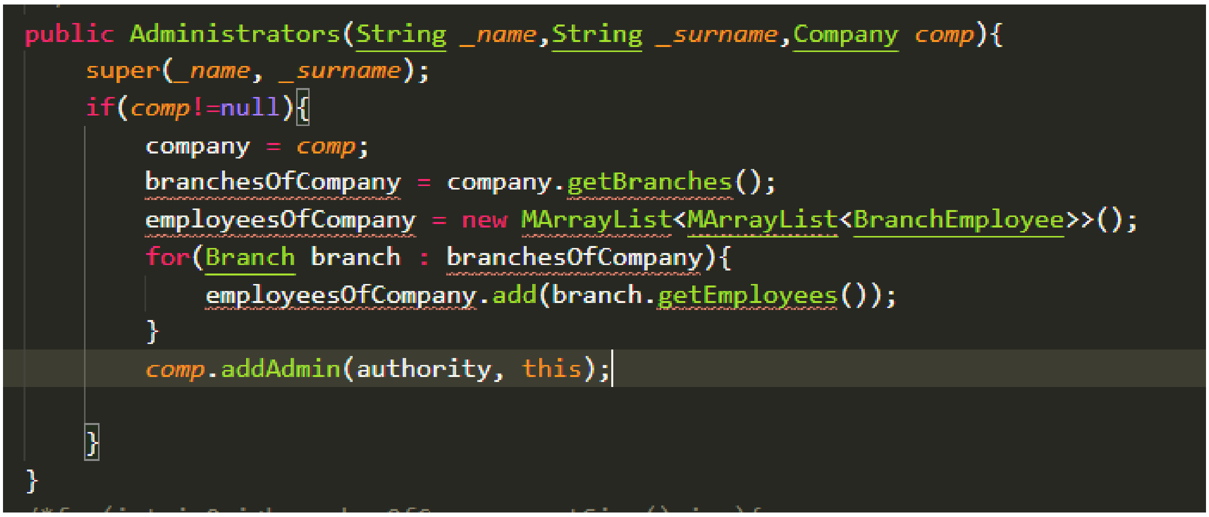
Adminitrators’ method:



```
public Administrators(String _name,String _surname,Company comp){
    super(_name, _surname);
    if(comp!=null){
        company = comp;
        branchesOfCompany = company.getBranches();
        employeesOfCompany = new MArrayList<MArrayList<BranchEmployee>>();
        for(Branch branch : branchesOfCompany){
            employeesOfCompany.add(branch.getEmployees());
        }
        comp.addAdmin(authority, this);

    }
}
```

GetBranches() is constant Teta(1)

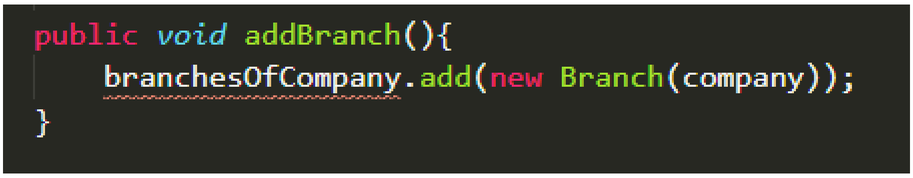
MArrayList constructor constant Teta(1)

For every branch in the given company, add the every employee of branch to the employee :

Teta(n) is for the iterator, + employeesOfCompany.add() is amortized contant time (Teta(1))

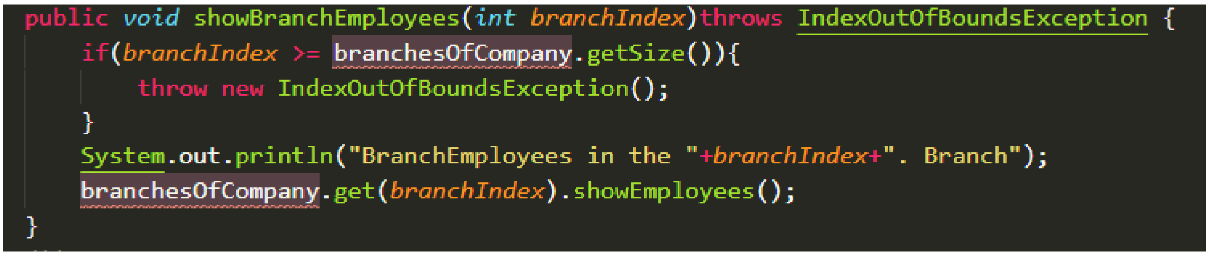
AddAdmin is amortized constant time (Teta(1))

All of them: Teta(n)



```
public void addBranch(){
    branchesOfCompany.add(new Branch(company));
}
```

BranchesOfCompany is Double Linked list , adding to the end of it is Teta(1)



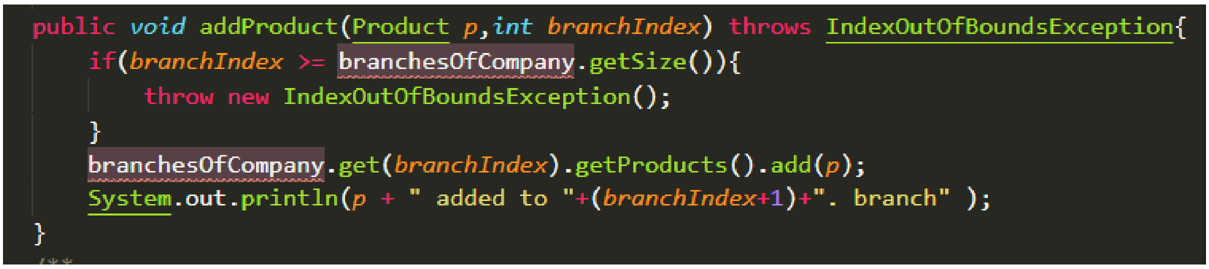
```
public void showBranchEmployees(int branchIndex)throws IndexOutOfBoundsException {
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    System.out.println("BranchEmployees in the "+branchIndex+". Branch");
    branchesOfCompany.get(branchIndex).showEmployees();
}
```

If n is the number of branches , and m is the number of employees in the given branch

BranchesOfCompany is Double Linked list , get element is O(n)

ShowEmployees prints all of the elements in the returned branch, so it is Teta(m)

All of them : O(max(n,m))

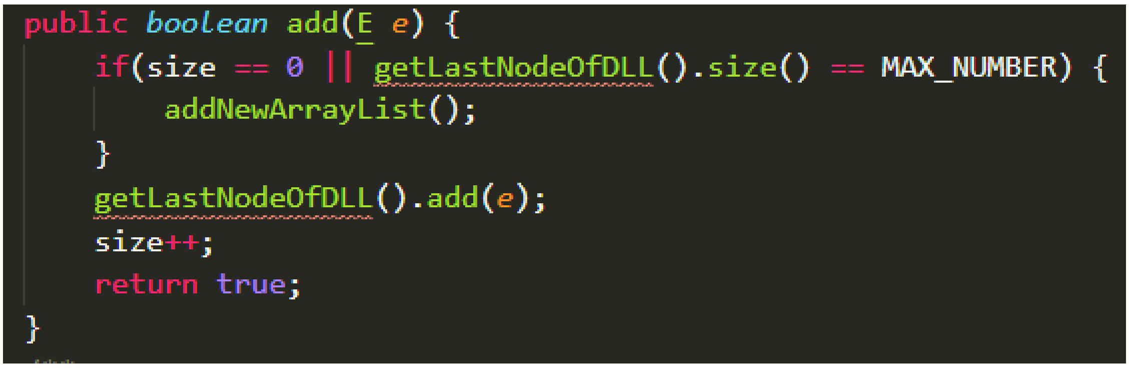


```java
public void addProduct(Product p,int branchIndex) throws IndexOutOfBoundsException{
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    branchesOfCompany.get(branchIndex).getProducts().add(p);
    System.out.println(p + " added to "+(branchIndex+1)+". branch" );
}
```

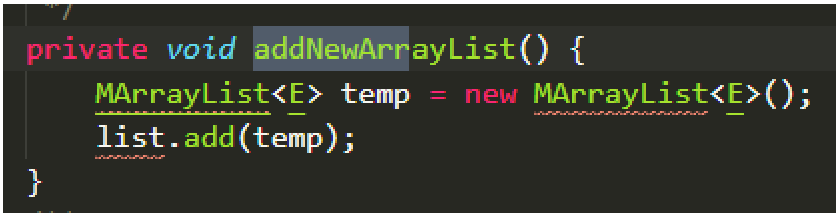
get = O(n)

GetProducts = teta(1)

Products hold in hybrid list , it's add method:

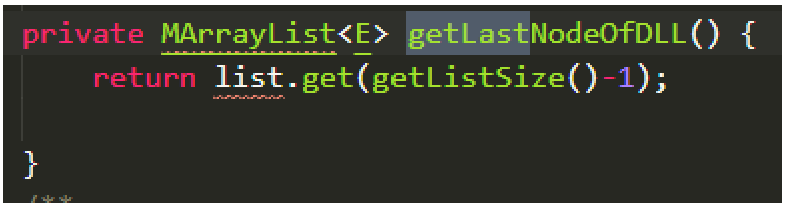


```java
public boolean add(E e) {
    if(size == 0 || getLastNodeOfDLL().size() == MAX_NUMBER) {
        addNewArrayList();
    }
    getLastNodeOfDLL().add(e);
    size++;
    return true;
}
```



```java
private void addNewArrayList() {
    MArrayList<E> temp = new MArrayList<E>();
    list.add(temp);
}
```

