GIT Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 3 Report

Emre SEZER
1901042640

SYSTEM REQUIREMENTS:

I created this homework on Windows 10 using terminal (Java Development Kit).My java version is 11.0.8 .You need to compile “Company.java” in order to test my homework.There are java files, a folder named “Javadoc” that includes javadoc files and report file in my homework.

PROBLEM SOLUTION APPROACH:

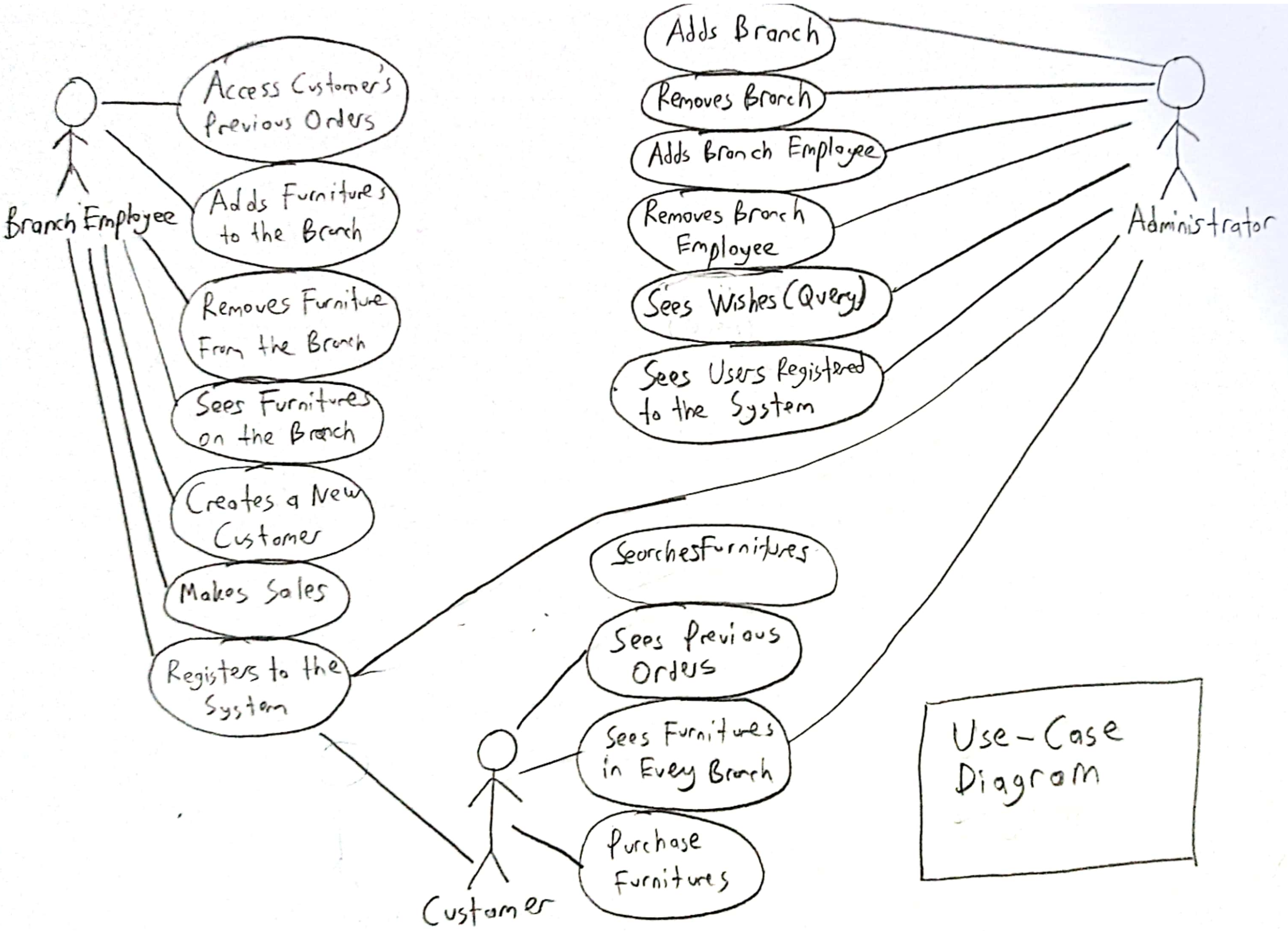
I used object oriented techniques while doing this homework as asked. First, i created User interface and Furniture abstract class to ensure Polymorphism. Then, i wrote Administrator, BranchEmployee and Customer classes implements User.Each of these classes has their own methods and methods from User.I wrote classes inherits Furniture (OfficeChair, OfficeDesk etc.). Finally, i wrote Company class.In that class i wrote a menu function to provide interactivity. I wrote driver method for testing cases. In this menu you can test my homework. I implemented KWArrayList, KWLinkedList and KWHybridList. These classes are for storing data.While writing KWHybridList, I used KWArrayList and KWLinkedList’s methods.

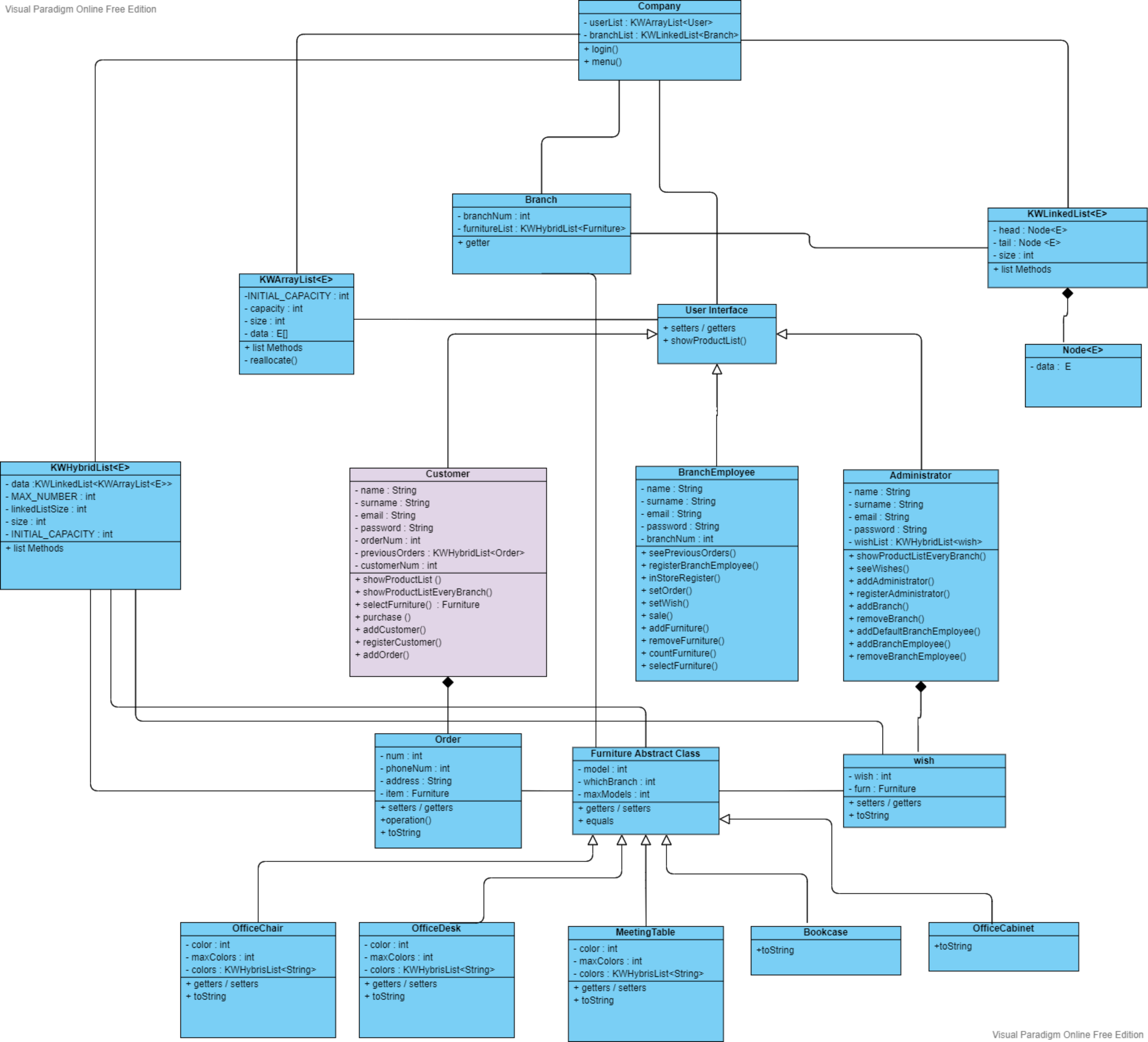
For menu :

There is manual registration for Administrator and BranchEmployee.Branch You can login as Admininstrator using “admin” as email and “1234” as password.You can also login as BranchEmployee using the “employeex” as e-mail and “1234” as password.At “employeex” x isBranch’s number. For example, if you want to login as index 3 Branch’s default BranchEmployee, You need to type “employee3” as e-mail.You can add as many BranchEmployee’s to any Branch with menu.

Test Cases : I created a driver method for testing my methods.At this method, i tested many cases.Like , succesfully working cases and throwing exception cases. Also, i created a menu method.This method is interactable.

In order to run driver method, you need to compile “driver.java”.In order to run menu method, you need to compile “menu.java”.

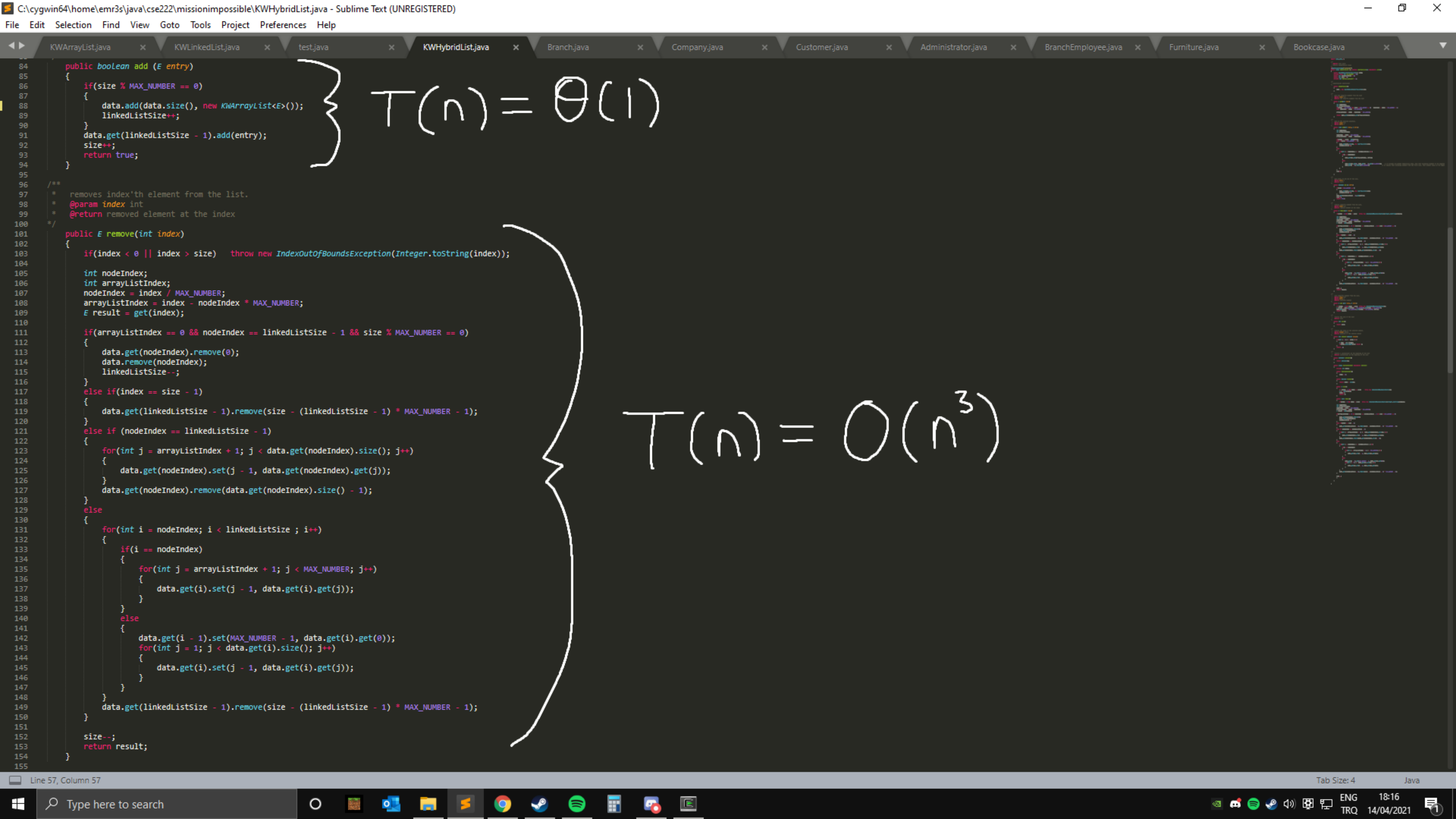

Branch Employee
Access Customer's Previous Orders
Adds Furnitures to the Branch
Removes Furniture From the Branch
Sees Furnitures on the Branch
Creates a New Customer
Makes Sales
Registers to the System
Adds Branch
Removes Branch
Adds Branch Employee
Removes Branch Employee
Sees Wishes (Query)
Sees Users Registered to the System
Administrator
SearchesFurnitures
Sees Previous Orders
Sees Furnitures in Evey Branch
Purchase Furnitures
Customer
Use-Case Diagram


