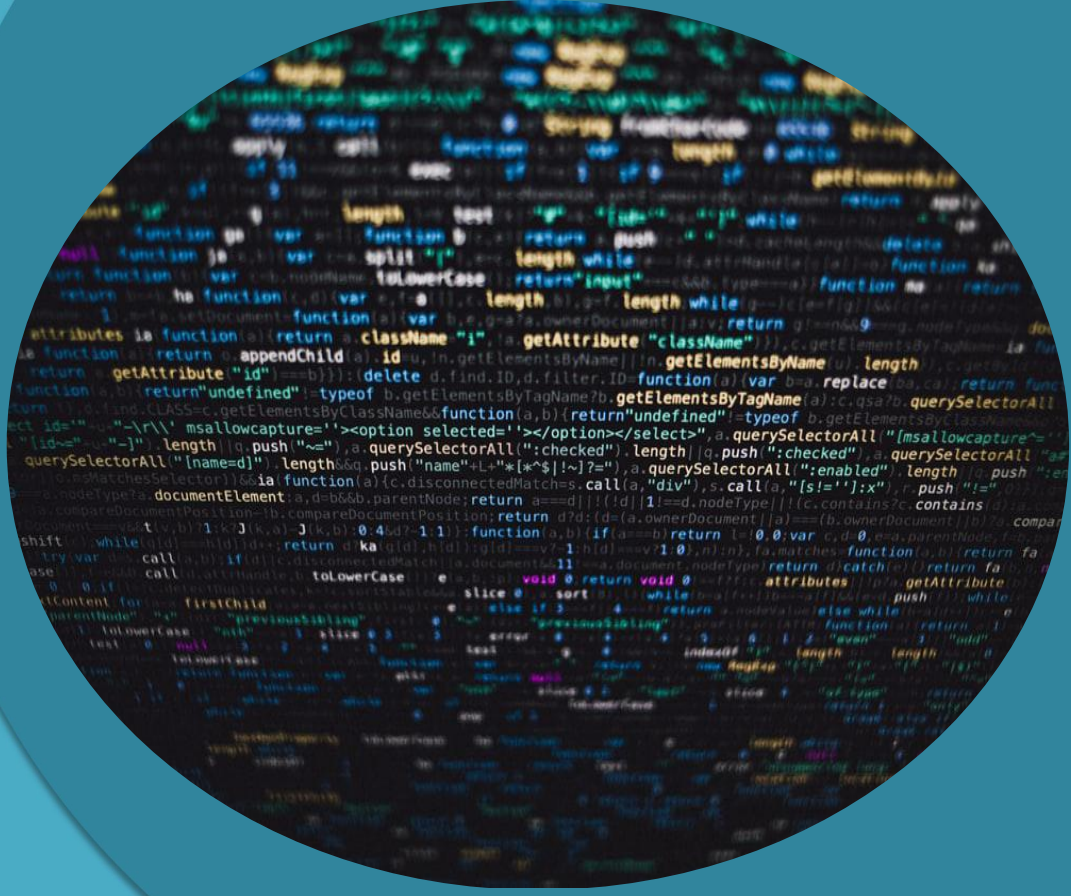


JAVA

Semester Project

180 315 019 -

Ege BULUT -

The BAG Project

Classes :

