REPORT

NAME: Ahmed ali elsayed saber khalil

ID: 7

NAME: Islam mostafa Abdelaziz gaber

ID: 13

IMPORTANT DECISIONS:

- We implemented the red black tree as a binary search tree with it’s attributes such as: key , left , right , parent , value.

- We added one attribute per node which is : color ; which is either Red or Black.

- As we insert or delete a node in the tree we fixup it by rotation methods.

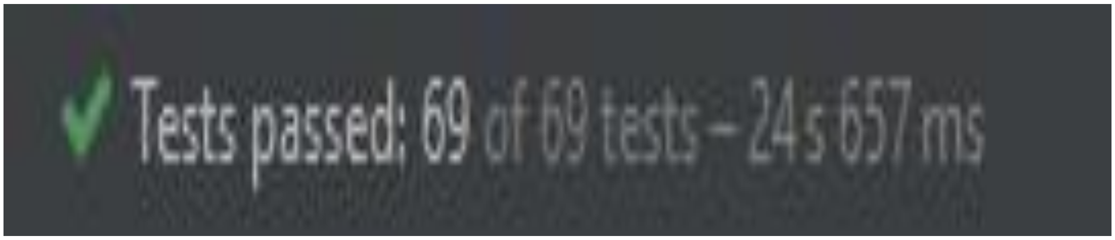
- We implemented an additional class “RBtreePrinter” has a method to print the red black tree.

- We applied all the properties of the Red-Black trees such as:
  - Every node is either red or black.
  - The root is black.
  - Every leaf is black.
  - If a node is red, then both it's children are black.
  - For each node, all paths from the node to descendant leaves contain the same number of black nodes.

- We implemented the treeMap as it is based on Red-Black tree.

The time analysis of implemented methods:

- The insertion has O(log n) time.
- Also , the deletion has O(log n) time.
- We passed all the tests of this lab and this is the time of them.

