CSE143 Final
Winter 2019

Name of Student: _____

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

The exam is divided into eight questions with the following points:

#	Problem Area	Points	Score
1	Binary Tree Traversal	6	_____
2	Binary Search Tree	4	_____
3	Inheritance/Polymorphism	10	_____
4	Comparable	15	_____
5	Collections	15	_____
6	Binary Tree Programming	10	_____
7	Binary Tree Programming	20	_____
8	LinkedList Programming	20	_____
EC	Extra Credit	+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.

The exam is not, in general, graded on style and you do not need to include comments. For the Collections questions, however, you are expected to use generics properly and to declare variables using interfaces when possible. 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. For the inheritance problem, you may abbreviate “compiler error” as CE and “runtime error” as RE.

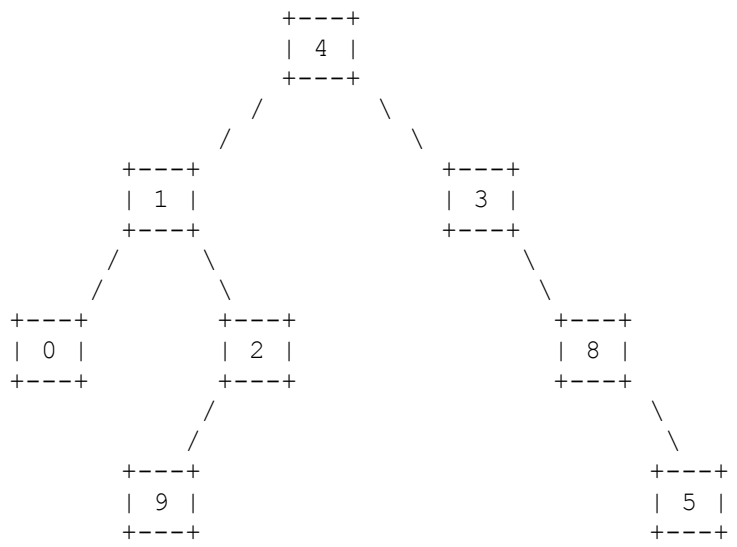
You are NOT to use any electronic devices while taking the test, including calculators. Anyone caught using an electronic device will receive a 10 point penalty.

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.

You are allowed to ask for scratch paper after the exam starts to use as additional space when writing answers, but you must indicate on the original page for the problem that part of the answer is on scratch paper. Scratch paper must be stapled to the end of your exam after you finish the test. Failure to do so may result in your work on scratch paper not being graded.

Initial here to indicate you have read and agreed to these rules: _____

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



Fill in each of the traversals below:

Preorder traversal _____

Inorder traversal _____

Postorder traversal _____

2. **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:

Jughead, Cheryl, Pop, Archie, Veronica, Betty, Kevin

Assume the search tree uses alphabetical ordering to compare words.

3. **Inheritance/Polymorphism, 10 points**: Assuming that the following classes have been defined:

```
public class Square extends Rectangle {
    public void method2() {
        System.out.println("Square 2");
    }

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

public class Circle extends Shape {
    public void method2() {
        System.out.println("Circle 2");
    }

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

public class Shape {
    public void method1() {
        System.out.println("Shape 1");
        method3();
    }

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

public class Rectangle extends Shape {
    public void method3() {
        System.out.println("Rectangle 3");
        super.method3();
    }
}
```

And assuming the following variables have been defined:

```
Shape var1 = new Rectangle();
Square var2 = new Square();
Shape var3 = new Circle();
Shape var4 = new Square();
Shape var5 = new Shape();
Object var6 = new Rectangle();
```

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; you may use the abbreviations "CE" and "RE" respectively.

