CSC 115: Fundamentals of Programming II
Assignment #2: Linked Lists

Due date

Wednesday, February 18, 2015 at 9:00 pm via submission to conneX.

How to hand in your work

Submit the requested files through the Assignment #2 link on the CSC 115 conneX site. Please ensure you follow all the required steps for submission (including confirming your submission).

Learning outcomes

When you have completed this assignment, you should have become acquainted with:

- How to create an implementation of TaskList using a *reference-based linked list*.
- How to *use reference value* including the *null* value.
- How to write a Tester class alongside your implementation.

Task Scheduling

Problem description

Modern computer operating systems (such as Mac OS X, Linux, and Windows 8) are designed and constructed around several main concepts. One concept is the *task* (sometimes called a *process* or *thread*). When you use the Java compiler, it runs as a task; when you start up a web browser, it runs as a task. There are also many tasks running "in the background". In fact, some tasks generate additional tasks in order to complete some larger, more involved computation. Every task has many properties, including those known as *task priority* and *task number*.

Since an operating system (or *OS*) is organized around the running of tasks, there exists a central algorithm that schedules tasks, i.e., it chooses the next task to run when the current task must halt or pause. Making this choice is relatively easy: the scheduler simply takes the next available task having the highest priority, and the task is then given a chance to run on the computer. A *task list* is a data collection used to arrange waiting tasks in an order suitable for the OS scheduler. That is:

- When a task is created, it is placed into the task list at a position appropriate for the task's priority and arrival time.
- When the OS must choose a task to execute, it takes the one at the head of the task list.

In this assignment will not be modifying an existing operating system (!) but we will play with an implementation of operations needed for a *Task List ADT*. The data structure to be used for the implementation is a *referenced-based linked list*.

Examples

Imagine the following four tasks arrivals (also called four *TA* events) that occur in the order shown:

```
(10   212)
(12   100)
(10   198)
(3   104)
```

The first number in each line corresponds to that task's *priority*, and the second number corresponds to the task *number*. Here our first task has a priority of 10 and a task number of 212. We want our list organized such that higher priority tasks are closer to the head, but tasks with the same priority are in order of arrival time (i.e., earlier tasks are closer to the head). If the four TA events are the first ones in our computer, then the resulting task list will be:

```
(12   100), (10   212), (10   198), (3   104)
```

When the scheduler chooses a task for execution (an event denoted as *SC*) it removes the first task from the list—in fact, what the scheduler needs most is the task number of the task at the head of the list. Therefore the next task to be scheduled is:

```
100
```

and the task list after the SC event is:

```
(10   212), (10   198), (3   104)
```

Sometimes an OS decides a task must *deleted* before the task has a chance to execute. This means removing the task from the list (a *TD* event). Suppose task number 198 is to be deleted. Once deletion occurs, our resulting task list is:

```
(10   212), (3   104)
```

Knowing the number of tasks in the task list (or asking for its status, or an *ST* event) can be important for other parts of the operating system. Since the number of tasks is simply the list's length, we can see there are currently two tasks in the system. If we add three more TA events:

```
(10  85)
(4  314)
(2  101)
```

then the resulting list is:

```
(10  212), (10  85), (4  314), (3  104), (2 101)
```

and an ST event would report:

```
[5]
```

Task List ADT

I have described four events:

- TA: Task Arrival (*appropriately insert a new task to task list*)
- SC: Schedule Task (*remove task from head of list*)
- TD: Task Deletion (*remove task from task list*)
- ST: Status (*report number of tasks in the list*)

and these can be implemented via operations on a Task List ADT. This ADT's operations are very similar to what we have already seen with a List ADT. What follows is a list of ***Task List ADT*** operations and how they relate to the four kinds of events above.

- `void insert(Task t)`: Needed for a TA event.
- `Task removeHead()` and `boolean isEmpty()`:: Needed for an SC event.
- `Task remove(Task t)`: Needed for a TD event.
- `int getLength()`: Needed for an ST event.

We can also describe one more operation available in our Task List ADT:

- `Task retrieve(int i)`: Returns the task at position `i` in the task list, where `i=0` refers to the first task in the list. The list is unchanged regardless of the result returned by this operation.

For this assignment you are to complete a reference-based linked-list implementation of the `TaskList.java` interface provided with the assignment.