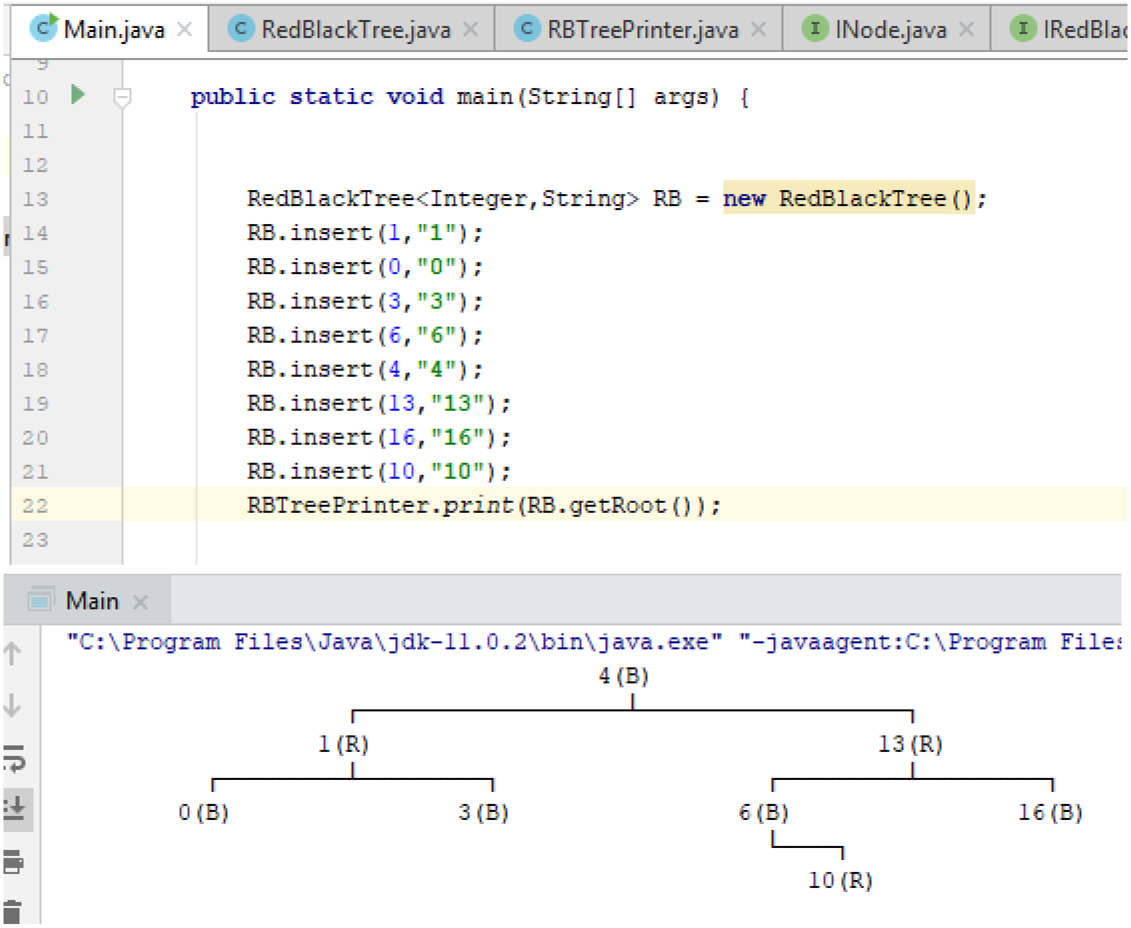


✓ Tests passed: 69 of 69 tests – 24 s 657 ms

Sample Runs:

Example 1:

Here we inserted some nodes in the red black tree , then we print the tree.

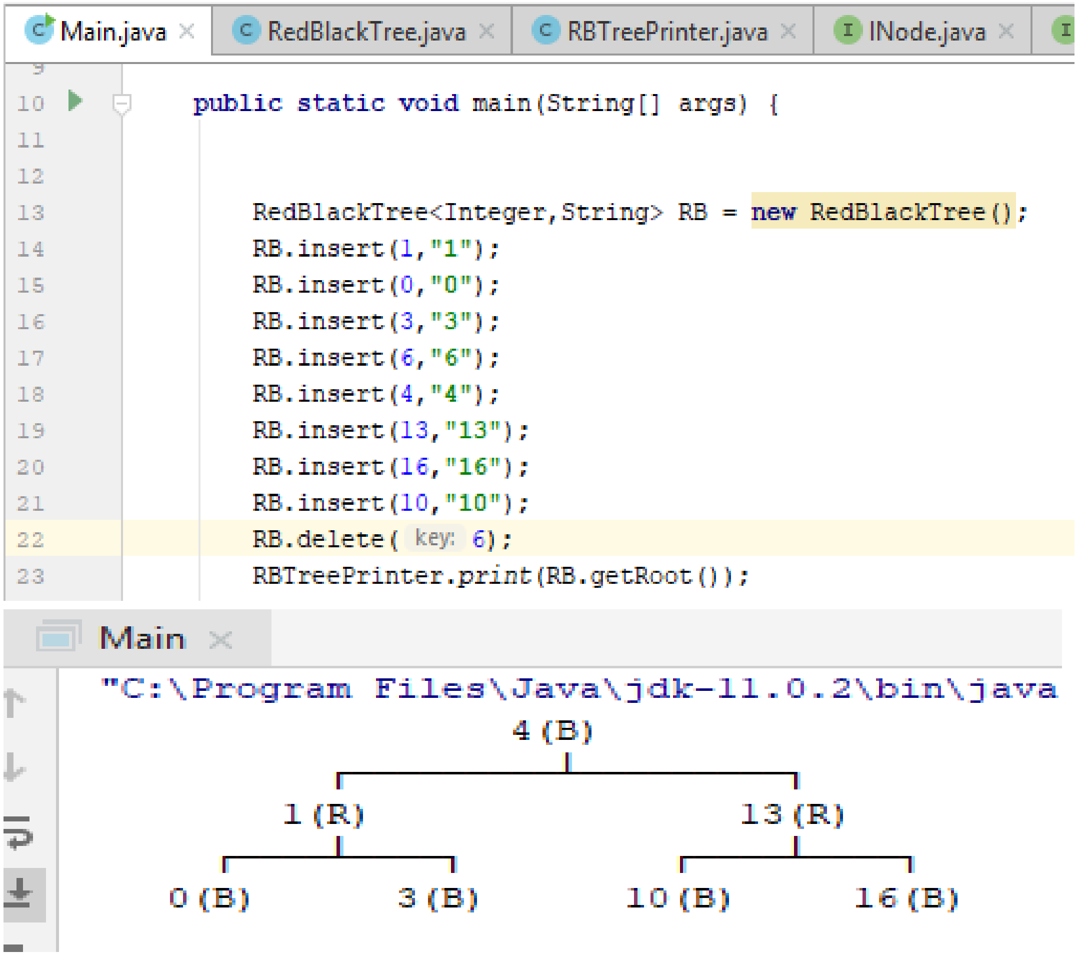


```java
    public static void main(String[] args) {


        RedBlackTree<Integer,String> RB = new RedBlackTree();
        RB.insert(1,"1");
        RB.insert(0,"0");
        RB.insert(3,"3");
        RB.insert(6,"6");
        RB.insert(4,"4");
        RB.insert(13,"13");
        RB.insert(16,"16");
        RB.insert(10,"10");
        RBTreePrinter.print(RB.getRoot());