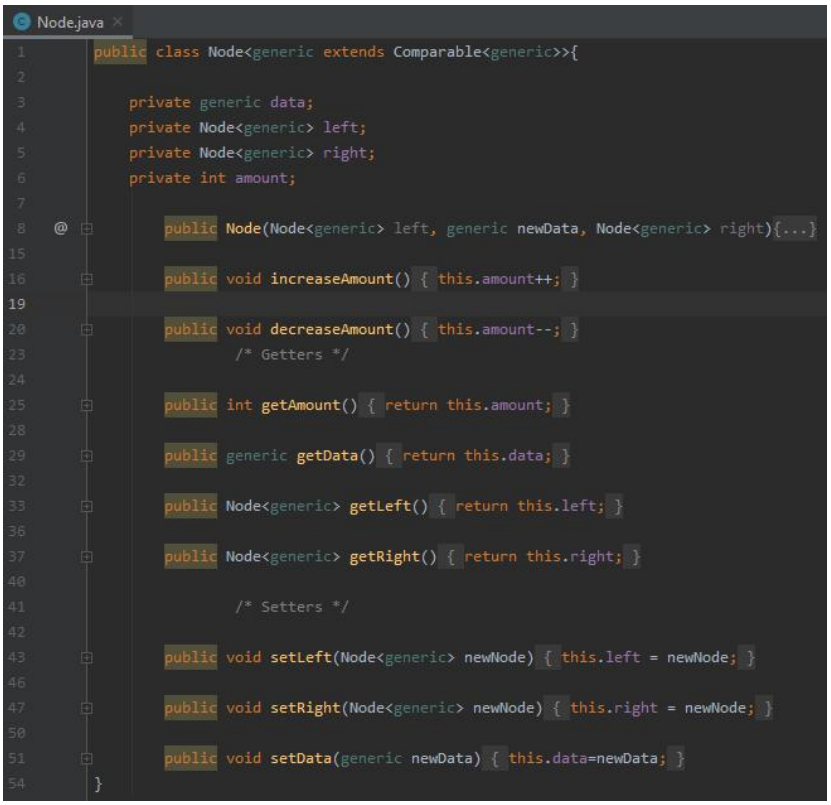
Node -

Bag -

Test -

Node.Java

There is the entire Node Class's structure:



```java
public class Node<generic extends Comparable<generic>>{

    private generic data;
    private Node<generic> left;
    private Node<generic> right;
    private int amount;

        public Node(Node<generic> left, generic newData, Node<generic> right){...}

        public void increaseAmount() { this.amount++; }

        public void decreaseAmount() { this.amount--; }
                /* Getters */

        public int getAmount() { return this.amount; }

        public generic getData() { return this.data; }

        public Node<generic> getLeft() { return this.left; }

        public Node<generic> getRight() { return this.right; }

                /* Setters */

        public void setLeft(Node<generic> newNode) { this.left = newNode; }

        public void setRight(Node<generic> newNode) { this.right = newNode; }

        public void setData(generic newData) { this.data=newData; }
}
```

First of all; on top of the code lines, there are some variables for declare the Node Class.

Secondly; we have a constructor that using for creating new Nodes.

Third; we have a 'increaseAmount' and 'decreaseAmount' method that use for increasing & decreasing the Node counter for every same Node addition or deletion.

fourth; we have some get methods and set methods for access and change private variables.

Node.java

```java
public class Node<generic extends Comparable<generic>>{

    private generic data;
    private Node<generic> left;
    private Node<generic> right;
    private int amount;
```

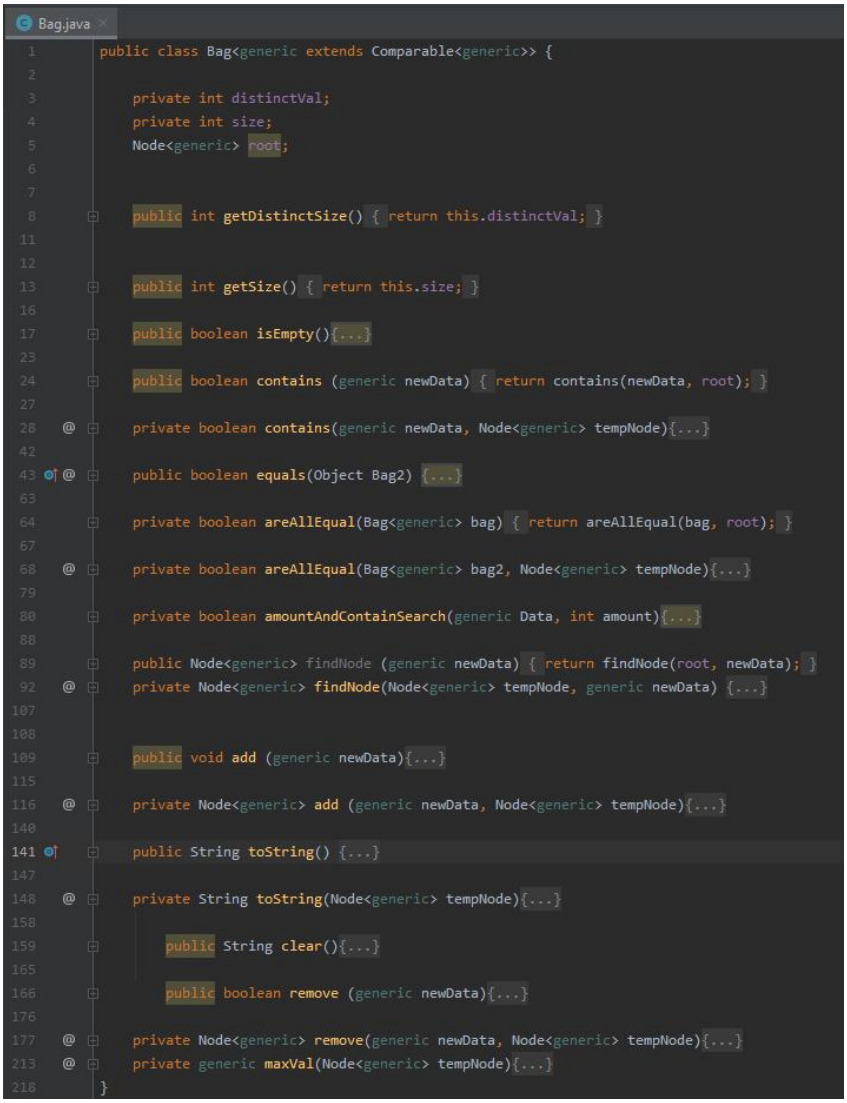
These are Node’s constructor values.

```java
    public int getAmount() { return this.amount; }

    public generic getData() { return this.data; }

    public Node<generic> getLeft() { return this.left; }

    public Node<generic> getRight() { return this.right; }

        /* Setters */

    public void setLeft(Node<generic> newNode) { this.left = newNode; }

    public void setRight(Node<generic> newNode) { this.right = newNode; }

    public void setData(generic newData) { this.data=newData; }
}
```

