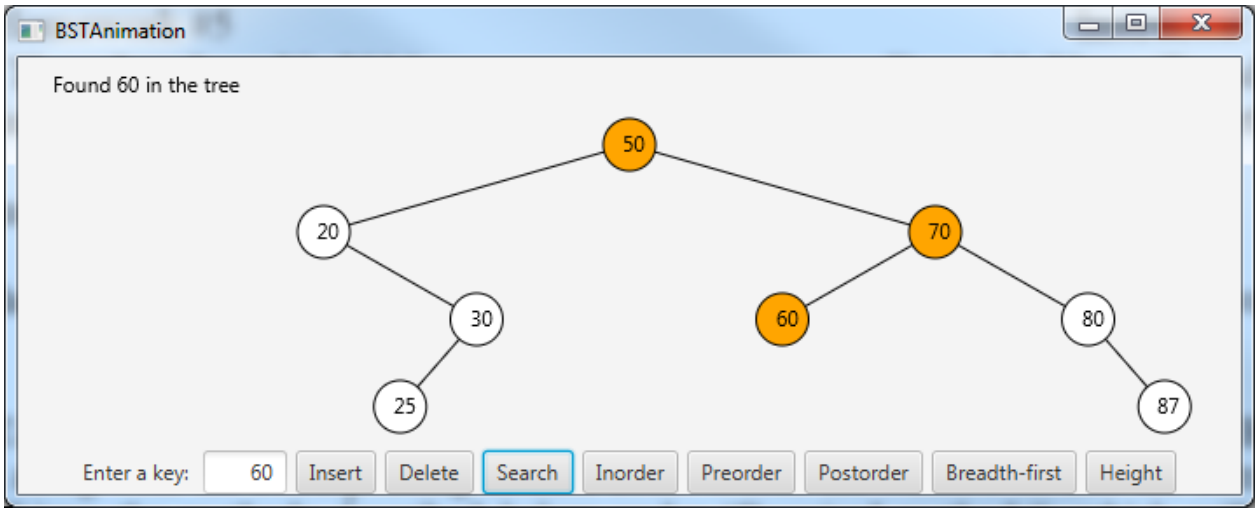
CSC 364 Homework #4 Instructor: Jeff Ward
Assigned: Thursday, February 23, 2017 Due: 11:59pm, Tuesday, March 21
Covers: Binary search trees Worth 40 points

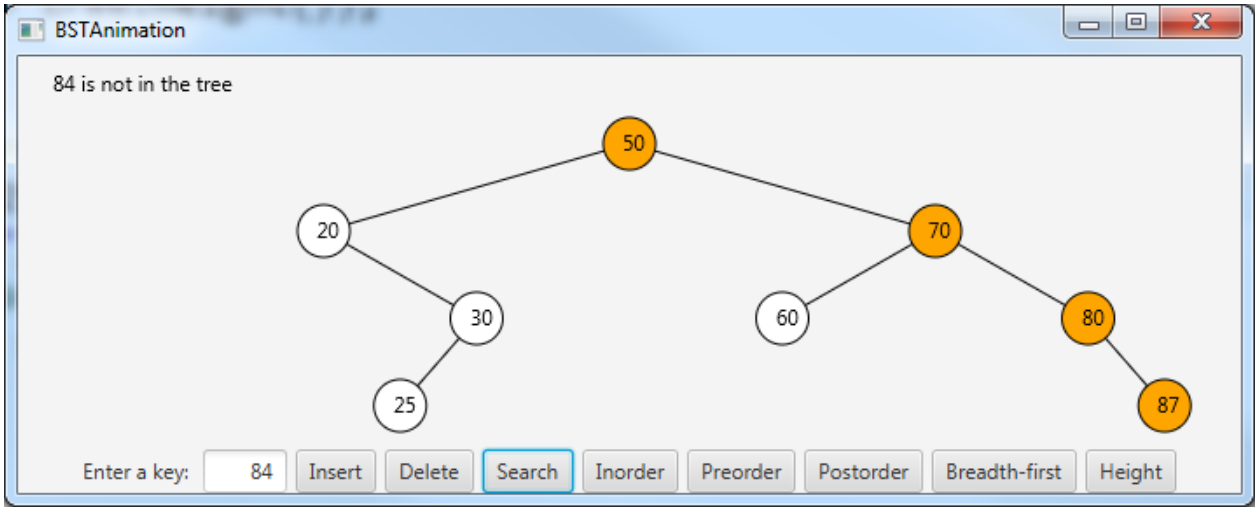
This assignment combines Programming Exercises 25.1 and 25.13 from the textbook.