```

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-javaagent:C:\Program Files
                                      4 (B)
                    ┌───────────────────┴───────────────────┐
                  1 (R)                                   13 (R)
          ┌─────────┴─────────┐                   ┌─────────┴─────────┐
        0 (B)               3 (B)               6 (B)               16 (B)
                                                  └────┐
                                                     10 (R)
```

- ## Example 2:

Here we inserted some nodes ,then delete a node with the key “6” ,then print the tree.

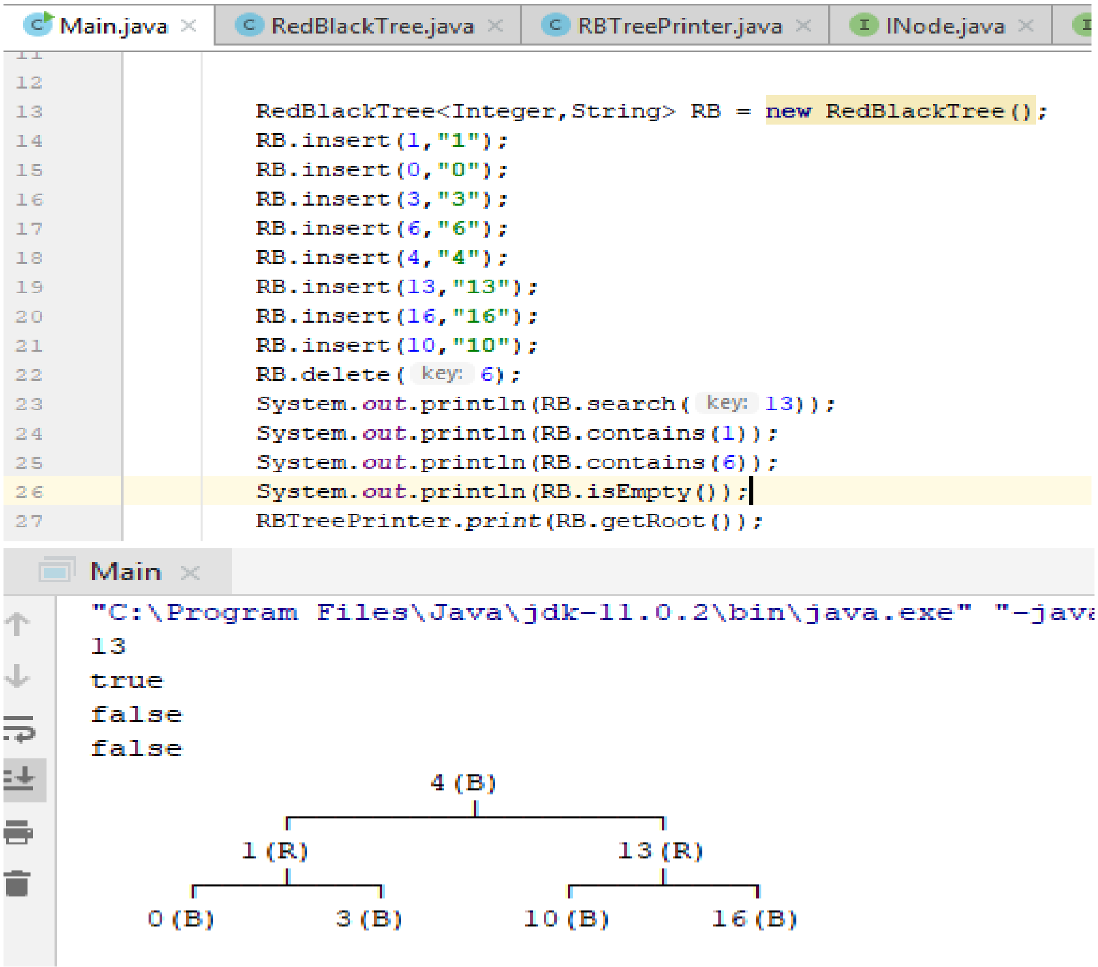


