The Vector Collection

- java.util.vector
  - much like an array however vector is expandable
  - extra high-level operations make vector very flexible in use
  - vector can hold any subtype of Object.
- Operations
  - size() , capacity() - returns the number of elements in the collection
  - isEmpty()
  - setSize(int) – set the size, trancuting or expanding as necessary

Using Vector as a Stack

- Recall stack operations
  - push(Object), pop():Object, peek():Object, empty()
- Using a vector
  - myVector.addElement(Object) *push(Object)*
  - myVector.lastElement() *peek()*
  - myVector.removeElementAt(myVector.size() -1) *pop(): Object*

Using Vector as a Queue

- Queues allow the addition of elements on one end and the removal of elements from the other (FIFO)
  - push(Object), pop():Object, peek():Object
- Using a vector
  - myVector.addElement(Object) *push(Object)*
  - myVector.firstElement() *peek()*
  - myVector.removeElementAt(0) *pop():Object*

Using Vector as a Set

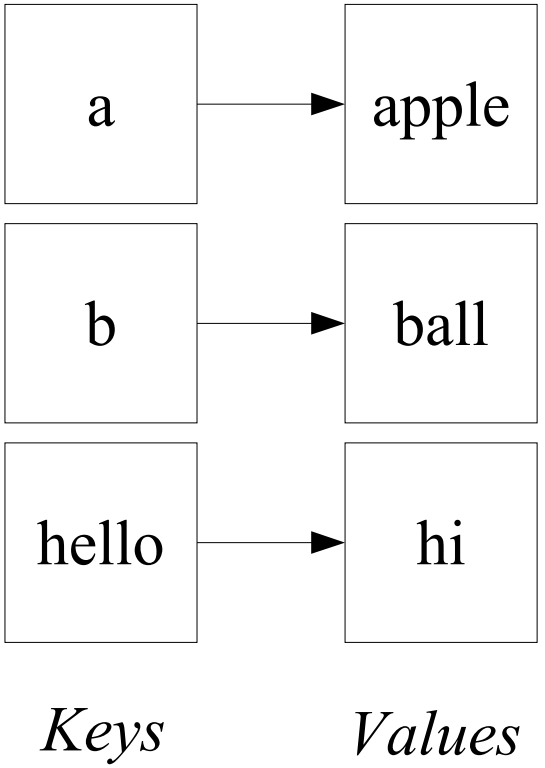
- A Set is a data structure that can hold an unordered set of values
  - add(Object), remove(Object), contains(Object):boolean
- Using a vector
  - myVector.addElement(Object) *add(Object)*
  - myVector.contains(Object) *contains(Object):boolean*
  - myVector.removeElement(Object) *remove(Object)*

Using Vector as a List

- A List allows the addition, removal and retrieval of elements at any location. Also the ability to find the location of an element
  - first(), last(),addFirst(Object), addLast(Object), contains (Object):boolean, removeFirst(), removeLast(), indexOf (Object):int, remove(int)
- Using a vector
  - myVector.firstElement() *first()*
  - myVector.lastElement() *last()*
  - myVector.indexOf(Object) *indexOf(Object)*
  - myVectore.removeElementAt(index) *remove(index)*

