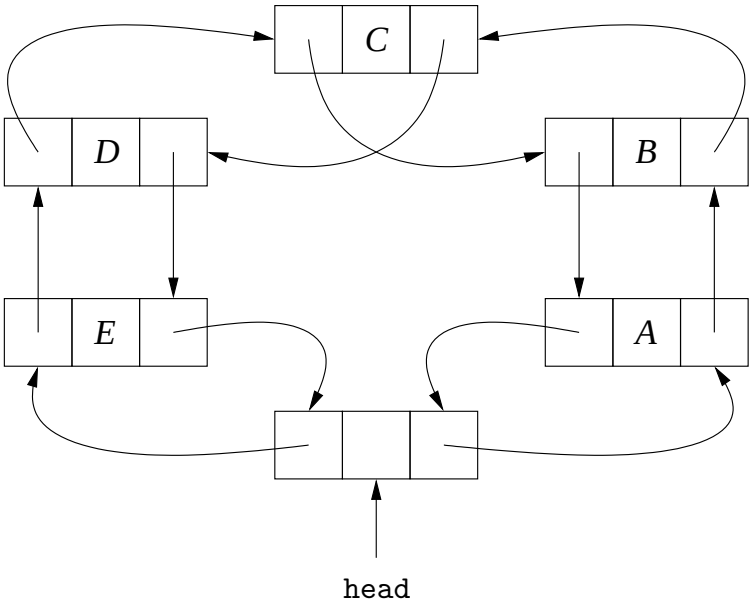
Computer Laboratory 11
CSci 1913: Introduction to Algorithms,
Data Structures, and Program Development
April 11/12, 2017

0. Introduction.

A *deque* (pronounced like *deck*) is a double-ended queue. Deques are like ordinary queues, except that objects can be inserted and deleted both at the front and the rear. Unlike the queues described in the lectures, they work without special cases. In this laboratory assignment, you will write a Java class called `Deque` that implements a deque.

1. Theory.

A deque can be implemented easily using a circular, doubly-linked list with a head node. For example, a deque containing the objects A, B, C, D, and E might use a list that looks like the following diagram. (To save space, the diagram does not show arrows pointing to those objects.)



The variable `head` points to the head node. The expression `head.right` points to the node at the front of the deque, which in turn points to the object A. The expression `head.left` points to the node at the rear of the deque, which in turn points to the object E.

2. Implementation.

You must write a class called `Deque` that implements a deque, using a doubly-linked circular list with a head node. Instances of `Deque` must be able to hold objects whose type is given by a class parameter `Base`. As a result, your class `Deque` must look like this, with your code in place of the three dots. To simplify grading, your code must use the same names for things that are used here.

```
class Deque<Base>
{
   ⋮
}
```

Your class `Deque` must have a `private` nested class called `Node`. The class `Node` implements the nodes of the doubly-linked list. It must have a `private` pointer slot called `object` that points to the object at the `Node`. It must have `private` pointer slots called `left` and `right` that point to the `Node`s at the left and right of the `Node`, respectively. It must have a constructor that initializes the `object`, `left`, and `right` slots of the `Node`.

Your class `Deque` must also have a `private` variable `head` that points to the head node of the circular doubly-linked list. Also, your class must have the following methods.

- `public Deque()`

  Constructor. Make a new, empty `Deque`. The variable `head` must be set to a new head node here.

- `public void enqueueFront(Base object)`

  Add a new `Node` at the front of the `Deque`. The `object` slot of the new `Node` must point to the parameter `object`.

- `public void enqueueRear(Base object)`

  Add a new `Node` at the rear of the `Deque`. The `object` slot of the new `Node` must point to the parameter `object`.

- `public Base dequeueFront()`

  If the `Deque` is empty, then throw an `IllegalStateException`. Otherwise, get the `Base` object from the `Node` at the front of the `Deque`. Delete the `Node` at the front of the `Deque`, and return the `Base` object.

- `public Base dequeueRear()`

  If the `Deque` is empty, then throw an `IllegalStateException`. Otherwise, get the `Base` object from the the `Node` at the rear of the `Deque`. Delete the `Node` at the rear of the `Deque`, and return the `Base` object.

- `public boolean isEmpty()`

  Return `true` if the `Deque` is empty. Return `false` otherwise.

The file `tests.java` on Moodle contains Java code that performs a series of tests. The tests call methods from the `Deque` class; some of them print what those methods return. Each test is also followed by a comment that tells how many points it is worth, and optionally what must be printed if it works correctly.

3. Deliverables.

Run the tests, then turn in the Java source code for your `Deque` class. Your TA will tell you how and where to turn it in. If your lab is on **Tuesday, April 11, 2017,** then your work must be turned in by **11:55 pm** on **Tuesday, April 18, 2017.** If your lab is on **Wednesday, April 12, 2017,** then your work must be turned in by **11:55 pm** on **Wednesday, April 19, 2017.** To avoid late penalties, do not confuse these two dates.