```
    public static void main(String[] args) {


        RedBlackTree<Integer,String> RB = new RedBlackTree();
        RB.insert(1,"1");
        RB.insert(0,"0");
        RB.insert(3,"3");
        RB.insert(6,"6");
        RB.insert(4,"4");
        RB.insert(13,"13");
        RB.insert(16,"16");
        RB.insert(10,"10");
        RB.delete( key: 6);
        RBTreePrinter.print(RB.getRoot());
```

```
"C:\Program Files\Java\jdk-11.0.2\bin\java
                  4 (B)
          ┌─────────┴─────────┐
        1 (R)              13 (R)
     ┌────┴────┐         ┌────┴────┐
   0 (B)     3 (B)    10 (B)    16 (B)
```

➢ Example 3:

Here we inserted some nodes , then delete an node with key “6”, then search the node with key “13” which return it’s value = 13 , then check if the tree contains the key “1” which is true , then check if the tree contains the key “6” that we deleted before which is false , then we check if the tree is empty which is false , then we print it.

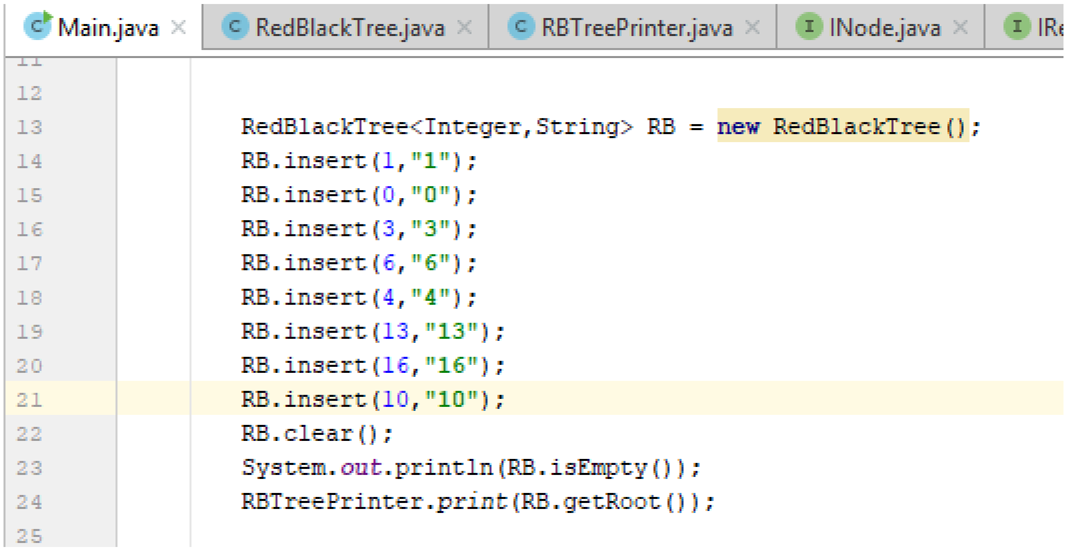


