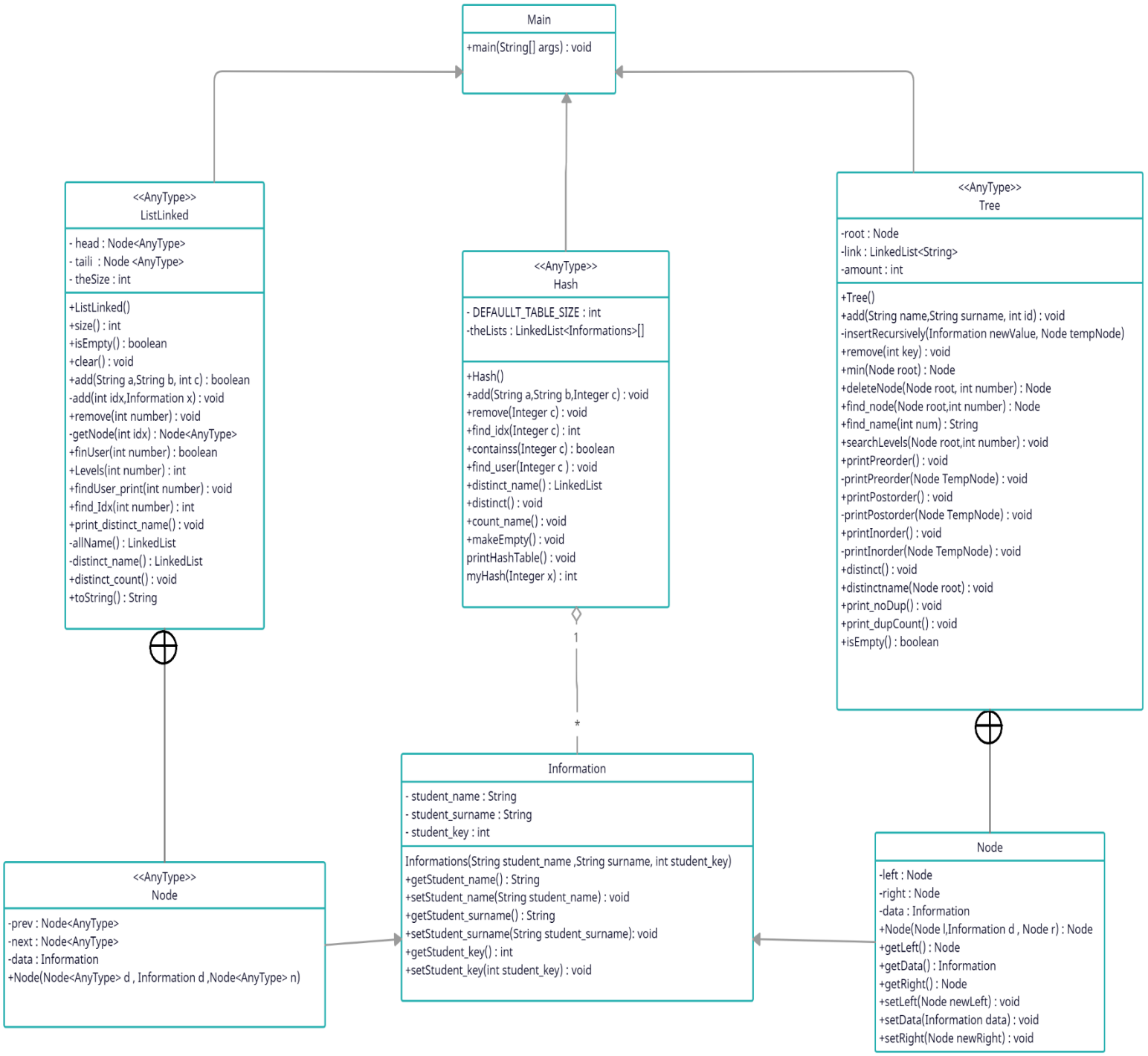
FALL SEMESTER PROJECT REPORT

- **I will show the classes I have created with UML diagram.**



MAİN CLASS

```
public class Main {
public static void main(String[] args) {

    ListLinked<Informations> link = new ListLinked<>();
    Tree<Informations> tree = new Tree<>();
    Hash<Informations> hash = new Hash<>();
    Scanner scan = new Scanner(System.in);

 String  menu = "select operation from below  menu \n"+
                "0) Exit\n"+
                "1) Add Student\n"+
                "2) Delete Student\n"+
                "3) Find Student\n"+
                "4) List All Student\n"+
                "5) List Distinct Student\n"+
                "6) List Name Count\n"+
                "7) About";
```

First, we start by creating objects from each structures class.

