount is less than entered value.");
                } else {
                    product.set_quantity(product.get_quantity() + (num * total));
                    /*
                     * if add, total will be +1, so +num
                     * if remove, total will be -1, so -num
                     */
                }
            }
        }
    }
    if (flag == false) {
        System.err.println(x:"There is no product that name.");
    }

}
```

- ➔ Comparisons, adding, multiplications have O(1)
- ➔ This method’s complexity is **O(n^2)** because of for’s.