Deque Interface API

```
public interface Deque<E> {
    /** Inserts the specified element at the front of this deque. */
    void addFirst(E e);

    /** Inserts the specified element at the end of this deque. */
    void addLast(E e);

    /** Retrieves, but does not remove, the first element of this deque. */
    E getFirst();

    /** Retrieves, but does not remove, the last element of this deque. */
    E getLast();

    /** Retrieves and removes the first element of this deque. */
    E removeFirst();

    /** Retrieves and removes the last element of this deque. */
    E removeLast();

    /** Returns the number of elements in this deque. */
    int size();

    /** Returns true if this deque contains no elements. */
    boolean isEmpty();
}

// Implementations of Deques covered in class
public class ArrayDeque<E> implements Deque<E> {...}
public class LinkedListDeque<E> implements Deque<E> {...}
```

List Interface API

```
public interface List<E> {
    /** Constructor that takes in a List and creates a new List copy with the same elements. */
    List<E> (List<E> e);

    /** Appends the specified element to the end of this list. Runs in constant time. */
    void add(E e);

    /** Returns the element at the specified position in this list. */
    E get(int index);


    /** Replaces the element at the specified position in this list with the specified element. */
    E set(int index, E element);
```

```
    /** Removes the element at the specified position in this list. */
    E remove(int index);

    /** Returns the number of elements in this list. */
    int size();

    /** Returns true if this list contains no elements. */
    boolean isEmpty();

    /** Returns a view of the portion of this list between the specified fromIndex, inclusive, and
    toIndex, exclusive. */
    List<E> subList(int fromIndex, int toIndex);

    /** Lists are defined to be equal if they contain the same elements in the same order. */
    boolean equals(Object o);

    /** Returns true if this list contains the specified element. */
    boolean contains(Object o);

    /** Returns an immutable list containing an arbitrary number of elements. For example,
    List<Integer> example = List.of(1, 2, 3); is a valid assignment. */
    static <E> List<E> of(E... elements);
}

// Implementations of Lists covered in class
public class ArrayList<E> implements List<E> {...}
public class LinkedList<E> implements List<E> {...}
```

Math Class API

```
public class Math {
    /** Returns the smaller of two int values. */
    public static int min(int a, int b) { ... }

    /** Returns the greater of two int values. */
    public static int max(int a, int b) { ... }

    /** Returns the value of the first argument raised to the power of the second argument. */
    public static double pow(double a, double b) { ... }

    /** Returns the correctly rounded positive square root of a double value. */
    public static double sqrt(double a) { ... }
}
```

Integer Class API

```
public class Integer {
    /** A constant holding the minimum value an int can have, -2^31. */
    public static final int MIN_VALUE = -2147483648;

    /** A constant holding the maximum value an int can have, 2^31-1. */
    public static final int MAX_VALUE = 2147483647;
}
```

String Class API

```
public class String {
    /** Returns the char value at the specified index. */
    public char charAt(int index) { ... }

    /** Returns the length of this string. */
    public int length() { ... }
}
```

Circular Doubly-Linked List Class

```
public class DLList<T> {
    private Node sentinel;
    private int size;

    public DLList() {
        sentinel = new Node(null);
        sentinel.prev = sentinel;
        sentinel.next = sentinel;
        size = 0;
    }

    private class Node {
        private T value;
        private Node next, prev;

        Node(T value) {
            this.value = value;
        }
    }
}
```