Note: ListLinked add Same Student Id !!

Created Menu

```
while(true){

  System.out.println("************************************");
  System.out.println(menu);
  System.out.println("************************************");
  int trans = scan.nextInt();
  switch(trans){

     case 0:System.out.println("Quit...");
      return;
      //*****************************************************
     case 1:
         System.out.println("Student ID: ");
         int id = scan.nextInt();
         scan.nextLine();
         System.out.println("Student Name ");
         String name = scan.nextLine();
         System.out.println("Student Surname ");
         String surname = scan.nextLine();
         hash.add(name, surname, id);
         link.add(name, surname, id);
         tree.add(name, surname, id);
         break;
```

Adding has been done in each structure

```
 case 2:
     System.out.println("Which student ID you want to delete ");
     int stu_id = scan.nextInt();
      hash.remove(stu_id);
      link.remove(stu_id);
      tree.remove(stu_id);
     break;
 //*****************************************************
 case 3:
     System.out.println("Which student ID you want to find ");
     int stu_id_find = scan.nextInt();
     if(link.findUser(stu_id_find)==false){
         System.out.println("User not found ...");
     }
     else{
     System.out.println("---------List---------");
     link.findUser_print(stu_id_find);
     System.out.println("----------Tree--------");
     tree.findUser_print(stu_id_find);
      System.out.println("---------Hash---------");
     hash.find_user(stu_id_find);
     }
  break;
```

Removing has been done in each structure

GENERAL NUMBER CONTROL

Finding has been done in each structure

Writes 1 if no jump for all of them(my choice) !!!!

MAİN CLASS

```
case 4:
     System.out.println("Which structure Do you want to continue ?\n"+
                        "1-List\n"
                      + "2-Tree\n"
                      + "3-Hash");
         int k = scan.nextInt();
    if(k==1)
     System.out.println(link);
    else if(k==2){
        System.out.println("----INORDER----");
         tree.printInorder();
            System.out.println("----POSTORDER----");
               tree.printPostorder();
                  System.out.println("----PREORDER----");
                     tree.printPreorder();
    }
    else if(k==3)
    hash.printHashTable();
    else
        System.out.println("Please choose one of theese");
    break;
 //*****************************************************
case 5:
       System.out.println("----HASH-----");
          hash.distinct();
             System.out.println("----TREE-----");
                tree.print_noDup();
                   System.out.println("----LİST-----");
                      link.print_distinct_name();
 break;
```

MIXED

INORDER

ORDER OF ADDITON

```
''
case 6:
    System.out.println("----HASH-----");
         hash.count_name();
             System.out.println("----TREE-----");
                tree.print_dupCount();
                   System.out.println("----LİST-----");
                       link.distinct_count();
```

MIXED

INORDER

ORDER OF ADDITON

```
 break;
 //*****************************************************
case 7:
     System.out.println("------------------\n"+
                "180316042      ****\n"
              + "Muratcan       ****\n"
              + "Erek           ****\n"
              + "------------------");
 break;
 default:
     System.out.println("Please Enter valid number !!");
 break;
```

INFORMATİON CLASS

```java
public class Informations {

    private String student_name;
    private String student_surname;
    private int student_key;

    public Informations(String student_name, String student_surname, int student_key) {
        this.student_name = student_name;
        this.student_surname = student_surname;
        this.student_key = student_key;
    }
```

Information class is a class where I create the properties of the data I want to load into the Nodes. I added 3 properties, namely name, Surname and key, and typed them into the constructor and created the object.

```java
public String getStudent_name() {
    return student_name;
}

public void setStudent_name(String student_name) {
    this.student_name = student_name;
}

public String getStudent_surname() {
    return student_surname;
}

public void setStudent_surname(String student_surname) {
    this.student_surname = student_surname;
}

public int getStudent_key() {
    return student_key;
}

public void setStudent_key(int student_key) {
    this.student_key = student_key;
}
```

I made it easy to use from another class by creating setter and getter methods.

