Tyler Bartlett

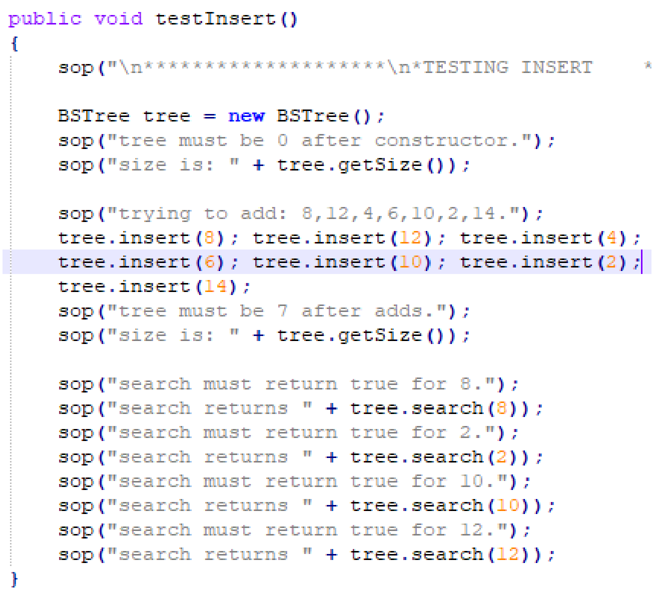
5/20/18

CS260

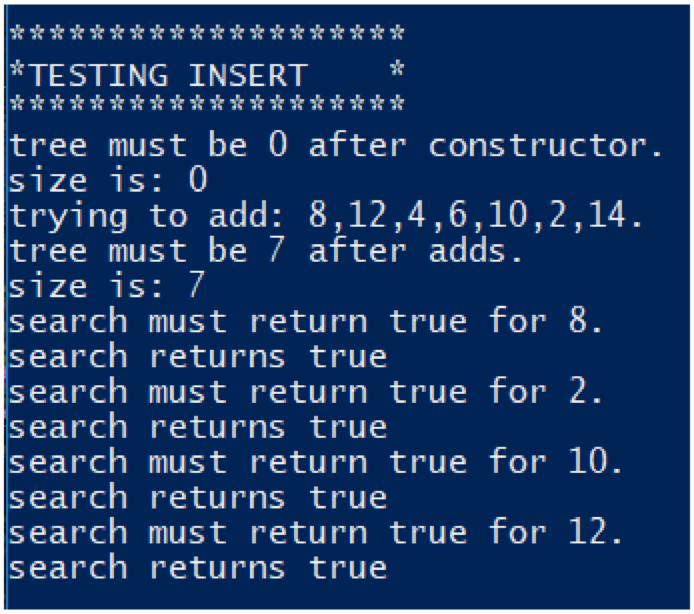
Lab 3 Write Up

Lab 3 covered Binary Search Trees. The Lab handout contained a C++ class definition for a generic TreeNode. I started the lab by copying the TreeNode code to a java file and rewriting it in Java syntax. I created the skeleton of the BSTree file containing empty methods that I needed to write and compiles everything to make sure I was good to go for starting to write the methods. I started with the insertion method and was trying to think of a way to do it recursively. Google lead me to sanfoundry where the author of that code used two insert methods, one that receives data to add, then the second received data to and the root. I used this idea to recursively build my insert function. I couldn't not think of a way to recursively add nodes to the tree when the insert only received an element. Not being able to overload operators in java caused a problem with my comparisons of generic data T. I had to lookup how to deal with this and found an example on stackoverflow that included "extends Comparable<T>". this enabled me to compare data correctly.

After I finished the insert method, I began writing the test file to test my code. Stacia Fry emailed out an updated tester file in C++ so I began converting all of it to Java in my SystemTest.java class. I have the code output to console and I have been checking it that way to see if the tests are passing. I print what the result should be and what the result is. If I saw any incorrect output, I went back to the code and made corrections. The test insert method used getSize and search methods from the BST so I worked on those next before testing my insert method. The following screen shots are my insert test and the results. I wrote a short method to short hand System.out.Println to sop which you will see in all the following screen shots. It is a much quicker way to output to the console than writing it out every time.