Company
- userList : KWArrayList<User>
- branchList : KWLinkedList<Branch>
+ login()
+ menu()
Branch
- branchNum : int
- furnitureList : KWHybridList<Furniture>
+ getter
KWLinkedList<E>
- head : Node<E>
- tail : Node <E>
- size : int
+ list Methods
Node<E>
- data : E
KWArrayList<E>
-INITIAL_CAPACITY : int
- capacity : int
- size : int
- data : E[]
+ list Methods
- reallocate()
User Interface
+ setters / getters
+ showProductList()
KWHybridList<E>
- data :KWLinkedList<KWArrayList<E>>
- MAX_NUMBER : int
- linkedListSize : int
- size : int
- INITIAL_CAPACITY : int
+ list Methods
Customer
- name : String
- surname : String
- email : String
- password : String
- orderNum : int
- previousOrders : KWHybridList<Order>
- customerNum : int
+ showProductList ()
+ showProductListEveryBranch()
+ selectFurniture() : Furniture
+ purchase ()
+ addCustomer()
+ registerCustomer()
+ addOrder()
BranchEmployee
- name : String
- surname : String
- email : String
- password : String
- branchNum : int
+ seePreviousOrders()
+ registerBranchEmployee()
+ inStoreRegister()
+ setOrder()
+ setWish()
+ sale()
+ addFurniture()
+ removeFurniture()
+ countFurniture()
+ selectFurniture()
Administrator
- name : String
- surname : String
- email : String
- password : String
- wishList : KWHybridList<wish>
+ showProductListEveryBranch()
+ seeWishes()
+ addAdministrator()
+ registerAdministrator()
+ addBranch()
+ removeBranch()
+ addDefaultBranchEmployee()
+ addBranchEmployee()
+ removeBranchEmployee()
Order
- num : int
- phoneNum : int
- address : String
- item : Furniture
+ setters / getters
+operation()
+ toString
Furniture Abstract Class
- model : int
- whichBranch : int
- maxModels : int
+ getters / setters
+ equals
wish
- wish : int
- furn : Furniture
+ setters / getters
+ toString
OfficeChair
- color : int
- maxColors : int
- colors : KWHybrisList<String>
+ getters / setters
+ toString
OfficeDesk
- color : int
- maxColors : int
- colors : KWHybrisList<String>
+ getters / setters
+ toString
MeetingTable
- color : int
- maxColors : int
- colors : KWHybrisList<String>
+ getters / setters
+ toString
Bookcase
+toString
OfficeCabinet
+toString

PART 2: I didn’t calculate Time Complexity of getter / setter methods and Constructors.For my homework these methods take constant time to run.

So, didn’t take screenshots of these methods.For this part I took screenshots of the methods and wrote equivalent of T(n) ‘s next to them.

KWHYBRIDLIST

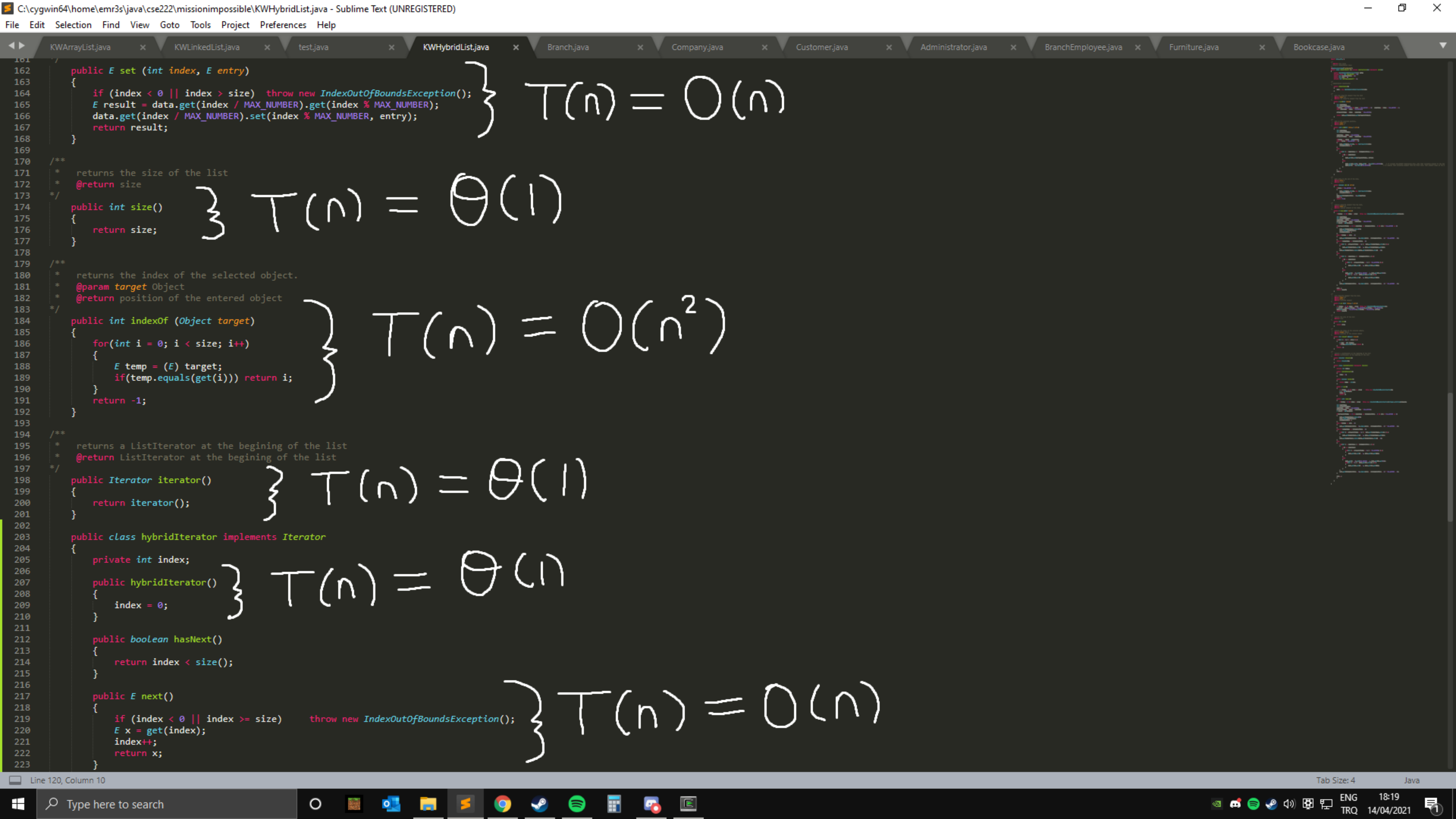


```java
    public boolean add (E entry)
    {
        if(size % MAX_NUMBER == 0)
        {
            data.add(data.size(), new KWArrayList<E>());
            linkedListSize++;
        }
        data.get(linkedListSize - 1).add(entry);
        size++;
        return true;
    }
```

$T(n) = \Theta(1)$

```java
    /**
     *  removes index'th element from the list.
     *  @param index int
     *  @return removed element at the index
     */
    public E remove(int index)
    {
        if(index < 0 || index > size)   throw new IndexOutOfBoundsException(Integer.toString(index));

        int nodeIndex;
        int arrayListIndex;
        nodeIndex = index / MAX_NUMBER;
        arrayListIndex = index - nodeIndex * MAX_NUMBER;
        E result = get(index);

        if(arrayListIndex == 0 && nodeIndex == linkedListSize - 1 && size % MAX_NUMBER == 0)
        {
            data.get(nodeIndex).remove(0);
            data.remove(nodeIndex);
            linkedListSize--;
        }
        else if(index == size - 1)
        {
            data.get(linkedListSize - 1).remove(size - (linkedListSize - 1) * MAX_NUMBER - 1);
        }
        else if (nodeIndex == linkedListSize - 1)
        {
            for(int j = arrayListIndex + 1; j < data.get(nodeIndex).size(); j++)
            {
                data.get(nodeIndex).set(j - 1, data.get(nodeIndex).get(j));
            }
            data.get(nodeIndex).remove(data.get(nodeIndex).size() - 1);
        }
        else
        {
            for(int i = nodeIndex; i < linkedListSize ; i++)
            {
                if(i == nodeIndex)
                {
                    for(int j = arrayListIndex + 1; j < MAX_NUMBER; j++)
                    {
                        data.get(i).set(j - 1, data.get(i).get(j));
                    }
                }
                else
                {
                    data.get(i - 1).set(MAX_NUMBER - 1, data.get(i).get(0));
                    for(int j = 1; j < data.get(i).size(); j++)
                    {
                        data.get(i).set(j - 1, data.get(i).get(j));
                    }
                }
            }
            data.get(linkedListSize - 1).remove(size - (linkedListSize - 1) * MAX_NUMBER - 1);
        }

        size--;
        return result;
    }
```

$T(n) = O(n^3)$



```
    public E set (int index, E entry)
    {
        if (index < 0 || index > size)  throw new IndexOutOfBoundsException();
        E result = data.get(index / MAX_NUMBER).get(index % MAX_NUMBER);
        data.get(index / MAX_NUMBER).set(index % MAX_NUMBER, entry);
        return result;
    }
```

$T(n) = O(n)$

```
    /**
     *  returns the size of the list
     *  @return size
     */
    public int size()
    {
        return size;
    }
```

$T(n) = \Theta(1)$

```
    /**
     *  returns the index of the selected object.
     *  @param target Object
     *  @return position of the entered object
     */
    public int indexOf (Object target)
    {
        for(int i = 0; i < size; i++)
        {
            E temp = (E) target;
            if(temp.equals(get(i))) return i;
        }
        return -1;
    }
```

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

```
    /**
     *  returns a ListIterator at the begining of the list
     *  @return ListIterator at the begining of the list
     */
    public Iterator iterator()
    {
        return iterator();
    }
```

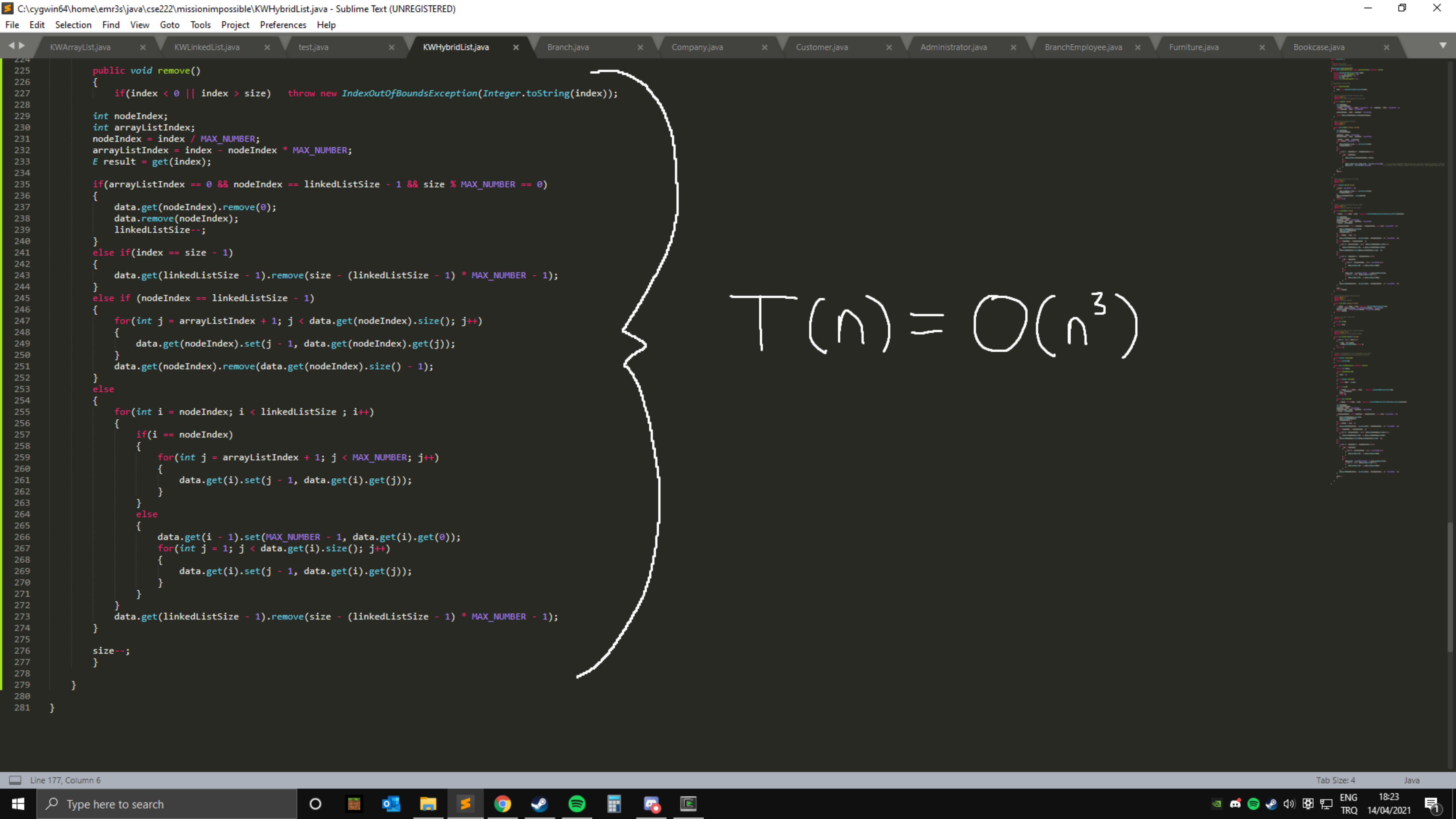
$T(n) = \Theta(1)$

```
    public class hybridIterator implements Iterator
    {
        private int index;

        public hybridIterator()
        {
            index = 0;
        }
```

