CSE143 Final
Summer 2017

Name of Student: _____

Section (e.g., AA):_____ Student Number: _____

The exam is divided into two parts. This is **part one** with 3 questions total.

#	Problem Area	Points	Score
1	Inheritance/Polymorphism	10	_____
2	Binary Trees	10	_____
3	LinkedIntList	20	_____
	Total	40	

This is a closed-book/closed-note exam. Space is provided for your answers. There is a "cheat sheet" at the end that you can use as scratch paper. You are not allowed to access any of your own papers during the exam.
You may not use calculators or any other devices.

The exam is not, in general, graded on style and you do not need to include comments. For the stack/queue and collections questions, however, you are expected to use generics properly and to declare variables using interfaces when possible. You may only use the Stack and Queue methods on the cheat sheet, which are the methods we discussed in class. You are not allowed to use programming constructs like break, continue, or returning from a void method on this exam. Do not use constructs from Java 8.

Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.

You are NOT to use any electronic devices while taking the test, including calculators. Anyone caught using an electronic device will receive a 10 point penalty.

Do not begin work on this exam until instructed to do so. Any student who starts early or who continues to work after time is called will receive a 10 point penalty.

If you finish the exam early, please hand your exam to the instructor and exit quietly through the front door.

1. Details of inheritance, 10 points. Assuming that the following classes have been defined:

```
public class Morty extends Beth {
    public void method1() {
        System.out.println("Morty 1");
    }

    public void method3() {
        System.out.println("Morty 3");
    }
}

public class Rick {
    public void method1() {
        System.out.println("Rick 1");
    }

    public void method2() {
        System.out.println("Rick 2");
        method1();
    }
}

public class Beth extends Rick {
    public void method1() {
        System.out.println("Beth 1");
        super.method1();
    }
}

public class Summer extends Rick {
    public void method3() {
        System.out.println("Summer 3");
    }
}
```

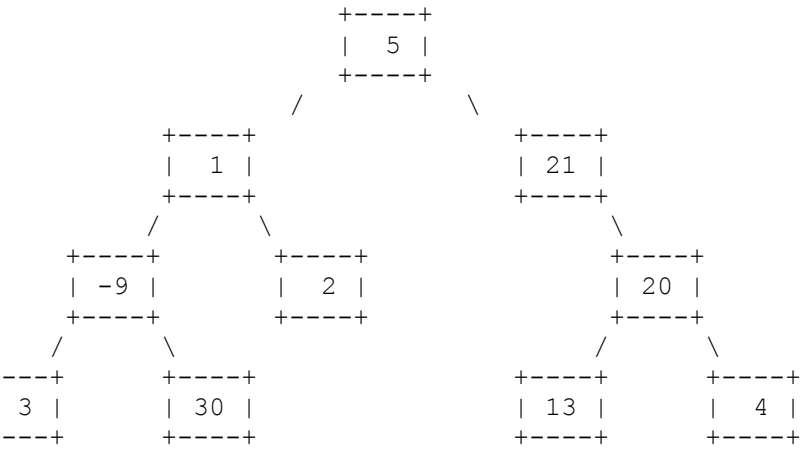
And assuming the following variables have been defined:

```
Rick var1 = new Morty();
Rick var2 = new Beth();
Rick var3 = new Rick();
Object var4 = new Beth();
Beth var5 = new Morty();
Object var6 = new Summer();
```

In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, fill in the right-hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

Statement	Output
`var1.method1();`	_____
`var2.method1();`	_____
`var3.method1();`	_____
`var4.method1();`	_____
`var5.method1();`	_____
`var6.method1();`	_____
`var1.method2();`	_____
`var2.method2();`	_____
`var3.method2();`	_____
`var4.method2();`	_____
`var5.method2();`	_____
`var6.method2();`	_____
`((Beth)var1).method3();`	_____
`((Summer)var6).method3();`	_____
`((Morty)var4).method1();`	_____
`((Rick)var6).method2();`	_____
`((Rick)var4).method1();`	_____
`((Beth)var6).method3();`	_____
`((Morty)var3).method3();`	_____
`((Morty)var5).method3();`	_____

