

Implementing the Queue ADT with a linked list

Lecturer: Arran Stewart

Overview

- ▶ How can we implement the Queue abstract data type using a linked list?

Queue ADT implementations

- ▶ We have seen previously how to implement the Queue ADT using *arrays*.
- ▶ But we saw that this had some disadvantages – for instance, the queue had a fixed maximum capacity.
- ▶ We can instead implement the Queue ADT using a *linked list* as the underlying data structure, which gives us more flexibility (though it makes some operations slower).

Example linked list code

- ▶ We will discuss the code in the `LinkedListQueue.java` file.

Queue ADT operations

Recall the operations a queue should support:

- ▶ A `Queue()` constructor to create an empty queue
- ▶ `enqueue(int item)` to add an item to the queue
- ▶ `int dequeue()` to remove an item from the queue

Outline of the Queue ADT implementation

- ▶ We will keep references to two things – the first and last elements of the queue.
- ▶ For convenience, we will also maintain a length variable as before to keep track of the number of elements in the queue.

Creating an empty queue

- ▶ When creating an empty linked list queue, we set both head and tail to `null` and length to 0, since there are no `ListNodes` in the queue yet.
- ▶ We will also define an `isEmpty()` method, the same as we did in the array implementation.
- ▶ We don't need an `isFull()` method (though if we wanted, we could write one that simply returns `false` all the time).
- ▶ We provide operations for users of the class to “get” the head, tail and length values.
 - ▶ These methods are called “getters” or *accessor methods*.

The dequeue operation

- ▶ We will discuss how to dequeue elements.
- ▶ The dequeue operation removes an element from the front (head) of the queue, so we look at the head link.
- ▶ If the queue is empty we can't dequeue anything, so we throw an exception to flag the error.
- ▶ Where should the new head be? It will be the next element after the first (which has now left the queue).
 - ▶ The new head is referenced by `head.next`, so we set `head = head.next`.
 - ▶ We also set `length = length - 1` to track that we have one fewer elements.

Garbage collection

- ▶ What happens to the old head nodes? Won't they be floating around and taking up space?
- ▶ No. The Java virtual machine (JVM) looks after this through a process called “garbage collection”.
- ▶ When objects are no longer being used anywhere in a program, the memory that was allocated to the object is released, and can be used to create more new objects.

The enqueue operation

- ▶ We will discuss how to enqueue elements.
- ▶ We will need to
 - ▶ create a new `ListNode`
 - ▶ add the new node to the back of the queue – so we will alter the “tail” link.
- ▶ We consider two cases:
 1. the queue does not have any elements yet (it `isEmpty`), and
 2. the new element is being appended to the end of a queue with elements.

The enqueue operation

- ▶ We need to take care about the order in which we do things – we must make sure we maintain links to all the nodes we need, and don't “lose” any.