$T(n) = \Theta(1)$

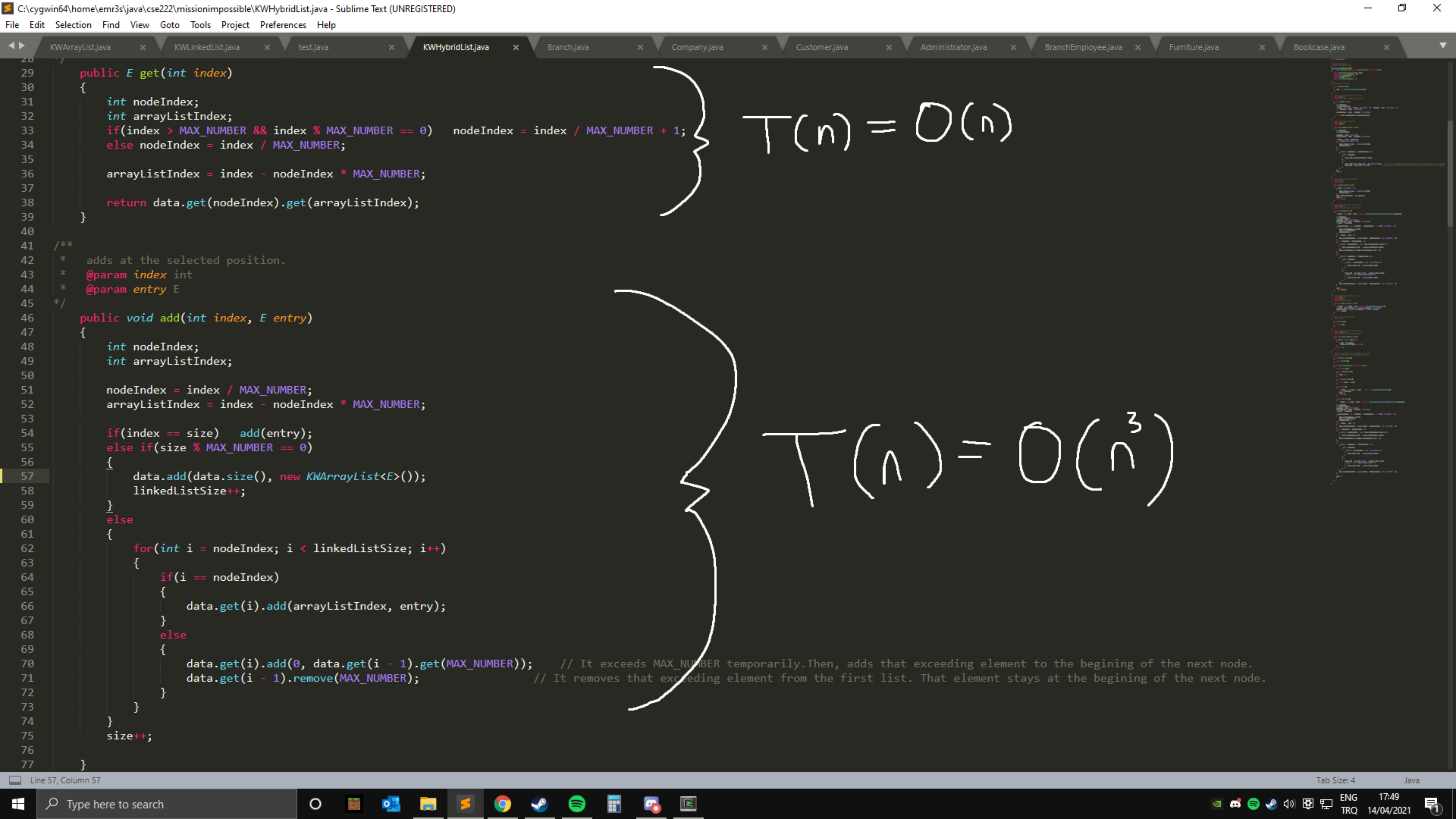
```
        public boolean hasNext()
        {
            return index < size();
        }

        public E next()
        {
            if (index < 0 || index >= size)     throw new IndexOutOfBoundsException();
            E x = get(index);
            index++;
            return x;
        }
```

$T(n) = O(n)$



```java
        public void remove()
        {
            if(index < 0 || index > size)   throw new IndexOutOfBoundsException(Integer.toString(index));

        int nodeIndex;
        int arrayListIndex;
        nodeIndex = index / MAX_NUMBER;
        arrayListIndex = index - nodeIndex * MAX_NUMBER;
        E result = get(index);

        if(arrayListIndex == 0 && nodeIndex == linkedListSize - 1 && size % MAX_NUMBER == 0)
        {
            data.get(nodeIndex).remove(0);
            data.remove(nodeIndex);
            linkedListSize--;
        }
        else if(index == size - 1)
        {
            data.get(linkedListSize - 1).remove(size - (linkedListSize - 1) * MAX_NUMBER - 1);
        }
        else if (nodeIndex == linkedListSize - 1)
        {
            for(int j = arrayListIndex + 1; j < data.get(nodeIndex).size(); j++)
            {
                data.get(nodeIndex).set(j - 1, data.get(nodeIndex).get(j));
            }
            data.get(nodeIndex).remove(data.get(nodeIndex).size() - 1);
        }
        else
        {
            for(int i = nodeIndex; i < linkedListSize ; i++)
            {
                if(i == nodeIndex)
                {
                    for(int j = arrayListIndex + 1; j < MAX_NUMBER; j++)
                    {
                        data.get(i).set(j - 1, data.get(i).get(j));
                    }
                }
                else
                {
                    data.get(i - 1).set(MAX_NUMBER - 1, data.get(i).get(0));
                    for(int j = 1; j < data.get(i).size(); j++)
                    {
                        data.get(i).set(j - 1, data.get(i).get(j));
                    }
                }
            }
            data.get(linkedListSize - 1).remove(size - (linkedListSize - 1) * MAX_NUMBER - 1);
        }

        size--;
        }

    }

}
```

$$T(n) = O(n^3)$$


C:\cygwin64\home\emr3s\java\cse222\missionimpossible\KWHybridList.java - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
KWArrayList.java
KWLinkedList.java
test.java
KWHybridList.java
Branch.java
Company.java
Customer.java
Administrator.java
BranchEmployee.java
Furniture.java
Bookcase.java

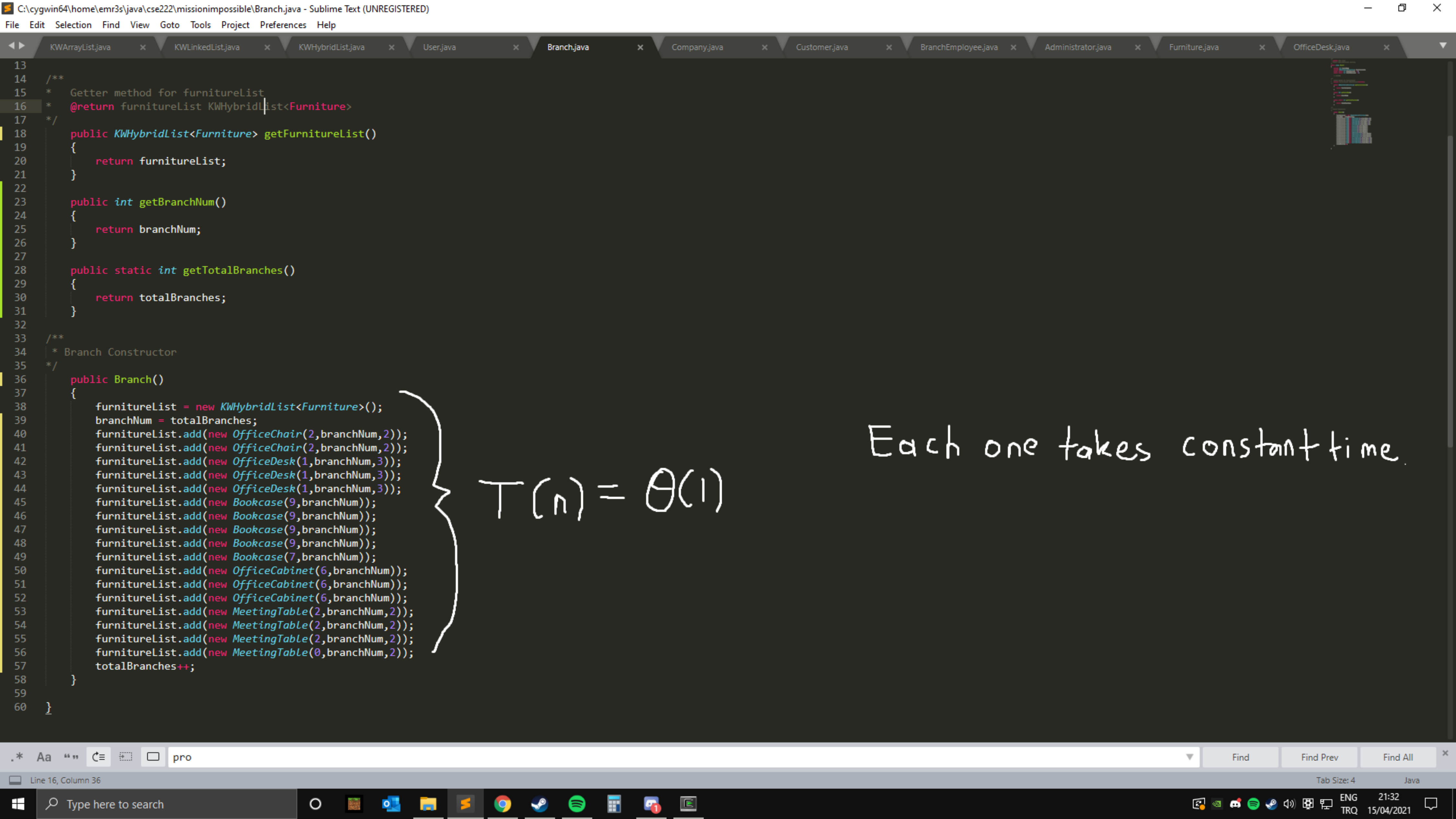
```
    */
    public E get(int index)
    {
        int nodeIndex;
        int arrayListIndex;
        if(index > MAX_NUMBER && index % MAX_NUMBER == 0)   nodeIndex = index / MAX_NUMBER + 1;
        else nodeIndex = index / MAX_NUMBER;

        arrayListIndex = index - nodeIndex * MAX_NUMBER;

        return data.get(nodeIndex).get(arrayListIndex);
    }
```

$T(n) = O(n)$

```
/**
 *   adds at the selected position.
 *   @param index int
 *   @param entry E
*/
    public void add(int index, E entry)
    {
        int nodeIndex;
        int arrayListIndex;

        nodeIndex = index / MAX_NUMBER;
        arrayListIndex = index - nodeIndex * MAX_NUMBER;

        if(index == size)   add(entry);
        else if(size % MAX_NUMBER == 0)
        {
            data.add(data.size(), new KWArrayList<E>());
            linkedListSize++;
        }
        else
        {
            for(int i = nodeIndex; i < linkedListSize; i++)
            {
                if(i == nodeIndex)
                {
                    data.get(i).add(arrayListIndex, entry);
                }
                else
                {
                    data.get(i).add(0, data.get(i - 1).get(MAX_NUMBER));    // It exceeds MAX_NUMBER temporarily.Then, adds that exceeding element to the begining of the next node.
                    data.get(i - 1).remove(MAX_NUMBER);                 // It removes that exceeding element from the first list. That element stays at the begining of the next node.
                }
            }
        }
        size++;

    }
```