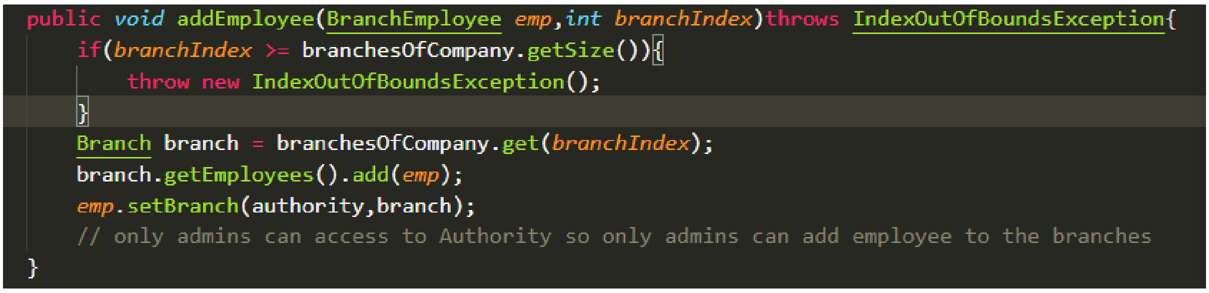
AddNewArrayList is constant teta(1)



```java
private MArrayList<E> getLastNodeOfDLL() {
    return list.get(getListSize()-1);

}
```

GetLastNodeOfDLL is teta(1) thanks to the tail of Double LL.

So, add product is O(n) because of get.



```
public void addEmployee(BranchEmployee emp,int branchIndex)throws IndexOutOfBoundsException{
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    Branch branch = branchesOfCompany.get(branchIndex);
    branch.getEmployees().add(emp);
    emp.setBranch(authority,branch);
    // only admins can access to Authority so only admins can add employee to the branches
}
```

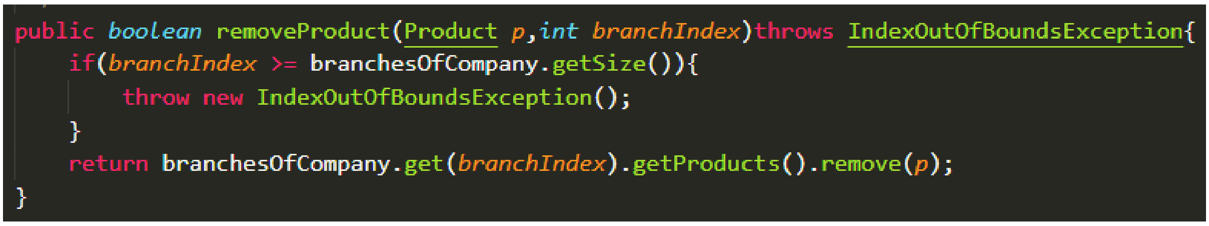
get of branches is O(n)

GetEmployees is Teta(1)

Employees hold in Arraylist so add method is amortized constant time (Teta(1))

SetBranch is Teta(1)

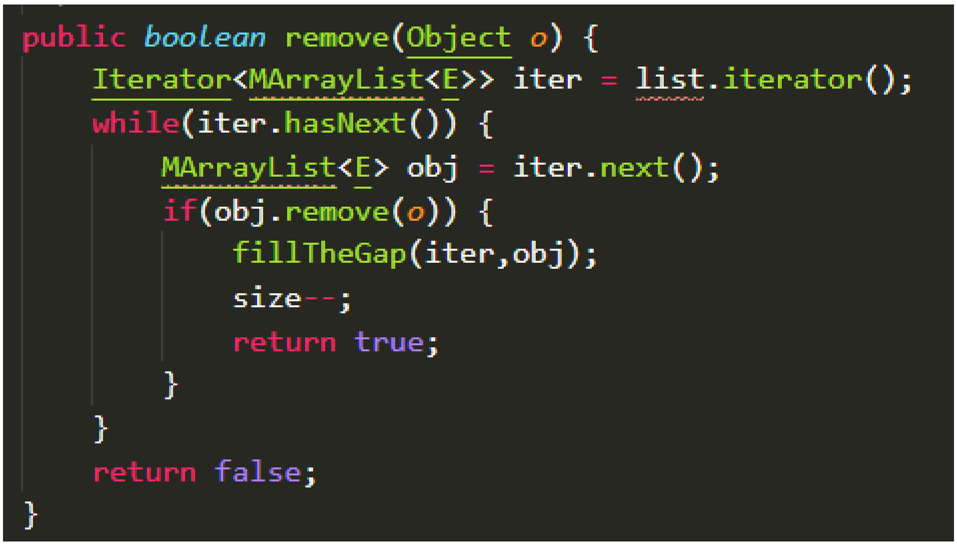
All of them : O(n)



```
public boolean removeProduct(Product p,int branchIndex)throws IndexOutOfBoundsException{
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    return branchesOfCompany.get(branchIndex).getProducts().remove(p);
}
```

Get O(n) // n number of branches

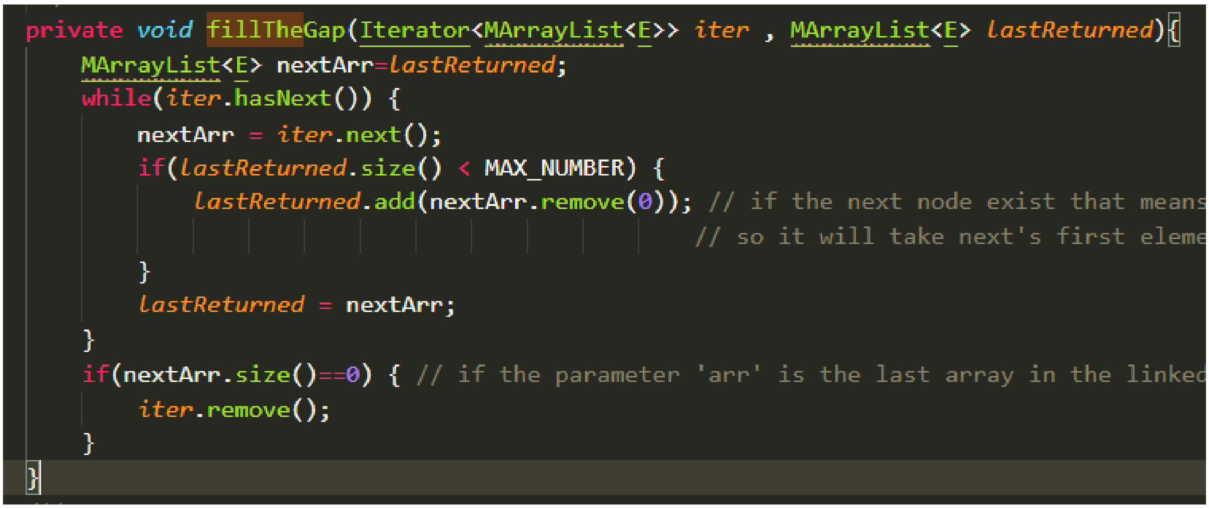
Getproducts teta(1)



```
public boolean remove(Object o) {
    Iterator<MArrayList<E>> iter = list.iterator();
    while(iter.hasNext()) {
        MArrayList<E> obj = iter.next();
        if(obj.remove(o)) {
            fillTheGap(iter,obj);
            size--;
            return true;
        }
    }
    return false;
}
```

Iterator goes through every element even if it finds the given object, because fill the gap function use it .FillTheGap function uses it to make shifts so that hybridList’s shape is not destructed. Iterator’s travel’s asymptotic notation is Teta(n) . (n is the number of nodes in the linked list which has the given iterator)

Obj.remove(o) ‘s asymptotic notation is O(m) // m is the size of given MarrayList



```
private void fillTheGap(Iterator<MArrayList<E>> iter , MArrayList<E> lastReturned){
    MArrayList<E> nextArr=lastReturned;
    while(iter.hasNext()) {
        nextArr = iter.next();
        if(lastReturned.size() < MAX_NUMBER) {
            lastReturned.add(nextArr.remove(0)); // if the next node exist that means
                                                // so it will take next's first eleme
        }
        lastReturned = nextArr;
    }
    if(nextArr.size()==0) { // if the parameter 'arr' is the last array in the linked
        iter.remove();
    }
}
```

We talked about iterator, add method’ s asymptotic notation is amortized constant time Teta(1) and it goes at most n times(n is the number of nodes in the linked list which has the given iterator)

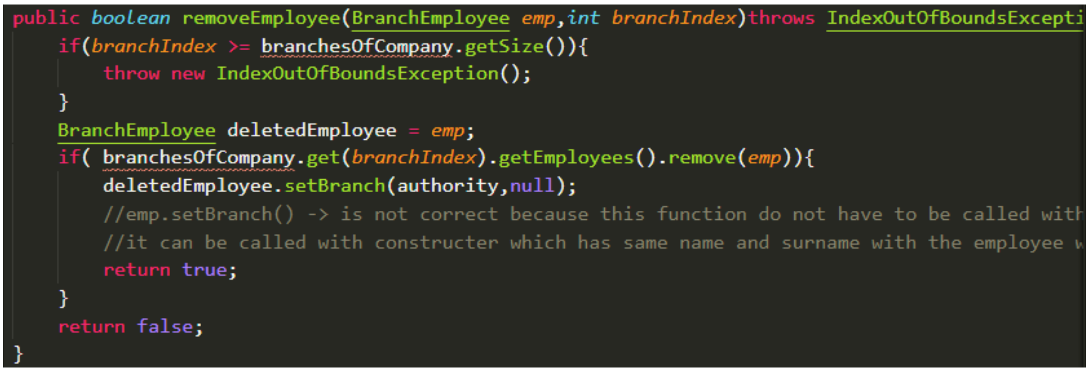
FillTheGap is called 1 time it’s asymptotic notation is O(n)

Obj.remove’s asymptotic notation is O(m) and is called n times

Remove from hybridList is max(O(m*n),O(n)) = O(m*n)

If we say that the number of elements in the HybridList is k , then we can say that it is linear time due to the number of elements in the HybridList.

