Project 2

Name: Khai Dong

Part 1: Repair Project 1

The original problem is with mutual exclusion where the `DLList` breaks down upon doing concurrency. To solve this problem, we wrap each of the method in a lock to convert the original data structure into a monitor.

```
<ret-type> <method-name>(self, <arugments>){
    lock.acquire();
    ...
    lock.release();
}
```

To allow `DLList.removeHead()` to wait until the `DLList` is not empty, I added a condition variable `empty`.

```
public Object removeHead() {
    lock.acquire();
    while(this.first == null) {
        this.empty.sleep();
    }
    ...
}
```

Then, `DLList.insert()` will signal when an object is added to the list. This is the only method in `DLList` that insert an object into the list (`DLList.prepend()` calls `DLList.insert()`).

```
public void insert(Object item, Integer sortKey) {
    if(!lock.isHeldByCurrentThread())
        this.lock.acquire();

    ++ this.size; // increment size counter
    DLLElement newElem = new DLLElement(item, sortKey);
    if(this.first == null){
        this.first = this.last = newElem;
        this.empty.wake();
        this.lock.release();
        return;
    }

    ...

    this.empty.wake();
```

```
        this.lock.release();
    }
```

`DLList.insert()` check if the current thread already acquire the lock before try to acquire the lock since it may be called by `DLList.prepend()` . Return to the 2 tests from project 1, there are no longer a `NullExceptionPointer` nor an invalid input.

```
([1,1] [3,3] [4,4] [6,6] (previously, ([1,1] [4, 4] [3,3] [6,6])
([1,1]) (previously throw a NullPointerException)
```

Part 2: Bounder Buffer

I implemented `BoundedBuffer` with several `KThread.yieldIfOughtTo()` within the `read()` and `write()` code to see if constant context switching would cause problems. Inside of `read` and `write` , there are also assertion to check for underflow and overflow. I made 3 tests to test `Bounded Buffer` .

- `BoundedBuffer_selfTestUnderflow` : First fork a `write` thread, then having a thread try to `read()` when the buffer is empty. The `read` thread will block itself, allow the `write` thread to run, then the `read` thread will print out the character. Expect no `AssertionError` and the correct character get printed out (also get asserted).
- `BoundedBuffer_selfTestOverflow` : First set up a buffer with a capacity of 1, then fill it with a character. Then, fork a `read` thread and run a `write` thread. Assert if `read` gives the right character. Expect no `AssertionError` (no overflow, underflow, nor invalid output). The read character also get printed out.
- `BoundedBuffer_selfTestConcurrency` : First set `Kthread.oughtToYield` to a sufficient long list of `True` . Then, fork several `read` and `write` threads then run them. For this test, since there is no exact order, I just expect all read values and the values remaining in the buffer added up to all the added characters. I used a `synchronizedList` to accumulate all the characters, then sequentially `read()` everything out of the buffer at the end. Also check for underflow and overflow in this test since assertions are built into `BoundedBuffer` .

The tests run correctly. Since there is no interleaving that could cause problem (besides starvation), there is no interleaving diagram. The assertions should already account for potential errors that may arise.

Part 3: `Condition2` Implementation

Changing from `Condition` to `Condition2` gives the same output.

```
KThread.DLL_selfTest() ...
([-11,B1] [-10,B3] [-9,B5] [-8,B7] [-7,B9] [-6,B11] [-5,A2] [-4,A4] [-3,A6] [-2,A8] [-1,A10] [0,A12])
([-11,B1] [-10,A2] [-9,B3] [-8,A4] [-7,B5] [-6,A6] [-5,B7] [-4,A8] [-3,B9] [-2,A10] [-1,B11] [0,A12])
([-11,B1] [-10,A2] [-9,B3] [-8,B5] [-7,A4] [-6,A6] [-5,B7] [-4,A8] [-3,B9] [-2,B11] [-1,A10] [0,A12])
([1,1] [3,3] [4,4] [6,6])
([1,1])
remove (1, 1) after waiting
1
()
```

```
()
KThread.BoundedBufferTest() ...
a
[]
a
[b]
Other tests ...
```