Assignment 3: Heaps

(replacement for P3c & P4a)

DUE: May 3rd at 11:59pm

Extra Credit Available for Early Submissions!

Setup

- Download the `assignment3.zip` and unzip it.
- This will create a folder `section-yourGMUUserName-a3`.
- Rename the folder replacing `section` with the `001`, `002`, `005`, etc. based on the lecture section you are in.
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address.
- After renaming, your folder should be named something like: `001-krusselc-a3`.
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`) .

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-a3.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:
      `001-krusselc-a3.zip --> 001-krusselc-a3 --> JavaFile1.java`
      `JavaFile2.java`
      `...`
- Submit to blackboard. DOWNLOAD AND VERIFY WHAT YOU HAVE UPLOADED THE RIGHT THING. ***Submitting the wrong files will result in a 0 on the assignment!***

Basic Procedures

You must:

- Have code that compiles with the command: `javac *.java` in your user directory.
- Have code that runs with the command: `java SimpleKTree`, `java PriorityTree`, and `java PA3 {min|max}` in your user directory.
- Have a style (indentation, good variable names, etc.) -- you must pass the style checker!
- Comment your code well in JavaDoc style -- you must pass the comments checker!

You may:

- Add additional methods and variables to both provided classes, however these methods **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional *public* methods or variables, or add any additional non-nested classes.
- Add any additional libraries/packages which require downloading or use any code from the internet.
- Import any additional libraries/packages or add any additional import statements (or use the "fully qualified name" to get around adding import statements). Therefore, you may not use any JCF classes that are not already imported for you.
- Alter any method/class signatures defined in this document or the template. Note: "throws" is part of the method signature in Java, don't add/remove these. You may also not alter provided classes/methods that are complete.
- Add `@SuppressWarnings` to any methods unless they are private helper methods for use with a method we provided which already has an `@SuppressWarnings` on it.

Grading Rubric

No Credit

- Non submitted assignments
- Assignments submitted after 5pm the Monday after the due date
- Non-compiling assignments
- Non-independent work
- Code that violates and restrictions or "you may not" mandates.
- "Hard coded" solutions
- Code that would win an obfuscated code competition with the rest of CS310 students

How will my assignment be graded?

- Automatic Testing (100%): To assess the correctness of programs.
- You CANNOT get points for code that doesn't compile or for submitting just the files given to you.
- Extra credit for early submissions:
  - 1% extra credit rewarded for every 24 hours your submission made before the due time
  - Up to 5% extra credit will be rewarded
  - Your latest submission before the due time will be used for grading and extra credit checking. You CANNOT choose which one counts.

Automated Testing Rubric

The JUnit tests used for grading will NOT be provided for you (you need to test your own programs!), but the tests will be based on what has been specified in the project description and the comments in the code templates. A breakdown of the point allocations is given below:

34 pts	SimpleKTree
+1 pt	toStringWithLevels() in SimpleKTree
50 pts	PriorityTree
14 pts	SimpleKTree methods work on PriorityTrees (inheritance…)
5 pts	heapsort() in PA3

-5pts ("off the top")	**Not following the submission format** Note: This is very, *very* important for these assignments; the graders need to return grades *very* fast. If you do not follow the submission format (the same one you've been using for P0, P1, and P2), then they will manually deduct 5pts from your score. No exceptions.
-5 pts ("off the top")	Not passing code style check
-5 pts ("off the top")	Not passing JavaDoc style check

"Off the top" points (i.e. items that will lose you points rather than earn you points).

Assignment Overview

You are going to build a k-ary tree class (building block for a heap), a priority queue class that extends your tree class (a heap), and sort values using your heap. The outline has been provided for these classes, but this time there are many methods which have been left out.

WARNING: Part of this assignment is to read and understand the Java documentation on the required interfaces. Do not ask other students, the TAs, Piazza, or any professors for an explanation of these interfaces.

Tasks Breakdown and Sample Schedule

There are **4** tasks in this assignment. It is suggested that you implement these tasks in the given order:

- Task 0: Get a Compiling Code Base
- Task 1: Make a K-ary Tree
- Task 2: Make a Heap
- Task 3: Sort Things

Need a schedule?

