1 Designing the Flexible Tree

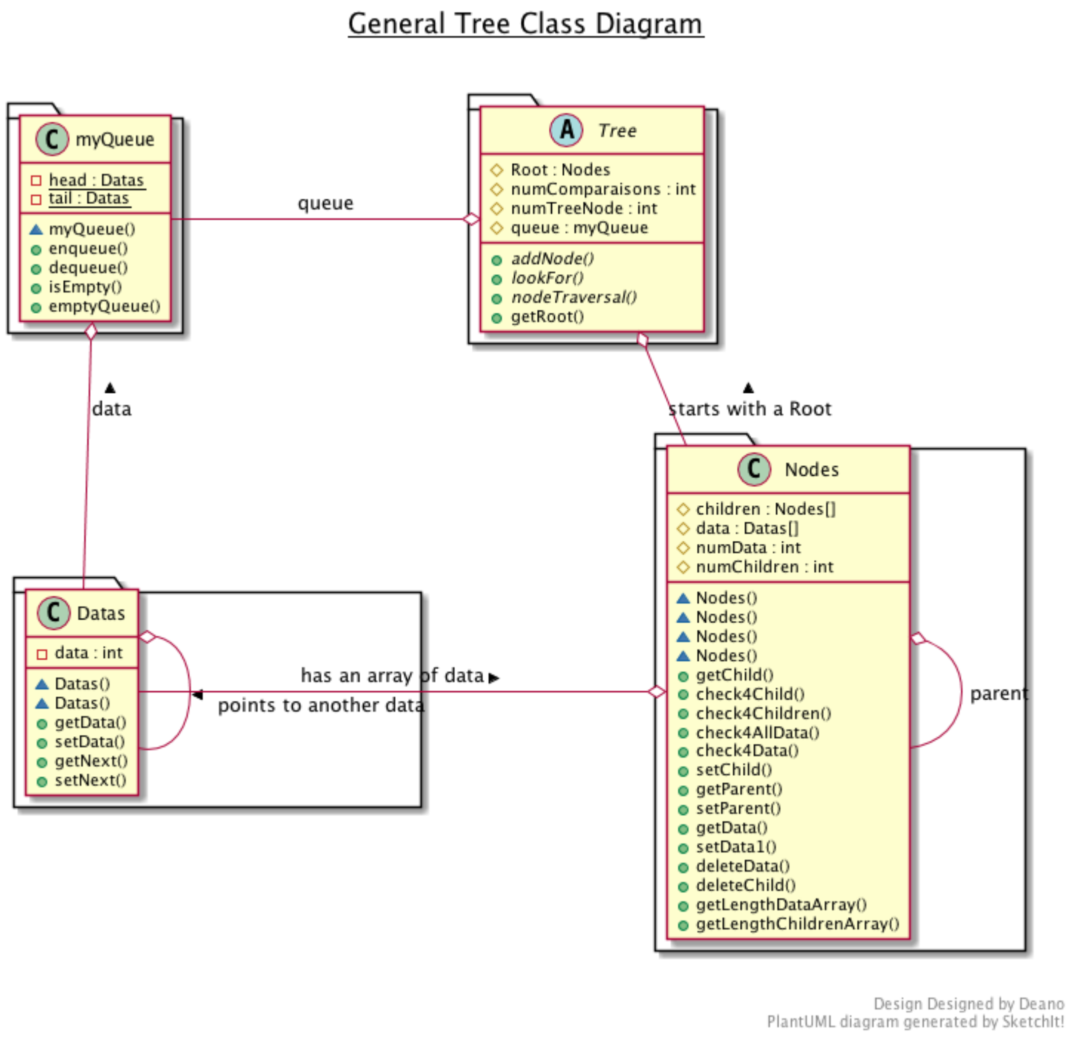

Figure 1 : Class Diagram of a General Tree

The class diagram above will be used to explain the implementation of a general tree structure.

The first class, the Datas, was created in order to save the data of the nodes of a tree. This class provides the ease of checking whether a node holds one data, more or simply none at all. Unlike the int, which is a primitive data type of Java, an instantiation of the Datas can be set to null; this provides the necessary mechanism to easily check whether a node contains data or not. The class Datas also has a pointer to its own type. This is there to facilitate the implementation of a queue data structure.

