To test addFirst, do we have to build the list twice to write assertion?

assertEquals(expected, computed)

assertEquals(?? , L1.addFirst(5))

?? LExp

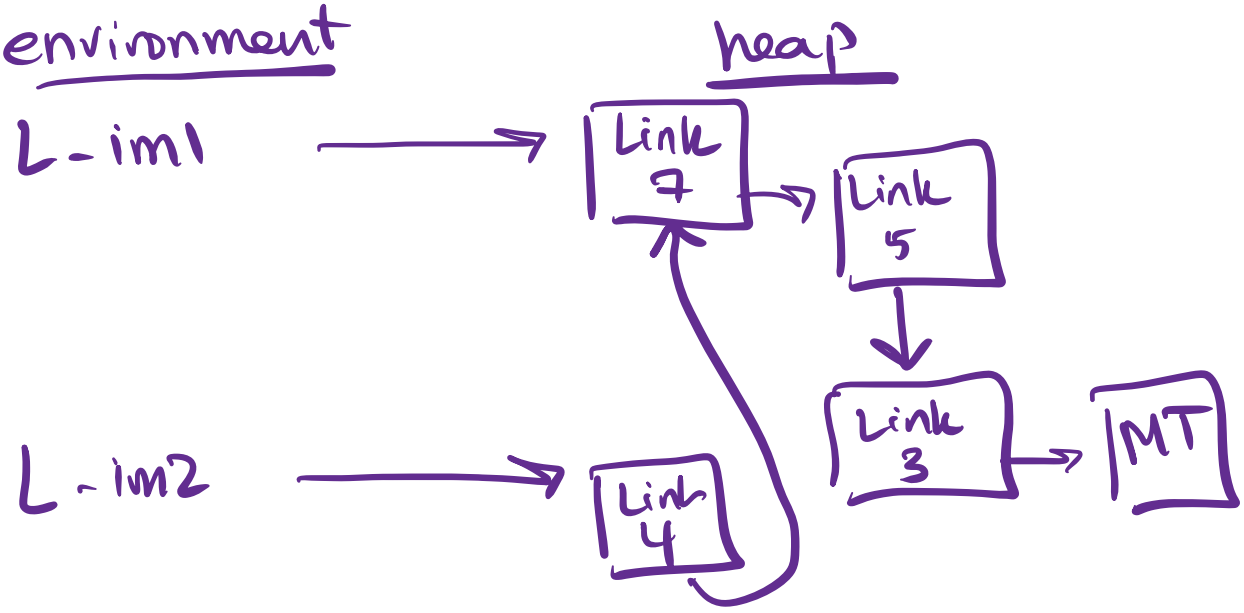
LExp = new EmptyList().addFirst(5)

use your other methods to check new methods

- size after addFirst
- contains after addFirst

Let's make mutable lists instead

Review immutable diagram



This diagram refers to the code

```
L-im1 = new EmptyList().addFirst(3).
                        addFirst(5).
                        addFirst(7);

L-im2 = L-im1.addFirst(4)
```

if we were to print out the contents of these lists, L-im1 would show [7,5,3] but not 4. L-im2 would show [4,7,5,3]

What should happen in a mutable list?

- if L_iml were mutable, then after calling addFirst, L_iml would also print as [4,7,5,3]

The question is how to make that happen. Let's start by looking at the level of the diagrams.

