CMPT 225: Data Structures & Programming – Unit 09 – Queues

Dr. Jack Thomas
Simon Fraser University
Spring 2021

Today's Topics

- The Queue!
- ADT for the Queue
- The Queue in Java
- Implementing and Analyzing
- Something else
  - Oooh! Mystery!

What is a Queue?

- Exactly what it sounds like – a queue, a line of things in the order they were added.
- A Queue is a data structure, with an ADT and implementations in most languages.
- It follows the “Fist In, First Out” rule, or FIFO.
- Each element added to the Queue goes behind the previous element.
- When you start taking things out of the Queue, you have to start with the oldest element, then the next-oldest, and so on until you get back to the most recent element.

Woah, Deja-Vu!

- Hah, more like Deja-*Queue!*
- I'm sorry.
- If you haven't noticed, **a Queue is extremely similar to a Stack** – the only difference in concept is the Stack is FILO (first-in-last-out) and the Queue is FIFO (first-in-first-out).
- That seemingly minor change has a **notable impact** on the implementation, when a queue is useful, etc.
- A point on terminology: when talking about **Stacks**, we tend to imagine them **top to bottom**, while **Queues** are described as **front to back**.

Just to Make Sure We're On The Same Page

- Here's a picture of a real-life queue at an airport, which were things we'd go to in the before-time, when travel was possible.

Image credit: https://www.internationalairportreview.com/article/81738/smarter-way-cut-queue/

Queue: The ADT

- A Queue stores a set of objects.
- Follows (FIFO) (first-in-first-out).
- Standard Queue operations include:
  - **Enqueue**: Add an element to the back of the queue.
  - **Dequeue**: Remove and return the element at the front of the queue.
  - **Front**: Return what's at the front of the queue without removing it.
  - **Size**: How many things are in the queue?
  - **isEmpty**: Is the queue empty? Yes or no.

Queue Examples in Software

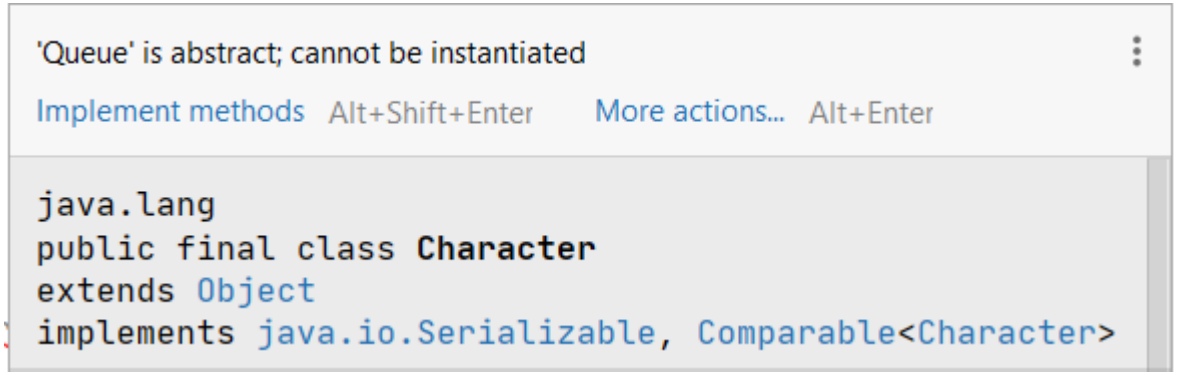
- **Resource scheduling**, as anyone who's been stuck waiting for the print queue to clear can tell you.
- In **multi-programming**, keeping track of when each program gets to submit operations to the processor is important.
- Essentially any situation of **planning out a sequence of future actions**, which makes sense with how Stacks are good at tracking a history going backward.

Queue in Java

- Java has a standard Queue in java.util, let's try it out!

```
Queue<Character> exampleQueue = new Queue<Character>();
```

- Wait, what?



Queue in Java

- It turns out, **the standard Queue in Java is actually an interface**, meaning it exists to be implemented by other Queues, but is not a standardly useable class itself.
- In practice, **LinkedList is a Queue**, and even implements the Queue interface.
- This is a useful reminder that **there's a difference between the ADTs for various data structures and their concrete implementations** in the code – the LinkedList Java class is a Queue, but is one instance of the Queue idea, which exists beyond Java.