- You've got 1.5 weeks and there are ~25 methods to write, while you also have other classes with exams/projects.
- Assume you want to spend the last half week getting EC or seeking additional help, fill in the following:
    - 4/20-4/23: _____ (first week period)
        - Suggestions: Task 0 & 1
    - 4/24-4/26: _____ (first weekend period)
        - Suggestions: Task 2 & 3
    - 4/27-4/30: _____ (second week period)
        - Suggestions: *Thorough* Testing and Debugging
    - 5/01-5/03: _____ (second weekend period)
        - Suggestions: Turn in early for Extra Credit

Task 0 and Code Overview for Tasks 1-3

This is the first time we've given you a code base that doesn't compile, this is an intentional challenge of this assignment. ***Before you do ANYTHING else, get the code to compile***. This will involve (1) putting in "dummy" return statements and parent constructor calls, (2) looking at the documentation for the provided interfaces and determining which methods you're missing, and (3) uncommenting the blocks of code in **`SimpleKTree`** and **`PriorityTree main()`**. Hints: See previous project templates for "how to" on dummy returns. Also, if you try to compile the classes, the compiler will give you a hint about return types and missing interface methods. Some IDEs may suggest the missing methods to you as well!

> **WARNING:** Do not just skip reading the interface documentation linked below... it will answer *many* of your questions about this assignment.

The SimpleKTree Class

This class, as outlined, is a k-ary tree representation of your data. It needs to implement the **`Collection`** interface (see http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html), however several methods from this interface have already been written for you (and may not be changed). Seven other methods (2 constructors, **`height()`**, and four **`toString`** methods) have been outlined for you and need to be implemented as well. *One* of the **`toString`** methods is extra credit (see grading rubric).

The PriorityTree Class

This class will become your priority queue. It extends **`SimpleKTree`** (in other words, it *is* a **`SimpleTree`**) and implements the **`Queue`** interface (see: http://docs.oracle.com/javase/8/docs/api/java/util/Queue.html). This class will use a **`Comparator`** to determine if elements being added to the queue are greater or less than each other (see: http://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html).

The PA2 Class

This class contains main method (with prompts) and a **`heapsort()`** method. You need to write the **`heapsort()`** method as described in the code template. Keep in mind that the main method uses a PriorityTree with k=3, but do not assume this is how we will be testing, your **`PriorityTree`** and **`SimpleKTree`** must work with other values of k.

Examples for Testing After Tasks 1, 2, and 3

There are no "yays" in the main methods, but you can write your own! When everything's done, you can run the main method in PA3 for an interactive program. You may also want to try changing the k-value for your heap in the main method for PA3 to visually see what's happening with heaps of different sizes.

The remainder of this document contains two sample runs of the PA3 program. ***Make sure you get the exact same output when you do the project.*** If your output looks different, then something's wrong in your implementation.

Note: In the example runs on the next few pages, `user input` is shown in `green highlight and underlined`.

Example Run of PA3 (max heap)

```
>java PA3 max

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 1

Current tree (height 0)
Level order walk: 1
Pre order walk:   1
Post order walk:  1
Level Order with Level Breaks:
1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 2

Current tree (height 1)
Level order walk: 2 1
Pre order walk:   2 1
Post order walk:  1 2
Level Order with Level Breaks:
2
1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 3

Current tree (height 1)
Level order walk: 3 1 2
Pre order walk:   3 1 2
Post order walk:  1 2 3
Level Order with Level Breaks:
3
1 2

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 1

Current tree (height 1)
Level order walk: 3 1 2 1
Pre order walk:   3 1 2 1
Post order walk:  1 2 1 3
Level Order with Level Breaks:
3
1 2 1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 3

Current tree (height 2)
Level order walk: 3 3 2 1 1
Pre order walk:   3 3 1 2 1
Post order walk:  1 3 2 1 3
Level Order with Level Breaks:
3
3 2 1
1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: -10
```

← Runs the main method, requesting to use a max heap.

← Shows height here.

← And all your walks.

← The method that does this is extra credit, but it's really useful for debugging.

```
Current tree (height 2)
Level order walk: 3 3 2 1 1 -10
Pre order walk:   3 3 1 -10 2 1
Post order walk:  1 -10 3 2 1 3
Level Order with Level Breaks:
3
3 2 1
1 -10

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: -1

Current tree (height 2)
Level order walk: 3 3 2 1 1 -10 -1
Pre order walk:   3 3 1 -10 -1 2 1
Post order walk:  1 -10 -1 3 2 1 3
Level Order with Level Breaks:
3
3 2 1
1 -10 -1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 3

Current tree (height 2)
Level order walk: 3 1 2 1 -1 -10
Pre order walk:   3 1 -1 -10 2 1
Post order walk:  -1 -10 1 2 1 3
Level Order with Level Breaks:
3
1 2 1
-1 -10

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 3

Current tree (height 2)
Level order walk: 2 1 -10 1 -1
Pre order walk:   2 1 -1 -10 1
Post order walk:  -1 1 -10 1 2
Level Order with Level Breaks:
2
1 -10 1
-1
```

