**Lesson 13 – Queues I**

**Read 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.

**The Abstract Data Type Queue:**

* Describe the concept of a queue ADT:
  + In the U.S., we do not commonly use the word ‘queue’. In the U.K., the term “queue” is used to refer to a “line” at the bank, at the movie theater, etc.
  + To a computer scientist, a queue means something very particular. A queue has the property that the first item placed on the queue will be the first item removed. This property is commonly referred to as **first-in, first- out**, or simply **FIFO**.
* The ADT Queue Operations:
  + Create an empty queue.
  + Determine whether a queue is empty.
  + Add a new item to the back of the queue (**enqueue**).
  + Remove the front item from the queue (**dequeue**).
  + Remove all the items from the queue.
  + Retrieve the item at the front of the queue without removing it (**peek**).
* Pseudocode Specification for the Queue ADT:

// QueueItemType is the type of the items stored in the stack.

+createQueue()

// Creates an empty queue

+isEmpty() : Boolean

// Determines whether a queue is empty

+enqueue(in newItem : QueueItemType) throws QueueException

// Adds newItem to the end of the queue. Throws

// QueueException if the insertion is not successful.

+dequeue() : QueueItemType throws QueueException

// Retrieves and then removes the item at the front of

// the queue. Throws QueueException if the deletion

// is not successful.

+dequeueAll()

// Removes all items from the queue.

+peek() : QueueItemType throws QueueException

// Retrieves the item at the front of the queue.

// That is, peek retrieves the item that has been

// in the queue the longest. Retrieval does not

// change the queue. Throws QueueException

// if the retrieval is not successful.

**Define an interface for the ADT Queue:**

* First, define QueueException:

public class QueueException extends RuntimeException {

    public QueueException(String s) {

       super(s);

    }

 }

* Now, write the interface based on the specification above (as ICE):

public interface Queueable<E> {

    public boolean isEmpty();

    public void dequeueAll();

    public E peek() throws QueueException;

    public void enqueue(E item) throws QueueException;

    public E dequeue() throws QueueException;

}

**An array-based Implementation of ADT Queue:**

* Let the students do enqueue and dequeue

public class QueueArrayBased<E> implements Queueable<E> {

    public final int MAX\_QUEUE = 50;

    private Object[] queueArray;

    private int size;

    // Create an empty queue

    public QueueArrayBased() {

        this.queueArray = new Object[this.MAX\_QUEUE];

        this.size = 0;

    }

    // Determine if queue is empty

    @Override

    public boolean isEmpty() {

        return this.size == 0;

    }

    // Adds new item to the end of the queue.

    @Override

    public void enqueue(E newItem) throws QueueException {

        if (this.size == this.MAX\_QUEUE)

            throw new QueueException("Error: Queue is full!");

        this.queueArray[this.size] = newItem;

        this.size++;

    }

    // Retrieves and then removes the item at the front

    // of the queue.

    @Override

    public E dequeue() throws QueueException {

        if (this.isEmpty())

            throw new QueueException("Error: Empty queue!");

        E returnItem = (E) this.queueArray[0];

        for (int i = 1; i < this.size; i++)

            this.queueArray[i - 1] = this.queueArray[i];

        this.size--;

        return returnItem;

    }

    // Remove all items from the queue

    @Override

    public void dequeueAll() {

        this.size = 0;

    }

    // Retrieves the item at the front of the queue.

    @Override

    public E peek() throws QueueException {

        if (this.isEmpty())

            throw new QueueException("Error: Empty queue!");

        return (E) this.queueArray[0];

    }

    @Override

    public String toString() {

        String s = "[";

        for (int i =0; i < size; i++) {

            s = s + this.queueArray[i].toString();

            if (i != size-1)

                s = s + ", ";

        }

        return s + "]";

    }

}

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

public class TestQueue {

    public static void main(String[] args) {

        QueueArrayBased<Integer> queue = new QueueArrayBased<Integer>();

        for (int i = 0; i < 10; i++)

            queue.enqueue(i);

        System.out.println("Enqueue: " + queue);

        System.out.println();

        for (int i = 0; i < 10; i++)

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

        System.out.println("Dequeue: " + queue);

        System.out.println();

        try {

            queue.dequeue();

        } catch (QueueException e) {

            System.out.println("Exception caught exceptionally well!");

            System.out.println();

        }

        for (int i = 0; i < 10; i++)

            queue.enqueue(i);

        System.out.println("Re-enqueue: " + queue);

        System.out.println();

        queue.dequeueAll();

        System.out.println("Dequeue All: " + queue);

        System.out.println();

    }

}

If time, **MessageQueue.java**