Implementing a List-Based Queue

- Let's implement our own version of Queue in Java for storing characters.

```
class Node {
    protected char letter;
    protected Node next;

    Node(char input) {
        letter = input;
        next = null;
    }
}
```

```
class ListQueue {
    protected Node head;
    protected Node tail;
    protected int size;
    public ListQueue() {
        head = null;
        tail = null;
        size = 0;
    }
```

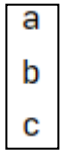
Implementing a List-Based Queue

```
public void enqueue(char element) {
    Node node = new Node(element);
    node.next = null;
    if (size == 0) {
        head = node;
    }
    else {
        tail.next = node;
    }
    tail = node;
    size++;
}
```

```
public char dequeue() throws RuntimeException{
    if(size == 0) {
        throw new RuntimeException("Queue is empty");
    }
    char result = head.letter;
    head = head.next;
    size--;
    if (size == 0){
        tail = null;
    }
    return result;
}
```

Implementing a List-Based Queue

```
ListQueue testQueue = new ListQueue();
testQueue.enqueue( element: 'a');
testQueue.enqueue( element: 'b');
testQueue.enqueue( element: 'c');
System.out.println(testQueue.dequeue());
System.out.println(testQueue.dequeue());
System.out.println(testQueue.dequeue());
```



Analyzing Our Queue

- Time analysis per method:
  - **size:** O(1)
  - **isEmpty:** O(1)
  - **front:** O(1)
  - **enqueue:** O(1)
  - **dequeue:** O(1)
- The only drawback: each element takes up much more memory space than usual by being stored in a node object.
- We could try implementing with a list instead!

That’s It, Right?

- Stacks and Queues seem to cover what you’d need from at least this type of data structure.
- There’s a linear ordering of elements, and you’re retrieving either the oldest or most recent.
- What’s left? Pulling at random from the middle?
- No.

Deques: The Forbidden Queue

- They're not forbidden, they're just annoying to say aloud. (Pronounced "deck", apparently).
- They're **Double-Ended Queues**, meaning you can take **either** the first **or** the last element.
- Useful when we might want to remove elements from either end – perhaps a history function that can be read backward or forward.

Deque: The ADT

- A Deque stores a set of objects.
- Follows neither FIFO nor FILO.
- Standard Deque operations include:
  - **addFirst**: Inserts a new element at the head.
  - **addLast**: Inserts a new element at the tail
  - **removeFirst**: Removes and returns the element at the head.
  - **removeLast**: Removes and returns the element at the tail.
  - **getFirst**: Returns (but doesn't remove) the element at the head.
  - **getLast**: Returns (but doesn't remove) the element at the tail.
  - **Size**: How many things are in the queue?
  - **isEmpty**: Is the queue empty? Yes or no.

The Deque and Java

- Java has a **Deque interface**, the same as Queue, but it also has the **ArrayDeque class** that works fine if you just want to use that.
- If we want to make our own, **a doubly-linked list** makes the most sense here, since we want to pull from both the head and tail.
- We should remember that adding **sentinel nodes** (blank header and trailer nodes) will make implementing our functions easier as well.

A Good Moment to Pause and Reflect

- The material so far has provided most of the **foundations for what data structures and algorithms are** in programming and how we use them.
- So far we've covered:
  - The basics of **object oriented programming**.
  - **Abstract data types**, the concept(s) behind data structures and algorithms.
  - The **primitive structures** like **arrays** and **lists** that form the basis of other structures.
  - A handful of **algorithms for accessing or sorting data** structures and tools like **Big-Oh notation** to analyze and compare their run-times.
  - A set of related standard data structures (**stacks, queues, deques**).

The Course Moving Forward

- Much of what comes next **builds on these foundations**, taking the ideas introduced here into different and more complicated directions.
- We'll be seeing more **data structures** for sorting data in more deliberate ways (trees, heaps…), more **algorithms** for new functions (searches) or alternative solutions (sorts**), design patterns** for useful programming tools, and plenty more **runtime analysis**.
- The **next assignment** will try to encapsulate all of these fundamentals, so take the opportunity to check your understanding before we proceed further.

Recap – Last In, Last Out

- We **introduced the Queue**, the cousin of the Stack, governed by **FIFO** (first-in-first-out).
- We learned how **Queue is implemented in Java**, including its relation to the **LinkedList**.
- We implemented our own **list-based Queue**.
- We introduced the **Double-Sided Queue** (the **Deque**).
- Finally, we reviewed the course up to this point.