C212 Lab 11

Last Updated 11/17/2021

Objectives

- Stacks
- Linked Lists
- Queues/Priority Queues
- Sets

Lab Instructions

Please complete the following exercises. For each, review the requirements given below and include all stated specifications. Please compress your project, which contains all of your files, into a .zip and submit it through Canvas. The grading scheme will also be provided in Canvas, which you may review as you work.

Required Files to Submit (by task)

- **Stack**
  - A Java File
- **LinkedList**
  - A Java File
- **Queues/Priority Queues**
  - Customer.java
  - Ride.java
  - AmusementPark.java
- **Sets**
  - A Java File

Suggestions

- Comment the start of each class with what its purpose is.
- Comment each function with the appropriate JavaDoc style of comments. Look at the Lab 03 solutions for examples.
- Separate each of the 4 tasks into their own package to help keep track of your progress across each.

1. Stacks

Task

- In this part of the Lab, you will be asked to create two methods that determine whether different inputs are palindromes **using collection Stack only**. This will test your knowledge of Java Collections Framework.

- Given the two method header below:

```
//Note that you can't convert this int to String for processing for isIntPalindrome.

//Output example:
//Boolean isIntPalindrome(101) -> true
//Boolean isIntPalindrome(1221) -> true
//Boolean isIntPalindrome(131415) -> false

public static Boolean isIntPalindrome(int num);

//for this isStrPalindrome(String str) ,
//you can't put the whole string in and return with compare, you must use
//substring or charAt.
//you can only store half of the given string to stack, and use that to compare,

//such as:
//input "abccba", you would like to store "a","b","c" in stack, and use it to compare with
//the rest of the elements.
//input "abzba", you would like to store "a","b","z" in stack, and use it to compare with
//the rest of the elements.

//LowerCase and CapitalCase are considered the same, such as "A" equal to "a", "Z"
//equal to "z".

//Output example:
//Boolean isStrPalindrome("abCcba") -> true
//Boolean isStrPalindrome("aBzba") -> true
//Boolean isStrPalindrome("acba") -> false

public static Boolean isStrPalindrome(String str);
```

- Please have concise comment code, and have at least two tests for each method in the main.

Helpful Sources/Hints

- How to Write Doc Comments for the Javadoc Tool

2. Linked Lists

Tasks

- Complete the given class using only the given ListNode class.(similar to Part 3 in Lab6)

```
class ListNode {
  int val;
  ListNode next;
  ListNode(int val) { this.val = val; }
  ListNode(int val, ListNode next) { this.val = val; this.next = next; }
  int size(ListNode node){
     int count=0;
     ListNode temp = node;
     while(temp!=null) {
        temp = temp.next;
        count++;
     }
     return count;
  }
}
```

- Here are the given method header you need to complete:

```
//reverse the given node from
// for example:
// input 5->4->3->2->1, 0 , 1 ,  return 4->5->3->2->1
// input 5->4->3->2->1, 0 , 4 ,  return 1->2->3->4->5
// input 5->4->3->2->1, 0 , 5,   return null, and print ”either start or end is wrong”.
// input 5->4->3->2->1, 5 , 0,   return null, and print ”end can't be smaller than
// start”
// input 5->4->3->2->1,  6 , 0,   return null, and print ”either start or end is wrong”.

// hint: You would need to implement a helper function to reverse the ListNode,
// then use the helper function in reverseList(ListNode node, int start,int end);

public static ListNode reverseList(ListNode node, int start,int end)

//check if there is a cycle in the list, which means there is an infinite loop inside
//the argument.
//example:
//input: 1->2->3->4->3->4->3->4->... , return true
//input: 1->2->3, return false
```

public static Boolean ifCycleInNode(ListNode node);

- **For each method in this part, please write a comment about your logic of how you solve the problem, to show us you understand the concept, moveover, two tests for each method in the main is required.**

Helpful Sources/Hints

- How to Write Doc Comments for the Javadoc Tool

3. Queues/Priority Queues

Overview

- Create an Amusement Park that can have rides.
- Rides can have Customers ride them.
- Customers can enter the lines for a ride.
- Priority will be given to FastPass ticket holders, if the ride supports it.

Task

- Create a **Customer** class. A customer will consist of a name, and if they are a FastPass ticket holder or not.
    - Make the class implement a *Comparable<Customer>* where the customers who are FastPass ticket holders are prioritized over the non-fastpass ticket holders.
    - Create a constructor with a given name and if they are a fastpass ticket holder.
- Create a **Ride** class. A ride class will consist of a name, ride capacity, and a Queue of customers.
    - Create a *run* function that removes people from the queue, up until the *ride capacity* is reached. **Print out the order the people were removed from the queue along with their FastPass status.**
    - Create a function *addToQueue* that can add a list of customers to the queue.
    - Create appropriate constructor.
- Create an **AmusementPark** class. An Amusement Park class consists of a list of rides.
    - Be able to print the statistics of each ride in your Amusement Park (name, capacity, number of people in queue for that ride).
- Create an **AmusementParkExample** class that holds your main method and simulates your amusement park. There should be **at least 2 rides** where **one ride supports FastPass** ticket holders, and the **other does not take FastPass status into consideration**.
    - Main Method
        - Add at least two rides described above to the AmusementPark list of rides, and each queue to the ride should have at least 10 customers of

differing FastPass statuses. You must use at least one PriorityQueue and one other queue of your choice that is not a priority queue.
    - Print the statistics of your park.
    - Run each ride until the queue of people is empty.

Helpful Sources/Hints

- One queue has to be a priority queue. One queue has to be any other queue that is not a priority queue.
- How to Write Doc Comments for the Javadoc Tool
- Queue (Java Platform SE 8)
- PriorityQueue (Java Platform SE 7)
- Comparable (Java Platform SE 8)
- AbstractQueue (Java Platform SE 8)
- LinkedList (Java Platform SE 7)

4. Sets

Overview

- Simulate mathematical set operations using Java's Set interface.

Task

- Create a **MathSet** class that contains the following functions:
    - A static function called *intersection* that when given two sets of Integers, returns the intersection of the two sets.
        - setA = {1, 2, 3}, setB = {2, 3} ----> result = {2, 3}
    - A static function called *union* that when given two sets of Integers, returns the union of the two sets.
        - setA = {1, 4, 5, 6}, setB = {2, 3} ----> result = {1, 2, 3, 4, 5, 6}
    - A static function called *complement* that when given two sets of Integers, returns the complement of set A given set B.
        - setA = {1, 4, 5, 6}, setB = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} ----> result = {2, 3, 7, 8, 9, 10}
- Create a **MathSetTest** class that contains *JUnit* tests for all of the functions inside of the *MathSet* class.
    - Must test *intersection, union,* and *complement* of the *MathSet* class above with multiple different sets.
    - Must work given any type of set (HashSet, TreeSet, etc.) of integers.

Helpful Sources/Hints

- How to Write Doc Comments for the Javadoc Tool

- Set (Java Platform SE 8)