Download the following files from Blackboard: BST.java, BSTAnimation.java, BTView.java, Tree.java.

The task:
Modify BSTAnimation.java to add six new buttons – Search, Inorder, Preorder, Postorder, Breadth-First, and Height, along with the corresponding listener code. Widen the window so that the extra buttons fit. Add to BST.java the methods described below under "Required methods", which will support the functionality of these six buttons. Modify BTView.java so that it will display search paths.

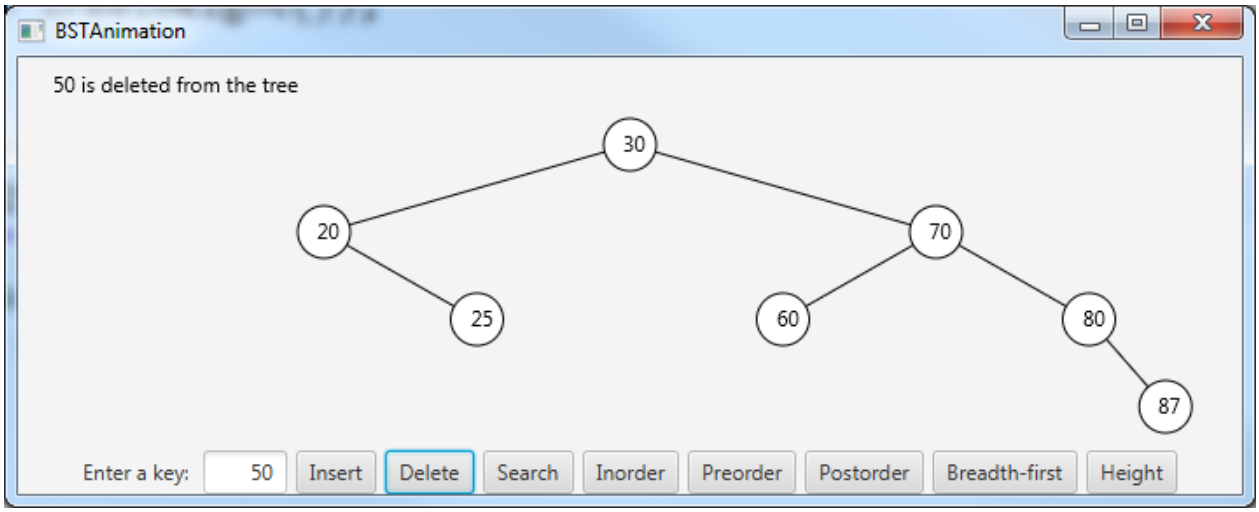
The Search button:
When a user enters a key into the text box and presses the Search button, the program should shade the search path for that key. (Recall that the search path is returned by the BST.path(E e) method, which is already provided in BST.java.)





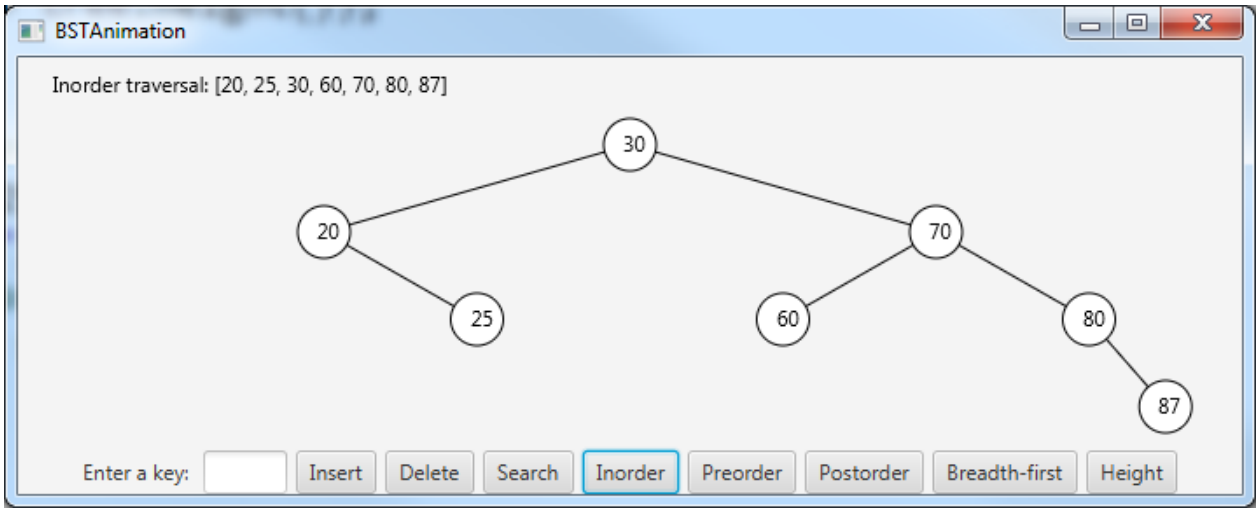
When a search is complete, the status text in the upper left-hand part of the window should display the result of the search. (See the "Found 50 in the tree" and "84 is not in the tree" messages above.)

