19. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E>* Pruning

The final *BinaryTree<E>* methods we will look at are the methods that prune the tree.

`#prune(pRoot: Node<E>): void`
Prunes both the left and right subtrees of the subtree rooted at *pRoot*.

`#pruneLeft(pRoot: Node<E>): void`
Prunes the left subtree of the subtree rooted at *pRoot*.

`#pruneRight(pRoot: Node<E>): void`
Prunes the right subtree of the subtree rooted at *pRoot*.

These are all protected method because they are not intended to be called directly on *BinaryTree<E>* objects, but rather, are called from the following *BinaryTree.Iterator<E>* methods which were discussed earlier in *Trees : Section 14*.

```
+prune(): void
+pruneLeft(): void
+pruneRight(): void
```

19. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E>* Pruning

The *prune(Node<E> pRoot)* methods prunes the subtree rooted at *pRoot:*

```
// Prunes the tree rooted at pTree by pruning the left and right subtrees and then
// setting the left and right references of the root node to null. Note: this
// method does not delete the data stored in the root node of pTree, nor does it
// set the root node of pTree to null.
protected void prune(Node<E> pRoot) {
  if (pRoot == null) return;
  prune(pRoot.getLeft());
  pRoot.setLeft(null);
  prune(pRoot.getRight());
  pRoot.setRight(null);
}
```

19. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E>* Pruning

pruneLeft() and *pruneRight*() are relatively simple methods. We prune the left (right) subtree of *pRoot* by calling *prune*(*Node*<*E*>) on the left (right) child of *pRoot*. Then we set the left (right) child reference to null.

```
// Prunes only the left subtree of this BinaryTree.
protected void pruneLeft(Node<E> pRoot) {
  prune(pRoot.getLeft());
  pRoot.setLeft(null);
}

// Prunes only the right subtree of this BinaryTree.
protected void pruneRight(Node<E> pRoot) {
  prune(pRoot.getRight());
  pRoot.setRight(null);
}
```