2. Binary Trees, 10 points. Write a method called hasPathSum that takes an integer n as a parameter and that returns true if there is some nonempty path from the overall root of a tree to a node of the tree in which the sum of the data stored in the nodes on that path adds up to n (returning false if no such path exists). For example if the variable t refers to the following tree:



Below are various calls and an explanation for the value returned:

```
t.hasPathSum(8) returns true because of the path (5, 1, 2)
t.hasPathSum(26) returns true because of the path (5, 21)
t.hasPathSum(0) returns true because of the path (5, 1, -9, 3)
t.hasPathSum(5) returns true because of the path (5)
t.hasPathSum(1) returns false because no path with that sum exists
```

You are writing a public method for a binary tree class defined as follows:

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    <constructors>
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class. You may not construct any extra data structures to solve this problem.

Space is provided on the next page.

This page is left blank for space on #2

3. Linked Lists, 20 points. Write a method called weave that takes a list of integers as a parameter and that combines the values from the second list into the first list in an alternating fashion, leaving the second list empty. The new list should start with the first value of the first list followed by the first value of the second list followed by the second value of the first list followed by the second value of the second list, and so on. For example, if the variables list1 and list2 store the following:

```
list1 = [0, 2, 4, 6], list2 = [1, 3, 5, 7]
```

If you make the following call:

```
list1.weave(list2);
```

then the lists should store the following values after the call:

```
list1 = [0, 1, 2, 3, 4, 5, 6, 7], list2 = []
```

Notice that the second list is empty after the call and that the values that were in the second list have been moved into the first list in an alternating fashion. If the call had instead been:

```
list2.weave(list1);
```

then the lists would have stored the following values after the call:

```
list1 = [], list2 = [1, 0, 3, 2, 5, 4, 7, 6]
```

These examples use sequential integers to make it easier to see the intended order, but the values can be any numbers at all. It is also possible that one list will have more values than the other, in which case its values should simply be appended to the end of the list. For example, if the lists store these values:

```
list1 = [3, 18, 9], list2 = [-5, 4, 13, 42, 0, 23]
```

then the call list1.weave(list2) should produce this result:

```
list1 = [3, -5, 18, 4, 9, 13, 42, 0, 23], list2 = []
```

while the call list2.weave(list1) should produce this result:

```
list1 = [], list2 = [-5, 3, 4, 18, 13, 9, 42, 0, 23]
```

You are writing a public method for a linked list class defined as follows:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

<continued on next page>

You are writing a method that will become part of the LinkedIntList class. Both lists are of type LinkedIntList. You may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. You are allowed to define your own variables of type ListNode, but you may not construct any new nodes, and you may not use any auxiliary data structure to solve this problem (no array, ArrayList, stack, queue, String, etc). You also may not change any data fields of the nodes. You MUST solve this problem by rearranging the links of the lists. Your solution must run in O(n) time where n is the length of the list.

CSE143 Final
Summer 2017

Name of Student: _____

Section (e.g., AA):_____ Student Number: _____

The exam is divided into two parts. This is **part two** with 5 questions total.

#	Problem Area	Points	Score
4	Binary Tree Traversal	6	_____
5	Binary Search Tree	4	_____
6	Collection Programming	10	_____
7	Comparable	20	_____
8	Binary Tree Programming	20	_____
9	**Extra Credit** Fiction	1	_____
	Total	100	_____

This is a closed-book/closed-note exam. Space is provided for your answers. There is a "cheat sheet" at the end that you can use as scratch paper. You are not allowed to access any of your own papers during the exam.
You may not use calculators or any other devices.

The exam is not, in general, graded on style and you do not need to include comments. For the stack/queue and collections questions, however, you are expected to use generics properly and to declare variables using interfaces when possible. You may only use the Stack and Queue methods on the cheat sheet, which are the methods we discussed in class. You are not allowed to use programming constructs like break, continue, or returning from a void method on this exam. Do not use constructs from Java 8.