Statement	Output
`var1.method1();`	_____
`var2.method1();`	_____
`var3.method1();`	_____
`var4.method1();`	_____
`var5.method1();`	_____
`var6.method1();`	_____
`var1.method2();`	_____
`var2.method2();`	_____
`var3.method2();`	_____
`var1.method3();`	_____
`var2.method3();`	_____
`var3.method3();`	_____
`((Square) var6).method1();`	_____
`((Rectangle) var3).method2();`	_____
`((Square) var4).method2();`	_____
`((Shape) var3).method2();`	_____
`((Circle) var3).method2();`	_____
`((Square) var1).method1();`	_____
`((Rectangle) var4).method3();`	_____
`((Shape) var6).method3();`	_____

4. **Comparable, 15 points**: Define a class called IceCream that represents some ice cream with potentially multiple flavors of various amounts. An IceCream starts with no ice cream scoops, but more can be added later. Your class should have the following public methods:

`IceCream()`	Constructs a new IceCream
`void add(String flavor, int scoops)`	Adds the given flavor of ice cream with the given quantity of scoops. The same flavor can be added multiple times, which should increase that flavor's scoop count.
`void addSprinkles()`	Adds sprinkles to this IceCream
`int getFlavor(String flavor)`	Returns the number of scoops of the given flavor in this IceCream (0 if no scoops of the given flavor have been added)
`String toString()`	Returns a string representation of this IceCream

When adding a flavor to the IceCream with the add method, the number of scoops of that flavor in the IceCream should be increased by the given number of scoops. A flavor may be added more than once to a particular IceCream and this would count as an addition of that flavor's number of scoops. If the value for scoops passed to the add method is less than 1, an IllegalArgumentException should be thrown.

If no scoops of ice cream have been added to the IceCream, then the toString method should return a string of the form:

```
"No ice cream :("
```

If flavors have been added, it should instead return a string with this format

```
"<scoops> scoops of ice cream with <flavors>"
```

For example, if the following lines are executed:

```
IceCream order0 = new IceCream();
IceCream order1 = new IceCream();
order1.add("vanilla", 1);
order1.add("chocolate", 2);
order1.add("vanilla", 2);
```

Then the following calls to toString would return:

```
order0.toString();    "No ice cream :("
order1.toString();    "5 scoops of ice cream with [chocolate, vanilla]"
```

The order of the flavors in the string representation does not matter.

In addition, the IceCream class should implement the Comparable<E> interface. IceCream objects are compared by total number of scoops in them, where the IceCream with more scoops is considered greater-than the other. Ties are broken by comparing number of flavors, where the IceCream with more flavors is considered less-than the other. Finally, ties are broken by comparing if the IceCream has sprinkles, where the IceCream with sprinkles is considered greater-than the other. Note that the number of times sprinkles are added does not matter for this comparison.

This page is left blank so you have extra space on #4

5. **Collections Programming, 15 points:** Write a method called favoriteFoods that takes a map indicating how each person rates various foods and a target rating and returns a map indicating all the foods each person has rated with at least the target rating.

The input map will have keys that are people's names (strings) and values which are maps with keys that are a food (strings) and values which are numbers in the range of 0.0 to 5.0 for the rating that person has given that food. An example would be if we had a variable called ratings that stored the following map in the format just described:

```
{"Porter"={"pie"=5.0, "ice cream"=5.0, "mushrooms"=0.0},
 "Erik"={"chicken strips"=4.3, "cranberry sauce"=4.2},
 "Yael"={"lettuce"=2.4},
 "Ken"={}}
```

In this example, we see that Porter has rated pie and ice cream as a 5.0 each and mushrooms as a 0.0, while Yael has only rated lettuce as a 2.4.

The favoriteFoods method you are writing should take a ratings map described above and a target rating and should return a map indicating all the foods each person has rated with at least the target rating. The map you are to return should use the people's names as keys and the set of all the foods that person rated with at least the target value as values.

For example, suppose the following call is made:

```
favoriteFoods(ratings, 4.3);
```

Given this call, the following map would be returned:

```
{"Erik"=["chicken strips"],
 "Ken"=[],
 "Porter"=["ice cream", "pie"],
 "Yael"=[]}
```

Notice that the value for the key "Porter" is the set ["ice cream", "pie"] because he rated only those foods with at least a rating of 4.3. The value for the key "Yael" is [] because Yael rated no foods with a rating of at least 4.3. Note that foods rated with a 4.3 should be included (see Erik).

