GIT

DEPARTMENT OF COMPUTER ENGINEERING

CSE 222/505 – Spring 2021

REPORT FOR HOMEWORK 1

SÜLEYMAN GÖLBOL

1801042656

1. ANALYZING TIME COMPLEXITY

When we are using T_{sum} and multiply the time cost with number of times executed, cost doesn't matter because we know it and it is constant. For example,

(for i=0 to n)

    sum = sum + 1;

return sum;

return sum; , i++, i<n and *sum = sum + 1;* normally takes constant time. But for loop will work n+1 times so *sum = sum + 1;* will work n times. Θ(n) + Θ(1) + Θ(1)+ Θ(1) = O(n) because constant time doesn't effect when there is a bigger degree like Θ(n).

So when I see a for loop like for i=0 to n, I will directly use Θ(n) (to not mention same thing in every part.)

Also the getters that just returns something, and setters that equalize something will cost Θ(1) – constant time. So I won't include them.

```
public void addBranch(Branch branch){
    allBranches.add(allBranches.size(), branch);
}
```

```
public void add(int index, E obj) {
    listIterator(index).add(obj);
}
```

```
public void add(E obj) {
    if (head == null) { // Add to an empty list.
        head = new Node<E>(obj);
        tail = head;
    }
    else if (nextItem == head) { // Insert at head.
        // Create a new node.
        Node<E> newNode = new Node<E>(obj);
        // Link it to the nextItem.
        newNode.next = nextItem; // Step 1
        // Link nextItem to the new node.
        nextItem.prev = newNode; // Step 2
        // The new node is now the head.
        head = newNode; // Step 3
    }
    else if (nextItem == null) { // Insert at tail.
        // Create a new node.
        Node<E> newNode = new Node<E>(obj);
        // Link the tail to the new node.
        tail.next = newNode; // Step 1
        // Link the new node to the tail.
        newNode.prev = tail; // Step 2
        // The new node is the new tail.
        tail = newNode; // Step 3
    }
    else { // Insert into the middle.
        // Create a new node.
        Node<E> newNode = new Node<E>(obj);
        // Link it to nextItem.prev.
        newNode.prev = nextItem.prev; // Step 1
        nextItem.prev.next = newNode; // Step 2
        // Link it to the nextItem.
        newNode.next = nextItem; // Step 3
        nextItem.prev = newNode; // Step 4
    }

    // Increase size and index and set lastItemReturned.
    size++;
    index++;
    lastItemReturned = null;
} // End of method add.
```

It works in Θ(1) because there are no any loop, all of them are constant time operations. Another important note for that listIterator's travelling in linkedlist takes linear time but we think as we are already in that node so it is not Θ(n),it is Θ(1).

```java
public void addAdministrator(Admins administrator){
    allAdmins.add(administrator);
}
```

```java
public boolean add(E anEntry) {
    if (size == capacity)
        reallocate();
    theData[size] = anEntry;
    size++;
    return true;
}
```

```java
private void reallocate() {
    capacity = 2 * capacity;
    theData = Arrays.copyOf(theData, capacity);
}
```

In best case it doesn't reallocate so Θ(1), in worst case it is Θ(n). Because reallocate uses copyOf method which is Θ(n). So average case becomes Θ(n/2) which is Θ(n).

```java
public void addCustomer(Customers customer){
    allCustomers.add(customer);
}
```

addCustomer method's add(E) method is same with above method. (Screenshot of add(E) is above). In best case it Θ(1), in worst case it is Θ(n). Because reallocate uses copyOf method which is Θ(n). So average case becomes Θ(n/2) which is Θ(n).

```java
public void removeBranch(Branch b){
    allBranches.remove(b);
}
```

```java
public void remove(E element){
    if(tail==null)
        throw new IllegalStateException();
    Node<E> temp = head;
    int counter=0;  //Counter to find inde
    while(temp != null){
        counter++;
        if(temp.data == element){
            remove(counter);
            break;
        }
        temp = temp.next;
    }
}
```

```java
public void remove(int index) {
    listIterator(index).remove();
}
```

```java
public void remove(){
    if(tail==null)
        throw new IllegalStateException();
    tail = tail.prev;
    tail.next = null;
    size--;
}
```

We don't calculate throw errors. In best case loop runs one time so Θ(1) but in worst case loop runs until the end so Θ(n). So average case becomes Θ(n/2) = Θ(n). Note that normally listIterator's remove method is Θ(1) but I used a another different method overloaded. Because of that it is Θ(n).

```java
public boolean checkCustomerNumber(int customerNumber){
    for(int i=0; i<allCustomers.size(); i++){
        if(allCustomers.get(i).getCustomerNumber() == customerNumber)
            return true;
    }
    return false;
}
```

Getters take constant time so for loop runs until the size() cost is Θ(n).

```java
public void addBranch(Company c, Branch b){
    c.addBranch(b);
}
```

c is company variable and company's addBranch method already calculated above and it is Θ(1).

```java
public void addBranchEmployee(Branch branch, BranchEmployees bE){
    branch.addBranchEmployee(bE);
}
```

```java
public void addBranchEmployee(BranchEmployees employee){
    branchEmployees.add(branchEmployees.size(), employee);
}
```

```java
public void add(int index, E obj) {
    listIterator(index).add(obj);
}
```

We also calculated this above it was Θ(1).

```java
public void removeBranchEmployee(Branch b, BranchEmployees employee){
    b.removeBranchEmployee(employee);
}
```

```java
public void removeBranchEmployee(BranchEmployees employee){
    branchEmployees.remove(employee);
}
```

The same remove() method calculated above and it was Θ(n).

```java
public void removeBranch(Company c, Branch b){
    c.removeBranch(b);
}
```

```java
public void removeBranch(Branch b){
    allBranches.remove(b);
}
```