LİSTLİNKED CLASS

```
ublic class ListLinked<AnyType>

    private Node<AnyType> head;
    private Node<AnyType> tail;
    private int theSize;

    public ListLinked()
    {
        clear();
    }

    public int size()
    {
        return theSize;
    }

    public boolean isEmpty()
    {
        return theSize == 0;
    }

    public void clear()
    {
        head = new Node<>(null, null, null);
        tail = new Node<>(head, null, null);
        head.next = tail;

        theSize = 0;
    }
```

First of all, I received the references of the objects as head and tail from the inner class node class. I created my objects by calling my clear method in Constructor and the head and tail were connected to each other.

```
public boolean add(String a, String b, int c)
{
    Informations stu = new Informations(a, b, c);
    add(theSize, stu);
    return true;
}

private void add(int idx, Informations x)
{

    Node<AnyType> p = getNode(idx);
    Node<AnyType> newNode = new Node<>( p.prev, x, p );
    newNode.prev.next = newNode;
    p.prev = newNode;

    theSize++;
}
```

Add (String a,String b,int c), method ,first of all, I have created an object for each student in the Information class. Then I called the add method with different parameters again(overloading) I set the parameter as thesize to make each student I want to add one by one, and after transferring the Information data into the node, we increase thesize.

LİSTLİNKED CLASS

```
public void remove(int number)
{
   if(find_Idx(number)==-1)
     System.out.println("Index " + find_Idx(number) + "; size " + theSize);
   else{
    Node<AnyType> p = getNode(find_Idx(number));
    p.prev.next = p.next;
    p.next.prev = p.prev;

    theSize--;
   }

}
```

Remove(int number) Method, first I set the parameter as the student number, then I checked the node with the student number using my method that finds the index number. If there is no such school number, I got an error. When the school number is found, I quickly reach the node using the getNode method, and then I'm decreasing one size

---

```
private Node<AnyType> getNode(int idx)
{
    Node<AnyType> p;

    if( idx <= theSize / 2 )
    {
        p = head.next;
        for( int i = 0; i < idx; i++ )
            p = p.next;
    }
    else
    {
        p = tail;
        for( int i = theSize; i > idx; i-- )
            p = p.prev;
    }

    return p;
}
```

getNode() method ,allows us to quickly access the index, which we have given here, shortens the path.

```
public boolean findUser(int number){

    Node<AnyType> p = head.next;

    for(int i =0;i<theSize;i++){

       if(p.data.getStudent_key()==number){

           return true;
       }
    p=p.next;
    }

   return false;

}
```

findUser(int number) Method, shows us whether that node exists when the student number is written.

LİSTLİNKED CLASS

```
public int Levels(int number){

    Node<AnyType> p = head.next;
    int count=0;
    for(int i =0;i<theSize;i++){

       if(p.data.getStudent_key()==number){

         return count;
       }
   p=p.next;
   count++;
    }
return -1;
}
```

Levels(int number) method, When the student number is written, the Levels method keeps a counter until it goes to that number and returns -1 if there is no student.

```
public int find_Idx(int number){


   Node<AnyType> p = head;
   int temp = 0,i =0;
   for(i =0;i<theSize;i++){

      if(p.next.data.getStudent_key()== number){
         return i ;
      }
      else{
         temp=-1;
      }
      p=p.next;
   }
      return temp ;
}
```

find_Idx(int number)method takes the variable i as a counter when the student number is written, if it matches. When it can't find it, it returns -1

```
public void findUser_print(int number){


        int idx= find_Idx(number);
        int level = Levels(number);

    System.out.println("Level: "+ (level+1)+"\n"+
                       "Student Name: "+getNode(idx).data.getStudent_name()+"\n"+
                       "Surname: "+getNode(idx).data.getStudent_surname());
```

findUser_print(int number) method, First we get the index with the find_Idx method and we find the number of jumps using the Levels method. Then, while the last process is printing them, we go to the node with the getNode method and perform the print operations.

