GEBZE TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CSE222/505 – Spring 2021

Homework 3 Report

Yakup Talha Yolcu

1801042609

PART 1

1.INTRODUCTION

1.1 PROBLEM DEFINITON

The automation system for a furniture company has users such as admins, branch employees and customers.

Admins can add and remove branch, branch employees. They can also ask to branch employee whether product supply is needed.

Branch Employee can inform about product need. They can add and remove any product. They can create subscription of the Person that want to be a Customer. They can make sale, view previous order of the any customer and add new order to this section. They can check the stocks of any product.

Customers can login to the system, list the products of the branches, list all products of the company and list the products with specific color name and model name. They can buy online and buy from shop. They can view previous orders themselves.

1.2 SYSTEM REQUIREMENTS

Firstly, we need to create a company. When creating company we need nothing to give as a parameter. In company's constructor some number of admin, branch are created by default. In branch, some number of Branch employee and product are created by default. When Company is created, some branches, admins, branch employees are added default.

```
Company c1=new Company();
```

Then a person should be created with name,surname,e mail and password

```
Person p1=new Person( n: "Talha", s: "Yolcu", e: "E_Mail", p: "Passwd");

Person p2=new Person( n: "Yakup", s: "Yolcu", e: "EMAIL", p: "PASSWD");
```

Admin of the given branch should add branch employee to the given branch

```
c1.getAdmin( index: 0).add_branch_employee( index: 0);
```

Person subscribes to the system

Customer logins to the system

```
Customer newc=p2.subscribe(c1.get_branch( index: 0).get_branchemployee( index: 0));
newc.login(newc.getCustomer_number(),newc.getE_mail(),newc.getPassword());
```

Instead of person's subscription, Branch Employee can subscribe a person

```
Customer customer=c1.get_branch( index: 0).get_branchemployee( index: 0).create_subscription(p1);
```

Customer lists the branches branch employee adds the products to the branches

Branch employee removes product from branch

```
customer.list_branch(c1, index: 0);
c1.get_branch( index: 0).get_branchemployee( index: 0).add_product(new OfficeChair( model: "OCHAIR_2", color: "BLUE", stock: 4));
customer.list_branch(c1, index: 0);
c1.get_branch( index: 0).get_branchemployee( index: 0).remove_product(new OfficeChair( model: "OCHAIR_1", color: "BLACK", stock: 2));
customer.list_branch(c1, index: 0);
```

Customer lists all the products of the company

Admin removes branch employee from the branch

```
customer.list_all(c1);
c1.getAdmin( index: 0).remove_branch_employee( index: 0);
```

Admin removes branch

```
c1.getAdmin( index: 0).remove_branch( index: 0);
```

Admin removes branch and customer search for a product

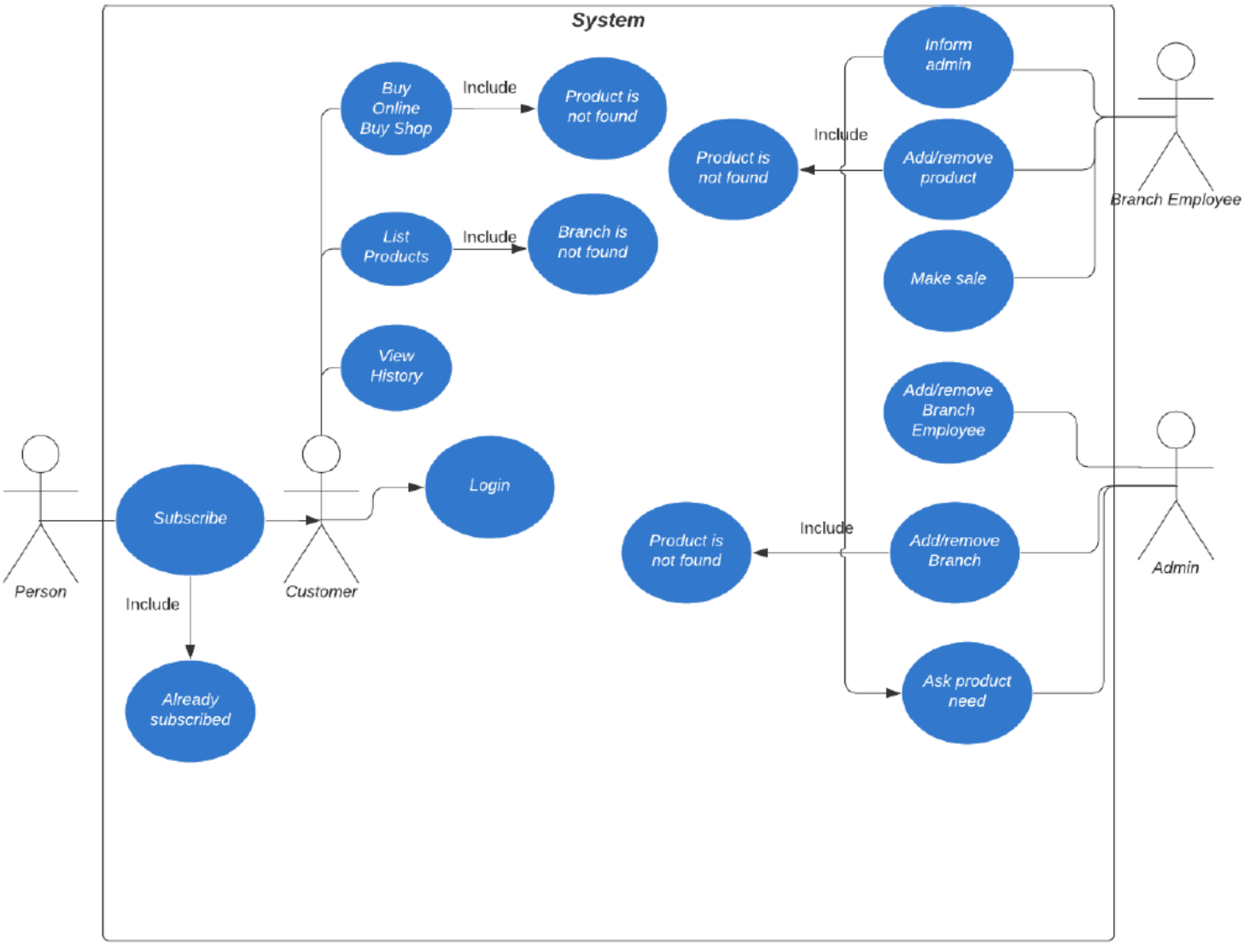
```
c1.getAdmin( index: 0).add_branch(new Branch( n: "BRANCH_B2"));
customer.list_all(c1);

customer.list_product(c1, name: "OCHAIR_1");
```