$T(n) = O(n^3)$

BRANCH

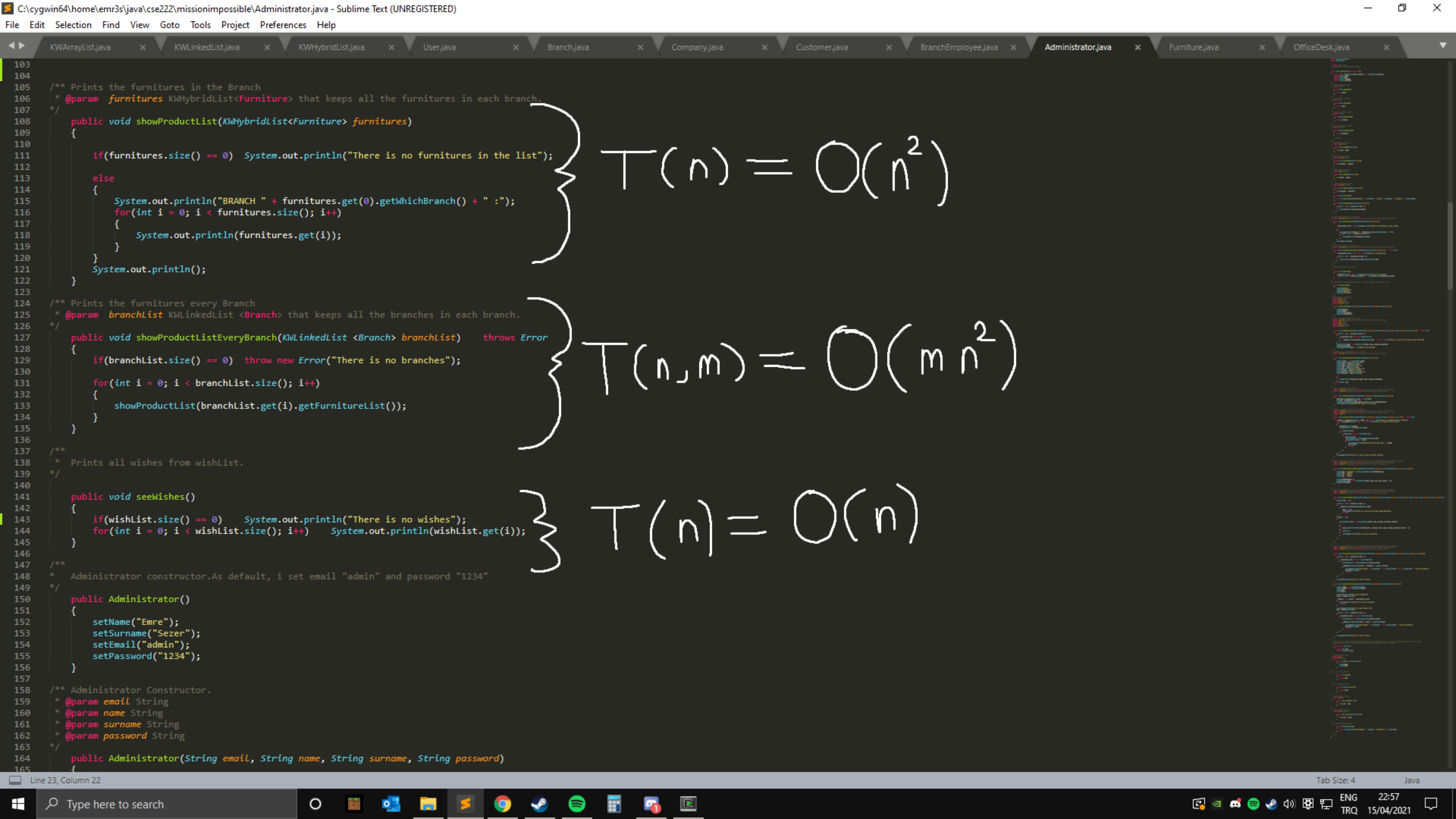


```java

/**
 *   Getter method for furnitureList
 *   @return furnitureList KWHybridList<Furniture>
 */
    public KWHybridList<Furniture> getFurnitureList()
    {
        return furnitureList;
    }

    public int getBranchNum()
    {
        return branchNum;
    }

    public static int getTotalBranches()
    {
        return totalBranches;
    }

/**
  * Branch Constructor
 */
    public Branch()
    {
        furnitureList = new KWHybridList<Furniture>();
        branchNum = totalBranches;
        furnitureList.add(new OfficeChair(2,branchNum,2));
        furnitureList.add(new OfficeChair(2,branchNum,2));
        furnitureList.add(new OfficeDesk(1,branchNum,3));
        furnitureList.add(new OfficeDesk(1,branchNum,3));
        furnitureList.add(new OfficeDesk(1,branchNum,3));
        furnitureList.add(new Bookcase(9,branchNum));
        furnitureList.add(new Bookcase(9,branchNum));
        furnitureList.add(new Bookcase(9,branchNum));
        furnitureList.add(new Bookcase(9,branchNum));
        furnitureList.add(new Bookcase(7,branchNum));
        furnitureList.add(new OfficeCabinet(6,branchNum));
        furnitureList.add(new OfficeCabinet(6,branchNum));
        furnitureList.add(new OfficeCabinet(6,branchNum));
        furnitureList.add(new MeetingTable(2,branchNum,2));
        furnitureList.add(new MeetingTable(2,branchNum,2));
        furnitureList.add(new MeetingTable(2,branchNum,2));
        furnitureList.add(new MeetingTable(0,branchNum,2));
        totalBranches++;
    }

}
```

$T(n) = \Theta(1)$

Each one takes constant time.

ADMINISTRATOR



```java


/** Prints the furnitures in the Branch
 * @param  furnitures KWHybridList<Furniture> that keeps all the furnitures in each branch.
 */
    public void showProductList(KWHybridList<Furniture> furnitures)
    {

        if(furnitures.size() == 0)  System.out.println("There is no furnitures in the list");

        else
        {
            System.out.println("BRANCH " + furnitures.get(0).getWhichBranch() + " :");
            for(int i = 0; i < furnitures.size(); i++)
            {
                System.out.println(furnitures.get(i));
            }
        }
        System.out.println();
    }
```

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

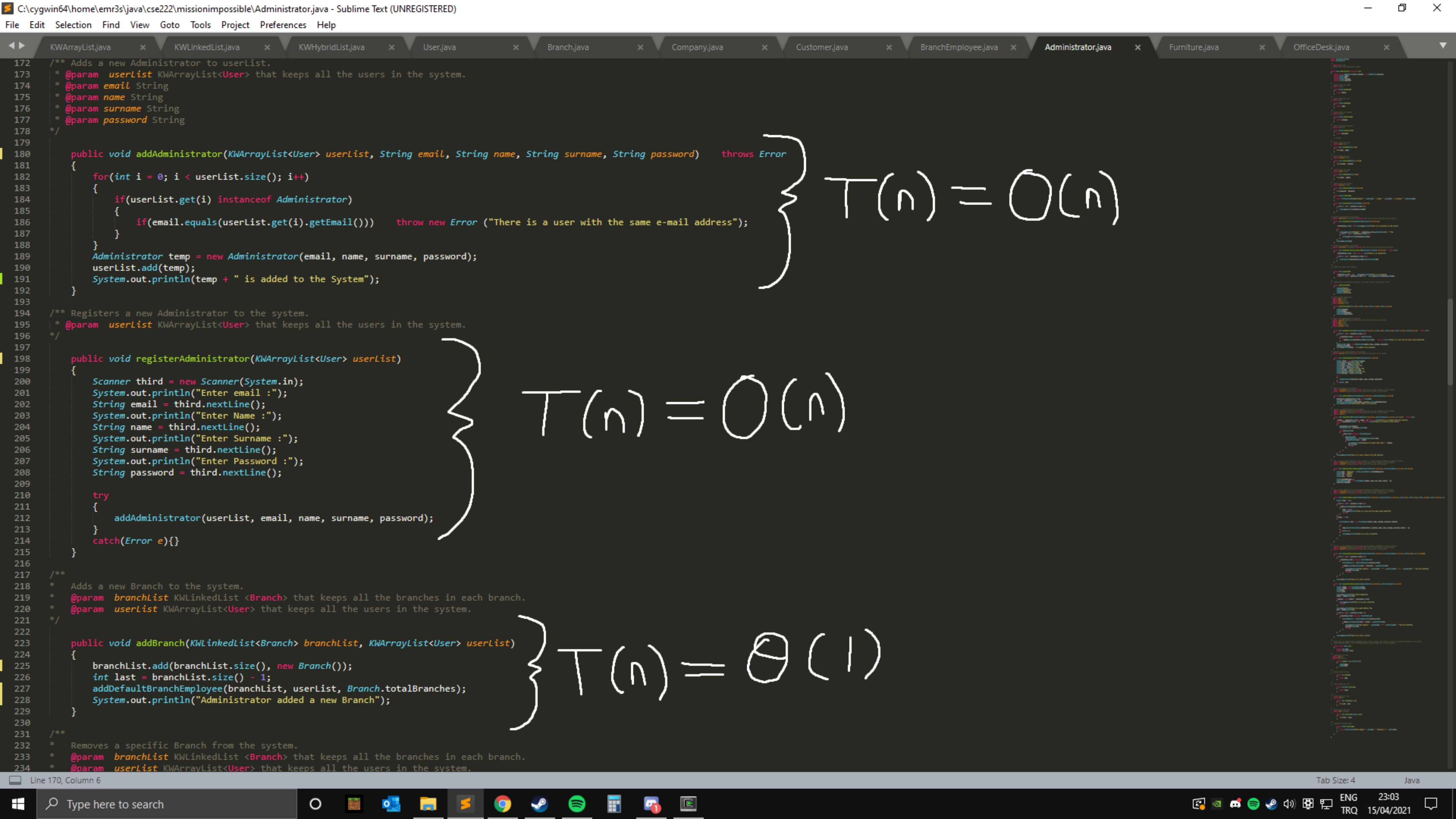
```java

/** Prints the furnitures every Branch
 * @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
 */
    public void showProductListEveryBranch(KWLinkedList <Branch> branchList)    throws Error
    {
        if(branchList.size() == 0)  throw new Error("There is no branches");

        for(int i = 0; i < branchList.size(); i++)
        {
            showProductList(branchList.get(i).getFurnitureList());
        }
    }
```

$T(n,m) = O(mn^2)$

```java

/**
 *  Prints all wishes from wishList.
 */

    public void seeWishes()
    {
        if(wishList.size() == 0)    System.out.println("There is no wishes");
        for(int i = 0; i < wishList.size(); i++)    System.out.println(wishList.get(i));
    }
```

$T(n) = O(n)$

```java

/**
 *  Administrator constructor.As default, i set email "admin" and password "1234"
 */
    public Administrator()
    {
        setName("Emre");
        setSurname("Sezer");
        setEmail("admin");
        setPassword("1234");
    }

/** Administrator Constructor.
 * @param email String
 * @param name String
 * @param surname String
 * @param password String
 */
    public Administrator(String email, String name, String surname, String password)
    {
```



```java
/** Adds a new Administrator to userList.
 * @param  userList KWArrayList<User> that keeps all the users in the system.
 * @param email String
 * @param name String
 * @param surname String
 * @param password String
 */

    public void addAdministrator(KWArrayList<User> userList, String email, String name, String surname, String password)    throws Error
    {
        for(int i = 0; i < userList.size(); i++)
        {
            if(userList.get(i) instanceof Administrator)
            {
                if(email.equals(userList.get(i).getEmail()))    throw new Error ("There is a user with the same e-mail address");
            }
        }
        Administrator temp = new Administrator(email, name, surname, password);
        userList.add(temp);
        System.out.println(temp + " is added to the System");
    }
```

$T(n) = O(n)$

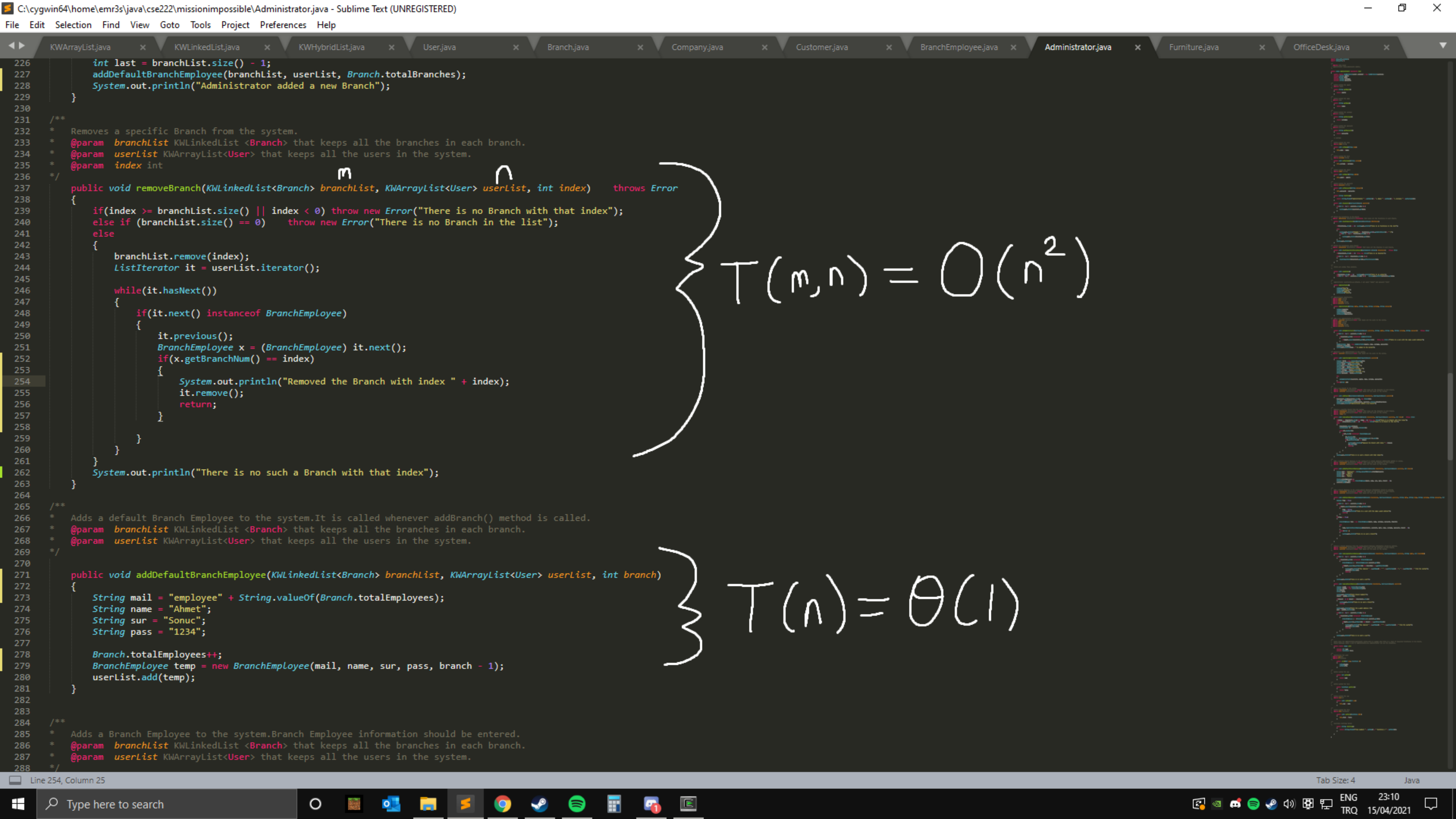
```java
/** Registers a new Administrator to the system.
 * @param  userList KWArrayList<User> that keeps all the users in the system.
 */

    public void registerAdministrator(KWArrayList<User> userList)
    {
        Scanner third = new Scanner(System.in);
        System.out.println("Enter email :");
        String email = third.nextLine();
        System.out.println("Enter Name :");
        String name = third.nextLine();
        System.out.println("Enter Surname :");
        String surname = third.nextLine();
        System.out.println("Enter Password :");
        String password = third.nextLine();

        try
        {
            addAdministrator(userList, email, name, surname, password);
        }
        catch(Error e){}
    }
```