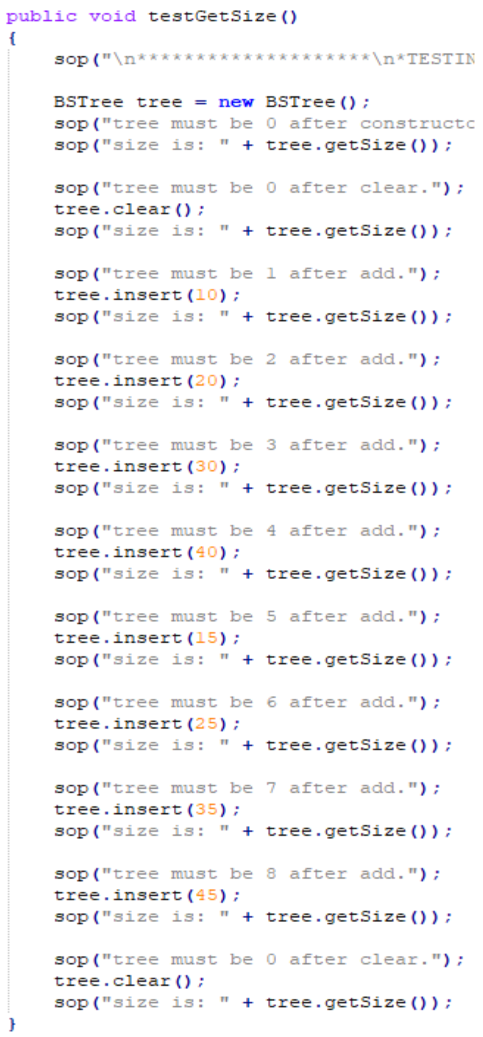


```java
public void testInsert()
{
    sop("\n********************\n*TESTING INSERT    *

    BSTree tree = new BSTree();
    sop("tree must be 0 after constructor.");
    sop("size is: " + tree.getSize());

    sop("trying to add: 8,12,4,6,10,2,14.");
    tree.insert(8); tree.insert(12); tree.insert(4);
    tree.insert(6); tree.insert(10); tree.insert(2);
    tree.insert(14);
    sop("tree must be 7 after adds.");
    sop("size is: " + tree.getSize());

    sop("search must return true for 8.");
    sop("search returns " + tree.search(8));
    sop("search must return true for 2.");
    sop("search returns " + tree.search(2));
    sop("search must return true for 10.");
    sop("search returns " + tree.search(10));
    sop("search must return true for 12.");
    sop("search returns " + tree.search(12));
}
```