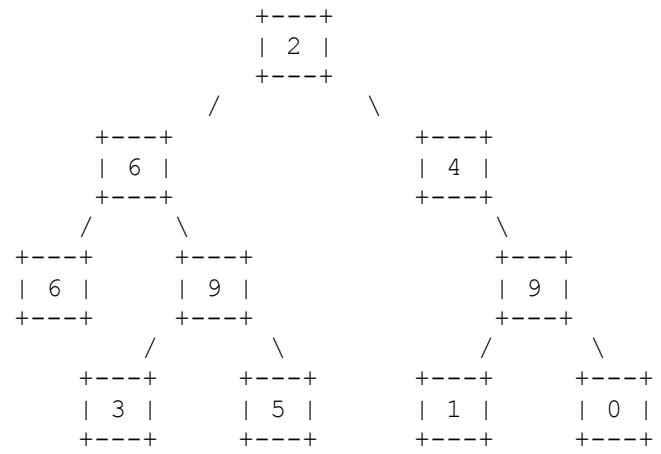
The map you return should have keys sorted alphabetically and the foods in the values should appear in alphabetical order as well.

Your method should not modify the provided map. You may assume that the map and none of its contents are null.

You should use space on the next page to write your answer.

This page is left blank so you have extra space on #5

6. **Binary Tree Programming, 10 points:** Write a method of the IntTree class called countMatching that returns a count of the number of siblings (nodes sharing the same direct parent node) that have data with matching parity (even/odd). For example, suppose that a variable t stores a reference to the following tree:



then the call t.countMatching() should return 2. The matching siblings come from the parent node that stores 2, whose left and right children store even numbers (6 and 4), and the parent node that stores 9, whose left and right children each store odd numbers (3 and 5).

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

```
public class IntTree {
    private IntTreeNode overallRoot;

    <methods>

    private static 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>
    }
}
```

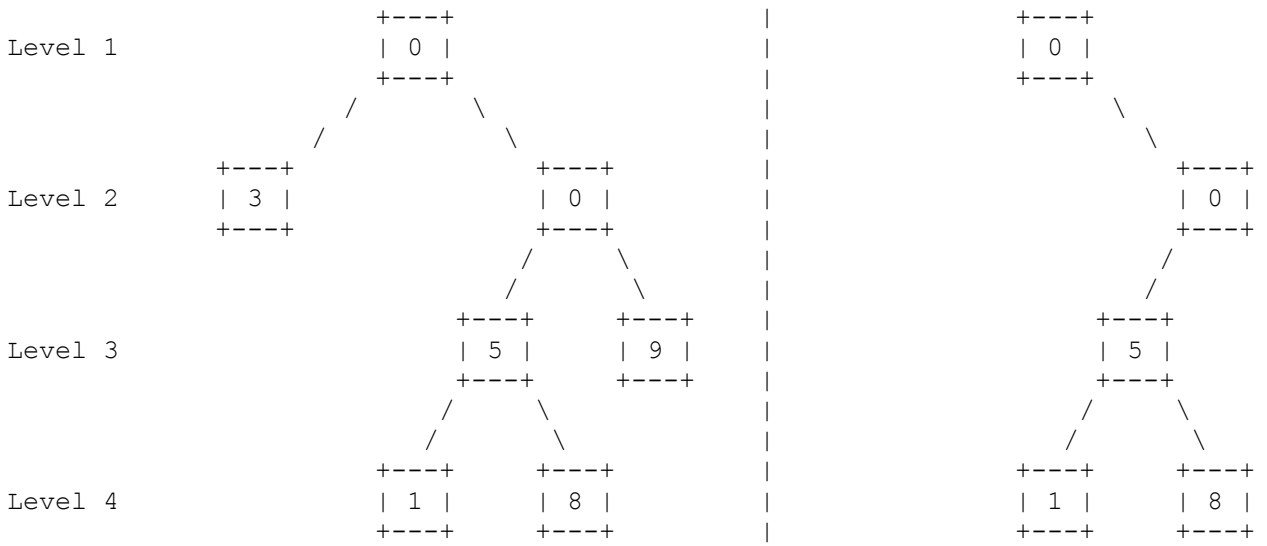
You are writing a method that will become part of the IntTree class. You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the class.