The same remove() method calculated above and it was Θ(n).

```java
public boolean queryToSupply(Company c, int customerNumber){
    for(int i=0; i<c.getAllCustomers().size(); i++)
        if( c.getAllCustomers().get(i).getCustomerNumber() == customerNumber ){
            for(int j=0; j<c.getAllCustomers().get(i).getPreviousOrders().size(); j++)
                if( c.getAllCustomers().get(i).getPreviousOrders().get(j).substring(0, 1).equals("N") ){
                    char[] temp = c.getAllCustomers().get(i).getPreviousOrders().get(j).toCharArray();
                    temp[0] = 'S'; //S means supplied.
                    String s = String.valueOf(temp);
                    c.getAllCustomers().get(i).setPreviousOrders(s, j);
                    return true;
                }
        }

    return false;
}
```

Setters/getters are Θ(1). toCharArray() and substring() methods Θ(n). Nested for loops Θ(n*n). The biggest degree matters so $\Theta(n^2)$.

```java
public void addLessAmount(LessAmount l){
    lessAmounts.add(l);
}
```

```java
public boolean add(E anEntry) {
    if (size == capacity)
        reallocate();
    theData[size] = anEntry;
    size++;
    return true;
}
```

This method calculated above. Best case Θ(1), average and worst case it costs Θ(n).

```java
public void isThereLessAmount(Company c, int newStock){
    int branchIndex, productIndex;
    if(lessAmounts.size() == 0){
        System.out.println("There is no less amount product.");
    }

    else if(lessAmounts.size() != 0){
        System.out.println("Less amont product found and now stock is increasing.");
        for(int i=0; i<lessAmounts.size(); i++){
            branchIndex = lessAmounts.get(i).getBranchIndex();
            productIndex = lessAmounts.get(i).getProductIndex();
            c.getAllBranches().get(branchIndex).getProducts().get(productIndex).setStock(newStock);
        }
    }
}
```

In best case, it enters first if and Θ(1). In worst case it enters in else if and because of loop it takes Θ(n).So average case is Θ(n/2) = Θ(n).

```java
public void addProduct(Products product){
    products.add(product);
}
```

```java
public void add(E e){
    int llSize = linkedList.size();
    if(llSize == 0) {
        linkedList.add( llSize , new KWArrayList<E>() );
        linkedList.get(0).add(e);
        return;
    }

    if(checkIsArrayListFull() != 0){ //arrayList is not full
        linkedList.get(llSize-1).add( e );
    }
    else{ //arraylist is full.
        linkedList.add( llSize , new KWArrayList<E>() );
        linkedList.get(llSize).add(e);
    }

}
```

Both linkedlist's and arraylist's add() costs already showed above. For arraylist, best case Θ(1), average/worst case Θ(n). For linkedlist, it costs Θ(1). So best case is Θ(1) and average/worst case Θ(n).

```java
public void addProductArray(HybridList<Products> newProducts){
    products.clear();
    this.products = newProducts;
}
```

```java
public void clear(){
    head = null;
    tail = null;
    size = 0;
}
```

Constant time so it will cost Θ(1).

```java
public void addBranchEmployee(BranchEmployees employee){
    branchEmployees.add(branchEmployees.size(), employee);
}
```

Add method already showed above for linkedlists which is Θ(1).

```java
public void removeBranchEmployee(BranchEmployees employee){
    branchEmployees.remove(employee);
}
```

Remove method already showed above which is Θ(n).

```java
public void clear(){
    head = null;
    tail = null;
    size = 0;
}
```

Constant time so Θ(1).

```java
public String toString(){
    return branchName;
}
```

Constant time so Θ(1).

```java
public int findProductIndex(Products.Product productType, Colors color, int model){
    int flag = -1;
    for(int i=0; i<products.size(); i++){
        if(productType == products.get(i).getProduct() && color == products.get(i).getColor() && model == products.get(i).getModel() )
            flag = i;
    }
    return flag;
}
```

Linear time Θ(n) because of for loop runs n times. Getters and setters constant time so won't effect.

```java
public void addProduct(Branch branchIn, Products product){
    branchIn.getProducts().add(product);
}
```

It was calculated HybridList's add method already above which was best case is Θ(1) and average/worst case Θ(n).

```java
public void addProduct(Branch branchIn, HybridList<Products> products){
    branchIn.addProductArray(products);
}
```

HybridList (already above calculated) best case is Θ(1) and average/worst case Θ(n).

```java
public void removeProduct(Branch branchIn, Products product){
    branchIn.getProducts().remove(product);
}
```

```java
public void remove(E element){
    int counter = 0;
    for(int i=0; i<linkedList.size(); i++)
        for(int j=0; j<linkedList.get(i).size(); j++){
            if(linkedList.get(i).get(j) == element){
                remove(counter);
                break;
            }
            else
                counter++;
        }
}
```

```java
public void remove(int index){
    if(linkedList.size() == 0)
        throw new NoSuchElementException();
    int remaining = index % MAX_NUMBER;
    int quotient  = index / MAX_NUMBER;

    linkedList.get(quotient).remove(remaining); //Removing last element from arrayList.
    //Checking if last arrayList in linkedList is empty or not.
    if( linkedList.get(quotient).size() == 0 )
        /*linkedList.remove();*/ linkedList.listIterator( linkedList.size()-1 ).remove();
}
```

```java
public E remove(int index) {
    if (index < 0 || index >= size)
        throw new ArrayIndexOutOfBoundsException(index);
    E returnValue = theData[index];
    for (int i = index + 1; i < size; i++)
        theData[i-1] = theData[i];
    size--;
    return returnValue;
}
```

Best case $\Theta(n^2)$ because of 2 nested for loops and doesn't enter in another remove method but worst/average case is $\Theta(n^3)$ because it enters another remove method which also has another for loop inside of it.(The reason for that not to make arraylist has empty spaces on the middle etc so it is for shifting)

```java
public void removeProduct(Branch branchIn, Products.Product product, Colors color, int model){
    branchIn.removeProduct(product, color, model);
}
```

```java
public void removeProduct(Products.Product productType, Colors color, int model){
    boolean flag = false;
    for(int i=0; i<products.size(); i++){
        if(productType == products.get(i).getProduct() && color == products.get(i).getColor() && model == products.get(i).getModel() ){
            products.remove(i);
            flag = true;
        }
    }
    if(flag == false)
        System.err.println("Product you want to delete couldn't find.");
}
```

