CSE201 Advanced Programming
Assignment 07 (Bonus marks)
IIIT-Delhi. 8th Nov 2019. Due by 11:59pm on 10th Nov 2019

Multithreading and Design Patterns

This is a take-home lab assignment. No extensions whatsoever will be provided. Any submission after the deadline will not be evaluated. If you see any ambiguity or inconsistency in a question, please seek clarification from the teaching staff. Please read the entire text below very carefully before starting your implementation.

Plagiarism: All submitted lab assignments are expected to be the result of your individual effort. You should never misrepresent someone else's work as your own. In case any plagiarism case is detected, it will be dealt as per IIITD plagiarism policy and without any relaxations: https://www.iiitd.ac.in/sites/default/files/docs/education/AcademicDishonesty.pdf

Please note that you are not allowed to discuss the design/solution of the lab assignment (e.g. classroom page discussions etc.). Anyone who is found doing this will be treated as a plagiarism case. No excuses!

Note that this a bonus assignment and is not mandatory. Please Turn-In (within the deadline) only in case you are interested in bonus marks.

We have seen two different parallel implementations of the Fibonacci number calculator during lectures 18 and 19. The first one was using Explicit Multithreading whereas the second implementation was using ForkJoinPool. We have also seen the implementation of the producer-consumer problem in lecture 20. This assignment is combining all those concepts along with the design patterns.

In this assignment, you have to implement a producer-consumer based Fibonacci number calculator in Java by using **explicit** multithreading. The program starts by asking the user about the total number of consumer threads to generate. This interaction will happen on a producer thread that will first create all the consumer threads, and then endlessly loop asking the user to enter the number whose Fibonacci number is to be calculated. The producer thread will not display the result for any input number provided by the user unless the user explicitly asks the producer thread to display the result of **all** pending queries. The producer will terminate only when specified by the user.

Every time the user enters a number whose Fibonacci value is to be calculated, the producer will push this number on a queue shared between the producer and the consumers. Note that there is a possibility that there could be multiple pending numbers in this queue in case all the consumers are busy. One of the consumer thread will take the number out from this queue and will calculate the Fibonacci result **recursively**, but **sequentially**, i.e., without using any

parallelism. Once it has calculated the result, it would push the result along with its computation time as one single task to another shared queue between the producer and the consumers. However, as you can note, this time the role is flipped and the thread pushing the result to the shared queue becomes a producer, as the result will be now be consumed by the original producer thread once instructed by the user to display all pending queries.

There is a small twist in this implementation. You must also ensure that none of the consumer threads should repeat the calculation of any specific Fibonacci number. Hence, your program would start with a slow computation time but will ultimately become very fast. You can assume that the user is not allowed to specify a number greater than 45. Also, your consumer threads should calculate the Fibonacci only for the number specified by the user.

You must use OOP techniques, explicit multithreading, and demonstrate all the following three design patterns in your implementation: a) Facade, b) Observer and c) Flyweight. You must ensure that you follow the coding style specific to each of these design patterns as mentioned in the lecture slides. You must ensure that all the threads in your program terminates gracefully and only then the program terminates. If the user asks for termination without asking for displaying all pending queries, your program should still display the results of all pending queries before initiating termination.

There is no need for a sample test case as the user interaction is very limited, and is clearly mentioned in the above description.