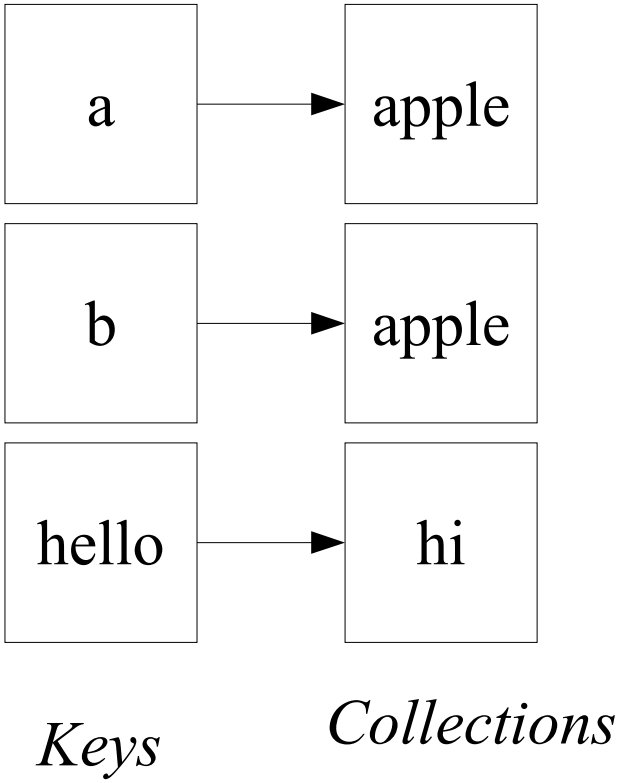
HashMap

- A collection values, each mapped to a *key*



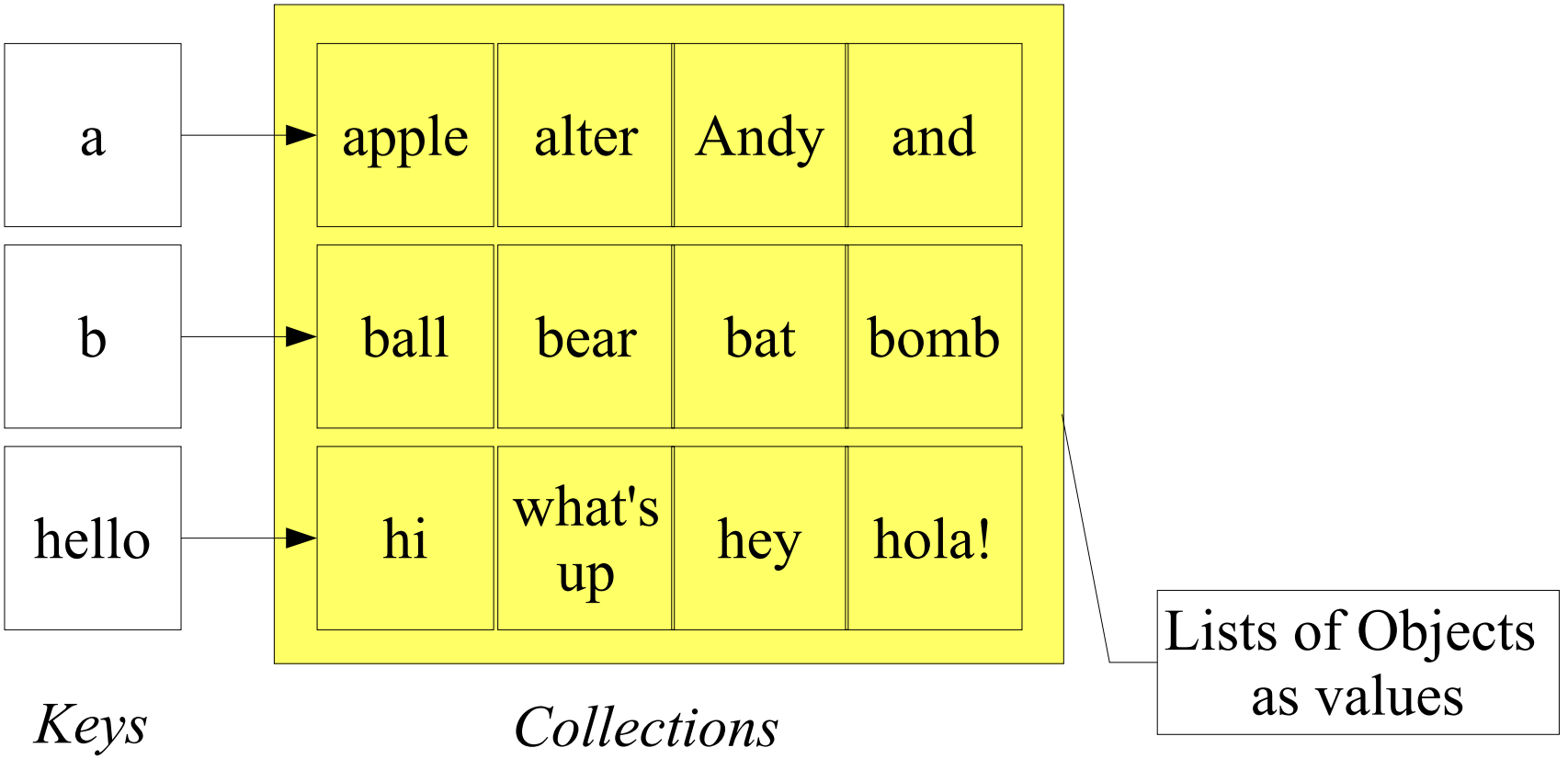
HashMap

- Keys have to be distinct!
- Values do not have to be distinct !



