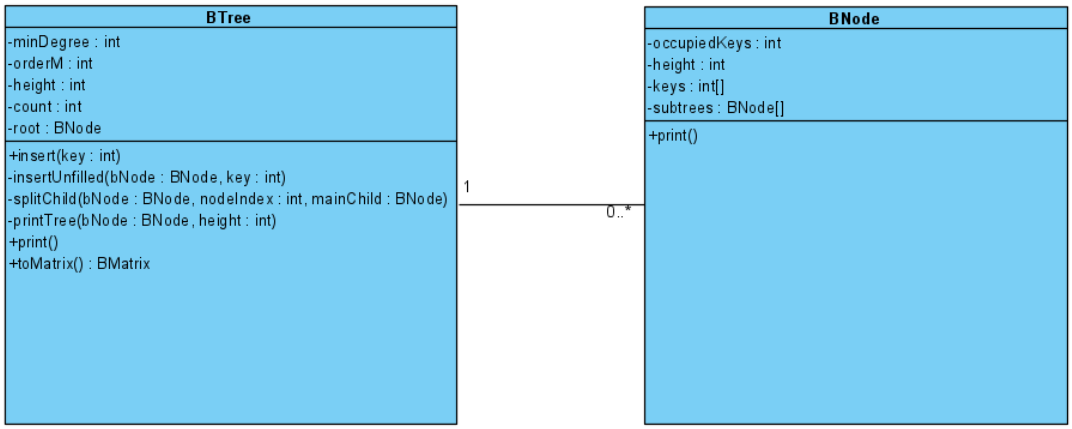
B-Tree Design and Implementation

For this, we need a simple B-Tree implementation as a Java class. For the B-Tree, we will first need a node class which we will call BNode. BNode will contain the total amount of keys it has, its current height, an array of its keys, an array of its sub-trees and a Boolean to check if the node is a leaf. The B-Tree class, which we will call BTree, will contain the following: Its minimum degree, its order (minDegree * 2 – 1), its total height, the total amount of keys in the tree and its root node.

For this implementation, the orderM class member denotes the maximum number of keys a node can have. Furthermore, orderM + 1 would be the maximum number of children a node could have.

Class Diagram



Matrix Design and Implementation

BMatrix

The matrix representation (named BMatrix), consists of four class members: a 2d array of MatrixCell objects, the number of columns, the number of rows and a counter to keep track which column is being generated. The number of columns the matrix has, corresponds to the number of keys that exist in the B-Tree. The number of rows, however, are always fixed at 3.

In the current version, each column's index corresponds with the value of a key within the B-Tree. This means that the B-Tree must always insert the keys from 0-n (whether in order or not), with n being the total amount of keys to insert, for the matrix to generate properly.

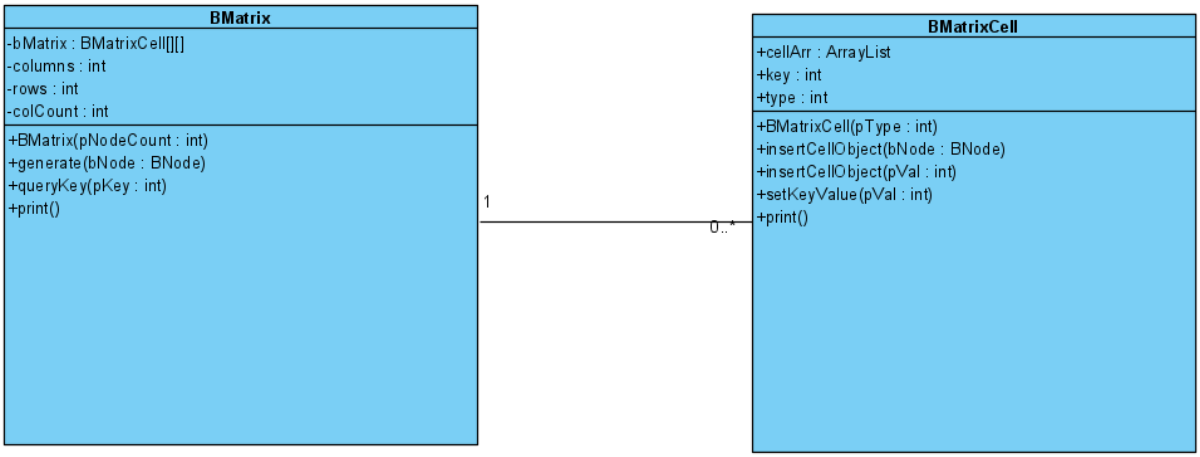
Each row corresponds to a different data point related the column's key. The first row contains the key's value. The second row contains a BMatrixCell object, which has the left and right sub-tree from the key's position. The third row contains a BMatrixCell object, which has the key's brothers (this referring to any other keys within the same node).

