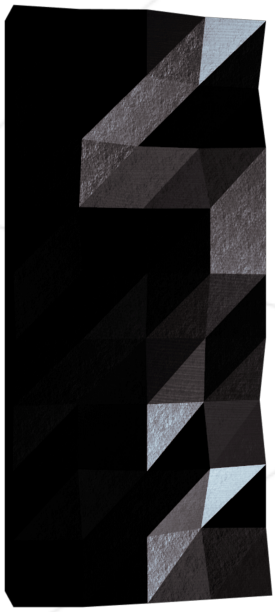

Java - Module 03

Threads

Summary:
In this module you will learn how to use basic multithreading mechanisms in Java.

Version: 1.00

Contents

I	**General Rules**	**2**
II	**Exercise 00: Egg, Hen... or Human?**	**3**
III	**Exercise 01: Egg, Hen, Egg, Hen...**	**5**
IV	**Exercise 02: Real Multithreading**	**6**
V	**Exercise 03: Too Many Threads...**	**8**

Chapter I

General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- You mus use the latest LTS version of Java. Make sure that compiler and interpreter of this version are installed on your machine.
- You must use both JVM and GraalVM to run your code.
- You can use IDE to write and debug the source code (we recommend IntelliJ Idea).
- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each exercise, make sure you follow the generally accepted Oracle standards
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output
- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter II

Exercise 00: Egg, Hen... or Human?

	Exercise 00
Egg, Hen... or Human?	
Turn-in directory : $ex00/$	
Files to turn in : `*.java`	
Allowed functions : `All` `Recommended types and their methods : Object, Thread, Runnable`	

The truth is born in a dispute—let's assume that each thread provides its own answer. The thread that has the last word is right. You need to implement the operation of two threads. Each of them must display its answer a few times, for example, 50:

```
$> java Program --count=50
Egg
Hen
Hen
Hen
...
Egg
$>
```

In this case, the egg thread wins. However, the program also contains main thread. Inside the thread, `public static void main(String args[])` method is executed. We need this thread to display all its responses at the end of program execution. Thus, the ultimate variant is as follows:

```
$> java Program --count=50
Egg
Hen
Hen
...
Egg
Hen
...
Human
...
...
Human
$>
```

It means that the program outputs Human message 50 times, which main thread prints.

Try to explore all ways of creating threads, it would be better if you solve this task using each approach.

Chapter III

Exercise 01: Egg, Hen, Egg, Hen...

	Exercise 01
Egg, Hen, Egg, Hen...	
Turn-in directory : *ex*01/	
Files to turn in : `*.java`	

Allowed functions : `All`
`Recommended types and their methods : Object, Thread, Runnable`
`Keywords : Synchronized`	

Let's orchestrate the argument. Now, each thread can provide its answer only after another thread has done so. Let's assume that the egg thread always answers first.

```
$> java Program --count=50
Egg
Hen
Egg
Hen
Egg
Hen
...
$>
```

To solve this task, we recommend to explore Producer-Consumer model operation principle

Chapter IV

Exercise 02: Real Multithreading

	Exercise 02
Real Multithreading	
Turn-in directory : $ex02/$	
Files to turn in : `*.java`	

Allowed functions : `All`
`Recommended types and their methods : Object, Thread, Runnable`
`Keywords : Synchronized`	

Try to use multithreading for its intended purpose: distribute computations across the program.

Let's assume there is an array of integer values. Your goal is to calculate the sum of array elements using several "summing" threads. Each thread computes a certain section inside the array. The number of elements in each section is constant, except for the last one (its size can differ upward or downward).

The array shall be randomly generated each time. Array length and the number of threads are passed as command-line arguments.

To make sure the program operates correctly, we should start by calculating the sum of array elements using a standard method.

Maximum number of array elements is 2,000,000. Maximum number of threads is no greater than current number of array elements. Maximum modulo value of each array element is 1,000. All data is guaranteed to be valid.

Example of the program operation (each array element equals 1):

```
$> java Program --arraySize=13 --threadsCount=3
Sum: 13
Thread 1: from 0 to 4 sum is 5
Thread 2: from 5 to 9 sum is 5
Thread 3: from 10 to 12 sum is 3
Sum by threads: 13
$>
```

In the above example, the size of the last summing-up section used by the third thread is less than others.

Threads can output the results of operation inconsistently.

Chapter V

Exercise 03: Too Many Threads...

	Exercise 03
Too Many Threads...	
Turn-in directory : $ex03/$	
Files to turn in : `*.java`	

Allowed functions : `All`
`Recommended types and their methods : Object, Thread, Runnable`
`Keywords : Synchronized`	

Let's assume that we need to download a list of files from a network. Some files are downloaded faster, while others are slower.

To implement this functionality, we can obviously use multithreaded downloading where each thread loads a specific file. But what should we do if there are too many files? A large number of threads cannot be run at the same time. Therefore, many of them will be waiting.

In addition, we should bear in mind that continuously creating and completing threads is a very costly operation we should avoid. It makes more sense to start N threads at once and, when either of them finishes downloading the file, it can take on the next file in the queue.

We need to create `files_urls.txt` file (file name shall be explicitly specified in program code) where you specify a list of URLs of files to be downloaded, for instance:

```
$> cat files_urls.txt
https://i.pinimg.com/originals/11/19/2e/11192eba63f6f3aa591d3263fdb66bd5.jpg
https://pluspng.com/img-png/balloon-hd-png-balloons-png-hd-2750.png
https://i.pinimg.com/originals/db/a1/62/dba162603c71cac00d3548420c52bac6.png
https://pngimg.com/uploads/balloon/balloon\_PNG4969.png
http://tldp.org/LDP/intro-linux/intro-linux.pdf
$>
```

Example of program operation:

```
$> java Program.java --threadsCount=3
Thread-1 start download file number 1
Thread-2 start download file number 2
Thread-1 finish download file number 1
Thread-1 start download file number 3
Thread-3 start download file number 4
Thread-1 finish download file number 3
Thread-2 finish download file number 2
Thread-1 start download file number 5
Thread-3 finish download file number 4
Thread-1 finish download file number 5
$>
```

Output created by the implemented program may differ from the illustration.

Each file is downloaded only once by a single thread.

The program may contain an "infinite loop" without the exit condition (in this case, the program can be shut down by interrupting the process).