COSC 102 — Spring 2015

Linked lists — Lab 9

Due date: **On Moodle before the start of next week lab**

The objectives of this lab are

1. to complete the implementation of a linked list class,
2. to implement a linked list that maintains a tail reference,
3. to implement a circular linked list and
4. to gain experience with JUnit, which is a unit testing framework for Java. JUnit is a simple framework to write repeatable tests. Read through the JUnit cookbook to get an intro, it's only one page!

Setup

Download the `zip` file from the course website.

Instead of DrJava you are welcomed to use Eclipse, which is a powerful IDE, Integrated Development Environment. Eclipse is installed on the department machines. See instructions below.

DrJava includes testing with JUnit. Do the following two steps to configure your DrJava environment.

- Create a folder `llist` in your `102/lab` where you paste both files
  1. `SingleLinkedList.java` and
  2. `SingleLinkedListTest.java`
- Open them in DrJava (close all other `.java` files)
  1. Try to compile `SingleLinkedList.java`: it doesn't work. Add default return values for both methods (i.e. complete them as stubs) or comment them out. It should run but nothing happens.
  2. Select `SingleLinkedListTest.java` and push the `"Test"` button. Six tests pass (green) and one fail (red). We will come back to them below.

In Eclipse do the following steps. For more details about Eclipse this tutorial is a great resource.

- Create a folder called `workspace` in your `102` directory.
- Find the `eclipse` program using the file system browser and double-click.
- When eclipse has started and is querying for a `workspace` folder browse to the one you just created.
- From the `File` menu,
  1. select `New` and add a `Java Project` called `llist`
  2. select `New` and add a `Class` called `SingleLinkedList.java` in the `llist` project
     paste into it the content of `SingleLinkedList.java` using `Notepad++` or `Textwrangler` to access its content
  3. select `New` and add a new `JUnit Test Case`
     - accept the default name `SingleLinkedListTest` and click `"Finish"`.
     - accept the `Add JUnit 4 library to build path` action.

    – replace the auto-generated method `test()` by the methods found in `SingleLinkedListTest.java` you downloaded.

- Try to compile `SingleLinkedList.java` by pushing the `"Run"` button, which is the basic green arrow. The code does not compile, red crosses have appeared.
- Add default return values for the `remove` methods or comment them out to get rid of the errors.
- `SingleLinkedList.java` now compiles but nothing happens when you try to run it: `Exception in thread "main" java.lang.NoSuchMethodError:  main .`
- Select `SingleLinkedListTest.java` and push `"Run"` again. A `JUnit` pane opens with test results

```
Runs 6/6                Errors 0                Failures 1
```

Congratulations you're done setting up your environment!

I. SingleLinkedList

Main

In `SingleLinkedList.java` add a `main` method in which

- a `SingleLinkedList` object called `classlist` is instantiated and
- the `add` method is called multiple times so that `classlist` is made of the following nodes

```
[Sam ==> Will ==> Troy ==> Allie]
```

Use `toString()` to check your work.

Testing

For this part you are working with `SingleLinkedListTest.java`, which has many commented and uncommented methods, each testing the functioning of one public method of `SingleLinkedList`.

- Comment out `test10()` and `test11()` (starting above the line `@Test`) as they are testing `remove()`, which isn't implemented yet. All the uncommented tests should pass now. Right?
- Read each of these testing methods which succeeded, i.e., `test1()`, `test2()`, `test4()`, `test5()` and `test8()`. Do they make sense? Go back and forth between the JUnit Test file and the class being tested, reading through the specific linked list methods that are being invoked. You want to understand their relations so that you could write those tests next time? Ask us if you have any concern.
- One-by-one for the commented methods, i.e. `test3()`, `test6()`, `test7()` and `test9()` do the following.
    - Read the test method and think how it works.
    - Uncomment the method run again `SingleLinkedListTest`. The console or JUnit pane displays

```
Errors/Failures 1 (test #) in red or blue
```

- - Read the test failure, especially the comparison given. In Eclipse double click on the line in the `"Failure Trace"` pane, which is at the bottom left.
  - Your task is to **fix the error**, which is in the **testing method of `SingleLinkedListTest.java`** (not in the linked list code).
- Read the tests for `remove` and do not uncomment them out for now. The tests are
  - `test10()` and `test11()` which are implemented and
  - `test12()` to `test20()` which have only a description of the case each will be testing. Since they are blank, think about their implementation. Feel free to write some of their body as comments for now.

Remove

`SingleLinkedList.java` provides two public methods to remove an element, which are not implemented yet. You are to write their bodies by doing the following.

- Read through `SingleLinkedList.java`. Notice the code contains two helper methods, `removeFirst` and `removeAfter`, as it is the case for the `add` operation. Think about **using them** to write concise code.
- Write and test each of the body for
  1. the `remove` method that removes a node at a specified index
     run the uncommented `test11()` and `test12()` to `test15()` which you need to develop
  2. the `remove(item)` method that removes the first occurrence of an element item
     run the uncommented `test10()` and `test16()` to `test20()` which you need to develop

II. SingleLinkedListTail

The goal of this part is to write a different implementation of the single linked list, where a `tail` reference is also maintained. This reference is used to efficiently add at the end of the list. **Do not traverse the list when it is unnecessary.**

Create a new class `SingleLinkedListTail` and the corresponding test class file, calling it `SingleLinkedListTailTestYOURNAME.java`. YOURNAME is critical for the test exchange (see below).

`SingleLinkedListTail` has to have the same public methods than `SingleLinkedList`, which are similar to the `List interface` definitions. The tests should only invoke those public methods; the helper methods, which are declared private, are tested indirectly because public methods use them.

The best practice is to use the following guidelines.

- As shown in class first *draw diagrams* for the general case and special cases. Refine them as you develop the code and tests. **You are required to show us those diagrams next week**.
- Develop a group of methods together. Start by the core ones, `add` and `toString`. Test as soon as possible, you don't need to have all the `add` functionality working before starting testing.
- Write helper methods that simplify the code of the public methods. Write each public method that uses them separately. Test each public method thoroughly before to start coding a different public method. Make sure that the private methods are tested indirectly.

While the tests in `SingleLinkedListTest` are inspiring and should be reproduced and adapted they are not sufficient. You should develop more tests to ensure your data structure is correctly implemented.

III. CircularLinkedList

The goal of this part is to implement a circular linked list. It is similar to `SingleLinkedListTail` except that it has **no head**. Head should not appear anywhere in the class.

Create a new class `CircularLinkedList` and the corresponding test class file, calling it `CircularLinkedListTestYOURNAME.java`. `CircularLinkedList` has to have the same public methods than `SingleLinkedList`. Write tests in `CircularLinkedListTestYOURNAME.java` to ensure your data structure is correctly implemented.

IV. Test Exchange

Once you are done implementing and testing `SingleLinkedListTail` and `CircularLinkedList`, find a classmate with who to exchange your two test files,

- `SingleLinkedListTailTestYOURNAME.java` and
- `CircularLinkedListTestYOURNAME.java`

Use the Forum on Moodle to find a classmate who is ready to exchange files. Run your classmate test files. Report in your `readme` about the process. Did all the tests passed right away in both cases? Did your classmate test file highlighted a bug in either of your implementation?

Submit

Before the start of next lab submit on Moodle a `zip` file containing your six `.java` files and your `readme` in plain text. Your `readme` should include

- your student information at the top,
- your report on the test file exchange and
- the worst case and best case running times of the add and remove methods for each of the list implementation (single, singletail, and circular)

Bring your diagrams to next week lab to show them to us.