LİSTLİNKED CLASS

```
public void print_distinct_name(){

    Node<AnyType> p = head.next;
    Node<AnyType> k;
    String temp ;
    int count=0;
    for(int i=0;i<theSize;i++){
        temp = p.data.getStudent_name();
        k=p;
        for(int j=find_Idx(k.data.getStudent_key());j>=0;j--){
            if(k.data.getStudent_name().equals(temp)){
                count++;
           }
            k =k.prev;
        }

        if(count==1){
            System.out.println(temp);     }
   p=p.next;
   count=0;
    }
}
```

print_distinct_name() method, First we keep the first value connected to the head in variable p, and next to the temporary k node and a counter, we create a nested loop, then we take the name in the first loop and in the second loop, we do a reverse check.If there is more than one of the same name, we do not print it only the counter When it is 1, we print and reset the counter to perform the operations again.

```
private LinkedList allName(){

    LinkedList<String> name = new LinkedList<>();
     Node<AnyType> p = head.next;
    for(int i =0;i<theSize;i++){

    name.add(p.data.getStudent_name());
    p = p.next;

    }
    return name;

}
```

allName() method browses through the List and puts all names in a LinkedList and returns a LinkedList.

LİSTLİNKED CLASS

```
private LinkedList distinct_name(){
    LinkedList<String> name = new LinkedList<>();
    Node<AnyType> p = head.next;
    Node<AnyType> k;
    String temp ;
    int count=0;
    for(int i=0;i<theSize;i++){
        temp = p.data.getStudent_name();
        k=p;
        for(int j=find_Idx(k.data.getStudent_key());j>=0;j--){
            if(k.data.getStudent_name().equals(temp)){
                count++;
            }
             k =k.prev;
        }

        if(count==1){
           name.add(temp);
        }
   p=p.next;
   count=0;
    }
return name;
 }
```

distinct_name() method performs the same operations in the print distinct_name() method, but returns LinkedList by throwing these values into the LinkedList I created at the beginning.

Distinct_count() method, We use two methods for this method 1.allName, 2.distinct_name method.distinct_name method, we get the name over the list and we make it scroll through the list with all names, we increase the counter for each name it finds and then we perform the print operation.

```
public void distinct_count(){

    int count =0;
   int i=0,j=0;

    System.out.print("{");
   for(i =0;i<distinct_name().size();i++){
       for( j =0;j<allName().size();j++){
           if(distinct_name().get(i).equals(allName().get(j))){
               count++;
           }
       }
       System.out.print(distinct_name().get(i)+"="+count);
       System.out.print(" ");
       count=0;
   }
    System.out.print("}\n");
       }
```

toString() We loop through LinkedList with a temporary temp node and examine the desired results.

```
public String toString(){

     String rStr ="<><><><><><><><><><><><><><><><><><><>\n";
    Node<AnyType> temp = head.next;
   for(int i=0; i<theSize; i++)
   {
       rStr = rStr +"["+(i+1)+"]"+"--->  "+ temp.data.getStudent_key() + " ";
       rStr = rStr + temp.data.getStudent_name()+ " ";
       rStr = rStr + temp.data.getStudent_surname() + " \n";
       temp = temp.next;
   }
   rStr = rStr + "<><><><><><><><><><><><><><><><><><><>";
    return rStr;
}
```

LİSTLİNKED CLASS

```
private class Node<AnyType>
{
    private Node<AnyType> prev;
    private Informations data;
    private Node<AnyType> next;

    public Node(Node<AnyType> p, Informations d, Node<AnyType> n)
    {
        prev = p;
        data = d;
        next = n;
    }
}
```

We create Inner Class named Node in it, we created prev and next variable in Node type and data variable from Information class type. Then Constructor was created.