Products.remove(int) was $\Theta(n)$ which is showed above already and because of for loop, best case is $\Theta(n)$ which never enters in if statement and worst/average case is $\Theta(n^2)$ which enters in if so calls the product's remove method and $n*n=n^2$.

```java
public int inquireStock(Products.Product productType, Colors color, int model){
    int index = -1;
    index = branchIn.findProductIndex(productType, color, model);
    try{
        if(index == -1) throw new NullPointerException();
    }catch(NullPointerException e){
        System.err.println("There is no product for that model. Please create product first.");
        return -1;
    }
    return branchIn.getProducts().get( index ).getStock();
}
```

findProductIndex was already calculated above and which was $\Theta(n)$. if, return and getters will cost $\Theta(1)$ so in general $\Theta(n)$.

```java
public boolean isAmountEnough(Products.Product productType, Colors color, int model, int requestedAmount){
    if( inquireStock(productType, color, model) < requestedAmount )
        return false;
    return true;
}
```

inquiereStock is above and it was $\Theta(n)$ so this is too.

```
public void informManager(Company c, Admins a, Products.Product productType, Colors color, int model, int requestedAmount){
    int branchIndex, productIndex;
    if(isAmountEnough(productType, color, model, requestedAmount) == false){
        LessAmount l;
        for(int i=0; i<c.getAllBranches().size(); i++)
            if( c.getAllBranches().get(i) == branchIn ){
                branchIndex = i;
                productIndex = c.getAllBranches().get(i).findProductIndex(productType, color, model);
                l = new LessAmount(branchIndex, productIndex);
                a.addLessAmount(l); //adds to admin's object that info.
            }

    }
}
```

isAmountEnough was Θ(n) and findProductIndex was Θ(n). Also addLessAmount() was Θ(1) in best case and Θ(n) in worst. So this method will cost in best case Θ(n) just because of checking if statement, and in worst it will enter the statement and for loop so it will be $\Theta(n^2)+\Theta(n)+\Theta(n)$ which is $\Theta(n^2)$ because smaller degrees doesn't effect.

```
public void accessPreviousOrderInfo(Customers customer){
    customer.ViewPreviousOrders(customer.getCustomerNumber());
}
```

```
public void ViewPreviousOrders(int customerNumber){
    for(int i=0; i<previousOrders.size(); i++){
        System.out.println( previousOrders.get(i) );
    }
}
```

Because of for loop it will cost Θ(n). Θ(1) doesn't effect.

```
public void sale(Branch branchIn, Products.Product type, Colors color,int model, Customers customer){
    int index = branchIn.findProductIndex(type, color, model);
    branchIn.getProducts().get(index).decreaseStock();
    customer.getPreviousOrders().add("Order-> Branch: " + branchIn.getBranchName() + " " + branchIn.getProducts().get(index).toString());
}
```

HybridList's add method already above which was best case is Θ(1) and average/worst case Θ(n). findProductsIndex was Θ(n) and the others are constant time so Θ(n). So add method's best or worst doesn't effect so Θ(n).

```
public void createSubscriptionForCustomer(int customerNumber, Customers customer){
    customer.setSubsription(true);
}
```

Θ(1) because of just a setter.

```
public void updatePreviousOrders(Company c, String branchName, int index, int productIndex, Customers cust){
    cust.getPreviousOrders().add("S|Order-> Branch: " + branchName + " " + c.getAllBranches().get(index).getProducts().get(productIndex).toString() );
    System.out.println("S|Order-> Branch: " + branchName + " " + c.getAllBranches().get(index).getProducts().get(productIndex).toString() );
}
```

It was calculated HybridList's add method already above which was best case is Θ(1) and average/worst case Θ(n) so same for this.

For customers,

```
public void setPreviousOrders(String previousOrder, int index) {
    this.previousOrders.remove(index);
    previousOrders.add(previousOrder);
}
```

Remove method was Θ(n) and add method was already above which was best case is Θ(1) and average/worst case Θ(n). So Θ(n+1) = Θ(n+n) which is Θ(n).

```
public void ListProducts(Branch b){
    for(int i=0; i<b.getProducts().size(); i++){
        System.out.println( b.getProducts().get(i).toString() );
    }
}
```

toString of Products is Θ(1) so because of for loop it will cost Θ(n).

```
public boolean searchForProduct(Company c, Products.Product productType, Colors color, int model){
    for(int i=0; i<c.getAllBranches().size(); i++)
        if( c.getAllBranches().get(i).findProductIndex(productType, color, model) != -1)
            return true;

    return false;
}
```

findProductIndex was Θ(n) and because of for loop it will cost $\Theta(n^2)$.

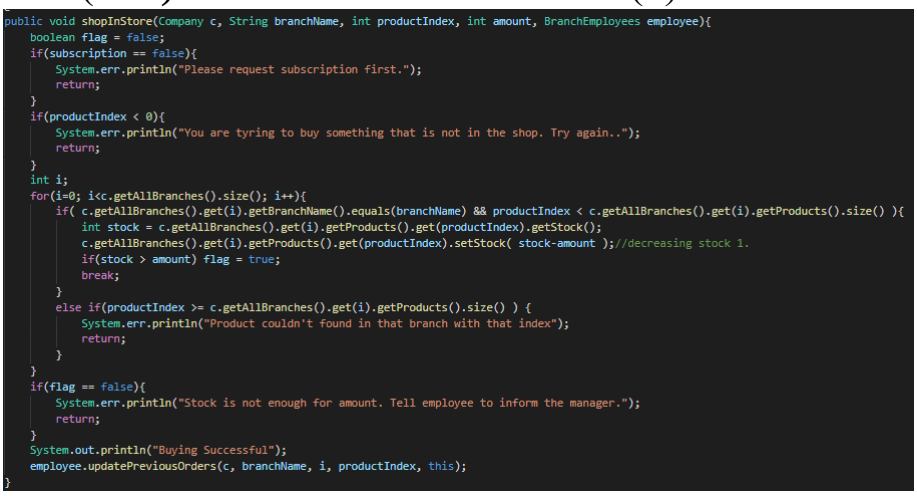
```
public int whichBranchIsIn(Company c, Products.Product productType, Colors color, int model){
    boolean flag = false;
    int index = -1;
    for(int i=0; i<c.getAllBranches().size(); i++)
        if( c.getAllBranches().get(i).findProductIndex(productType, color, model) != -1){
            int stock = c.getAllBranches().get(i).getProducts().get( c.getAllBranches().get(i).findProductIndex(productType, color,model) ).getStock()
            System.out.println("Product found in " + c.getAllBranches().get(i).getBranchName() + " store / stock is " + stock +  " and product index i
            index = c.getAllBranches().get(i).findProductIndex(productType,color,model);
            flag = true;
        }
    if(flag == false){
        index = -1;
        System.out.println("Product you searched couldn't find.");
    }
    return index;
}
```