The second class, the myQueue class, is the data structure explained above. It is a queue, implemented on a list structure, so one data points to the next data instead of an array which uses a fix amount of consecutive space in memory. Data can therefore be added without having to worry about the size constraint of an array. It is believed that this data structure can be useful for the implementation of trees.

Next is the class Nodes, which is a representation of the different nodes of a tree. It contains a link to a parent node which can be set to null, an array of children and data because the general tree can have any fix number of children and data as well as the usual setters and getters. The array of children will hold references for the children of a node while the array of data will hold references for its data. The parent link will provide a link to the parent of a node. The root node will have its parent node set to null, thus indicating that it is the root. The class also includes a variety of useful methods. For instance, they provides one the options to check if all children are set or if a node contains a data; they also allows one to delete children and datas from nodes. In addition, several constructors are also implemented to facilitate the different implementations when such a Node will be used.

The next class is an abstract Tree class. It is made abstract because each specific tree will then inherit from it. It has a protected node already present because every tree has a root node. There are also a queue, and some counters such as the amount of comparisons and number of nodes created. The queue data structure is there so any tree which inherits from this class will have access to this data structure in order to reduce the complexity of the implementations. The Tree class also has three abstract methods, namely : addNote, lookFor and nodeTraversal. Each specific tree which inherits from the tree class would then need to implement these three methods because each tree would have different implementations. The other public method is there to return the root of a tree.