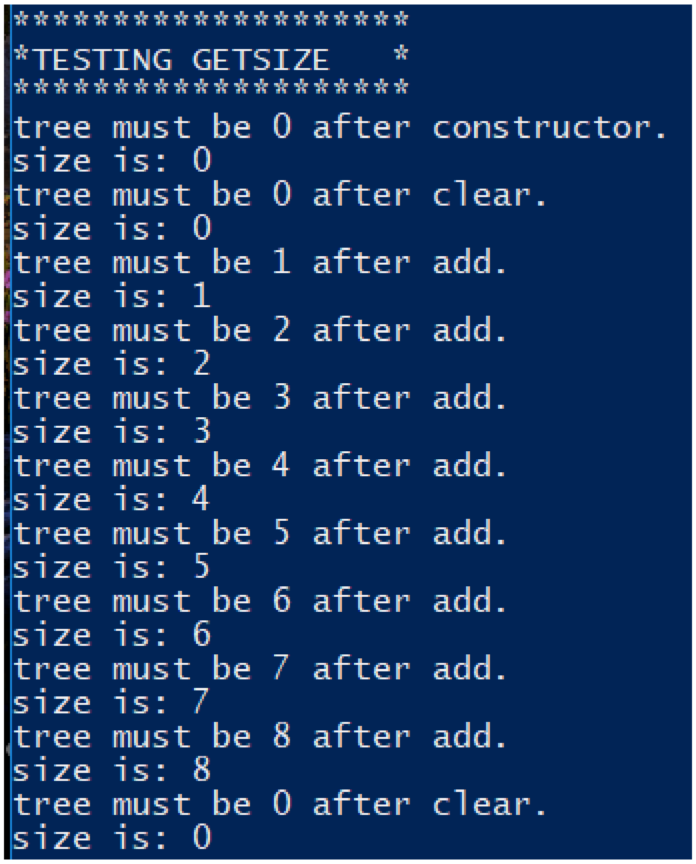


```
********************
*TESTING INSERT    *
********************
tree must be 0 after constructor.
size is: 0
trying to add: 8,12,4,6,10,2,14.
tree must be 7 after adds.
size is: 7
search must return true for 8.
search returns true
search must return true for 2.
search returns true
search must return true for 10.
search returns true
search must return true for 12.
search returns true
```

I tested my get size method next and converted all the C++ test code to Java in my tester file. The following screen shots are of my test getSize method and the results from the test.

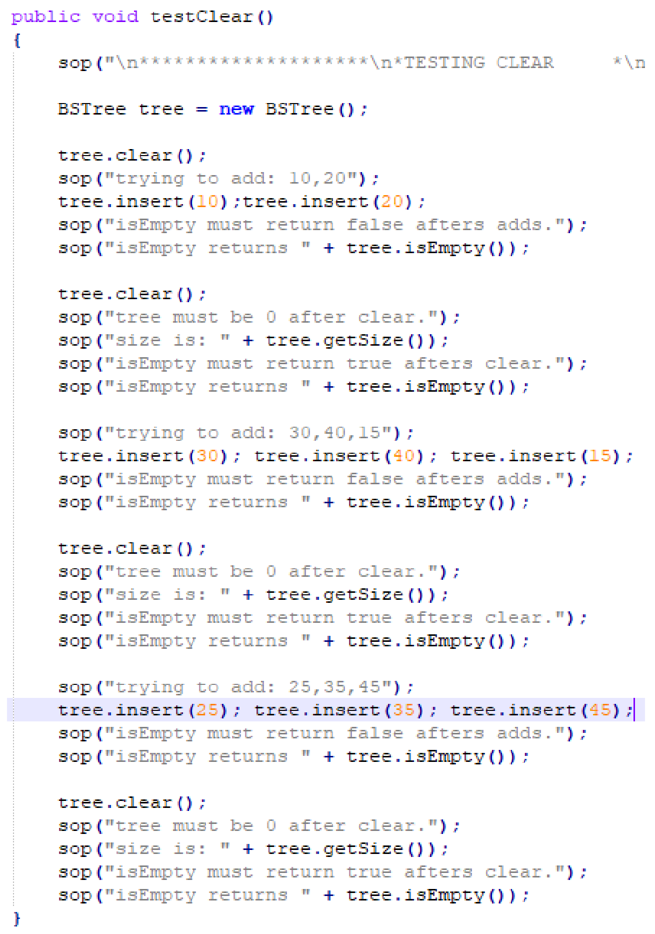


```java
public void testGetSize()
{
    sop("\n********************\n*TESTIN

    BSTree tree = new BSTree();
    sop("tree must be 0 after constructo
    sop("size is: " + tree.getSize());

    sop("tree must be 0 after clear.");
    tree.clear();
    sop("size is: " + tree.getSize());

    sop("tree must be 1 after add.");
    tree.insert(10);
    sop("size is: " + tree.getSize());

    sop("tree must be 2 after add.");
    tree.insert(20);
    sop("size is: " + tree.getSize());

    sop("tree must be 3 after add.");
    tree.insert(30);
    sop("size is: " + tree.getSize());

    sop("tree must be 4 after add.");
    tree.insert(40);
    sop("size is: " + tree.getSize());

    sop("tree must be 5 after add.");
    tree.insert(15);
    sop("size is: " + tree.getSize());

    sop("tree must be 6 after add.");
    tree.insert(25);
    sop("size is: " + tree.getSize());

    sop("tree must be 7 after add.");
    tree.insert(35);
    sop("size is: " + tree.getSize());

    sop("tree must be 8 after add.");
    tree.insert(45);
    sop("size is: " + tree.getSize());

    sop("tree must be 0 after clear.");
    tree.clear();
    sop("size is: " + tree.getSize());
}
```

