CSE 2105 – DATA STRUCTURES

2016 – 2017 FALL SEMESTER PROJECT REPORT

THE BAG ADT

I guess ı can implement the BAG by using binary search tree. I can hold the data and its count in the same Node.

I used this structure, Because dynamic size and faster.

Class Element<T>: I created a Element<T> class which has a 4 instances variables T data, int count, Element<T> left, Element<T> right.

I inserted a setter and getter methods of variables to reach them and also, 2 more method to up and down the count of the data, which is how many element there is the BAG.

Class Bag <E extends Comparable<E>> : I created Element<E> root to refer to the root of the tree.I had to extend Comparable <E>.

And I added the size field that holds the size and ı created get method of size.

And ı added temp pointer of root. And insert getter method

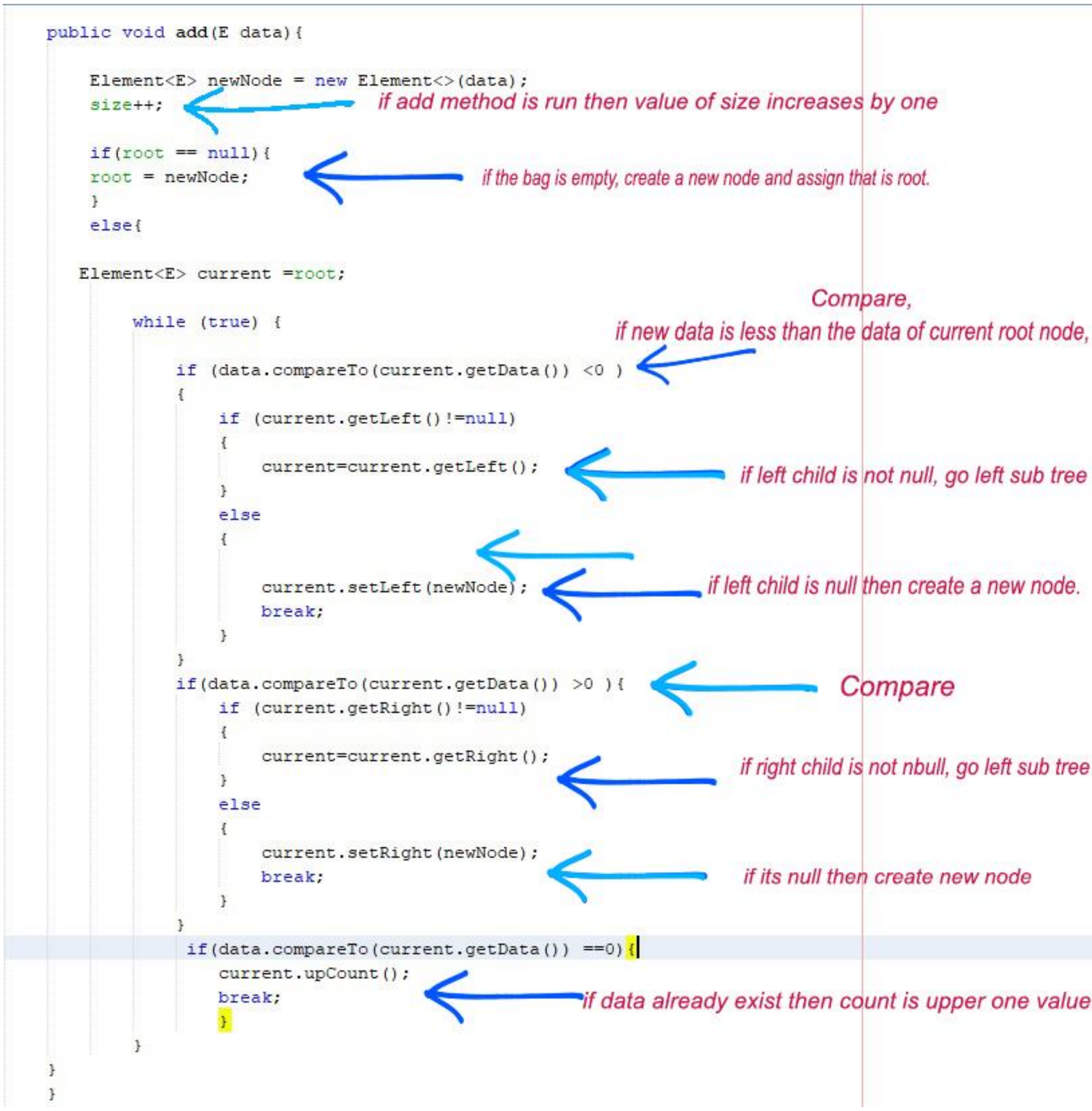
Because ı need to compare two objects while inserting, searching or deleting object from tree.

Bag class includes methods such as; add,remove,clear,contains…

Class Test: I created a test class, that contains a run method.

METHODS:

1- Add(E data) :



```
public void add(E data){

    Element<E> newNode = new Element<>(data);
    size++;

    if(root == null){
    root = newNode;
    }
    else{

   Element<E> current =root;

        while (true) {

            if (data.compareTo(current.getData()) <0 )
            {
                if (current.getLeft()!=null)
                {
                    current=current.getLeft();
                }
                else
                {

                    current.setLeft(newNode);
                    break;
                }
            }
            if(data.compareTo(current.getData()) >0 ){
                if (current.getRight()!=null)
                {
                    current=current.getRight();
                }
                else
                {
                    current.setRight(newNode);
                    break;
                }
            }
             if(data.compareTo(current.getData()) ==0){
                current.upCount();
                break;
                }
        }
}
}
```

if add method is run then value of size increases by one

if the bag is empty, create a new node and assign that is root.

