CS 284: Homework Assignment 3
Due: March 28, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This assignment consists of implementing a priority queue *for task scheduling.* The first task in our to-do list will be the highest priority task that is stored at the front of the queue. Tasks will be added at the rear if no priority is assigned to a certain task, or added at their respective place according to their priority. The priority queue will be implemented by using a single-linked list.

Before going further, let's take a step back and recall some basic notions regarding single-linked lists.

As explained in the lectures, a single-linked list is a list in which each node has a reference to the next one and head/front indicates the first node in the list. The corresponding Java class for our priority queue, named `ListQueue<E>`, has two data fields or attributes:

- `Node<E> front`
- `int size`

Accessing the elements of the list is therefore realized through the reference to the `front` Node. For example, the i-th element is obtained by starting from `front` and then jumping through $i-1$ nodes. In regular single-linked lists, accessing an element is of time complexity $\mathcal{O}(n)$. But we will incorporate the addition of priority information into the nodes and this way, we will accomplish the highest priority task in $\mathcal{O}(1)$ time, because the highest priority task will always be the first task in the queue.

There will be two add methods in this queue implementation. `offer(E item, int priority)` method will add a task into the queue in its respective place according to its priority. `addRear(E item)` method will add a task at the end of the queue and will assume the task has the lowest priority (which is MAX_VALUE of the Integer class). If there exists at least two equal priority tasks, they will be ordered in the first-come first-serve fashion.

When a user cross off a task from the list, user can do it in two ways: removes the most urgent/high priority task or removes a task in the ordered list according to its task number. Please see the example output file for more information.

You are requested to implement three classes `ListQueue<E>, TaskList<E>, TestTaskList<E>` that encodes the priority queue, following the guidelines presented in the next section. You can embed `Node<E>` and an iterator class `Iter` to the `ListQueue` class.

2.1 Design of the Class `ListQueue<E>`

2.1.1 The Inner Class `Node<E>` (6 points)

First of all, a public inner class `Node<E>` should be declared. This class should include three data fields:

- `E data`
- `Node<E> next`
- `int priority`

It should also include the following operations:

- `Node(E dataItem)`, a constructor that creates a node holding `dataItem`. No priority as a parameter is given, therefore the task will be assigned LOW_PRIORITY.
- `Node(E dataItem, int priority)`, a constructor that creates a node holding `dataItem`, with `priority` as the priority of the task.
- `Node(E dataItem, Node<E> next, int priority)`, a constructor that creates a node holding `dataItem`, with `next` as next and `priority` as the priority of the task.
- `E getData()` will return the data stored in the attribute.
- `Node<E> getNext()` will return the next node.

2.1.2 The Inner Class `Iter` (4 points)

Then, another inner class `Iter` should be declared. This class should include one data field:

- `Node<E> next = front`

It should also include the following operations:

- `boolean hasNext()` will return true when the next Node not equal to null.
- `E next()` return the data stored in the next node (attribute) and will update the next node attribute with the next node of next node (we will go to the next node with this update). It will throw `NoSuchElementException` if next node is null.
- `remove()` throws UnsupportedOperationException.

2.1.3 The Class `ListQueue<E>` (35 points)

The class `ListQueue<E>` should include the declaration of the inner public class `Node<E>` and the private class `Iter`. Apart from that, it should have two data fields:

- `Node<E> front`
- `int size`

You are requested to implement the following operations (a summary is provided at the end of this assignment, in a UML diagram) for `ListQueue<E>`:

- (1 point) `public ListQueue()`, that creates an empty single-linked list representing the priority queue.
- (2 points) `public ListQueue(Node<E> first)`, that creates a one-element single-linked list representing the priority queue. `first` parameter will be stored in `front` of the queue.
- (3 points) getter/setter methods for `front` and getter for `size` attribute.
- (2 points) `E peek()` method returns the information at the front of the queue.
- (8 points) `public boolean offer (E item, int priority)` that adds `item` to a position according to its `priority`. If there exists same-priority tasks in the list, item will be added after the existing same-priority level tasks. Throws `NullPointerException` if the item sent to the method is null.
- (5 points) `public boolean addRear (E item)` that adds `item` at the end of queue. It always returns true except it throws `NullPointerException` if the item sent to the method is null.
- (5 points) `public E poll()` returns the data at the front of the queue and removes it from the queue. Throws `NullPointerException` if the item at the front of the queue is null.
- (8 points) `public boolean remove (Node<E> toBeRemoved)` takes a node to be removed and removes it from the queue. Correct links needs to be established after the node is removed.
- (1 point) `public Iterator<E> iterator()` method will return an instance of Iter class that we defined inside the `ListQueue<E>` class.

