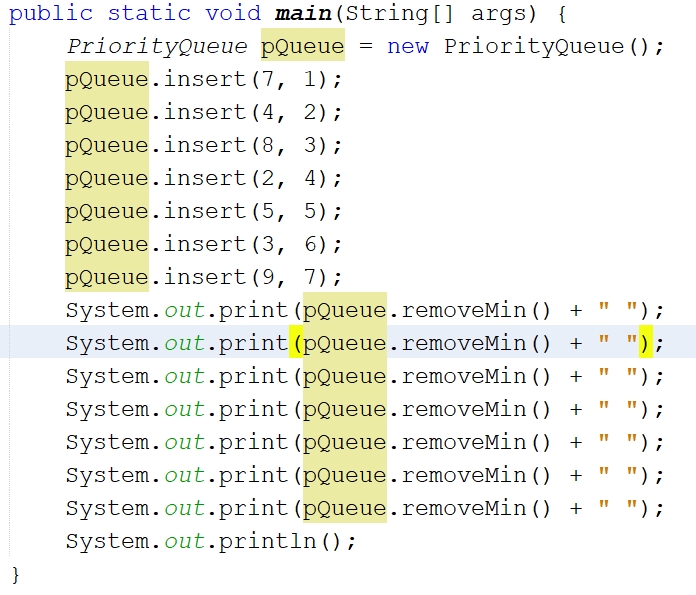
**Program 1**

This program allows for the implementation of the PriorityQueue class.



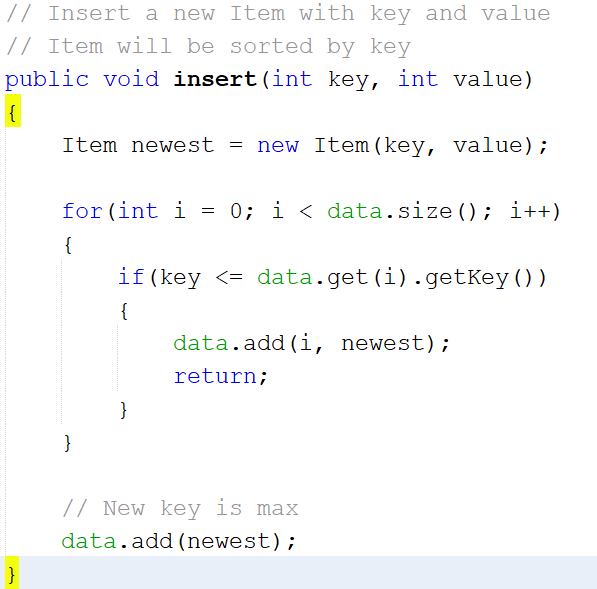


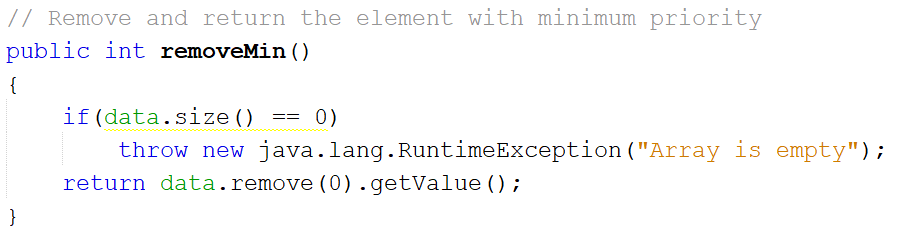
|  |  |
| --- | --- |
| Class | PriorityQueue |
| Fields | Data |
| Behaviors | insert(key, value), removeMin() |

Upon construction, PriorityQueue establishes a list of Items. Each Item stores 2 integers: key and value.

|  |  |
| --- | --- |
| Class | Item |
| Fields | key, value |
| Behaviors | getKey(), getValue() |

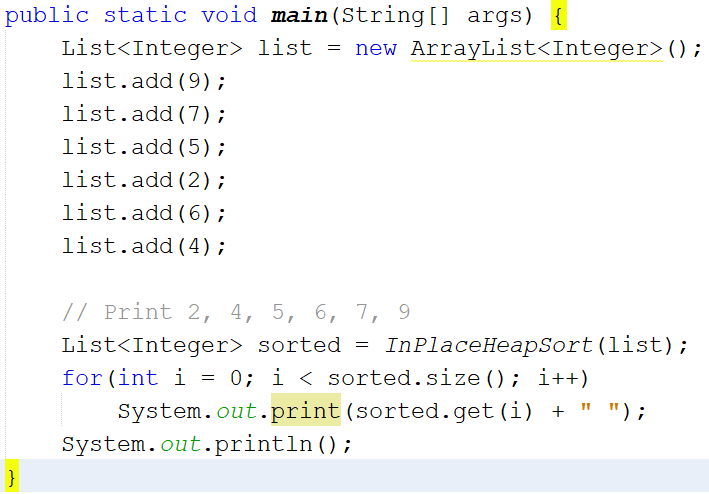
PriorityQueue’s data is sorted by key, from lowest to highest. insert() searches through the data for an index to insert the new Item at based on its key. If the entirety of the data is iterated through and no index is found, the new Item is inserted at the end of the data.



removeMin() returns the Item with the lowest priority. If this method is called until the data is empty, the sorted Items will be returned. 

