Assignment 3

This assignment has two parts. The first part is a practice of using Java's *LinkedList* class. The second part is a practice of writing a code that involves a binary tree, which is implemented using a linked structure.

Part 1 (50 points)

Create a file named *Hw3_Part1.java* that performs the following requirements. An incomplete code is posted on Blackboard.

- Read employee information from *employee_info.txt* file.
- Store employees in a linked list (**you must use Java's *LinkedList***).
- Create an iterator.
- Print the youngest employee and the employee with the highest salary. When you find the youngest employee and the highest earning employee, you **must use the iterator** created above (this is a practice of using an interator). A sample output is shown below:

```
Youngest employee is:

Name = Jake
Age = 27
Employee id = 4567
Classification = assistant
Salary = 60000.0

Highest earning employee is:

Name = Pederica
Age = 53
Employee id = 5678
Classification = CEO
Salary = 500000.0
```

- Write a method that sorts employees in the linked list in non-decreasing order of salary. You must implement the following pseudocode.

  Algorithm sortBySalary
  Input: a linked list with n employees; assume $n > 0$
  Output: a sorted linked list, where employees are sorted by non-decreasing order of salary

  For $i = n$-1 down to 1 // $i = n$-1, n-2, …, 1
  Find the employee with the highest salary in the list(0 .. i) and move it to list(i)

  In the above pseudocode, list(0 .. i) is a *sublist* that contains the elements list(0) through list(i)

The algorithm is illustrated below with a list of integers:

Initial list = (5, 3, 7, 10, 6), $n = 5$

$i = 4$
The largest integer in list(0 .. 4) is 10. 10 is moved to list(4)
list = (5, 3, 7, 6, 10)

$i = 3$
The largest integer in list(0 .. 3) is 7. 7 is moved to list(3)
list = (5, 3, 6, 7, 10)

$i = 2$
The largest integer in list(0 .. 2) is 6. 6 is moved to list(2)
list = (5, 3, 6, 7, 10)

$i = 1$
The largest integer in list(0 .. 1) is 5. 10 is moved to list(1)
list = (3, 5, 6, 7, 10)

- The signature of the method must be:

```
public static void sortBySalary(LinkedList<Employee> empList)
```

You must write your own code that sorts the employees in the list (i.e., **you should not use other person's code**).

- Test the sorting method within the main method as follows:
    - Print all employees in the linked list.
    - Sort the list by invoking the *sortBySalary* method.
    - Print the sorted list.

The following is a sample output:

```
All employees before sorting:

Name = Susan
Age = 32
Employee id = 2345
Classification = director
Salary = 200000.0

Name = Kelsey
Age = 45
```

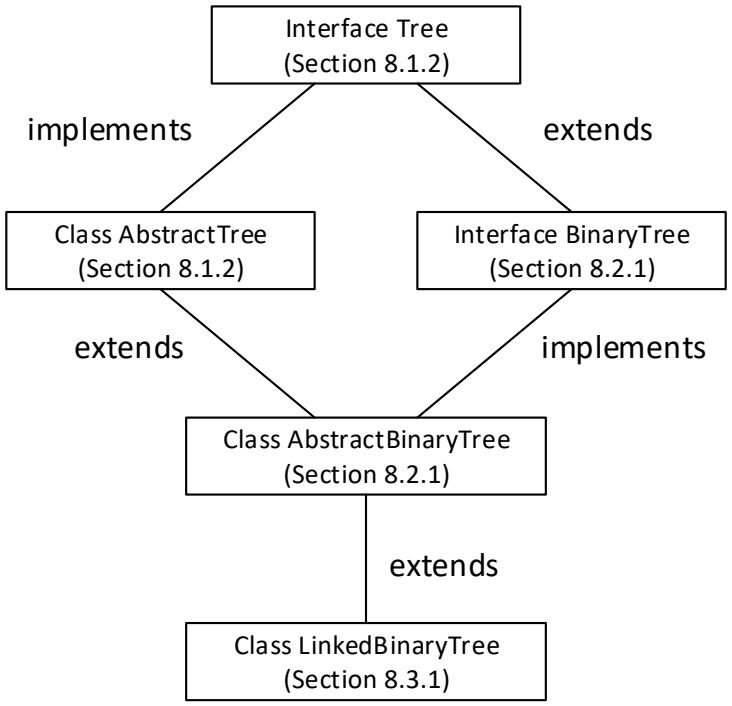
```
Employee id = 3456
Classification = director
Salary = 250000.0

Name = Jake
Age = 27
Employee id = 4567
Classification = assistant
Salary = 60000.0

Name = Pederica
Age = 53
Employee id = 5678
Classification = CEO
Salary = 500000.0

Name = Yapsiong
Age = 36
Employee id = 6789
Classification = manager
Salary = 100000.0

All employees after sorting:

Name = Jake
Age = 27
Employee id = 4567
Classification = assistant
Salary = 60000.0

Name = Yapsiong
Age = 36
Employee id = 6789
Classification = manager
Salary = 100000.0

Name = Susan
Age = 32
Employee id = 2345
Classification = director
Salary = 200000.0

Name = Kelsey
Age = 45
Employee id = 3456
Classification = director
Salary = 250000.0

Name = Pederica
Age = 53
```

```
Employee id = 5678
Classification = CEO
Salary = 500000.0
```

Part 2 (50 points)

This part is an implementation of a ***generic binary search tree***, which extends the *LinkedBinaryTree* class. Binary trees are discussed in Section 8.2 and Section 8.3.1 in the textbook.