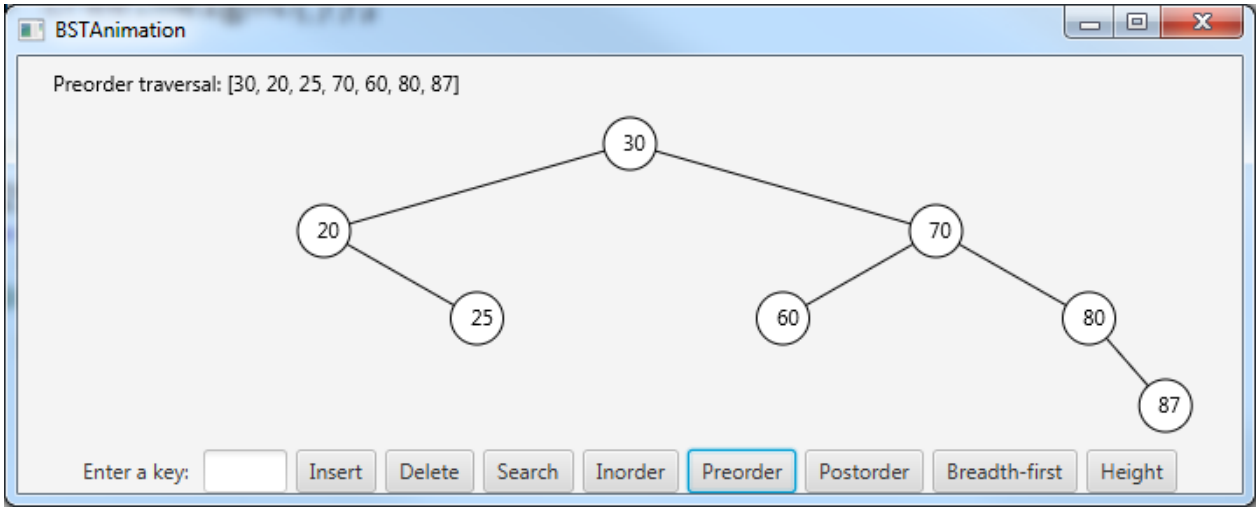
Immediately after a node on the highlighted search path is deleted from the tree via the Delete button, there should be no highlighted path.

