TCP1201 Object-Oriented Programming and Data Structures

Lab07 JCF Lists, Stacks, and Queues

Exercise 1: ArrayList

Write a program that performs the following functions:

1. Generates a set of 10 random integers of range 0-99. Use an array list to store the integers.
2. Show a menu to the user. The menu options include creating a new random list, sort, reverse, and shuffle the list.

*(**Hint**: Make use of Java ArrayList class and static methods from Collections class)*

Sample run

```
Random list: [50, 6, 52, 92, 22, 46, 65, 56, 44, 50]
Enter your choice:
1: New random list
2: Sort list
3: Reverse list
4: Shuffle list
0: Quit
> 1

Random list: [45, 17, 19, 25, 76, 66, 91, 78, 48, 93]
Enter your choice:
1: New random list
2: Sort list
3: Reverse list
4: Shuffle list
0: Quit
> 2

Sorted list: [17, 19, 25, 45, 48, 66, 76, 78, 91, 93]
Enter your choice:
1: New random list
2: Sort list
3: Reverse list
4: Shuffle list
0: Quit
> 3

Reversed list: [93, 91, 78, 76, 66, 48, 45, 25, 19, 17]
Enter your choice:
1: New random list
2: Sort list
3: Reverse list
4: Shuffle list
0: Quit
> 4

Shuffled list: [78, 48, 45, 76, 17, 25, 91, 19, 93, 66]
Enter your choice:
1: New random list
```

```
2: Sort list
3: Reverse list
4: Shuffle list
0: Quit
> 0
```

Exercise 2: LinkedList

Modify the program in Exercise 1 by **replacing the ArrayList with a LinkedList**.

*(**Hint**: Make use of Java LinkedList class and static methods from Collections class. The modification should be very simple since both ArrayList and LinkedList implement List<E> interface.*

Exercise 3: Stack

Understand the TowerOfHanoi problem by playing the animation at
https://yongdanielliang.github.io/animation/web/TowerOfHanoi.html

Then, write a program to let the user solve the Tower of Hanoi of 3 disks by themselves. You are not required to check input error, illegal move, and whether the problem is solved or not.

*(**Hint**: Use Java Stack class to represent the tower.)*

Sample run

```
Tower 0: [3, 2, 1]
Tower 1: []
Tower 2: []
Enter source tower and destination tower (-1 to exit): 0 1
Tower 0: [3, 2]
Tower 1: [1]
Tower 2: []
Enter source tower and destination tower (-1 to exit): 0 1
Tower 0: [3]
Tower 1: [1, 2]
Tower 2: []
Enter source tower and destination tower (-1 to exit): 1 2
Tower 0: [3]
Tower 1: [1]
Tower 2: [2]
```

Exercise 4: Queue and Priority Queue

The following program is provided.

1. Run the program. What output do you get?
2. Create a Queue of `Task`. Add some tasks to the queue. Then remove and print the tasks in the queue one by one until it is empty. Sample tasks:
   ```
   Task 1: value = 3, name = Press
   Task 2: value = 8, name = Drill
   Task 3: value = 1, name = Knock
   ```
3. Test your queue. What output do you get?

4. Create a PriorityQueue of Task. Add some tasks to the queue (use the same tasks as above). Then remove the tasks in the priority queue one by one until it is empty. The task with the largest value should be removed first. You need to update the Task class so that it can become elements in PriorityQueue.

   Sample run:

```
Removed from PriorityQueue<Task>: Task <value=8 name=Drill>
Removed from PriorityQueue<Task>: Task <value=3 name=Press>
Removed from PriorityQueue<Task>: Task <value=1 name=Knock>
```

```
import java.util.*;

public class Task {
  private int value;
  private String name;

  public Task(int value, String name) {
    this.value = value;
    this.name = name;
  }

  public String toString() {
    return "Task <value=" + value + " name=" + name + ">";
  }
}

class TestQueues {
  public static void main (String[] args) {
    PriorityQueue<Integer> intPQ = new PriorityQueue<>();
    intPQ.add (40);
    intPQ.add (60);
    intPQ.add (20);
    while (!intPQ.isEmpty())
      System.out.println ("Removed from PriorityQueue<Integer>: " +
        intPQ.poll());
}
```