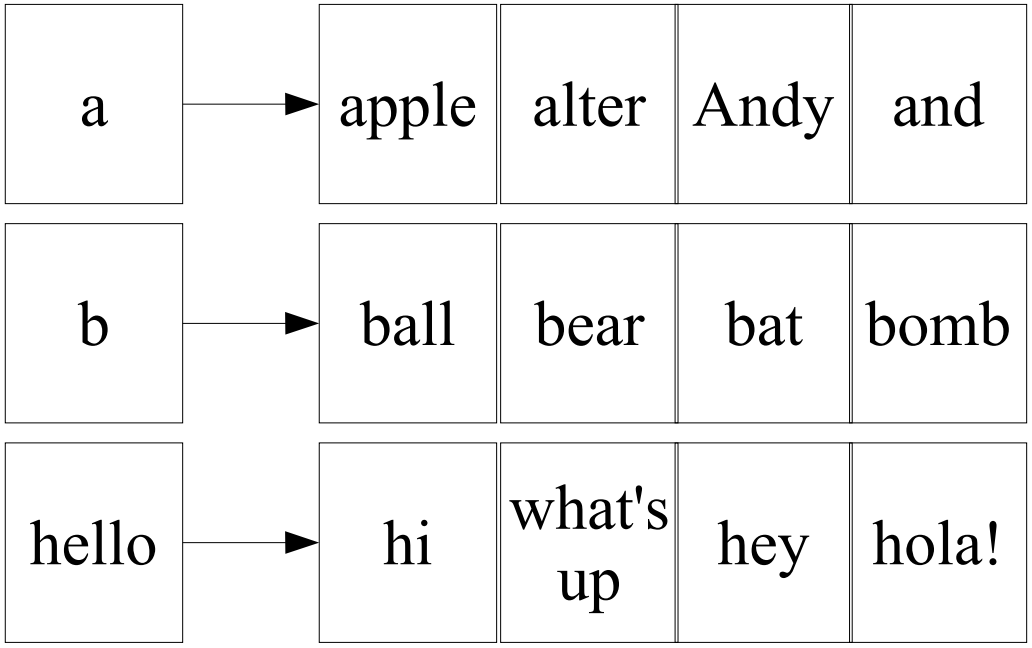
HashMap

- In Java both keys and values can be of type *Object*
- You can create interesting data structure !



HashMap

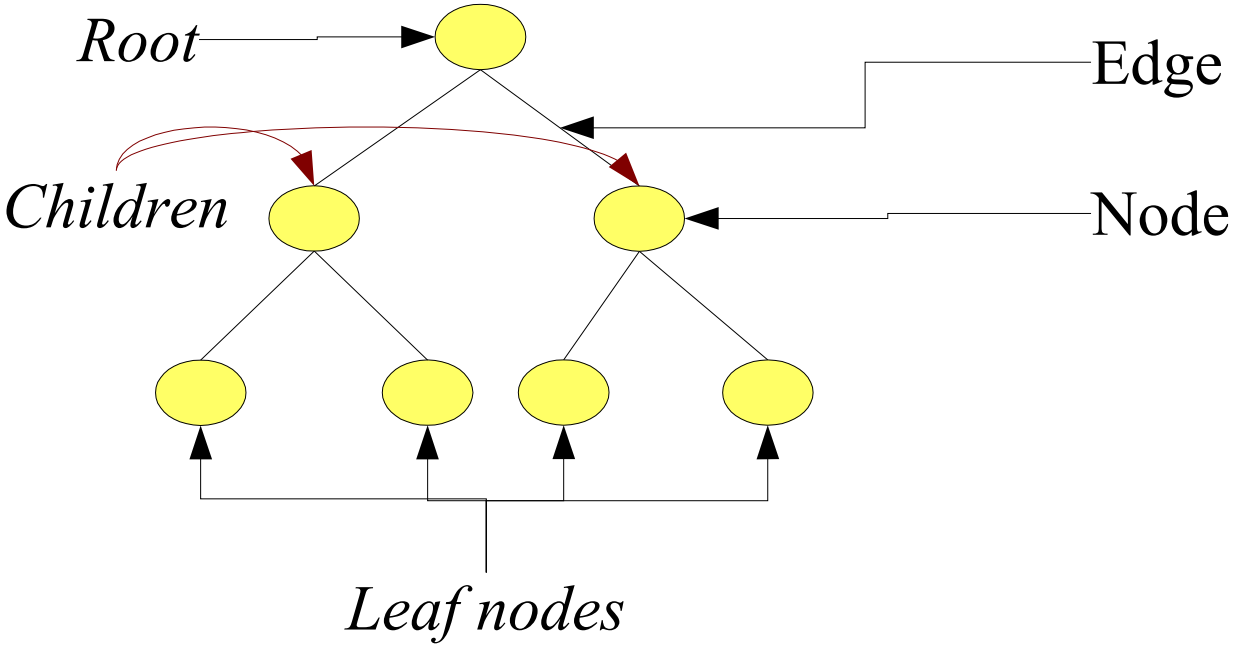
- Operations on HashMaps
  - containsKey(Object):boolean, containsValue(Object): boolean, put(Object, Object):Object.



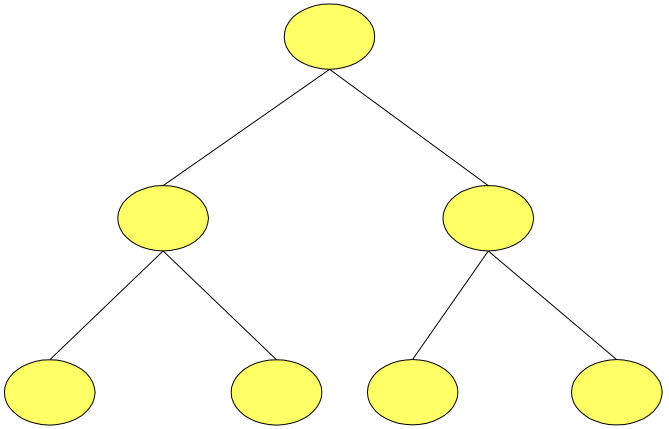

```
LinkedList aList = new LinkedList();
aList.add(“apple”);
aList.add(“alter”);
aList.add(“Andy”);
aList.add(“and”);
// initialize bList and helloList
HashMap hMap = new HashMap();
hMap.put(“a”, aList);
hMap.put(“b”, bList);
hMap.put(“hello”, helloList);
```

