# OOPS Lab

## Programming Assignment № 11

Exception Handling and Multithreaded Programming in Java

---

CSE Department, JMI 14/11/2023

## Read carefully before you begin:

- Total Marks: 30. Each question carries 10 marks.
- You have 2 hours to complete the assignment. Failure to have your program evaluated before you leave the lab will cause forfeiture of the grade for that lab.
- In order to receive full marks, you must demo the full working code and show the output and given an explanation of your approach where applicable.
- Please **save your code** throughout the semester in a place where you do not lose it. You will be required to submit it at the end.
- Use proper filenaming conventions and commenting. Code that is hard to read or understand will incur a penalty.
- Collaboration must kept to general discussions only. Please do NOT share code or directly share answers with each other. Plagiarism is unacceptable.

## Problem 1 : Clear backlog from last week (10 marks)

1. Go back to last week's lab and finish the remaining tasks. You should have a working multi-threaded mergesort algorith in order to continue. Hint: https://www.geeksforgeeks.org/merge-sort-using-multi-threading/

## Problem 2: Exception Handling in Merge Sort

1. Modify both your single-threaded and multi-threaded merge-sort code to handle array out of bounds exception using a try-catch block.
2. Modify the norm function in your **PointN** class to handle the divide-by-zero exception using a try-catch block.

## Problem 3: Multithreaded Merge Sort (10 marks, 60 minutes)

1. Prepare a PDF report with answers to each question in this section.

| Algorithm | Size (n) | Exec. Time (ms) | Speed-up (over baseline) |
|---|---|---|---|
| Single-threaded (baseline) | 1,000 | | - |
| Multi-threaded | 1,000 | | |
| Multi-threaded w. manual optimization | 1,000 | | |
| Single-threaded (baseline) | 10,000 | | - |
| Multi-threaded | 10,000 | | |
| Multi-threaded w. manual optimization | 10,000 | | |
| Single-threaded (baseline) | 100,000 | | - |
| Multi-threaded | 100,000 | | |
| Multi-threaded w. manual optimization | 100,000 | | |
| Single-threaded (baseline) | 1,000,000 | | - |
| Multi-threaded | 1,000,000 | | |
| Multi-threaded w. manual optimization | 1,000,000 | | |

Table 1: Performance Gains Through Multithreading for Merge-Sort

2. In your report, describe your multi-threaded merge sort in words. How many threads are you spawning? How are you dividing up the work among them? What will the Java scheduler do in your program? It might be helpful to read about the **join** function (https://www.geeksforgeeks.org/joining-threads-in-java/).

3. Now, modify your program to catch the **InterruptedException** thrown by the **join** function and print out a statement "Thread interrupted". How many interruptions occur in your program? Document this in your report.

4. Manually optimize your multi-threaded sort using the techniques discussed in previous labs. Move non-iterative code outside of loops. Compute norms only once etc. In your report, provide a performance table as in Table 1. Be aware that performance is measured by running an algorithm at least 50 times and taking the average performance time to smooth over any random variations in execution time. Graph your results (size, n on x-axis) separately for execution time and speedup. Include the graph in your report. Your report should have a header that specifies the programming environment including hardware (processor name, speed, number of cores, RAM, cache), OS, compiler and Java development version, date, time and your name and roll number .