```java
/*
 *    An implementation of a generic Binary Search Tree.
 */

import java.util.Scanner;

public class BST<T extends Comparable <T>> implements BSTInterface<T>
{
    private Node root = null;   // The root of the binary tree.

    public BST()
    {
        root = null;
    }

    public void makeEmpty()
    {
        root = null;
    }

    public boolean isEmpty()
    {
        return root == null;
    }


    /*
     *   equals() − return true if two BST’s have the same
     *              structure and data values, else
     *              return false.
     */

    public boolean equals(BST<T> b)
    {
        return requals(this.root, b.root);
    }


    private boolean requals(Node r, Node b)
    {
        if(r==null&&b==null){
            return true;
        }
        if(r!=null&&b!=null){
            return ((r.data.equals(b.data))&&requals(r.lchild,b.lchild)&&requals(r.rchild,b.rchild));
        }
        return false;
    }




    /*
     *  insert() − insert a node with data value x
     *             in the Binary Search Tree.
     */

    public void insert(T x)
    {
        this.root = rinsert(this.root, x);
    }


    /*
     *   rinsert() − return a pointer to the root of a BST
     *               with data item x inserted.  Do not
     *               insert duplicate data items.
     */

    private Node rinsert(Node root, T x)
    {
        if (root == null)                          // Base Step − Empty tree
            root =  new Node(x, null, null);
        else if (x.compareTo(root.data) < 0)        // Smaller values go in
            root.lchild =  rinsert(root.lchild, x); // the left subtree,
        else if (x.compareTo(root.data) > 0)        // larger values go in
            root.rchild =  rinsert(root.rchild, x); // the right subtree.

        return root;

    }


    public void printTree()
    {
        rPrintTree(root,0);
    }

    /*
     *    rPrintTree() − the usual quick recursive method to print a tree.
     */

    public void rPrintTree(Node r, int level)
    {

        if (r == null)          // Empty tree.
            return;

        rPrintTree(r.rchild, level + 1);   // Print the right subtree.

        for (int i = 0; i < level; i++)
            System.out.print("   ");

        System.out.println(r.data.toString());

        rPrintTree(r.lchild, level + 1);
    }

    public void preorder()
    {
        rpreorder(this.root);  //stub
    }

    /*
     *    rpreorder() − a recursive routine to perform
     *                  a preorder traversal of a BST.
     *                  We will simply write the data items
     *                  the order they are visited by the traversal.
     */

    private void rpreorder(Node r)
    {
        if(r==null){
            return;
        }
        System.out.print(r.data+" ");
        rpreorder(r.lchild);
        rpreorder(r.rchild);
    }

    public void postorder()
    {
        rpostorder(this.root);
    }

    /*
     *    rpostorder() − a recursive routine to perform
     *                 a postorder traversal of a BST.
     *                 We will simply write the data items
     *                 the order they are visited by the traversal.
     */

    private void rpostorder(Node r)
    {
        if(r==null){
            return;
        }
        rpostorder(r.lchild);
        rpostorder(r.rchild);
        System.out.print(r.data+" ");
    }


    /*
     *   Perform an inorder traversal of the tree.
     */

    public void inorder()
    {
        rinorder(this.root);
    }

    /*
     *    rinorder() − a recursive routine to perform
     *                 an inorder traversal of a BST.
     *                 We will simply write the data items
     *                 the order they are visited by the traversal.
     */

    private void rinorder(Node r)
    {
        if(r==null){
            return;
        }
        rinorder(r.lchild);
        System.out.print(r.data+" ");
        rinorder(r.rchild);
    }


    public T find(T x)
    {
        Node ptr=root;
        while(ptr!=null){
            if(ptr.data.equals(x)){
                return ptr.data;
            }
            else if(x.compareTo(ptr.data)>0){
                ptr=ptr.rchild;
            }
            else{
                ptr=ptr.lchild;
            }
        }
        return null;
    }


    public T findMax()
    {
        if(root==null){
            return null;
        }
        Node ptr=root;
        while(ptr.rchild!=null){
            ptr=ptr.rchild;
        }
        return ptr.data;
    }


    public T findMin()
    {
        if(root==null){
            return null;
        }
        Node ptr=root;
        while(ptr.lchild!=null){
            ptr=ptr.lchild;
        }
        return ptr.data;
    }

    public void removeMin()
    {
        T x = findMin();

        if (x == null)
           return;

        remove(x);
    }

    public void removeMax()
    {
        T x = findMax();

        if (x == null)
            return;

        remove(x);
    }
```

```java
    public void remove(T x)
    {
        Node ptr = root;
        Node follow = ptr;
        boolean done = false;

        while (!done) {
            if (ptr == null)
                return;
            else if (x.compareTo(ptr.data) == 0) {
                done = true;
            }
            else if (x.compareTo(ptr.data) < 0) {
                follow = ptr;
                ptr = ptr.lchild;
            }
            else {
                follow = ptr;
                ptr = ptr.rchild;
            }
        }

        // Handle the case where the node to delete is a leaf.

