

## Corrections

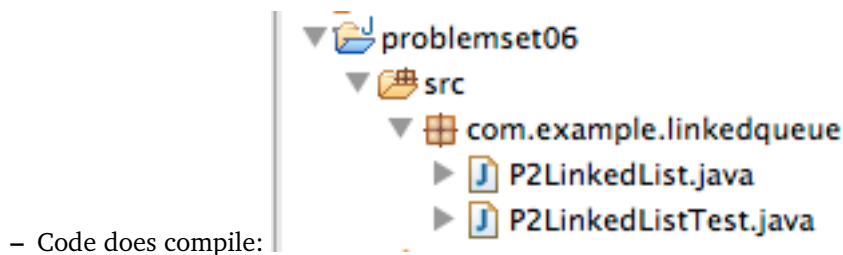
by AK

You should really take another look at Markdown. It's really nifty, if you use it carefully (e.g. don't do crazy stuff in the alternative text of pictures). but you used the Debugger successfully and described coding conventions. Speaking of which: `counter++` instead of `++counter` is not a good convention, as it seems rather random, maybe you could call it "don't try to squeeze as much code as possible on one line"

You pass problemset06 #Our hypotheses on why tests fail.

### Hypothesis 1: The code won't compile.

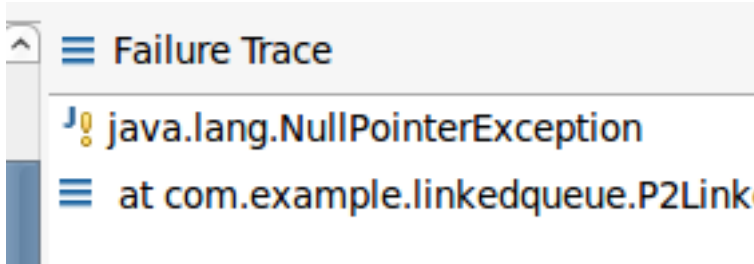
- Verification:



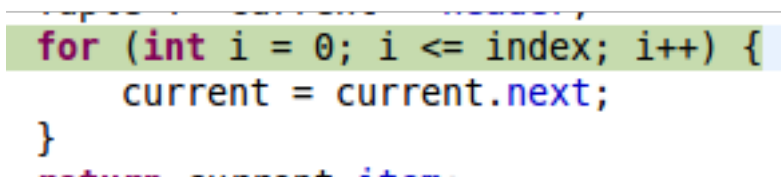
- Explanation:
  - The code is valid Java.

### Hypothesis 2: The 'get' method does not work properly.

- Verification: The test 'shouldFindNumberAtFirstIndexAfterAdd' fails:



- Explanation: The 'for'-loop lets the current tuple point to the next element, although the right element would already be in 'current'.



Could fix this easily by modifying the for loop:

```
for (int i = 0; i < index; i++) {
```

### Hypothesis 3 & 4: Tests fail because the 'get' method does not work properly.

```
@Test
public void shouldFind() {
    list.add(42);
    list.add(13);
    assertTrue(list.contains(42));
    assertTrue(list.contains(13));
}
```

- Verification: In both cases the problem occurs when calling the 'get' method:

```
@Test
public void shouldAddCollectionRight() {
    LinkedList<Integer> integerList = new LinkedList<Integer>();
    integerList.add(3);
    integerList.add(4);
    integerList.add(5);
    this.list.addAll(integerList);
    assertTrue(list.size() == 3);
    assertTrue(list.get(0) == 3);
    assertTrue(list.get(1) == 4);
    assertTrue(list.get(2) == 5);
}
```

- Explanation: Both methods seem to work properly, but there is a problem in the 'get' method (as seen before in hypothesis 2).

### Hypothesis 5: The addAt method does not work properly.

(Hint: I corrected the problem in the 'get' method first, because else there's always a problem with this 'get' method) \* Verification: \* the item 2 is put to the wrong position (pos. 3 instead of 2) in the linked list.