Compare,

if new data is less than the data of current root node,

if left child is not null, go left sub tree

if left child is null then create a new node.

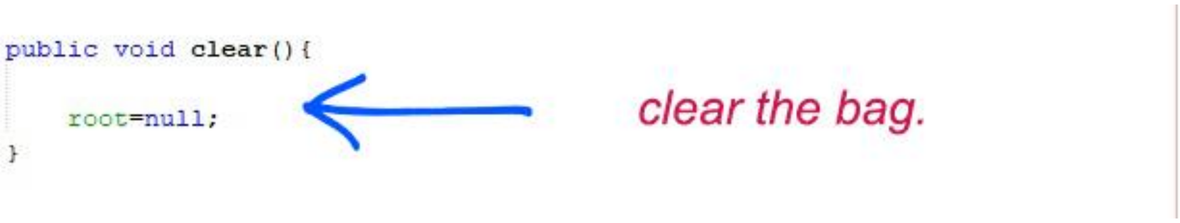
Compare

if right child is not nbull, go left sub tree

if its null then create new node

if data already exist then count is upper one value

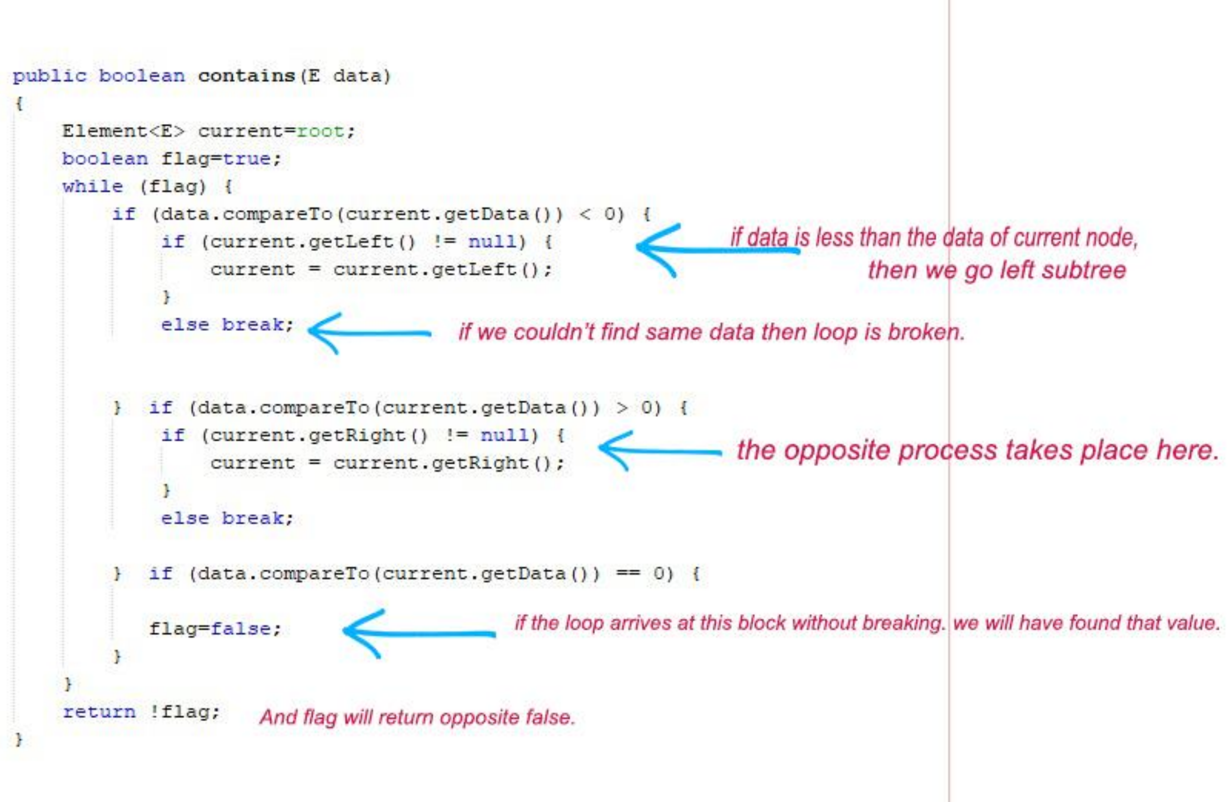
2- clear():



```
public void clear(){

    root=null;
}
```

clear the bag.