Use the next page to write your answer

This page is left blank so you have extra space on #6

7. **Binary Tree Programming, 20 points:** Write a method called trimTo that removes all leaves in the tree up to and including a given level while leaving all leaves after that level unchanged. The root of the tree is level 1, its children are level 2, and so on.

For example, suppose a variable t stores a reference to the tree on the left, then after the call t.trimTo(3), t would reference the tree on the right



The nodes missing in the after picture are all of the leaf nodes that appear at or before level 3 in the tree.

If the given level is less than 1, your method should throw an IllegalArgumentException.

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

```
public class IntTree {
    private IntTreeNode overallRoot;

    <methods>

    private static 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>
    }
}
```

Your solution must meet the following restrictions:

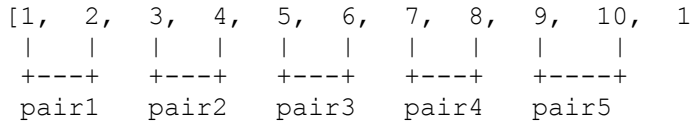
* You may define private helper methods to solve this problem, but otherwise you may not assume that any particular methods are available.
* You are not allowed to change the data fields of the existing nodes in the tree
* You are not allowed to construct new nodes or additional data structures
* Your solution must run in O(n) time where n is the number of nodes in the tree.
* Additionally, your solution must also be efficient in the sense that it does not continue to recurse unnecessarily once it reaches the given level (since no nodes will be removed after that point).

The next page is provided for your solution.

This page is left blank so you have extra space on #7

8. **LinkedList Programming, 20 points:** Write a method of the LinkedIntList class called removeAlternating that removes a node from each pair of nodes in a list in an alternating fashion. In the first pair, the first number should be removed. In the second pair, the second number should be removed. In the third pair, it goes back to the first number being removed. This pattern repeats for pairs that follow. If the list has odd length, the last value should never be removed since it is not part of a pair. The values removed from the list should be returned in a new LinkedIntList with the values appearing the same order as the original list.

For example, if we had a variable list1 that stores the values:



After the following method call:

```
LinkedIntList list2 = list1.removeAlternating();
```

list1 and list2 would store the following values:

list1: [2, 3, 6, 7, 10, 11] list2: [1, 4, 5, 8, 9]

Notice that list2 stores the first values from pairs pair1, pair3, and pair5 and the second values from pairs pair2 and pair4 from the original list that were removed while all of the other values remained in list1. Notice that the number 11 was not removed from the list because it does not belong to a pair.

You are writing a public method for the LinkedIntList class defined as follows:

```
public class LinkedIntList {
    private ListNode front;
    <methods>

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

        <constructors>
    }
}
```

Your solution should follow the following restrictions:

* 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.
* 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 original list.

Grading: About half the points in this problem are extra credit that is awarded for having the code that handles the requisite components to solve the problem. The remaining points are considered “external correctness” where certain points are awarded based on if the written solution correctly works with lists of various lengths (length 0, length 1, length2, etc.). This means even if you don’t know how to fully solve the problem, you can still get a partial credit for attempting to write the necessary code and getting it to work in small cases. However, in order to receive full credit you must verify that the code actually works on all input types.

This page is left blank so you have extra space on #8

Extra Credit: Draw or describe an emoji that represents your TA and write a short sentence explaining why. If there is currently no emoji that represents your TA perfectly, draw a new one and explain why your new emoji fits your TA better than any of the existing ones. Note that artistic ability is not required to earn this point; any work that demonstrates at least one minute of effort and is not inappropriate, offensive, or disrespectful will be given credit.

