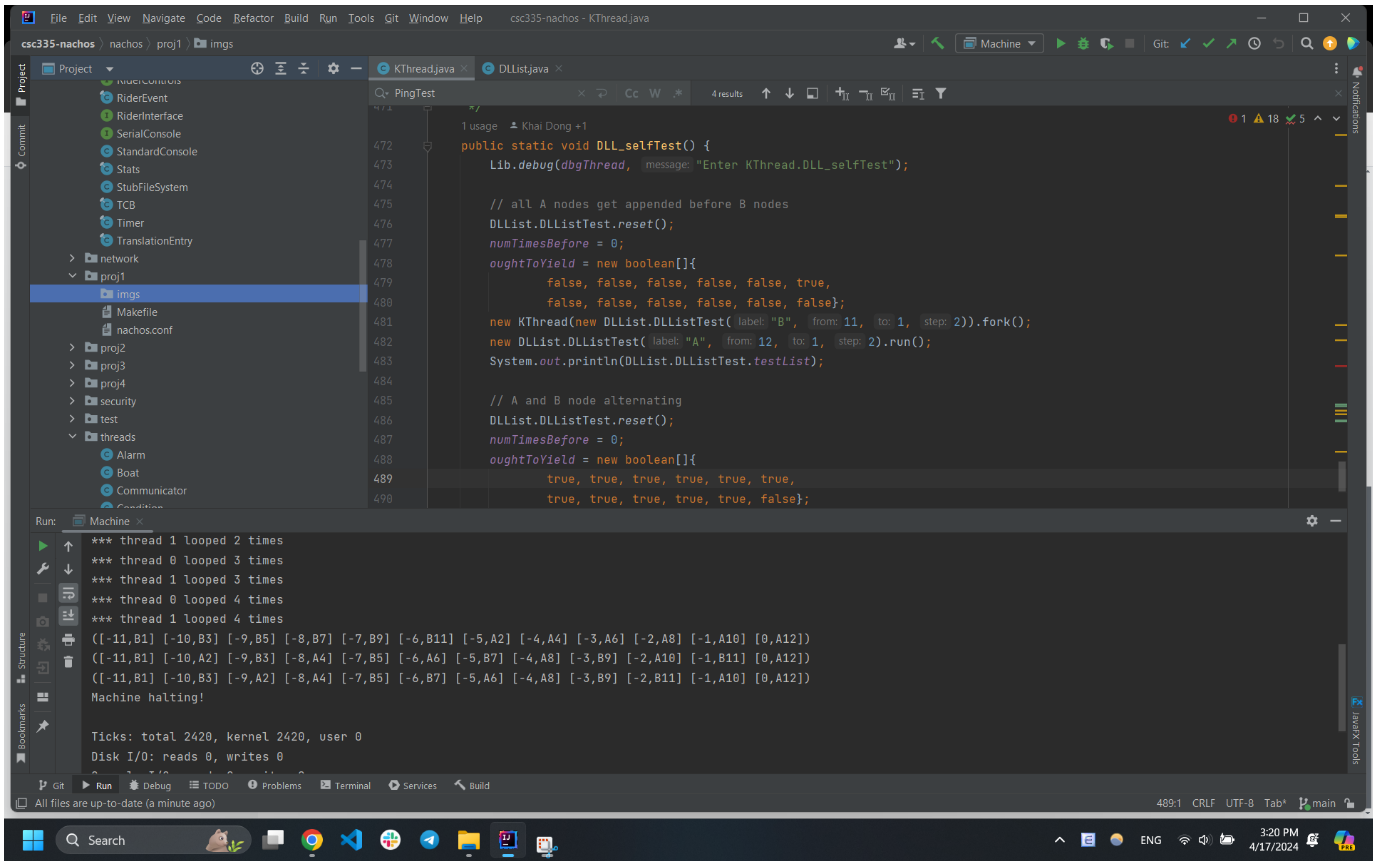
# Project 1

Name: Khai Dong

## Part 1

The source code for the interleaving is in `KThread.DLL_selfTest`.



```java
public static void DLL_selfTest() {
    Lib.debug(dbgThread, "Enter KThread.DLL_selfTest");

    // all A nodes get appended before B nodes
    DLList.DLListTest.reset();
    numTimesBefore = 0;
    oughtToYield = new boolean[]{
            false, false, false, false, false, true,
            false, false, false, false, false, false};
    new KThread(new DLList.DLListTest("B", 11, 1, 2)).fork();
    new DLList.DLListTest("A", 12, 1, 2).run();
    System.out.println(DLList.DLListTest.testList);

    // A and B node alternating
    DLList.DLListTest.reset();
    numTimesBefore = 0;
    oughtToYield = new boolean[]{
            true, true, true, true, true, true,
            true, true, true, true, true, false};
```

```
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
([-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,B3] [-9,A2] [-8,A4] [-7,B5] [-6,B7] [-5,A6] [-4,A8] [-3,B9] [-2,B11] [-1,A10] [0,A12])
Machine halting!

Ticks: total 2420, kernel 2420, user 0
Disk I/O: reads 0, writes 0
```