HybridList’s remove is Teta(k)



```
public boolean removeEmployee(BranchEmployee emp,int branchIndex)throws IndexOutOfBoundsExcepti
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    BranchEmployee deletedEmployee = emp;
    if( branchesOfCompany.get(branchIndex).getEmployees().remove(emp)){
        deletedEmployee.setBranch(authority,null);
        //emp.setBranch() -> is not correct because this function do not have to be called with
        //it can be called with constructer which has same name and surname with the employee w
        return true;
    }
    return false;
}
```

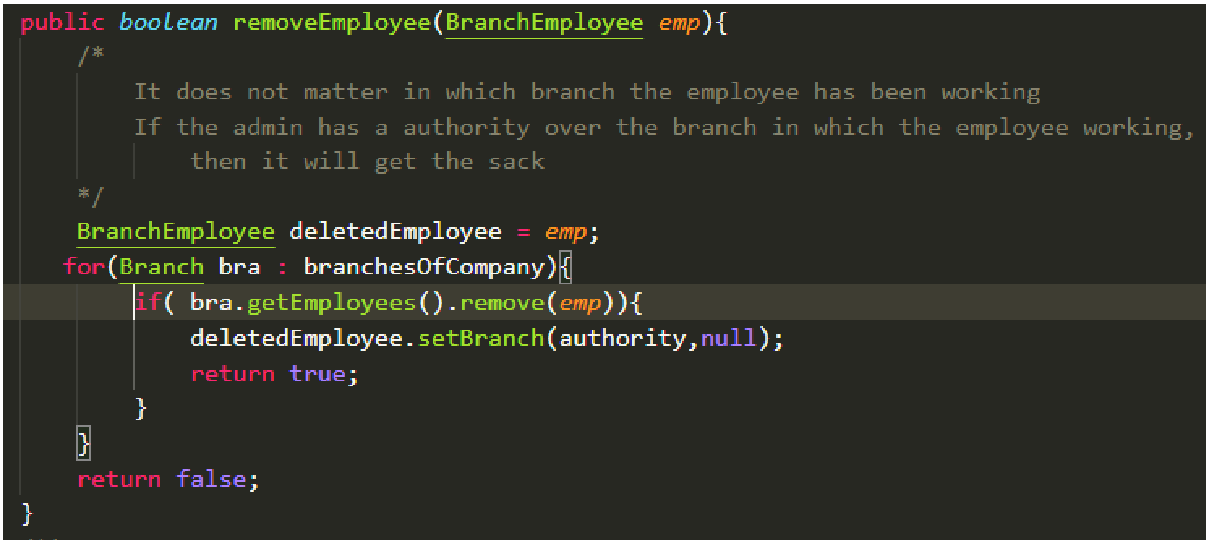
Get is O(n) n is the number of branches (because branchesOfCompany is LinkedList)

GetEmployees is Teta(1)

Remove(emp) is O(m) // m is the number of employees in the given MarrayList

SetBranch is constant

RemoveEmployee's asymptotic notation is O(m)+O(n) = O(max(m,n))



```
public boolean removeEmployee(BranchEmployee emp){
    /*
        It does not matter in which branch the employee has been working
        If the admin has a authority over the branch in which the employee working,
            then it will get the sack
    */
    BranchEmployee deletedEmployee = emp;
   for(Branch bra : branchesOfCompany){
        if( bra.getEmployees().remove(emp)){
            deletedEmployee.setBranch(authority,null);
            return true;
        }
    }
    return false;
}
```

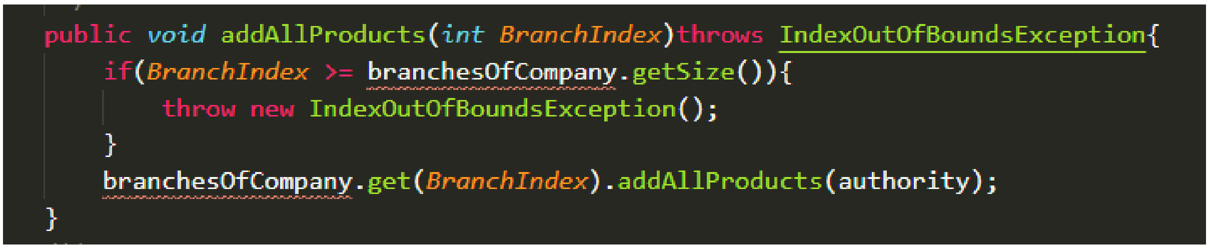
Iterator Teta(n)

GetEmployee Teta(1)

Remove emp= O(m)// number of employees in the branch which has biggest number of employees

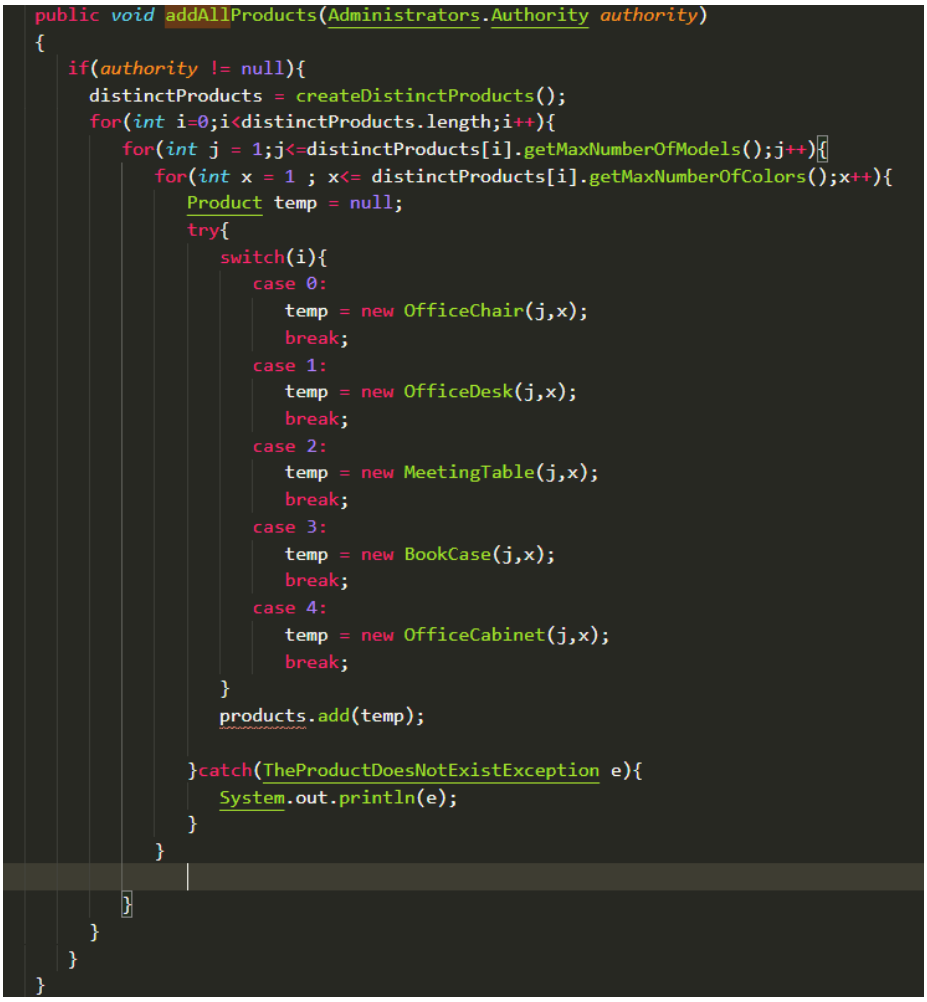
SetBranch = teta(1)

All o them = O(n*m)



```
public void addAllProducts(int BranchIndex)throws IndexOutOfBoundsException{
    if(BranchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    branchesOfCompany.get(BranchIndex).addAllProducts(authority);
}
```