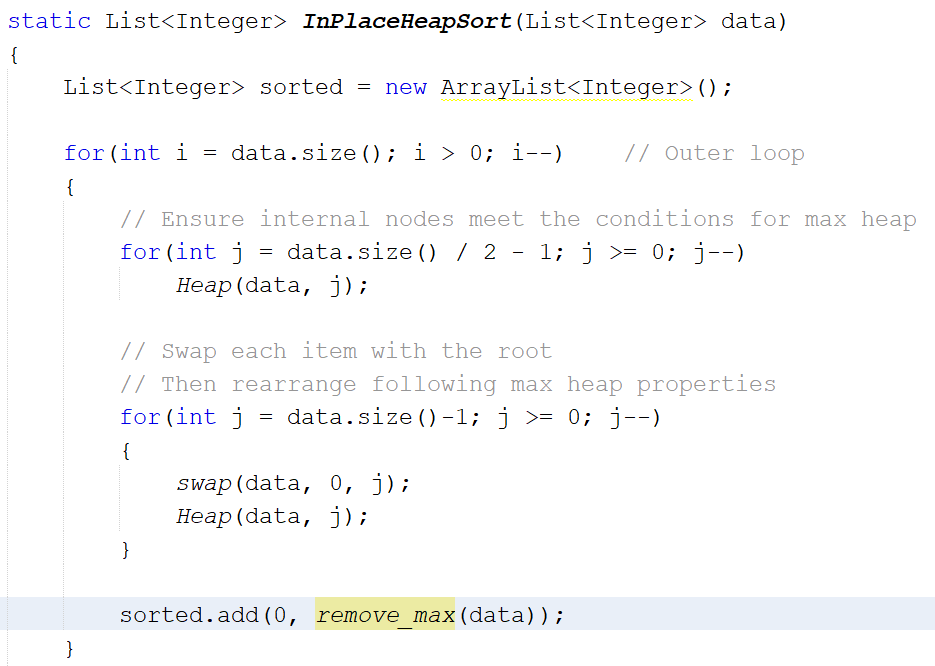
**Program 2**

This program implements methods for in-place heap-sort.

