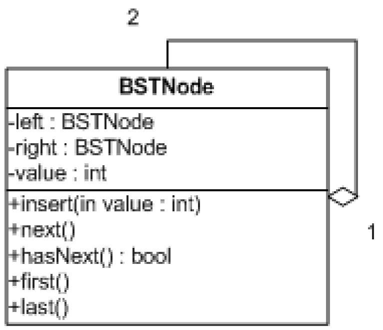
Question 3 (v)

(a) For the next example, we decided to develop a way to iterate over a binary search tree, a simple data structure that can be used to quickly search for an item.

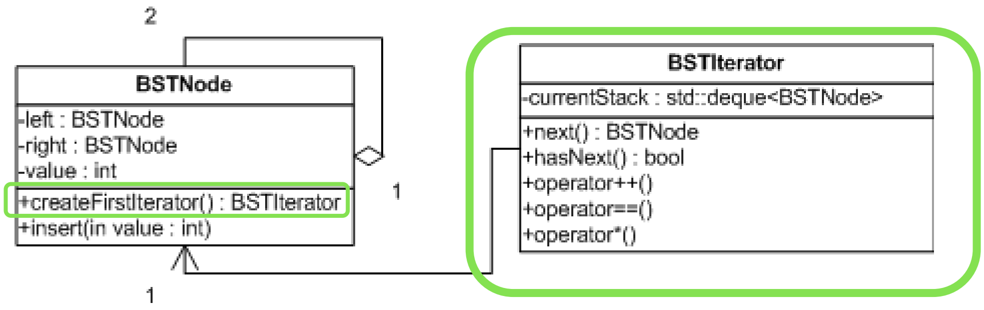
Binary search trees aren't often used if we require to access each item in a sequence, so it isn't effective to include a whole bunch of functions in the binary search tree node class to aid with traversal (such as prev, next, first, last, etc.).

Therefore, to create a convenient interface that the client can use to traverse all the nodes in the tree when he or she needs to is done using the **Iterator** pattern. Examples of this include when client programmer may want to print out all the values in the tree.

The following UML diagram is an example of what the class layout might look like without the iterator. The actual node class provides a many operations that can be used to iterate through the tree. This isn't usually a problem since a class doesn't carry around its function declarations, but it may still need to include a bunch of data members used to keep track of any traversals.



(b) The next UML diagram includes the iterator class that can be used to traverse through the binary search tree. This class not only removes the burden mentioned above from the BST nodes, but also allows for easier addition or modification to the current methods available to traverse the tree. The added components are highlighted in green.


2
BSTNode
-left : BSTNode
-right : BSTNode
-value : int
+createFirstIterator() : BSTIterator
+insert(in value : int)
1
1
BSTIterator
-currentStack : std::deque<BSTNode>
+next() : BSTNode
+hasNext() : bool
+operator++()
+operator==()
+operator*()