Get of branchesOfCompany is O(n)

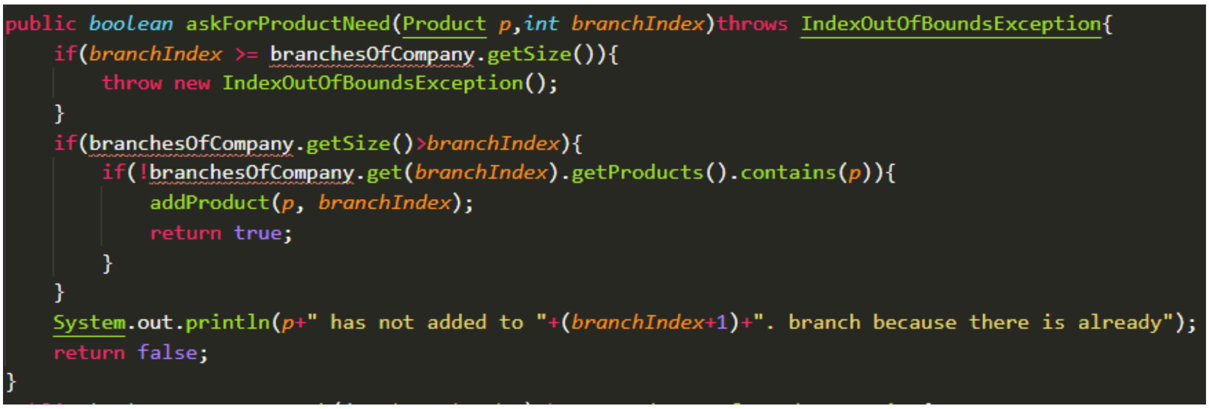


```
public void addAllProducts(Administrators.Authority authority)
{
    if(authority != null){
        distinctProducts = createDistinctProducts();
        for(int i=0;i<distinctProducts.length;i++){
            for(int j = 1;j<=distinctProducts[i].getMaxNumberOfModels();j++){
                for(int x = 1 ; x<= distinctProducts[i].getMaxNumberOfColors();x++){
                    Product temp = null;
                    try{
                        switch(i){
                            case 0:
                                temp = new OfficeChair(j,x);
                                break;
                            case 1:
                                temp = new OfficeDesk(j,x);
                                break;
                            case 2:
                                temp = new MeetingTable(j,x);
                                break;
                            case 3:
                                temp = new BookCase(j,x);
                                break;
                            case 4:
                                temp = new OfficeCabinet(j,x);
                                break;
                        }
                        products.add(temp);

                    }catch(TheProductDoesNotExistException e){
                        System.out.println(e);
                    }
                }

            }
        }
    }
}
```

AddAllProducts adds specific number of elements and add is a amortized constant time so

GetMaxNumbers are given in the pdf so they are constants.

AddAllProducts' asymptotic notation is O(n)



```java
public boolean askForProductNeed(Product p,int branchIndex)throws IndexOutOfBoundsException{
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }
    if(branchesOfCompany.getSize()>branchIndex){
        if(!branchesOfCompany.get(branchIndex).getProducts().contains(p)){
            addProduct(p, branchIndex);
            return true;
        }
    }
    System.out.println(p+" has not added to "+(branchIndex+1)+". branch because there is already");
    return false;
}
```

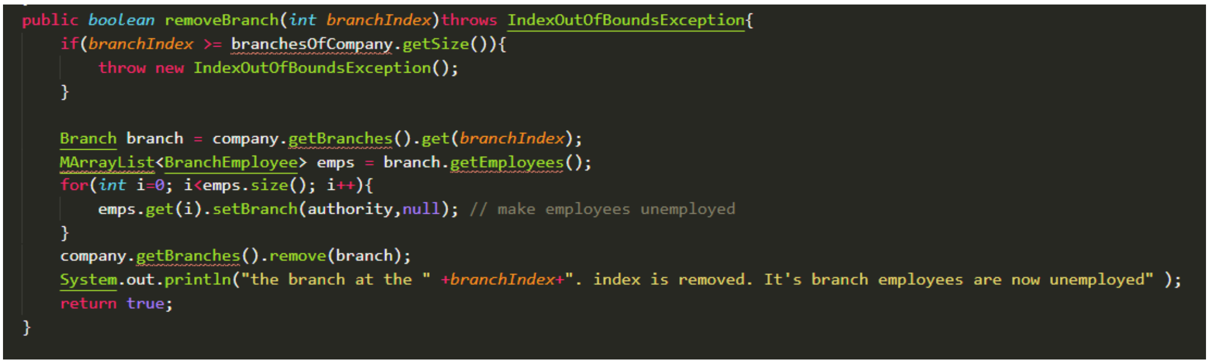
Get(branchIndex) is O(n)

GetProducts is teta(1)

Contains of HybridList is O(n) n is the number of products in the hybridlist

AddProduct is O(n) as we mentioned earlier.

So , askForProductNeed’s asymptotic notation is O(n).



```java
public boolean removeBranch(int branchIndex)throws IndexOutOfBoundsException{
    if(branchIndex >= branchesOfCompany.getSize()){
        throw new IndexOutOfBoundsException();
    }

    Branch branch = company.getBranches().get(branchIndex);
    MArrayList<BranchEmployee> emps = branch.getEmployees();
    for(int i=0; i<emps.size(); i++){
        emps.get(i).setBranch(authority,null); // make employees unemployed
    }
    company.getBranches().remove(branch);
    System.out.println("the branch at the " +branchIndex+". index is removed. It's branch employees are now unemployed" );
    return true;
}
```

GetBranches is teta(1)

Get() is O(n) // n is the number of branches

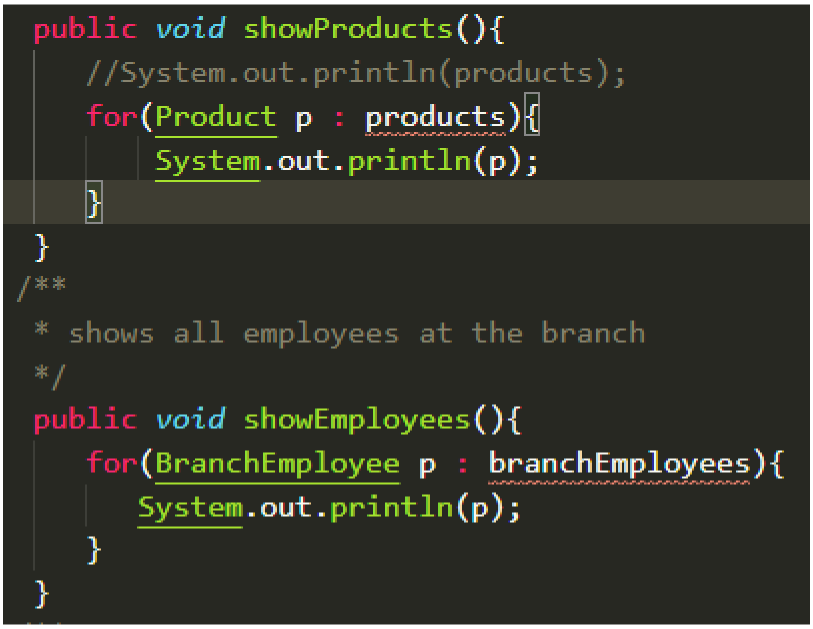
GetEmployees is teta(1)

Employees are holded in the arraylist so get is constant time set branch is constant time and they happens m (emps.size()) times so it is O(m)

Remove(branch) is O(n)

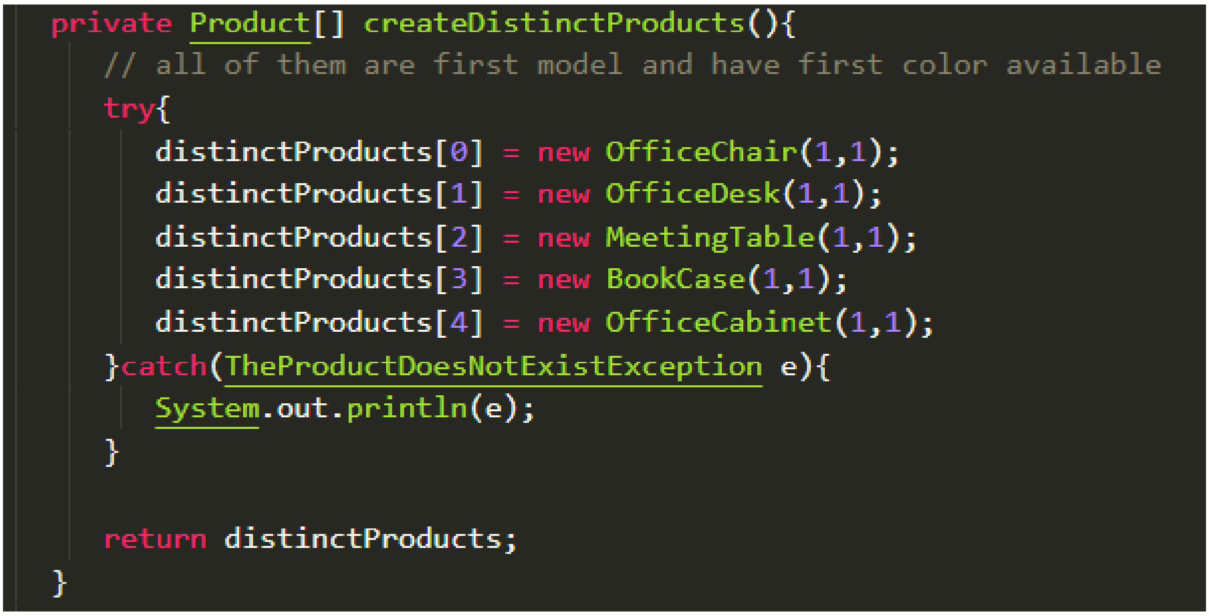
So , removeBranch’s asymptotic notation is O(max(n,m))

