**Lesson 14 – Queues II**

**Review Chapter 8, Sections 1-3 of the text.**

**Learning Objectives:**

* Describe the abstract-level functionality of an ADT Queue.
* List the Queue ADT operations.
* Define a queue interface in Java.
* Write a reference-based implementation of the ADT Queue.

**A Reference-based Implementation of ADT Queue:**

* Let’s re-implement the Node class to support ADT Queue (same Node class used in Stack except only one constructor)
* We could use two references: **front** and **last**
* **Let the students do peek, equeue and dequeue**

public class **QueueFirstLast**<E> implements Queueable<E> {

    Node<E> first;

    Node<E> last;

    public QueueFirstLast() {

**this**.first = *null*;

**this**.last = *null*;

    }

    @Override

    public boolean isEmpty() {

        return (**this**.first == *null*);

    }

    @Override

    public void dequeueAll() {

**this**.first = *null*;

**this**.last = *null*;

    }

    @Override

    public E peek() throws QueueException {

        if (**this**.isEmpty())

            throw new QueueException("Queue is empty.");

        else

            return **this**.first.getItem();

    }

    @Override

    public void enqueue(E item) {

        Node<E> node = new Node<E>(item);

        if (**this**.isEmpty())

**this**.first = node;

        else

**this**.last.setNext(node);

**this**.last = node;

    }

    @Override

    public E dequeue() throws QueueException {

        if (**this**.isEmpty())

            throw new QueueException("Queue is empty.");

        else {

            Node<E> next = **this**.first.getNext();

            E front = **this**.first.getItem();

**this**.first = next;

            return front;

        }

    }

    @Override

    public String toString() {

        String s = "[";

        Node<E> current = **this**.first;

        while (current != *null*) {

            s = s + current.getItem().toString();

            if (current.getNext() != *null*)

                s += ", ";

            current = current.getNext();

        }

        return s + "]";

    }

}

**Testing**

public class **TestQueueFirstLast** {

    public static void main(String[] args) {

        QueueFirstLast<String> line = new QueueFirstLast<String>();

        String[] names = new String[]{"Tony", "Bruce", "Natasha", "Scott", "Wanda", "Stephen"};

        for (String name : names) {

            line.enqueue(name);

        }

        System.out.println("After Enqueue: ");

        System.out.println(line);

        System.out.println();

        System.out.println("Test Dequeue: ");

        for (int i = 0; i < 3; i++)

            System.out.println(line.dequeue());

        System.out.println(line);

        System.out.println(        );

        System.out.println("Test Peek:");

        System.out.println(line.peek());

        System.out.println();

        System.out.println("Test Dequeue All");

        line.dequeueAll();

        System.out.println(line);

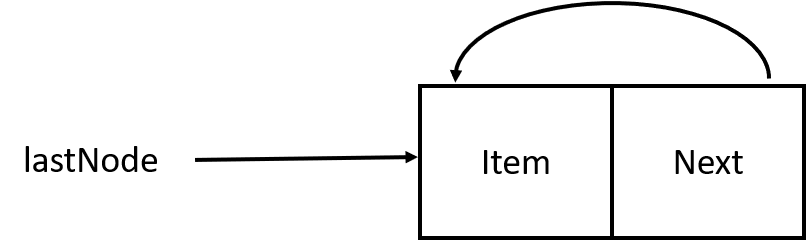
        System.out.println("Is Empty: " + line.isEmpty());