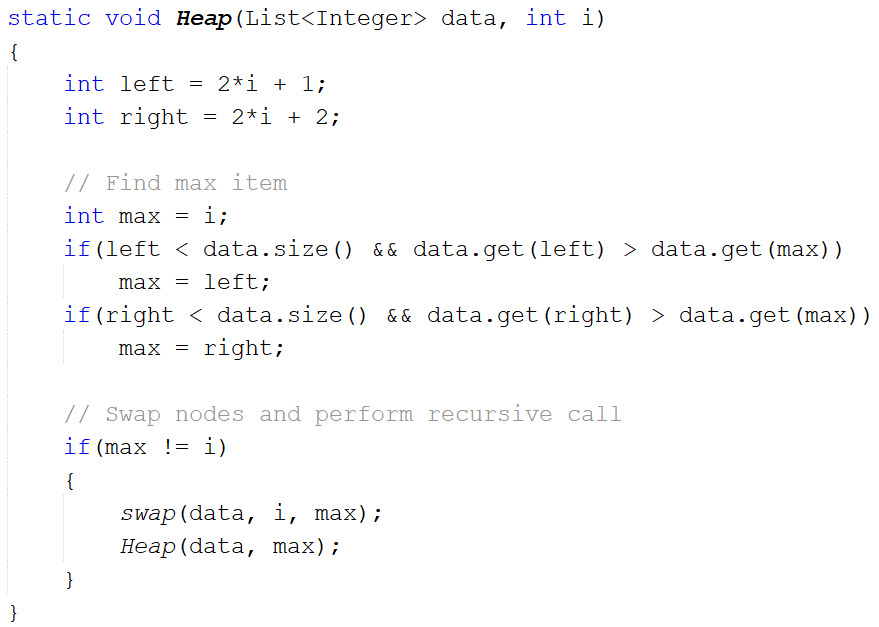




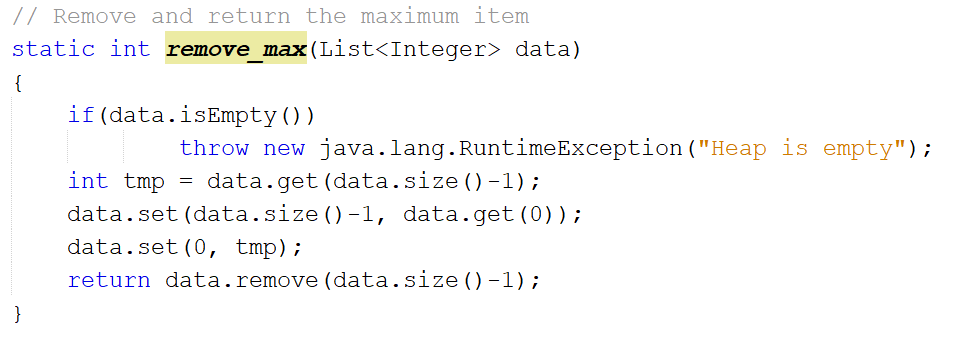
A heap should be represented by a List<Integer>, where the index of each integer is its rank in the heap. The program sorts the heap into a maximum heap, where the parent node is greater than each of its children. The program assumes that each internal node of the heap has 2 children, with the exception of the parent of the rightmost external mode, which will have 1 child if there are an odd number of nodes.



InPlaceHeapSort(data) iterates through each item in the data. During each iteration, 2 inner loops are run. The first inner loop ensures that each internal node and its children follow the max heap properties. The second inner loop iterates through the data backwards, swapping each item with the root item and then ensuring the root (at its new position) and its children follow the max heap properties. Once the max heap is properly arranged, the maximum item is removed and added to the sorted list. This process repeats until the heap is empty.



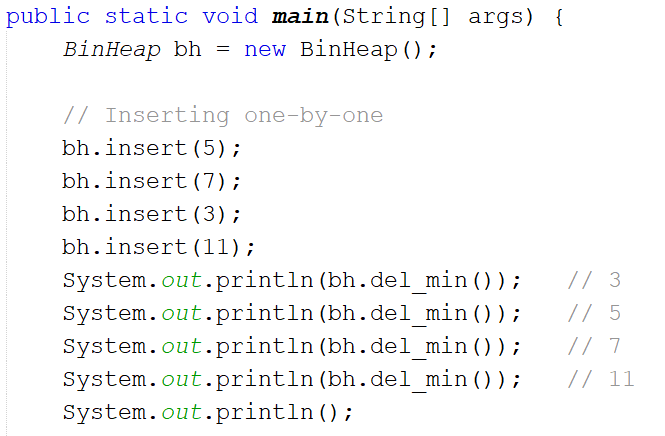
Heap(data, i) looks at i and its 2 children (if they exist). The method finds the maximum value among the parent and children, and then – if a child is greater than its parent – swaps the parent and child values. If a swap takes place, the method makes a recursive call, passing in the updated child node. Otherwise, the method’s execution ends.

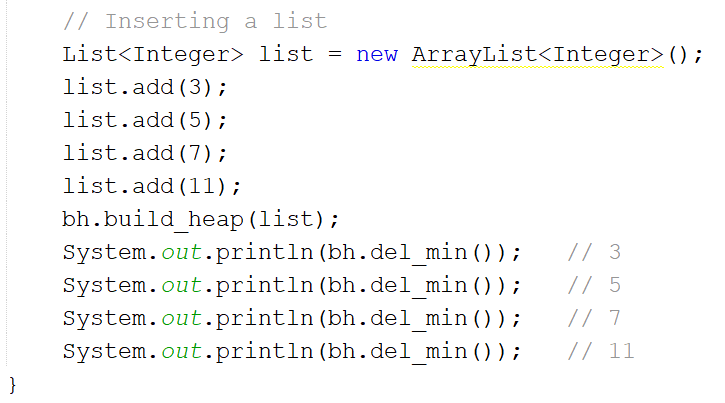


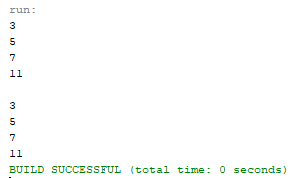
remove\_max(data) removes the maximum value of the heap. The method assumes that data has already been sorted into a max heap. Once removed, the item with the highest rank takes the maximum item’s place. Because this new item may not be the maximum value in the heap, it will need to be sorted again.

**Program 3**

This program allows for the implementation of the BinHeap class. This class can be used to create and modify a minimum binary heap, where each parent node is less than its children and each internal node has 2 children, with the exception of the parent of the rightmost external mode, which will have 1 child if there are an odd number of nodes.