$T(n) = O(n)$

```java
/**
*   Adds a new Branch to the system.
*   @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
*   @param  userList KWArrayList<User> that keeps all the users in the system.
*/

    public void addBranch(KWLinkedList<Branch> branchList, KWArrayList<User> userList)
    {
        branchList.add(branchList.size(), new Branch());
        int last = branchList.size() - 1;
        addDefaultBranchEmployee(branchList, userList, Branch.totalBranches);
        System.out.println("Administrator added a new Branch");
    }
```

$T(n) = \Theta(1)$

```java
/**
*   Removes a specific Branch from the system.
*   @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
*   @param  userList KWArrayList<User> that keeps all the users in the system.
```



```
        int last = branchList.size() - 1;
        addDefaultBranchEmployee(branchList, userList, Branch.totalBranches);
        System.out.println("Administrator added a new Branch");
    }

/**
*   Removes a specific Branch from the system.
*   @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
*   @param  userList KWArrayList<User> that keeps all the users in the system.
*   @param  index int
*/
    public void removeBranch(KWLinkedList<Branch> branchList, KWArrayList<User> userList, int index)    throws Error
    {
        if(index >= branchList.size() || index < 0) throw new Error("There is no Branch with that index");
        else if (branchList.size() == 0)    throw new Error("There is no Branch in the list");
        else
        {
            branchList.remove(index);
            ListIterator it = userList.iterator();

            while(it.hasNext())
            {
                if(it.next() instanceof BranchEmployee)
                {
                    it.previous();
                    BranchEmployee x = (BranchEmployee) it.next();
                    if(x.getBranchNum() == index)
                    {
                        System.out.println("Removed the Branch with index " + index);
                        it.remove();
                        return;
                    }

                }
            }
        }
        System.out.println("There is no such a Branch with that index");
    }
```

m

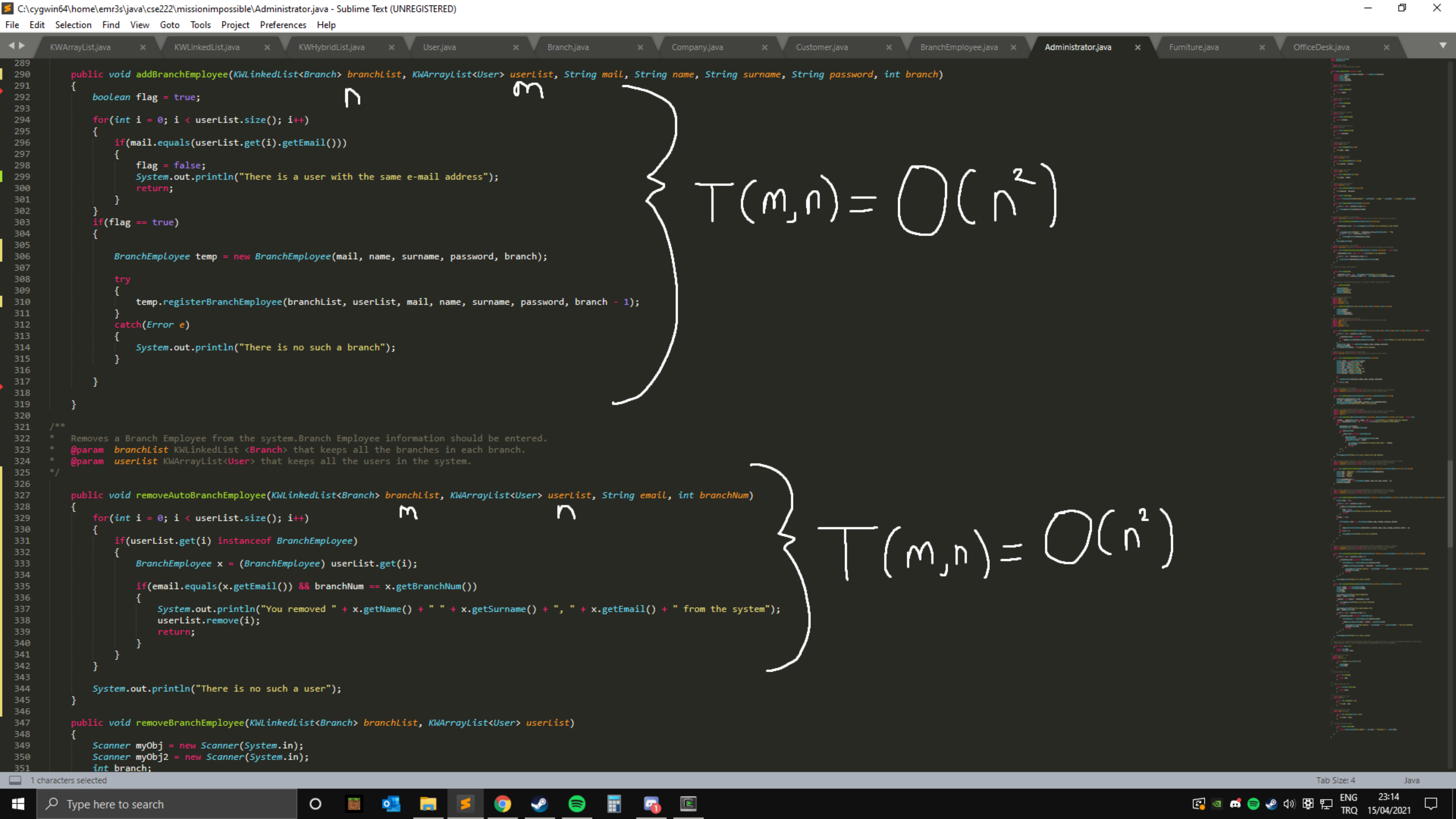
n

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

```
/**
*   Adds a default Branch Employee to the system.It is called whenever addBranch() method is called.
*   @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
*   @param  userList KWArrayList<User> that keeps all the users in the system.
*/

    public void addDefaultBranchEmployee(KWLinkedList<Branch> branchList, KWArrayList<User> userList, int branch)
    {
        String mail = "employee" + String.valueOf(Branch.totalEmployees);
        String name = "Ahmet";
        String sur = "Sonuc";
        String pass = "1234";

        Branch.totalEmployees++;
        BranchEmployee temp = new BranchEmployee(mail, name, sur, pass, branch - 1);
        userList.add(temp);
    }
```

$T(n) = \Theta(1)$

```
/**
*   Adds a Branch Employee to the system.Branch Employee information should be entered.
*   @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
*   @param  userList KWArrayList<User> that keeps all the users in the system.
*/
```


C:\cygwin64\home\emr3s\java\cse222\missionimpossible\Administrator.java - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
KWArrayList.java
KWLinkedList.java
KWHybridList.java
User.java
Branch.java
Company.java
Customer.java
BranchEmployee.java
Administrator.java
Furniture.java
OfficeDesk.java

```java

    public void addBranchEmployee(KWLinkedList<Branch> branchList, KWArrayList<User> userList, String mail, String name, String surname, String password, int branch)
    {
        boolean flag = true;

        for(int i = 0; i < userList.size(); i++)
        {
            if(mail.equals(userList.get(i).getEmail()))
            {
                flag = false;
                System.out.println("There is a user with the same e-mail address");
                return;
            }
        }
        if(flag == true)
        {

            BranchEmployee temp = new BranchEmployee(mail, name, surname, password, branch);

            try
            {
                temp.registerBranchEmployee(branchList, userList, mail, name, surname, password, branch - 1);
            }
            catch(Error e)
            {
                System.out.println("There is no such a branch");
            }

        }

    }
```

n m

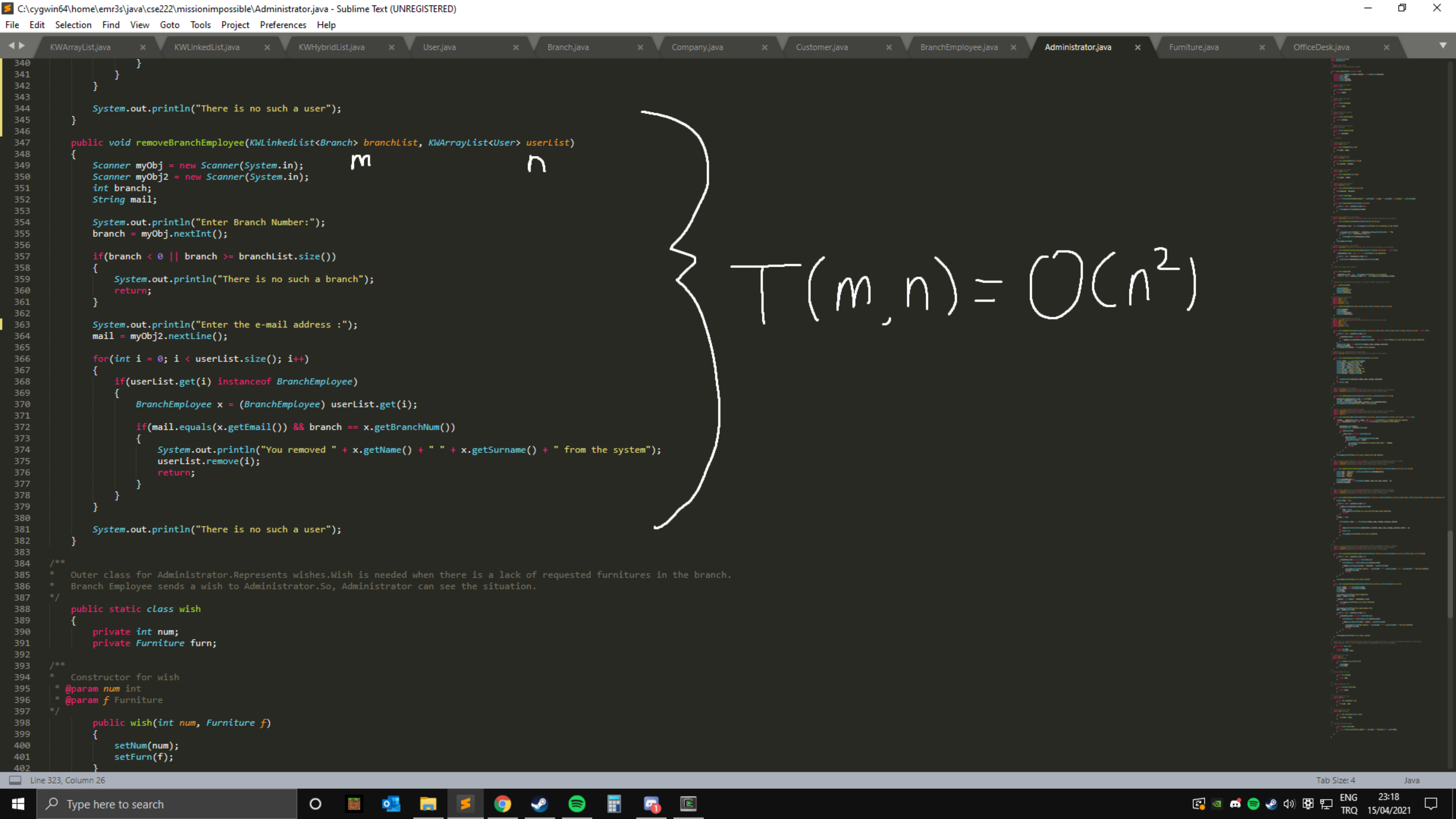
$$\left.\right\} T(m,n) = O(n^2)$$

```java

    /**
     *   Removes a Branch Employee from the system.Branch Employee information should be entered.
     *   @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
     *   @param  userList KWArrayList<User> that keeps all the users in the system.
     */

    public void removeAutoBranchEmployee(KWLinkedList<Branch> branchList, KWArrayList<User> userList, String email, int branchNum)
    {
        for(int i = 0; i < userList.size(); i++)
        {
            if(userList.get(i) instanceof BranchEmployee)
            {
                BranchEmployee x = (BranchEmployee) userList.get(i);

                if(email.equals(x.getEmail()) && branchNum == x.getBranchNum())
                {
                    System.out.println("You removed " + x.getName() + " " + x.getSurname() + ", " + x.getEmail() + " from the system");
                    userList.remove(i);
                    return;
                }
            }
        }

        System.out.println("There is no such a user");
    }

    public void removeBranchEmployee(KWLinkedList<Branch> branchList, KWArrayList<User> userList)
    {
        Scanner myObj = new Scanner(System.in);
        Scanner myObj2 = new Scanner(System.in);
        int branch;
```