        if (ptr.lchild == null && ptr.rchild == null) {
            if (ptr == root)
                root = null;
            else if (follow.lchild == ptr)
                follow.lchild = null;
            else
                follow.rchild = null;
            return;
        }

        // Handle the case where the node to delete has a left subtree
        // but no right subtree.

        if (ptr.rchild == null) {
            if (ptr == root)
                root = ptr.lchild;
            else if (follow.lchild == ptr)
                follow.lchild = ptr.lchild;
            else if (follow.rchild == ptr)
                follow.rchild = ptr.lchild;
            return;
        }

        // Handle the case where the node to delete has a right subtree
        // but no left subtree.

        if (ptr.lchild == null) {
            if (ptr == root)
                root = ptr.rchild;
            else if (follow.lchild == ptr)
                follow.lchild = ptr.rchild;
            else if (follow.rchild == ptr)
                follow.rchild = ptr.rchild;
            return;
        }
        // Handle the case where the node to delete has both subtrees.
        // In this case, we swap the data in the node with the
        // smallest data value in its right subtree, then we delete this
        // "smallest data" node.

        Node smallPtr = ptr.rchild;

        // Locate the smallest data value in the right subtree.

        follow = ptr;

        while (smallPtr.lchild != null) {
            follow = smallPtr;
            smallPtr = smallPtr.lchild;
        }

        T temp = ptr.data;           // Swap the values.
        ptr.data = smallPtr.data;
        smallPtr.data = temp;

        // Now delete the smallPtr node.

        if (ptr.rchild == smallPtr) {
            ptr.rchild = smallPtr.rchild;
        }
        else {
            follow.lchild = smallPtr.rchild;
        }




    }




    //
    //  inner Node class
    //

    private class Node
    {
        T data;
        Node lchild;
        Node rchild;


        public Node()
        {
            this.data = null;
            this.lchild = null;
            this.rchild = null;
        }


        public Node(T data, Node lchild, Node rchild)
        {
            this.data = data;
            this.lchild = lchild;
            this.rchild = rchild;
        }
    }


    public static void main(String[] args)
    {
        Scanner keyb = new Scanner(System.in);

        BST<Integer> t1 = new BST<Integer>();  // Create Binary Search Trees
        BST<Integer> t2 = new BST<Integer>();
        BST<Integer> t3 = new BST<Integer>();
        BST<Integer> t4 = new BST<Integer>();
        BST<Integer> t5 = new BST<Integer>();
        BST<Integer> t6 = new BST<Integer>();
        BST<Integer> t7 = new BST<Integer>();
        BST<Integer> t8 = new BST<Integer>();
        BST<Integer> t9 = new BST<Integer>();
        BST<Integer> t10 = new BST<Integer>();
        BST<Integer> t11 = new BST<Integer>();
        BST<Integer> t12 = new BST<Integer>();
        BST<Integer> t13 = new BST<Integer>();


        t1.insert(40);   // Two identical trees are created to test the equals()
        t1.insert(60);   // method.  To test trees that are not equal, simply
        t1.insert(20);   // comment out one of the Insertions − or change one of
        t1.insert(new Integer(50));   // the data values.
        t1.insert(80);
        t1.insert(70);
        t1.insert(new Integer(10));
        t1.insert(48);
        t1.insert(41);
        t1.insert(46);
        t1.insert(47);
        t1.insert(44);
        t1.insert(5);
        t1.insert(15);


        t2.insert(40);
        t2.insert(60);
        t2.insert(20);
        t2.insert(50);
        t2.insert(80);
        t2.insert(70);
        t2.insert(10);
        t2.insert(48);
        t2.insert(41);
        t2.insert(46);
        t2.insert(47);
        t2.insert(44);
        t2.insert(5);
        t2.insert(15);

        System.out.println(" tree t1 ----------------");
        t1.printTree();
        System.out.println(" tree t2 ----------------");
        t2.printTree();
        System.out.println("-------------------------");

        System.out.println("t1 and t2 should be equal.");


        if (t1.equals(t2))
            System.out.println("Tree t1 equals tree t2\n");
        else
            System.out.println("Tree t1 doesn’t equal tree t2\n");

        System.out.println("---------------------");

        t4.insert(40);   // Two identical trees are created to test the equals()
        t4.insert(60);   // method.  To test trees that are not equal, simply
        t4.insert(20);   // comment out one of the Insertions - or change one of
        t4.insert(new Integer(50));   // the data values.