Branch’s methods:



```java
public void showProducts(){
    //System.out.println(products);
    for(Product p : products){
        System.out.println(p);
    }
}
/**
 * shows all employees at the branch
 */
public void showEmployees(){
    for(BranchEmployee p : branchEmployees){
        System.out.println(p);
    }
}
```

Both’s asymptotic notation is Teta(n), n is the number of elements for both



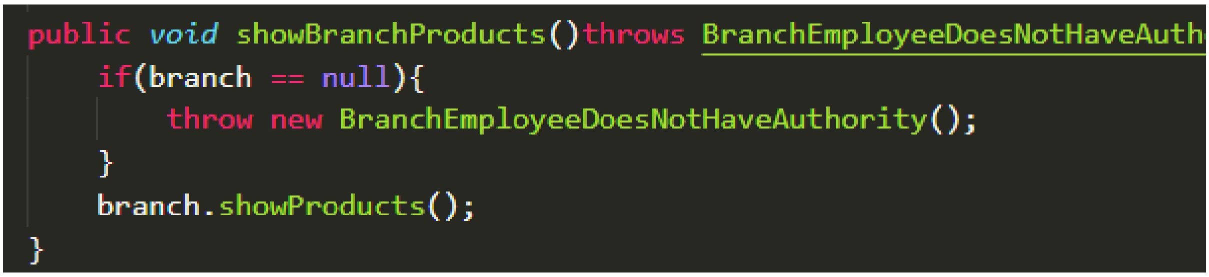
```java
private Product[] createDistinctProducts(){
    // all of them are first model and have first color available
    try{
        distinctProducts[0] = new OfficeChair(1,1);
        distinctProducts[1] = new OfficeDesk(1,1);
        distinctProducts[2] = new MeetingTable(1,1);
        distinctProducts[3] = new BookCase(1,1);
        distinctProducts[4] = new OfficeCabinet(1,1);
    }catch(TheProductDoesNotExistException e){
        System.out.println(e);
    }

    return distinctProducts;
}
```

It is a helper function and does constant time job, it’s asymptotic notation is Teta(1)

We talked about addAllProducts earlier

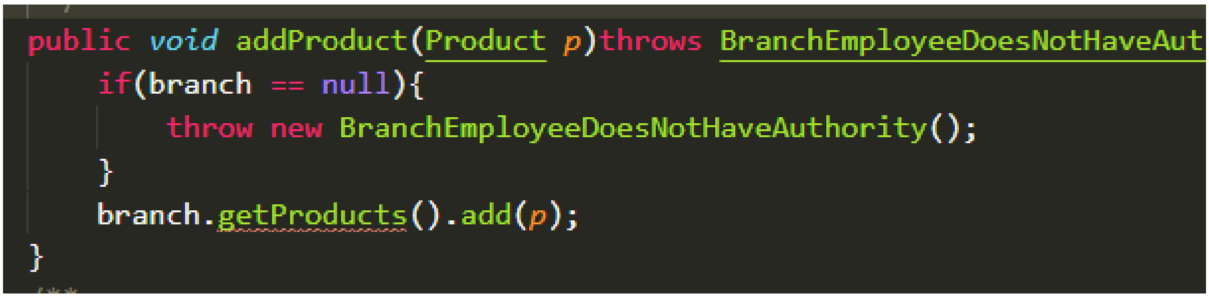
The other methods are simple get and set methods , they are all Teta(1)

BranchEmployee’s methods:



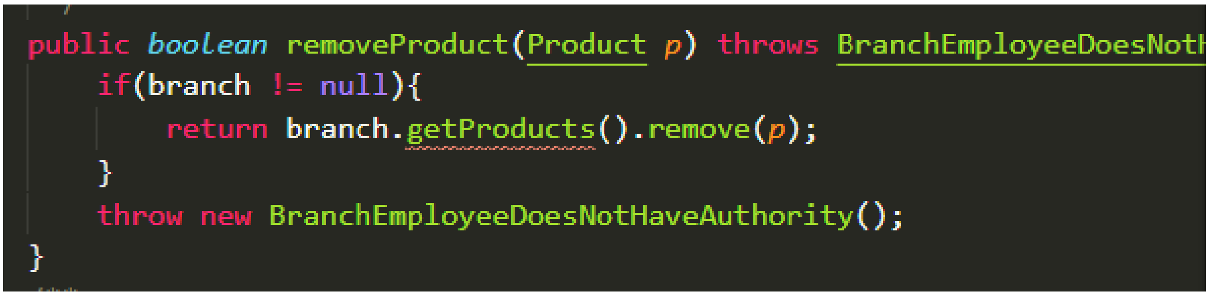
```
public void showBranchProducts()throws BranchEmployeeDoesNotHaveAuth
    if(branch == null){
        throw new BranchEmployeeDoesNotHaveAuthority();
    }
    branch.showProducts();
}
```

Branch's showProducts’ asymptotic notation is Teta(n), as we mentioned earlier, so showBranchProducts’ asymptotic notation is Teta(n) too.



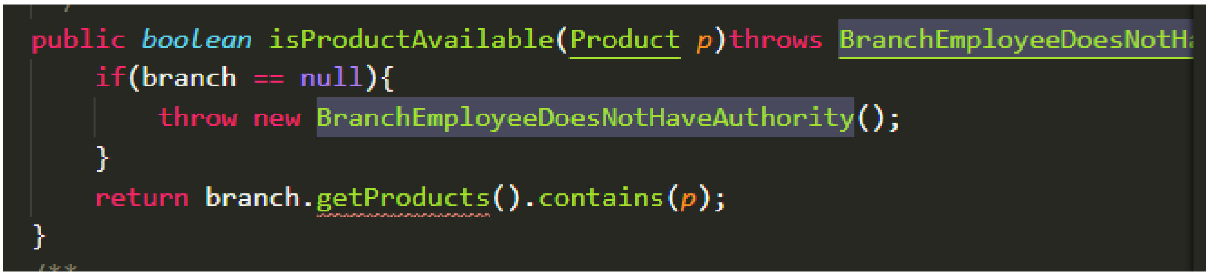
```
public void addProduct(Product p)throws BranchEmployeeDoesNotHaveAut
    if(branch == null){
        throw new BranchEmployeeDoesNotHaveAuthority();
    }
    branch.getProducts().add(p);
}
```

HybridList adds to the end in amortized constant time so this method’s asymptotic notation is Teta(1)



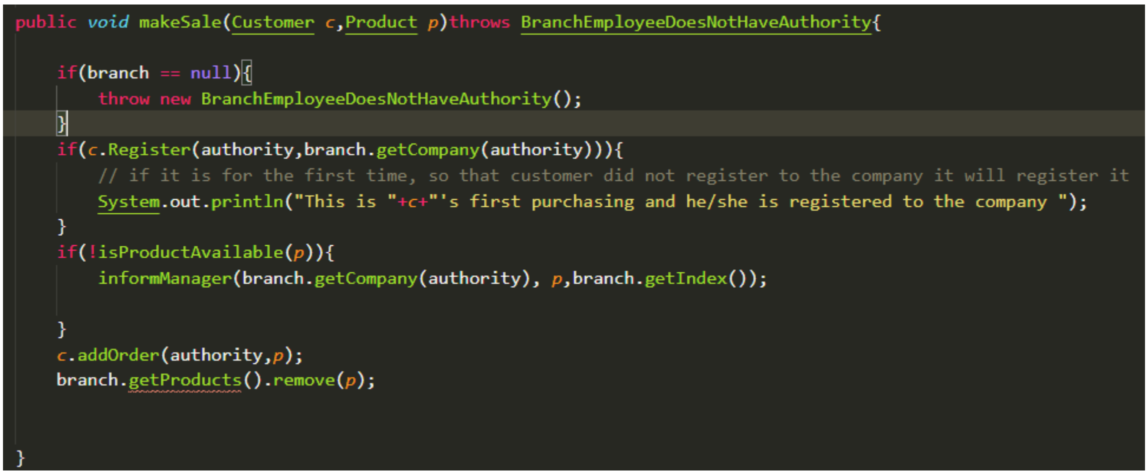
```
public boolean removeProduct(Product p) throws BranchEmployeeDoesNotH
    if(branch != null){
        return branch.getProducts().remove(p);
    }
    throw new BranchEmployeeDoesNotHaveAuthority();
}
```