m n

$$\left.\right\} T(m,n) = O(n^2)$$

1 characters selected

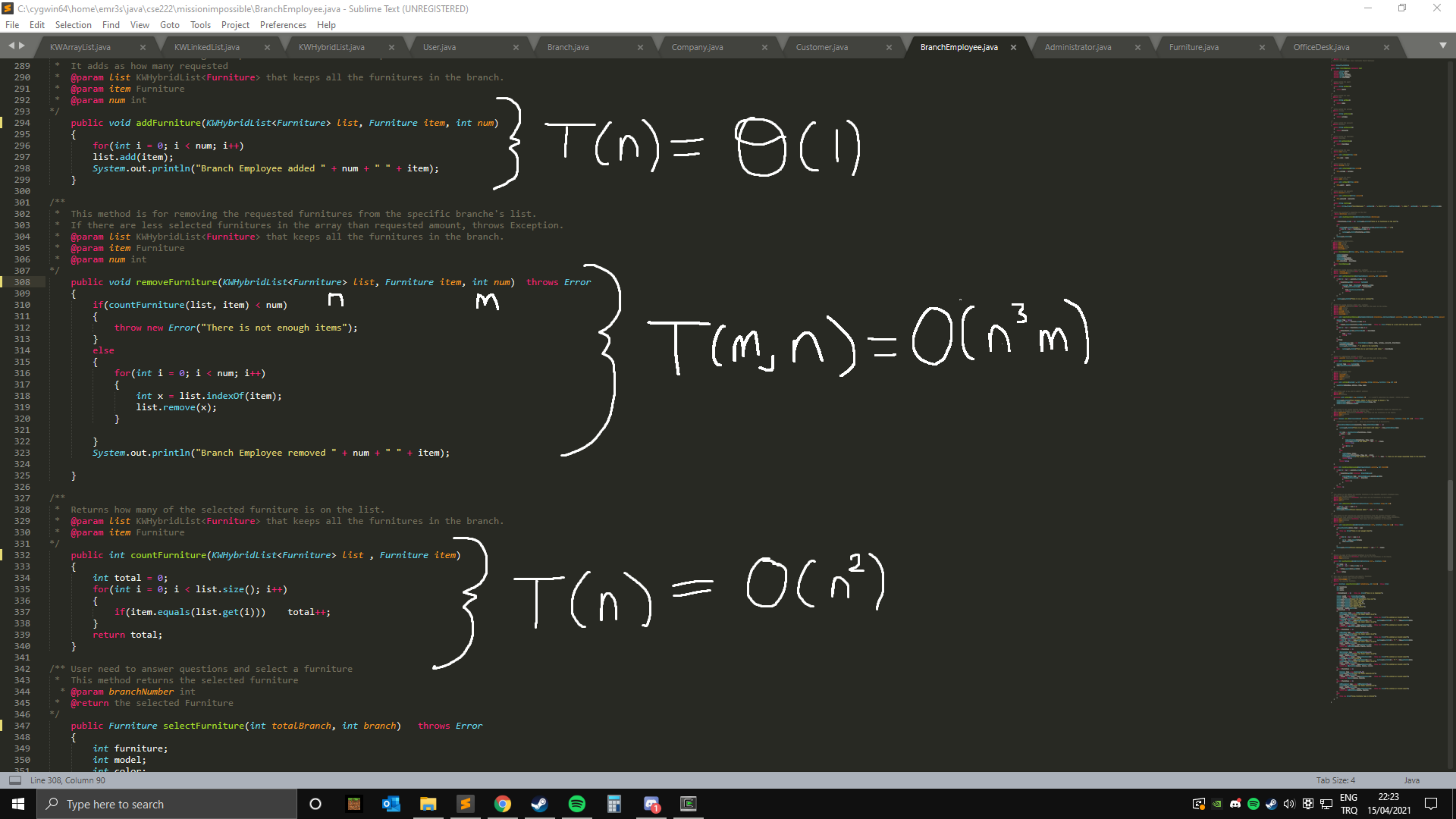


```
                }
            }
        }

        System.out.println("There is no such a user");
    }

    public void removeBranchEmployee(KWLinkedList<Branch> branchList, KWArrayList<User> userList)
    {
        Scanner myObj = new Scanner(System.in);
        Scanner myObj2 = new Scanner(System.in);
        int branch;
        String mail;

        System.out.println("Enter Branch Number:");
        branch = myObj.nextInt();

        if(branch < 0 || branch >= branchList.size())
        {
            System.out.println("There is no such a branch");
            return;
        }

        System.out.println("Enter the e-mail address :");
        mail = myObj2.nextLine();

        for(int i = 0; i < userList.size(); i++)
        {
            if(userList.get(i) instanceof BranchEmployee)
            {
                BranchEmployee x = (BranchEmployee) userList.get(i);

                if(mail.equals(x.getEmail()) && branch == x.getBranchNum())
                {
                    System.out.println("You removed " + x.getName() + " " + x.getSurname() + " from the system");
                    userList.remove(i);
                    return;
                }
            }
        }

        System.out.println("There is no such a user");
    }

/**
*   Outer class for Administrator.Represents wishes.Wish is needed when there is a lack of requested furnitures in the branch.
*   Branch Employee sends a wish to Administrator.So, Administrator can see the situation.
*/
    public static class wish
    {
        private int num;
        private Furniture furn;

/**
*   Constructor for wish
 * @param num int
 * @param f Furniture
*/
        public wish(int num, Furniture f)
        {
            setNum(num);
            setFurn(f);
        }
```

m n

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

BRANCH EMPLOYEE



```
 *  It adds as how many requested
 *  @param list KWHybridList<Furniture> that keeps all the furnitures in the branch.
 *  @param item Furniture
 *  @param num int
*/
    public void addFurniture(KWHybridList<Furniture> list, Furniture item, int num)
    {
        for(int i = 0; i < num; i++)
        list.add(item);
        System.out.println("Branch Employee added " + num + " " + item);
    }
```

$$T(n) = \Theta(1)$$

```

/**
 *  This method is for removing the requested furnitures from the specific branche's list.
 *  If there are less selected furnitures in the array than requested amount, throws Exception.
 *  @param list KWHybridList<Furniture> that keeps all the furnitures in the branch.
 *  @param item Furniture
 *  @param num int
*/
    public void removeFurniture(KWHybridList<Furniture> list, Furniture item, int num)  throws Error
    {
        if(countFurniture(list, item) < num)
        {
            throw new Error("There is not enough items");
        }
        else
        {
            for(int i = 0; i < num; i++)
            {
                int x = list.indexOf(item);
                list.remove(x);
            }

        }
        System.out.println("Branch Employee removed " + num + " " + item);

    }
```

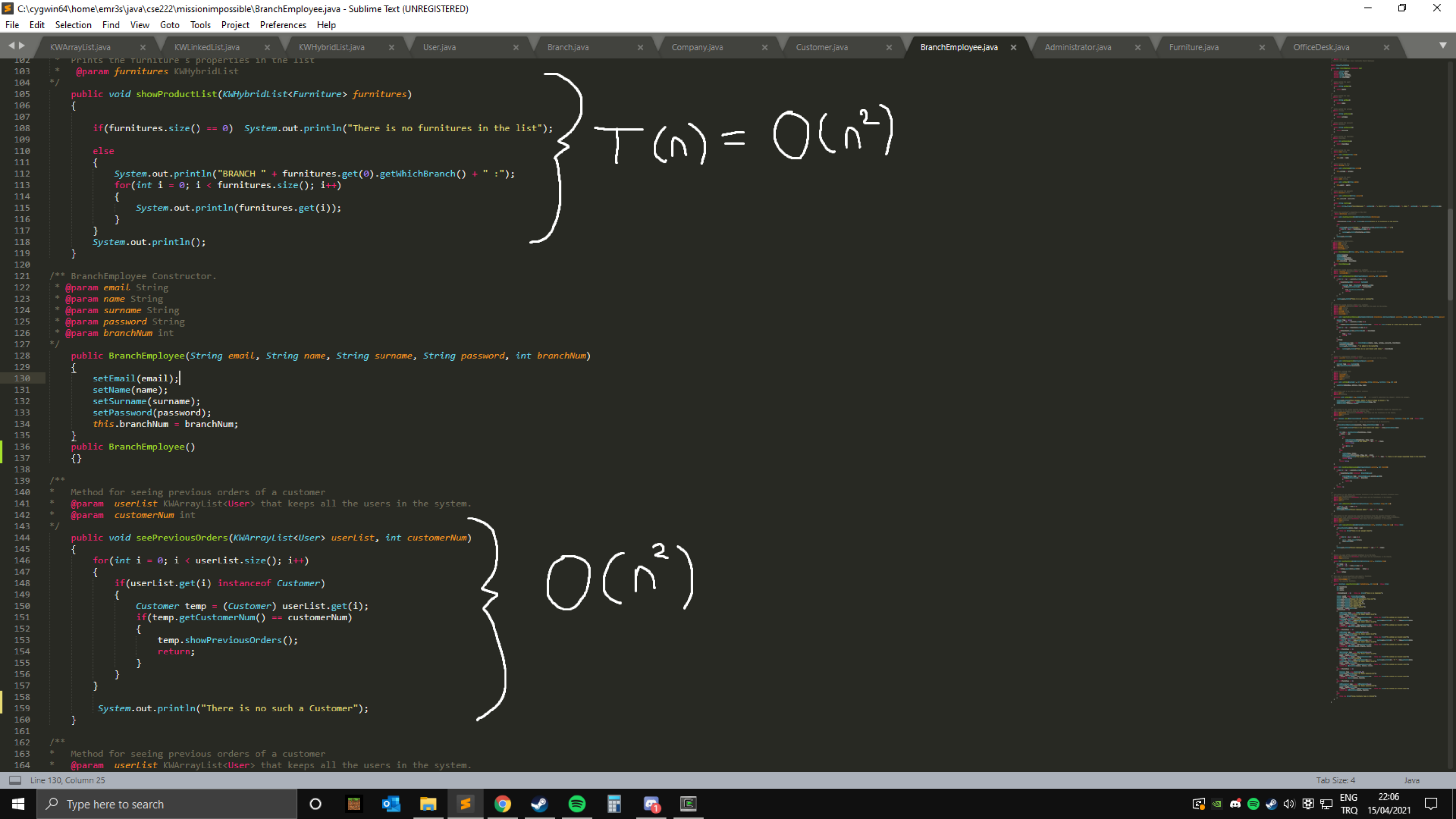
n m

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

```

/**
 *  Returns how many of the selected furniture is on the list.
 *  @param list KWHybridList<Furniture> that keeps all the furnitures in the branch.
 *  @param item Furniture
*/
    public int countFurniture(KWHybridList<Furniture> list , Furniture item)
    {
        int total = 0;
        for(int i = 0; i < list.size(); i++)
        {
            if(item.equals(list.get(i)))    total++;
        }
        return total;
    }
```

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

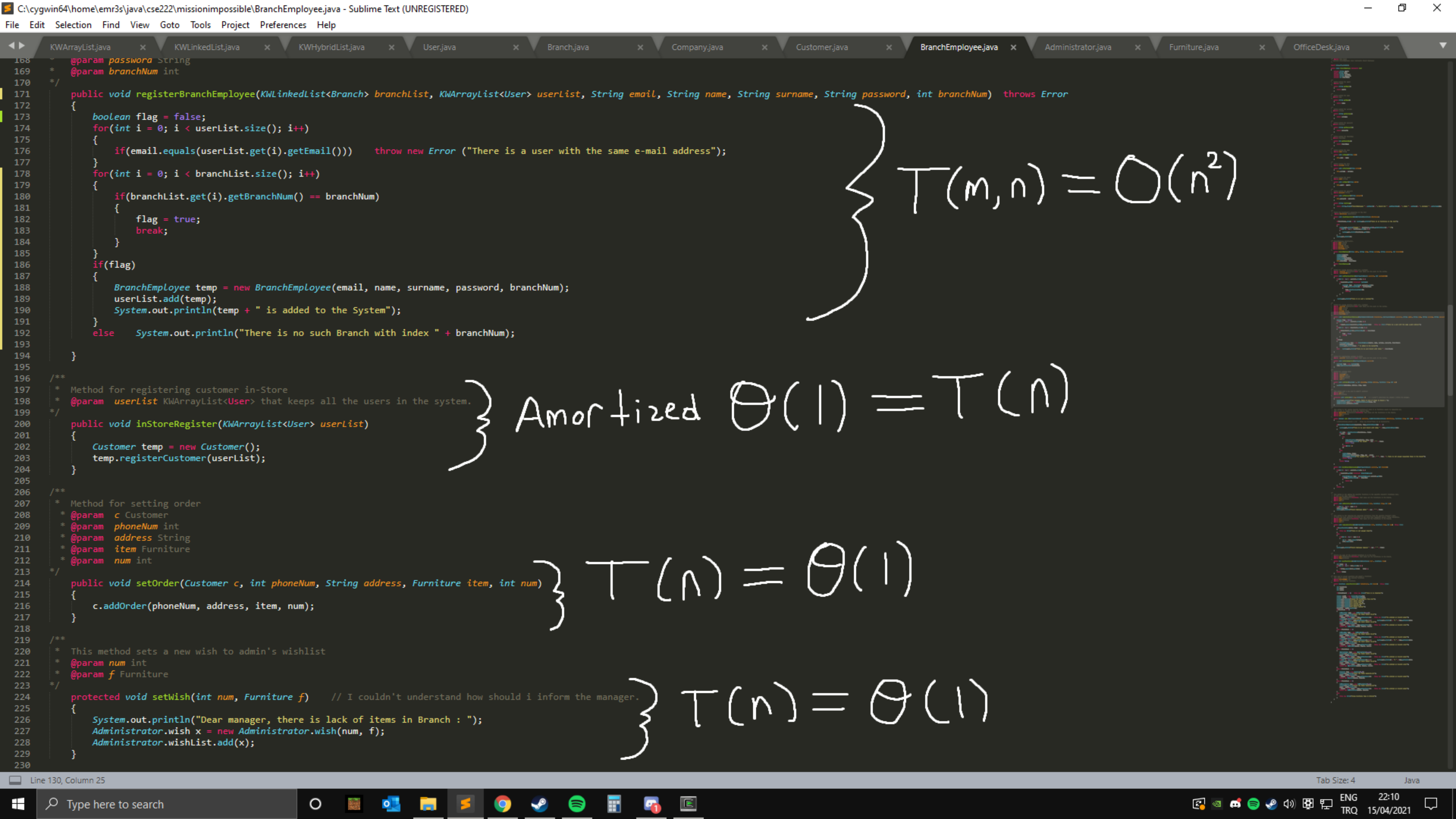
```

/** User need to answer questions and select a furniture
 *  This method returns the selected furniture
  * @param branchNumber int
 *  @return the selected Furniture
*/
    public Furniture selectFurniture(int totalBranch, int branch)   throws Error
    {
        int furniture;
        int model;
        int color;
```



```
 *  Prints the furniture's properties in the list
 *   @param furnitures KWHybridList
 */
    public void showProductList(KWHybridList<Furniture> furnitures)
    {

        if(furnitures.size() == 0)  System.out.println("There is no furnitures in the list");

        else
        {
            System.out.println("BRANCH " + furnitures.get(0).getWhichBranch() + " :");
            for(int i = 0; i < furnitures.size(); i++)
            {
                System.out.println(furnitures.get(i));
            }
        }
        System.out.println();
    }

/** BranchEmployee Constructor.
 * @param email String
 * @param name String
 * @param surname String
 * @param password String
 * @param branchNum int
 */
    public BranchEmployee(String email, String name, String surname, String password, int branchNum)
    {
        setEmail(email);
        setName(name);
        setSurname(surname);
        setPassword(password);
        this.branchNum = branchNum;
    }
    public BranchEmployee()
    {}

/**
 *   Method for seeing previous orders of a customer
 *   @param  userList KWArrayList<User> that keeps all the users in the system.
 *   @param  customerNum int
 */
    public void seePreviousOrders(KWArrayList<User> userList, int customerNum)
    {
        for(int i = 0; i < userList.size(); i++)
        {
            if(userList.get(i) instanceof Customer)
            {
                Customer temp = (Customer) userList.get(i);
                if(temp.getCustomerNum() == customerNum)
                {
                    temp.showPreviousOrders();
                    return;
                }
            }
        }

         System.out.println("There is no such a Customer");
    }

/**
 *   Method for seeing previous orders of a customer
 *   @param  userList KWArrayList<User> that keeps all the users in the system.
```