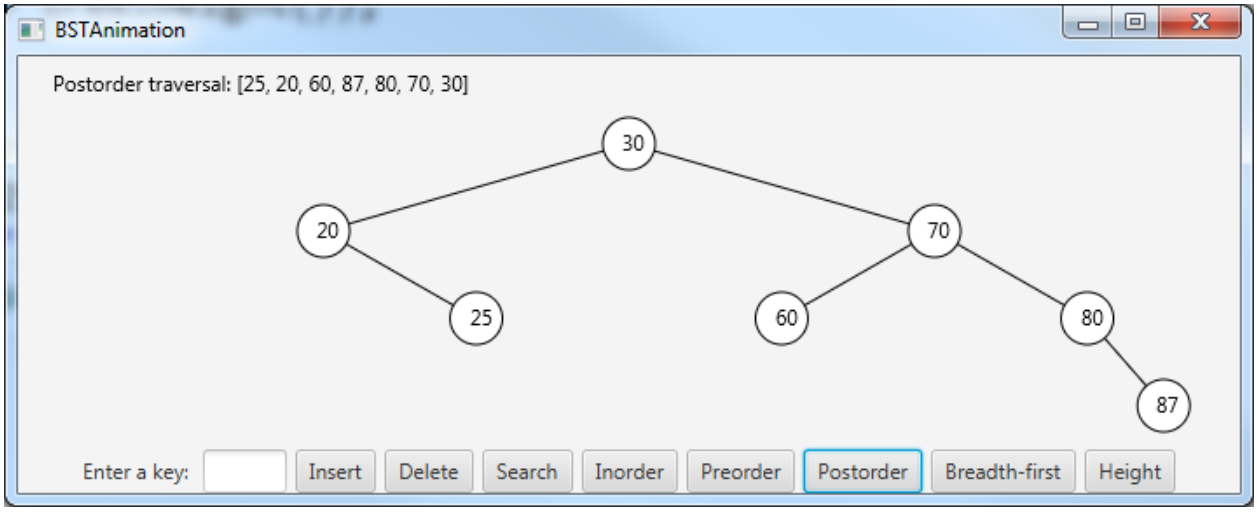


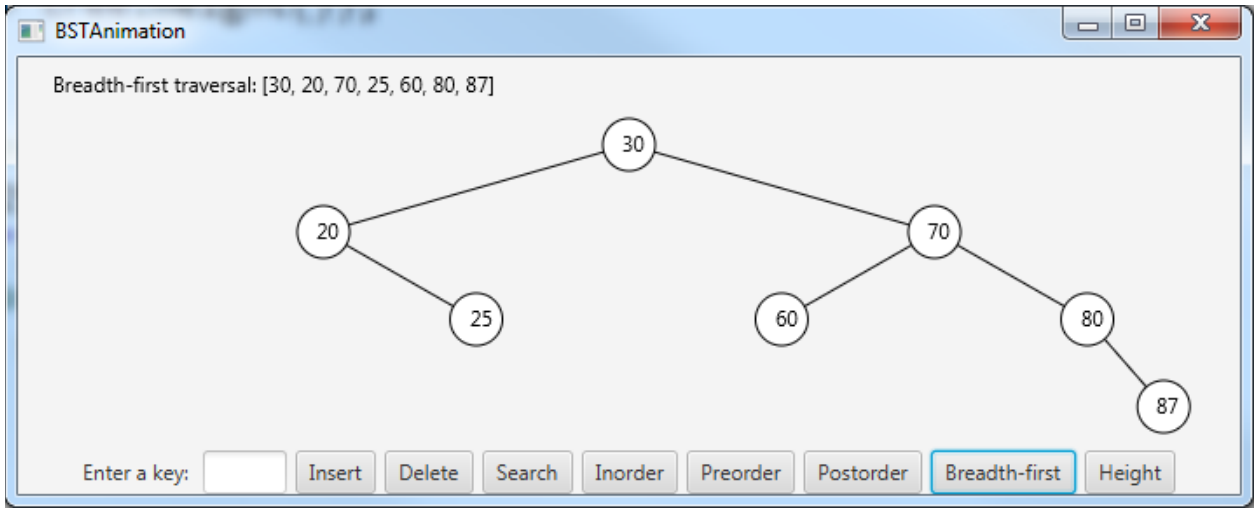
Search button implementation tip:
Implementing the Search button functionality should be fairly simple. Get the path list from the BST.path(E e) method. When drawing a node in the BTView class, check whether the node is in the search path. If it is then its fill color should be orange.

The tree traversal buttons:
The Inorder, Preorder, Postorder, and Breadth-first buttons each display a text message that lists the tree elements in the corresponding tree traversal ordering:



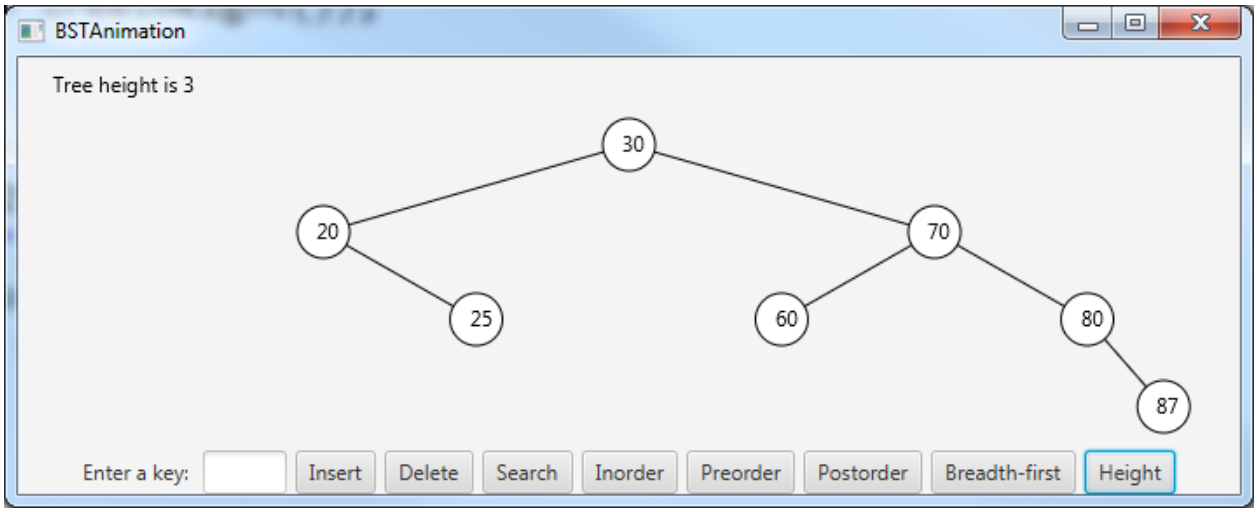






Suggestions for implementing the traversals are provided later in this write-up.