11 Designing the TwoThree Tree

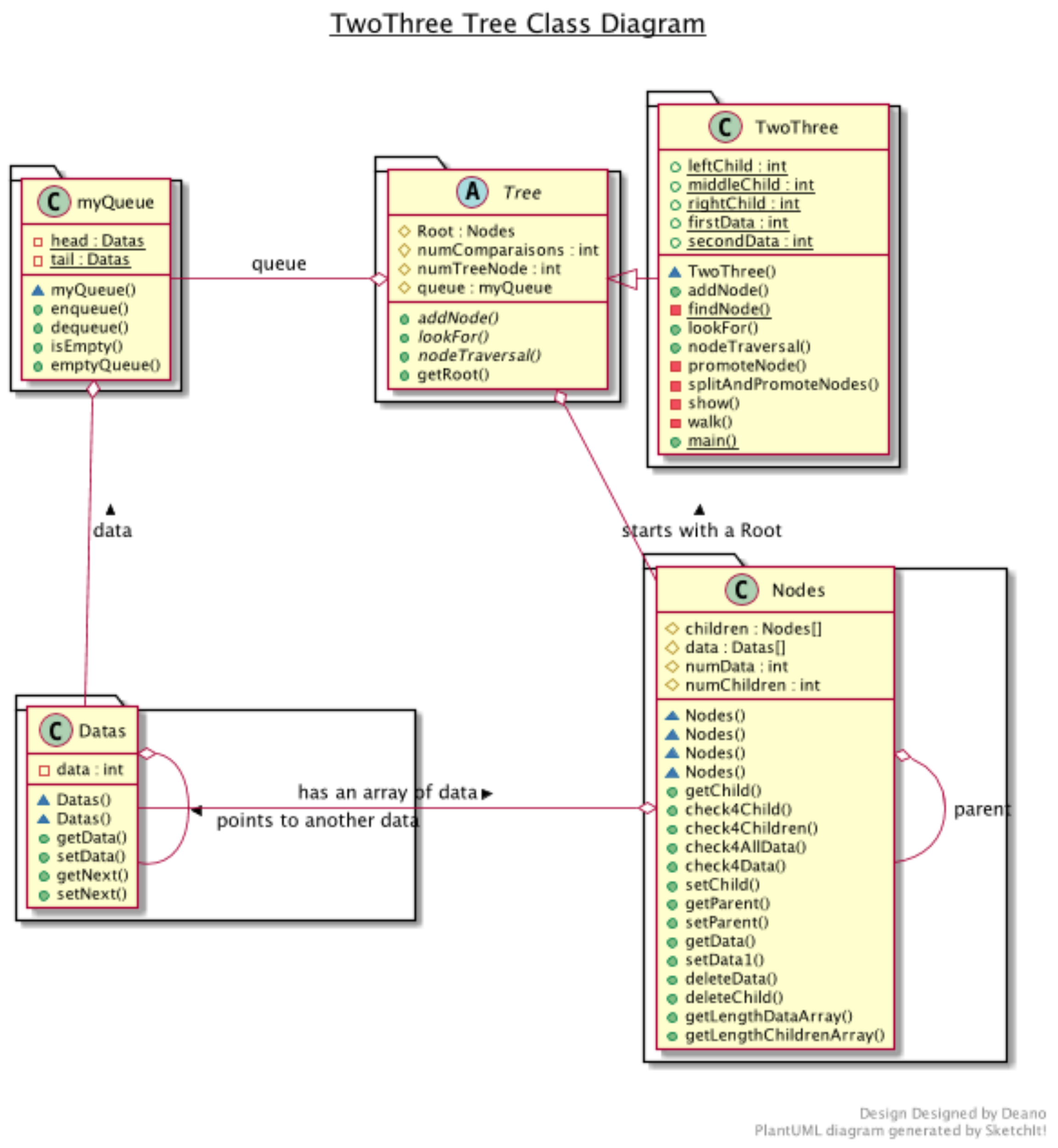

Figure 2 : Class Diagram of a TwoThree Tree

The figure 2 above is a class diagram for the TwoThree Tree class, which inherits from the abstract Tree class. In addition to implementing the 3 abstract methods of its parent class, It also has five public static variables, namely: leftChild, middleChild, rightChild, firstData and secondData. These are names for indexes which will be used in the children and data arrays respectively, since a TwoThree tree can have a maximum number of three children and two datas. Those static variables will make the implementation easier and reduce confusion while making the code easier to read. They are also made public so they can be used outside of the class to facilitate implementation. The TwoThree Class also has five private methods, which are used within the class itself. They include a findNode, promoteNode, splitAndPromoteNode, show and walk method. The findNode is there to find a node where a value does and returns the node. This quickly shows which nodes should be dealt with. The promoteNode and splitAndPromoteNode are use for the addition of a value in the tree. They deal with the addition of values different depending on different scenarios. The addition of a node to the tree has been divided into several methods to promote modularity and facilitate the reading of the code. The last two walk and show methods are part of the nodeTraversal method. The walk method walks from node to node and puts the data in the queue data structure. The show method then gets the data from the queue and displays it to the user. The modular way of dividing the work into several methods stays true to the values of divide and conquer. It also helps in facilitating the implementation as it can be very complex.

