CSE 214 – Homework I

Instructor: Dr. Ritwik Banerjee

This homework document consists of **??** pages. Carefully read the entire document before you start coding. You may realize that there are things in this homework that make you write code that is not worth too many points (or may not even be explicitly mentioned in this document), but is needed so that the things worth more points can run properly.

You are asked to implement three classes in a single project. You must use IntelliJ IDEA and Java SDK 14.0.2, as specified in the course syllabus. All three classes must reside in a package called `cse214hw1`. There are two interfaces provided to you here. These must also be included in the same package.

1. The first task is to write a few utility methods in the `ArrayUtils` class. These methods must be `static`, since they depend only on their input parameters and not any instance of the `ArrayUtils` class. (10)

   1. **Array rotation.** [5 points]

   ```
   /**
    * Rotates the array given array by r number of elements to the left, i.e., for each index i, a[i]
    * moves to a[(i+r) mod a.length].
    * @param  a: the input array of <code>int</code>s
    */
   public static void rotate(int[] a, int r);
   ```

   2. **Array rotation.** [2 points]

   ```
   /**
    * Rotates the array given array by r number of elements to the left, i.e., for each index i, a[i]
    * moves to a[(i+r) mod a.length].
    * @param  a: the input array of <code>char</code>s
    */
   public static void rotate(char[] a, int r);
   ```

   3. **Array merge.** [3 points]

   ```
   /**
    * Creates a merged array c such that c.length = a.length + b.length, and all the elements of b
    *  appear in c in the original order, but only after all elements of a (again, in the original
    * order). For example, merge([1,2,3], [4,5]) yields the array [1,2,3,4,5].
    * @param  a: the first of the two arrays to be merged
    * @param  b: the second of the two arrays to be merged
    * @return c: the merged array
    */
   public static void merge(int[] a, int[] b);
   ```

2. This second part requires you to completely implement an `ArrayDeque` class based on the `Deque` interface, where you must use a backing array. The interface is defined as follows: (30)

```
public interface Deque<T> {

    /**
     * Inserts the specified element at the front of this deque.
     * @param t the element to add
     */
    void addFirst(T t); // [5 points]

    /**
     * Inserts the specified element at the end of this deque.
     * @param t the element to add
     */
    void addLast(T t); // [5 points]
```

```
    /**
     * Retrieves and removes the first element of this deque, throwing an exception if this deque is empty.
     * @return the first element of this deque
     * @throws java.util.NoSuchElementException if this deque is empty
     */
    T removeFirst(); // [5 points]

    /**
     * Retrieves and removes the last element of this deque, throwing an exception if this deque is empty.
     * @return the first element of this deque
     * @throws java.util.NoSuchElementException if this deque is empty
     */
    T removeLast(); // [5 points]
}
```

Your `ArrayDeque` class may contain more methods, beyond the ones defined in the interface it implements, but such methods must be `private` or `protected`. Further, your implementation must use the circular array and modular arithmetic approach discussed in the lectures so that your code has better time complexity compared to the naïve approach.

Constructors. [6 points]

```
Deque<String> a = new ArrayDeque<>();    // creates an empty array deque of some default capacity
Deque<String> b = new ArrayDeque<>(100); // creates an empty array deque with capacity 100.
```

Your class must contain constructors such that the above lines of code are valid.

Static methods. [4 points]

```
/* Creates a deque where the first element is 2.0, and the last item is 5.25. */
ArrayDeque<Double> doubles = ArrayDeque.of(2.0, 4.0, 5.25);
```

Your class must have a method of the following signature, so that the above line compiles:

```
public static <T> ArrayDeque<T> of(T... args);
```

3. Just like the double-ended queue interface, here you are provided with the definition of the `Queue` abstract data type. (10)

```
public interface Queue<T> {
    /**
     * Inserts the specified element into this queue
     * @param t the element to add
     */
    void add(T t); // [3 points]

    /**
     * Retrieves and removes the head of this queue.
     * @return the first item of this queue
     * @throws java.util.NoSuchElementException if this queue is empty
     */
    T remove(); // [3 points]

    /**
     * Retrieves, but does not remove, the head of this queue.
     * @return the first item of this queue
     * @throws java.util.NoSuchElementException if this queue is empty
     */
    T peek(); // [3 points]
}
```

Your task is to implement an `ArrayQueue` class that implements the above interface using a backing array. This class must also use the circular array and modular arithmetic approach, and have a constructor so that the following line is valid (1 point):

```
Queue<Character> q = new ArrayQueue<>(); // creates an empty queue of some default capacity
```

NOTES:

- As always, **late submissions** or **uncompilable code** will not be graded.
- Please remember to verify what you are submitting. Make sure you are, indeed, submitting what you think you are submitting!
- What to submit? A single `.zip` file containing the three `.java` files. Do not submit the package as a whole folder, and definitely do not submit your entire project. Doing so may end up including settings that are specific to your computer, and the project may not run on a different machine! Also do not submit the interfaces. The grading process will use the original interfaces provided to you to ensure that the ADT definitions have not been modified. **This assignment may be graded by a script, so be absolutely sure that the submission follows this structure**.

Submission Deadline: Sep 22, 2020, 11:59 pm