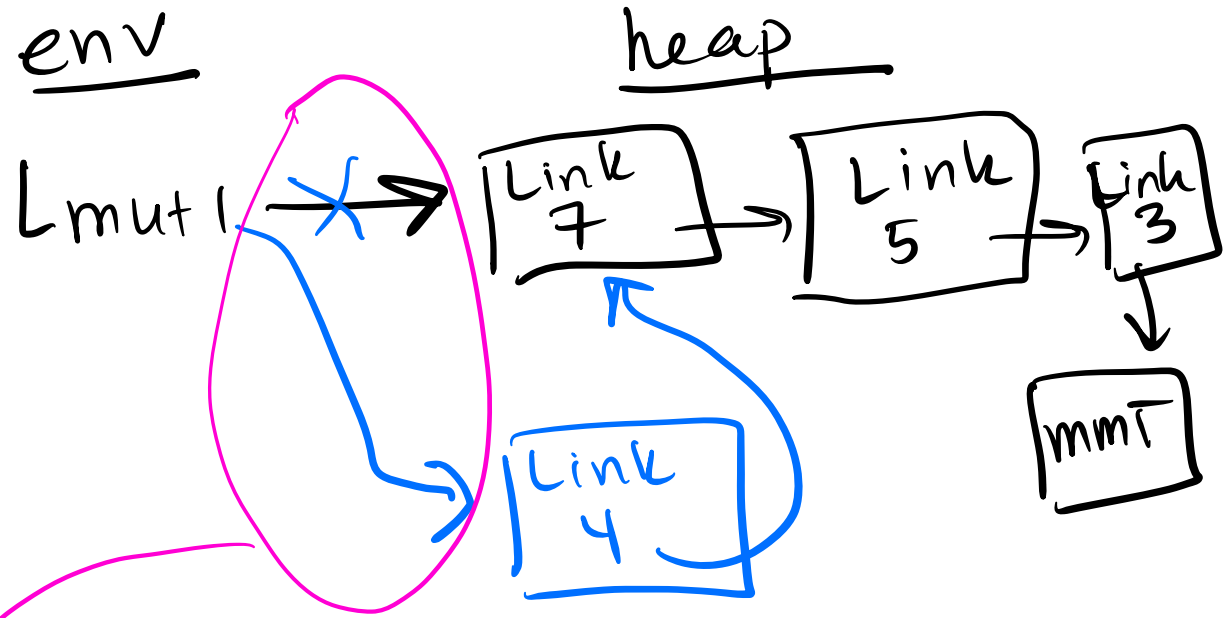
Here's a code sample and the diagram we might want to match the code:

```
Lmut1 = new MutEmptyList().
        addFirst(3).
        addFirst(5).
        addFirst(7)
```

printing Lmut1 shows [7,5,3]

```
Lmut1.addFirst(4)
```

printing Lmut1 shows [4,7,5,3]



The Java code we wrote in blue above does not produce this picture. The only code that can produce this picture is:

```
Lmut1 = Lmut1.addFirst(4)
```

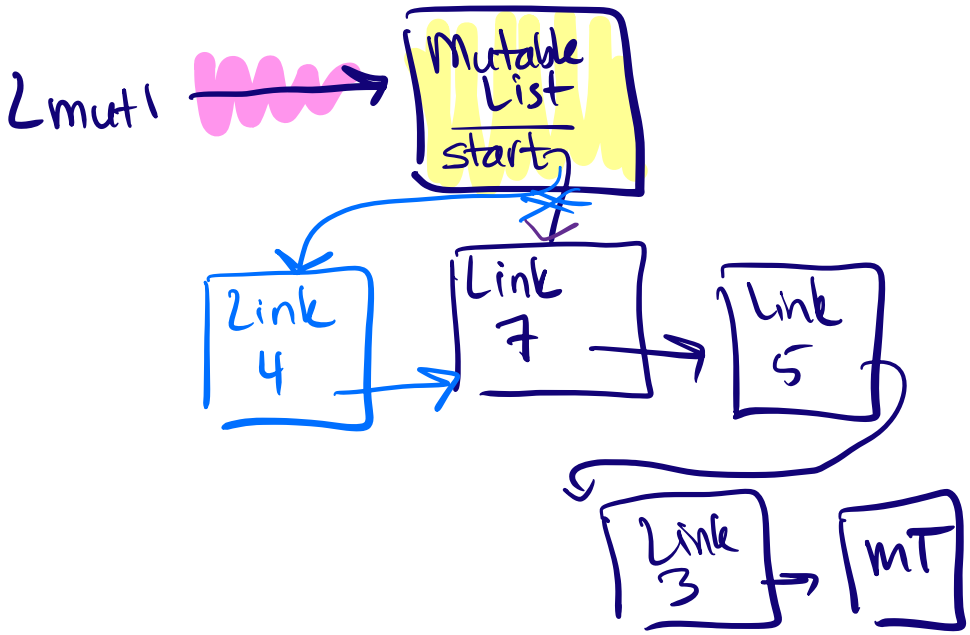
only way to change relationship between Lmut1 & objects in Java (or other langs)

Let's summarize:

- We want to write the blue code (easier for programmers than the green code)
- By the rules of Java, we have to write the green code given our existing Link class
- No Java code can build the picture that we drew (due to how the language works)

Instead, we need a picture in which no arrows from the names/env have to change, but the list contents in the heap can still change.

Here's a picture that does this:



Lmut1.addFirst(4)

We introduce a new class called MutableList that serves as a consistent access point to the rest of the list. addFirst changes what objects the MutableList refers to (that's the mutable part), but the mapping from Lmut1 to the MutableList object stays fixed.