TREE CLASS

```
public class Tree<AnyType> {

   private Node root;
   private LinkedList<String> link = new LinkedList<>();
   int amount = 0;
    public Tree() {

   }
```

Tree<AnyType> Class, I created a Node type root and created a temporary linkedlist and a counter to keep Value.

```
public void add(String name,String surname,int id)
{
    Informations stu = new Informations(name, surname, id);
    root = insertRecursively(stu, root);
}

private Node insertRecursively(Informations newValue, Node tempNode)
{
    if( tempNode == null )
        return new Node(null,newValue,null);

    if( newValue.getStudent_key() < tempNode.getData().getStudent_key())
        tempNode.setLeft(insertRecursively(newValue, tempNode.getLeft()));
    else if(newValue.getStudent_key() > tempNode.getData().getStudent_key())
        tempNode.setRight(insertRecursively(newValue, tempNode.getRight()));
    else{
        System.out.println("It is Private key so try to other thing");
    }
    return tempNode;
}
```

Add() method allows it to hold data from the Information class and calls insert recursive method.
InsertRecursive() method looks at primary key(Student_key) and adds according to Binary type.

```
public void remove(int key)
{

    root = deleteNode(root,key);
}
```

remove(int key),Calls the deletenode method and starts method from root

```
public  Node min(Node root) {
            if (root.left == null)
                    return root;
            else {
                    return min(root.left);
            }
    }
```

min(Node root),Finds the smallest value starting from root

TREE CLASS

```
public Node deleteNode(Node root, int number) {
        if (root == null)
                return null;
        if (root.data.getStudent_key() > number) {
                root.left = deleteNode(root.left, number);
        } else if (root.data.getStudent_key() < number) {
                root.right = deleteNode(root.right, number);

        } else {

                if (root.left != null && root.right != null) {
                        Node temp = root;

                        Node TempNode = min(temp.right);

                        root.data = TempNode.data;

                        root.right = deleteNode(root.right, TempNode.data.getStudent_key());

                }

                else if (root.left != null) {
                        root = root.left;
                }

                else if (root.right != null) {
                        root = root.right;
                }

                else
                        root = null;
        }
        return root;
}
```

deleteNode(Node root, int number),First of all, starting from the root, you go to the desired node. If the node to be deleted has a child, the information of the ancestor of that node is transferred to its child. If it has two children, the right smallest one is selected. Then the brought node is deleted from its old place.

```
public Node find_node(Node root, int number)
{
if (root==null || root.data.getStudent_key()==number)
    return root;


if (root.data.getStudent_key() < number) {

   return find_node(root.right, number);
}
else{

   return find_node(root.left, number);
}
}
```

Find_node (Node root, int number), goes to the top of the tree node according to the number typed and returns the node

```
 public String find_name(int num){

    return find_node(root, num).getData().getStudent_name();

 }
```

Find_name(int num), It takes the name above the node and returns it by calling find_node()

TREE CLASS

```
public void findUser_print(int number){
    Node Temp =find_node(root, number);
    searchLevels(root, number);

        System.out.println("Level: "+(amount+1));
     amount=0;

    System.out.println("Student Name: "+Temp.data.getStudent_name());
    System.out.println("Surname: "+Temp.data.getStudent_surname());


}
```

FindUser_print (int number), goes to the top of the node with find_node () and create a temporary node.SearchLevels () method gives the jump, then we reset the counter and finally we print.

```
  public void searchLevels(Node root, int number)
  {

  if (root==null || root.data.getStudent_key()==number) {
     return  ;
    }

  if (root.data.getStudent_key() < number) {
    amount++;
     searchLevels(root.right, number);

  }
  else{
     amount++;
     searchLevels(root.left, number);

  }

  }
```

searchLevels(Node root, int number) we increment the counter we keep outside until we get over the node we want

```
  public void distinct(){
  distinctname(root);

  }
  public void distinctname(Node root){

       if(root == null){
    return;
     }

     distinctname(root.getLeft());
     link.add(root.data.getStudent_name());
     distinctname(root.getRight());

  }
```

distinct(),Firstly distinctname() called.starting from root.