```
        RedBlackTree<Integer,String> RB = new RedBlackTree();
        RB.insert(1,"1");
        RB.insert(0,"0");
        RB.insert(3,"3");
        RB.insert(6,"6");
        RB.insert(4,"4");
        RB.insert(13,"13");
        RB.insert(16,"16");
        RB.insert(10,"10");
        RB.delete( key: 6);
        System.out.println(RB.search( key: 13));
        System.out.println(RB.contains(1));
        System.out.println(RB.contains(6));
        System.out.println(RB.isEmpty());
        RBTreePrinter.print(RB.getRoot());
```

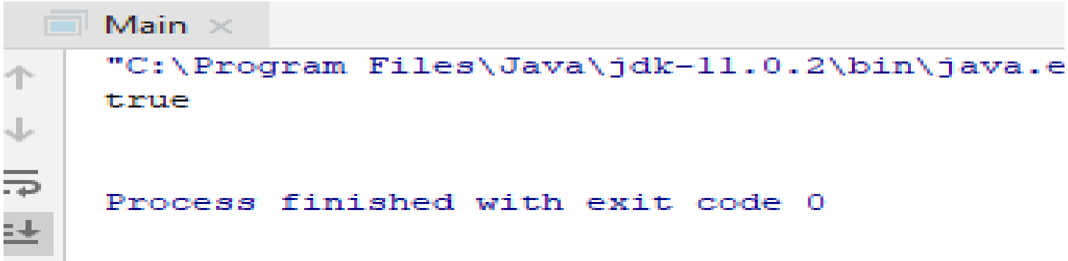
```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-java
13
true
false
false
                  4(B)
                     |
          ┌──────────┴─────────┐
        1(R)                13(R)
          |                    |
     ┌────┴────┐          ┌────┴────┐
   0(B)      3(B)       10(B)      16(B)
```

➤ Example 4:

Here we inserted some nodes , then we clear the tree , then we check if the tree is empty which is true , then we print it which is empty.



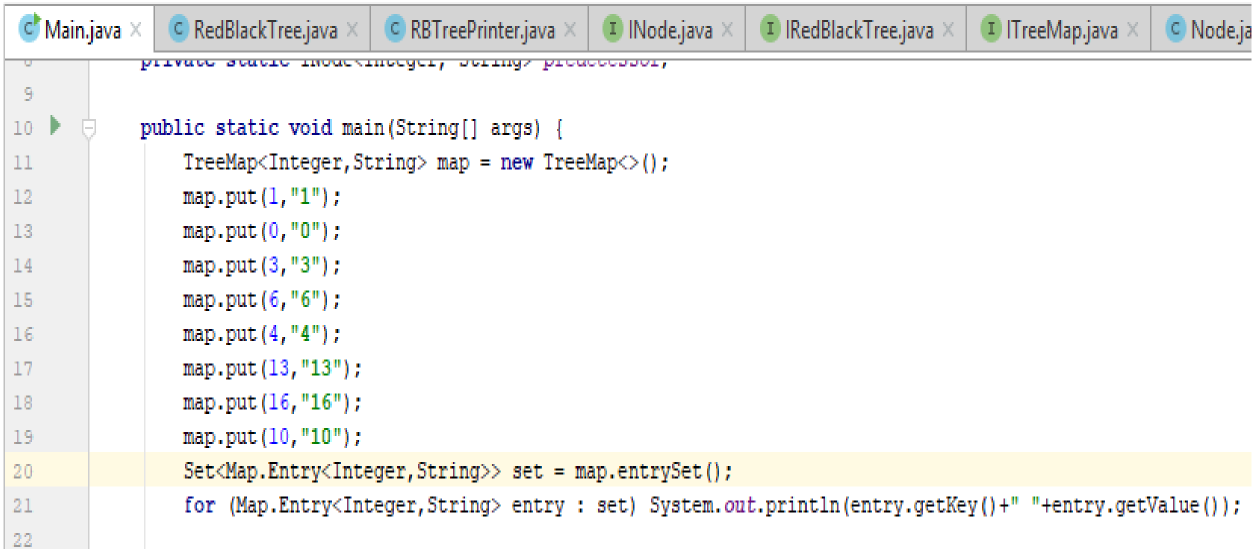
```
        RedBlackTree<Integer,String> RB = new RedBlackTree();
        RB.insert(1,"1");
        RB.insert(0,"0");
        RB.insert(3,"3");
        RB.insert(6,"6");
        RB.insert(4,"4");
        RB.insert(13,"13");
        RB.insert(16,"16");
        RB.insert(10,"10");
        RB.clear();
        System.out.println(RB.isEmpty());
        RBTreePrinter.print(RB.getRoot());
```



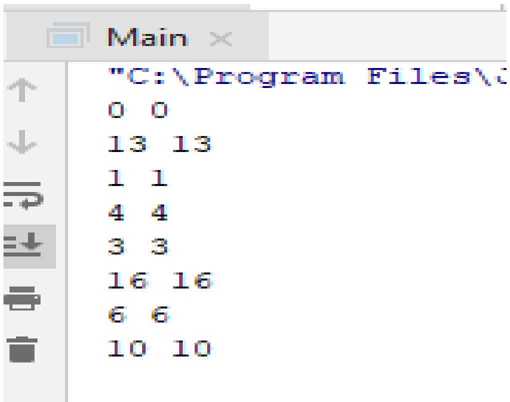
```
"C:\Program Files\Java\jdk-11.0.2\bin\java.e
true