3- contains(E data):



```
public boolean contains(E data)
{
    Element<E> current=root;
    boolean flag=true;
    while (flag) {
        if (data.compareTo(current.getData()) < 0) {
            if (current.getLeft() != null) {
                current = current.getLeft();
            }
            else break;

        }  if (data.compareTo(current.getData()) > 0) {
            if (current.getRight() != null) {
                current = current.getRight();
            }
            else break;

        }  if (data.compareTo(current.getData()) == 0) {

            flag=false;
        }
    }
    return !flag;
}
```

if data is less than the data of current node, then we go left subtree

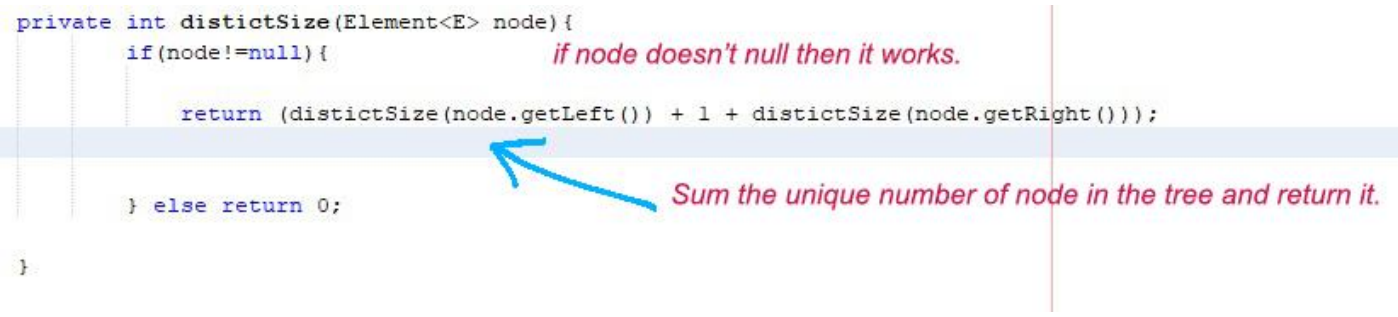
if we couldn't find same data then loop is broken.

the opposite process takes place here.

if the loop arrives at this block without breaking. we will have found that value.

And flag will return opposite false.

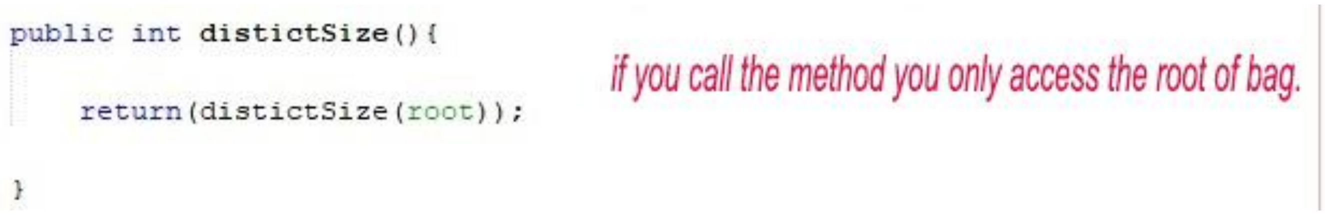
4- distictSize():



```
private int distictSize(Element<E> node){
        if(node!=null){

            return (distictSize(node.getLeft()) + 1 + distictSize(node.getRight()));


        } else return 0;

}
```

if node doesn't null then it works.

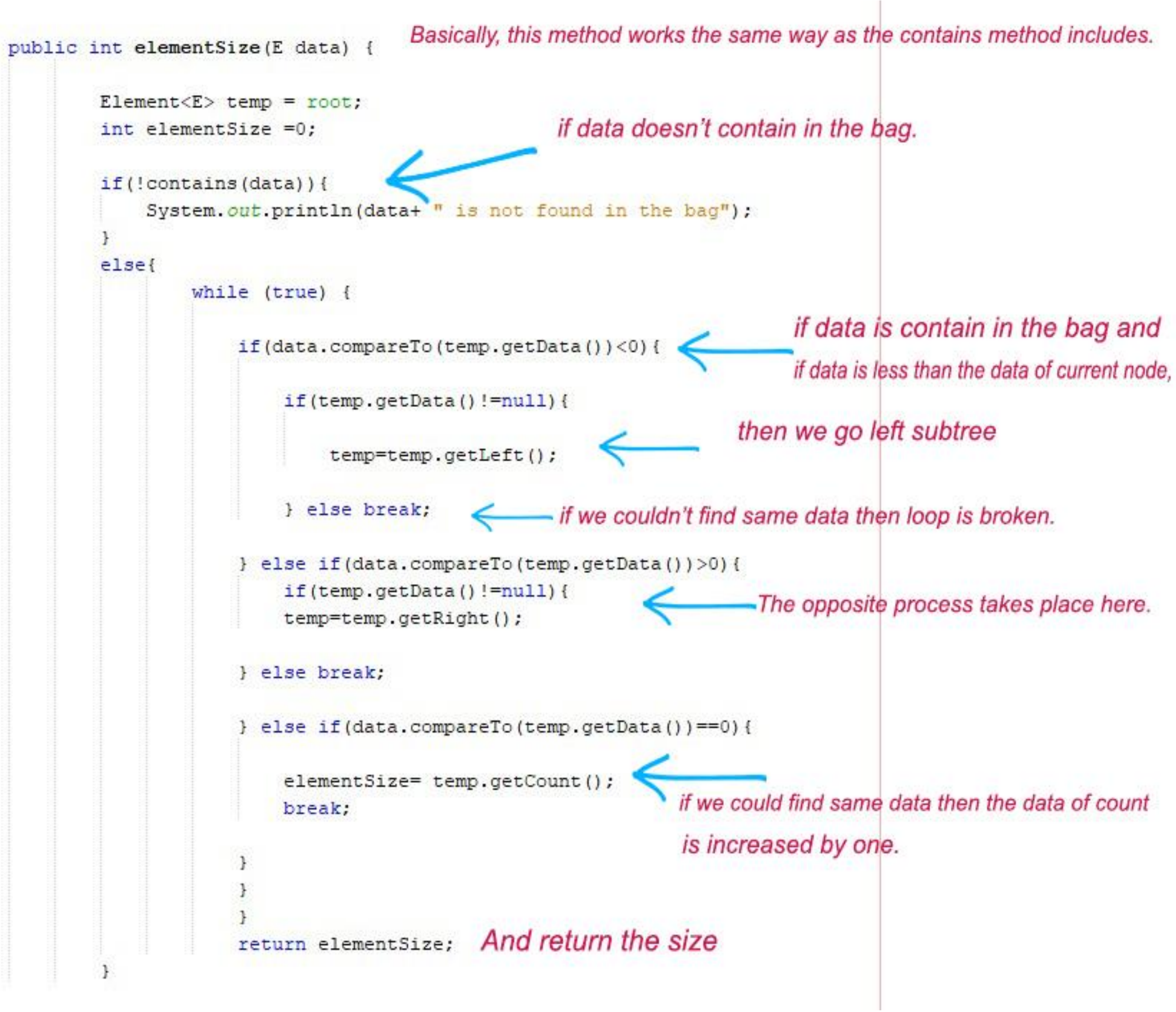
Sum the unique number of node in the tree and return it.



```
public int distictSize(){

    return(distictSize(root));

}
```