Customer buys online and view order history (In buy online method, buy shop method is called)

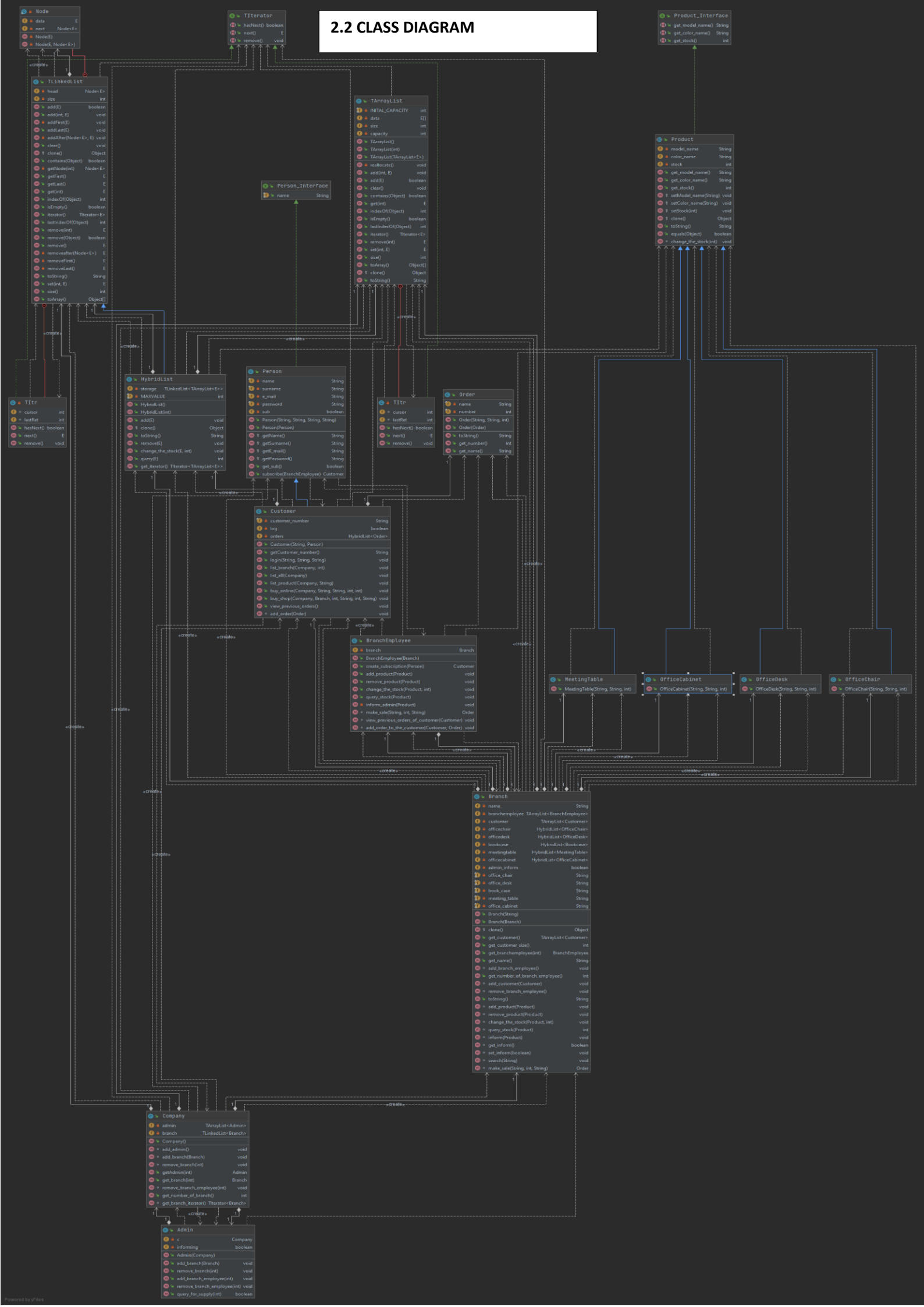
```
customer.buy_online(c1, name: "OCABINET_1", name2: "BLACK", stock: 1, branch_index: 0);
customer.view_previous_orders();
customer.list_branch(c1, index: 0);
```

2. USE CASE AND CLASS DIAGRAMS

2.1 USE CASE DIAGRAM



2.2 CLASS DIAGRAM



3. PROBLEM SOLUTION APPROACH

In this homework I used ArrayList for users of the system which are :

1. Admin
2. Branch Employee
3. Customer

I used LinkedList for information about branches

1. Branch

I used HybridList for Products and Orders

1. Office Chair
2. Office Desk
3. Meeting Table
4. Book Case
5. Office Cabinet
6. Order

In HybridList there is a LinkedList component named storage. LinkedList's each Node holds an ArrayList which stores E type as Product

In ArrayList and LinkedList classes there are iterators.