Process finished with exit code 0
```

➢ Example 5:

Here we put nodes in the treeMap , then we called “entrySet” method to return the contents of the mapTree then we print the keys and the values of the contents.



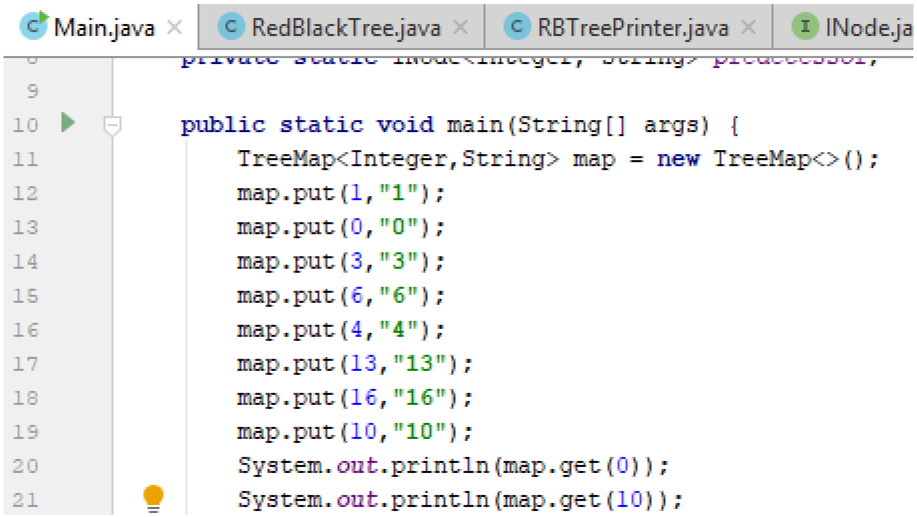
```java
    private static INode<Integer, String> predecessor;