if you call the method you only access the root of bag.

5- elementSize(E data):



```
public int elementSize(E data) {

        Element<E> temp = root;
        int elementSize =0;

        if(!contains(data)){
            System.out.println(data+ " is not found in the bag");
        }
        else{
                while (true) {

                    if(data.compareTo(temp.getData())<0){

                        if(temp.getData()!=null){

                            temp=temp.getLeft();

                        } else break;

                    } else if(data.compareTo(temp.getData())>0){
                        if(temp.getData()!=null){
                        temp=temp.getRight();

                    } else break;

                    } else if(data.compareTo(temp.getData())==0){

                        elementSize= temp.getCount();
                        break;

                    }
                    }
                    }
                    return elementSize;
        }
```

Basically, this method works the same way as the contains method includes.

if data doesn't contain in the bag.

if data is contain in the bag and
if data is less than the data of current node,

then we go left subtree

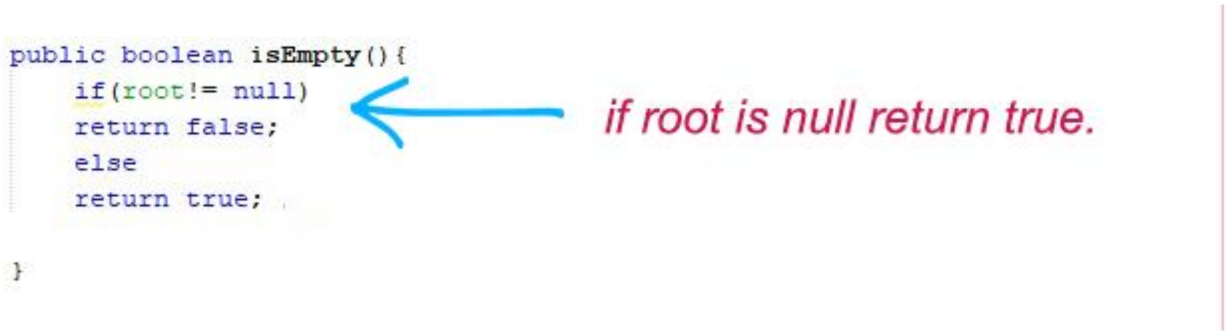
if we couldn't find same data then loop is broken.

The opposite process takes place here.

if we could find same data then the data of count
is increased by one.

