**Tree Code:Binary Tree**

import java.lang.annotation.ElementType;  
  
public class BinaryTree<E> {  
 private Node<E>root=null;  
 private int Size=0;  
  
 public boolean isEmpty(){  
 return Size==0;  
 }  
  
 public int size(){  
 return Size;  
 }  
  
 protected Node<E> createNode(E e,Node<E>p,Node<E>L,Node<E>R){  
 return new Node<E>(e,p,L,R);  
 }  
  
 protected Node<E>ValidateNode(Position<E>p){  
 if(!(p instanceof Node))  
 throw new IllegalArgumentException("p is not position");  
 Node<E>node=(Node<E>)p;  
 if(node.getParent()==node)  
 throw new IllegalArgumentException("p is deleted");  
 return node;  
 }  
  
 public Position<E> root(){  
 return root;  
 }  
  
 public Position<E> parent(Position<E>p){  
 Node<E> n=ValidateNode(p);  
 return n.getParent();  
 }  
  
 public Position<E> Left(Position<E>p){  
 Node<E> n=ValidateNode(p);  
 return n.getLeft();  
 }  
  
 public Position<E> Right(Position<E>p){  
 Node<E> n=ValidateNode(p);  
 return n.getRight();  
 }  
  
 public int NumOfChild(Position<E>p){  
 int Count=0;  
 if(Left(p)!=null)  
 Count++;  
 if (Right(p)!=null)  
 Count++;  
 return Count;  
 }  
  
 public Position<E>addRoot(E element){  
 if (!isEmpty())  
 throw new IllegalArgumentException("root is found");  
 root=createNode(element,null,null,null);  
 Size=1;  
 return root;  
 }  
  
 public Position<E>addLeft(Position<E>p,E element){  
 Node<E>parent=ValidateNode(p);  
 if(parent.getLeft()!=null)  
 throw new IllegalArgumentException("There is already Left child");  
 Node<E>Child=createNode(element,parent,null,null);  
 parent.setLeft(Child);  
 Size++;  
 return Child;  
  
 }  
  
 public Position<E>addRight(Position<E>p,E element){  
 Node<E>parent=ValidateNode(p);  
 if(parent.getRight()!=null)  
 throw new IllegalArgumentException("There is already Right child");  
 Node<E>Child=createNode(element,parent,null,null);  
 parent.setRight(Child);  
 Size++;  
 return Child;  
  
 }  
  
 public E SetElement(Position<E>p,E element){  
 Node<E>n=ValidateNode(p);  
 E del=n.getElement();  
 n.setElement(element);  
 return del;  
 }  
  
 public E remove(Position<E>p){  
 Node<E>n=ValidateNode(p);  
 if (NumOfChild(p)==2)  
 throw new IllegalArgumentException("P has tow child");  
 Node<E>child=(n.getLeft()!=null?n.getLeft():n.getRight());  
 child.setParent(n.getParent());  
 if(n==root)  
 root=child;  
 else {  
 Node<E>parent=n.getParent();  
 if(n==parent.getLeft())  
 parent.setLeft(child);  
 else  
 parent.setRight(child);  
 }  
 Size--;  
 E del=n.getElement();  
 n.setElement(null);  
 n.setLeft(null);  
 n.setRight(null);  
 n.setParent(n);  
 return del;  
 }  
  
 private static class Node<E> implements Position<E> {  
  
 E element;  
 Node<E> parent;  
 Node<E> Left;  
 Node<E> Right;  
  
 public Node(E element, Node<E> parent, Node<E> left, Node<E> right) {  
 this.element = element;  
 this.parent = parent;  
 Left = left;  
 Right = right;  
 }  
  
 public void setElement(E element) {  
 this.element = element;  
 }  
  
 public Node<E> getParent() {  
 return parent;  
 }  
  
 public void setParent(Node<E> parent) {  
 this.parent = parent;  
 }  
  
 public Node<E> getLeft() {  
 return Left;  
 }  
  
 public void setLeft(Node<E> left) {  
 Left = left;  
 }  
  
 public Node<E> getRight() {  
 return Right;  
 }  
  
 public void setRight(Node<E> right) {  
 Right = right;  
 }  
  
 @Override  
 public E getElement() {  
 return element;  
 }  
 }  
}

public interface Position<E> {  
 E getElement();  
  
}

public class Test {  
 public static void main(String[] args) {  
 BinaryTree<String>t=new BinaryTree<>();  
 Position<String>p=t.addRoot("A");  
 t.addLeft(p,"B");  
 t.addRight(p,"C");  
 System.out.println(t.root().getElement());  
 System.out.println(t.size());  
 System.out.println(t.Left(p).getElement());  
 System.out.println(t.Right(p).getElement());  
 }  
}