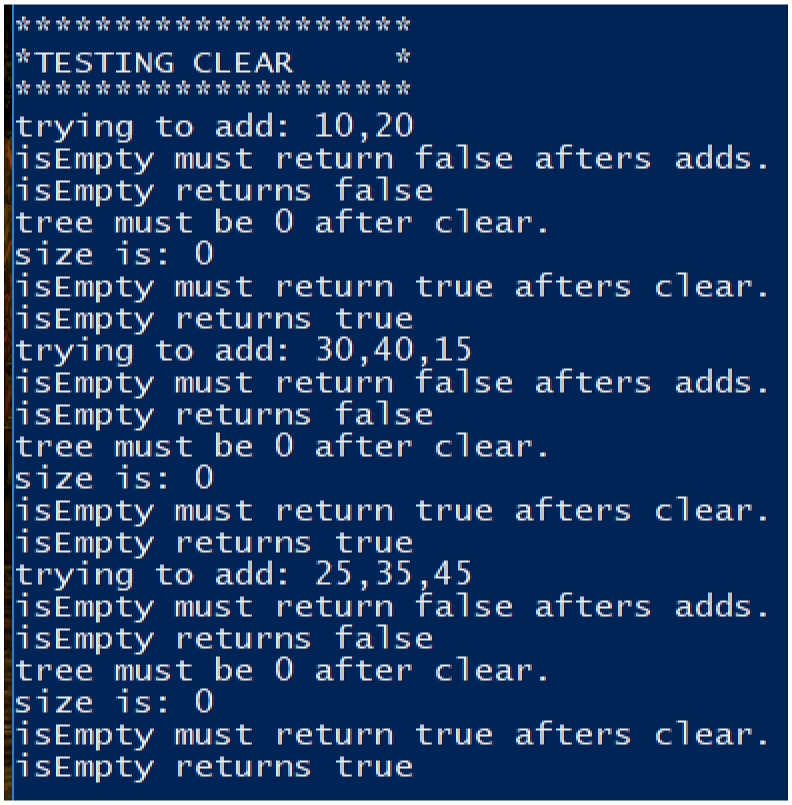


```
********************
*TESTING GETSIZE   *
********************
tree must be 0 after constructor.
size is: 0
tree must be 0 after clear.
size is: 0
tree must be 1 after add.
size is: 1
tree must be 2 after add.
size is: 2
tree must be 3 after add.
size is: 3
tree must be 4 after add.
size is: 4
tree must be 5 after add.
size is: 5
tree must be 6 after add.
size is: 6
tree must be 7 after add.
size is: 7
tree must be 8 after add.
size is: 8
tree must be 0 after clear.
size is: 0
```

I worked on testing the clear method next. It turns out the garbage collector in Java is amazing. when you dereference an object, the garbage collector will wipe out everything deference, so you do not leave a bunch of stuff in memory. By setting root to null, the garbage collector takes down the entire tree. Here are screen shots of the test and the results.