There are some get and set method for access / change private values.

Bag.Java

There is the entire Bag Class's structure:



First of all; there are some variables for declare the Bag Class.

Secondly; we have some get methods for access some values about bag's size and different element's amount(distinctSize).

And now let's take a closer look at the other methods.

Size | DistinctSize | Contains

```
public int getDistinctSize(){
    return this.distinctVal;
}

public int getSize(){
    return this.size;
}
```

There are two Size method for calling distinctValue and Bag's size.

```
public boolean isEmpty(){
    if(this.root == null){
        return true;
    }
    return false;
}
```

There is a method that using for control the bag and says it Empty or Not Empty.

```
public boolean contains (generic newData){
    return contains(newData, root);
}

private boolean contains(generic newData, Node<generic> tempNode){

    if(tempNode == null){
        return false;
    }
    else if (newData.compareTo(tempNode.getData())<0){
        return contains(newData, tempNode.getLeft());
    }
    else if (newData.compareTo(tempNode.getData())>0) {
        return contains(newData, tempNode.getRight());
    }

    return true;
}
```

This method controls that a Bag include a specific data.

Contains method is include a compare part for bigger or smaller comparison

Equals and sub Methods

```java
public boolean equals(Object Bag2) {
    if (Bag2 == this) {
        return true;
    }
    else {

        if (this.getClass() == Bag2.getClass()){
            Bag<generic> bag2 = (Bag<generic>) Bag2;
            if(bag2.getSize() != this.getSize()){
                return false;
            }
            if(bag2.getDistinctSize() != this.getDistinctSize()){
                return false;
            }
            else
                return this.areAllEqual( bag: this) && bag2.areAllEqual(bag2);
        }
        else return false;
    }
}

private boolean areAllEqual(Bag<generic> bag){
    return areAllEqual(bag, root);
}

private boolean areAllEqual(Bag<generic> bag2, Node<generic> tempNode){
    if(tempNode == null){
        return true;
    }

    if(bag2.amountAndContainSearch(tempNode.getData(), tempNode.getAmount())){
        return areAllEqual(bag2, tempNode.getLeft()) && areAllEqual(bag2, tempNode.getRight());
    }
    else
        return false;
}
```

```java
    private boolean amountAndContainSearch(generic Data, int amount){
        Node<generic> searchNode = findNode(root, Data);
        if(searchNode.getAmount() != amount){
            return false;
        }
        else
            return true;
    }

    public Node<generic> findNode (generic newData){
        return findNode(root, newData);
    }
    private Node<generic> findNode(Node<generic> tempNode, generic newData) {

        if (tempNode == null){
            return null;
        }

        if (newData.compareTo(tempNode.getData()) > 0) {
            return findNode(tempNode.getRight(), newData);
        }
        else if  (newData.compareTo(tempNode.getData()) < 0){
            return findNode(tempNode.getLeft(), newData);
        }
        else
            return tempNode;
    }
```

First I compare both object. If the Objects are same, method returns ‘true’.

