15. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E> isEmpty(), clear()*

`+isEmpty(): boolean`

An empty binary tree is one that contains no nodes. In our implementation, the *BinaryTree* is empty if the *mRoot* reference is null. Therefore, *isEmpty*() is easily implemented:

```
// Returns true if this BinaryTree is an empty tree.
public boolean isEmpty() {
  return getRoot() == null;
}
```

`+clear(): void`

The *clear*() method is used to remove all of the *Node*s in this *BinaryTree*. After *clear*() returns, the *BinaryTree* will be empty. To clear the tree, we prune both the left and right subtrees of the *mRoot* node and then set *mRoot* to null.

```
// Makes this BinaryTree empty.
  public void clear() {
      prune(getRoot());
      setRoot(null);
  }
```

15. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E> getHeight()*

The height of a binary tree is the maximum of the levels of each node, where the level of the root is 0, the levels of the left and right children of the root are 1, the levels of those children's children are 2, and so on. The *BinaryTree<E>* class contains two overloaded *getHeight*() methods. The first public one is designed to be called on a *BinaryTree<E>* object and will return the height of the entire tree:

```
// Returns the height of this BinaryTree.
public int getHeight() {
  return getHeight(getRoot());
}
```

This method makes use of a protected helper method which returns the height of a subtree rooted at *pRoot*. Note that *getHeight*() is a recursive method, i.e., the height of the left subtree of *pRoot* is determined by calling *getHeight*() with the left child *Node* of *pRoot* as the parameter—if there is a left child. We determine the height of the subtree rooted at the right child *Node* of *pRoot* in a similar manner. The height of the subtree rooted at *pRoot* then is the maximum of the height of the left subtree + 1 and the height of the right subtree + 1. For a leaf node, which has no left or right child nodes, *getHeight*() returns 0.

15. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E> getHeight()*

```
// Returns the height of the subtree rooted at pRoot where the height is the
// maximum of the heights of the left and right subtrees of pRoot.
protected int getHeight(Node pRoot) {
  int heightLeft = 0, heightRight = 0;
  if (pRoot == null) return 0;
  if (pRoot.hasLeft()) heightLeft = getHeight(pRoot.getLeft()) + 1;
  if (pRoot.hasRight()) heightRight = getHeight(pRoot.getRight()) + 1;
  return Math.max(heightLeft, heightRight);
}
```

15. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E> getHeight()*

For example:

```
public class Main {
  public static void main(String[] pArgs) { new Main().run(); }
  private void run() {
    BinaryTree<Integer> tree = new BinaryTree<>(1); // root contains 1
    BinaryTree.Iterator it = tree.iterator(); // it refers to the root of tree
    it.addLeft(2); it.addRight(3);            // root's lf child is 2, rt child is 3
    it.moveLeft();                            // it refers to 2
    it.addLeft(4); it.addRight(5);            // 2's lf child is 4, rt child is 5
    it.moveUp();                              // it refers to root
    it.moveRight();                           // it refers to 3
    it.addLeft(6); it.addRight(7);            // 3's lf child is 6, rt child is 7
    System.out.println("tree height = " + tree.getHeight());
    System.out.println("it height = " + it.getHeight());
  }
}
```

Output

```
tree height = 2, it height = 1
```