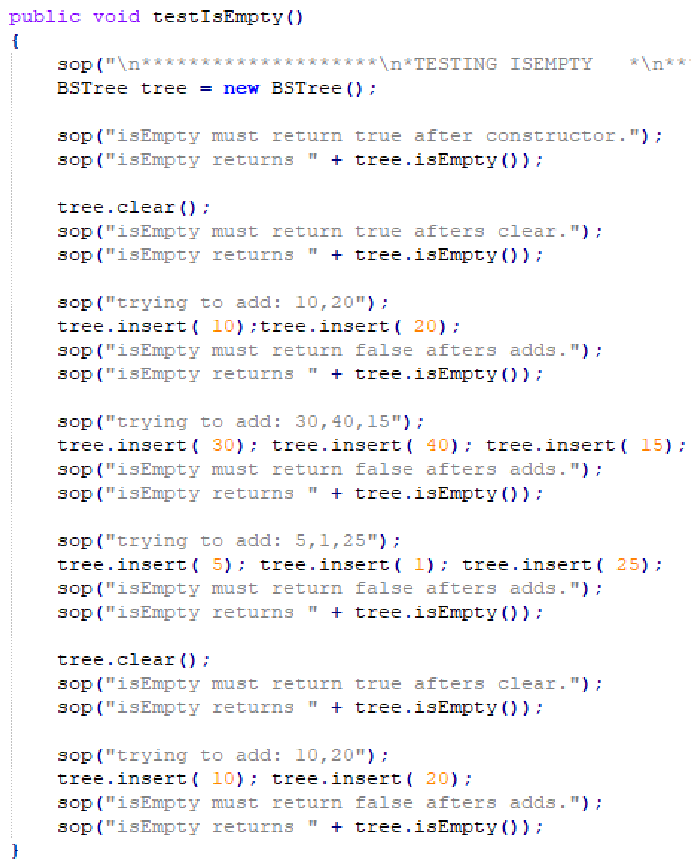


```java
public void testClear()
{
    sop("\n********************\n*TESTING CLEAR     *\n

    BSTree tree = new BSTree();

    tree.clear();
    sop("trying to add: 10,20");
    tree.insert(10);tree.insert(20);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());

    tree.clear();
    sop("tree must be 0 after clear.");
    sop("size is: " + tree.getSize());
    sop("isEmpty must return true afters clear.");
    sop("isEmpty returns " + tree.isEmpty());

    sop("trying to add: 30,40,15");
    tree.insert(30); tree.insert(40); tree.insert(15);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());

    tree.clear();
    sop("tree must be 0 after clear.");
    sop("size is: " + tree.getSize());
    sop("isEmpty must return true afters clear.");
    sop("isEmpty returns " + tree.isEmpty());

    sop("trying to add: 25,35,45");
    tree.insert(25); tree.insert(35); tree.insert(45);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());

    tree.clear();
    sop("tree must be 0 after clear.");
    sop("size is: " + tree.getSize());
    sop("isEmpty must return true afters clear.");
    sop("isEmpty returns " + tree.isEmpty());
}
```