I set up the MAX_VALUE as 10

I reallocate the arrays when capacity is full and I increased the capacity by 2

In Customer class, I have HybridList class which stores the Orders of customer.

At first , when a company is created, some branches and admins are created.

When branch is created, branch employees and customers are created automatically

Products are also added automatically to prevent null pointers.

4. TEST CASES

TEST CASE 1

Person should subscribed properly

```
SUBSCRIPTION WILL BE DONE
1) PERSON MODE
2) CUSTOMER MODE
3) ADMIN MODE
4) BRANCH EMPLOYEE MODE
0) EXIT
Enter a number:
1
There are 2 persons, choose one
1) Person{ name='Talha', surname='Yolcu', e_mail='E_Mail', password='Passwd', sub=true}
2) Person{ name='Yakup', surname='Yolcu', e_mail='EMAIL', password='PASSWD', sub=false}
1
SUBSCRIPTION WILL BE DONE
This person already subscribed
1) PERSON MODE
2) CUSTOMER MODE
3) ADMIN MODE
4) BRANCH EMPLOYEE MODE
0) EXIT
```

TEST CASE 2

Customer should login,list branch, list all of the products, search for a product

Login ->

```
1
The customer with customer number ABCDEF has login to the system successfully
1)LOGIN TO THE SYSTEM, IT MUST BE DONE BEFORE THE OTHERS
2)LIST A BRANCH'S PRODUCT
3)LIST WHOLE COMPANY
4)SEARCH FOR A PRODUCT
5)BUY PRODUCT ONLINE
6)BUY PRODUCT FROM SHOP
7)VIEW PREVIOUS ORDERS
0)EXIT
1
This customer already logined
1)LOGIN TO THE SYSTEM, IT MUST BE DONE BEFORE THE OTHERS
```

Listing a branch ->

```
1)LOGIN TO THE SYSTEM, IT MUST BE DONE BEFORE THE OTHERS
2)LIST A BRANCH'S PRODUCT
3)LIST WHOLE COMPANY
4)SEARCH FOR A PRODUCT
5)BUY PRODUCT ONLINE
6)BUY PRODUCT FROM SHOP
7)VIEW PREVIOUS ORDERS
0)EXIT
2
Enter an index for branch(starts from zero), max index:1
0
THIS IS BRANCH_B0
PRODUCT NAME    COLOR   STOCK
OCHAIR_1            BLACK       2
ODESK_1         BLUE        5
BCASE_1         RED     10
MTABLE_1            BLACK       4
OCABINET_1          BLACK       3
```

Listing all of the company

```
0)EXIT
3
WHOLE COMPANY WILL BE LISTED...

THIS IS BRANCH_B0
PRODUCT NAME    COLOR   STOCK
OCHAIR_1            BLACK       2
ODESK_1         BLUE        5
BCASE_1         RED     10
MTABLE_1            BLACK       4
OCABINET_1          BLACK       3

THIS IS BRANCH_B1
PRODUCT NAME    COLOR   STOCK
OCHAIR_1            BLACK       2
ODESK_1         BLUE        5
BCASE_1         RED     10
MTABLE_1            BLACK       4
OCABINET_1          BLACK       3
```

Searching for a product

```
4
Enter name of the product
BCASE_1
IN BRANCH BRANCH_B0
BCASE_1     RED     10

IN BRANCH BRANCH_B1
BCASE_1     RED     10
```

Buy Product Online

```
5
Enter name of the model name
MTABLE_1

Enter name of the color name
BLACK

How much do you want to buy (enter stock)
1
```

```
Previous orders will be viewed
MTABLE_1 BLACK 1
```

Buy Product Online

```
Enter branch index
1
Enter name of the model name
MTABLE_1
Enter name of the color name
BLACK
How much do you want to buy (enter stock)
1
```

```
Previous orders will be viewed
MTABLE_1 BLACK 1
```

TEST CASE 3

Admin should add branch,remove branch,add branch employee remove branch employee and ask for any supply

Add branch ->

```
1
New branch is added, name is : THIS IS BRANCH_B2
PRODUCT NAME    COLOR   STOCK
OCHAIR_1            BLACK       2
ODESK_1         BLUE        5
BCASE_1         RED     10
MTABLE_1            BLACK       4
OCABINET_1          BLACK       3
```

Remove branch

```
2
Enter index between 0 - 2
0
Branch is removed successfully
```

Add Branch Employee

```
3
Enter index to determine which branch to add branch employee (0-1
0
Branch employee is added successfully
```

Remove Branch Employee

```
4
Enter index to determine from the which branch branch employee will be removed (0-1
1
Branch employee is removed successfully
```

Ask for any supply need

```
5
Enter branch index to query
1
Supply is not needed
```

TEST CASE 4

Branch Employee should add and remove product. I couldn’t show remove product and some other operations on test cases but they are implemented anyway.

```
Enter a number:
4
Enter branch index to be the branch employee (0-1)
0
1)ADD PRODUCT
2)REMOVE PRODUCT
0)EXIT
1
1)ADD OFFICE CHAIR
2)ADD OFFICE DESK
3)ADD MEETING TABLE
4)ADD BOOK CASE
5)ADD OFFICE CABINET
Enter:
2
Sample model name:
ODESK_X
Colors:
BLACK-BLUE-RED-GREEN
There can be only 5 models
Enter number 1-5)
2
Enter color (all big letter)
RED
1)ADD PRODUCT
2)REMOVE PRODUCT
0)EXIT
```