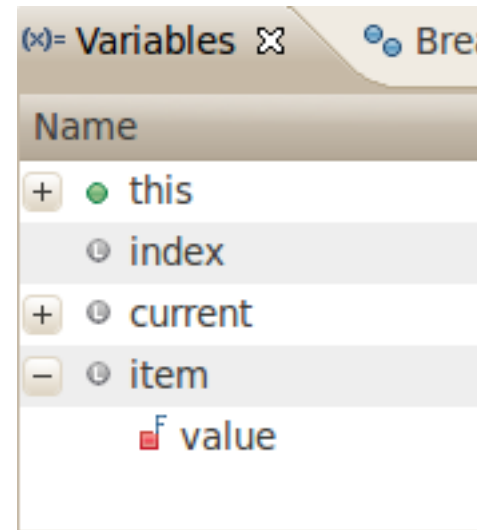
Name	Value
current	P2LinkedList
item	Integer (id: ...)
value	14
next	P2LinkedList
item	Integer (id: ...)
value	2
next	P2LinkedList
newItem	P2LinkedList
item	Integer (id: ...)
value	2
next	P2LinkedList

- Explanation: We can change the iteration in a way that the 'current' item is one item before the place where we want to insert a new item [for (int i = 0; i < index-1; i++)] and rewrite the bottom part of the method in the following way:

```
newItem.next = current.next;
current.next = newItem;
```

### Hypothesis 6: The 'remove' method does not work as it should.

- Verification:



- We see that the method returns the wrong item (with value 3 instead of 2):

\*Explanation: The for-loop goes one element too far. We can easily fix that by adding a -1 to the loop:

```
for(int i = 0; i < index-1; i++)
```

#Our hypotheses on why tests fail.

### Hypothesis 7: There must be an error in the 'addAll' method

- Verification:

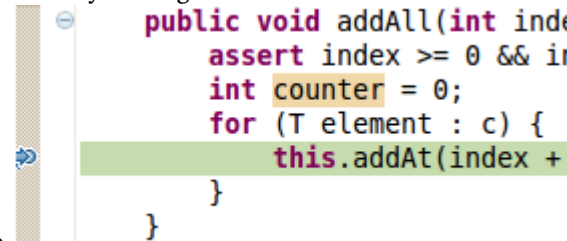
- The 'shouldAddCollectionAtIndex()' test method indicates an failure when calling the 'addAll' me-

```
@Test
public void shouldAddCollectionAtIndex() {
    LinkedList<Integer> integerList = new LinkedList<Integer>();
    integerList.add(3);
    integerList.add(4);
    integerList.add(5);
    this.list.add(1);
    this.list.add(2);
    this.list.add(6);
    this.list.addAll(2, integerList);
    assertTrue(this.list.size() == 6);
    assertTrue(this.list.get(0) == 1);
    assertTrue(this.list.get(1) == 2);
}
```

thod:

- Explanation:

- The counter does not increase in this loop. We can easily correct for that by writing 'counter++' instead



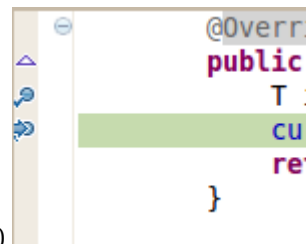
```
public void addAll(int index, Collection c) {
    assert index >= 0 && index < c.size();
    int counter = 0;
    for (T element : c) {
        this.addAt(index + counter, element);
    }
}
```

of '++counter' (We cannot be sure if that is the only error though)

## Hypothesis 8 & 9: Probably there is a problem in the 'next' method

- Verification:

- The returned item is the current item instead of the next item (see T item = current.item)



```
@Override
public T next() {
    for (T item = current.item; item != null; item = current.next.item) {
        return current.item;
    }
}
```

- Explanation:

- I am not totally sure if that is the right explanation, but I think that we should rewrite the code as follows: T item = current.next.item