distinctname(),hovers on the tree and assigns it to the outside temporary LinkedList

TREE CLASS

```
public void printInorder()
{
    printInorder(root);
}

private void printInorder(Node tempNode)
{
   if(tempNode == null)
      return;



    printInorder(tempNode.getLeft());
    System.out.print("-----> "+tempNode.getData().getStudent_key() + " "+
                               tempNode.getData().getStudent_name()+" "+
                               tempNode.getData().getStudent_surname()+"\n");
    printInorder(tempNode.getRight());
}
```

printInorder(),strolling in the tree and prints as Inorder (L, Root, R)

```
  public void printPostorder()
  {
      printPostorder(root);
  }

  private void printPostorder(Node tempNode)
  {
      if(tempNode == null)
          return;

      printPostorder(tempNode.getLeft());
      printPostorder(tempNode.getRight());
      System.out.print("-----> "+tempNode.getData().getStudent_key() + " "+
                                 tempNode.getData().getStudent_name()+" "+
                                 tempNode.getData().getStudent_surname()+"\n");
  }
```

printPostorder(),strollin g in the tree and prints as postorder (L, R, Root)

```
public void printPreorder()
{
    printPreorder(root);
}

private void printPreorder(Node tempNode)
{
    if(tempNode == null)
        return;

    System.out.print("-----> "+tempNode.getData().getStudent_key() + " "+
                               tempNode.getData().getStudent_name()+" "+
                               tempNode.getData().getStudent_surname()+"\n");
    printPreorder(tempNode.getLeft());
    printPreorder(tempNode.getRight());
}
```

printPreorder(),strolling in the tree and prints as Inorder (Root, L, R)

TREE CLASS

```
public void print_noDup(){
  link.clear();
  distinct();
 TreeSet<String> list = new TreeSet<String>(link);
 for(String temp : list){
     System.out.println(temp);
 }

}
```

Print_noDup() , the list is first cleared as a counter, then the names are thrown back to the list and the list is thrown to the set created from the TreeSet class.Finally, loop printing is performed

```
public void print_dupCount(){
  link.clear();
  distinct();
  TreeMap<String,Integer> map = new TreeMap<>();
  String[] array = link.toArray(new String[link.size()]);
  int count ;

  for(String temp : array){
      String word = temp.toLowerCase();

    if(map.containsKey(word)){
        count=map.get(word);
        map.put(word,count+1);
    }
    else{
        map.put(word,1);
    }

  }
   System.out.println("Word count: ");
      System.out.println(map);
```

Print_dupCount(),the list is first cleared as a counter, then the names are thrown back to the list. The object is created from the TreeMap class and the LinkedList is transformed into an array, then the counter is kept by looping it, and finally, the print operation is performed.

```
}
    public class Node
{
    private Node left;
    private Informations data;
    private Node right;

    public Node(Node l, Informations d, Node r)
    {
        left = l;
        data = d;
        right = r;
    }
```

The Node class is included in the Tree class as the Inner class. Left and right is created from the Node type, and the data part is created in the Information type. For this, the constructor is written and it is included in the setter getter methods.

Hash CLASS

```
public class Hash<AnyType> {

    private static final int DEFAULT_TABLE_SIZE = 10;

    private LinkedList<Informations>[] theLists;

    public Hash()
    {
        theLists = new LinkedList[DEFAULT_TABLE_SIZE];

        for(int i=0; i<theLists.length; i++)
            theLists[i] = new LinkedList<>();
    }
```

First of all, because our key is integer in **Hash Class**, we make size 10 and write final so that it cannot be changed. Then we create an array containing linkedlists. We mark each element linkedlist in Constructor.

```
public void add(String a,String b,Integer c)
{
    Informations ınf = new Informations(a, b, c);
    LinkedList<Informations> whichList = theLists[myHash(c)];

    if(!containss(c)){
         whichList.add(ınf);
    }
```

Add (String a, String b, Integer c) method , We start the by creating objects from the Informations class. Then, according to the last digit of the Integer, we point to the linkedList to which the array is connected. After checking it with the Contains() method, we assign it into the LinkedList