In best case it doesn't enter the inside of if, so because of for loop it will cost Θ(n) in best case. If enters also there two Θ(n) findProductIndex but because of we sum each other Θ(n) + Θ(n) + Θ(n) will cost Θ(n) which same in all cases.

```
public void shopOnline(String address, int phoneNumber, Company c, String branchName, int productIndex, int amount){
    boolean flag = false;
    int i, stock = 0;
    if(subscription == false){
        System.err.println("Please request subscription first.");
    }
    for(i=0; i<c.getAllBranches().size(); i++){
        if( c.getAllBranches().get(i).getBranchName().equals(branchName) ){
            stock = c.getAllBranches().get(i).getProducts().get(productIndex).getStock();
            if(stock > amount){
                c.getAllBranches().get(i).getProducts().get(productIndex).setStock(stock-amount);//decreasing stock.
                flag = true;
                break;
            }
        }
    }
    if(flag == false){
        System.err.println("Stock is not enough for amount. Tell employee to inform the manager.");
        return;
    }

    System.out.println("Buying Successful");
    previousOrders.add("N|Order-> Branch: " + branchName + " " + c.getAllBranches().get(i).getProducts().get(productIndex).toString() + " Address: "+
    System.out.println("N|Order-> Branch: " + branchName + " " + c.getAllBranches().get(i).getProducts().get(productIndex).toString() + " Address: "+
}
```

Add method belongs to HybridList and HybridList's add method already above which was best case is Θ(1) and average/worst case Θ(n). Also for loop costs Θ(n). So Θ(n+1) or Θ(n+n) doesn't matter will cost Θ(n) .



```
public void shopInStore(Company c, String branchName, int productIndex, int amount, BranchEmployees employee){
    boolean flag = false;
    if(subscription == false){
        System.err.println("Please request subscription first.");
        return;
    }
    if(productIndex < 0){
        System.err.println("You are tyring to buy something that is not in the shop. Try again..");
        return;
    }
    int i;
    for(i=0; i<c.getAllBranches().size(); i++){
        if( c.getAllBranches().get(i).getBranchName().equals(branchName) && productIndex < c.getAllBranches().get(i).getProducts().size() ){
            int stock = c.getAllBranches().get(i).getProducts().get(productIndex).getStock();
            c.getAllBranches().get(i).getProducts().get(productIndex).setStock( stock-amount );//decreasing stock 1.
            if(stock > amount) flag = true;
            break;
        }
        else if(productIndex >= c.getAllBranches().get(i).getProducts().size() ) {
            System.err.println("Product couldn't found in that branch with that index");
            return;
        }
    }
    if(flag == false){
        System.err.println("Stock is not enough for amount. Tell employee to inform the manager.");
        return;
    }
    System.out.println("Buying Successful");
    employee.updatePreviousOrders(c, branchName, i, productIndex, this);
}
```

In best case shopInStore doesn't enter in loop so cost is Θ(1). For average/worst case it enters in for loop and because of inside of it full setters and getters(Θ(1)), it will cost Θ(n) because of for loop.

```
public void ViewPreviousOrders(int customerNumber){
    for(int i=0; i<previousOrders.size(); i++){
        System.out.println( previousOrders.get(i) );
    }
}
```

Because of for loop it will cost Θ(n).

```
public void requestSubscription(BranchEmployees bE){
    bE.createSubscriptionForCustomer(customerNumber, this);
}
```

```
public void createSubscriptionForCustomer(int customerNumber, Customers customer){
    customer.setSubsription(true);
}
```

Just calls a constant time setter so Θ(1). Other classes/enums methods are full of setters/getters and Θ(1) so didn't add here. Also lastly for KWLinkedList, KWArrayList and HybridList toString method's complexity is below.

