11. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Node<E>* Class

A *BinaryTree<E>* object encapsulates one instance variable *mRoot* which is a reference to a *BinaryTree.Node<E>* object where *Node<E>* is a generic static nested class within *BinaryTree<E>*:

```
public class BinaryTree<E> {
  Node<E> mRoot;
  ...
  protected static class Node<E> {
    // The data stored in this Node. We use E to represent the type of mData.
    E mData;

    // A reference to the left child Node of this Node.
    Node<E> mLeft;

    // A reference to the right child Node of this Node.
    Node<E> mRight;
    ...
  }
}
```

11. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Node<E>* Class

The *Node* class should look familiar because it is very similar to the *DList.Node* class. However, rather than a previous and next reference to the next *Node* in a list, we have a left and right reference to the left and right children of this *Node*. If *mLeft* or *mRight* is null then the *Node* does not have a left or right child. The *Node* class constructors are:

```
// Creates a new Node storing no data and with mLeft and mRight set to null.
public Node() {
  this(null);
}

// Creates a new Node storing pData as the data and with mLeft and mRight set to
// null.
public Node(E pData) {
  this(pData, null, null);
}

// Creates a new Node storing pData as the data, mLeft initialized to pLeft, and
// mRight initialized to pRight.
public Node(E pData, Node<E> pLeft, Node<E> pRight) {
  setData(pData);
  setLeft(pLeft);
  setRight(pRight);
}
```

11. Trees :: Binary Trees :: Java Implementation :: *BinaryTree.Node<E>* Class

The following accessor and mutator methods are straightforward to implement, so we shall not discuss them in detail, but note that the type of the data stored in each *Node* is specified as *E* which is the generic *BinaryTree<E>* class type parameter.

`+getData(): E`, `+setData(pData: E): void`
Accessor/mutator methods for *mData*

`+getLeft(): Node<E>`, `+setLeft(pLeft: Node<E>): void`
Accessor/mutator methods for *mLeft*

`+getRight(): Node<E>`, `+setRight(pRight: Node<E>): void`
Accessor/mutator methods for *mRight*

getNumChildren(), *hasLeft*(), *hasRight*(), *isLeaf*() are useful helper methods:

```
// Returns the number of children of this Node.
public int getNumChildren() {
  int num = 0;
  if (hasLeft()) ++num;
  if (hasRight()) ++num;
  return num;
}
```

11. Trees :: Binary Trees :: Java Implementation :: BinaryTree.Node<E> Class

```
// Returns true if this Node has a left child Node.
public boolean hasLeft() { return getLeft() != null; }

// Returns true if this Node has a right child Node.
public boolean hasRight() { return getRight() != null; }

// Returns true if this Node is a leaf node.
public boolean isLeaf() { return !hasLeft() && !hasRight(); }
```