Queues

Authored by AllisonP

A queue is an abstract data type that uses a principle called *First-In-First-Out* (*FIFO*), meaning that the first object added to the queue must be the first object removed from it. You can analogize this to a checkout line at a store where the line only moves forward when the person at the head of it has been helped, and each person in the line is directly behind the person whose arrival immediately preceded theirs.

At minimum, any queue should be able to perform the following two operations:

- *Enqueue*: Add an object to the back of the line.
- *Dequeue*: Remove the object at the head of the line and return it; the element that was previously second in line is now at the head of the line.

The diagram below demonstrates these simple operations on an empty queue:

In addition, it's often helpful to implement a method to check whether or not a queue is empty to ensure you are not attempting to perform dequeue operations on an empty queue.

Sample Java Implementation

EXAMPLE

The code below demonstrates a simple Java Queue implementation.

```
import java.util.*;

class Queue<E> {

    private class Element<E> {
        // The data value of the element
        private E data;
        // The next element in the queue
        private Element<E> next;

        Element(E data) {
            this.data = data;
            this.next = null;
        }
    }

    // The first element in the queue
    private Element<E> front;
    // The last element in the queue
    private Element<E> back;

    /** Create an empty queue **/
    public Queue() {
        this.front = null;
        this.back = null;
    }

    /** @return true if the queue is empty, false if it is not.
    **/
    public boolean isEmpty() {
        return front == null || back == null;
    }

    /**
        Enqueues a value into the queue.
        @param value The data to be enqueued.
    **/
    public void enqueue(E value) {
        // Create new element
        Element<E> newElement = new Element<E>(value);

        // If the queue is empty
        if(isEmpty()) {
            this.front = newElement;
        }
        else { // Queue is not empty
            // Link the old last element to the new last element
            this.back.next = newElement;
        }

        // Set new back element regardless of whether or not queue is empty
        this.back = newElement;
    }

    /**
        Dequeues the queue's first element.
        @return the data associated with the queue's dequeued element.
        @throws NoSuchElementException if the queue contains no elements.
    **/
    public E dequeue() {
        if(isEmpty()) {
            throw new NoSuchElementException();
        }

        Element<E> head = front;
        this.front = front.next;

        return head.data;
    }

    /**
        'View' the element at the front of the queue.
        @return the data associated with the queue's first element.
        @throws NoSuchElementException if the queue contains no elements.
    **/
    public E peekFirst() {
        if(isEmpty()) {
            throw new NoSuchElementException();
        }

        return front.data;
    }

```

Table Of Contents

```java
    /**
        'View' the element at the tail of the queue.
        @return the data associated with the queue's first element.
        @throws NoSuchElementException if the queue contains no elements.
    **/
    public E peekLast() {
        if(isEmpty()) {
            throw new NoSuchElementException();
        }

        return back.data;
    }
}

class QueueDemo {
    public static void main(String[] args) {
        Queue<Integer> intQueue = new Queue<Integer>();

        try {
            intQueue.dequeue();
        }
        catch(NoSuchElementException e) {
            System.out.println("We cannot dequeue from an empty queue.");
        }

        for(int i = 0; i < 4; i++) {
            intQueue.enqueue(i);
            System.out.println(
                "First: " + intQueue.peekFirst() +
                "; Last: " + intQueue.peekLast()
            );
        }
        for(int i = 0; i < 4; i++) {
            System.out.println("Dequeued: " + intQueue.dequeue());
        }

        try {
            intQueue.peekFirst();
        }
        catch(NoSuchElementException e) {
            System.out.println("We cannot peek at an empty queue.");
        }
    }
}
```

Output

Run