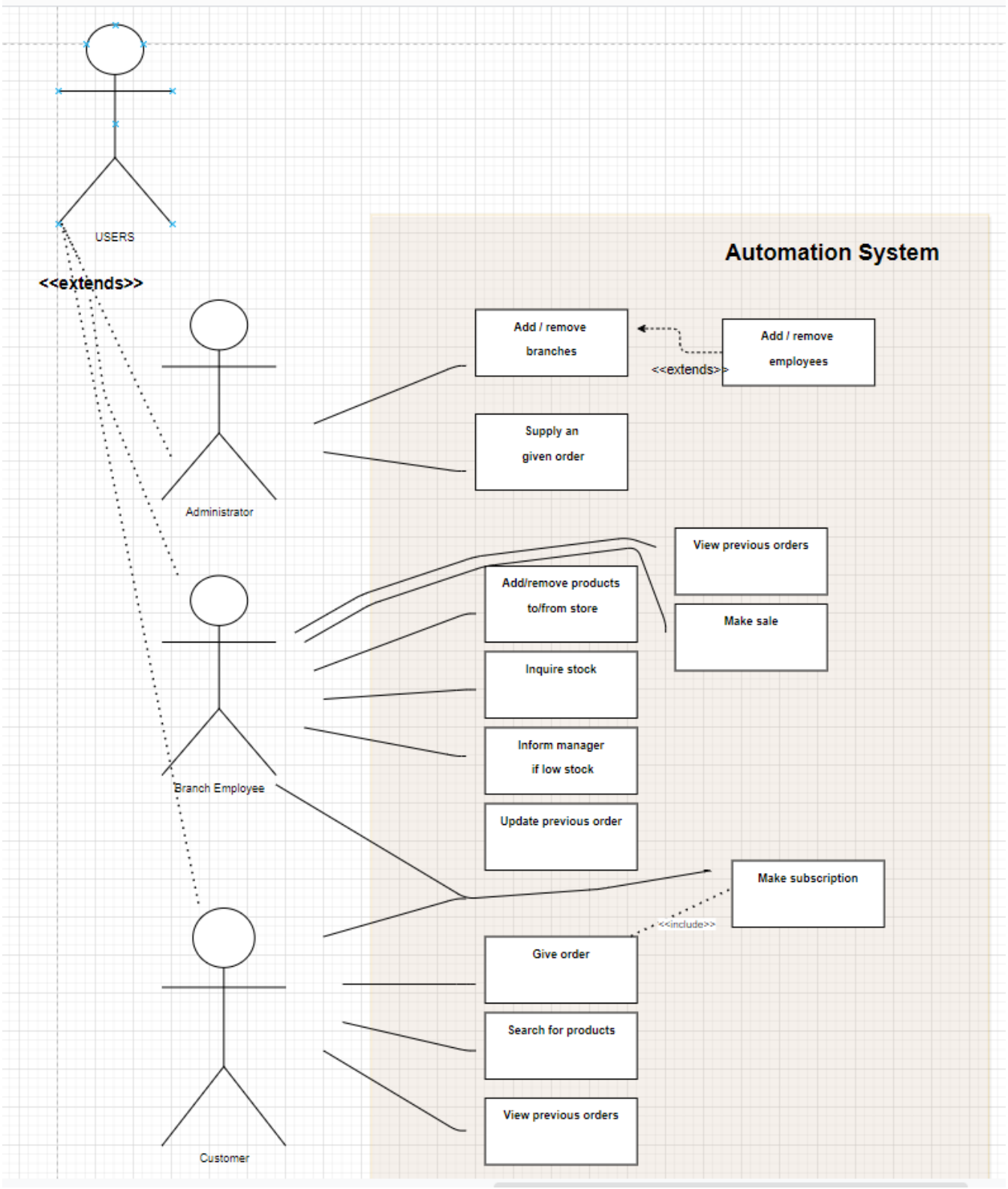
```
public String toString(){
    Node<E> temp = head;
    StringBuilder sb= new StringBuilder("[ ");
    for(int i=0; i<size(); i++){
        if(i != size()-1) sb.append( temp.data + ", ");
        else              sb.append( temp.data + " " );
        temp = temp.next;
    }
    sb.append("]");
    return sb.toString();
}
```

If I used normal String instead of StringBuilder it would cost $\Theta(n^2)$ because strings are immutable so every time new string would be created and references would going to be changed. But because of StringBuilder we don’t need that.Because of for loop it will cost $\Theta(n)$.