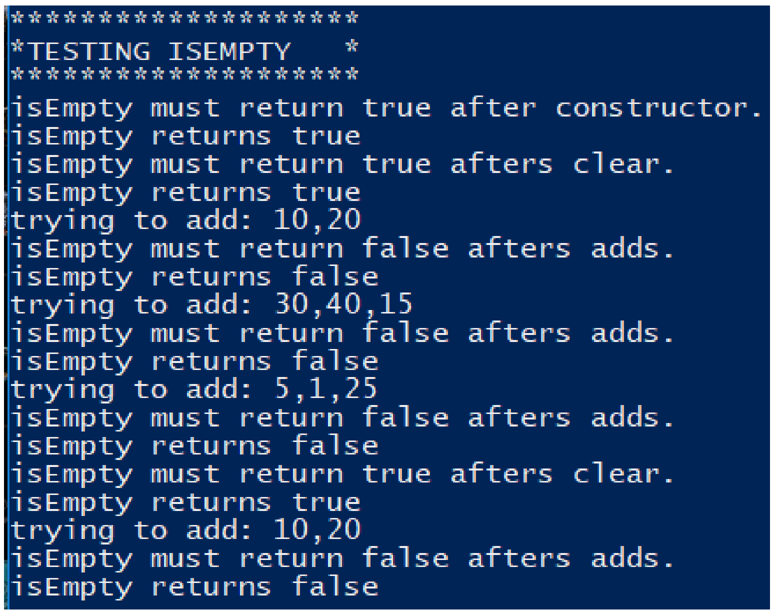


```
********************
*TESTING CLEAR     *
********************
trying to add: 10,20
isEmpty must return false afters adds.
isEmpty returns false
tree must be 0 after clear.
size is: 0
isEmpty must return true afters clear.
isEmpty returns true
trying to add: 30,40,15
isEmpty must return false afters adds.
isEmpty returns false
tree must be 0 after clear.
size is: 0
isEmpty must return true afters clear.
isEmpty returns true
trying to add: 25,35,45
isEmpty must return false afters adds.
isEmpty returns false
tree must be 0 after clear.
size is: 0
isEmpty must return true afters clear.
isEmpty returns true
```

The following screen shots are my test for my isEmpty method and the results of the test.

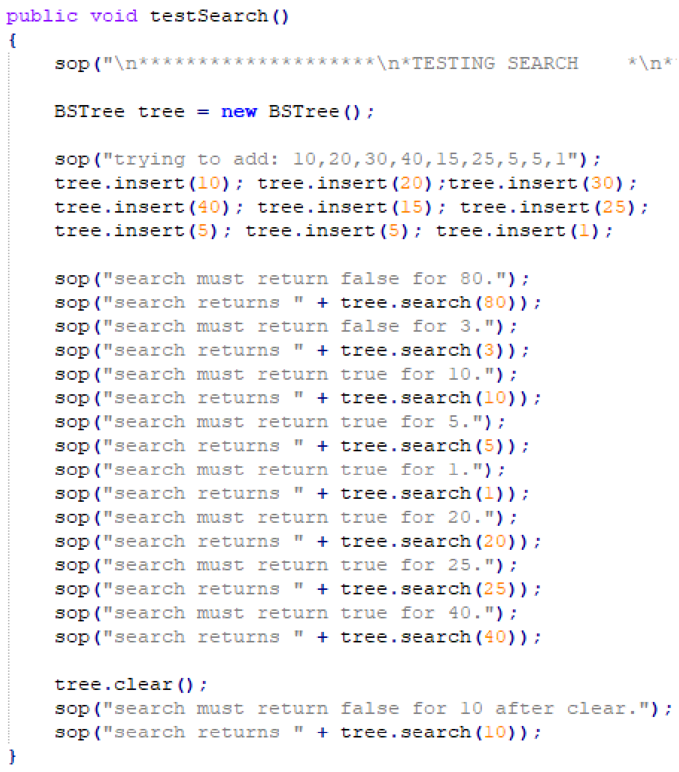


```java
public void testIsEmpty()
{
    sop("\n********************\n*TESTING ISEMPTY   *\n**
    BSTree tree = new BSTree();

    sop("isEmpty must return true after constructor.");
    sop("isEmpty returns " + tree.isEmpty());

    tree.clear();
    sop("isEmpty must return true afters clear.");
    sop("isEmpty returns " + tree.isEmpty());

    sop("trying to add: 10,20");
    tree.insert( 10);tree.insert( 20);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());

    sop("trying to add: 30,40,15");
    tree.insert( 30); tree.insert( 40); tree.insert( 15);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());

    sop("trying to add: 5,1,25");
    tree.insert( 5); tree.insert( 1); tree.insert( 25);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());

    tree.clear();
    sop("isEmpty must return true afters clear.");
    sop("isEmpty returns " + tree.isEmpty());

    sop("trying to add: 10,20");
    tree.insert( 10); tree.insert( 20);
    sop("isEmpty must return false afters adds.");
    sop("isEmpty returns " + tree.isEmpty());
}
```