Hybrid list removes in Teta(n) time because there is a shift operation. So,This methods asymptotic notation is Teta(n) too. (n represents number of elements in the HybridList)

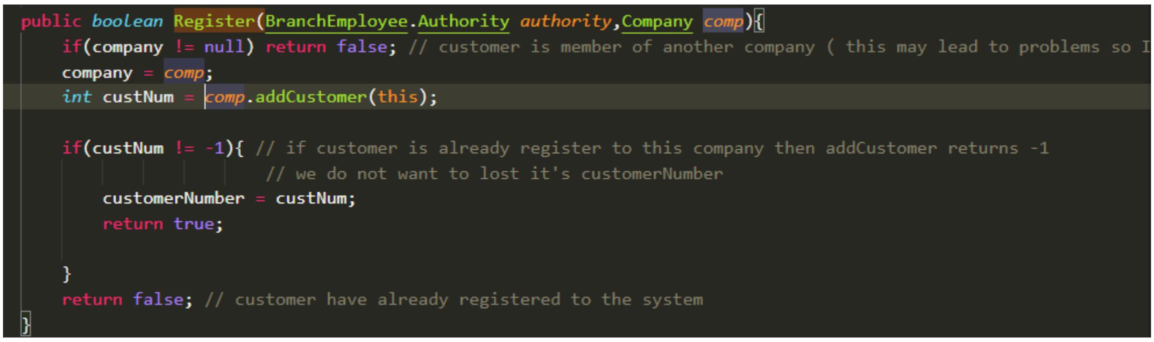


```
public boolean isProductAvailable(Product p)throws BranchEmployeeDoesNotHa
    if(branch == null){
        throw new BranchEmployeeDoesNotHaveAuthority();
    }
    return branch.getProducts().contains(p);
}
```

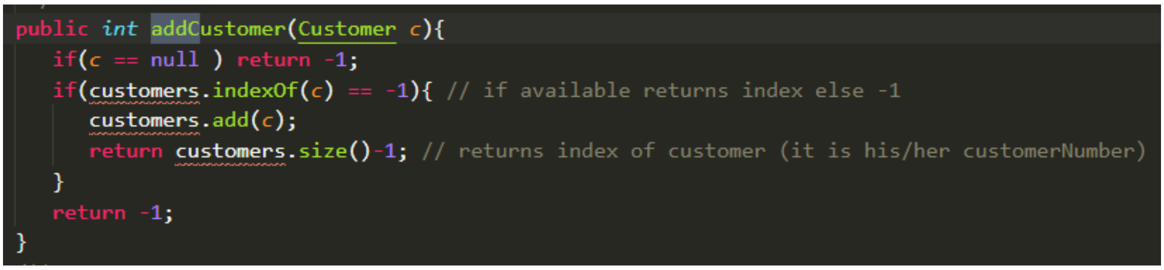
HybridList contains method takes lineer time so , this method asymptotic notation is O(n)



```java
public void makeSale(Customer c,Product p)throws BranchEmployeeDoesNotHaveAuthority{

    if(branch == null){
        throw new BranchEmployeeDoesNotHaveAuthority();
    }
    if(c.Register(authority,branch.getCompany(authority))){
        // if it is for the first time, so that customer did not register to the company it will register it
        System.out.println("This is "+c+"'s first purchasing and he/she is registered to the company ");
    }
    if(!isProductAvailable(p)){
        informManager(branch.getCompany(authority), p,branch.getIndex());

    }
    c.addOrder(authority,p);
    branch.getProducts().remove(p);


}
```



```java
public boolean Register(BranchEmployee.Authority authority,Company comp){
    if(company != null) return false; // customer is member of another company ( this may lead to problems so I
    company = comp;
    int custNum = comp.addCustomer(this);

    if(custNum != -1){ // if customer is already register to this company then addCustomer returns -1
                        // we do not want to lost it's customerNumber
        customerNumber = custNum;
        return true;

    }
    return false; // customer have already registered to the system
}
```

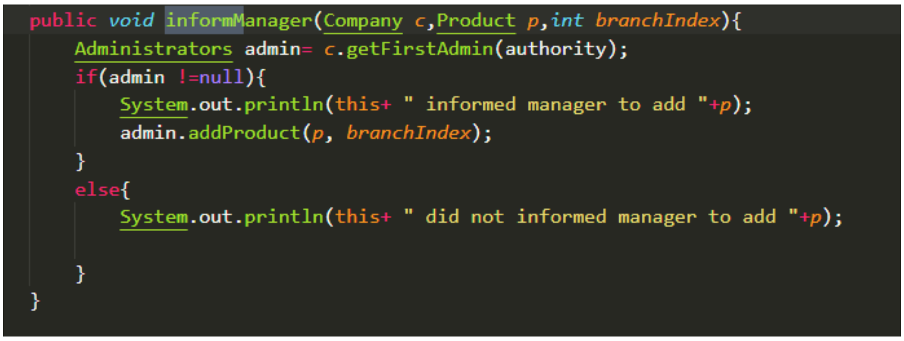


```java
public int addCustomer(Customer c){
   if(c == null ) return -1;
   if(customers.indexOf(c) == -1){ // if available returns index else -1
      customers.add(c);
      return customers.size()-1; // returns index of customer (it is his/her customerNumber)
   }
   return -1;
}
```

Add customer takes amortized constant time so, this method takes Teta(1)

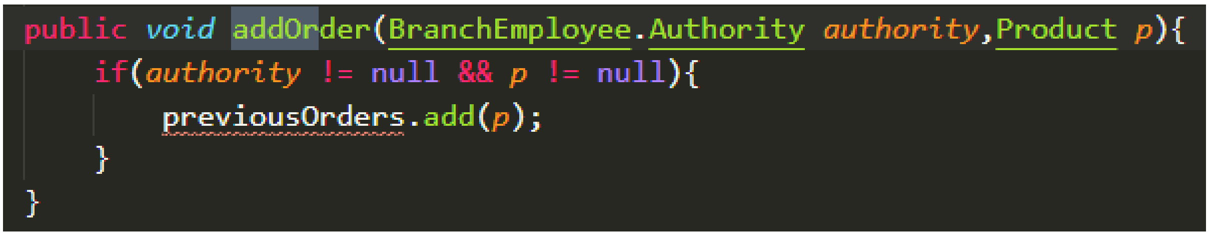
Register takes Teta(1) too.

IsProductAvailable's asymptotic notation is O(n)



```java
public void informManager(Company c,Product p,int branchIndex){
    Administrators admin= c.getFirstAdmin(authority);
    if(admin !=null){
        System.out.println(this+ " informed manager to add "+p);
        admin.addProduct(p, branchIndex);
    }
    else{
        System.out.println(this+ " did not informed manager to add "+p);

    }
}
```

InformManager takes teta(1)



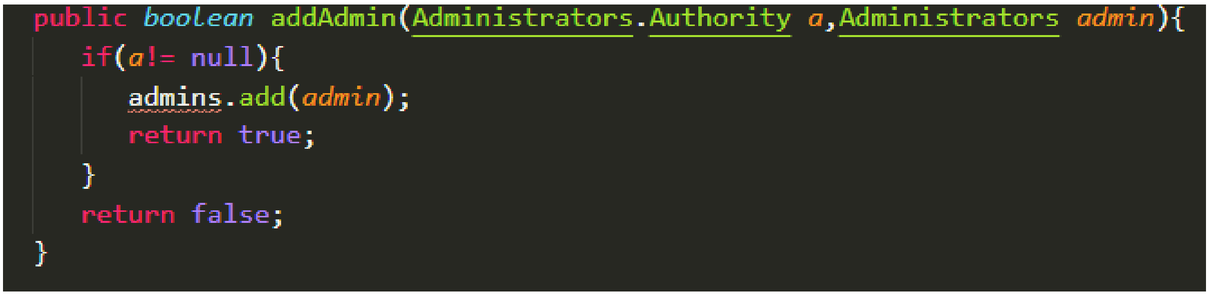
```
public void addOrder(BranchEmployee.Authority authority,Product p){
    if(authority != null && p != null){
        previousOrders.add(p);
    }
}
```

AddOrder takes amortized constant time = Teta(1)

Remove product takes Teta(n) because of the search and shift operation of HybridList.