Trees

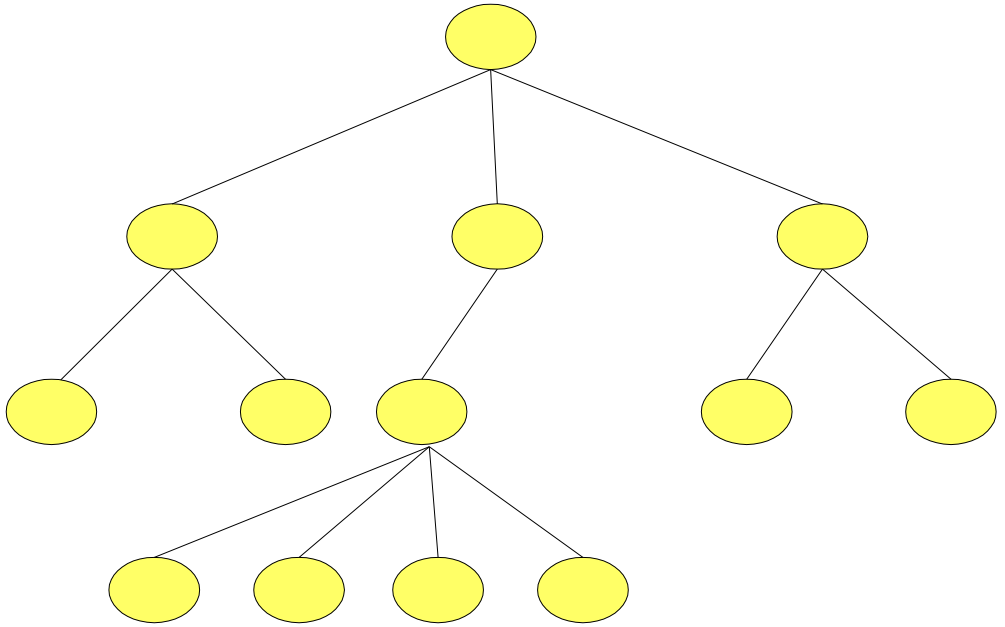
- Extensively used data structure
- A set of nodes (N) and edges (E).
  - A tree grows downwards