If they are not equal as Object. I compare them Classes, if they are in the same Class, method compare their size’s and distinctsize’s, if they are not equal returns false. But if these are equal too, this time equals method calls “areAllEqual”method for control their Data. ‘areAllEqual’ method is simply for compare Data and data’s values in order with ‘amountAndContainSearch’ method’s help.

If all data is equal with the other bag’s data, than comparison ends with a ‘true’ return. Otherwise return ‘false’.

‘FindNode’ method is using for find a Data’s node.

Add Method

```
public void add (generic newData){
    if(newData!=null) {
        this.size++;
        root = add(newData, root);
    }
}

private Node<generic> add (generic newData, Node<generic> tempNode){
    if(newData == null){
        return null;
    }
  else
    if(tempNode==null){
        Node<generic> newNode = new Node<~> ( left: null, newData,  right: null);
        distinctVal++;
        newNode.increaseAmount();
        return newNode;
    }

    if (newData.compareTo(tempNode.getData())==0){
        tempNode.increaseAmount();
    }
    else if (newData.compareTo(tempNode.getData())<0){
        tempNode.setLeft(add(newData, tempNode.getLeft()));
    }
    else if (newData.compareTo(tempNode.getData())>0){
        tempNode.setRight(add(newData, tempNode.getRight()));
    }

    return tempNode;
}
```

This method's first job is check the data if its null or not. If the data isn't null; compare it with the root:

If the newData is bigger than root, method calls root's Right Node to compare with newData.

If the newData is smaller than root, method calls root's Left Node to compare with newData.

If newData and tempNode are equal, method increases tempNode's amount.

If method reaches the leaf of tree (with null check) add the newData here and increases them amount and distinctValue.

Than return the all values that before the newData's Node, back to their Parents.

toString Method

```
public String toString() {
    String dataArray = toString(root);
    if(dataArray.isEmpty())
        return "empty";
     return dataArray;
    }

private String toString(Node<generic> tempNode){
            if(tempNode == null){
                return "";
            }

            String bag = "";

            bag += toString (tempNode.getLeft());
            bag += tempNode.getData()+"["+tempNode.getAmount()+"]   ";
            bag += toString(tempNode.getRight());
        return bag;
    }
```

In here, we create an Array that named ‘dataArray’ for collect the all data together .

In private toString method, first method check the root was empty or not. If it was empty method return an empty String.

Otherwise, if the root was not empty, method creates an empty String that named ‘bag’. Than fill it recursively with values via ‘inorder traversal’.

Than return that-bag- String.

Clear Method

```
public String clear(){
this.root = null;
size=0;
distinctVal=0;
return "Cleared sucssesfully!";
}
```

In this method basically set the trees rootNode null, and reset the bag's size and distinctValue. Than return a success message to the screen.

Remove & maxVal Method

```
private Node<generic> remove(generic newData, Node<generic> tempNode){

    if (tempNode == null) {
        return null;
    }

    if (newData.compareTo(tempNode.getData()) < 0) {
        tempNode.setLeft(remove(newData, tempNode.getLeft()));
    }
    else if  (newData.compareTo(tempNode.getData()) > 0){
        tempNode.setRight(remove(newData, tempNode.getRight()));
    }
    else {
        if(tempNode.getAmount()>1) {
            tempNode.decreaseAmount();
        }
        else{
            distinctVal--;
            if (tempNode.getRight() == null && tempNode.getLeft() == null){
                return null;
            }
            else if(tempNode.getRight() == null){
                return tempNode.getLeft();
            }
            else if (tempNode.getLeft() == null){
                return tempNode.getRight();
            }
            else{
                generic maxVal = maxVal(tempNode.getLeft());
                tempNode.setData(maxVal);
                tempNode.setLeft(remove(maxVal, tempNode.getLeft()));
            }
        }
    }
    return tempNode;
}
private generic maxVal(Node<generic> tempNode){
    if(tempNode.getRight()!=null)
        return maxVal(tempNode.getRight());
    return tempNode.getData();
}
```

First things first in this method controls the Node, if the Node was null, it returns null.

Than if it wasn't null. The method compare Node's data with the data that we want to remove.

If the data is smaller than the Node's data, method calls 'setLeft' method recursively.

If the data is bigger than the Node's data, method calls 'setRight' method recursively.

If the data and the Node's data are both equal, method controls the amount of Node. If the amount is bigger than 1, method calls 'decreaseAmount' method. Otherwise method decreases the distinctValue and controls the children of Node. If there was a no Child, return null. If there was a only right child, it returns Node's Right for don't lose the data. If there was a only left child, it returns Node's Left for don't lose the data. But if there was a two children the method calls 'maxVal' method that find and returns left child's rightmost child. And than equalize it with the tempNode.