Hash CLASS

```
public void remove(Integer c)
{
    LinkedList<Informations> whichList = theLists[myHash(c)];

    if(containss(c))  {
        int s = find_idx(c);
        if(s == -1){
            System.out.println("There is no user for this number");
        }
        whichList.remove(s);
    }
}
```

remove (Integer c) method, we go to the node according to the last digit of the value we have written and if there is the value we are looking for, we find the index number and perform the deletion. In case of absence, we send the user a notification by taking the index -1.

```
public int find_idx(Integer c){
    LinkedList<Informations> whichList = theLists[myHash(c)];
    for(int i =0;i<whichList.size();i++){
        if(whichList.get(i).getStudent_key()==c){

            return i;
        }

    }
   return  -1;
}
```

find_idx (Integer c) method goes to the node in the last digit of the Integer and returns the index when it finds the interval on that node, if not, it returns -1.

---

containss (Integer c) method goes to the node according to the last digit of the number and performs key control on that node.

```
public boolean containss(Integer c)
{
    LinkedList<Informations> whichList = theLists[myHash(c)];
     for(int i= 0 ;i<whichList.size();i++){
         if(whichList.get(i).getStudent_key()==c){
             return true;
         }
     }

          return false;
}
```

Hash CLASS

```
public void find_user(Integer c){
    LinkedList<Informations> whichList = theLists[myHash(c)];
     int s = find_idx(c);

    System.out.println("Levels: "+(s+1));
    System.out.println("Student Name: "+whichList.get(s).getStudent_name());
    System.out.println("Surname: "+whichList.get(s).getStudent_surname());

}
```

find_user (Integer c) method goes to the node according to the value received. We find the jump by navigating with find_idx and print the values on the node we reached.

```
public LinkedList distinct_name(){
    LinkedList<String> names = new LinkedList<>();
    LinkedList<Informations> whichList;

    for(int i=0; i<theLists.length; i++)
    {
        whichList = theLists[i];
        for(int j=0; j<whichList.size(); j++) {
        names.add(whichList.get(j).getStudent_name());

        }
    }
return names;
}
```

distinct_name (), First a temporary LInkedList is created and hovering above each node, the names are transferred to the temporary LinkedList, then the LinkedList is returned.

```
public void distinct(){

    HashSet<String> list = new HashSet<String>(distinct_name());
    for(String temp : list){
        System.out.println(temp);
    }
}
```

distinct (), This method performs the printing process by throwing the LinkedList from the distinct_name method into the HashSet.

Hash CLASS

```
public void count_name(){
    LinkedList<String> link = distinct_name();
    HashMap<String,Integer> map = new HashMap<>();
    String[] array = link.toArray(new String[link.size()]);
   int count ;

   for(String temp : array){
       String word = temp.toLowerCase();

      if(map.containsKey(word)){
          count=map.get(word);
          map.put(word,count+1);
      }
      else{
          map.put(word,1);
      }

   }
    System.out.println("Word count: ");
       System.out.println(map);


}
```

In the **count_name ()** method, we temporarily assign the list we received from the distinct_name() method to a reference. We convert the LinkedList we have received to an array, then we eliminate the problem by making all the names small. Using the ready-made HashMap class, we check the words in my list and as the same words come, we increase the value next to it and find the number of times and finally we print.

printHashTable(), we display the values as we want by hovering over each node with a nested loop.

```
public void printHashTable()
{
    LinkedList<Informations> whichList;

    for(int i=0; i<theLists.length; i++)
    {
        whichList = theLists[i];

        System.out.print("|" + i + "|" + " --> ");

        for(int j=0; j<whichList.size(); j++) {
            System.out.print("("+whichList.get(j).getStudent_key()+
                             ")"+whichList.get(j).getStudent_name()+" "+
                             whichList.get(j).getStudent_surname()+" --> ");
        }

        System.out.println();
    }
}
```

```
private int myHash(Integer x)
{
    return (x % theLists.length);
}
```

myHash(Integer x),Returns the last digit of the value we have written into it.