    }

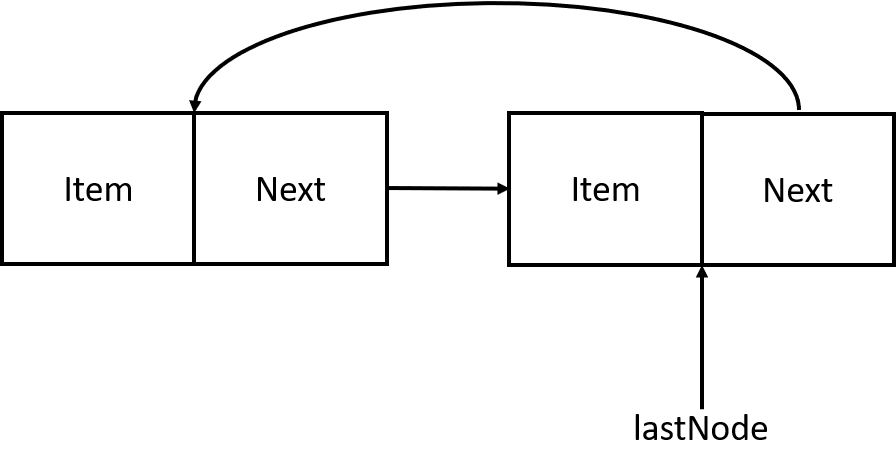
}

**Our reference-based implementation of ADT Queue:**

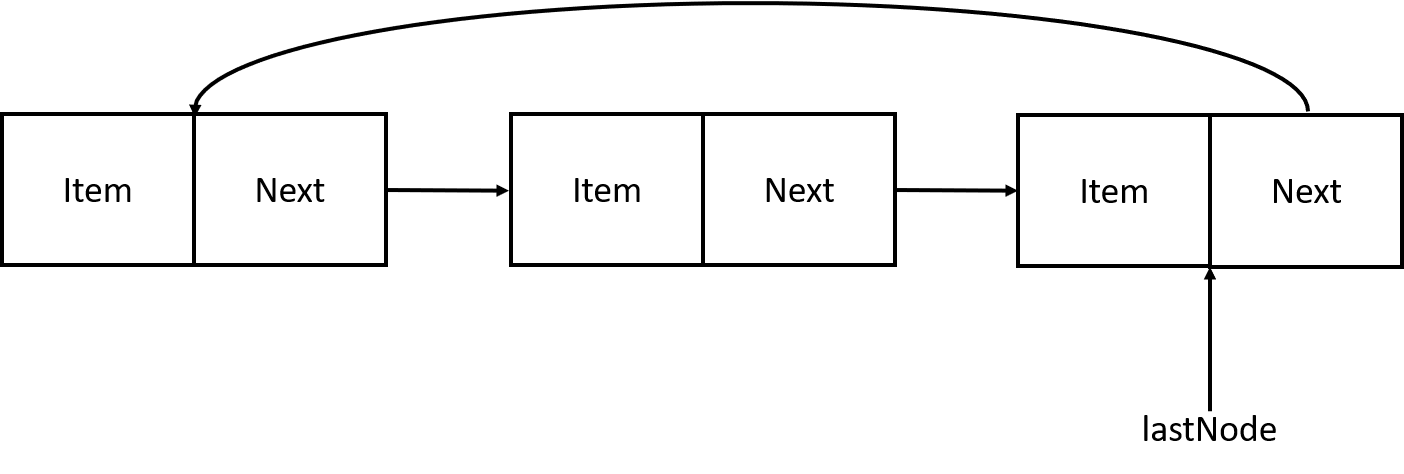
* We could use one reference to our queue at the front.
  + With this, the enqueue() method would need to traverse the list to the end each time to add the new Node.
  + We could simplify this by using two references: one to the front of the queue and one to the back of the queue.
  + Or, we could make a circular queue with one reference (pointing to the last element added) and the last node’s ‘next’ pointing to the first Node added.
* To implement this ‘circular queue’, we focus on the enqueue() method:



* + If we add a new node to an empty queue, the next field is self-referential…it points to ‘this’ node.



* + Adding a second node, the lastNode attribute points to the previous node (which is also the ‘oldest’ node in the queue).



* + As we add more nodes, notice that next attribute of the last node added always points to the ‘oldest’ node in the queue. This gives us easy access to this ‘oldest’ node for easy access in dequeue().
* Now, to implement:

public class **QueueCircular**<E> implements Queueable<E> {

    Node<E> last;

    public QueueCircular() {

**this**.last = *null*;

    }

    @Override

    public boolean isEmpty() {

        return (**this**.last == *null*);

    }

    public boolean isSingleton() {

        if (**this**.isEmpty())

            return *false*;

        else

            return (**this**.last == **this**.last.getNext());

    }

    @Override

    public void dequeueAll() {

**this**.last = *null*;

    }

    @Override

    public E peek() throws QueueException {

        if (**this**.isEmpty())

            throw new QueueException("Queue is empty.");

        else {

            Node<E> first = last.getNext();

            return first.getItem();

        }

    }

    @Override

    public void enqueue(E item) {

        Node<E> node = new Node<E>(item);

        Node<E> first;

        if (isEmpty()) {

            first = node;

**this**.last = node;

        } else {

            first = **this**.last.getNext();

        }

        node.setNext(first);

**this**.last.setNext(node);

**this**.last = node;

    }

    @Override

    public E dequeue() throws QueueException {

        if (**this**.isEmpty())

            throw new QueueException("Queue is empty.");

        Node<E> first = **this**.last.getNext();

        if (**this**.isSingleton())

**this**.last = *null*;

        else {

            Node<E> next = first.getNext();

**this**.last.setNext(next);

        }

        return first.getItem();

    }

    @Override

    public String toString() {

        if (**this**.isEmpty())

            return "[]";

        else {

            String s = "[";

            Node<E> current = **this**.last.getNext();

            while (current != **this**.last) {

                s = s + current.getItem().toString() + ", ";

                current = current.getNext();

            }

            s = s + **this**.last.getItem(); // add the last item

            return s + "]";

        }

    }

}

* We can write a simple main method to test our implementation:

TestCircularQueue.java – same as before, just change the top line

MessageQueue.java