5. RUNNING AND RESULTS

It is done at Test cases section already…

PART 2

Complexity Analysis

1. **Complexity of Admin Class' methods**

Constructor : **Θ(1)**

```
public Admin(Company c1) { c=c1; }
```

Add branch: Complexity of Company's Add_branch method => **Θ(1)**

```
public void add_branch(Branch b) { c.add_branch(b); }
```

Remove branch: Complexity of Company's Remove_branch method **O(n)**

```
public void remove_branch(int index) { c.remove_branch(index); }
```

Add branch employee : Complexity of Company's get branch + Branch's add branch employee method **O(n)** + **Θ(1)** => **O(n)**

```
public void add_branch_employee(int index) { c.get_branch(index).add_branch_employee(); }
```

Remove branch employee: Complexity of Company's get branch + Branch's remove branch employee method **O(n) + O(n) => O(n)**

```
public void remove_branch_employee(int index) { c.remove_branch_employee(index); }
```

Query for supply need: Complexity of Company's get branch and branch's get inform

```
public boolean query_for_supply(int branch_index) {
    if(c.get_branch(branch_index).get_inform()) {
        informing=true;
        c.get_branch(branch_index).set_inform(false);
    }
    return informing;
}
```

O(n) + Θ(1) => Θ(1)

2. **Complexity of OfficeChair, OfficeDesk, MeetingTable, Bookcase and Office Cabinet's** Consturctors (All constructors are the same, so we don't need to analyze each of them.

Constructor:

Θ(1) because we just have assignment statements in the methods that we are going to analyze soon.

```
public Bookcase(String model,String color,int stock) {
    setModel_name(model);
    setColor_name(color);
    setStock(stock);
}
```

3. Complexity of Branch Class' methods

```
public Branch(String n) {
    /*
    We should create needed things like Hybridlists,ArrayLists
     */
    name=n;
    branchemployee=new TArrayList<>();
    customer=new TArrayList<>();
    customer.add(new Customer( cn: "NULL",new Person( n: "NULL", s: "NULL", e: "NULL", p: "NULL")));

    officechair=new HybridList<>();
    officedesk=new HybridList<>();
    bookcase=new HybridList<>();
    meetingtable=new HybridList<>();
    officecabinet=new HybridList<>();

    /*
    At the beginning we should have the products
     */

    officechair.add(new OfficeChair( model: office_chair+"_1", color: "BLACK", stock: 2));
    officedesk.add(new OfficeDesk( model: office_desk+"_1", color: "BLUE", stock: 5));
    bookcase.add(new Bookcase( model: book_case+"_1", color: "RED", stock: 10));
    meetingtable.add(new MeetingTable( model: meeting_table+"_1", color: "BLACK", stock: 4));
    officecabinet.add(new OfficeCabinet( model: office_cabinet+"_1", color: "BLACK", stock: 3));

    /*
    We added some branch employees
     */
    branchemployee.add(new BranchEmployee( b1: this));

}
```

Constructor:

At first there are simple assignments that have **Θ(1)** complexity

After these, there are some add methods.

These are HybrList's add methods.

It is : **$O(n^2)$ => $T(n) = O(n^2)$**

Add branch employee method

```
void add_branch_employee() { branchemployee.add(new BranchEmployee( b1: this)); }
```

It calls linkedlist's add method

It is **Θ(1)**

Add customer method

```
void add_customer(Customer customer1) { customer.add(customer1); }
```

It calls linkedlist's add method

It is **Θ(1)**

Remove branch employee method

```
void remove_branch_employee() { branchemployee.remove( index: branchemployee.size()-1); }
```

It calls linkedlist's remove method

It is **O(n)**

Add product method

```
void add_product(Product p) throws NoSuchElementException{
    if(p instanceof OfficeChair) {
        officechair.add((OfficeChair)p);
    }
    else if(p instanceof OfficeDesk) {
        officedesk.add((OfficeDesk) p);
    }
    else if(p instanceof Bookcase) {
        bookcase.add((Bookcase) p);
    }
    else if(p instanceof MeetingTable) {
        meetingtable.add((MeetingTable) p);
    }
    else if(p instanceof OfficeCabinet) {
        officecabinet.add((OfficeCabinet) p);
    }
    else {
        throw new NoSuchElementException("Wrong product");
    }
}
```

In every if – else statements, it calls hybrid List's add methods.

It is **$O(n^2)$**

Remove product method

In every case, it calls hybridlist's remove method.

It is **O(n^4*m)**

```
void remove_product(Product p) {
    if(p instanceof OfficeChair) {
        officechair.remove((OfficeChair)p);
    }
    else if(p instanceof OfficeDesk) {
        officedesk.remove((OfficeDesk) p);
    }
    else if(p instanceof Bookcase) {
        bookcase.remove((Bookcase) p);
    }
    else if(p instanceof MeetingTable) {
        meetingtable.remove((MeetingTable) p);
    }
    else if(p instanceof OfficeCabinet) {
        officecabinet.remove((OfficeCabinet) p);
    }
    else {
        throw new NoSuchElementException("Wrong product");
    }
}
```

