

# Project 6: Priority Queue and Heap

Implement a heap using an array.

<b>Overview</b>	<b>1</b>
<b>Learning Goals:</b>	<b>1</b>
<b>Style Guide:</b>	<b>1</b>
<b>Project Specification:</b>	<b>2</b>
<b>-Tasks and TODOs</b>	<b>3</b>
<b>Export and Submit</b>	<b>6</b>

---

## Overview

For this assignment you will first implement a heap using an array, then use the heap to implement a min-priority queue and a max-priority queue.

## Learning Goals:

- Develop your understanding of the priority queue abstraction and its implementation as a heap.
- Use the ArrayList class to implement a heap.
- Work with the Comparator interface, an abstraction of comparison behavior.
- Gain experience implementing algorithms in an Object-Oriented code base.
- Demonstrate your ability to read and understand a specification by implementing it.
- Experience debugging and testing code.

## Style Guide:

Follow these style guidelines:

1. All unnecessary lines of code have been removed.
2. Proper indenting must be used.
3. Optimal use of blank lines and spaces for operators.
4. Use of camelCase for variable/parameter and method names.

5. Variables declared early (not within code), and initialized where appropriate and practical.
6. Descriptive variable and method names.

## Project Specification:

### Background.

This project works with several Java library classes: Comparator and ArrayList. Some background and references in the Java API are provided in the Tasks and TODOs section. Refer to the descriptions of the implementation of heaps and priority queues in that section as well as in your text and in other course material.

### Code Structure.

This project contains the following important folders:

**lib:** This is where you can find libraries that are included with the project. At the very least you will find two jar files that are used to run the JUnit test framework.

**src:** This is the source folder that contains the following subdirectories. All code you submit will be in one or more of these subdirectories. You do not create any new subdirectories.

**src/test:** This is the package folder where you can find all of the public unit tests.

**src/comparators:** This directory contains the Comparators you will develop and use in the heap and priority queue classes.

**src/structures:** This directory contains the heap and priority queue classes you will develop.

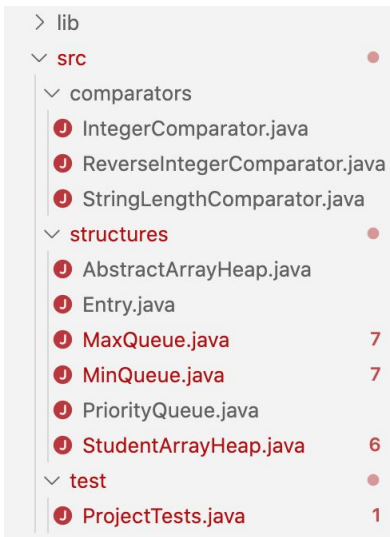


Fig 1: Code structure for the project.

## Tasks and TODOs

There are two main “problems” in this project, each problem containing a number of TODOs. In this project, you will be providing entire implementations of classes that extend an abstract class or implement interfaces.

### Problem 1: Extend and Implement AbstractArrayHeap

For this part of the assignment, you will implement a heap. A heap is a binary tree of ordered objects that respects the heap property. The heap property states that, for every node in the tree, the children of that node will be “less than (or equal to)” the node itself. In addition to this the tree must be complete. Your implementation will keep the tree in an instance of the `java.util.ArrayList` class (<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>). An `ArrayList` contains an array, but provides an easy to use interface and deals with generic types in a more convenient way than a basic array in Java (You can also see that `ArrayList` has an `iterator` method). The `ArrayList` class is also part of the Collections framework, and so the `sort` method can be used, as well as other useful Collections methods.

Start this part of the assignment by opening up the `AbstractArrayHeap` class. Review what is already provided for you and what you have to implement. Your implementation code must be written in the class named `StudentArrayHeap` that extends `AbstractArrayHeap`. There is a file with that name already in your `src` directory. It will not compile as distributed; you will need to complete it.

Recall that a class that is a subclass of an abstract class has access to the public and protected instance variables and methods in the abstract class it extends. Furthermore, the subclass must provide an implementation for all methods in the superclass declared as abstract (there are 5). Three of these methods are simple index calculations. The majority of your work is implementing the `bubbleUp` and `bubbleDown` methods. Use the code in `AbstractArrayHeap` as an example to guide you in implementing the abstract methods in the `StudentArrayHeap` class. Remember, you do not repeat any variables or methods that are already implemented in the superclass.

A note on the public final `List<Entry<P, V>> asList()` method in the `AbstractArrayHeap` class. It is worth mentioning that it is important to guard a collection of data against modifications made by multiple users. Sometimes a user may modify the data in a collection while another user is accessing it. This method returns `Collections.unmodifiableList(heap)`, which is an “immutable” copy of the current heap, which means the original heap is safe from modification.

## **Heap Data**

Your heap is going to store pairs consisting of a priority and a value. For comparison purposes all that will be compared is the priority. The definition of the class is generic and uses two type variables `P` and `V`. `P` is the type of the priority values and `V` the type of the paired value. Note that the declaration for `P` doesn't state that it must extend `Comparable`. So how do we compare them? The constructor for the `AbstractArrayHeap` class takes an argument, a comparator of type `Comparator<P>`. This allows us to provide arbitrary ordering for Priorities. If we want a max-heap, we simply pass in a `Comparator` that says larger values come first.

Since we are storing priority-value pairs you will notice that our `add` method is a little bit different in that it takes in two arguments: a priority and a value. This allows us to insert two elements with the same value but different priorities. After receiving these inputs, we transform them into an `Entry<P,V>` and use this to store our element. This is commonly referred to as a priority-value pair.