The Height button:
The Height button brings up a text message indicating the height of the tree. Recall from page 930 that the height of a tree with a single node is 0 and the height of an empty tree is -1.



Required methods:
In support of the above functionality you are required to add the following methods to the BST class. Your method headers must match these headers exactly. Each method returns a list of the tree elements, according to a inorder, preorder, postorder, or breadth-first order, respectively:

```
public java.util.List<E> inorderList()
public java.util.List<E> preorderList()
public java.util.List<E> postorderList()
public java.util.List<E> breadthFirstOrderList()
```

You also must add to the BST class a height method that returns the height of the tree:

```
public int height()
```

Inorder, preorder, postorder, and height implementation tips:
Each of these four methods are probably easiest to implement by using recursive helper methods. For example, the inorderList() can be implemented by using a recursive helper method with the following signature:

```
private void inorderList(TreeNode<E> root, List<E> list)
// Adds to list the elements in the indicated subtree,
// using an inorder traversal.
```

The height() method can be implemented by using a recursive helper method with this signature:

```
private int height(TreeNode<E> root)
// Returns the height of the indicated subtree.
```

Breadth-first traversal:
Breadth-first traversal may be implemented easily with a loop and a queue. The following pseudocode from Wikipedia should be useful in designing your `breadthFirstOrderList` code. The article refers to breadth-first order as "level order".

```
levelorder(root)
  q = empty queue
  q.enqueue(root)
  while not q.empty do
    node := q.dequeue()
    visit(node)
    if node.left ≠ null then
      q.enqueue(node.left)
    if node.right ≠ null then
      q.enqueue(node.right)
```

What to turn in:
Submit the four program files, BST.java, BSTAnimation.java, BTView.java, and Tree.java, on Blackboard. You probably will not need to modify Tree.java. As always, follow the coding conventions for the course (provided again below for your convenience).

CODING CONVENTIONS (IMPORTANT):
For this homework, and each subsequent homework, you are required to follow the coding conventions listed below. Failure to follow these conventions in full may result in a loss of up to 20% of your grade on the assignment, even if your code works perfectly.

Required comments:
Each file that you write or modify should have comments at the beginning that state your name, the course number (CSC 364-001), and a brief description of what the program does. Of course, you are always free to provide additional comments that you feel make the program code easier to understand.

Indentation and spacing:
As stated in Section 1.9.2: "Indent each subcomponent or statement at least two spaces more than the construct within which it is nested." Also, make sure that statements that are nested at the same level have their left-most characters line up evenly.

Good:

```
if (avg >= 92.0)
{
   grade = "A";
   System.out.println("Great job!");
}
```

Bad:

```
if (avg >= 92.0)
{
grade = "A";
   System.out.println("Great job!");
}
```

Bad:

```
if (avg >= 92.0)
{
   grade = "A";
      System.out.println("Great job!");
}
```

Furthermore, "a single space should be added on both sides of a binary operator".

Good:

```
int i = 3 + 4 * 4;
```

Bad:

```
int i= 3+4 * 4;
```

Block styles:
Review Section 1.9.3. You may use either the "Next-line style" or the "End-of-line style" of creating code blocks. Each has its advantages and disadvantages. In a given file, you should use one style or the other: do not mix block styles within a single source code file.

Next-line style:

```
public class Test
{
   public static void main(String[] args)
   {
      System.out.println("Block Styles");
   }
}
```

End-of-line style:

```
public class Test {
   public static void main(String[] args) {
      System.out.println("Block Styles");
   }
}
```

Naming conventions:
As specified in Section 2.8 of the textbook, you should:

- Begin the name of each variable and method with a lower case letter. "If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word – for example, the variables **radius** and **area** and the method **showInputDialog**."
- "Capitalize the first letter of each word in a class name – for example, the class names **WindChill**, **Math**, and **JOptionPane**."
- "Capitalize every letter in a constant, and use underscores between words – for example, the constants **PI** and **MAX_VALUE**."

Tip: Always start the name of a .java source file with a capital letter. E.g., **WindChill.java**, *NOT* **windChill.java**. This will force you to start the name of the public class therein with a capital letter.