And return the size

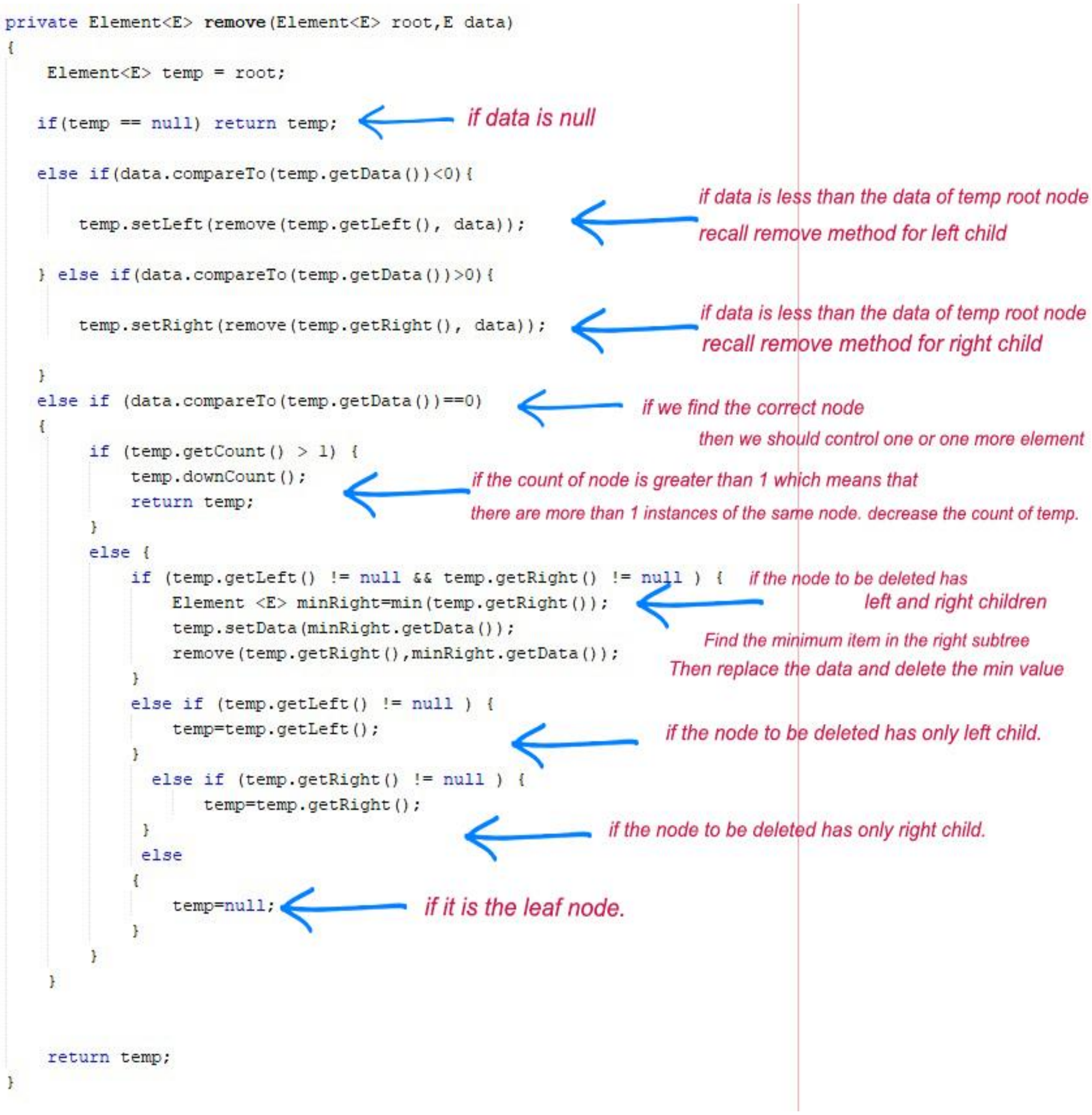
6- isEmpty():



```
public boolean isEmpty(){
    if(root!= null)
    return false;
    else
    return true;

}
```

if root is null return true.

7- remove(E data):



```
private Element<E> remove(Element<E> root,E data)
{
    Element<E> temp = root;

   if(temp == null) return temp;

   else if(data.compareTo(temp.getData())<0){

       temp.setLeft(remove(temp.getLeft(), data));

   } else if(data.compareTo(temp.getData())>0){

       temp.setRight(remove(temp.getRight(), data));

   }
   else if (data.compareTo(temp.getData())==0)
   {
        if (temp.getCount() > 1) {
            temp.downCount();
            return temp;
        }
        else {
            if (temp.getLeft() != null && temp.getRight() != null ) {
                Element <E> minRight=min(temp.getRight());
                temp.setData(minRight.getData());
                remove(temp.getRight(),minRight.getData());
            }
            else if (temp.getLeft() != null ) {
                temp=temp.getLeft();
            }
              else if (temp.getRight() != null ) {
                   temp=temp.getRight();
             }
             else
            {
                temp=null;
            }
        }
    }


    return temp;
}
```

if data is null

if data is less than the data of temp root node
recall remove method for left child

if data is less than the data of temp root node
recall remove method for right child

if we find the correct node
then we should control one or one more element

if the count of node is greater than 1 which means that
there are more than 1 instances of the same node. decrease the count of temp.

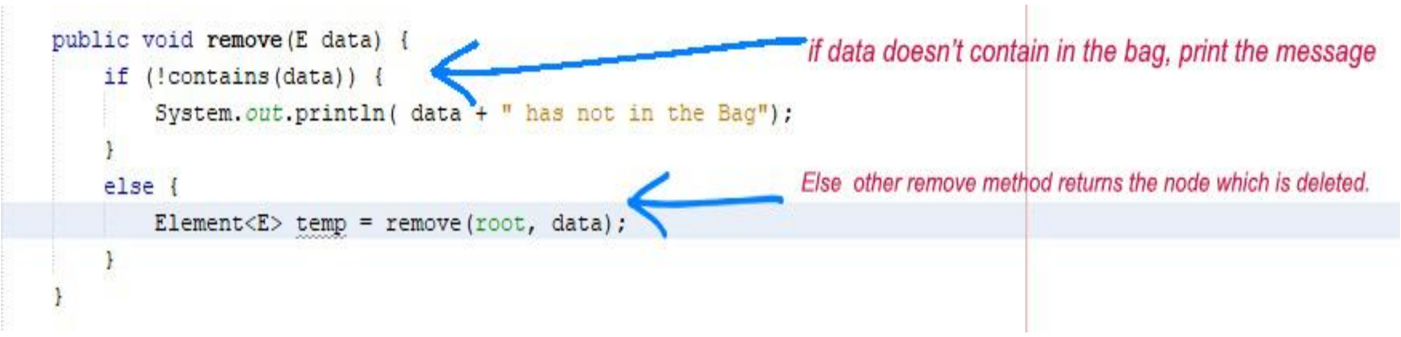
if the node to be deleted has
left and right children

Find the minimum item in the right subtree
Then replace the data and delete the min value

if the node to be deleted has only left child.

if the node to be deleted has only right child.

if it is the leaf node.