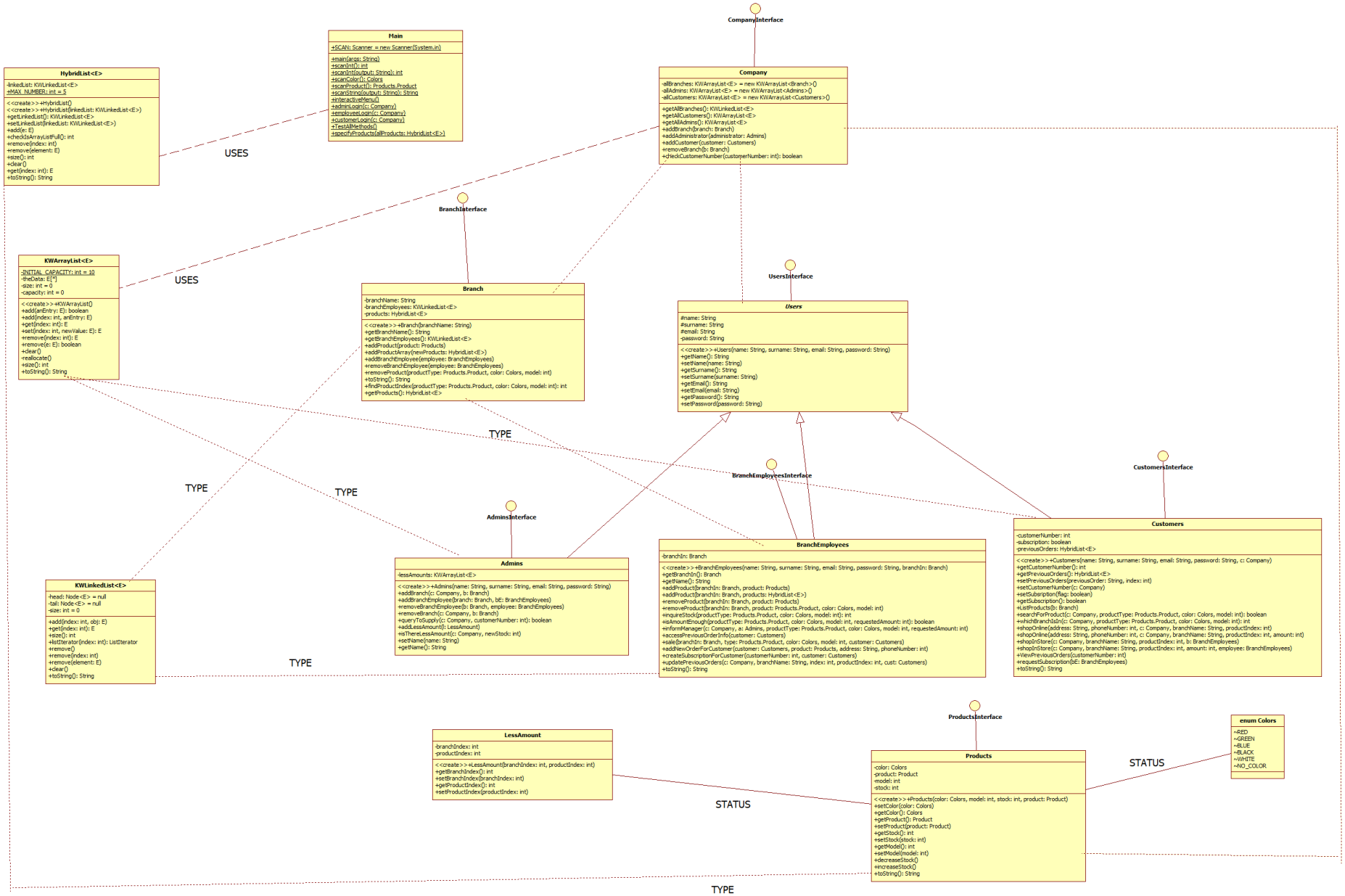
2. SYSTEM REQUIREMENTS

Because of the application is going to be used by administrators , employees work in branch and customers, it should give different options for user.
Administrators can add/remove employees of branches or branches to the company.
Branch employees can add/remove products to store. They can inquire stocks and if it is less they can inform the manager(admin). They can sell and add new order for customers.
If customer doesn’t have a subscription yet, they can add a subscription for customer.
Every customer has a special customer number.
Customers can search for products but firstly they have to choose color and product model or they can choose branch and then they can list all the products that store has. Another option is they can give product type and color info(if there has a color option) and they can find which branch has that product.
They can access previous orders by their special customer number.
Branch employees can not change their branch. They cannot add administrators or branches. They cannot see customer’s previous orders without customer info.

3. USE CASE DIAGRAMS



4. CLASS DIAGRAMS



5. PROBLEM SOLUTION APPROACH

My solution approach was changing my Dynamic array class with just using simple arrays in Homework 1. If I hadn't use them, I would have to create methods like addToArray(), removeFromArray() etc. for every class and it would be really repeating and bad. But in Homework 1, I used MutableArray and my solution in Homework 3 was changing MutableArray class with KWLinkedList, KWArrayList and HybridList.

My another approach was using abstract class for Users. I created variables like name, surname, email, id inside of it. So when I created other user classes like Admins, BranchEmployees and Customers, I just extended it from Users class.

Other problem was product's color and type. For these I created enums. So for example when I need to use blue color, I just used Colors.BLUE. And for the products that don't have color info, I used Colors.NO_COLOR.

My last problem and approach was controlling customers and products from Admins class at the same time. To solve this I created Company class. This class helped me a lot because I created branches admins and customers in this class. So connecting them became easy.

6. TEST CASES

6.1) CREATING ADMIN AND BRANCHES

```
Company company = new Company();
Branch branch1, branch2, branch3, branch4, tempBranch;
BranchEmployees be1, be2, be3, be4, tempEmployee;
Admins admin = new Admins("Suleyman", "Golbol", "info@sglbl.com", "1234");
System.out.println(admin.getName());
System.out.println("----TEST ALL----");
System.out.println("ADDING 4 BRANCHES ");
admin.addBranch(company, branch1 = new Branch("Hatay Branch")   );
admin.addBranch(company, branch2 = new Branch("Adana Branch")   );
admin.addBranch(company, branch3 = new Branch("Mersin Branch")  );
admin.addBranch(company, branch4 = new Branch("Kocaeli Branch") );
admin.addBranch(company, tempBranch = new Branch("Temp Branch") );
System.out.println(company.getAllBranches().toString() );
```

6.2) ADMIN ADDS BRANCH EMPLOYEES

```
System.out.println("--PERSONS(USERS) SUBSCRIBING TO SYSTEM WITH DEFINING THEIR INFO--");
System.out.println("--ADDING BRANCH EMPLOYEES--");
admin.addBranchEmployee(branch1, be1 = new BranchEmployees("Cemil", "Koçak", "cemil@b.com","xyzt" ,  branch1)   );
admin.addBranchEmployee(branch2, be2 = new BranchEmployees("Ahmet", "Açık", "ahmet@b.com","1iki" ,   branch1)   );
admin.addBranchEmployee(branch3, be3 = new BranchEmployees("Furkan", "Boz", "furkan@bc.com","sifre" ,branch1)   );
admin.addBranchEmployee(branch4, be4 = new BranchEmployees("Yahya", "Tutmaz", "yahya@c.com","kocaeli" , branch1));
admin.addBranchEmployee(tempBranch, tempEmployee=new BranchEmployees("Ali", "Can", "alia@c.com","pass" , tempBranch));
```

6.3) REMOVING BRANCH AND BRANCH EMPLOYEES

```
System.out.print(branch1.getBranchEmployees().toString() + branch2.getBranchEmployees().toString());
System.out.println( branch3.getBranchEmployees().toString() + branch4.getBranchEmployees().toString() + tempBranch.getBranchEmployees().toString());
System.out.println("--REMOVING BRANCH EMPLOYEE--");
admin.removeBranchEmployee(tempBranch, tempEmployee);
System.out.print(branch1.getBranchEmployees().toString() + branch2.getBranchEmployees().toString());
System.out.println( branch3.getBranchEmployees().toString() + branch4.getBranchEmployees().toString() + tempBranch.getBranchEmployees().toString());
System.out.println("--REMOVING BRANCH--");
admin.removeBranch(company, tempBranch);
System.out.println(company.getAllBranches().toString() );
```

TERMINAL OUTPUT

```
sglbl@SglblPC:/mnt/c/Araclar/GTU/2.Sınıf/2.Dönem/Cse222/HW1$ make
javac -d classfiles *.java
java -cp classfiles Main
Admin: Suleyman Golbol
----TEST ALL----
ADDING 4 BRANCHES
[ Hatay Branch, Adana Branch, Mersin Branch, Kocaeli Branch, Temp Branch ]
--PERSONS(USERS) SUBSCRIBING TO SYSTEM WITH DEFINING THEIR INFO--
--ADDING BRANCH EMPLOYEES--
[ Cemil Koçak ][ Ahmet Açık ][ Furkan Boz ][ Yahya Tutmaz ][ Ali Can ]
--REMOVING BRANCH EMPLOYEE--
[ Cemil Koçak ][ Ahmet Açık ][ Furkan Boz ][ Yahya Tutmaz ][ ]
--REMOVING BRANCH--
[ Hatay Branch, Adana Branch, Mersin Branch, Kocaeli Branch ]
```