The *LinkedBinaryTree.java*, which comes with the textbook, is a concrete implementation of a binary tree that uses a linked structure. It extends the *AbstractBinaryTree* class. The relevant class hierarchy is shown below:



The *LinkedBinaryTree* inherits variables and methods from its superclasses and it also implements its own methods.

Your task is as follows:

Create a generic class named *MyBST* as a subclass of *LinkedBinaryTree*. The *MyBST* is an implementation of a binary search tree using a linked tree structure.

A *binary search tree* is a binary tree that satisfies the following *binary search tree property*.

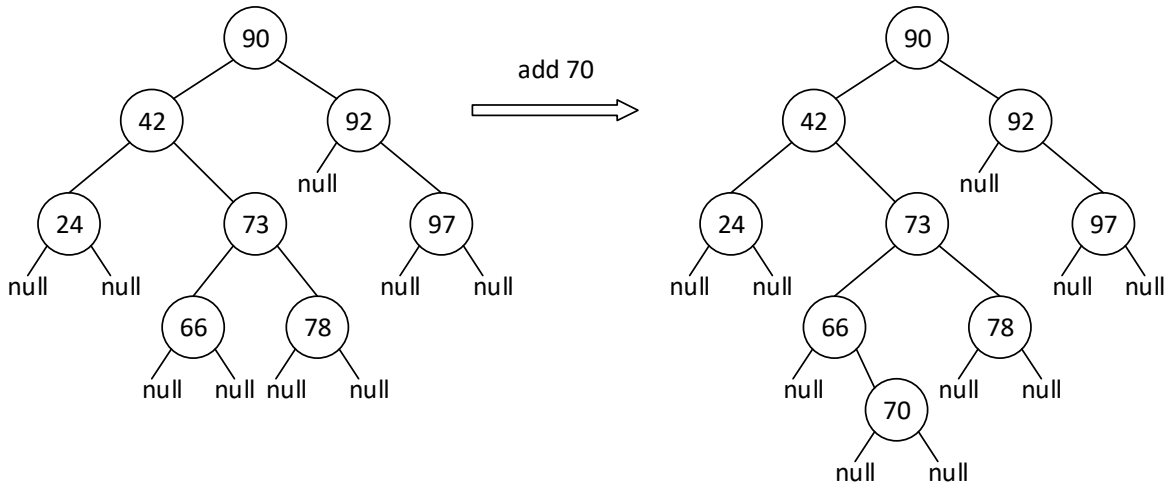
For each internal position *p*:

- Elements stored in the left subtree of *p* are less than *p*'s element.
- Elements stored in the right subtree of *p* are greater than *p*'s element.

Then, you are required to implement an *add* method, whose pseudocode is given below:

```
Algorithm add(p, e)
  Input parameters:
    p: The position of the root of the tree (or subtree) to which
       a new node is added
    e: The element of the new node to be added
  Output: Returns the position of the new node that was added.
    If there already is a node with e in the tree, returns null.

  if p == null // this is an empty tree
    create a new node with e and make it the root of the tree
    increment size // size is the number of nodes currently in the tree
    return the root

  x = p
  y = x
  while (x is not null) {
    if (the element of x) is the same as e, return null
    else if (the element of x) > e {
      y = x
      x = left child of x
    }
    else {
      y = x
      x = right child of x
    }
  } // end of while

  temp = new node with element e
  y becomes the parent of temp
  if (the element of y) > e
    temp becomes the left child of y
  else
    temp becomes the right child of y
  increment size // size is the number of nodes currently in the tree
  return temp
```

The following figure illustrates adding a node to a binary search tree. If you add a node with 70 to the tree on the left, the result is the tree on the right.



You may want to read and study the codes of *LinkedBinaryTree* and its superclasses and super interfaces carefully before writing a code.

An incomplete code of *MyBST.java* is posted on Blackboard. You need to complete this code as indicated and test the add method within the main method.

Documentation

No separate documentation is needed. However, you must include sufficient inline comments within your program.

Deliverables

You need to submit the following files:

- *Hw3_Part*1.*java*
- *MyBST.java*
- All other files that are needed to compile and run your program

Combine all files into a single archive file. Name the archive file *LastName_FirstName_hw*3.*EXT*, where *EXT* is an appropriate file extension, such as *zip* or *rar*. Then, upload it to Blackboard.

Grading

Part 1: Your facilitator will run your program with a test input file, which has information of five employees, and points will be deducted as follows:

- Up to 10 points are deducted if the youngest employee is wrong.
- Up to 10 points are deducted if the employee with the highest salary is wrong.
- Up to 15 points are deducted if employees are not sorted correctly.

Part 2: Your facilitator will run your program and add two sequences of integers to an empty tree. Up to 15 points will be deducted for each input sequence if the resulting tree is wrong.

If you do not follow any requirement (such as “use an iterator when finding the youngest employee” or “use the given pseudocode of the *add* method,” etc.), additional points will be deducted.

Points will be deducted up to 20 points if your program does not have sufficient inline comments.