Change the stock method

```
void change_the_stock(Product p,int number) {
    if(p instanceof OfficeChair) {
        officechair.change_the_stock((OfficeChair)p,number);
    }
    else if(p instanceof OfficeDesk) {
        officedesk.change_the_stock((OfficeDesk) p,number);
    }
    else if(p instanceof Bookcase) {
        bookcase.change_the_stock((Bookcase) p,number);
    }
    else if(p instanceof MeetingTable) {
        meetingtable.change_the_stock((MeetingTable) p,number);
    }
    else if(p instanceof OfficeCabinet) {
        officecabinet.change_the_stock((OfficeCabinet) p,number);
    }
    else {
        throw new NoSuchElementException("Wrong product");
    }
}
```

⇨ It calls hybridlist's change the stock method. It is **O(n^2*m)**

It's complexity is **O(n^3*m)**

Query stock method: It calls HybridList's query stock method **O(n^3*m)**

<=

```
int query_stock(Product p) {
    int x=0;
    if(p instanceof OfficeChair) {
        x=officechair.query((OfficeChair)p);
    }
    else if(p instanceof OfficeDesk) {
        x=officedesk.query((OfficeDesk) p);
    }
    else if(p instanceof Bookcase) {
        x=bookcase.query((Bookcase) p);
    }
    else if(p instanceof MeetingTable) {
        x=meetingtable.query((MeetingTable) p);
    }
    else if(p instanceof OfficeCabinet) {
        x=officecabinet.query((OfficeCabinet) p);
    }
    else {
        throw new NoSuchElementException("Wrong product");
    }
    return x;
}
```

ToString method

We have hybridlist's toString method

Which have complexity **O(n^2 * m)**

```
public String toString() {
    StringBuilder str=new StringBuilder();
    str.append("THIS IS " + get_name());
    str.append("\nPRODUCT NAME\tCOLOR\tSTOCK\n");
    str.append(officechair.toString());
    str.append(officedesk.toString());
    str.append(bookcase.toString());
    str.append(meetingtable.toString());
    str.append(officecabinet.toString());

    return str.toString();
}
```

Search method

Firstly we

Have iterator.

In while loop,

We have next

And we have

Another while

Loop.

In the second

While loop,

There are

Constant

Operations.

So Complexity

Is for Best case

$\Theta(1)$

Worst case

$\Theta(n^2)$

$T(n)=O(n^2)$

```
void search(String name) throws NoSuchElementException{
    /*
     * Depend on the name, determine the product and search for it
     */
    int count=0;
    if(name.contains(office_chair)) {
        TIterator<TArrayList<OfficeChair>> iter=officechair.get_iterator();
        while (iter.hasNext()) {
            TArrayList<OfficeChair> x=iter.next();
            TIterator<OfficeChair> iter2=x.iterator();
            while (iter2.hasNext()) {
                OfficeChair p1=iter2.next();
                if(name.equals(p1.get_model_name())) {
                    System.out.println(name + "\t\t" + p1.get_color_name() + "\t\t" + p1.get_stock()+"\n");
                    count+=1;
                }
            }
        }
    }
    else if(name.contains(office_desk)) {
        TIterator<TArrayList<OfficeDesk>> iter=officedesk.get_iterator();
        while (iter.hasNext()) {
            TArrayList<OfficeDesk> x=iter.next();
            TIterator<OfficeDesk> iter2=x.iterator();
            while (iter2.hasNext()) {
                OfficeDesk p1=iter2.next();
                if(name.equals(p1.get_model_name())) {
                    System.out.println(name + "\t\t" + p1.get_color_name() + "\t\t" + p1.get_stock()+"\n");
                    count+=1;
                }
            }
        }
    }
    else if(name.contains(meeting_table)) {
        TIterator<TArrayList<MeetingTable>> iter=meetingtable.get_iterator();
        while (iter.hasNext()) {
            TArrayList<MeetingTable> x=iter.next();
            TIterator<MeetingTable> iter2=x.iterator();
            while (iter2.hasNext()) {
                MeetingTable p1=iter2.next();
                if(name.equals(p1.get_model_name())) {
                    System.out.println(name + "\t\t" + p1.get_color_name() + "\t\t" + p1.get_stock()+"\n");
                    count+=1;
                }
            }
        }
    }
    else if(name.contains(book_case)) {
        TIterator<TArrayList<Bookcase>> iter=bookcase.get_iterator();
        while (iter.hasNext()) {
            TArrayList<Bookcase> x=iter.next();
            TIterator<Bookcase> iter2=x.iterator();
            while (iter2.hasNext()) {
                Bookcase p1=iter2.next();
                if(name.equals(p1.get_model_name())) {
                    System.out.println(name + "\t\t" + p1.get_color_name() + "\t\t" + p1.get_stock()+"\n");
                    count+=1;
                }
            }
        }
    }
    else if(name.contains(office_cabinet)) {
        TIterator<TArrayList<OfficeCabinet>> iter=officecabinet.get_iterator();
        while (iter.hasNext()) {
            TArrayList<OfficeCabinet> x=iter.next();
            TIterator<OfficeCabinet> iter2=x.iterator();
            while (iter2.hasNext()) {
                OfficeCabinet p1=iter2.next();
                if(name.equals(p1.get_model_name())) {
                    System.out.println(name + "\t\t" + p1.get_color_name() + "\t\t" + p1.get_stock()+"\n");
                    count+=1;
                }
            }
        }
    }
    else {
        throw new NoSuchElementException("There is no such product with this name in this Product:" + get_name());
    }
    if(count==0) {
        System.out.println("There is no such prouct in this product with this name\n");
    }
}
```

Make sale method

Similar to search,

We have 2 while

Loops.

We have constant

Operations in the

Inner loop.