    public static void main(String[] args) {
        TreeMap<Integer,String> map = new TreeMap<>();
        map.put(1,"1");
        map.put(0,"0");
        map.put(3,"3");
        map.put(6,"6");
        map.put(4,"4");
        map.put(13,"13");
        map.put(16,"16");
        map.put(10,"10");
        Set<Map.Entry<Integer,String>> set = map.entrySet();
        for (Map.Entry<Integer,String> entry : set) System.out.println(entry.getKey()+" "+entry.getValue());
```



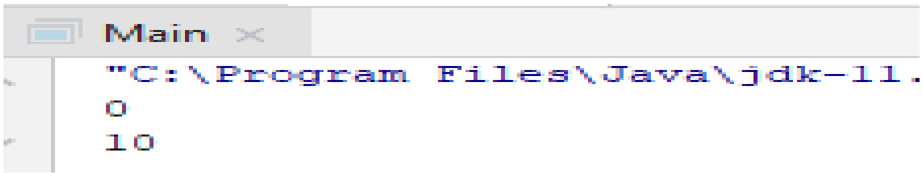
```
"C:\Program Files\
0 0
13 13
1 1
4 4
3 3
16 16
6 6
10 10
```

➢ Example 6:

Here we put some elements in the mapTree , then we print the values of the keys “0” , “10”.



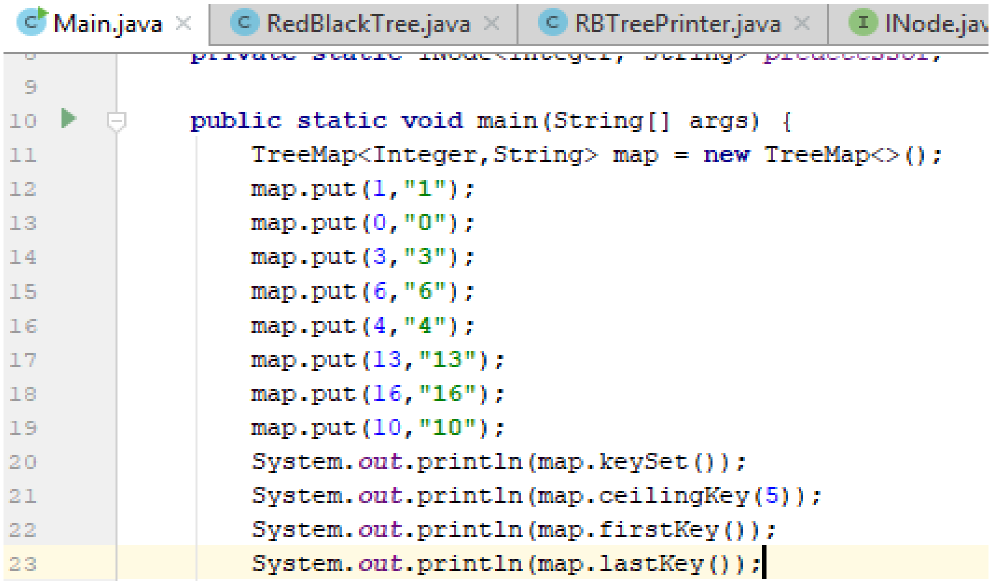
```
    private static INode<Integer, String> predecessor;

