Laboratory 12: Heaps and Priority Queues

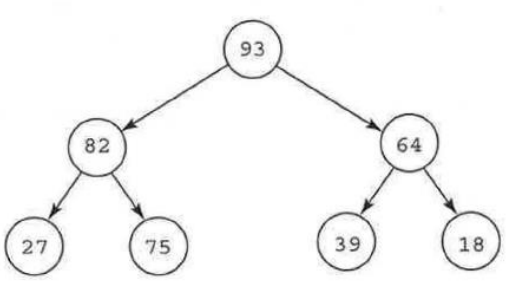
Introduction

This lab introduces Heaps, a type of tree in which the children of each node have a smaller value than the parent (these are actually called Max heaps, as opposed to Min heaps, in which the children are larger than the parent). Heaps are also normally implemented with a new way of representing trees, using an array. Heaps are a very efficient means of keeping data in loosely sorted order, with the largest item always found at the root, and they have log2(N) efficiency in terms of adding and removing items. This leads to a better way of implementing Priority Queues, since the highest priority item is always located at the root of the tree.

Finally, we will take a brief look at Graphs, and how they can be used to represent data with arbitrary patterns of connectivity.

Part A: Heaps

Linked structures are not the only way in which you can represent trees. If you take the binary tree shown below and copy its contents into an array in level order, you produce the following array.



Index	Entry
0	not used
1	93
2	82
3	64
4	27
5	75
6	39
7	18

Examining the relationship between positions in the tree and entries in the array, you see that if an element is stored in entry N in the array, then the element's left child is stored in entry 2N , its right child i stored in entry 2N + 1, and its parent is stored in entry N/2. These mappings make it easy to move through the tree stepping from parent to child (or vice versa). Note that we begin our array at cell 1.

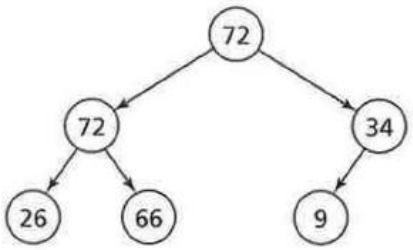
To avoid having nulls in the middle of the array, we have to restrict the trees we represent to "complete" binary trees, which are trees that have all levels completely full with the exception of the last level, which is full starting at the left side up to some final point in the last level.

In this laboratory, you focus on a different type of tree called a heap. A heap is a binary tree that meets the following conditions.

- The tree is complete. That is, every level in the tree is filled, except possibly the bottom level. If the bottom level is not filled, then all the missing elements occur on the right.

- Each element in the tree has a corresponding value. For each element E, all of E's descendants have values that are less than or equal to E's value. Therefore, the root stores the maximum of all values in the tree. (Note: In this laboratory we are using a max-heap. There is another heap variant called a min-heap. In a min-heap, all of E's descendants have values that are greater than or equal to E's value.)

The tree shown at the beginning of this laboratory is a heap, as is the tree shown below.



Interface

The interface for MaxHeap is shown below. Note that you can only remove the largest item in a MaxHeap:

```
public interface MaxHeapInterface<T extends Comparable<? super T>>
{
   /** Adds a new entry to the heap.
      @param newEntry  an object to be added */
   public void add(T newEntry);

   /** Removes and returns the largest item in the heap.
      @return either the largest object in the heap or,
              if the heap is empty before the operation, null */
   public T removeMax();

   /** Retrieves the largest item in the heap.
      @return either the largest object in the heap or,
              if the heap is empty, null */
   public T getMax();

   /** Detects whether the heap is empty.
     @return true if the heap is empty, else returns false */
   public boolean isEmpty();

   /** Gets the size of the heap.
      @return the number of entries currently in the heap */
   public int getSize();

   /** Removes all entries from the heap. */
   public void clear();
} // end MaxHeapInterface
```

Activities

1) Open the Heaps folder in the lab12 download and run the PartAHeapDriver. The first part of this program creates a MaxHeap and adds four values to it. We will start by defining the add, removeMax and getMax methods (**solutions of which are in the MaxHeap-Working.txt file in the project folder**).

   Then Add code to problem 1 that uses a loop to remove and print all the items in the MaxHeap.

2) Because a heap is a complete tree, an unambiguous representation in tree form can be generated by outputting the heap level-by-level, with each level output on a separate line.

```
void writeLevels ( )
```

Precondition:

None.

