Berikut adalah salah satu cara untuk mengimplementasikan ADT Tree (Binary Tree).

Lengkapi kelas **BinaryNode** berikut yang merepresentasikan sebuah **Node** atau sebuah **Vertex** pada Binary Tree ! Beberapa static method terkait operasi pada Binary Tree bisa diletakkan di dalam kelas ini.

```
public class BinaryNode<E> {

	E element;
	BinaryNode<E> left;
	BinaryNode<E> right;


	public BinaryNode (E element, BinaryNode<E> left, BinaryNode<E> right) {
		this.element = element;
		this.left = left;
		this.right = right;
	}


	public BinaryNode (E element) {
		this (element, null, null);
	}


	//Static method yang merupakan operasi pada tree. Method-method ini
	//nantinya akan dipanggil dari kelas BinaryTree

	//menghitung ukuran dari node t, yaitu banyaknya node pada left subtree
	//+ banyakny node pada right subtree + 1 (node t itu sendiri)
	public static <E> int size (BinaryNode<E> t) {









	}

	//menghitung kedalaman/tinggi dari tree, root ada pada t
	public static <E> int height (BinaryNode<E> t) {
		if (t == null) {
			return -1;
		} else {
			return 1 + Math.max(height(t.left), height(t.right));
		}
	}

	//mencari elemen x pada tree yang dimulai dari node t,
	//jika ada True, False sebaliknya
	public static <E> boolean find (BinaryNode<E> t, E x) {
```

```
    }

    //banyaknya kemunculan x pada tree yang dimulai dari note t
    public static <E> int count (BinaryNode<E> t, E x) {









    }


    //count Leaves, berapa banyak daun yang ada pada tree yang dimulai
    //dari node t
    public static <E> int countLeaves (BinaryNode<E> t) {









    }

    //reflection, bayangkan Tree dicerminkan ! subtree kiri menjadi subtree
    //kanan, dan sebaliknya
    public static <E> void reflect (BinaryNode<E> t) {









    }

    //duplicate, menghasilkan tree baru yang serupa. Lalu, alamat root dari
    //dari tree baru tersebut dikembalikan
    public static <E> BinaryNode<E> duplicate (BinaryNode<E> t) {









    }
```

```
	/******traverse pada tree******/
	//method-method berikut BUKAN static method, artinya milik current object

	//inorder dimulai dari current node
	public void printInOrder () {




	}

	//postorder dimulai dari current node
	public void printPostOrder () {




	}

	//preorder dimulai dari current node
	public void printPreOrder () {




	}

	//levelorder dimulai dari current node
	public void printLevelOrder () {




	}
}
```

Berikut adalah kelas **BinaryTree** yang merepresentasikan sebuah Binary Tree. Di dalam kelas ini, kita hanya perlu menyimpan sebuah pointer yang bernama **root**, yang merupakan sebuah **BinaryNode**.

```
public class BinaryTree<E> {

	private BinaryNode<E> root;

	public BinaryTree() {
		root = null;
	}

	public BinaryTree(E rootItem) {

	}

	public BinaryTree(E rootItem, BinaryNode<E> left, BinaryNode<E> right) {

	}
```

```
	public void makeEmpty() {
		root = null;
	}

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

	//ukuran tree, jadi panggil static method size dimulai dari root
	public int size() {
		return BinaryNode.size(root);
	}

	//tinggi tree, jadi panggil static method height dimulai dari root
	public int height() {
		return BinaryNode.height(root);
	}



	// tambahkan find(...), count(...), dan countLeaves(...)
	// dengan cara yang sama seperti size() dan height() di atas


	public void reflect() {

	}

	public BinaryNode<E> duplicate() {

	}

	public void printPreOrder() {
		if (root != null)
			root.printPreOrder();
	}


	public void printInOrder() {


	}

	public void printPostOrder() {


	}

	public void printLevelOrder() {


	}
}
```