6.4) CREATING CUSTOMERS WITH INFO

```
System.out.println("--CREATING CUSTOMERS WITH NAME EMAIL AND PASSWORD--");
Customers customer1, customer2, customer3, customer4;
company.addCustomer( customer1 = new Customers("Ali", "Cem", "alicem@gmail.com", "1iki3", company) );
company.addCustomer( customer2 = new Customers("Oya", "Su", "oya@gmail.com", "4321", company) );
company.addCustomer( customer3 = new Customers("Talat","Ay", "talat@gmail.com", "1233", company) );
company.addCustomer( customer4 = new Customers("Baran", "Solmaz", "brn@gmail.com", "1111", company) );
```

6.5) CREATING PRODUCTS

```
System.out.println("CREATING PRODUCTS OF BOOK CASES");
for(int i=0; i<12; i++)
    allProducts.add( new Products(Colors.NO_COLOR, i+1, 10, Products.Product.BOOK_CASES) );
System.out.println("CREATING PRODUCTS OF OFFICE CABINETS");
for(int i=0; i<12; i++)
    allProducts.add( new Products(Colors.NO_COLOR, i+1, 10, Products.Product.OFFICE_CABINETS)

System.out.println("CREATING PRODUCTS OF MEETING TABLES");
for(int i=0; i<10; i++)
    allProducts.add( new Products(Colors.BLACK, i+1, 10, Products.Product.MEETING_TABLES) );
for(int i=0; i<10; i++)
    allProducts.add( new Products(Colors.GREEN, i+1, 10, Products.Product.MEETING_TABLES) );
for(int i=0; i<10; i++)
    allProducts.add( new Products(Colors.BLUE, i+1, 10, Products.Product.MEETING_TABLES) );
for(int i=0; i<10; i++)
    allProducts.add( new Products(Colors.RED, i+1, 10, Products.Product.MEETING_TABLES) );

System.out.println("CREATING PRODUCTS OF OFFICE DESKS");
for(int i=0; i<5; i++)
    allProducts.add( new Products(Colors.BLACK, i+1, 10, Products.Product.OFFICE_DESKS) );
```

6.6) BRANCH EMPLOYEES ADD PRODUCTS TO BRANCH

```
System.out.println("BRANCH EMPLOYEES ADDS PRODUCTS");
be1.addProduct(branch1, allProducts); //branch employee ads products to her/his branch.
be2.addProduct(branch2, allProducts);
be3.addProduct(branch3, allProducts);
be4.addProduct(branch4, new Products(Colors.BLUE, /*model*/1, /*stock*/2, Products.Product.MEETING_TABLES ));
//Branch4 has just one product in it.
```

6.7) CUSTOMER LISTS ALL PRODUCTS
6.8) CUSTOMER SEARCH FOR PRODUCTS
6.9) TRYING TO FIND PRODUCT DOESN'T EXIST
6.10) TRYING TO FIND PRODUCT EXISTS
6.11) CUSTOMER SHOPPING FROM STORE
6.12) CUSTOMER LEARNS WHICH BRANCH HAS PRODUCT
6.13) TRY BUYING PRODUCT WITHOUT SUBSCRIPTION
6.14) REQUESTING SUBSCRIPTION FOR CUSTOMER FROM EMPLOYEE
6.15) BUYING PRODUCT WITH SUBSCRIPTION
6.16) ADMIN QUERIES IF NEED TO BE SUPPLIED
6.17) IF NEEDED, SUPPLIES
6.18) TRYING TO BUY PRODUCT COULDN'T FIND IN A BRANCH
6.19) EMPLOYEE TRY TO REMOVE A PRODUCT FROM BRANCH DOESN'T EXIST
6.20) CUSTOMER SHOPPING ONLINE
6.21) CUSTOMER TRIES TO BUY WITH A BIG AMOUNT WHICH IS NOT ENOUGH
6.22) BRANCH EMPLOYEE INQUIRES STOCK
6.23) EMPLOYEE INFORMS ADMIN, ADMIN INCREASING STOCK
6.24) BRANCH EMPLOYEE SELLS PRODUCT TO CUSTOMER
6.25) BRANCH EMPLOYEE ACCESS CUSTOMER'S PREVIOUS ORDER

