Queues

Queue data structure methods

The queue data structure has three main methods:

- `enqueue` (adds a node to the back of the queue)
- `dequeue` (removes node at the front of the queue)
- `peek` (returns value of node at the front of the queue, without removing it)

Queue follows FIFO protocol

A queue is a data structure composed of nodes, which follows a first in, first out (FIFO) protocol.
This is analogous to a line at a grocery store, for which the first customer in the queue is the first to checkout.

Java Queue: Overloaded Constructor

The constructor in the Java `Queue` class can be overloaded in order to create an unbounded queue. The main constructor takes one argument, `maxSize`, which it assigns as the maximum size of the queue.
The overloaded constructor doesn't take any arguments but assigns the maximum size to be `Integer.MAX_VALUE`, which is the greatest integer value in Java. This is stored in a variable `DEFAULT_MAX_VALUE`. If no specified max size is provided as a parameter to a constructor, the overloaded constructor calls the main constructor using `DEFAULT_MAX_VALUE` as its parameter.

```
public Queue() {
    this(DEFAULT_MAX_SIZE);
}

public Queue(int maxSize) {
    this.queue = new LinkedList();
    this.size = 0;
    this.maxSize = maxSize;
}
```

Java Queue: Helper Methods

The Java `Queue` class should include two helper methods to determine what actions can be taken with the queue:

- `.hasSpace()` returns a `boolean` representing whether or not there is room left in a bounded queue

```
public boolean hasSpace() {
    return this.size < this.maxSize;
}

public boolean isEmpty() {
```

- .isEmpty() returns a boolean representing whether or not the queue is empty

These methods use the Queue instance variables, size and maxSize , to determine what value should be returned.

```
    return this.size == 0;
}
```

Java Queue: enqueue()

The .enqueue() method of the Java Queue class is used to add new data to the queue. It takes a single argument, data , which is added to the end of the queue using the LinkedList method .addToTail() . A print statement can be included to describe the addition. The method then increases size and throws an error if the queue is full. The helper method .hasSpace() is used to verify if the queue is full.

```
public void enqueue(String data) {
    if (this.hasSpace()) {
        this.queue.addToTail(data);
        this.size++;
        System.out.println("Added " + data
+ "! Queue size is now " + this.size);
    } else {
        throw new Error("Queue is full!");
    }
}
```

Java Queue: dequeue()

The .dequeue() method of the Java Queue class removes the head of the queue using the LinkedList method, .removeHead() , and then returns the head's data. A statement can be printed describing this removal. The method also decreases size and throws an error if the queue is empty. The helper method .isEmpty() verifies if the queue is empty.

```
public String dequeue() {
    if (!this.isEmpty()) {
        String data =
this.queue.removeHead();
        this.size--;
        System.out.println("Removed " +
data + "! Queue size is now " + this.size
+ ".");
        return data;
    } else {
        throw new Error("Queue is
empty!");
    }
}
```

Java Queue: peek()

The .peek() method of the Java Queue class allows us to see the element at the head of the queue without removing it. If a head exists (the queue is not empty), this method returns the data in the head. Otherwise, it returns null . This is verified using the helper method .isEmpty() .

```
public String peek() {
    if (this.isEmpty()) {
      return null;
    } else {
      return this.stack.head.data;
```

```
    }
}
```