← Now removing things.

```
1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 2

Current tree (height 1)
Level order walk: 1 -1 -10 1
Pre order walk:   1 -1 -10 1
Post order walk:  -1 -10 1 1
Level Order with Level Breaks:
1
-1 -10 1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 1

Current tree (height 1)
Level order walk: 1 -1 -10
Pre order walk:   1 -1 -10
Post order walk:  -1 -10 1
Level Order with Level Breaks:
1
-1 -10

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 1

Current tree (height 1)
Level order walk: -1 -10
Pre order walk:   -1 -10
Post order walk:  -10 -1
Level Order with Level Breaks:
-1
-10

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: -1

Current tree (height 0)
Level order walk: -10
Pre order walk:   -10
Post order walk:  -10
```

```
Level Order with Level Breaks:
-10

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: -10

Current tree (height -1)
Level order walk:
Pre order walk:
Post order walk:
Level Order with Level Breaks:


1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: null

Current tree (height -1)
Level order walk:
Pre order walk:
Post order walk:
Level Order with Level Breaks:


1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 3
Enter numbers to sort, separated by
spaces: 100 1 -12 44 2 42 0
Sorted: -12 0 1 2 42 44 100

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 4
```

← Note that sorting has nothing to do with the values in the current tree, it only operates on the number provided.

Example Run of PA3 (min heap)

```
>java PA3 min

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 1

Current tree (height 0)
Level order walk: 1
Pre order walk:   1
Post order walk:  1
Level Order with Level Breaks:
1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 2

Current tree (height 1)
Level order walk: 1 2
Pre order walk:   1 2
Post order walk:  2 1
Level Order with Level Breaks:
1
2

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 3

Current tree (height 1)
Level order walk: 1 2 3
Pre order walk:   1 2 3
Post order walk:  2 3 1
Level Order with Level Breaks:
1
2 3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 1
```

← Runs the main method, requesting to use a min heap.

```
Current tree (height 1)
Level order walk: 1 2 3 1
Pre order walk:   1 2 3 1
Post order walk:  2 3 1 1
Level Order with Level Breaks:
1
2 3 1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 3

Current tree (height 2)
Level order walk: 1 2 3 1 3
Pre order walk:   1 2 3 3 1
Post order walk:  3 2 3 1 1
Level Order with Level Breaks:
1
2 3 1
3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: -10

Current tree (height 2)
Level order walk: -10 1 3 1 3 2
Pre order walk:   -10 1 3 2 3 1
Post order walk:  3 2 1 3 1 -10
Level Order with Level Breaks:
-10
1 3 1
3 2

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: -1

Current tree (height 2)
Level order walk: -10 -1 3 1 3 2 1
Pre order walk:   -10 -1 3 2 1 3 1
Post order walk:  3 2 1 -1 3 1 -10
Level Order with Level Breaks:
-10
-1 3 1
3 2 1

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: -10

Current tree (height 2)
Level order walk: -1 1 3 1 3 2
Pre order walk:   -1 1 3 2 3 1
Post order walk:  3 2 1 3 1 -1
Level Order with Level Breaks:
-1
1 3 1
3 2

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: -1

Current tree (height 2)
Level order walk: 1 2 3 1 3
Pre order walk:   1 2 3 3 1
Post order walk:  3 2 3 1 1
Level Order with Level Breaks:
1
2 3 1
3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 1

Current tree (height 1)
Level order walk: 1 2 3 3
Pre order walk:   1 2 3 3
Post order walk:  2 3 3 1
Level Order with Level Breaks:
1
2 3 3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 1
```

```
Current tree (height 1)
Level order walk: 2 3 3
Pre order walk:   2 3 3
Post order walk:  3 3 2
Level Order with Level Breaks:
2
3 3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 2

Current tree (height 1)
Level order walk: 3 3
Pre order walk:   3 3
Post order walk:  3 3
Level Order with Level Breaks:
3
3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 3

Current tree (height 0)
Level order walk: 3
Pre order walk:   3
Post order walk:  3
Level Order with Level Breaks:
3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 2
Removed: 3

Current tree (height -1)
Level order walk:
Pre order walk:
Post order walk:
Level Order with Level Breaks:


1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit
```

```
Enter your choice: 1
Enter a number to add: 5

Current tree (height 0)
Level order walk: 5
Pre order walk:   5
Post order walk:  5
Level Order with Level Breaks:
5

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 1

Current tree (height 1)
Level order walk: 1 5
Pre order walk:   1 5
Post order walk:  5 1
Level Order with Level Breaks:
1
5

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 3
Enter numbers to sort, separated by
spaces: 100 1 -12 44 2 42 0
Sorted: -12 0 1 2 42 44 100

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 1
Enter a number to add: 3

Current tree (height 1)
Level order walk: 1 5 3
Pre order walk:   1 5 3
Post order walk:  5 3 1
Level Order with Level Breaks:
1
5 3

1. Enqueue a number
2. Dequeue a number
3. Sort a list of numbers
4. Exit

Enter your choice: 4
```

← Again, sorting does not affect the current tree.