Assignment 2 – Thread-Safety

The 2nd assignment introduces the thread-safety concept learnt in class to our audio player. A correct implementation of this assignment must use thread-safe singletons, synchronization, lock-free implementations, Future objects (10 pts bonus), and interrupts.

Application:

There are two types of operations in this assignment – adding and removing a *Song* to a *Player.* Those operations are represented by *OP* instances. The “main thread” of your program will add *OP* instances to an *OPStack,* while six other threads will read the operations and execute them on the *Player.* You must make sure that the *OPStack* and *Player* objects are thread-safe singletons, and that all threads terminate.

Objects:

*Player –* must be implemented as a **thread-safe singleton.** Should support adding / removing a *Song*, and querying its size (number of songs it holds).

* You **must not** use any built-in data structures in your implementation of the player. Make your own linked list.
* Your implementation must be **lock-free.**
* The *size()* method of the *Player* should iterates over all songs and count them on the fly. It must not read the content of some counter.
* Since the *size()* method of the *Player* is called only upon termination, you can assume that no other threads interfere with it.

*OPStack –* shared among all threads in the program. Implements the *Stack* interface, and must be implemented as a **thread-safe singleton.**

* You can synchronize all the methods of the *OPStack.*
* You can use the built-in *ArrayList* in your implementation. You must not use any other built-in data structure.
* A thread that tries to pop an object from an empty *OPStack* **must be blocked,** until an *OP* is added to the *OPStack.*
* Pushing a (distinct) *OP* to the *OPStack* returns a *Future* object, that will hold (eventually) the “implied” order of execution.

*Future –* a wrapper for a value that will eventually be known.

* In case the value is already known, the *get()* method returns the value; otherwise, the calling thread will be **blocked** until the value will be known.
* The *resolve(int val)* method resolves the return value to be *@val.*

*AddSong implements Runnable –* this type of threads is responsible for adding songs to the player.

* If a thread of this type pops (from the *OPStack*) an *OP* object with *int op = 1,* it adds a *Song* to the *Player.* Otherwise, it re-pushes the *OP* object to the *OPStack.*
* After adding a *Song,* the thread will resolve the value of the corresponding *Future* with the “order” of the operation in execution.

*RemoveSong implements Runnable –* this type of threads is responsible for removing songs from the player.

* If a thread of this type pops (from the *OPStack*) an *OP* object with *int op = 2,* it removes a *Song* to the *Player.* Otherwise, it re-pushes the *OP* object to the *OPStack.*
* After removing a *Song,* the thread will resolve the value of the corresponding *Future* with the “order” of the operation in execution.

Main.java:

This file is fully provided for you. The main thread first creates an *OPStack* instance, and initializes six threads (that immediately try to fetch and execute messages). Then, the main thread reads the command line arguments, pushes corresponding *OP* objects to the *OPStack,* and stores the *Future* objects it gets it return in a *List.* After 1000 milliseconds, the main thread interrupts all other threads, waits for them, and prints some “debug output”.

**Make sure that all other threads terminate as well…**

Submission:

* The due date for submission is 15/6/2024.
* Submission is in pairs only.
* Submit only .java files.

Input-output:

Multi-threaded programs often have non-deterministic output. Make sure that your output makes sense, especially when you run your application with small outputs, and make sure all threads terminate.

Suppose for example your input is **2 1 2.** A reasonable output is –  
>  
3  
1  
2  
size 0  
>

This output indicates that the size of the *Player* is indeed 0 (after adding one *Song* and removing), and that the “add” operation was executed first.

If the output is **1 1 1 2 2,** the following output looks good –  
>  
3  
4  
1  
2  
5  
>