Postcondition:

Outputs the elements in a heap in level order, one level per line. If the heap is empty, then output "Empty heap".

The tree shown on the first page of this laboratory, for example, yields the following output.

```
93
82 64
27 5 39 18
```

Add this method to the MaxHeap class. Then in problem 2 in PartAHeapDriver add 10 values to the heap and print the heap in level order.

Part B: Priority Queues

The Priority Queue ADT we've worked with earlier has a behavior that is easily modelled with a Heap. The interface for the PriorityQueue is as follows:

Interface

```
/**
   An interface for the ADT priority queue.

   @author Frank M. Carrano
   @version 3.0
*/
public interface PriorityQueueInterface <T extends Comparable<? super T>>
{
   /** Adds a new entry to this priority queue.
       @param newEntry  an object */
   public void add(T newEntry);

   /** Removes and returns the item with the highest priority.
       @return either the object with the highest priority or, if the
               priority queue is empty before the operation, null */
   public T remove();

   /** Retrieves the item with the highest priority.
       @return either the object with the highest priority or, if the
               priority queue is empty, null */
   public T peek();

   /** Detects whether this priority queue is empty.
       @return true if the priority queue is empty, or false otherwise */
   public boolean isEmpty();

   /** Gets the size of this priority queue.
       @return the number of entries currently in the priority queue */
   public int getSize();

   /** Removes all entries from this priority queue */
   public void clear();
} // end PriorityQueueInterface
```

Activities

3) Implement the priority queue class as an adapter for MaxHeap according to the interface given above. In other words, declare a MaxHeap as the private field inside the PriorityQueue and use the MaxHeap methods to implement all the Priority Queue methods.
4) The file OS_Simulator uses a Priority Queue of Jobs to simulate the way computers process tasks. The Job class is defined, but it needs to be augmented to implement Comparable<Job>. Do this now.
5) Add a main method to your PriorityQueue class to demonstrate that it works properly.

Operating System Simulation

Operating systems commonly use priority queues to regulate access to system resources such as printers, memory, disks, software, and so forth. Each time a task requests access to a system resource, the task is placed on the priority queue associated with that resource. When the task is dequeued, it is granted access to the resource-to print, store data, and so on.

Suppose you wish to model the flow of tasks through a priority queue having the following properties:

- One task is dequeued every minute (assuming that there is at least one task waiting to be dequeued during that minute).
- From zero to two tasks are enqueued every minute, where there is a 50% chance that no tasks are enqueued, a 25% percent chance that one task is enqueued, and a 25% chance that two tasks are enqueued.
- Each task has a priority value of zero (low) or one (high), where there is an equal chance of a task having either of these values.

You can simulate the flow of tasks through the queue during a time period n minutes long using the following algorithm:

```
Initialize the queue to empty.
for ( minute = 0 ; minute < n ; ++minute )
{
      If the queue is not empty, then remove the task at the front of the queue.
      Compute a random integer k between 0 and 3.
      If k is 1, then add one task to the queue. If k is 2, then add two tasks.
            Otherwise (if k is 0 or 3), do not add any tasks to the queue.
            Compute the priority of each task by generating a random value, 0 or 1
            (assuming there are only 2 priority levels).
}
```

These steps are similar to the ones used in the simulation program for the Queue ADT in Laboratory 4. Therefore, it may help to review the file StoreSim.java in the Lab4 Java download. Notice that in OS_Simulator.java the number of priority levels and the length of the simulation are read from the keyboard.

Step 1: Using the program shell given in the file OS_Simulator.java as a basis, create a program that uses the Priority Queue ADT to implement the task scheduler described above. Your program should output the following information about each task as it is dequeued: the task's priority, when it was enqueued, and how long it waited in the queue.

Step 2: Use your program to simulate the flow of tasks through the priority queue and complete the following table.

Time	Longest wait for any low priority (0) job (minutes)	Longest wait for any high priority (1) job (minutes)
10		
30		
60		
600		

Step 3: Is your priority queue task scheduler unfair-that is, given two tasks T1 and T2 of the same priority, where task T1 is enqueued at time N and task T2 is enqueued at time N + i (i > 0), is task T2 ever dequeued before task T1? If so, how can you eliminate this problem and make your task scheduler fair?