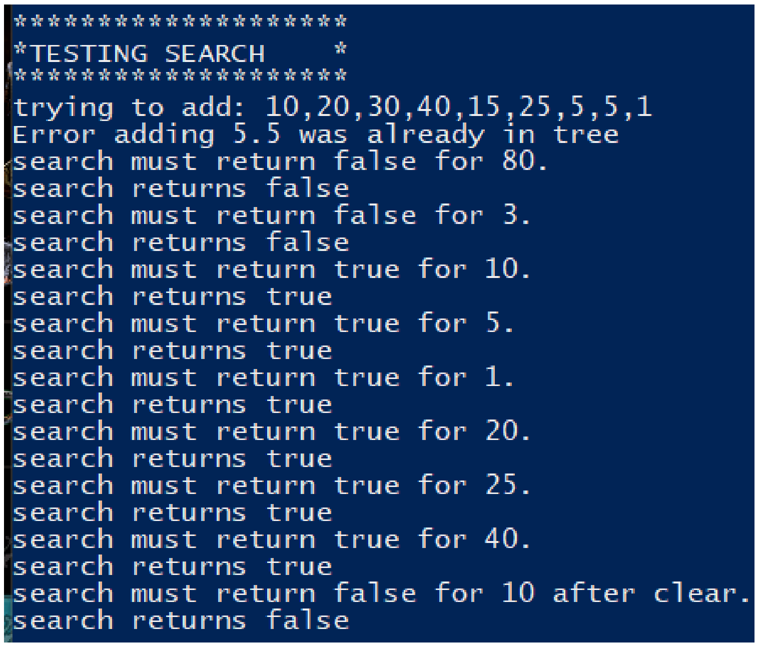


```
********************
*TESTING ISEMPTY   *
********************
isEmpty must return true after constructor.
isEmpty returns true
isEmpty must return true afters clear.
isEmpty returns true
trying to add: 10,20
isEmpty must return false afters adds.
isEmpty returns false
trying to add: 30,40,15
isEmpty must return false afters adds.
isEmpty returns false
trying to add: 5,1,25
isEmpty must return false afters adds.
isEmpty returns false
isEmpty must return true afters clear.
isEmpty returns true
trying to add: 10,20
isEmpty must return false afters adds.
isEmpty returns false
```

The following screen shots are the test for my search method and the results of the test.

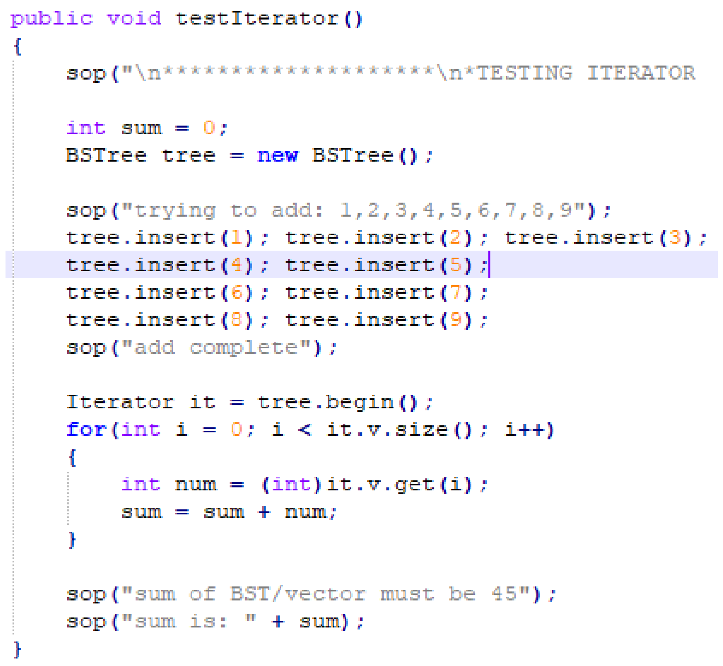


```
public void testSearch()
{
    sop("\n********************\n*TESTING SEARCH    *\n*

    BSTree tree = new BSTree();

    sop("trying to add: 10,20,30,40,15,25,5,5,1");
    tree.insert(10); tree.insert(20);tree.insert(30);
    tree.insert(40); tree.insert(15); tree.insert(25);
    tree.insert(5); tree.insert(5); tree.insert(1);

    sop("search must return false for 80.");
    sop("search returns " + tree.search(80));
    sop("search must return false for 3.");
    sop("search returns " + tree.search(3));
    sop("search must return true for 10.");
    sop("search returns " + tree.search(10));
    sop("search must return true for 5.");
    sop("search returns " + tree.search(5));
    sop("search must return true for 1.");
    sop("search returns " + tree.search(1));
    sop("search must return true for 20.");
    sop("search returns " + tree.search(20));
    sop("search must return true for 25.");
    sop("search returns " + tree.search(25));
    sop("search must return true for 40.");
    sop("search returns " + tree.search(40));

    tree.clear();
    sop("search must return false for 10 after clear.");
    sop("search returns " + tree.search(10));
}
```