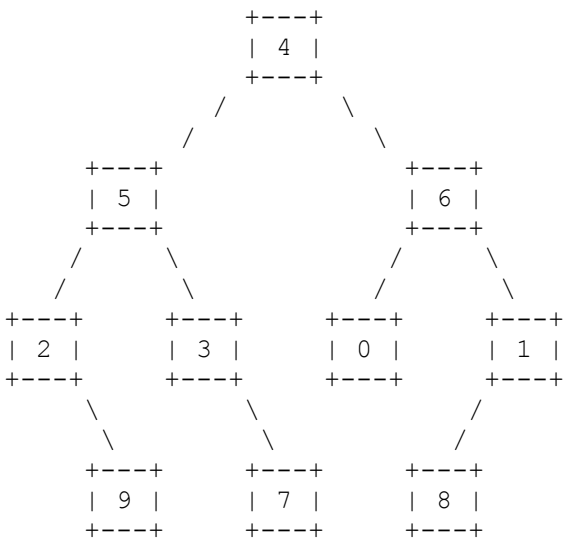
Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.

You are NOT to use any electronic devices while taking the test, including calculators. Anyone caught using an electronic device will receive a 10 point penalty.

Do not begin work on this exam until instructed to do so. Any student who starts early or who continues to work after time is called will receive a 10 point penalty.

If you finish the exam early, please hand your exam to the instructor and exit quietly through the front door.

4. **Binary Tree Traversals**, 6 points. Consider the following tree.



Fill in each of the traversals below:

Preorder traversal _____

Inorder traversal _____

Postorder traversal _____

5. **Binary Search Tree**, 4 points. Draw a picture below of the binary search tree that would result from inserting the following words into an empty binary search tree in the following order:

Lucille, Gob, Maeby, Lindsay, Tobias, Buster, Michael

Assume the search tree uses alphabetical ordering to compare words.

6. **Collections Programming**, 10 points. Write a method called split that takes a set of strings as a parameter and that returns the result of splitting the strings into different sets based on the length of the strings. In particular, your method should return a map whose keys are integers and whose values are sets of strings of that length. For example, if a variable called words contains the following set of strings:

```
[to, be, or, not, that, is, the, question]
```

then the call split(words) should return a map whose values are sets of strings of equal length and whose keys are the string lengths:

```
{2=[be, is, or, to], 3=[not, the], 4=[that], 8=[question]}
```

Notice that strings of length 2 like "be" and "is" appear in a set whose key is 2. If the set had instead stored these strings:

```
[four, score, and, seven, years, ago, our, fathers, brought, forth]
```

Then the method would return this map:

```
{3=[ago, and, our], 4=[four], 5=[forth, score, seven, years],
 7=[brought, fathers]}
```

The set of strings passed to your method will not necessarily be in order, but the map returned by your method should be ordered numerically by key and each set contained in the map should be ordered alphabetically, as in the examples above.

Your method should construct the new map and each of the sets contained in the map but should otherwise not construct any new data structures. It should also not modify the set of words passed as a parameter.

7. **Comparable class,** 20 points. Define a class called AdmissionsEntry that keeps track of information for an admissions candidate and how that candidate is rated by reviewers (real numbers between 0.0 and 5.0). The class has the following public methods:

AdmissionsEntry(id)	constructs an AdmissionsEntry object with given ID
rate(rating)	records a rating for the candidate
flag()	flags the candidate to definitely be discussed
getID()	returns the ID of the candidate
getRating()	returns the average rating (0.0 if no ratings)
toString()	returns a String with ID and average rating

Below is an example for a candidate that has been reviewed four times:

```
AdmissionsEntry entry = new AdmissionsEntry("2222222");
entry.rate(3.75);
entry.rate(3.65);
entry.rate(3.8);
entry.rate(3.75);
entry.flag();
```

After these calls, the call entry.getRating() would return 3.7375 (the average of the ratings). The toString method should return a string composed of the ID, a colon, and the average rating rounded to 2 digits after the decimal point ("2222222: 3.74" for this example). If there are no ratings, then getRating and toString should indicate a rating of 0.0.

Each AdmissionsEntry object should keep track of whether that candidate should be discussed by the admissions committee. Any candidate who receives a score of 4.0 or higher should be discussed even if their average rating is below 4.0. Notice also that a candidate can be flagged for discussion even if none of the ratings are 4.0 or higher, as in the example above.

