2. Linked Lists :: Implementation :: Node Class

We will look at an implementation of a **doubly linked list** where the type of each element is an *Integer*. If you understand how the code works, it is simple to change it to store data of a different type.

A doubly linked list stores the data as a **linear sequence** of nodes where each node contains three instance variables:

1. The data (*mData*)
2. A reference to the preceding node (*mPrev*)
3. A reference to the succeeding node (*mNext*)

We shall name our doubly linked list class *DList* and it shall contain a protected **static nested class** named *Node*.

```
public class DList {
  ...
  protected static class Node {
    // The data stored in this Node.
    Integer mData;
    // A reference to the succeeding Node in the DList.
    Node mNext;
    //A reference to the preceding Node in the DList.
    Node mPrev;
  }
}
```

2. Linked Lists :: Implementation :: Node Class :: Static Nested Classes

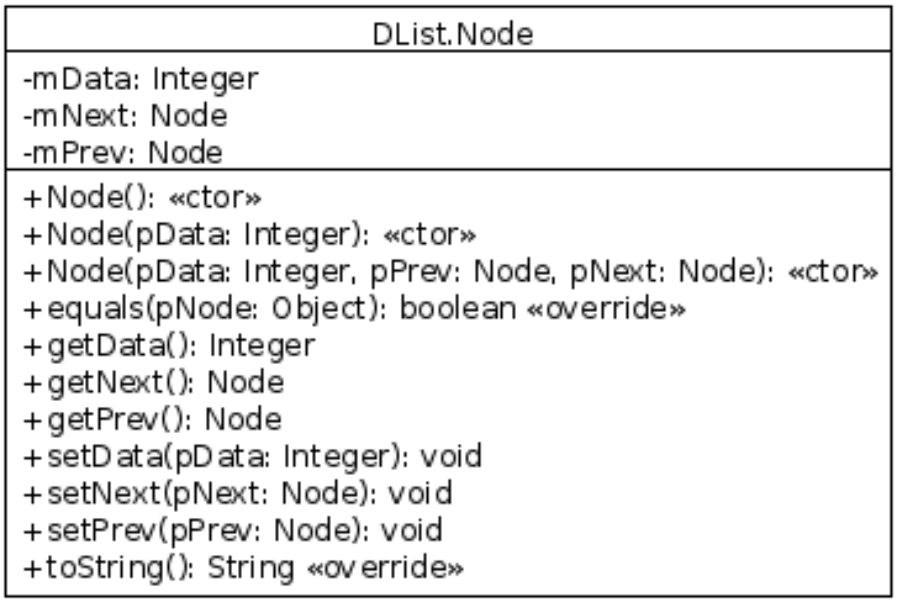
We first talked about **nested classes** in *Objects and Classes II : Section 1* when we looked at the three types of **nonstatic nested classes**: (1) inner classes; (2) local classes; and (3) anonymous classes.

A **static nested class** is not an inner class. What static means in this sense is that the class exists independent of objects of the outer class. In essence, a static class is simply a class nested within an outer class for **name hiding purposes**, i.e., we do not want to create another source code file and add another class name to the global namespace.

As a static class, *Node* has no access to the instance variables of *DList* objects, but that is fine because *Node* will never need to access those instance variables anyway. In fact, other than being encapsulated within *DList, Node* has no knowledge of the existence of the *DList* class.

2. Linked Lists :: Implementation :: Node Class :: UML Class Diagram

The remainder of the *Node* class is fairly simple. It simply consists of some constructors, accessor and mutator methods for the instance variables, and overridden *equals*() and *toString*() methods (inherited from *Object*):



We will look at the implementation of some of these methods in the next section.