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

$O(n^2)$



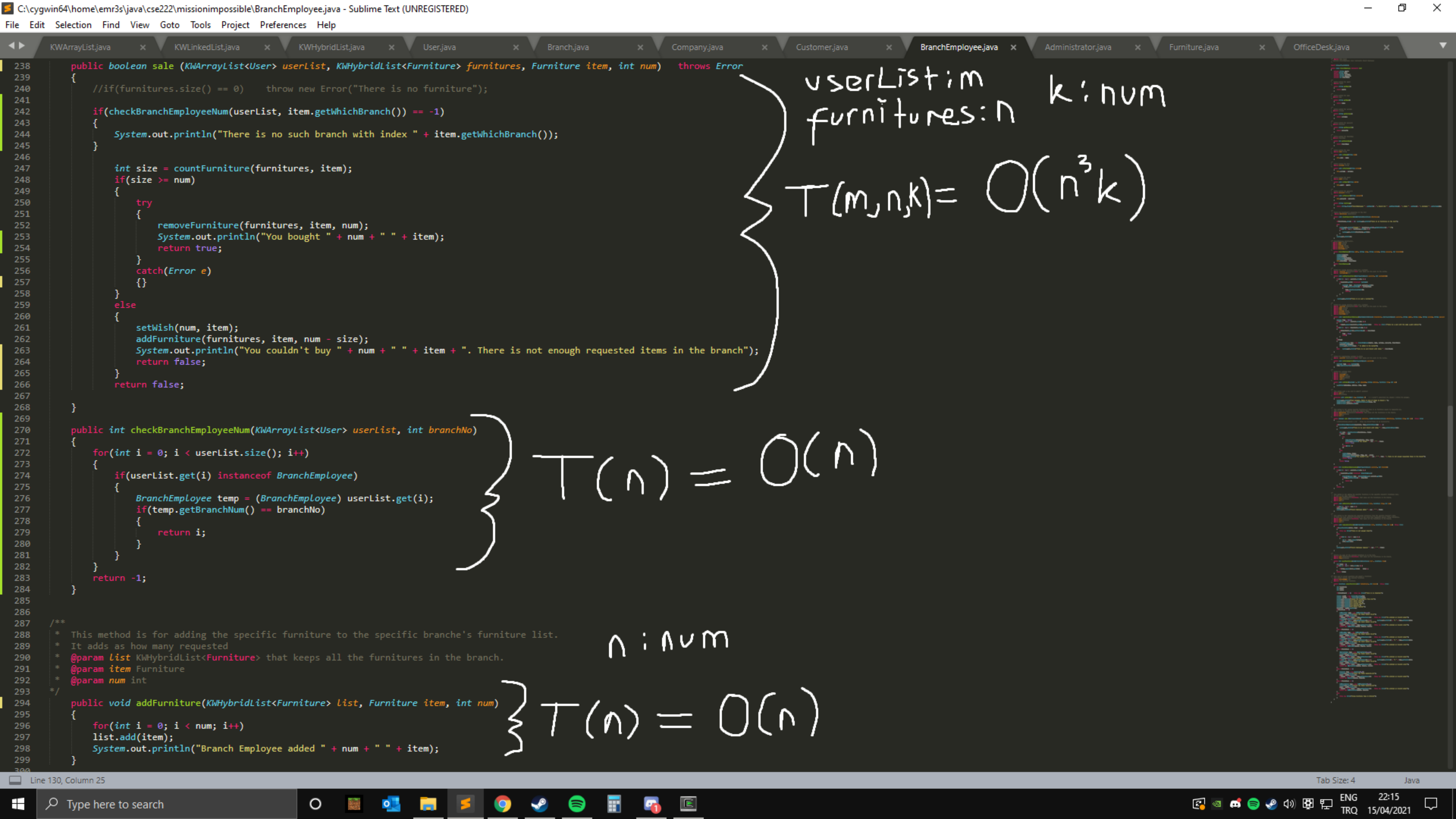
```java
     * @param password String
     * @param branchNum int
     */
    public void registerBranchEmployee(KWLinkedList<Branch> branchList, KWArrayList<User> userList, String email, String name, String surname, String password, int branchNum)  throws Error
    {
        boolean flag = false;
        for(int i = 0; i < userList.size(); i++)
        {
            if(email.equals(userList.get(i).getEmail()))    throw new Error ("There is a user with the same e-mail address");
        }
        for(int i = 0; i < branchList.size(); i++)
        {
            if(branchList.get(i).getBranchNum() == branchNum)
            {
                flag = true;
                break;
            }
        }
        if(flag)
        {
            BranchEmployee temp = new BranchEmployee(email, name, surname, password, branchNum);
            userList.add(temp);
            System.out.println(temp + " is added to the System");
        }
        else    System.out.println("There is no such Branch with index " + branchNum);

    }

    /**
     * Method for registering customer in-Store
     * @param  userList KWArrayList<User> that keeps all the users in the system.
     */
    public void inStoreRegister(KWArrayList<User> userList)
    {
        Customer temp = new Customer();
        temp.registerCustomer(userList);
    }

    /**
     * Method for setting order
     * @param  c Customer
     * @param  phoneNum int
     * @param  address String
     * @param  item Furniture
     * @param  num int
     */
    public void setOrder(Customer c, int phoneNum, String address, Furniture item, int num)
    {
        c.addOrder(phoneNum, address, item, num);
    }

    /**
     * This method sets a new wish to admin's wishlist
     * @param num int
     * @param f Furniture
     */
    protected void setWish(int num, Furniture f)    // I couldn't understand how should i inform the manager.
    {
        System.out.println("Dear manager, there is lack of items in Branch : ");
        Administrator.wish x = new Administrator.wish(num, f);
        Administrator.wishList.add(x);
    }
```

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

Amortized $\Theta(1) = T(n)$

$T(n) = \Theta(1)$

$T(n) = \Theta(1)$



```java
    public boolean sale (KWArrayList<User> userList, KWHybridList<Furniture> furnitures, Furniture item, int num)   throws Error
    {
        //if(furnitures.size() == 0)    throw new Error("There is no furniture");

        if(checkBranchEmployeeNum(userList, item.getWhichBranch()) == -1)
        {
            System.out.println("There is no such branch with index " + item.getWhichBranch());
        }

            int size = countFurniture(furnitures, item);
            if(size >= num)
            {
                try
                {
                    removeFurniture(furnitures, item, num);
                    System.out.println("You bought " + num + " " + item);
                    return true;
                }
                catch(Error e)
                {}
            }
            else
            {
                setWish(num, item);
                addFurniture(furnitures, item, num - size);
                System.out.println("You couldn't buy " + num + " " + item + ". There is not enough requested items in the branch");
                return false;
            }
            return false;

    }
```

userList:m

furnitures:n

k:num

$T(m,n,k) = O(n^3k)$

```java
    public int checkBranchEmployeeNum(KWArrayList<User> userList, int branchNo)
    {
        for(int i = 0; i < userList.size(); i++)
        {
            if(userList.get(i) instanceof BranchEmployee)
            {
                BranchEmployee temp = (BranchEmployee) userList.get(i);
                if(temp.getBranchNum() == branchNo)
                {
                    return i;
                }
            }
        }
        return -1;
    }
```

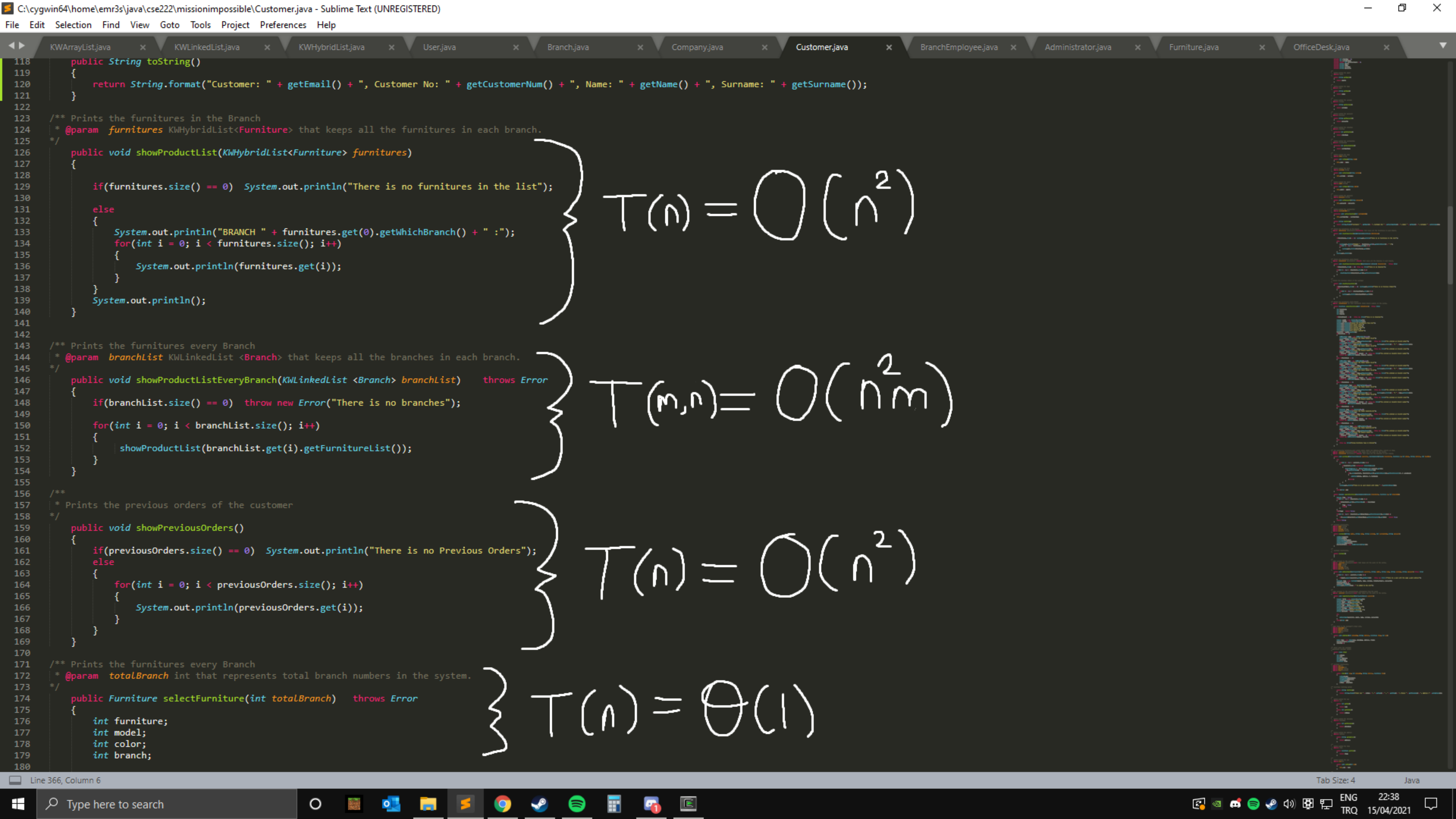
$T(n) = O(n)$

```java
/**
 * This method is for adding the specific furniture to the specific branche's furniture list.
 * It adds as how many requested
 * @param list KWHybridList<Furniture> that keeps all the furnitures in the branch.
 * @param item Furniture
 * @param num int
 */
    public void addFurniture(KWHybridList<Furniture> list, Furniture item, int num)
    {
        for(int i = 0; i < num; i++)
        list.add(item);
        System.out.println("Branch Employee added " + num + " " + item);
    }
```

n:num

$T(n) = O(n)$

CUSTOMER



```
    public String toString()
    {
        return String.format("Customer: " + getEmail() + ", Customer No: " + getCustomerNum() + ", Name: " + getName() + ", Surname: " + getSurname());
    }

/** Prints the furnitures in the Branch
 * @param  furnitures KWHybridList<Furniture> that keeps all the furnitures in each branch.
 */
    public void showProductList(KWHybridList<Furniture> furnitures)
    {

        if(furnitures.size() == 0)  System.out.println("There is no furnitures in the list");

        else
        {
            System.out.println("BRANCH " + furnitures.get(0).getWhichBranch() + " :");
            for(int i = 0; i < furnitures.size(); i++)
            {
                System.out.println(furnitures.get(i));
            }
        }
        System.out.println();
    }
```