So Complexity

Is for Best case

Θ(1)

Worst case

Θ(n^2)

T(n)=O(n^2)

```
Order make_sale(String name,int stock,String name2) throws NoSuchElementException{
    int count=0;
     /*
      * Depend on the names, determine the product type and find it
      * Change the stock of the product
      */
    if(name.contains(office_chair)) {
        TIterator<TArrayList<OfficeChair>> iter=officechair.get_iterator();
        while (iter.hasNext()) {
            TArrayList<OfficeChair> x=iter.next();
            TIterator<OfficeChair> iter2=x.iterator();
            while (iter2.hasNext()) {
                OfficeChair p1=iter2.next();
                if(name.equals(p1.get_model_name())  && p1.get_stock()>=stock && p1.get_color_name().equals(name2)) {
                    p1.setStock(p1.get_stock()-stock);
                    return new Order(name,name2,stock);
                }
            }
        }
    }
    else if(name.contains(office_desk)) {
        TIterator<TArrayList<OfficeDesk>> iter=officedesk.get_iterator();
        while (iter.hasNext()) {
            TArrayList<OfficeDesk> x=iter.next();
            TIterator<OfficeDesk> iter2=x.iterator();
            while (iter2.hasNext()) {
                OfficeDesk p1=iter2.next();
                if(name.equals(p1.get_model_name())  && p1.get_stock()>=stock && p1.get_color_name().equals(name2)) {
                    p1.setStock(p1.get_stock()-stock);
                    return new Order(name,name2,stock);
                }
            }
        }
    }
    else if(name.contains(meeting_table)) {
        TIterator<TArrayList<MeetingTable>> iter=meetingtable.get_iterator();
        while (iter.hasNext()) {
            TArrayList<MeetingTable> x=iter.next();
            TIterator<MeetingTable> iter2=x.iterator();
            while (iter2.hasNext()) {
                MeetingTable p1=iter2.next();
                if(name.equals(p1.get_model_name())  && p1.get_stock()>=stock && p1.get_color_name().equals(name2)) {
                    p1.setStock(p1.get_stock()-stock);
                    return new Order(name,name2,stock);
                }
            }
        }
    }
    else if(name.contains(book_case)) {
        TIterator<TArrayList<Bookcase>> iter=bookcase.get_iterator();
        while (iter.hasNext()) {
            TArrayList<Bookcase> x=iter.next();
            TIterator<Bookcase> iter2=x.iterator();
            while (iter2.hasNext()) {
                Bookcase p1=iter2.next();
                if(name.equals(p1.get_model_name())  && p1.get_stock()>=stock && p1.get_color_name().equals(name2)) {
                    p1.setStock(p1.get_stock()-stock);
                    return new Order(name,name2,stock);
                }
            }
        }
    }
    else if(name.contains(office_cabinet)) {
        TIterator<TArrayList<OfficeCabinet>> iter=officecabinet.get_iterator();
        while (iter.hasNext()) {
            TArrayList<OfficeCabinet> x=iter.next();
            TIterator<OfficeCabinet> iter2=x.iterator();
            while (iter2.hasNext()) {
                OfficeCabinet p1=iter2.next();
                if(name.equals(p1.get_model_name())  && p1.get_stock()>=stock && p1.get_color_name().equals(name2)) {
                    p1.setStock(p1.get_stock()-stock);
                    return new Order(name,name2,stock);
                }
            }
        }
    }
    else {
        throw new NoSuchElementException("There is no such product with this name in this Product:" + get_name());
    }
    return null;
}
```

4. Complexity of Branch Employee Class' methods

Create Subscription

we have get sub method in the if condition. It is constant time.
Then we are appending to StringBuilder which is complexity is **Θ(n) .**
Then we are calling the Add customer method of the branch class.
It's complexity is **Θ(1)**
T(n)= **Θ(n)**

```
public Customer create_subscription(Person p1) throws AlreadyExistException{
    /*
    If this user is already a customer
     */
    if(p1.get_sub()) {
        throw new AlreadyExistException("This person already subscribed");
    }
    StringBuilder str=new StringBuilder();
    str.append("CUSTOMER_");
    str.append(branch.get_name()+"_");
    str.append(branch.get_customer_size());
    Customer customer=new Customer(str.toString(),p1);
    branch.add_customer(customer);
    return customer;
}
```

Add product

```
public void add_product(Product p) { branch.add_product(p); }
```

It calls branch class' add product method. It's complexity is **O(n^2)**

Remove Product

```
public void remove_product(Product x) { branch.remove_product(x); }
```

It calls branch class' remove product method.
It's complexity is **O(n^4*m)**

Make Sale method

```
Order make_sale(String name,int stock,String name2) {
    Order o=branch.make_sale(name,stock,name2);
    if(o==null) {
        System.out.println("Sorry,there is not enough to buy it");
    }
    return o;
}
```

It calls branch's make sale method which Have O(n^2) . Then we have simple if Statement which is constant.
This method's complexity is **O(n^2)**

Change the stock method

```
public void change_the_stock(Product p,int number) { branch.change_the_stock(p,number); }
```

It calls branch's change the stock method which is **O(n^4*m)**

Inform admin method

```
private void inform_admin(Product p) { branch.inform(p); }
```

It calls branch class' inform method. Which is **Θ(1)**

View previous orders method

```
void view_previous_orders_of_customer(Customer c1) { c1.view_previous_orders(); }
```

It calls a customer's previous orders method which is **Θ(n)**

Add order method

```
void add_order_to_the_customer(Customer c1,Order o1) { c1.add_order(o1); }
```