So, make sale’s asymptotic notation is O(n)

Company’s methods:

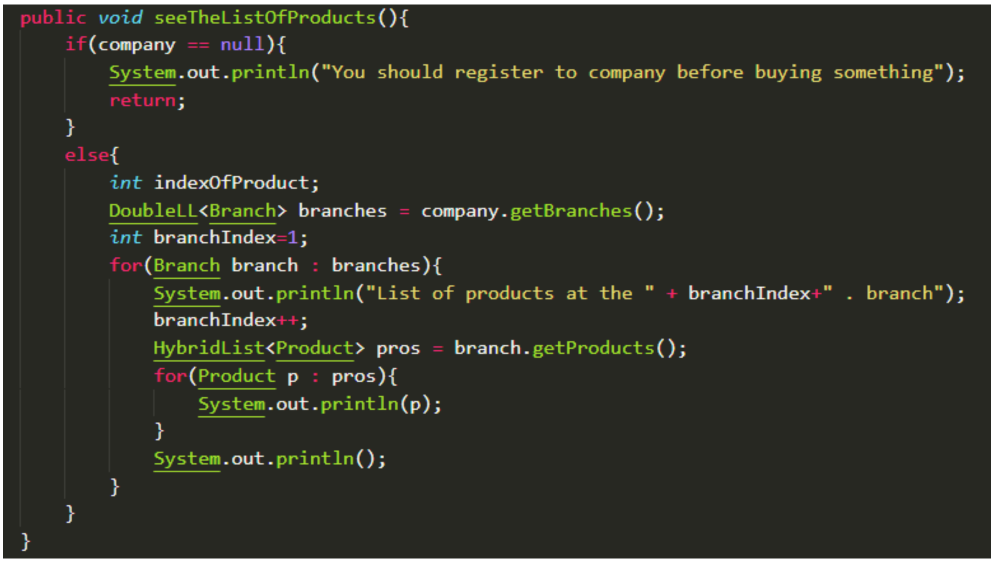


```
public boolean addAdmin(Administrators.Authority a,Administrators admin){
    if(a!= null){
        admins.add(admin);
        return true;
    }
    return false;
}
```

AddAdmin's running time is amortized constant time , so teta(1)

Other methods of Company are get and set methods they are all Teta(1)

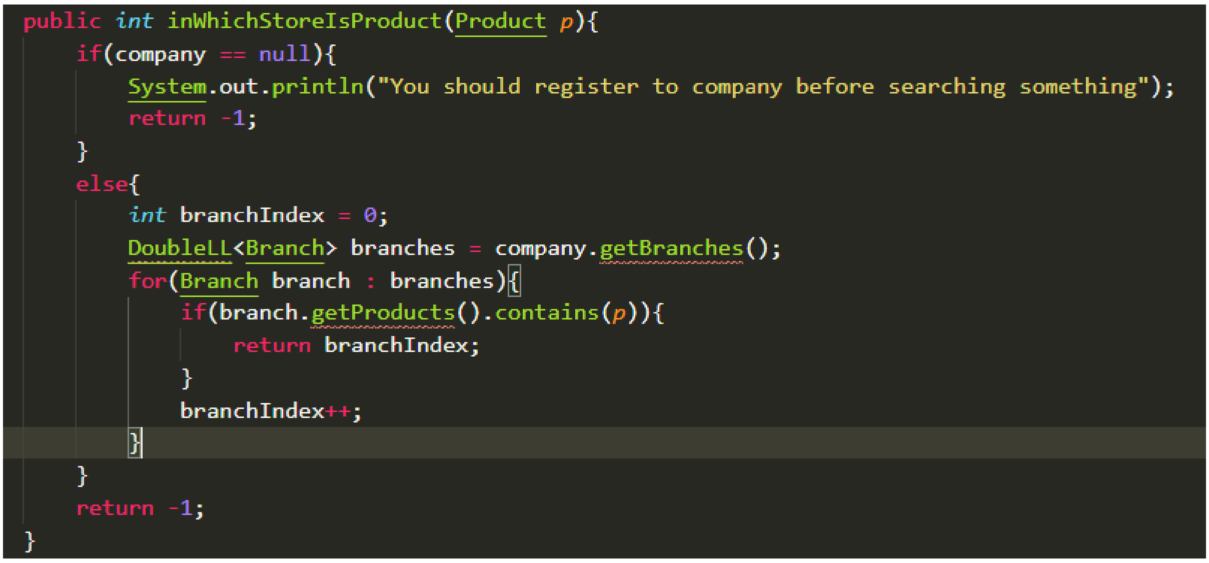
Customer’s methods:



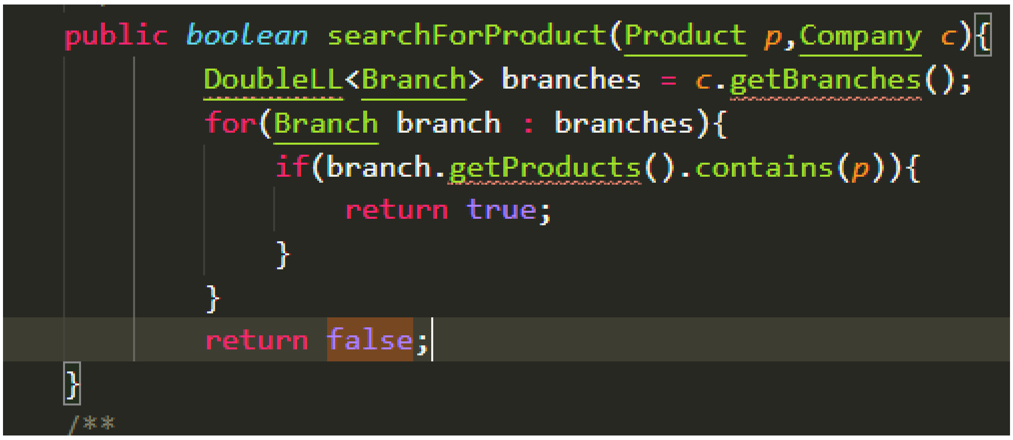
```
public void seeTheListOfProducts(){
    if(company == null){
        System.out.println("You should register to company before buying something");
        return;
    }
    else{
        int indexOfProduct;
        DoubleLL<Branch> branches = company.getBranches();
        int branchIndex=1;
        for(Branch branch : branches){
            System.out.println("List of products at the " + branchIndex+" . branch");
            branchIndex++;
            HybridList<Product> pros = branch.getProducts();
            for(Product p : pros){
                System.out.println(p);
            }
            System.out.println();
        }
    }
}
```

It’s asymptotic notation is Teta(n) teta is the number of products in the company,

n= m*k (m is the number of branches, k is the number of products at that branch)

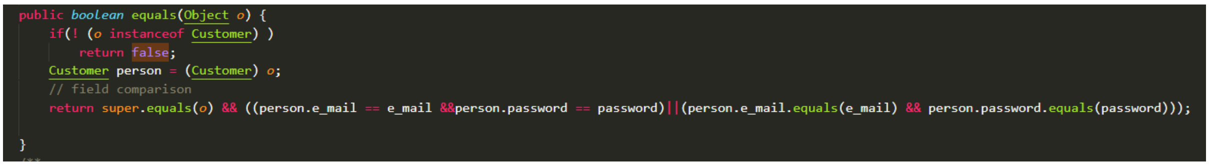


```
public int inWhichStoreIsProduct(Product p){
    if(company == null){
        System.out.println("You should register to company before searching something");
        return -1;
    }
    else{
        int branchIndex = 0;
        DoubleLL<Branch> branches = company.getBranches();
        for(Branch branch : branches){
            if(branch.getProducts().contains(p)){
                return branchIndex;
            }
            branchIndex++;
        }
    }
    return -1;
}
```



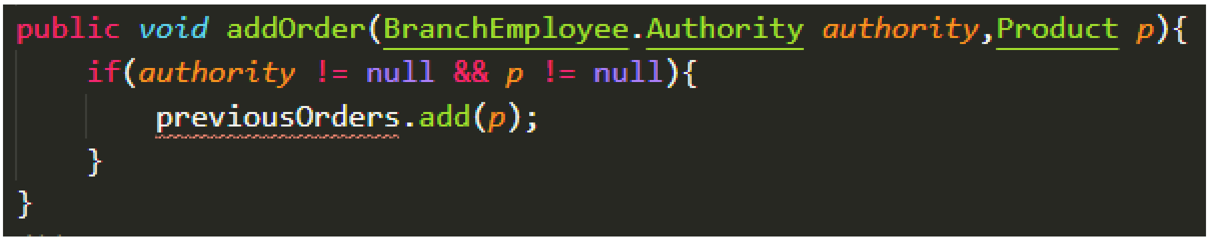
```
    public boolean searchForProduct(Product p,Company c){
        DoubleLL<Branch> branches = c.getBranches();
        for(Branch branch : branches){
            if(branch.getProducts().contains(p)){
                return true;
            }
        }
        return false;
    }
    /**
```

These 2 method’s running times are proportional to the products in the company, So it is O(n) , n is the number of products in the company.