This BMatrix implementation allows for constant-time access to a key's left and right sub-tree. This is because a key's information can be directly accessed using the key as an index.

BMatrixCell

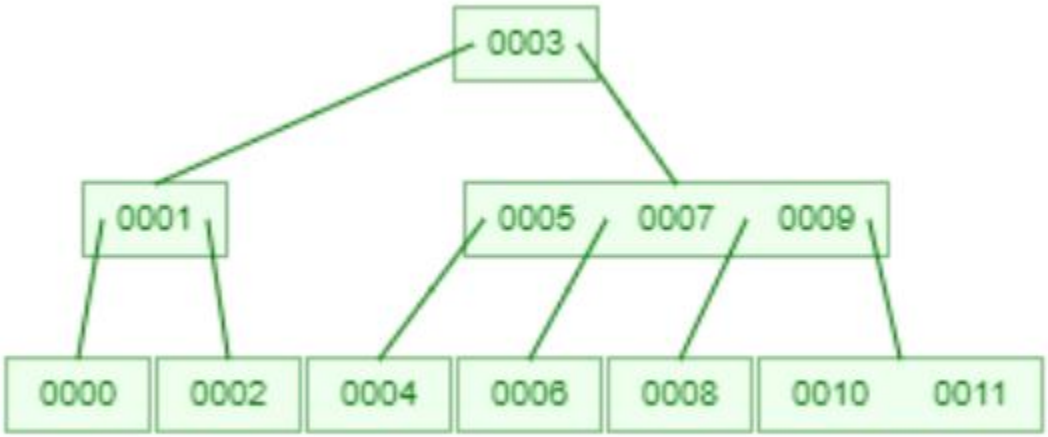
The MatrixCell class (named BMatrixCell), consists of three class members: a generic ArrayList, an integer "key" and an integer "type". The type can be a digit from 0 to 2, each corresponding to what each row in BMatrix would contain. If the type is 0, the ArrayList will remain uninitialized. If the type is 1, the ArrayList will be cast as a BNode ArrayList. If the type is 2, the ArrayList will be cast as an Integer ArrayList. In this way each cell in the matrix is capable of containing different information while being cast from the same object.

Class Diagram:



Example Test Case

Given the following B-Tree:



Generated from inserting 0 through 11 in order, with a minDegree of 2, such as:

```
BTree bTree = new BTree(2);
for(int index = 0;index<12;index++){
    bTree.insert(index);
}
```

The BTree's print gives the following:

```
Node Height - 0 | Key Count - 1
3
Node Height - 1 | Key Count - 1
1
Node Height - 2 | Key Count - 1
0
Node Height - 2 | Key Count - 1
2
Node Height - 1 | Key Count - 3
5 7 9
Node Height - 2 | Key Count - 1
4
Node Height - 2 | Key Count - 1
6
Node Height - 2 | Key Count - 1
8
Node Height - 2 | Key Count - 2
10 11
```

The generated matrix's print gives the following:

```
COLUMN 0---------
Key: 0
Left Tree Keys:
Right Tree Keys:
Key Brothers:
------------------
COLUMN 1---------
Key: 1
Left Tree Keys: 0
Right Tree Keys: 2
Key Brothers:
------------------
COLUMN 2---------
Key: 2
Left Tree Keys:
Right Tree Keys:
Key Brothers:
------------------
COLUMN 3---------
Key: 3
Left Tree Keys: 1
Right Tree Keys: 5 7 9
Key Brothers:
------------------
COLUMN 4---------
Key: 4
Left Tree Keys:
Right Tree Keys:
Key Brothers:
------------------
COLUMN 5---------
Key: 5
Left Tree Keys: 4
Right Tree Keys: 6
Key Brothers: 7 9
------------------
COLUMN 6---------
Key: 6
Left Tree Keys:
Right Tree Keys:
Key Brothers:
------------------
COLUMN 7---------
Key: 7
Left Tree Keys: 6
Right Tree Keys: 8
Key Brothers: 5 9
------------------
COLUMN 8---------
Key: 8
Left Tree Keys:
Right Tree Keys:
Key Brothers:
------------------
COLUMN 9---------
Key: 9
Left Tree Keys: 8
Right Tree Keys: 10 11
Key Brothers: 5 7
------------------
COLUMN 10---------
Key: 10
Left Tree Keys:
Right Tree Keys:
Key Brothers: 11
------------------
COLUMN 11---------
Key: 11
Left Tree Keys:
Right Tree Keys:
Key Brothers: 10
------------------
```

By querying the keys 5, 7 and 9 we get the following:

```
Query result for key 5:
Left Tree Keys: 4
Right Tree Keys: 6
Query result for key 7:
Left Tree Keys: 6
Right Tree Keys: 8
Query result for key 9:
Left Tree Keys: 8
Right Tree Keys: 10 11
```

For further testing, modify the *main* function in “BTree.java” with the desired values.