CSE143 Cheat Sheet

Linked Lists (16.2)

Below is an example of a method that could be added to the LinkedIntList class to compute the sum of the list:

```
public int sum() {
    int sum = 0;
    ListNode current = front;
    while (current != null) {
        sum += current.data;
        current = current.next;
    }
    return sum;
}
```

`Math` Methods (3.2) *mathematical operations*

`Math.abs(`*value*`)`	absolute value
`Math.min(`*v1*`, `*v2*`)`	smaller of two values
`Math.max(`*v1*`, `*v2*`)`	larger of two values
`Math.round(`*value*`)`	nearest whole number
`Math.pow(`*b*`, `*e*`)`	b to the e power
`Math.signum(v)`	signum of v

`Iterator<E>` Methods (11.1) *(An object that lets you examine the contents of any collection)*

`hasNext()`	returns `true` if there are more elements to be read from collection
`next()`	reads and returns the next element from the collection
`remove()`	removes the last element returned by `next` from the collection

`List<E>` Methods (10.1) *(An ordered sequence of values)*

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value at given index, shifting subsequent values right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`isEmpty()`	returns `true` if the list's size is 0
`contains(`**value**`)`	returns `true` if the given value is found somewhere in this list
`remove(`**value**`)`	finds and removes the given value from this list if it is present
`iterator()`	returns an object used to examine the contents of the list

`Set<E>` Methods (11.2) *(A fast-searchable set of unique values)*

`add(`**value**`)`	adds the given value to the set
`contains(`**value**`)`	returns `true` if the given value is found in the set
`remove(`**value**`)`	removes the given value from the set if it is present
`clear()`	removes all elements of the set
`size()`	returns the number of elements in the set
`isEmpty()`	returns `true` if the set's size is 0
`iterator()`	returns an object used to examine the contents of the set

`Map<K, V>` Methods (11.3)

(A fast mapping between a set of keys and a set of values)

`put(`**`key, value`**`)`	adds a mapping from the given key to the given value
`get(`**`key`**`)`	returns the value mapped to the given key (`null` if none)
`containsKey(`**`key`**`)`	returns `true` if the map contains a mapping for the given key
`remove(`**`key`**`)`	removes any existing mapping for the given key
`clear()`	removes all key/value pairs from the map
`size()`	returns the number of key/value pairs in the map
`isEmpty()`	returns `true` if the map's size is 0
`keySet()`	returns a `Set` of all keys in the map
`values()`	returns a `Collection` of all values in the map
`putAll(`**`map`**`)`	adds all key/value pairs from the given map to this map

`String` Methods (3.3)

(An object for storing a sequence of characters)

`charAt(`**`i`**`)`	the character in this String at a given index
`contains(`**`str`**`)`	`true` if this String contains the other's characters inside it
`endsWith(`**`str`**`)`	`true` if this String ends with the other's characters
`equals(`**`str`**`)`	`true` if this String is the same as *str*
`equalsIgnoreCase(`**`str`**`)`	`true` if this String is the same as *str*, ignoring capitalization
`indexOf(`**`str`**`)`	first index in this String where given String begins (-1 if not found)
`lastIndexOf(`**`str`**`)`	last index in this String where given String begins (-1 if not found)
`length()`	number of characters in this String
`isEmpty()`	`true` if this String is the empty string
`startsWith(`**`str`**`)`	`true` if this String begins with the other's characters
`substring(`**`i`**`)`	characters in this String from index *i* (inclusive) to the end
`substring(`**`i, j`**`)`	characters in this String from index *i* (inclusive) to *j* (exclusive)
`toLowerCase(),toUpperCase()`	a new String with all lowercase or uppercase letters

Collections Implementations

`List<E>`	`ArrayList<E>` and `LinkedList<E>`
`Set<E>`	`HashSet<E>` and `TreeSet<E>` (values ordered)
`Map<K, V>`	`HashMap<K, V>` and `TreeMap<K, V>` (keys ordered)