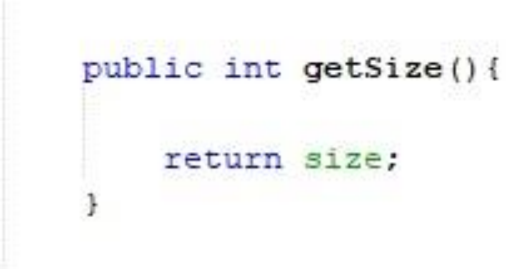


```java
public void remove(E data) {
    if (!contains(data)) {
        System.out.println( data + " has not in the Bag");
    }
    else {
        Element<E> temp = remove(root, data);
    }
}
```

if data doesn't contain in the bag, print the message

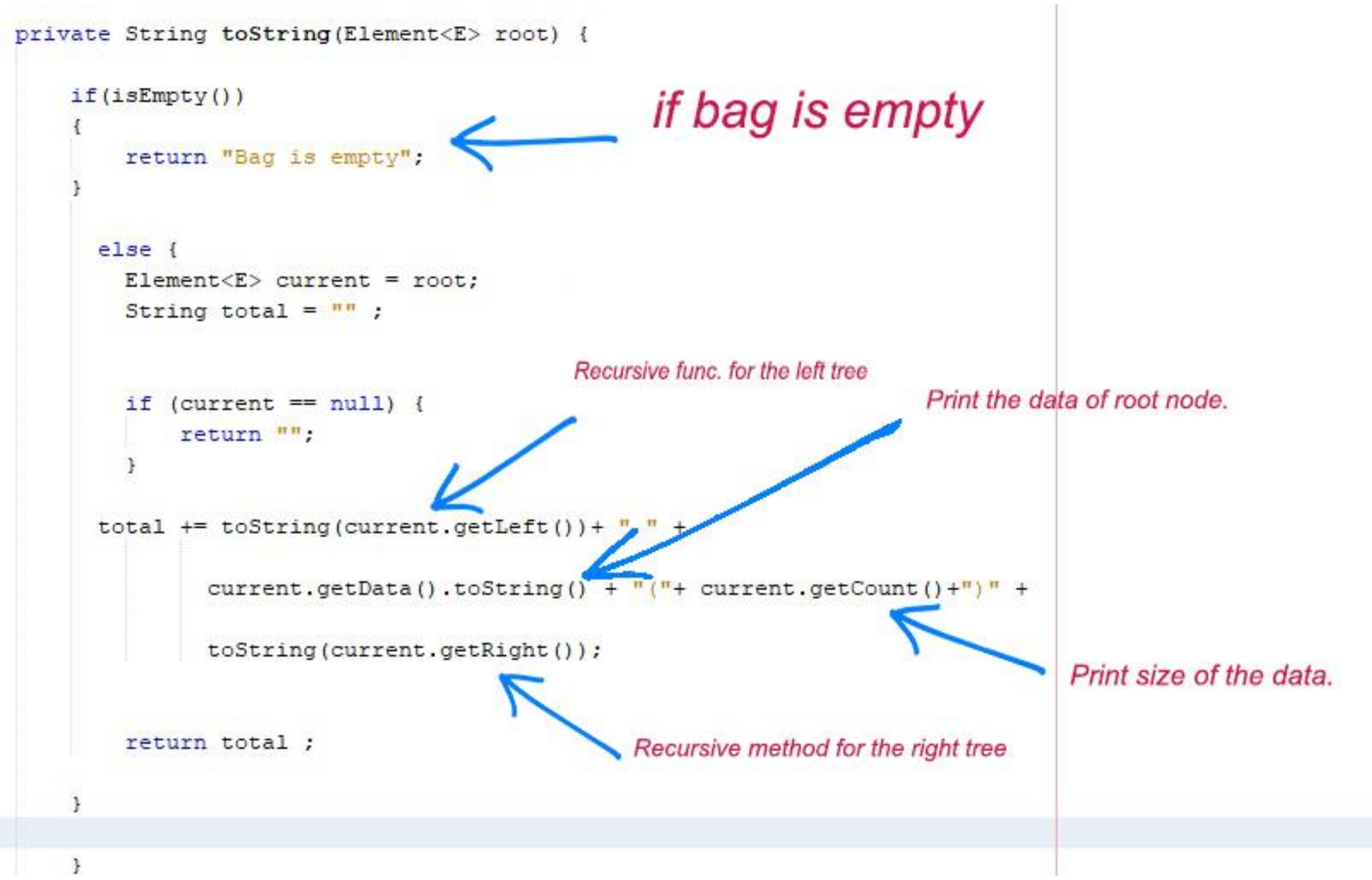
Else other remove method returns the node which is deleted.