        t4.insert(80);
        t4.insert(90);
        t4.insert(new Integer(10));
        t4.insert(48);
        t4.insert(41);
        t4.insert(46);
        t4.insert(47);
        t4.insert(44);
        t4.insert(5);
        t4.insert(15);


        t5.insert(40);
        t5.insert(60);
        t5.insert(20);
        t5.insert(50);
        t5.insert(80);
        t5.insert(70);
        t5.insert(10);
        t5.insert(48);
        t5.insert(41);
        t5.insert(46);
        t5.insert(47);
        t5.insert(44);
        t5.insert(5);
        t5.insert(15);


        System.out.println(" tree t4 ----------------");
        t4.printTree();
        System.out.println(" tree t5 ----------------");
        t5.printTree();
        System.out.println("-------------------------");


        System.out.println("t4 and t5 should not be equal.");

        if (t5.equals(t4))
            System.out.println("Tree t4 equals tree t5\n");
        else
            System.out.println("Tree t4 doesn’t equal tree t5\n");

        System.out.println(" tree t1 ----------------");
        t1.printTree();
        System.out.println(" tree t3 ----------------");
        t3.printTree();

        System.out.println("---------------------");
        System.out.println("t1 and t3 should not be equal.");
        if (t1.equals(t3))
            System.out.println("Tree t1 equals tree t3\n");
        else
            System.out.println("Tree t1 doesn’t equal tree t3\n");

        System.out.println(" tree t9 ----------------");
        t9.printTree();
        System.out.println(" tree t3 ----------------");
        t3.printTree();

        System.out.println("t9 and t3 should be equal.");
        if (t9.equals(t3))
            System.out.println("Tree t9 equals tree t3\n");
        else
            System.out.println("Tree t9 doesn’t equal tree t3\n");

        System.out.println("---------------------");
        t10.insert(20);
        t10.insert(10);
        t10.insert(2);
        t10.insert(40);

        t11.insert(20);
        t11.insert(10);
        t11.insert(2);

        System.out.println(" tree t10 ---------------");
        t10.printTree();
        System.out.println(" tree t11 ---------------");
        t11.printTree();
        System.out.println("-------------------------");

        System.out.println("---------------------");
        System.out.println("t10 and t11 should not be equal.");
        if (t10.equals(t11))
            System.out.println("Tree t10 equals tree t11\n");
        else
            System.out.println("Tree t10 doesn’t equal tree t11\n");

        System.out.println("---------------------");
        t12.insert(20);
        t12.insert(10);
        t12.insert(40);
        t12.insert(50);

        t13.insert(20);
        t13.insert(10);
        t13.insert(2);
        t13.insert(40);
        t13.insert(50);

        System.out.println(" tree t12 ---------------");
        t12.printTree();
        System.out.println(" tree t13 ---------------");
        t13.printTree();
        System.out.println("-------------------------");

        System.out.println("−−−−−−−−−−−−−−−−−−−−−");
        System.out.println("t12 and t13 should not be equal.\n");
        if (t10.equals(t11))
            System.out.println("Tree t12 equals tree t13");
        else
            System.out.println("Tree t12 doesn’t equal tree t13\n");

        System.out.println("\n\nTesting findMin()\n");
        System.out.println("The minimum value in t1 is " + t1.findMin());
        System.out.println("The minimum value in t10 is " + t10.findMin());
        System.out.println("The minimum value in t8 is " + t8.findMin());


        System.out.println("\n\nTesting findMax()\n");
        System.out.println("The maximum value in t1 is " + t1.findMax());
        System.out.println("The maximum value in t10 is " + t10.findMax());
        System.out.println("The maximum value in t8 is " + t8.findMax());


        System.out.println("\n\nTesting inorder traversal\n\n");
        System.out.println("\nAn inorder traversal of t1 is:");
        t1.inorder();
        System.out.println("\nAn inorder traversal of t10 is:");
        t10.inorder();
        System.out.println("\nAn inorder traversal of t11 is:");
        t11.inorder();

        System.out.println("\n\nTesting preorder traversal\n\n");
        System.out.println("\nA preorder traversal of t1 is:");
        t1.preorder();
        System.out.println("\nA preorder traversal of t10 is:");
        t10.preorder();
        System.out.println("\nA preorder traversal of t11 is:");
        t11.preorder();

        System.out.println("\n\nTesting postorder traversal\n\n");
        System.out.println("\nA postorder traversal of t1 is:");
        t1.postorder();
        System.out.println("\nA postorder traversal of t10 is:");
        t10.postorder();
        System.out.println("\nA postorder traversal of t11 is:");
        t11.postorder();

        System.out.println("\n\nTest find()\n");

        Integer  n;
        do {
            System.out.print("Enter a value to search for in t1 (−1 to quit): ");
            n = keyb.nextInt();
            Integer value = t1.find(n);
            if (value == null)
                System.out.println(n.toString() + " is not in t1");
            else
                System.out.println(value.toString() + " is in t1");
        } while (n != −1);

    }

}
```