Test Class

Test.java

```java
public class Test {
    public static void main(String[] args) {
        Bag<String> bag =  new Bag();
        bag.add("C#");
        bag.add("Java");
        bag.add("C++");
        bag.add("C#");
        bag.add("java");

        Bag<String> bag2 = new Bag();
        bag2.add("java");
        bag2.add("Java");
        bag2.add("C#");

        Bag<String> bag3 = new Bag();
        bag3.add(null);

        Bag<String> bag4 = new Bag();
        bag4.add("Assembly");
        bag4.add("Kotlin");

        Bag<String> bag5 = new Bag();
        bag5.add("java");
        bag5.add("Java");

     //Bags and sizes//
            System.out.println("");
            System.out.println("bag1 :"+bag);
            System.out.println("Bag Size : "+bag.getSize());
            System.out.println("1 Distinct value : "+bag.getDistinctSize());
            System.out.println("bag2 :"+bag2);
            System.out.println("Bag Size : "+bag2.getSize());
            System.out.println("2 Distinct value : "+bag2.getDistinctSize());
            System.out.println("bag3 :"+bag3);
            System.out.println("Bag Size : "+bag3.getSize());
            System.out.println("3 Distinct value : "+bag3.getDistinctSize());
            System.out.println("bag4 :"+bag4);
            System.out.println("Bag Size : "+bag4.getSize());
            System.out.println("4 Distinct value : "+bag4.getDistinctSize());
            System.out.println("bag5 :"+bag5);
            System.out.println("Bag Size : "+bag5.getSize());
            System.out.println("5 Distinct value : "+bag5.getDistinctSize());
    //isEmpties
            System.out.println("__________");
            System.out.println("bag1 isEmpty? : " + bag.isEmpty());
            System.out.println("bag2 isEmpty? : " + bag2.isEmpty());
            System.out.println("bag3 isEmpty? : " + bag3.isEmpty());
            System.out.println("bag4 isEmpty? : " + bag4.isEmpty());
            System.out.println("bag5 isEmpty? : " + bag5.isEmpty());
    //Contains
            System.out.println("__________");
            System.out.println("bag1 contains 'C#' : "+bag.contains("C#"));
            System.out.println("bag2 contains 'C#' : "+bag2.contains("C#"));
            System.out.println("bag3 contains 'C#' : "+bag3.contains("C#"));
            System.out.println("bag4 contains 'C#' : "+bag4.contains("C#"));
            System.out.println("bag5 contains 'C#' : "+bag5.contains("C#"));
    //Removes
            System.out.println("__________");
            System.out.println("bag1 remove 'C#' : "+bag.remove( newData: "C#"));
            System.out.println("bag2 remove 'C#' : "+bag2.remove( newData: "C#"));
            System.out.println("bag3 remove 'C#' : "+bag3.remove( newData: "C#"));
            System.out.println("bag4 remove 'C#' : "+bag4.remove( newData: "C#"));
            System.out.println("bag5 remove 'C#' : "+bag5.remove( newData: "C#"));
    //Bags and sizes// new
            System.out.println("__________");
            System.out.println("bag1 :"+bag);
            System.out.println("Bag Size : "+bag.getSize());
            System.out.println("1 Distinct value : "+bag.getDistinctSize());
            System.out.println("bag2 :"+bag2);
            System.out.println("Bag Size : "+bag2.getSize());
            System.out.println("2 Distinct value : "+bag2.getDistinctSize());
            System.out.println("bag3 :"+bag3);
            System.out.println("Bag Size : "+bag3.getSize());
            System.out.println("3 Distinct value : "+bag3.getDistinctSize());
            System.out.println("bag4 :"+bag4);
            System.out.println("Bag Size : "+bag4.getSize());
            System.out.println("4 Distinct value : "+bag4.getDistinctSize());
            System.out.println("bag5 :"+bag5);
            System.out.println("Bag Size : "+bag5.getSize());
            System.out.println("5 Distinct value : "+bag5.getDistinctSize());
    //Equals
            System.out.println("__________");
            System.out.println("equal bag2 to bag5 : "+ bag2.equals(bag5));
            System.out.println("equal bag5 to bag2 : "+ bag5.equals(bag2));
            System.out.println("equal bag4 to bag5 : "+ bag4.equals(bag5));
            System.out.println("equal bag3 to bag4 : "+ bag3.equals(bag4));
    //Clears
            System.out.println("__________");
            System.out.println("clear bag1 :"+bag.clear());
            System.out.println("clear bag2 :"+bag2.clear());
            System.out.println("clear bag4 :"+bag4.clear());
            System.out.println("clear bag5 :"+bag5.clear());
    //Bags and sizes// new
            System.out.println("__________");
            System.out.println("bag1 :"+bag);
            System.out.println("Bag Size : "+bag.getSize());
            System.out.println("1 Distinct value : "+bag.getDistinctSize());
            System.out.println("bag2 :"+bag2);
            System.out.println("Bag Size : "+bag2.getSize());
            System.out.println("2 Distinct value : "+bag2.getDistinctSize());
            System.out.println("bag3 :"+bag3);
            System.out.println("Bag Size : "+bag3.getSize());
            System.out.println("3 Distinct value : "+bag3.getDistinctSize());
            System.out.println("bag4 :"+bag4);
            System.out.println("Bag Size : "+bag4.getSize());
            System.out.println("4 Distinct value : "+bag4.getDistinctSize());
            System.out.println("bag5 :"+bag5);
            System.out.println("Bag Size : "+bag5.getSize());
            System.out.println("5 Distinct value : "+bag5.getDistinctSize());
    //Equals
            System.out.println("__________");
            System.out.println("equal bag1 to bag3 : "+ bag.equals(bag3));
            System.out.println("equal bag5 to bag2 : "+ bag5.equals(bag2));
            System.out.println("equal bag3 to bag1 : "+ bag3.equals(bag));
    // END of the Program.
            System.out.println("__________");
        }
    }
```