8- size():



```java
public int getSize(){

    return size;
}
```

9- toString():



```java
private String toString(Element<E> root) {

    if(isEmpty())
    {
        return "Bag is empty";
    }

     else {
        Element<E> current = root;
        String total = "" ;

        if (current == null) {
            return "";
        }

     total += toString(current.getLeft())+ ", " +

                current.getData().toString() + "("+ current.getCount()+")" +

                toString(current.getRight());

        return total ;

    }

    }
```

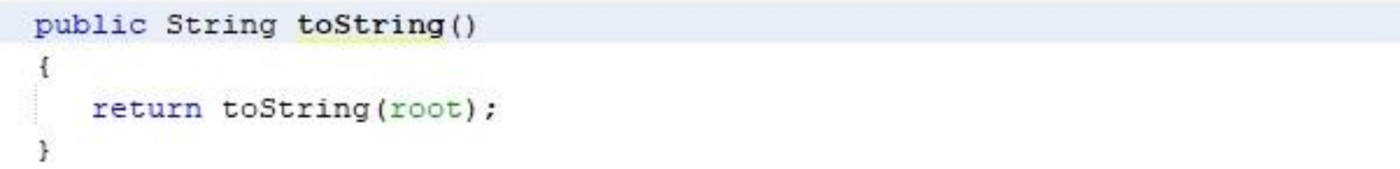
if bag is empty

Recursive func. for the left tree

Print the data of root node.

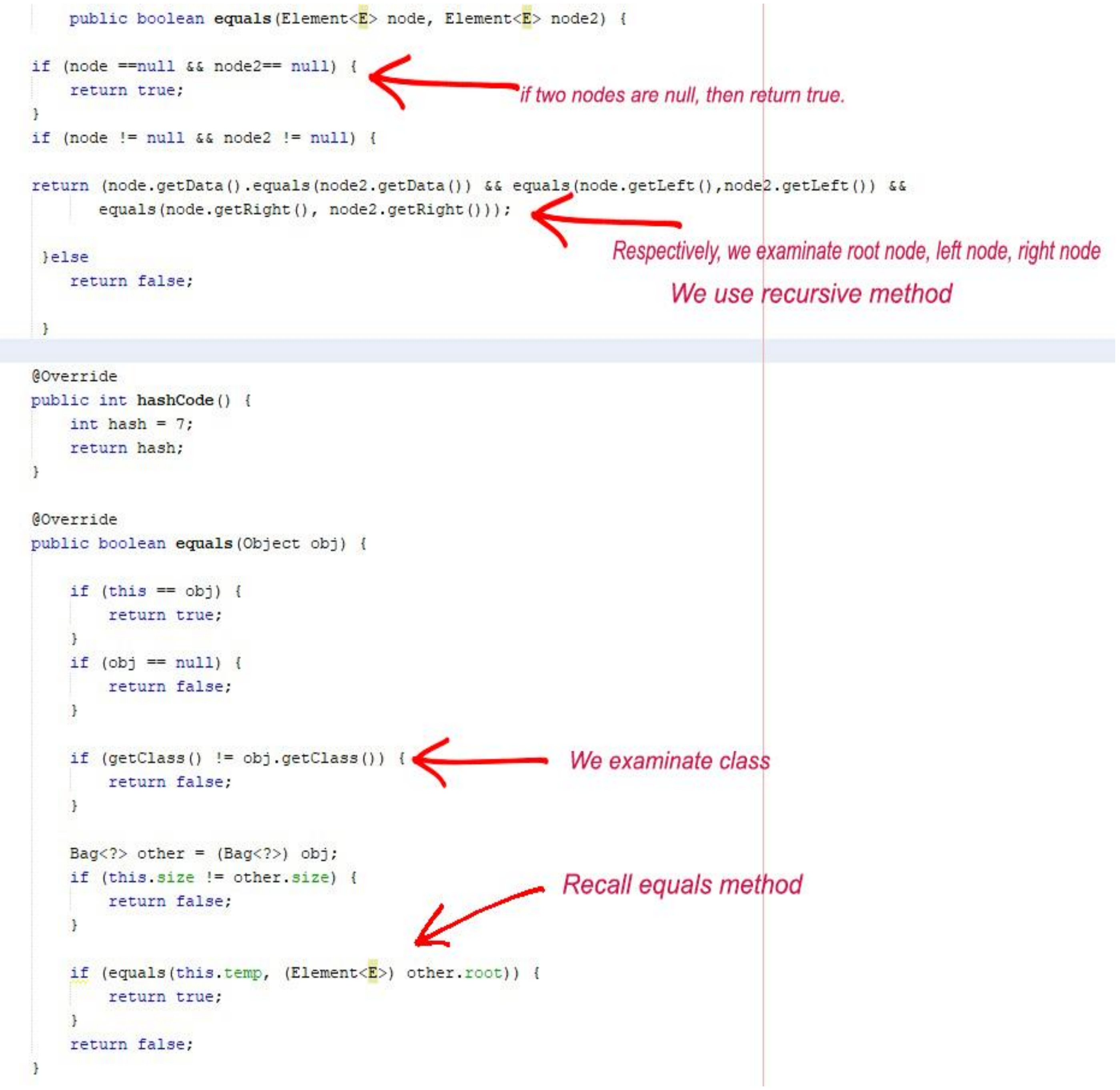
Print size of the data.

