Assignment 4: Queues

Part 1: Queues

CS3305/W01 Data Structures

Casey Hampson

September 9, 2024

Program Output

```
Added chore "Do laundry"(priority 5).
Added chore "Do homework"(priority 10).
Added chore "Clean the dishes"(priority 3).
Added chore "Clean room"(priority 6).
Added chore "Take out the trash"(priority 1).
Added chore "Fix the television"(priority 8).

Popping chores from priority queue:
Time to "Do homework"(priority 10)!
Time to "Fix the television"(priority 8)!
Time to "Clean room"(priority 6)!
Time to "Do laundry"(priority 5)!
Time to "Clean the dishes"(priority 3)!
Time to "Take out the trash"(priority 1)!
```

Source Code

```java
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 04-Part-1-Queues


import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;


class Queue<S extends Chore> {
    final private LinkedList<S> chore_list;

    // default constructor to initialize the linked list
    public Queue() {
        chore_list = new LinkedList<>();
    }

    // searches through the linked list and addes the element
    // at an appropriate position for its priority
    // : higher priority corresponds to a higher number
    public void Add(S item) {
        // if the chore is not inserted (i.e. this variable remains false),
        // then it must be the lowest priority (or something weird happened)
        boolean chore_inserted = false;
        for (int i=0; i<(this.chore_list.size()); i++) {
            if (Chore.ComparePriorities(item, this.chore_list.get(i)) == 0) {
                chore_list.add(i, item);
                chore_inserted = true;
                break;
            }
        }
        // aforementioned case where this chore has lowest priority
        if (!chore_inserted) {
            chore_list.addLast(item);
        }

        System.out.printf(
            "Added chore \"%s\"(priority %d).\n",
            item.GetDescription(), item.GetPriority()
        );
    }
    // removes the first, and highest priority, chore
    public S Pop() {
        return chore_list.removeFirst();
    }

    // returns true if there are still elements in the queue
    // used to parse through the chorelist later
    public boolean HasNext() {
```

```
        return !this.chore_list.isEmpty();
    }
}


// chore class to store priority
class Chore {
    final private String  description;
    final private int     priority;


    public Chore(String desc, int priority) {
        this.description = desc;
        this.priority = priority;
    }

    public String   GetDescription() { return this.description; }
    public int      GetPriority() { return this.priority; }

    // returns 0 if the first chore has a greater priority than the second chore,
    // otherwise returns non-zero
    public static int ComparePriorities(Chore chore1, Chore chore2) {
        if (chore1.priority >= chore2.priority) return 0;
        return 1;
    }

}


public class P1 {
    public static void main(String[] args) {
        // create a bunch of chores in a random order with various priorities
        ArrayList<Chore> chore_list = new ArrayList<>();
        Collections.addAll(chore_list,
            new Chore("Do laundry", 5),
            new Chore("Do homework", 10),
            new Chore("Clean the dishes", 3),
            new Chore("Clean room", 6),
            new Chore("Take out the trash", 1),
            new Chore("Fix the television", 8)
        );
        Queue<Chore> chore_queue = new Queue<>();
        chore_list.forEach(chore -> chore_queue.Add(chore));

        // pop them out in order to get highest priority ones first
        System.out.printf("\nPopping chores from priority queue:\n");
        while(chore_queue.HasNext()) {
            Chore current_chore = chore_queue.Pop();
            System.out.printf(
                "Time to \"%s\"(priority %d)!\n",
                current_chore.GetDescription(), current_chore.GetPriority()
            );
        }
```

```
    }
}
```