Assignment 1: Basic use of Java threads

I. Introduction

Concurrency means more than one thing happening at the same time, something normal in our everyday life, which is full of examples where we do more than one task at same time.

The programs we have written so far been performed one task at a time, but there are many applications where a program needs to do several things at the same time, or concurrently. For example, an Internet chat program that would allow multiple users to participate in a discussion group. The program should read messages from multiple users simultaneously and transmit them to all participants in the group. Read and sending tasks should be conducted simultaneously.

In Java, concurrent programming can be implemented by using object Thread.

II. What is a Thread?

Processes and threads:

http://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

A thread is a single sequence of instructions within a program. In Java applications, the control flow starts at the first instruction in the *main()* method and continues sequentially.

A single thread has a beginning, an end and a sequence. However, a thread itself is not a program. It cannot run alone but within one program. In fact a sequential program has one single thread of execution. But the real game is the possibility of one program running multiple threads at once to perform different tasks.

Some textbooks use term *light weight process* (LWP) instead of thread. They are described as being *light* because they run within the context of a comprehensive program (process) and take advantage of the resources allocated to this process. Although a thread must get some of its own resources within a program execution, (e.g.: execution stack and program counter) it shares memory allocated for a process with the other threads of the same process.

III. Creating Java threads

Please refer to Section 1.8 in Unit 1 for a review of the 2 ways to create and run Java threads.

IV. Solve the following tasks:

A. Get the reference of the main thread

A sequential program has one single thread called the *main thread*. We can capture this thread by using a static method of class Thread:

Thread h = **Thread.currentThread ();**

keeping the reference to the main thread, we can use public methods of class Thread on it:

- Write a sequential program which prints the first 50 odd numbers.
- Catch the main thread and print its name, then change that name and display it next to each odd number, making the thread sleep for 2 seconds at each iteration.

B. Several threads of type Thread

Create class T_*PrintID* which extends *Thread*:

- It must contain a variable ID whose value is first set through the class constructor.
- Method *run()* has a loop which iterates 10 times displaying ID.

Create class *Numbers* which instantiates 5 T_PrintID objects (with ID 1 to 5) and runs them.

Can you detect some problem? Can you explain the reason?

C. Several Runnable threads

Modify the previous program so that class T_PrintID is created implementing *Runnable* (call the new class R_PrintID). And now the loop sleeps the thread a random time between 0 and 1.

D. Waiting (joining) and interrupting

If a thread wants to wait for another thread to finish its execution, the former needs to call method otherthreadname**.join().** Use it without time limit.

If a thread T1 wants to interrupt another thread T2, the former will set the *interrupted* flag on the latter by calling public method ***interrupt()*** on it. T2 will check periodically its interrupted flag, calling static method **Thread.interrupted()**) method to know if some process wants it to finish. If the flag is set, then it is up to T2 to decide what to do: going on, or free resources, close buffers,… and then finish leaving the system in a stable state.

That said, you must write a Java program which uses 2 threads: the main thread (Hurry) and a new one (Lazy).

Hurry must do the following:

- Creates the Lazy thread.
- Waits for Lazy to finish. Each second it shows the message: "**Aren't you ready yet?"** and checks if Lazy is finished.
- If Lazy isn't finished after 5 seconds, Hurry will claim "**You are resting in your laurels… and I am leaving!**" and interrupts Lazy. Then it waits for Lazy to finish.
- However, if Lazy finishes before the 5 seconds deadline, Hurry says "**At last, a turtle runs rings round you!".**

Lazy must:

- Enter in a loop which iterates a random number of times between 2 to 8.
- At each iteration, it displays a random message among: "**I am dressing up…",** "**Just a sec, please…",** or "**These clothes do not suit me…**".
- If Hurry interrupts Lazy before it is finished, Lazy claims: "**That's not cricket, please play the game!**".
- If Lazy finishes before being interrupted, it says "**I am ready**, **the early bird catches the worm!**".

V. Evaluation

We will use **2 lab sessions** to solve this assignment. Please upload your final code through Campus Virtual.

Create a new package for each task and use appropriate names, such as "Ass1A", "Ass1B",…

Remember you will be evaluated using this rubric:

Respect to the global interests of the group	The group is not well organized. Members did not share the work to carry on. ***0 points***		Some problems arised, and many were solved. ***1 points***	No problems have affected the performance of the group, or were successfully solved, ***2 points***
Correctness of the assignment	It was not solved correctly at all. ***0 points***	Only a few points were right ***1 points***	Mostly correct ***3 points***	Perfect ***4 points***
Correctness of answers to questions about the code (Algorithms and structures used)	No answers/wrong answers ***0 points***	A few correct, or correct but only member answered ***1 points***	Both members gave mostly correct answers. ***2 points***	Both members answered correctly to all questions. ***4 points***