Output of test Class

```
bag1 :C#[2]  C++[1]  Java[1]  java[1]
Bag Size : 5
1 Distinct value : 4
bag2 :C#[1]  Java[1]  java[1]
Bag Size : 3
2 Distinct value : 3
bag3 :empty
Bag Size : 0
3 Distinct value : 0
bag4 :Assembly[1]  Kotlin[1]
Bag Size : 2
4 Distinct value : 2
bag5 :Java[1]  java[1]
Bag Size : 2
5 Distinct value : 2
__________
bag1 isEmpty? : false
bag2 isEmpty? : false
bag3 isEmpty? : true
bag4 isEmpty? : false
bag5 isEmpty? : false
__________
bag1 contains 'C#' : true
bag2 contains 'C#' : true
bag3 contains 'C#' : false
bag4 contains 'C#' : false
bag5 contains 'C#' : false
__________
bag1 remove 'C#' : true
bag2 remove 'C#' : true
bag3 remove 'C#' : false
bag4 remove 'C#' : false
bag5 remove 'C#' : false
__________
bag1 :C#[1]  C++[1]  Java[1]  java[1]
Bag Size : 4
1 Distinct value : 4
bag2 :Java[1]  java[1]
Bag Size : 2
2 Distinct value : 2
bag3 :empty
Bag Size : 0
3 Distinct value : 0
bag4 :Assembly[1]  Kotlin[1]
Bag Size : 2
4 Distinct value : 2
bag5 :Java[1]  java[1]
Bag Size : 2
5 Distinct value : 2
```

```
__________
bag1 :C#[1]  C++[1]  Java[1]  java[1]
Bag Size : 4
1 Distinct value : 4
bag2 :Java[1]  java[1]
Bag Size : 2
2 Distinct value : 2
bag3 :empty
Bag Size : 0
3 Distinct value : 0
bag4 :Assembly[1]  Kotlin[1]
Bag Size : 2
4 Distinct value : 2
bag5 :Java[1]  java[1]
Bag Size : 2
5 Distinct value : 2
__________
equal bag2 to bag5 : true
equal bag5 to bag2 : true
equal bag4 to bag5 : true
equal bag3 to bag4 : false
__________
clear bag1 :Cleared sucssesfully!
clear bag2 :Cleared sucssesfully!
clear bag4 :Cleared sucssesfully!
clear bag5 :Cleared sucssesfully!
__________
bag1 :empty
Bag Size : 0
1 Distinct value : 0
bag2 :empty
Bag Size : 0
2 Distinct value : 0
bag3 :empty
Bag Size : 0
3 Distinct value : 0
bag4 :empty
Bag Size : 0
4 Distinct value : 0
bag5 :empty
Bag Size : 0
5 Distinct value : 0
__________
equal bag1 to bag3 : true
equal bag5 to bag2 : true
equal bag3 to bag1 : true
__________

Process finished with exit code 0
```