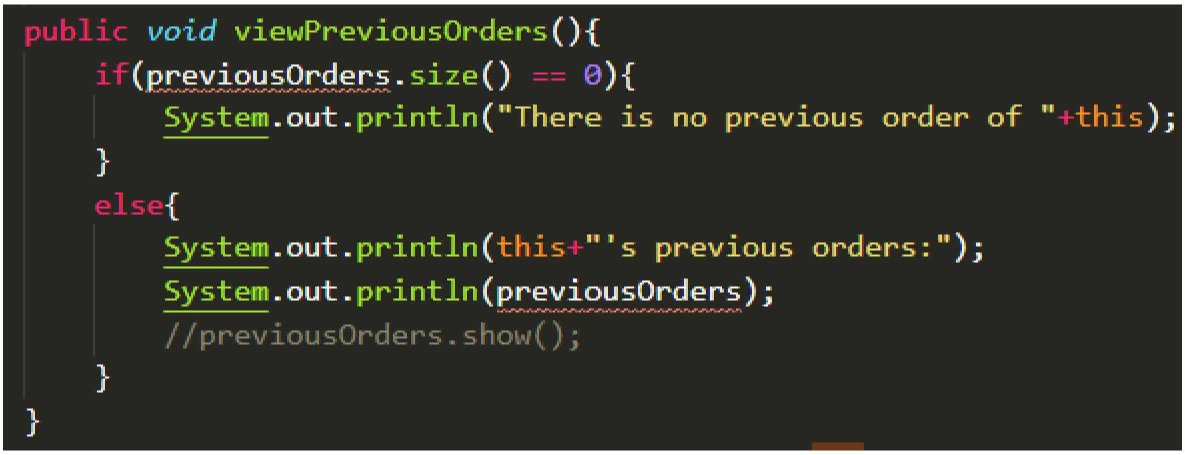
```
public boolean equals(Object o) {
    if(! (o instanceof Customer) )
        return false;
    Customer person = (Customer) o;
    // field comparison
    return super.equals(o) && ((person.e_mail == e_mail &&person.password == password)||(person.e_mail.equals(e_mail) && person.password.equals(password)));

}
```

Equals method’s asmptotic notation is teta(1)



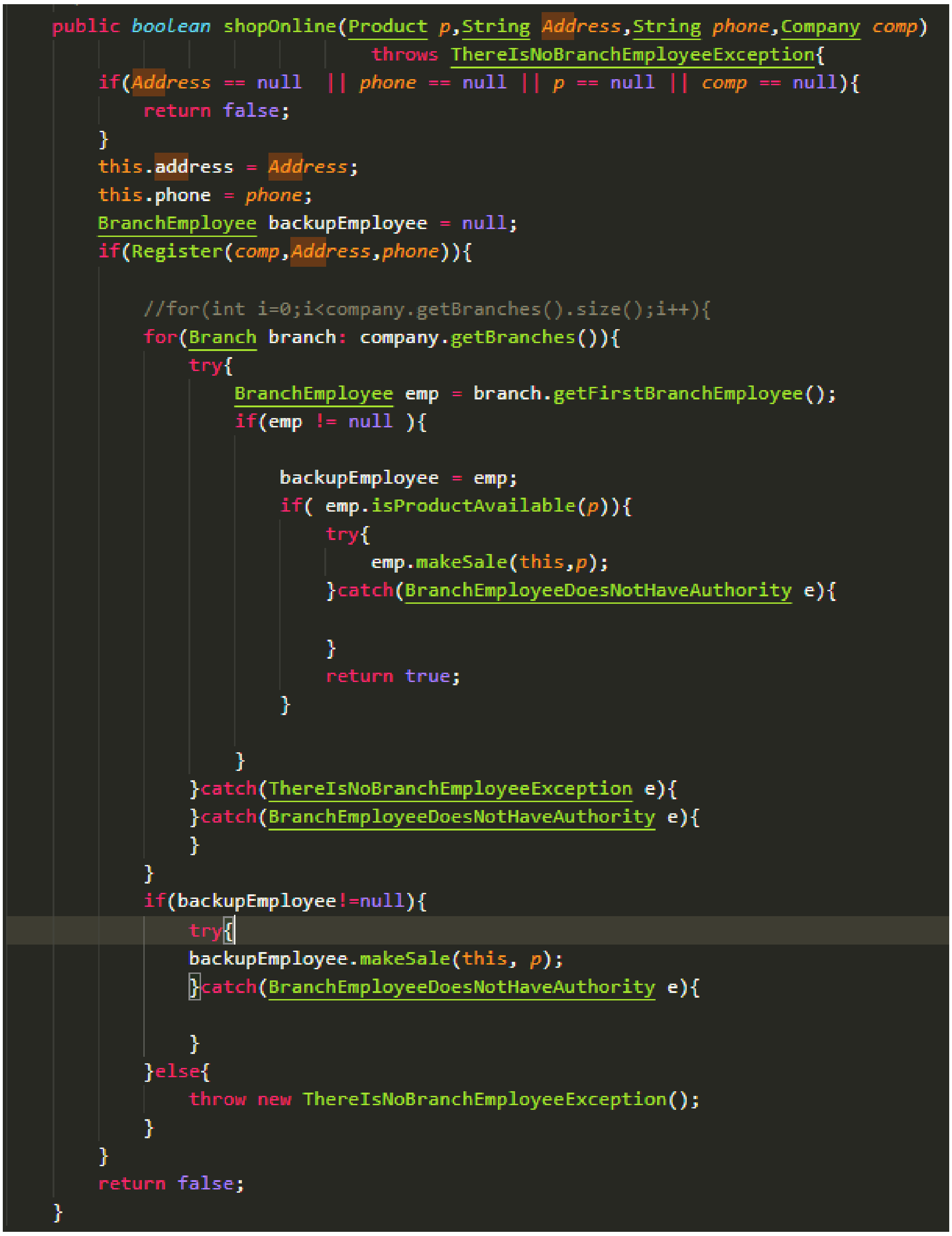
```
public void addOrder(BranchEmployee.Authority authority,Product p){
    if(authority != null && p != null){
        previousOrders.add(p);
    }
}
```

PreviousOrders is instance of HybridList so this addition is constant time operation ==> teta(1)



```
public void viewPreviousOrders(){
    if(previousOrders.size() == 0){
        System.out.println("There is no previous order of "+this);
    }
    else{
        System.out.println(this+"'s previous orders:");
        System.out.println(previousOrders);
        //previousOrders.show();
    }
}
```

If n represent the number of previous orders of the customer , then this function’s running time is Teta(n)



```
public boolean shopOnline(Product p,String Address,String phone,Company comp)
                            throws ThereIsNoBranchEmployeeException{
    if(Address == null  || phone == null || p == null || comp == null){
        return false;
    }
    this.address = Address;
    this.phone = phone;
    BranchEmployee backupEmployee = null;
    if(Register(comp,Address,phone)){

        //for(int i=0;i<company.getBranches().size();i++){
        for(Branch branch: company.getBranches()){
            try{
                BranchEmployee emp = branch.getFirstBranchEmployee();
                if(emp != null ){

                    backupEmployee = emp;
                    if( emp.isProductAvailable(p)){
                        try{
                            emp.makeSale(this,p);
                        }catch(BranchEmployeeDoesNotHaveAuthority e){

                        }
                        return true;
                    }

                }
            }catch(ThereIsNoBranchEmployeeException e){
            }catch(BranchEmployeeDoesNotHaveAuthority e){
            }
        }
        if(backupEmployee!=null){
            try{
            backupEmployee.makeSale(this, p);
            }catch(BranchEmployeeDoesNotHaveAuthority e){

            }
        }else{
            throw new ThereIsNoBranchEmployeeException();
        }
    }
    return false;
}
```

Register = teta(1)

For loop takes O(n) // n is the number of branches

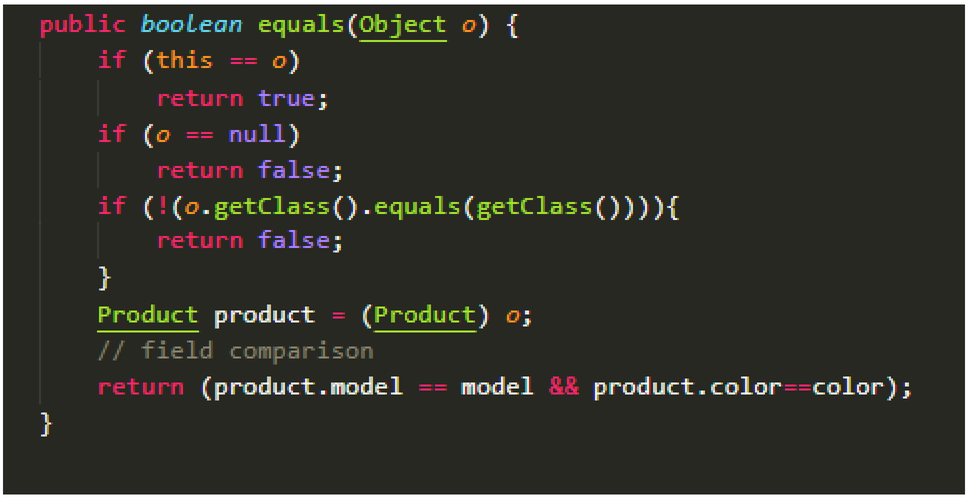
GetFirstBranchEmployee teta(1)

IsProductAvailable takes O(m) // m is the number of the products in the given branch

makeSale's asymptotic notation is O(k) because of remove method of makeSale. k represents the number of products in the branch which is the employee is working.

So it is O(n*m) + O(k) so it is O(n*m) and n represent the number of branches and m represents the number of products in the given branch , if we say that b is the number of products in the company, then this method takes O(b) time.

Product’s methods:



```
public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null)
        return false;
    if (!(o.getClass().equals(getClass()))){
        return false;
    }
    Product product = (Product) o;
    // field comparison
    return (product.model == model && product.color==color);
}
```

It is teta(1)

All of the product class’ method is teta(1). They are simple methods.