Examples of Trees

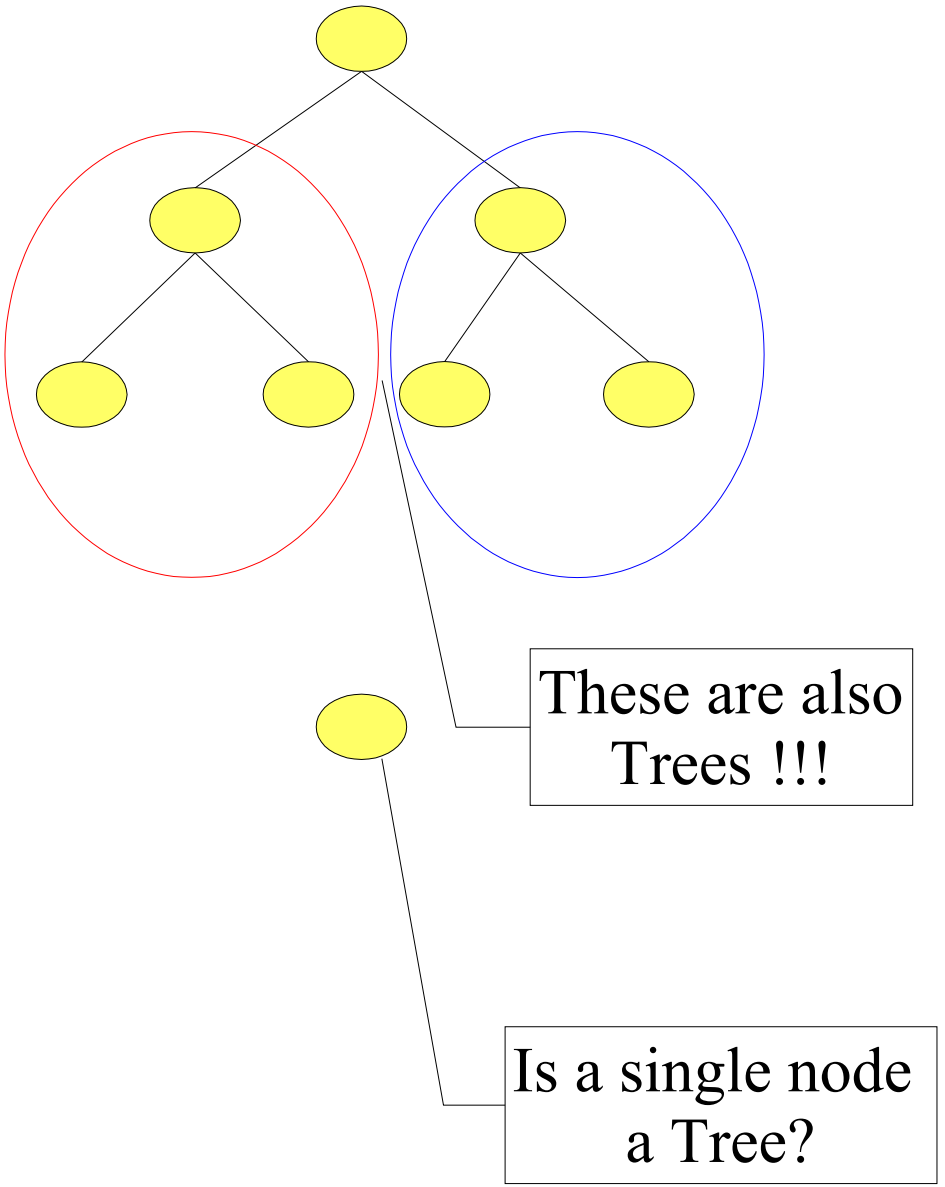


- Binary Tree:
  - each node has at most 2 children



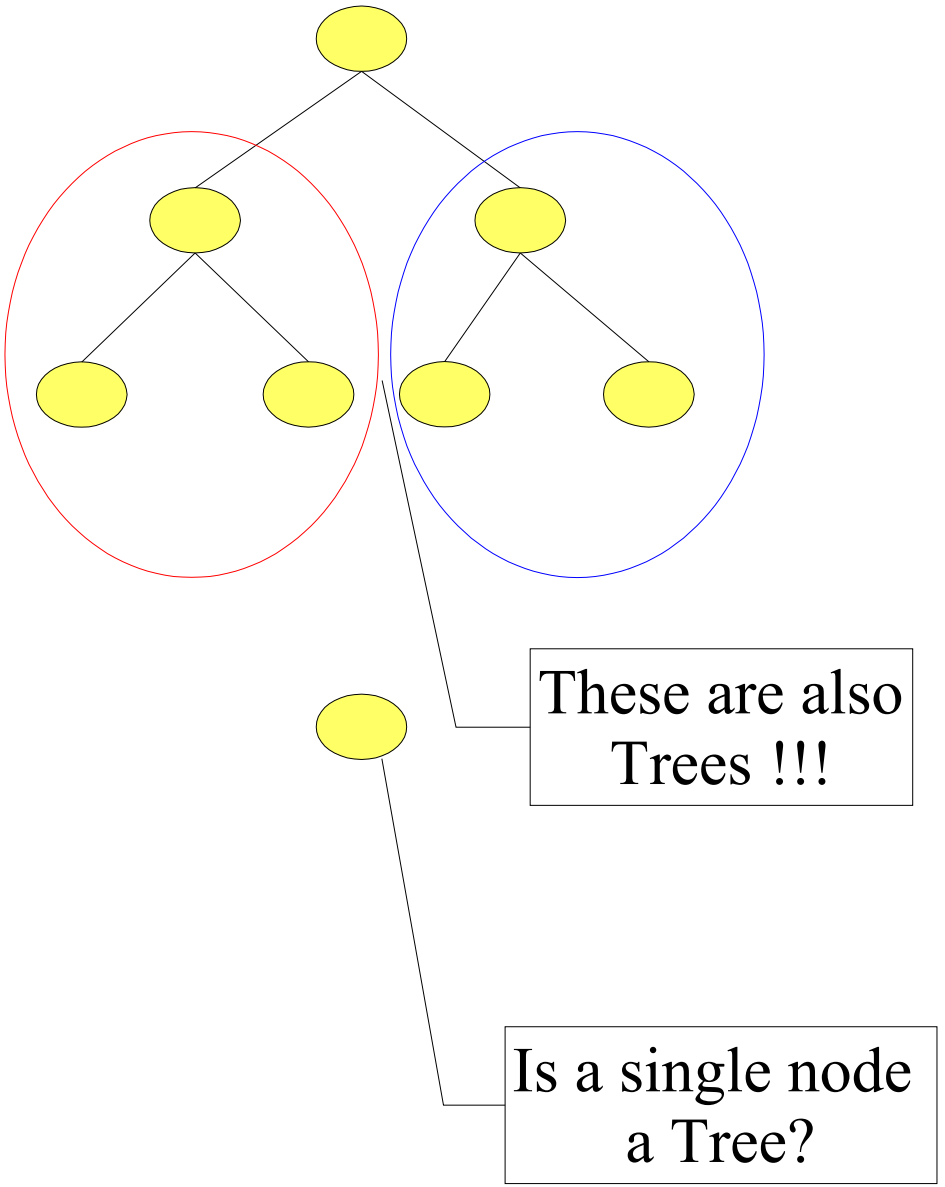
- N-ary trees:
  - each node has at most *n* number of children