    public static void main(String[] args) {
        TreeMap<Integer,String> map = new TreeMap<>();
        map.put(1,"1");
        map.put(0,"0");
        map.put(3,"3");
        map.put(6,"6");
        map.put(4,"4");
        map.put(13,"13");
        map.put(16,"16");
        map.put(10,"10");
        System.out.println(map.get(0));
        System.out.println(map.get(10));
```



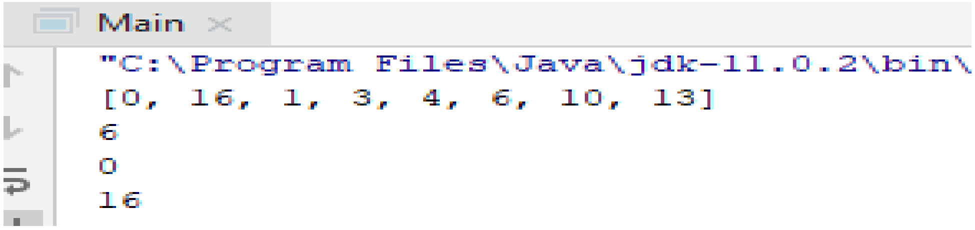
```
"C:\Program Files\Java\jdk-11.
0
10
```

➢ Example 7:

Here we put some elements in the treeMap , then we print the set of keys , then we print the ceiling key(5) which is 6 in the treeMap , then we print the first key in the treeMap which is 0 , then we print the last key in the treeMap which is 16.



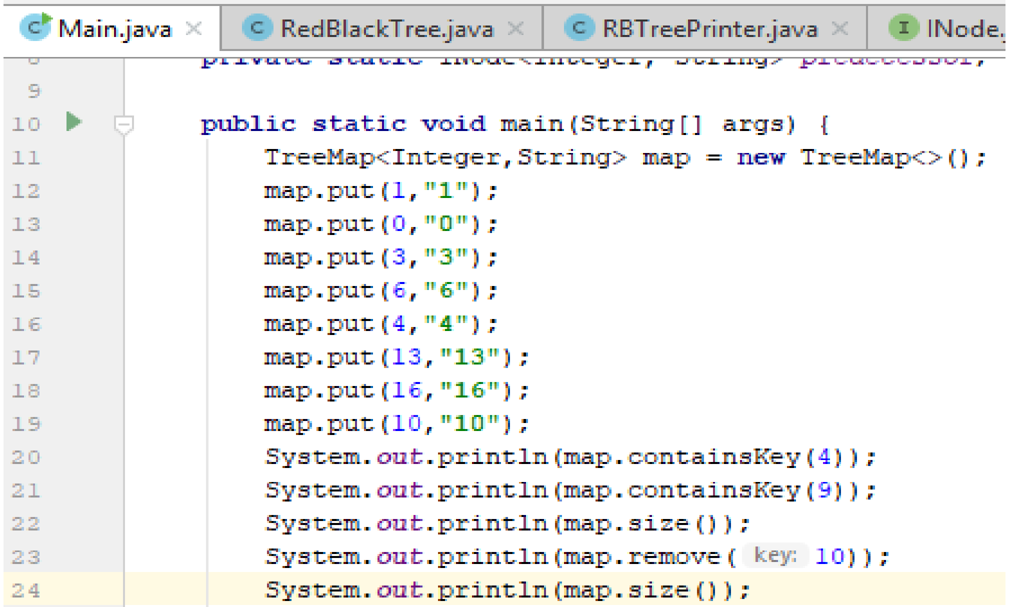
```
    private static INode<Integer, String> predecessor;

    public static void main(String[] args) {
        TreeMap<Integer,String> map = new TreeMap<>();
        map.put(1,"1");
        map.put(0,"0");
        map.put(3,"3");
        map.put(6,"6");
        map.put(4,"4");
        map.put(13,"13");
        map.put(16,"16");
        map.put(10,"10");
        System.out.println(map.keySet());
        System.out.println(map.ceilingKey(5));
        System.out.println(map.firstKey());
        System.out.println(map.lastKey());
```



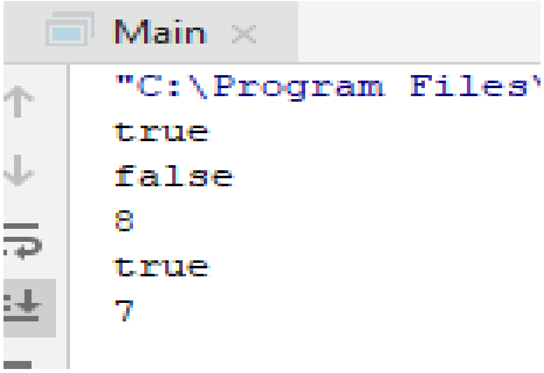
```
"C:\Program Files\Java\jdk-11.0.2\bin\
[0, 16, 1, 3, 4, 6, 10, 13]
6
0
16
```

➢ Example 8:

Here we put some nodes in the treeMap , then we check if it contains key(4) which is true , then we check if it contains key(9) which is false, then we print the size of the treeMap , then we remove the element of key 10 ,then we print the size again.



```java
    private static INode<Integer, String> predecessor;

    public static void main(String[] args) {
        TreeMap<Integer,String> map = new TreeMap<>();
        map.put(1,"1");
        map.put(0,"0");
        map.put(3,"3");
        map.put(6,"6");
        map.put(4,"4");
        map.put(13,"13");
        map.put(16,"16");
        map.put(10,"10");
        System.out.println(map.containsKey(4));
        System.out.println(map.containsKey(9));
        System.out.println(map.size());
        System.out.println(map.remove( key: 10));
        System.out.println(map.size());
```



```
"C:\Program Files
true
false
8
true
7
```