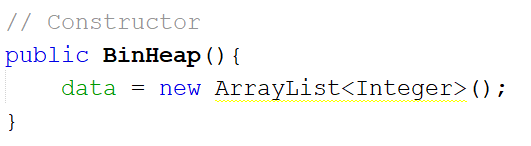


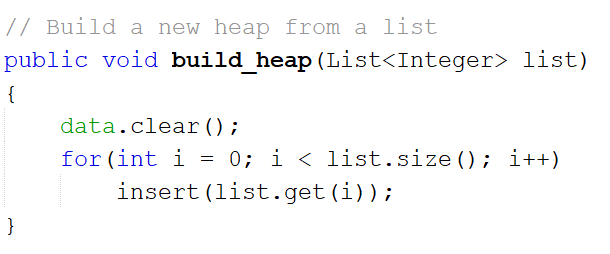




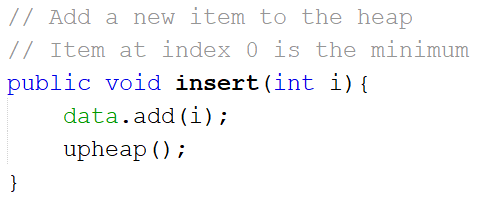
|  |  |
| --- | --- |
| Class | BinHeap |
| Fields | data |
| Behaviors | insert(key), find\_min(), del\_min(), is\_empty(), size(), build\_heap(list), upheap(), downheap(), swap(x,y) |

A new BinHeap can be constructed with the constructor method and a series of input() methods, or with the build\_heap() method.

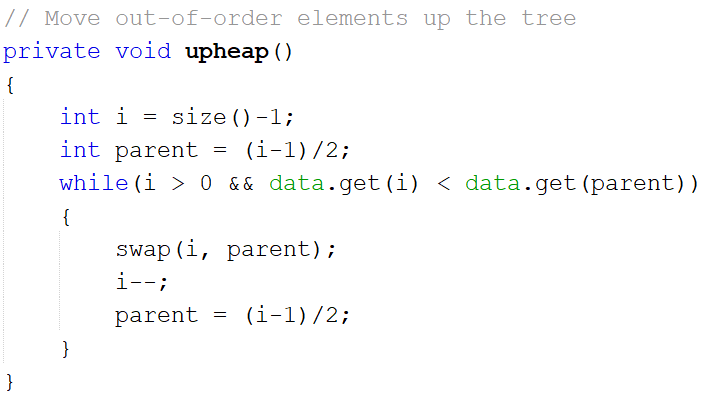




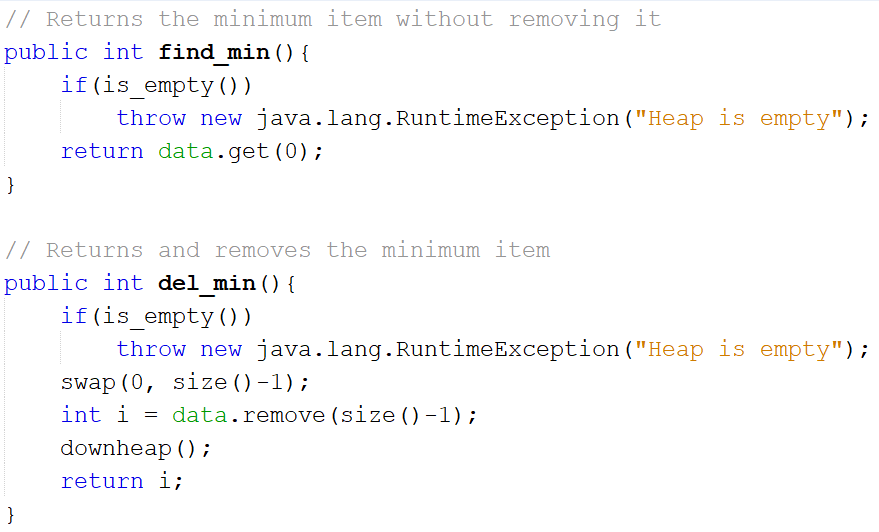
The items in the heap are inserted using insert().



upheap() ensures that the properties of a minimum binary heap are maintained after a new item is added. This new item gets swapped with its parent until it is less than its children.



The current minimum item of the heap can be returned using find\_min() or del\_min(). del\_min deletes the current minimum item and replaces it with the highest-ranked item.



downheap() ensures that the properties of a minimum binary tree are maintained after the minimum item is deleted. The new minimum item is swapped with its smaller child until it is less than its children.