2.2 The Class `TaskList<E>` (30 points)

The class `TaskList<E>` should have five data fields:

- `ListQueue<E> all`
- `ListQueue<E> completed`
- `ListQueue<E> active`
- `int LOW_PRIORITY = Integer.MAX_VALUE`
- `int HIGH_PRIORITY = 1`

You are requested to implement the following operations (a summary is provided at the end of this assignment, in a UML diagram) for `TaskList<E>`:

- (2 points) `public TaskList()`, that initializes all the ListQueues in the attributes.
- (3 points) `boolean createTask(E item)` will add the item into `active` and `all` queues with default priority as LOW_PRIORITY. If item is null, it will return false. Otherwise returns true.
- (3 points) `boolean createTask(E item, int priority)` will add the item into `active` and `all` queues. If item is null, it will return false. Otherwise returns true.
- (3 points) getter methods for all the list queues `all, completed and active.`
- (2 points) `void printTopThreeTasks` method prints the top 3 highest priority tasks.
- (3 points) `void showActiveTasks(), showAllTasks(), showCompletedTasks()` will call the helper method `printTasks(ListQueue<E> queue)`, queue is indicating the respective list we sent to this helper method (active, all or completed).
- (4 points) `private void printTasks(ListQueue<E> queue)` helper method will use the iterator() to iterate through the queue elements and print them with numbers. Front of the queue will have the task number 1 and each next node will have an increasing task number.
- (5 points) `boolean crossOffMostUrgent()` will remove the highest priority task from the front of the queue and returns true if it successfully removes. If it does not exist, it prints an error message and returns false.
- (5 points) `boolean crossOffTask(int taskNumber)` removes the task at the location identified by taskNumber. Front of the queue will have 1 as the taskNumber and numbers will be incremented by one for each next task. Numbers can be seen in the printed list. Returns true for successful run. If task number is greater than the size of the active list or could not remove it successfully, method returns false.

2.3 The Class `TestTaskList<E>` (10 points)

The class `TestTaskList<E>` should have two data fields:

- `TaskList<E> toDoList`
- `Scanner scan`

You are requested to implement the following operations (a summary is provided at the end of this assignment, in a UML diagram) for `TestTaskList<E>`:

- (1 points) `static void main(String[] args)` creates a TestTaskList object and calls the printMenu method from it.
- (3 points) An operation `void printMenu()` for printing the 8 menu items as shown in example output file. In this method, we take a number from the user which indicates the operation user chooses. From this method, call the helper method processMenuItem(). Repeat printing the menu until user enters 8 to quit the program. Check if the user entered values between 1 and 8, if not print an error message and ask for input again.
- (6 points) A helper method `private boolean processMenuItem(int menuItem)` takes one input: operation that user chose (menuItem). It calls the appropriate functions for each operation from the toDoList object.

3 Submission instructions

Submit a single file named `HW3.zip` through Canvas that includes `ListQueue.java`, `TaskList.java` and `TestTaskList.java` with your main method. No report is required. Your grade will be determined as follows:

- You will get 0 if your code does not compile.
- The code must implement the following UML diagram precisely.
- We will try to feed erroneous and inconsistent inputs to all methods. All arguments should be checked.
- Partial credit may be given for style, comments (5 pts) and readability (5 pts).
- Provide your name and pledge on all Java files (5 pts).

The private inner class `Node<E>` and `Iter` should follow the UML diagrams:

```
Node
---------------------------------------------
– E data – Node<E> next
– int priority
---------------------------------------------
+ Node(E dataItem)
+ Node(E dataItem, int priority)
+ Node(E dataItem, Node<E> ref, int priority)
+ E getData()
+ Node<E> getNext()
```

```
Iter implements Iterator<E>
---------------------------
– Node<E> next = front
---------------------------
+ boolean hasNext()
+ E next()
+ void remove()
```

The class `ListQueue<E> ,TaskList<E>, TestTaskList<E>` should include the following operations:

ListQueue
– Node<E> front – int size
+ ListQueue() + ListQueue(Node<E> first) + Node<E> getFront() + void setFront(Node<E> front) + int getSize() + E peek() + boolean offer(E item, int priority) + boolean addRear(E item) + E poll() + boolean remove(Node<E> toBeRemoved) + Iterator<E> iterator()

TaskList
– ListQueue <E> all – ListQueue <E> active – ListQueue <E> completed – int LOW_PRIORITY = Integer.MAX_VALUE – int HIGH_PRIORITY = 1
+ TaskList() + boolean createTask(E item) + boolean createTask(E item, int priority) + void printTopThreeTasks() + void showActiveTasks() + void showCompletedTasks() + void showAllTasks() – void printTasks(ListQueue<E> queue) + boolean crossOffMostUrgent() + boolean crossOffTask(int taskNumber) + ListQueue<E> getAll() + ListQueue<E> getCompleted() + ListQueue<E> getActive()

TestTaskList
– TaskList <E> toDoList – Scanner scan
+ void main(String[] args) + void printMenu() – boolean processMenuItem(int menuItem)