/\*

StaticThreadController.h - Controlls a list of Threads with different timings

Basicaly, what it does is to keep track of current Threads and run when

necessary.

StaticThreadController is an extended class of Thread, because of that,

it allows you to add a StaticThreadController inside another kind of ThreadController...

It works exact as ThreadController except you can't add or remove treads dynamically.

Created by Alex Eremin, September, 2016.

Released into the public domain.

\*/

#ifndef StaticThreadController\_h

#define StaticThreadController\_h

#include "Thread.h"

template <int N>

class StaticThreadController: public Thread{

protected:

//since this is a static controller, the pointers themselves can be const

//it should be distinguished from 'const Thread\* thread[N]'

Thread \* const thread[N];

public:

template <typename... T>

StaticThreadController(T... params) :

Thread(),

thread{params...}

{

#ifdef USE\_THREAD\_NAMES

// Overrides name

ThreadName = "StaticThreadController ";

ThreadName = ThreadName + ThreadID;

#endif

};

// run() Method is overrided

void run() override

{

// Run this thread before

if(\_onRun != nullptr && shouldRun())

\_onRun();

for(int i = 0; i < N; i++){

// Is enabled? Timeout exceeded?

if(thread[i]->shouldRun()){

thread[i]->run();

}

}

// StaticThreadController extends Thread, so we should flag as runned thread

runned();

}

// Return the quantity of Threads

static constexpr int size() { return N; };

// Return the I Thread on the array

// Returns nullptr if index is out of bounds

Thread\* get(int index) {

return (index >= 0 && index < N) ? thread[index] : nullptr;

};

// Return the I Thread on the array

// Doesn't perform any bounds checks and behaviour is

// unpredictable in case of index > N

Thread& operator[](int index) {

return \*thread[index];

};

};

#endif