Designing a binary tree



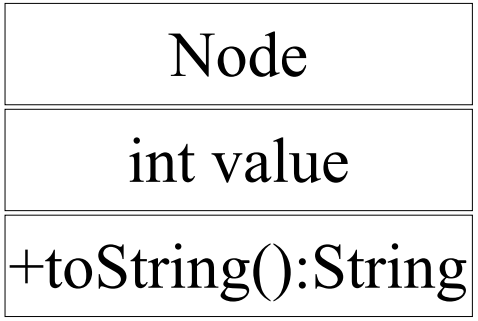
- Need to represent
  - nodes
  - edges
  - the whole tree
- Nodes
  - take at most 2 children that are themselves nodes.
  - We can also store some information on each node i.e. Root, Leaf, Color etc

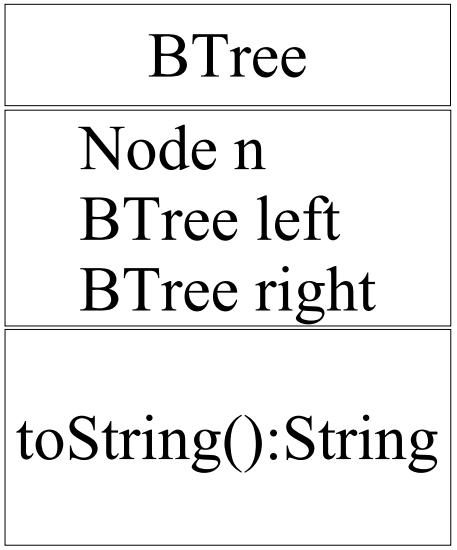
Designing a binary tree



- Everything is a Tree and a Tree can be
  - empty
  - one node
  - one node with one child (left or right)
  - one node and 2 children
- And each child is a .... TREE !!!!