It calls customers add order method

Which is **O(n^2)**

5. Complexity of Company Class' methods

```
public Company() {
    /*
    admins and branches are created,admins branch employees and customers are added
     */
    admin=new TArrayList<>();
    branch=new TLinkedList<>();
    admin.add(new Admin( c1: this));
    admin.add(new Admin( c1: this));
    add_admin();
    branch.add(new Branch( n: "BRANCH_B"+branch.size()));
    branch.add(new Branch( n: "BRANCH_B"+branch.size()));
    branch.get(0).add_branch_employee();
    branch.get(1).add_branch_employee();
    branch.get(0).add_customer(new Customer( cn: "NULL1",new Person( n: "NULL1", s: "NULL1", e: "NULL1", p: "NULL1")));
    branch.get(1).add_customer(new Customer( cn: "NULL2",new Person( n: "NULL2", s: "NULL2", e: "NULL2", p: "NULL2")));
}
```

Constructor

We have arraylist add **Θ(1)**
And linkedlist adds. **Θ(1)**
These are … **Θ(1)**
Then we have add Customer **Θ(1)**
and we have add branch employee. **Θ(1)**
Our complexity is … **Θ(1)**

Add admin

```
void add_admin() { admin.add(new Admin( c1: this)); }
```

We have arraylist's add method which is **Θ(1)**

Add branch

We have linkedlist's add(**Θ(1)**)method and
Branch's add branch employee method **Θ(1)**
T(n)= **Θ(1)**

```
void add_branch(Branch b) {
    branch.add(b);
    branch.get(branch.size()-1).add_branch_employee();
}
```

Remove Branch

We have linkedlist's remove method.
Which is … **O(n)**

```
void remove_branch(int index) { branch.remove(index); }
```

Remove Branch employee

We have linkedlist get method and **O(n)**
Branch's remove branch employee method. Which is **O(n)** **T(n) = O(n)**

```
void remove_branch_employee(int index) { branch.get(index).remove_branch_employee(); }
```

Get Admin method

We have arraylist's get method which is
Θ(1)

```
public Admin getAdmin(int index) { return admin.get(index); }
```

Get Branch method

We have linkedlist's get method which is **O(n)**

```
public Branch get_branch(int index) { return this.branch.get(index); }
```

Get Number Of Branch : **Θ(1)**

```
public int get_number_of_branch() { return branch.size(); }
```

Get branch iterator : **Θ(1)**

```
TIterator<Branch> get_branch_iterator() { return branch.iterator(); }
```

6. Complexity of Customer Class' methods.

Get customer number: Θ(1)

```
public String getCustomer_number() { return customer_number; }
```

Login Θ(1)

Some String'Sequals methods are called.These are constant.

```
public void login(String cn,String e,String p) throws AlreadyExistException{
    /*
    If customer already loginned, throw exception
     */
    if(log) {
        throw new AlreadyExistException("User already login to the system");
    }
    if(cn.equals(customer_number) && e.equals(getE_mail()) && p.equals(getPassword())) {
        log=true;
        System.out.println("The customer with customer number " + cn +
                " has login to the system successfully");
    }
    else {
        System.out.println("Some information is wrong,try again");
    }
}
```

List Branch method

It calls company's get branch method => O(n)

And ensures to call toString method of Branch class. It's **O(n^2 * m)**

We have **O(n^2 * m)** complexity

```
public void list_branch(Company c1,int index) throws OperationNotSupportedException{
    if(!log) {
        throw new OperationNotSupportedException("This customer has not login to the system yet");
    }
    else {
        System.out.println(c1.get_branch(index));
    }
}
```

List all method

Number of branches : n

List branch complexity **O(n^2 * m)**

T(n,m)= **O(n^3 * m)**

```
public void list_all(Company c1) throws OperationNotSupportedException {
    if(!log) {
        throw new OperationNotSupportedException("This customer has not login to the system yet");
    }
    else {
        for(int i=0;i<c1.get_number_of_branch();i++) {
            try {
                list_branch(c1, i);
            }
            catch (OperationNotSupportedException ne) {
                throw new OperationNotSupportedException();
            }
        }
    }
}
```

List Product method

We have get branch Iterator method which Is constant time.

We have while loop.

In while loop we have Branch's search method

Our complexity is T(n,m)=**O(n^2 * m)**

```
public void list_product(Company c1,String name) throws OperationNotSupportedException{
    if(!log) {
        throw new OperationNotSupportedException("This customer has not login to the system yet");
    }
    else {
        TIterator<Branch> iter=c1.get_branch_iterator();
        while (iter.hasNext()) {
            Branch x=iter.next();
            System.out.println("IN BRANCH " + x.get_name());
            x.search(name);
        }
    }
}
```

N=number oof products In a branch

M=number of branchs

```java
public void buy_online(Company c1,String name,String name2,int stock,int branch_index) {
    try {
        list_all(c1);
        buy_shop(c1,c1.get_branch(branch_index),branch_index,name,stock,name2);

    }
    catch (OperationNotSupportedException ne) {
        System.out.println(ne);
    }
}
```

Buy online
List all method: **O(n^3 * m)**
Buy shop method: **O(n^2)**

Our complexity is: **O(n^3 * m)**

```java
public void buy_shop(Company c1,Branch b1,int branch_index,String name,int stock,String name2)
    try {
        list_branch(c1,branch_index);
        Order o=b1.get_branchemployee( index: 0).make_sale(name,stock,name2);
        orders.add(o);
    }
    catch (OperationNotSupportedException ne) {
        System.out.println(ne);
    }
}
```