All files are up-to-date (a minute ago)

## Part 2.1 (Invalid Data)

This interleaving is in `KThread.DLL_selfTest2`. Initially, `testList` contains 2 nodes `(1, 1)` and `(6, 6)`.

| Thread 1: testList.insert(3, 3) | Thread 2: testList.insert(4, 4) |
|---|---|
| `++ this.size; // increment size counter`<br>`DLLElement newElem =`<br>        `new DLLElement(item, sortKey);`<br>`if(this.first == null){`<br>    `this.first = this.last = newElem;`<br>    `return;`<br>`}`<br>`DLLElement curElem = this.first;` | |

```
    while(curElem != null                 |
            && curElem.key <= sortKey)    |
        curElem = curElem.next;           |
                                          |    ++ this.size; // increment size counter
                                          |    DLLElement newElem =
                                          |            new DLLElement(item, sortKey);
                                          |    if(this.first == null){
                                          |        this.first = this.last = newElem;
                                          |        return;
                                          |    }
                                          |    DLLElement curElem = this.first;
                                          |    while(curElem != null
                                          |            && curElem.key <= sortKey)
                                          |        curElem = curElem.next;
                                          |
                                          |    if(curElem == null){
                                          |        this.last.next = newElem;
                                          |        newElem.prev = this.last;
                                          |        this.last = newElem;
                                          |    } else {
                                          |        newElem.prev = curElem.prev;
                                          |        newElem.next = curElem;
                                          |        curElem.prev = newElem;
                                          |        if (newElem.prev == null)
                                          |            this.first = newElem;
                                          |        else newElem.prev.next = newElem;
                                          |    }
    if(curElem == null){                  |
        this.last.next = newElem;         |
        newElem.prev = this.last;         |
        this.last = newElem;              |
    } else {                              |
        newElem.prev = curElem.prev;      |
        newElem.next = curElem;           |
        curElem.prev = newElem;           |
        if (newElem.prev == null)         |
            this.first = newElem;         |
        else newElem.prev.next = newElem; |
    }                                     |
```

In Thread 1, after the while loop, `curElem` will be the node `(6, 6)`. Then, a context switch happens, Threads 2 starts and finishes. Here, since Thread 1 has made no change to `testList`, so after Thread 2 (inserting `(4, 4)` into the list), the list will become

```
(1, 1), (4, 4), (6, 6)
```

Context switching back to Thread 1, since `curElem` is still the node `(6, 6)`, `(3, 3)` is inserted just before `(6, 6)`, yielding the final list:

```
(1, 1), (4, 4), (3, 3), (6, 6)
```

This is invalid data since the list supposed to be sorted by sortKey.

## Part 2.1 (Null Pointer Exception)

This interleaving is in `KThread.DLL_selfTest3`. Initially, `testList` contains a single node `(1, 1)`.

| **Thread 1: testList.prepend(0)** | **Thread 2: testList.removeHead()** |
|---|---|
| `int key = 0;` | |
| `if (this.first != null){` | |
| | `if(this.first == null) {` |
| | `    return null;` |
| | `}` |
| | `Object returnData = this.first.data;` |
| | `-- this.size; // decrement the size` |
| | `this.first = this.first.next; // update new head` |
| | `if(this.first == null) { // if pop the last element` |
| | `    this.last = null;` |
| | `} else {` |
| | `    this.first.prev = null;` |
| | `}` |
| | `return returnData;` |
| `    key = this.first.key - 1;` | |
| `}` | |
| `this.insert(item, key);` | |

Since `testList` initially is not empty, Thread 1 will do the if branch. We do a context switch just after Thread 1 done evaluating `this.first != null`. Then, we do the entirety of `testList.removeHead()`. Since the Thread 1 has not made any change to `testList`, `testList` will end up empty with `testList.first` become `null`. Then, when Thread 1 tries to access `this.first.key` (equivalent to `null.key`), it will result in a `NullPointerException`.