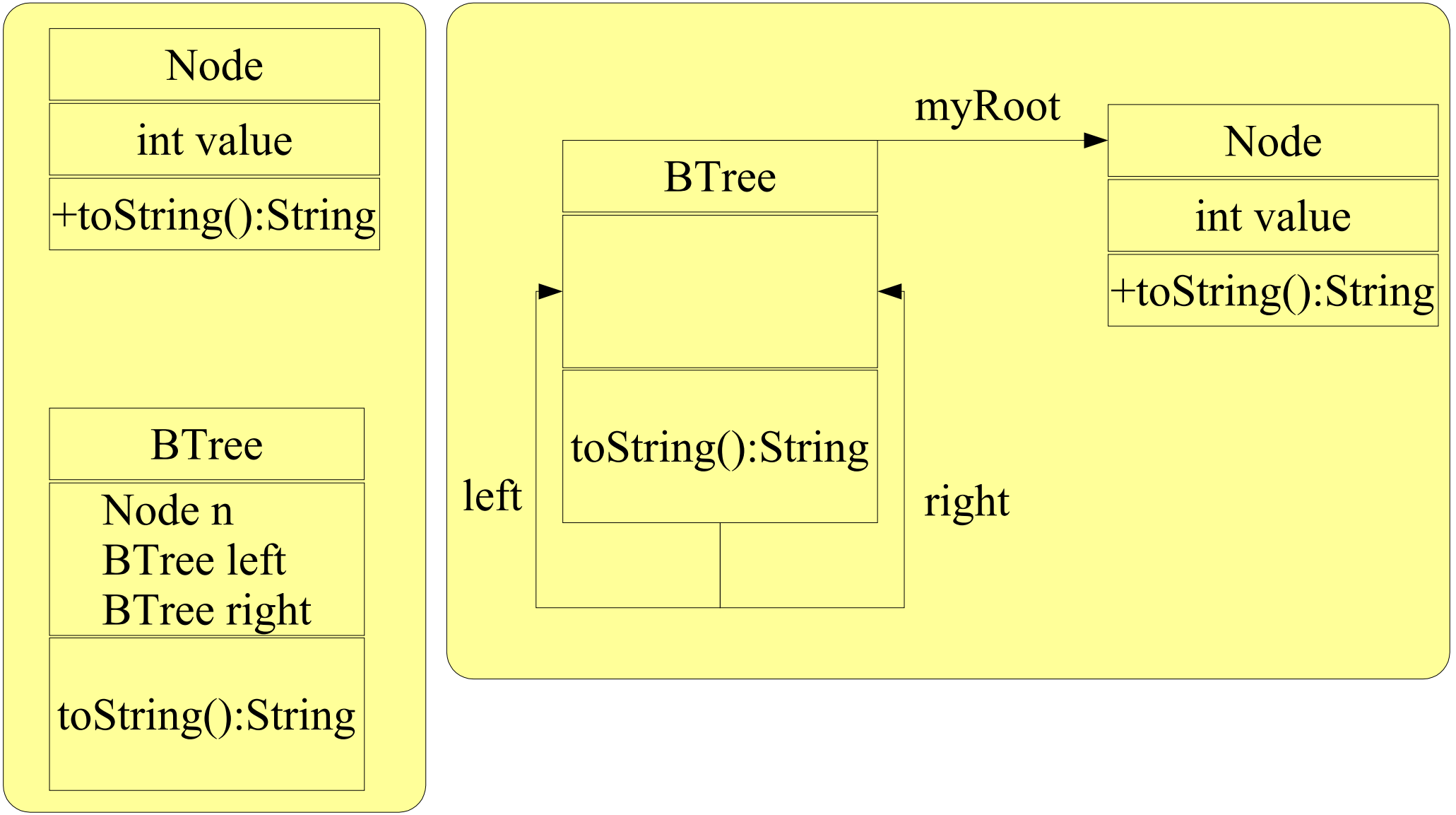
Designing a binary tree



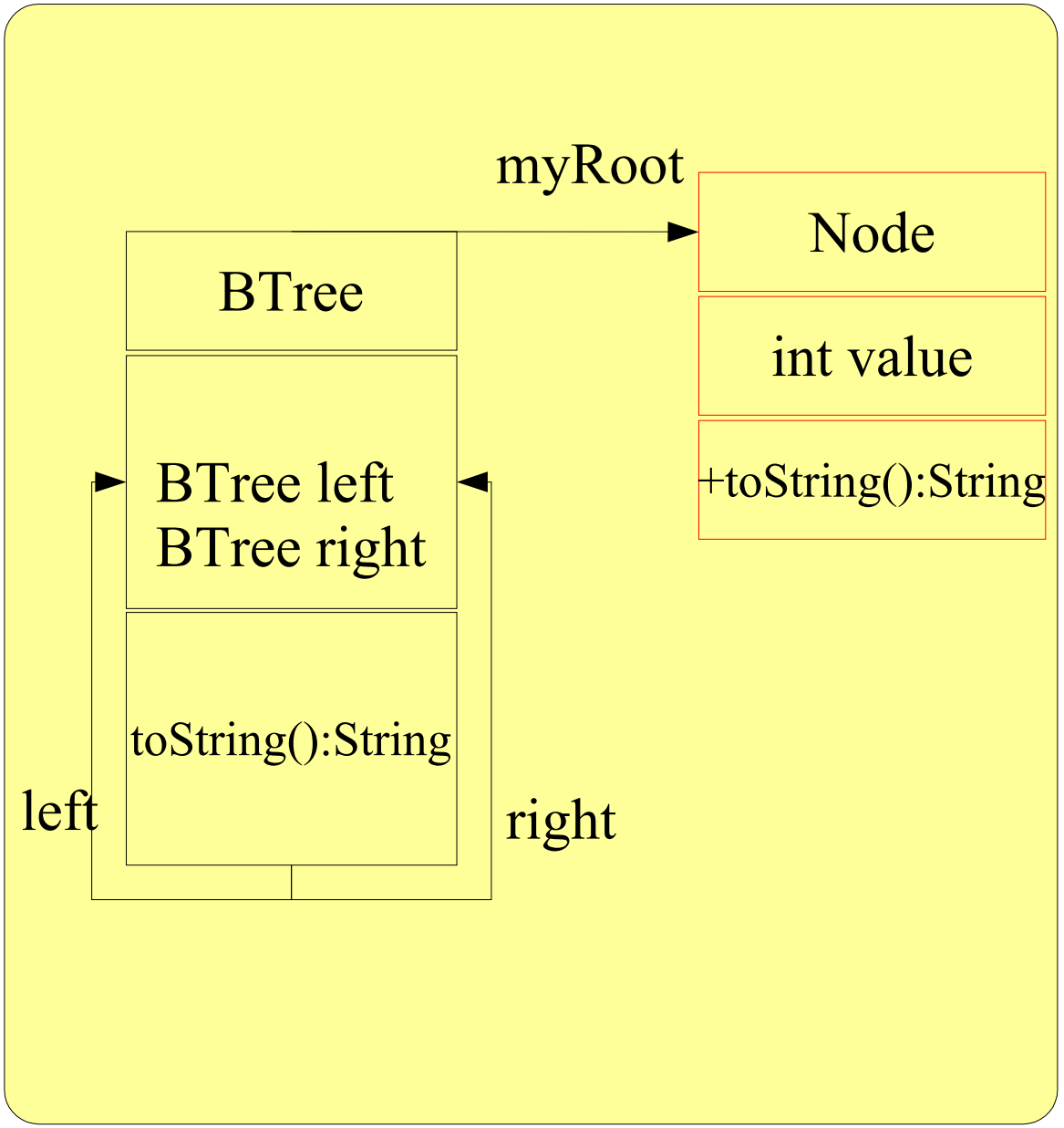


- Everything is a Tree and a Tree can be
  - empty
  - one node
  - one node with one child (left or right)
  - one node and 2 children
- And each child is a .... TREE !!!!

