CS 501B – Introduction to JAVA Programming
Fall 2023 Semester
Due: 11/17/2023 Friday at 11:59 PM

Instructions:

1. You are not allowed to use any package/library unless stated otherwise.
2. A skeleton Assignment8.java file is attached as a stub of code for Question 1, 3 and 4. For Question 2, Follow the question instructions.
3. You are required to make the necessary changes in the Assignment8.java file.
4. We will be testing your code on hidden test cases.
5. Test your code with your own test cases.
6. Please comment your name and CWID in the first two lines of the Assignment1.java file.
7. Add comments in your code about what you are doing.
8. You are required to zip only Assignment7.java file as FirstName_LastName_Assignment#.zip (Ex: Roushan_Kumar_Assignment1.zip).
9. This assignment covers topics from week 10.
10. Students are not allowed to collaborate with classmates and any other people outside. All work must be done individually. Any work having evidence of showing academic dishonesty violation is subjected to zero for the assignment.

Penalty:

1. 10 marks will be deducted for invalid format of file / assignment submission.
2. If you submit after the due date then 10 marks will be deducted for every day after the due day.
3. If you submit an assignment after two weeks from the due date then you will get zero marks.

Questions:
Each question carries **25** points and total points is **100.**

1. Complete the method **`public static <T extends Comparable<T>> T findMaxInRange(T[] dataArray, int fromIndex, int toIndex)`** that finds the maximum element within a specified range of a generic array. The **`fromIndex`** is inclusive, and **`toIndex`** is exclusive, following zero-based indexing. If **`fromIndex`** is greater than or equal to **`toIndex`**, or either index is out of bounds, the method should throw an **`IllegalArgumentException`**. For **`String`** arrays, the maximum element is the one which would appear last in a list sorted alphabetically (in lexicographical order).

   Example 1 :
   Integer[] numbers = {1, 3, 5, 6, 2, 9};
   Integer max = findMaxInRange(numbers, 1, 5);

System.out.println(max); // Output: 6

Example 2:
String[] words = {"apple", "orange", "banana", "pear"};
String maxWord = findMaxInRange(words, 0, 3);
System.out.println(maxWord); // Output: "orange"

2. Design the class `**MaxStack**` which extends a standard `**Stack<Integer>**` to support retrieving the maximum value in the stack in constant time. Implement the following operations: `**push(int x)**`, `**int pop()**` and `**int peekMax()**`.

   Example 1:
   MaxStack stack = new MaxStack();
   stack.push(2);
   stack.push(1);
   stack.push(5);
   stack.push(3);
   int max = stack.peekMax();
   System.out.println(max); // Output: 5
   stack.pop();
   stack.pop();
   max = stack.peekMax();
   System.out.println(max); // Output: 2

3. Implement the method `**public static <T> void interleaveQueue(Queue<T> queue)**` which rearranges the elements of a queue by interleaving the elements from the first half with those from the second half. If the queue has an odd number of elements, the middle element should stay in the same place. The method should work in-place using only queue operations.

   Interleaving in the context of queues is a process of rearranging the elements such that they are alternated from two halves of the original queue. For example, if you have a queue `**[1, 2, 3, 4, 5, 6]**`, after interleaving, you should have `**[1, 4, 2, 5, 3, 6]**`, which means the first element of the first half (1) is followed by the first element of the second half (4), and so on. If the queue has an odd number of elements, the middle element should remain in its place after interleaving, so a queue `**[1, 2, 3, 4, 5]**` would become `**[1, 4, 3, 2, 5]**`.

   Example 1:
   Queue<Integer> queue = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
   interleaveQueue(queue);
   System.out.println(queue); // Output: [1, 4, 3, 2, 5]

4. Complete the method `**public static <T extends Comparable<T>> T findMaxInRangeUsingPriorityQueue(T[] dataArray, int fromIndex, int toIndex)**` that efficiently finds the maximum element within a specific range of a generic array using a priority queue. The elements in the array must be of a type that implements the `**Comparable**` interface. The range within the array from which you should find the maximum element is defined by the `**fromIndex**` (inclusive) and the `**toIndex**` (exclusive), and it should follow zero-based indexing.

   Requirements:

   - If the `**fromIndex**` is equal to or greater than the `**toIndex**`, or if any of the indices are out of the array's bounds, the method should throw an `**IllegalArgumentException**`.

   - For arrays with elements of type `**String**`, determine the maximum element based on lexicographical (alphabetical) order—the element that would come last if sorted alphabetically.

   Notes:

   - Consider using Java's `**PriorityQueue**` to keep track of the maximum element as you iterate through the specified range of the array.

   - You are required to handle all edge cases related to the bounds of the array indices and invalid index ranges gracefully, throwing an exception when appropriate.

   Example 1:
   Integer[] numbers = {1, 3, 5, 6, 2, 9};
   Integer max = findMaxInRange(numbers, 1, 5);
   System.out.println(max); // Should print "6"