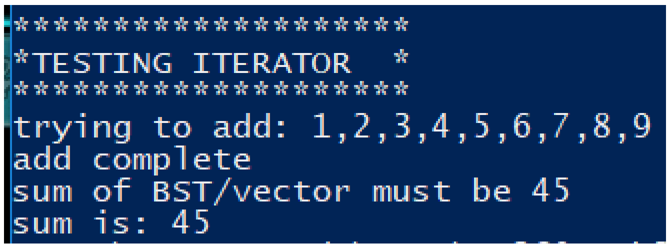


```
********************
*TESTING SEARCH    *
********************
trying to add: 10,20,30,40,15,25,5,5,1
Error adding 5.5 was already in tree
search must return false for 80.
search returns false
search must return false for 3.
search returns false
search must return true for 10.
search returns true
search must return true for 5.
search returns true
search must return true for 1.
search returns true
search must return true for 20.
search returns true
search must return true for 25.
search returns true
search must return true for 40.
search returns true
search must return false for 10 after clear.
search returns false
```

I am in a Discord server with many of the other students in class and we worked together a bit on the lab. When it became time to work on my iterator code, I did not have a fey strong grasp on how it worked. They helped explain it a bit to me. I got the operator working to the minimum requirement for the code to run but I did not work on any of the operators. I would have to create some methods to do effectively same thing as the operators because Java does not support overloading of operators. The following screen shots are the test for my iterator and the results of the test.



```
public void testIterator()
{
    sop("\n********************\n*TESTING ITERATOR

    int sum = 0;
    BSTree tree = new BSTree();

    sop("trying to add: 1,2,3,4,5,6,7,8,9");
    tree.insert(1); tree.insert(2); tree.insert(3);
    tree.insert(4); tree.insert(5);
    tree.insert(6); tree.insert(7);
    tree.insert(8); tree.insert(9);
    sop("add complete");

    Iterator it = tree.begin();
    for(int i = 0; i < it.v.size(); i++)
    {
        int num = (int)it.v.get(i);
        sum = sum + num;
    }

    sop("sum of BST/vector must be 45");
    sop("sum is: " + sum);
}
```



```
********************
*TESTING ITERATOR  *
********************
trying to add: 1,2,3,4,5,6,7,8,9
add complete
sum of BST/vector must be 45
sum is: 45
```

I played around with the heightFix method a little bit using an example Megan shared but I was not able to get it to work. It set the height of the root to one but the height of the rest of my nodes remained at 0.

<u>Resources:</u>

My peers on discord

https://stackoverflow.com/questions/11263244/java-how-do-i-implement-a-generic-binary-search-tree

https://www.sanfoundry.com/java-program-implement-binary-search-tree/

https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html