Assignment 5 – Part 1: Queues

Ryan Brinson

9/22/2023

Output:

```
rbrinson2@DESKTOP-U8KJ4OP:~/Documents/CS3305/Assignments/A5$ java A5
Enter chore: swim
Enter Priority (1-100): 50
eat 75
clean 75
swim 50
sweep 46
bath 15

clean 75
swim 50
sweep 46
bath 15

swim 50
sweep 46
bath 15

sweep 46
bath 15

bath 15

rbrinson2@DESKTOP-U8KJ4OP:~/Documents/CS3305/Assignments/A5$
```

Code:

```java
// Name: Ryan Brinson
// Class: CS 3305 W04
// Term: Fall 2023
//  Instructor:  Carla McManus
//  Assignment:  05-Part-1-Queues


import java.util.LinkedList;
import java.util.Scanner;

public class A5 {
```

```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        ChoreQueue choreQu = new ChoreQueue();

        // Ask the user to input a custom chore to test user input
        choreQu = enterChore(input, choreQu);

        // Assign two chores with random priority
        choreQu.offerChore("bath", (int)(Math.random() * 101));
        choreQu.offerChore("sweep", (int)(Math.random() * 101));

        // Assign two chores with the same priority
        choreQu.offerChore("eat", 75);
        choreQu.offerChore("clean", 75);


        do{
            // Print the chore list
            choreQu.printChores();
            System.out.println();
            // Dequeu the choes as they are done
            choreQu.pollChore();

        } while (!choreQu.quIsEmpty()); // Test if the chore list is emptry
    }

    public static ChoreQueue enterChore(Scanner in, ChoreQueue c){
        String tempC = new String();
        int tempP = 0;

        // Ask the user to input chore and grab it
        System.out.print("Enter chore: ");
        tempC = in.nextLine();

        // Ask the user to input priority of the chore and grab it
        System.out.print("Enter Priority (1-100): ");
        tempP = in.nextInt();

        // Offer the chore and priority to the queue and then return it
        c.offerChore(tempC, tempP);
        return c;
    }
}

// The first class is the Chore Class
```

```
//
// This binds the chore and priority together
// for later use in tandom in the queue. It
// has limitited extended functionality and only lets
// you set and get the store content

class Chore {
    private Integer priority;
    private String chore;

    // Cunstructor with no initial input
    Chore(){
        priority = 0;
        chore = "\0";
    }

    // Cunstroctor that takes in chore and priority
    Chore(String chore, Integer priority){
        this.chore = chore;
        this.priority = priority;
    }

    public Integer getPriority(){
        return priority;
    }
    public String getChore(){
        return chore;
    }
    public void setPriority(Integer priority){
        this.priority = priority;
    }
    public void setChore (String chore){
        this.chore = chore;
    }

}

// The second class that queues up the chores
//
// This class was setup to store the Chore class
// into a Linked List. It has some specific extended
// functionality, such as a sort that sorts the
// list based off it's priority each time a chore
// is queued
```

```
class ChoreQueue{
    private LinkedList<Chore> choreQueue;

    // Cunstructor that initializes the Linked List
    ChoreQueue(){
        choreQueue = new LinkedList<>();
    }

    // Runs through each chore in the list and prints
    // the chore and it's priority
    public void printChores(){
        // Using the modified for each
        for (Chore chore : choreQueue) {
            // Outputs the chore name then a space then the priority
            System.out.println(chore.getChore() + " " + chore.getPriority());
        }
    }

    // A simple bubble sort that is called each time an item
    // is stored in the linked list
    public void sortChores(){
        // k is the leading index pointer
        for (int k = 1; k < choreQueue.size(); k++){
            // i is the lagging index pointer
            for (int i = 0; i < choreQueue.size() - k; i++){
                // Each iteration checks the current indexed chore priority
                // and compares it to the next one. If the next priority
                // is smaller, then they are swapped
                if (choreQueue.get(i).getPriority() < choreQueue.get(i +
1).getPriority()){
                    Chore temp = choreQueue.get(i);
                    choreQueue.set(i, choreQueue.get(i + 1));
                    choreQueue.set(i + 1, temp);
                }
            }
        }
    }

    // Method that uses the standard offer but adds a sort
    // Takes as input a Chore class
    public void offerChore(Chore chore){
        choreQueue.offer(chore);
        sortChores();
    }
```

```java
    // Method that uses the standard offer but adds a sort
    // Takes as input a separate chore and priority
    public void offerChore(String chore, Integer priority){
        Chore temp = new Chore(chore, priority);
        choreQueue.offer(temp);
        sortChores();
    }

    // A standard peek that returns
    public Chore peekChore(){
        return choreQueue.peekLast();
    }

    // A standard poll that checks if the queue is empty first
    public Chore pollChore(){
        // If it's empty return null
        if (quIsEmpty()) {
            System.out.println("Chore Queue is empty");
            return null;
        }
        else return choreQueue.poll();
    }

    public boolean quIsEmpty(){
        return choreQueue.isEmpty();
    }

}
```