- The implementation will be called *TaskListRefBased.java.*
- At the same time as you write the implementation of this class, you must also write your *TaskListTester.java* program.
- Once *TaskListRefBased.java* is complete, you will modify *TaskSchedulerSim.java* to use an instance of your class inside the simulation. (This is a change to a single line.)

The five pairs of test files provided for this assignment are meant for use by the provided event simulator. To run the simulator using the first test case, we would invoke the following command:

```
$ java TaskSimulatorSim -i in01.txt -o output.txt
```

and here the results stored in `output.txt` must match the contents of the provided out01.txt. A noisy and incomplete implementation of the `TaskList` interface called *TaskListEcho.java* is provided for you so that *TaskSimulatorSim.java* can compile.

What you are to write

1. Please keep all code within a single directory. We will take advantage of what is called the *default package* in Java, i.e., all Java files in the same directory are considered to be in the same package. This will permit you to use package-private access for the assignment without a complex directory structure.

2. Write *TaskListRefBased.java* (you may want to start with the method headers in *TaskListEcho.java*). **The constructor for this class must not specify a maximum size for the linked list!**

3. Use the class *Task.java* to pass task information to methods as parameters, and to receive task information from methods as return values.

4. Use either *TaskListNode.java* or *TaskListNodeAlt.java* for the nodes of your linked list. Each of these files has a slightly different structure, so please choose the one which you feel is most natural for your coding style.

5. As you complete the work for step 2, you must write tests in *TaskListTester.java*. These tests will be in order of increasing difficulty and this order should reflect your implementation strategy. Ideally you will write a test in *TaskListTester.java* before adding relevant implementation code in *TaskListRefBased.java*. (An appendix at the end of this assignment description gives an example a test based on the first part of the example given earlier in this document.) **You must have at least ten tests, and <u>none</u> may be equivalent to the five sets of test files.**

6. Modify *TaskSchedulerSim.java* in order to use your new `TaskListRefBased` class. Test the result with the inputs provided with this assignment.

7. In all of your coding work, please follow the coding conventions posted at conneX (i.e., a document in the "Lectures & stuff" section).

Files to submit (six in total):

1. *TaskListRefBased.java*
2. *TaskList.java*
3. *TaskListNode.java or TaskListNodeAlt.java (depending on your choice)*
4. *TaskListTester.java*
5. *Your modified TaskSchedulerSim.java (note: only one line is to be changed)*
6. *A file named test_output.txt (which also includes a description of what each test is checking followed by the test input and output)*

Grading scheme

- "A" grade: An exceptional submission demonstrating creativity and initiative going above and beyond the assignment requirements. The program runs without any problems and implements the required classes and methods in those classes. Public methods are tested with output showing tests and their results. Any extra work appears in class files having the name *<file>Extra.java*, and identified within the file (i.e., Class comment) are details of your extension to the assignment demonstrating creativity and initiative.

- "B" grade: A submission completing the requirements of the assignment. The program runs without any problems and implements the required classes and methods in those classes. Public methods are tested with output showing tests and their results.

- "C" grade: A submission completing most of the requirements of the assignment. The program runs with some problems but does implement the required classes and methods, and yet might not have the expected output. Methods are tested with output showing tests and their results.

- "D" grade: A serious attempt at completing requirements for the assignment. The program runs with major problems, or does not contain implementations of all required classes and methods.

- "F" grade: Either no submission given, the submission does not compile, or submission represents very little work.

Appendix

Example of a test case suitable for *TaskListTester.java*

```
// Test based on assignment example. Is the task number
// returned really 100?  Is the resulting list really
// that with tasks numbered 212, then 198, then 104?
boolean passed = true;
int expected[] = {212, 198, 104};
TaskList tl = new TaskListRefBased();
tl.insert(new Task(10, 212));
tl.insert(new Task(12, 100));
tl.insert(new Task(10, 198));
tl.insert(new Task(3, 104));
Task head = tl.removeHead();
if (head.number != 100) {
    passed = false;
}
for (int i = 0; i < expected.length; i++) {
    Task t = tl.retrieve(i);
    if (t != null && t.number != expected[i]) {
        passed = false;
        break;
    } else if (t == null) {
        passed = false;
        break;
    }
}
if (passed) {
    System.out.println("First assignment example: pass");
} else {
    System.out.println("First assignment example: FAIL");
}
```