Recursive method for the right tree



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

10- equals(Object obj):



```
    public boolean equals(Element<E> node, Element<E> node2) {

if (node ==null && node2== null) {
    return true;
}
if (node != null && node2 != null) {

return (node.getData().equals(node2.getData()) && equals(node.getLeft(),node2.getLeft()) &&
       equals(node.getRight(), node2.getRight()));

 }else
    return false;

 }

@Override
public int hashCode() {
    int hash = 7;
    return hash;
}

@Override
public boolean equals(Object obj) {

    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }

    if (getClass() != obj.getClass()) {
        return false;
    }

    Bag<?> other = (Bag<?>) obj;
    if (this.size != other.size) {
        return false;
    }

    if (equals(this.temp, (Element<E>) other.root)) {
        return true;
    }
    return false;
}
```

if two nodes are null, then return true.

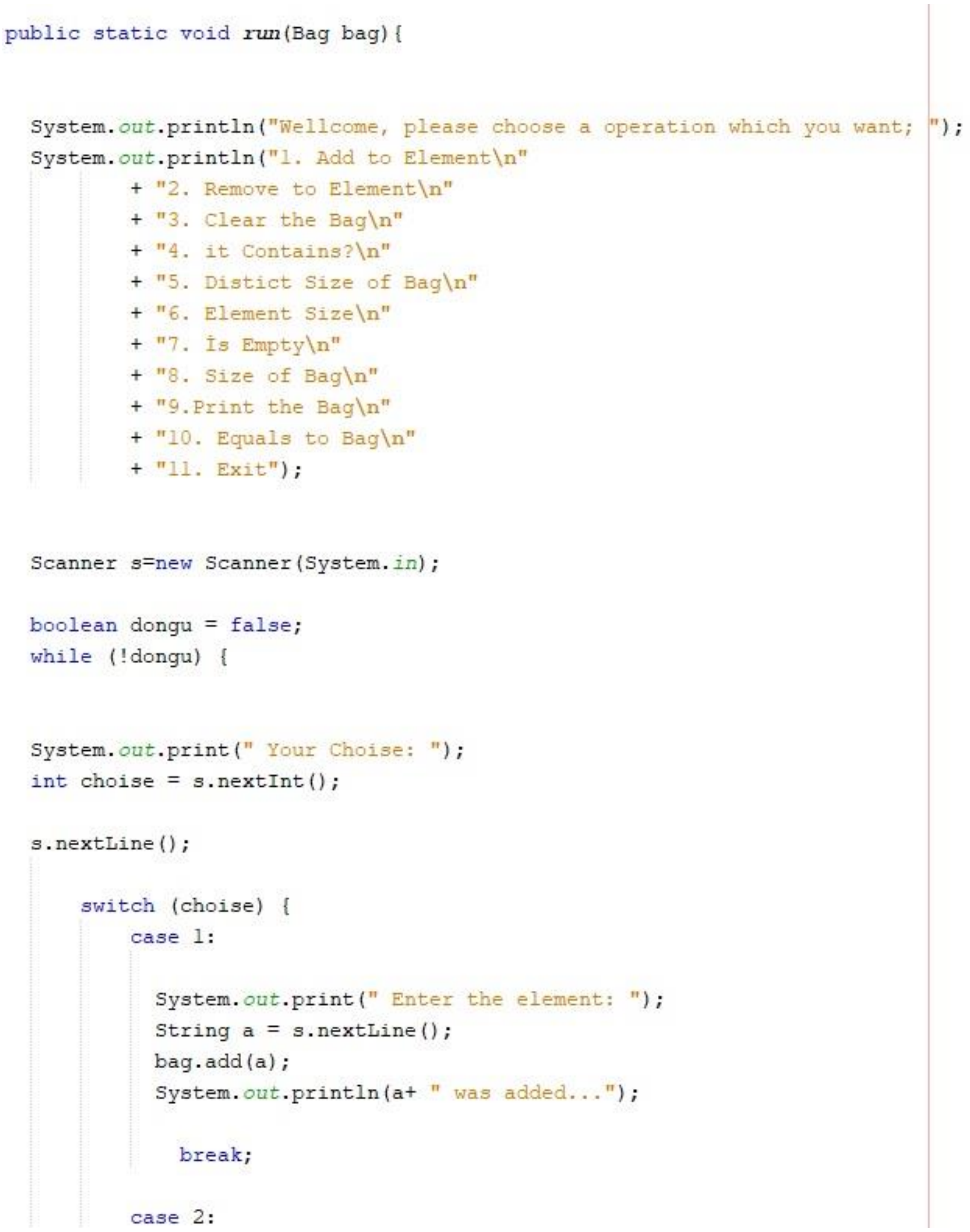
Respectively, we examinate root node, left node, right node

We use recursive method

We examinate class

Recall equals method

11- run(Bag bag): Control panel



```
public static void run(Bag bag){


  System.out.println("Wellcome, please choose a operation which you want; ");
  System.out.println("1. Add to Element\n"
          + "2. Remove to Element\n"
          + "3. Clear the Bag\n"
          + "4. it Contains?\n"
          + "5. Distict Size of Bag\n"
          + "6. Element Size\n"
          + "7. İs Empty\n"
          + "8. Size of Bag\n"
          + "9.Print the Bag\n"
          + "10. Equals to Bag\n"
          + "11. Exit");


  Scanner s=new Scanner(System.in);

  boolean dongu = false;
  while (!dongu) {


  System.out.print(" Your Choise: ");
  int choise = s.nextInt();

  s.nextLine();

      switch (choise) {
          case 1:

            System.out.print(" Enter the element: ");
            String a = s.nextLine();
            bag.add(a);
            System.out.println(a+ " was added...");

              break;

          case 2:
```