Buy shop
List branch method:
Get branch employee-> **Θ(1)**
Make sale -> **O(n^2)**
Hybridlist add -> **O(n^2)**

T(n)= **O(n^2)**

```java
public void view_previous_orders() {
    TIterator<TArrayList<Order>> iter= orders.get_iterator();
    while (iter.hasNext()) {
        System.out.println(iter.next());
    }
}
```

View previous orders: **Θ(n)**

Add order: hybridlist's add -> **O(n^2)**

```java
void add_order(Order o) { orders.add(o); }
public int get_number_of_orders() {
    return orders.size();
}
```

Get number of orders: **Θ(1)**

7. **Complexity of HybridList class' methods**

```java
public HybridList(int init) {
    storage=new TLinkedList<>();
    storage.add(new TArrayList<E>(init));
    storage.add(new TArrayList<E>(init));
}
```

Constructor : LinkedList'add method: : **Θ(1)**

Add: Tb -> while loop, linkedlist get, = **O(n^2)**
Tw-> while loop,linkedlist get, arraylist add **O(n^2)** => while loop total complexity -> **O(n^2)**
Linkedlist add -> **Θ(1)**
LinkedList get **-> O(n)**
T(n) = **O(n^2)**

```java
public void add(E e) {
    int i=0;
    while(i<storage.size()) {
        if(storage.get(i).size()<MAXVALUE) {
            storage.get(i).add(e);
            return;
        }
    }
    storage.add(new TArrayList<>());
    storage.get(storage.size()-1).add(e);
}
```

toString:

```
public String toString() {
    StringBuilder str=new StringBuilder();
    for(int i=0;i<storage.size();i++) {
        str.append(storage.get(i).toString());
    }
    return str.toString();
}
```

number of linkedlist -> n
linkedlist toString -> **Θ(m)**
linkedlist get -> **O(n)**
T(n,m) = **$O(n^2 * m)$**
Number of arraylist in a linkedlist -> m

Remove

```
public void remove(E e) {
    for(int i=0;i<storage.size();i++) {
        for(int k=0;k<storage.get(i).size();k++) {
            if(storage.get(i).get(k).equals(e)) {
                storage.get(i).remove(k);
            }
        }
    }
}
```

Number of linkedlist -> n
Number of arraylist in a linkedlist -> m
Linkedlist get -> **O(n)**
Arraylist get -> **Θ(1)**
LinkedList remove -> **O(n)**
Tb= **$O(n^2*m)$** Tw= **$O(n^4*m)$**
T(n,m) = **$O(n^4*m)$**

Change the stock

```
public void change_the_stock(E e,int number) {
    for(int i=0;i<storage.size();i++) {
        for(int k=0;k<storage.get(i).size();k++) {
            if(storage.get(i).get(k).equals(e)) {
                Product p=(Product) e;
                p.change_the_stock(number);
            }
        }
    }
}
```

Number of linkedlist -> n
Number of arraylist in a linkedlist -> m
Linkedlist get -> **O(n)**
Arraylist get -> **Θ(1)**
Product change the stock **Θ(1)**

Tb= **$O(n^2*m)$** Tw= **$O(n^2*m)$**
T(n,m) = **$O(n^2*m)$**

Query

```
public int query(E e) {
    int x=0;
    for(int i=0;i<storage.size();i++) {
        for(int k=0;k<storage.get(i).size();k++) {
            if(storage.get(i).get(k).equals(e)) {
                Product p=(Product) storage.get(i).get(k);
                x+=p.get_stock();
            }
        }
    }
    return x;
}
```

Number of linkedlist -> n
Number of arraylist in a linkedlist -> m
Linkedlist get -> **O(n)**
Arraylist get -> **Θ(1)**
Tb= **$O(n^2*m)$** Tw= **$O(n^3*m)$**
T(n,m) = **$O(n^3*m)$**

Get iterator Θ(1)

Size -> **Θ(n)**

```
public TIterator<TArrayList<E>> get_iterator() { return storage.iterator(); }
public int size() {
    int x=0;
    for(int i=0;i<storage.size();i++) {
        x+=storage.get(i).size();
    }
    return x;
}
```

8. **Complexity of Order Class' methods**

```
public Order(String name1,String name2,int stock) {
    StringBuilder str=new StringBuilder();
    str.append(name1);
    str.append(" ");
    str.append(name2);
    str.append(" ");
    str.append(stock);
    name=str.toString();
    number=stock;
}
```

Constructor: Θ(1)

```
public String toString() { return name; }
```

toString : Θ(1)

```
public int get_number() { return number; }

/**
 * Getter for name of the product
 * @return name of the product
 */
public String get_name() { return name; }
```

get number : **Θ(1)**
get name : **Θ(1)**

9. **Complexity of Person Class' methods**

```
public Person(Person p1) {
    this.name=p1.getName();
    this.surname=p1.getSurname();
    this.e_mail=p1.getE_mail();
    this.password=p1.getPassword();
    this.sub=p1.sub;
}
```

Constructor: **Θ(1)**

```
public Person(String n,String s,String e,String p) {
    name=n;
    surname=s;
    e_mail=e;
    password=p;
}
```

Subscribe:

Branch employee's create subscription is: **Θ(r**

T(n) = **Θ(n)**

```
public Customer subscribe(BranchEmployee be1) throws AlreadyExistException {
    if(get_sub()) {
        throw new AlreadyExistException("This person already subscribed");
    }
    try {
        Customer custom=be1.create_subscription( p1: this);
        sub=true;
        return custom;
    }
    catch (AlreadyExistException ne) {
        throw ne;
    }
}
```