111 Designing a Red Black Tree

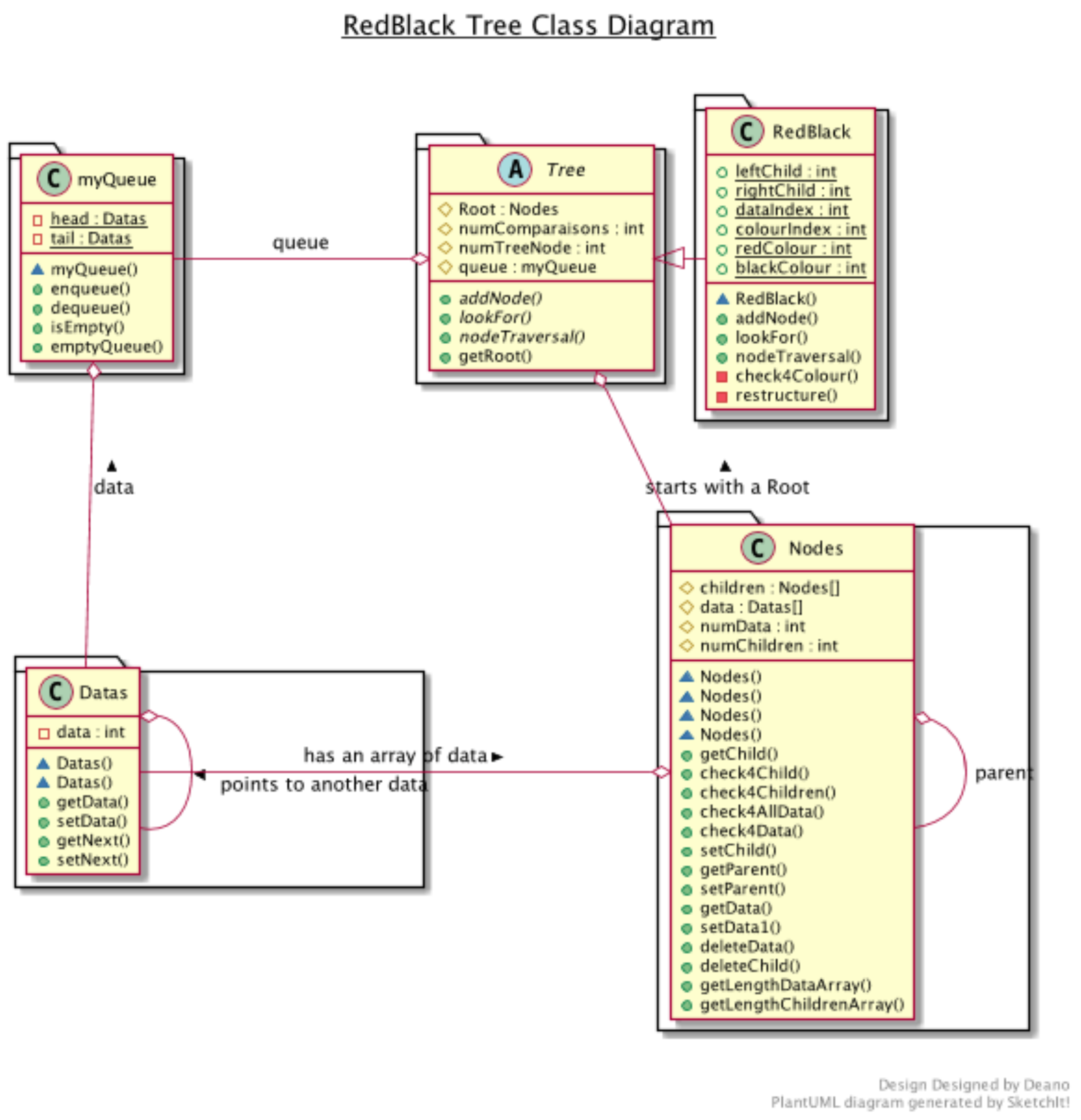

Figure 3 : Class Diagram of a RedBlack Tree

The image above is a Class Diagram for the implementation of a RedBlack Tree. The class will still inherit from the abstract Tree class just like the TwoThree Tree but include different implementations. This means that the RedClass tree will also have two different arrays, namely: the Nodes and Datas arrays. The RedBlack Class will also include static variables such leftChild, rightChild, dataIndex, colourIndex, redColour and blackColour. The Nodes array will hold a Node's children and the leftChild and rightChild indexes will be used for keeping track of the children kept in the array. Having only two children shows that it will be implemented as a special binary tree. The Datas array will not only hold the data, but also the colour of a node. The dataIndex will be the index for the data while the colourIndex will be used to track the colour of the node in the data array, which will still be of length two. The colour in the array will be a value, which will either be of redColour or blackColour. As a result, a child and parent of the same redColour, will have a red link while the others having black colour will indicate the black link of a typical RedBlack Tree. These colours will be at the core of the implementations as they represent the main properties of a RedBlack Tree. These will then change the implementations for the methods of addNote, lookFor and nodeTraversal, compared to the TwoThree Tree. Two private methods will help in implement the tree: check4Colour and restructure. The check4Colour method will look for the colour of nodes between a parent and its children and thus establish their relationship. Finally, the restructure method will be responsible for restructuring the nodes when a new value is added to the tree and then change the colours of nodes as required to keep the properties of the RedBlack Tree.

Advantages of TwoThree Tree over RedBlack Tree

One of the advantages of the TwoThree Tree compared to the RedBlack Tree resides in the implementations. The RedBlack Tree is going to be implemented as a binary tree. However, for its properties to hold, it will be required to keep track of the red and black nodes/edges. These might prove to be much more complicated than the TwoThree tree. For the latter, a simple array of two datas and three children will be enough for the implementations while the RedBlack tree will require more code to figure out where each data goes. The restructuring method of the RedBlack Tree might also be more complex than the promote and split nodes from the TwoThree Tree.