## **BubbleUp and BubbleDown**

How does `bubbleUp` work? The `add` method simply creates an entry and adds it to the end of the array. Then it calls to `bubbleUp` on the index of the newly added element. The `bubbleUp` method will cause the new entry to float up the heap until it is in a location such that the properties of a heap are maintained.

How does `bubbleDown` work? The `remove` method has the opposite problem; it removes the first element in the array, and replaces it with the very last. It then calls `bubbleDown` to bubble the element down until the heap property is restored.

These methods need to do comparisons. Note that the `StudentArrayHeap` class constructor takes a `Comparator` argument. Use the `Comparator` object's `compare` method to make comparisons in both of these methods.

**NOTE:** The `StudentArrayHeap` constructor must be implemented before the JUnit test file will compile. See this TODO in the `step()` method in `ProjectTests.java`:

// TODO: Implement the constructor in `StudentArrayHeap` to get rid of this compilation error. You can use the call as a guide if needed.

## Problem 2: Implement `MinQueue` and `MaxQueue`

Using the `StudentArrayHeap` class implemented in the previous part, finish two classes: `MinQueue<V>` and `MaxQueue<V>`. Each of these should be a `PriorityQueue` that uses `Integer` classes to define priorities. The `MinQueue` is implemented such that lower integer values have the highest priority. The `MaxQueue` is implemented such that higher integer values have the highest priority. (Stubs for these classes are present in your `src/structures` directory).

## Comparators

To do this part, you will also need to finish two classes, one called `IntegerComparator` and the other called `ReverseIntegerComparator`, both of which implement the `Comparator<Integer>` interface (see <https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html>).

A `Comparator` provides a way to abstract the code that determines how comparisons are done. For example, if we wanted to compare `Employee` objects, we could compare them by last name, then first name. We have used the `Comparable` interface to do that by making the `Employee` class implement `Comparable`, and the code that determines how `Employees` are compared is written in the `compareTo` method. But what if we wanted to compare `Employees` by some other means, such as by the number of years they have been with a company? We would have to change the `compareTo` code. The problem then is that what if we wanted to switch back to comparing by name? The solution is to encapsulate the comparison code as `Objects` that implement the `Comparator` interface. Each type of comparison can be passed to the `Collections.sort` method, along with the `ArrayList` to achieve the ordering desired (see <https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html>, and look for the `sort` method in the method summary).

In this project, you will complete two `Comparator` classes that deal with `Integers`. There are stubs for these classes already in your `src/comparators` folder. An example `Comparator` has been provided for you: `comparators.StringLengthComparator`. This `Comparator` orders `Strings` such that shorter `Strings` have lower priority than longer `Strings`.

**Remember: Do not modify any existing instance variables, method signatures or any of the starter code provided.**

# Export and Submit

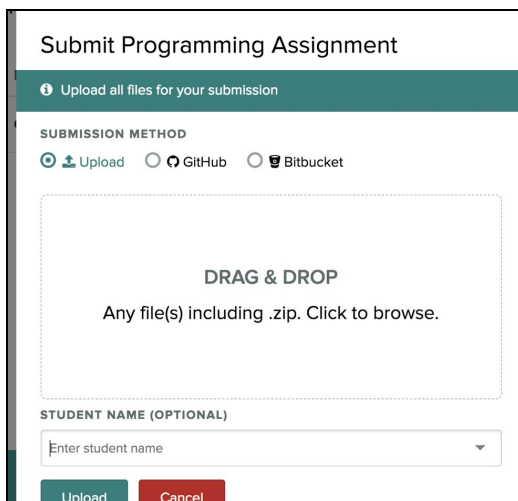
## Step 1: Export your project

Use the Archive extension or other means to create the correct zip file. If your zip file is not in the correct form, the autograder will not process your code and you will not receive any credit. The autograder will not run (often stating an error occurred) if your code does not compile or if your zip file is not correctly made.

***Note that we will not grade code manually. If your code does not compile or if your zip file is not correctly structured you will not receive any credit for the project. Do not import any libraries or change the starter code method signatures.***

## Step 2: Submit the zip file to Gradescope

Log into Gradescope, select the assignment, and submit the zip file for grading.



The screenshot shows a web form titled "Submit Programming Assignment". At the top, there is a teal banner with a white information icon and the text "Upload all files for your submission". Below this, the "SUBMISSION METHOD" section has three radio buttons: "Upload" (selected), "GitHub", and "Bitbucket". The "Upload" option is accompanied by a green upload icon. Below the radio buttons is a large dashed rectangular box containing the text "DRAG & DROP" and "Any file(s) including .zip. Click to browse." Below this box is a section labeled "STUDENT NAME (OPTIONAL)" with a text input field containing the placeholder "Enter student name" and a dropdown arrow. At the bottom of the form are two buttons: a green "Upload" button and a red "Cancel" button.

There are usually more tests in the autograder than provided with the starter code. If your code passes all provided tests, it is a good indication that your code will pass all of the autograder tests. If that is not the case, you are likely not considering some of the “edge” cases in your algorithm. Re-examine your code, use the debugger to troubleshoot. Pose specific questions on piazza for some outside help. Remember that these projects take longer than you estimate. Starting early means you have more time to get help if you are stuck.

The autograder will not run successfully if you do submit a **correctly formatted zip file**- it has to have the same **names and directory structure as described above**. The autograder will also

not run if your code **does not compile**, or if you **import libraries** that were not specifically allowed in the instructions.

**Remember, you can re-submit the assignment to gradescope as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does. Attend office hours for help or post your questions in Piazza.**