Diagram notation



Designing a binary tree

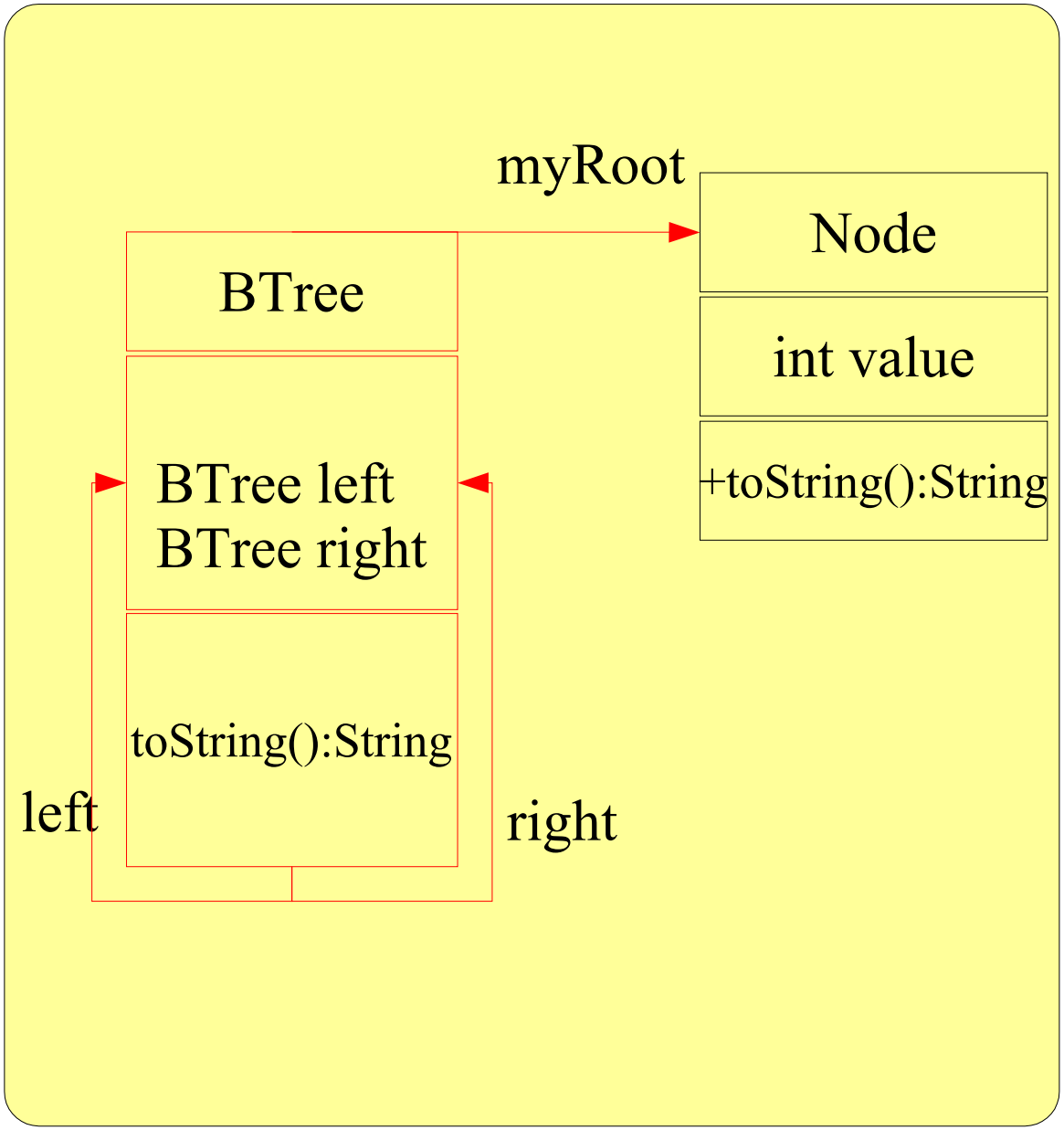



```
public class Node {
  int value;

  Node(int newVal){
    this.value = newVal;
  }

  public String toString(){
    return new String("NODE:
"+value +"\n");
  }
}
```

Designing a binary tree




```
public class BTree {
  Node myRoot;
  BTree left;
  BTree right;

  BTree(Node n) {
    this.myRoot = n;
    this.left = null;
    this.right = null;
  }

  BTree(Node n, BTree left){
    this.myRoot = n;
    this.left = left;
    this.right = null;
  }

  BTree(Node n, BTree left,
        BTree right){
    this.myRoot = n;
    this.left = left;
    this.right = right;
  }
```

Printing back the Tree

- Flatten
  - walk the tree and print the values on each node.
  - Order
    - left subtree
    - node
    - right subtree