7. RUNNING AND RESULTS

These are just the driver for test.

```
sglbl@SglblPC:/mnt/c/Araclar/GTU/2.Sınıf/2.Dönem/Cse222/HW1$ make
javac -d classfiles *.java
java -cp classfiles Main
Admin: Suleyman Golbol
----TEST ALL----
ADDING 4 BRANCHES
[ Hatay Branch, Adana Branch, Mersin Branch, Kocaeli Branch, Temp Branch ]
--PERSONS(USERS) SUBSCRIBING TO SYSTEM WITH DEFINING THEIR INFO--
--ADDING BRANCH EMPLOYEES--
[ Cemil Koçak ][ Ahmet Açık ][ Furkan Boz ][ Yahya Tutmaz ][ Ali Can ]
--REMOVING BRANCH EMPLOYEE--
[ Cemil Koçak ][ Ahmet Açık ][ Furkan Boz ][ Yahya Tutmaz ][ ]
--REMOVING BRANCH--
[ Hatay Branch, Adana Branch, Mersin Branch, Kocaeli Branch ]
--CREATING CUSTOMERS WITH NAME EMAIL AND PASSWORD--
Ali Cem's customer number is 54
Oya Su's customer number is 2
Talat Ay's customer number is 78
Baran Solmaz's customer number is 95
--PRINTING ALL CUSTOMERS--
[ Name: Ali Cem Email: alicem@gmail.com, Name: Oya Su Email: oya@gmail.com, Name: Talat Ay Email: talat@gmail.com, Name: Baran Solmaz Email: brn@gmail.com ]
--CREATING PRODUCTS--
CREATING PRODUCTS OF BOOK CASES
CREATING PRODUCTS OF OFFICE CABINETS
CREATING PRODUCTS OF MEETING TABLES
CREATING PRODUCTS OF OFFICE DESKS
CREATING PRODUCTS OF OFFICE CHAIRS
```

```
BRANCH EMPLOYEES ADDS PRODUCTS
--CUSTOMER LISTS ALL PRODUCTS--
Model: 1 Color: NO_COLOR Type: BOOK_CASES
Model: 2 Color: NO_COLOR Type: BOOK_CASES
Model: 3 Color: NO_COLOR Type: BOOK_CASES
Model: 4 Color: NO_COLOR Type: BOOK_CASES
Model: 5 Color: NO_COLOR Type: BOOK_CASES
Model: 6 Color: NO_COLOR Type: BOOK_CASES
Model: 7 Color: NO_COLOR Type: BOOK_CASES
Model: 8 Color: NO_COLOR Type: BOOK_CASES
Model: 9 Color: NO_COLOR Type: BOOK_CASES
Model: 10 Color: NO_COLOR Type: BOOK_CASES
Model: 11 Color: NO_COLOR Type: BOOK_CASES
Model: 12 Color: NO_COLOR Type: BOOK_CASES
Model: 1 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 2 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 3 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 4 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 5 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 6 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 7 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 8 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 9 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 10 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 11 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 12 Color: NO_COLOR Type: OFFICE_CABINETS
Model: 1 Color: BLACK Type: MEETING_TABLES
Model: 2 Color: BLACK Type: MEETING_TABLES
Model: 3 Color: BLACK Type: MEETING_TABLES
```

```
Model: 6 Color: BLUE Type: OFFICE_CHAIRS
Model: 7 Color: BLUE Type: OFFICE_CHAIRS
--CUSTOMER SEARCH FOR PRODUCTS--
--TRYING TO FIND BLACK OFFICE CABINETS(WHICH SHOULD BE FALSE BECAUSE THERE ISN'T COLOR CHOICE)--
false
--TRYING TO FIND BLUE OFFICE DESKS--
true
---CUSTOMER SHOPPING FROM STORE---
--CUSTOMER TRIES TO FIND A PRODUCT BRANCH DOESN'T HAVE( BLACK BOOK CASES)--
Product you searched couldn't find.

--CUSTOMER LEARNS WHICH BRANCH HAS THAT PRODUCT( BLUE OFFICE DESK)--
Product found in Hatay Branch store / stock is 10 and product index is 79
Product found in Adana Branch store / stock is 10 and product index is 79
Product found in Mersin Branch store / stock is 10 and product index is 79

--BUYING PRODUCT WITH INDEX NUMBER IN-STORE (BLUE OFFICE DESK) WITHOUT SUBSCRIPTION--
Please request subscription first.

--REQUESTING SUBSCRIPTION FOR CUSTOMER FROM EMPLOYEE--
--BUYING PRODUCT WITH INDEX NUMBER IN-STORE (BLUE OFFICE DESK) WITH SUBSCRIPTION--
Product found in Hatay Branch store / stock is 10 and product index is 79
Product found in Adana Branch store / stock is 10 and product index is 79
Product found in Mersin Branch store / stock is 10 and product index is 79
Buying Successful
BRANCH EMPLOYEE IS UPDATING CUSTOMER'S PREVIOUS ORDER
S|Order-> Branch: Adana Branch Model: 1 Color: BLUE Type: OFFICE_DESKS
--ADMIN QUERIES IF NEED TO BE SUPPLIED (IF NEEDED, SUPPLIES AND PUTS "S" INSTEAD "N")--
S|Order-> Branch: Adana Branch Model: 1 Color: BLUE Type: OFFICE_DESKS
```

```
--TRYING TO BUY PRODUCT COULDN'T FIND IN A BRANCH--
Please request subscription first.

PRINTING BRANCH4 PRODUCTS -> [ Model: 1 Color: BLUE Type: MEETING_TABLES ]
--EMPLOYEE TRIES TO REMOVE A PRODUCT FROM A BRANCH THAT DOESN'T EXIST--
Product you want to delete couldn't find.

--EMPLOYEE REMOVES A PRODUCT FROM A BRANCH THAT EXISTS--
PRINTING BRANCH4 PRODUCTS -> [ ]
```

```
--CUSTOMER SHOPPING ONLINE--
--SEARCH FOR A SPECIFIC PRODUCT(IF FOUND RETURN TRUE)--
true
--CUSTOMER LEARNS WHICH BRANCH HAS THAT PRODUCT( OFFICE CABINET WHICH DOESN'T HAVE COLOR CHOICE)--
Product found in Hatay Branch store / stock is 10 and product index is 14
Product found in Adana Branch store / stock is 10 and product index is 14
Product found in Mersin Branch store / stock is 10 and product index is 14
Product found in Hatay Branch store / stock is 10 and product index is 13
Product found in Adana Branch store / stock is 10 and product index is 13
Product found in Mersin Branch store / stock is 10 and product index is 13
```

```
--CUSTOMER TRIES TO BUY WITH A BIG AMOUNT WHICH IS NOT ENOUGH--
Stock is not enough for amount. Tell employee to inform the manager.
--BRANCH EMPLOYEE INQUIRES STOCK AND CHECKS IF AMOUNT IS ENOUGH (RETURNS TRUE IF ENOUGH)--
false
--ADMIN INFORMED, ADMIN INCREASING STOCK--
--CUSTOMER TRIES TO BUY AGAIN--
Buying Successful
N|Order-> Branch: Mersin Branch Model: 2 Color: NO_COLOR Type: OFFICE_CABINETS Address: Turunclu, Hatay 31160 Phone: 2233870
--BRANCH EMPLOYEE SELLS PRODUCT TO CUSTOMER--
--BRANCH EMPLOYEE ACCESS CUSTOMER'S PREVIOUS ORDER--
Order-> Branch: Mersin Branch Model: 2 Color: NO_COLOR Type: OFFICE_CABINETS
S|Order-> Branch: Mersin Branch Model: 2 Color: NO_COLOR Type: OFFICE_CABINETS Address: Turunclu, Hatay 31160 Phone: 2233870

sglbl@SglblPC:/mnt/c/Araclar/GTU/2.Sınıf/2.Dönem/Cse222/HW1$
```