14. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Iterator<E>* Methods

`+get(): E`

Returns the data stored in this *Iterator*'s current *Node*.

`+set(pData: E): void`

Changes the data stored in this *Iterator*'s current *Node*.

`+find(pData: E): boolean`

Searches the binary tree rooted at this *Iterator*'s current *Node* for *pData*. If found, this *Iterator*'s current *Node* is changed to the *Node* containing *pData* and true is returned. If not found, this *Iterator*'s current *Node* will not be changed and false will be returned.

For example:

```
Integer x = it.get();          // x ← 1
it.moveLeft();                 // it refers to 2
Integer y = it.get();          // y ← 2
it.set(99);                    // Change data in node from 2 to 99
y = it.get();                  // y ← 99
it.set(2);                     // Change data in node from 99 back to 2
boolean found = it.find(3);    // 3 not found. it still refers to 2, found ← false
it.moveToRoot();               // it refers to 1
found = it.find(3);            // it refers to 3, found ← true
```

14. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Iterator<E>* Methods

`+getHeight(): int`

Returns the height of the subtree rooted at this *Iterator*'s current Node.

For example:

```
int h = it.height();  // h ← 1
it.moveToRoot();      // it refers to 1
h = it.height();      // h ← 2
```

`+traverse(pWhich: int, pVisitor: BinaryTreeVisitor): void`

Performs the type of traversal specified by *pWhich* (which must be one of the constants *INORDER*, *LEVEL_ORDER*, *POSTORDER*, or *PREORDER* declared in *BinaryTree<E>*) on the subtree rooted at this *Iterator*'s current *Node*.

pVisitor is an object which implements the *BinaryTreeVisitor* interface.

```
public interface BinaryTreeVisitor<E> {
  void visit(E pData);
}
```

As each *Node* is visited during the traversal, the *visit*(*E data*) method will be called on *pVisitor*.

We will see how the *traverse*() method and the *BinaryTreeVisitor* interface are used in a subsequent section.

14. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Iterator<E>* Methods

`+prune(): void`

Prunes both the left and right subtrees of this *Iterator*'s current *Node*.

`+pruneLeft(): void`

Prunes the left subtree of this *Iterator*'s current *Node*.

`+pruneRight(): void`

Prunes the right subtree of this *Iterator*'s current *Node*.

The prune methods are used to remove an entire subtree. For example:

```
it.moveToRoot();  // it refers to 1
it.moveRight();   // it refers to 3
it.pruneLeft();   // Prunes the left subtree rooted at 3
                  // Level order traversal: 1 2 3 4 5 7
it.moveToRoot();  // it refers to 1
it.pruneLeft();   // Prunes the left subtree rooted at 1
                  // Level order traversal: 1 3 7
it.moveToRoot();  // it refers to 1
it.prune();       // Prunes the left and right subtrees rooted at 1
                  // Level order traversal: 1
```

Note that *prune*() does not eliminate the root node, i.e., this *Iterator*'s current node.