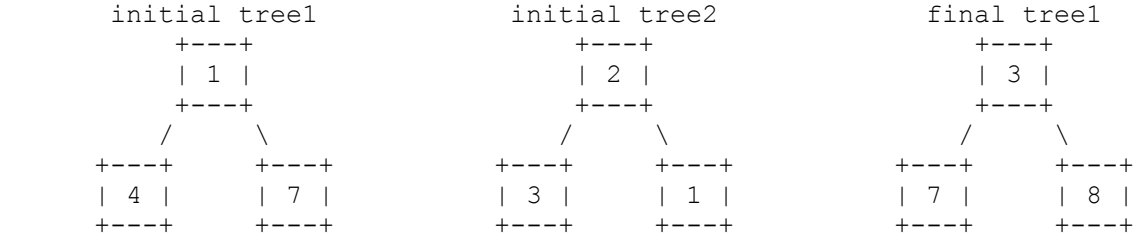
The AdmissionsEntry class should implement the Comparable<E> interface. Define the method so that when sorted, a list of entries will have students to be discussed appearing first followed by students not to be discussed. Within those groups, students with higher average ratings should appear earlier in the list. Students with the same discussion status and the same average rating should appear in increasing order by ID. Recall that values considered "less" appear earlier in a sorted list. You may not use Double objects or methods of the Double class to solve this problem.

This page is left blank for space on #7

8. **Binary Trees,** 20 points. Write a method called add that takes as a parameter a reference to a second binary tree and that adds the values in the second tree to this tree. If the method is called as follows:

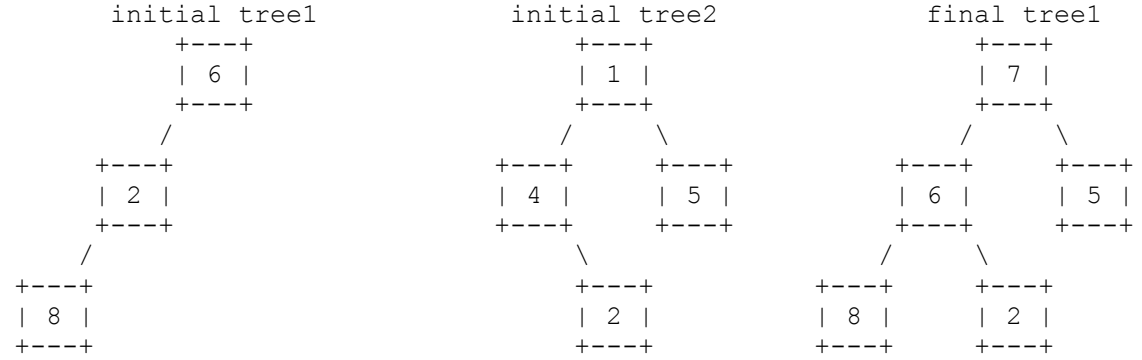
```
tree1.add(tree2);
```

it should add all values in tree2 to the corresponding nodes in tree1. In other words, the value stored at the root of tree2 should be added to the value stored at the root of tree1 and the values in tree2's left and right subtrees should be added to the corresponding positions in tree1's left and right subtrees. The values in tree2 should not be changed by your method.



If tree1 has a node that has no corresponding node in tree2, then that node is unchanged. For example, if tree2 is empty, tree1 is not changed at all.

It is also possible that tree2 will have one or more nodes that have no corresponding node in tree1. For each such node, create a new node in tree1 in the corresponding position with the value stored in tree2's node. For example:



You are writing a public method for a binary tree class defined as follows:

```
public class IntTreeNode {
    public int data;          // data stored in this node, note: not final
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

<continued on next page>

You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available. You are NOT to replace any of the existing nodes in the tree. You will, however, construct new nodes to be inserted into the tree as described above.

9. Fiction, 1 point extra credit. Your TA has woken up and discovered that they now have a super power. Decide what super power your TA would have and what they would do now that they are a super. You may express your answer in any way you choose by writing a story, drawing a picture, writing a poem, etc. If your answer below indicates at least 1 minute of effort, you will receive full credit, although your TA would probably appreciate it if you put in a little more effort.