18. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E> findNode()*

`#findNode(pRoot: Node<E>, pData: E): Node<E>`

This protected method is called by the *BinaryTree.Iterator<E>* method to find a *Node<E>* in the subtree rooted at *pRoot* containing the data *pData*. If such a *Node<E>* is located, the method returns a reference to the *Node<E>*, otherwise it returns null.

```
protected Node<E> findNode(Node<E> pRoot, E pData) {
  if (pRoot == null) return null;  // pData will not be found in an empty tree
  if (pRoot.getData().equals(pData)) return pRoot;  // is pData at pRoot
  Node<E> node = findNode(pRoot.getLeft(), pData);  // Search the left subtree
  if (node != null) return node;                    // If found, return node ref
  return findNode(pRoot.getRight(), pData);         // Search the right subtree
}
```

The algorithm compares the data stored in *pRoot* to see if it is equal to *pData* and if so, *pRoot* is the *Node* containing *pData* so *pRoot* is returned. Otherwise, perform a recursive method call to search the left subtree of *pRoot*. If that search returns a nonnull reference, then *node* is a reference to the *Node* containing *pData* so we return *node*. Otherwise, we perform a recursive method call to search the right subtree of *pRoot*. That method call will return either null (*pData*) was not found or it will return a reference to the *Node* where *pData* was found. In either case, we simply return what *findNode*() returns.

Note that this search procedure essentially performs a preorder traversal to locate *pData*. It also does not consider that there may be multiple *Node*s in the tree storing *pData* and will simply return a reference to the first *Node* containing *pData* it encounters during the preorder traversal.

18. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Iterator<E> find()*

The *BinaryTree<E>.findNode()* method is called by *BinaryTree.Iterator<E>.find()* which is implemented thusly:

```
// Searches the binary tree rooted at the current node for pData. If found, the
// current Node is changed to the Node containing pData and true is returned. If
// not found, the current node will not be changed and false will be returned.
public boolean find(E pData) {
  Node<E> node = getTree().findNode(getCurrent(), pData);
  if (node != null) setCurrent(node);
  return node != null;
}
```

For example:

```
BinaryTree<Integer> tree = new BinaryTree<>(1);
BinaryTree.Iterator<Integer> it = tree.iterator();
it.addLeft(2); it.addRight(3);
it.moveLeft(); it.addLeft(4); it.addRight(5);
it.moveUp(); it.moveRight(); it.addLeft(6); it.addRight(7);
it.moveToRoot();
boolean found = it.find(2);   // found is true, it refers to 2,
System.out.println(it.get()); // Displays 2
found = it.find(6);           // found is false, it still refers to 2
System.out.println(it.get()); // Displays 2
```