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

```
/** Prints the furnitures every Branch
 * @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
 */
    public void showProductListEveryBranch(KWLinkedList <Branch> branchList)    throws Error
    {
        if(branchList.size() == 0)  throw new Error("There is no branches");

        for(int i = 0; i < branchList.size(); i++)
        {
            showProductList(branchList.get(i).getFurnitureList());
        }
    }
```

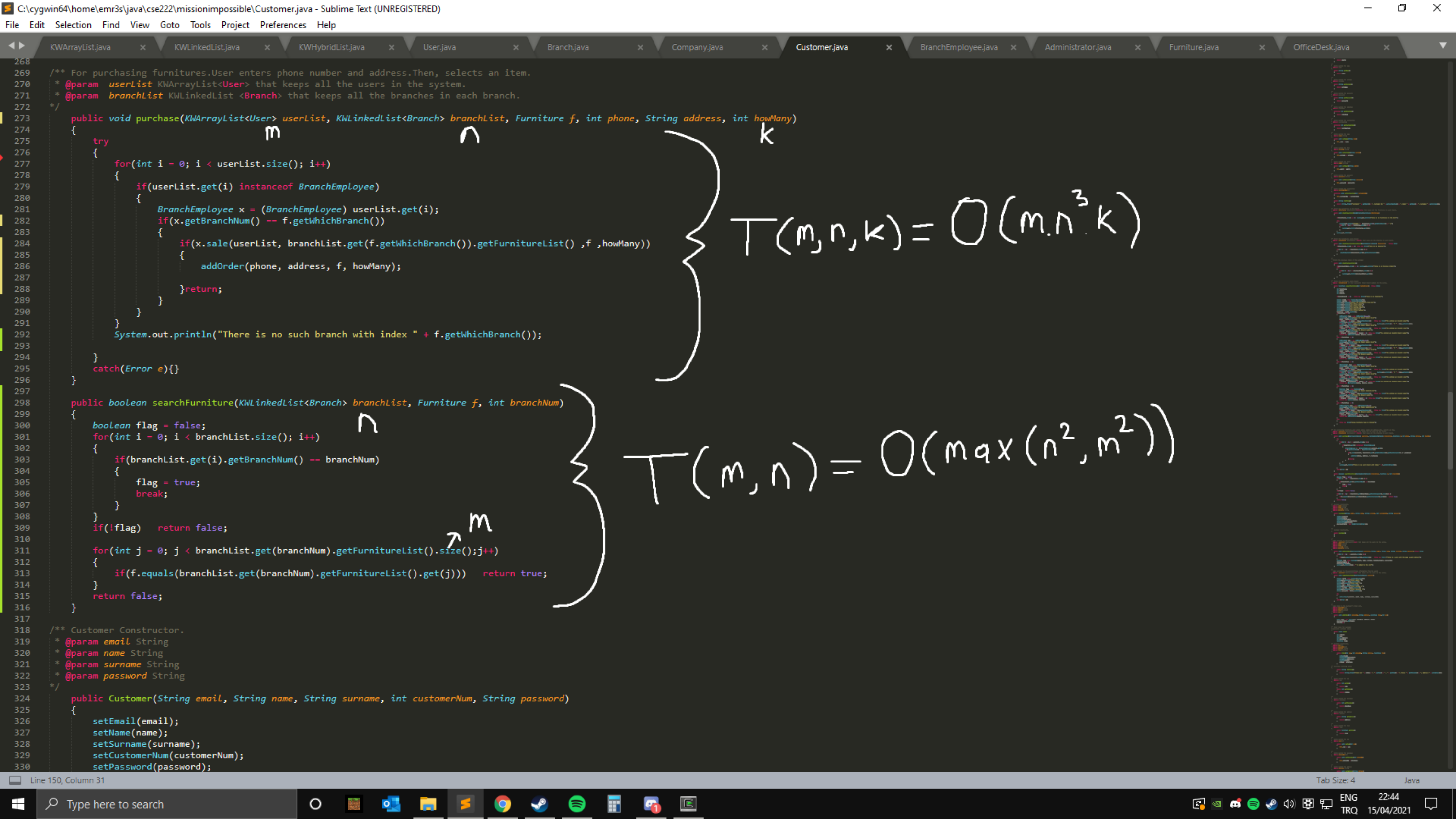
$T(m,n) = O(n^2 m)$

```
/**
 * Prints the previous orders of the customer
 */
    public void showPreviousOrders()
    {
        if(previousOrders.size() == 0)  System.out.println("There is no Previous Orders");
        else
        {
            for(int i = 0; i < previousOrders.size(); i++)
            {
                System.out.println(previousOrders.get(i));
            }
        }
    }
```

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

```
/** Prints the furnitures every Branch
 * @param  totalBranch int that represents total branch numbers in the system.
 */
    public Furniture selectFurniture(int totalBranch)   throws Error
    {
        int furniture;
        int model;
        int color;
        int branch;

```

$T(n) = \Theta(1)$


C:\cygwin64\home\emr3s\java\cse222\missionimpossible\Customer.java - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
KWArrayList.java
KWLinkedList.java
KWHybridList.java
User.java
Branch.java
Company.java
Customer.java
BranchEmployee.java
Administrator.java
Furniture.java
OfficeDesk.java

```java
    /** For purchasing furnitures.User enters phone number and address.Then, selects an item.
     * @param  userList KWArrayList<User> that keeps all the users in the system.
     * @param  branchList KWLinkedList <Branch> that keeps all the branches in each branch.
    */
        public void purchase(KWArrayList<User> userList, KWLinkedList<Branch> branchList, Furniture f, int phone, String address, int howMany)
        {
            try
            {
                for(int i = 0; i < userList.size(); i++)
                {
                    if(userList.get(i) instanceof BranchEmployee)
                    {
                        BranchEmployee x = (BranchEmployee) userList.get(i);
                        if(x.getBranchNum() == f.getWhichBranch())
                        {
                            if(x.sale(userList, branchList.get(f.getWhichBranch()).getFurnitureList() ,f ,howMany))
                            {
                                addOrder(phone, address, f, howMany);

                            }return;
                        }
                    }
                }
                System.out.println("There is no such branch with index " + f.getWhichBranch());

            }
            catch(Error e){}
        }

        public boolean searchFurniture(KWLinkedList<Branch> branchList, Furniture f, int branchNum)
        {
            boolean flag = false;
            for(int i = 0; i < branchList.size(); i++)
            {
                if(branchList.get(i).getBranchNum() == branchNum)
                {
                    flag = true;
                    break;
                }
            }
            if(!flag)   return false;

            for(int j = 0; j < branchList.get(branchNum).getFurnitureList().size();j++)
            {
                if(f.equals(branchList.get(branchNum).getFurnitureList().get(j)))   return true;
            }
            return false;
        }

    /** Customer Constructor.
     * @param email String
     * @param name String
     * @param surname String
     * @param password String
    */
        public Customer(String email, String name, String surname, int customerNum, String password)
        {
            setEmail(email);
            setName(name);
            setSurname(surname);
            setCustomerNum(customerNum);
            setPassword(password);
```

m

n

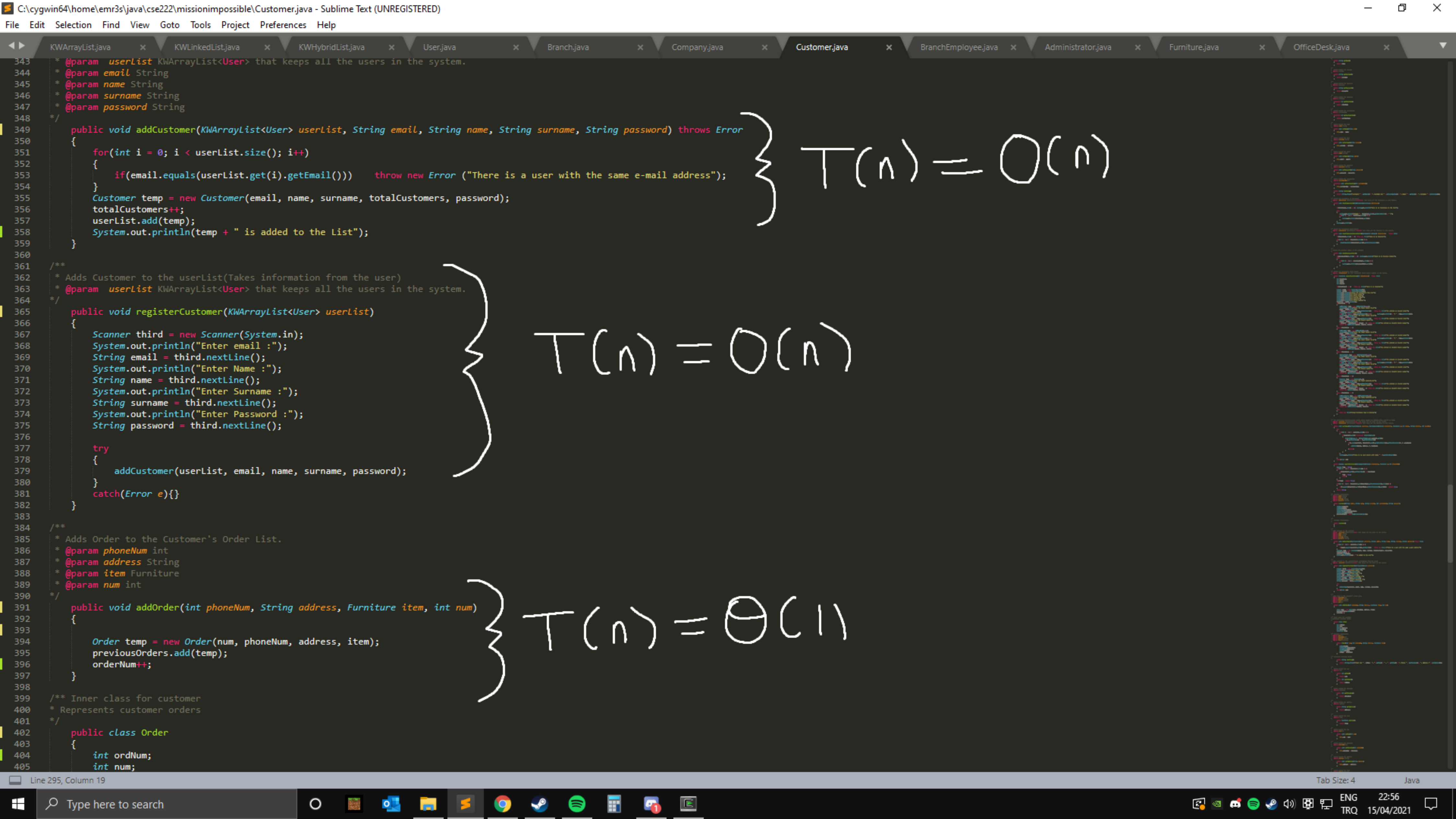
k

$$T(m,n,k) = O(m.n^3.k)$$

n

$$T(m,n) = O(max(n^2, m^2))$$

$\nearrow m$



```
 * @param  userList KWArrayList<User> that keeps all the users in the system.
 * @param email String
 * @param name String
 * @param surname String
 * @param password String
 */
    public void addCustomer(KWArrayList<User> userList, String email, String name, String surname, String password) throws Error
    {
        for(int i = 0; i < userList.size(); i++)
        {
            if(email.equals(userList.get(i).getEmail()))    throw new Error ("There is a user with the same e-mail address");
        }
        Customer temp = new Customer(email, name, surname, totalCustomers, password);
        totalCustomers++;
        userList.add(temp);
        System.out.println(temp + " is added to the List");
    }
```

$T(n) = O(n)$

```
/**
 * Adds Customer to the userList(Takes information from the user)
 * @param  userList KWArrayList<User> that keeps all the users in the system.
 */
    public void registerCustomer(KWArrayList<User> userList)
    {
        Scanner third = new Scanner(System.in);
        System.out.println("Enter email :");
        String email = third.nextLine();
        System.out.println("Enter Name :");
        String name = third.nextLine();
        System.out.println("Enter Surname :");
        String surname = third.nextLine();
        System.out.println("Enter Password :");
        String password = third.nextLine();

        try
        {
            addCustomer(userList, email, name, surname, password);
        }
        catch(Error e){}
    }
```

$T(n) = O(n)$

```
/**
 * Adds Order to the Customer's Order List.
 * @param phoneNum int
 * @param address String
 * @param item Furniture
 * @param num int
 */
    public void addOrder(int phoneNum, String address, Furniture item, int num)
    {

        Order temp = new Order(num, phoneNum, address, item);
        previousOrders.add(temp);
        orderNum++;
    }
```

$T(n) = \Theta(1)$

```
/** Inner class for customer
 * Represents customer orders
 */
    public class Order
    {
        int ordNum;
        int num;
```