

1 Objectives

- Learn how to implement a priority queue.
- Get practice using the Java library `PriorityQueue`.

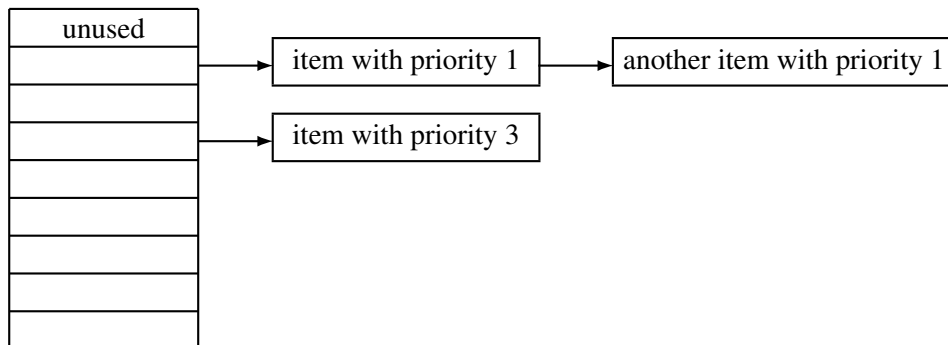
2 Preliminaries

Create a new project for today's lab and import the files you will need from

`~csci204/2009-fall/student/labs/lab11`

Commit the files to your subversion repository.

`FastPriorityQueue` implements a priority queue by using an `ArrayList`. Each component of the `ArrayList` is a linked list that contains elements with the same priority.



Component 1 of the `ArrayList` is a linked list that contains the priority 1 items, component 2 has a list of the priority 2 items, etc. If a list is empty, there are no items with that priority. Items with priority 1 have the highest priority. **Note:** Component 0 of the `ArrayList` is unused.

The `FastPriorityQueue` constructor requires a parameter that specifies the highest priority. The constructor needs this information so that it can create an appropriately sized `ArrayList`. Look at the constructor code and make sure you understand what it is doing.

3 Exercise 1: Complete your FastPriorityQueue

You need to complete three methods for this class to work.

The `add()` method has two parameters. One is the priority of the item being added and the other is the item itself. The method needs to add the item to the end of the list specified by the priority.

The second method you need to complete is the `remove()` method. This method will search through the lists sequentially until it finds one that is not empty. It will remove the first item from the list and return it. If all of the lists are empty, it returns `null`.

The last method you need to complete is the `size()` method. This method should do exactly what you think it should do. Implement it!

The `WorkOrder` class can be used to test the `FastPriorityQueue`. Make a `readme.txt` file that contains the result of running `WorkOrder`'s `main()` method with your completed `FastPriorityQueue`. Make sure that `readme.txt` is committed to SVN.

4 Exercise 2: Use Java's Priority Queue

The Java library has a `PriorityQueue` class that is very similar to the one that you just completed. In this exercise, you will learn how to use the Java priority queue by implementing a similar example using the Java library instead of the `FastPriorityQueue` you wrote for the first exercise.

The primary difference when using the Java priority queue is that the `add()` method has just one parameter, the item being added. The items that you add to Java's priority queue need to know how to compare their priority to another item's priority.

4.1 Create a Job Order Class

Create a new class called `JobOrder` that will be very similar to our `WorkOrder` class. There are also some important differences.

4.1.1 Job Order Constructor

A `JobOrder` object will require two instance fields. One is an integer that contains its priority, and the other will be a string containing its description. The `JobOrder` constructor requires two parameters that will initialize these fields.

4.1.2 toString() Method

Your `JobOrder` class should have a `toString()` method. The output should contain the item's priority and its description. Your output should have the format "Priority %d: %s". For example, a `JobOrder` class with priority 1 and description "pay bills" would have the string representation "Priority 1: pay bills".

4.1.3 Make it Comparable

Java's `PriorityQueue` class will ask these objects to compare themselves to one another so that it can honor their priorities. Your class *must* implement `Comparable<JobOrder>`. This means that your class will have a `compareTo()` method that compares two `JobOrder` objects.

```
job1.compareTo(job2)
```

If `job1` has a higher priority than `job2` this will return a negative number. It will return 0 if they have the same priority, and a positive number if `job1` has lower priority. What values correspond to higher and lower priority, respectively? Which positive and negative numbers should it return? Answer these questions in your [readme.txt](#).

4.1.4 Main Method

Create a `main()` method that tests your class. Use Java's `PriorityQueue` instead of your `FastPriorityQueue`. Use the same items that we put into the `FastPriorityQueue`. That is, they will have the same priority and description, but they will be instances of `JobOrder`. See the Java API for `PriorityQueue` to add items to it.

Insert these in the same order as before and then use a loop to remove them until the queue is empty. Print each item as it is removed. Copy the output to your [readme.txt](#).

5 Upon Completion

Make sure all your methods have *good* Javadoc comments and then save them to the repository. Do not forget to commit your [readme.txt](#).

Note: For Exercise 2, the items at the same priority may come out in a different order. Java is not using the standard *FIFO* rule that you used in Exercise 1.