CSE 522
Homework 3: Generics, Collections, and Threads

Assigned: October 5, 2017
Due: October 16, 2017 (11:59 pm)

Part 1: Generic Tree Iterator

Lecture 11 provides the outline of a generic tree iterator class, `TreeIterator<T>`. Complete the definition of this class by writing the code for the `next()` and `hasNext()` methods. Place your definition of `TreeIterator<T>` in the program `GenericIterator.java`, run the program, and save the object diagram (stacked with tables) and sequence diagram in files `part1_obj.png` and `part1_seq.png`.

Implementing a tree iterator. Refer to the file `TreeIterator.png` for a pictorial guidance on how to implement a tree iterator. The key idea is to maintain a private stack of trees in class `TreeIterator<T>`. The stack maintains the invariant that the next value to be returned by the iterator is present at the root of the tree pointed to by the top of the stack. Each time a value is returned by `next()`, the stack should be updated so that the next value to the returned can be accessed from the top of the stack.

What to Submit: Prepare a top-level directory named *HW3_Part1_UBITId* where *UBITId* is your UBIT id. In this directory, place your source file `GenericIterator.java` and your diagrams `part1_obj.png` and `part1_seq.png`. Compress the directory and submit the compressed file using the `submit_cse522` command.

Part 2: Elevator Scheduler

Consider a single elevator that serves several people, each of whom goes from one floor to another (randomly chosen) floor in a repetitive manner. The elevator scheduler also operates in a repetitive cycle serving all requests in one direction before reversing its direction. The scheduler and the people are modeled as threads in `ElevatorScheduler.java`. This skeletal program provides the complete details of the `main` thread, the class `Person` and class `Elevator`. It also outlines the overall strategy of the elevator scheduler in class `Scheduler` - your task is to complete the code for this class.

The scheduler maintains two sorted lists (`SortedSets`), called `up` and `down`, for recording the requests (floor numbers) of people who want to up and down, respectively. When the elevator is moving up, for example, it serves all requests on the `up` list until it becomes empty; and, it continues to move up so that it may serve people on floors above its current position but want to go down. It behaves in a similar way when moving down.

Run your program for three different cases: when only one person is using the elevator; when three people are using the elevator; and when six people are using the elevator. For each case, run the program for at least 50,000 JIVE events and obtain the state diagram for the fields

`Elevator:1->current_floor` and `Elevator:1->direction`.

Name the three state diagrams as `part2_case1.png`, `part2_case2.png`, and `part2_case3.png` respectively.

Your scheduler can be considered to be correctly designed if the state diagram has the following property: For every pair of adjacent nodes in the state diagram, (f1,d1) → (f2,d2):

(a) if d1=d2=down, then f1 > f2;
(b) if d1=d2=up, then f1 < f2; and
(c) if d1 ≠ d2, then f1 = f2.

Property (a) is for the elevator moving down; property (b) is for the elevator moving up; and property (c) is for the elevator changing its direction.

What to Submit: Prepare a top-level directory named *HW3_Part2_UBITId* where *UBITId* is your UBIT id. In this directory, place your source file `ElevatorScheduler.java` and your diagrams `part2_case1.png`, `part2_case2.png`, and `part2_case3.png`. Compress the directory and submit the compressed file using the `submit_cse522` command.

End of Homework #3