Java Material:

**Key Concepts for Concurrency:** Threads, Synchronization andRace Conditions

**Recommended:** Java 1.8, NetBeans IDE

**Reading Material 1:**

Read and make notes of the sections: “Processes and Threads”, and “Thread Objects” in the following link [Concurrency](http://download.oracle.com/javase/tutorial/essential/concurrency/index.html)

* What is the difference between Process and Thread?
* What makes a Thread Object different from a regular Thread?

**During Class:**

* Some example of Threads, brief talk about java.

(**show** HelloRunnable, HelloThread, SleepMessages, JoinOther)

* What really happens with the java compiler?
* Concurrent programing
* Parallel programing
* What are the difference between both of them?
* Provide an example of an application of concurrent programing.
* Provide an example of an application of parallel programing.
* What are the problems caused by concurrent programing?
* What are the problems caused by parallel programing?
* Implements vs extends <http://stackoverflow.com/questions/10839131/implements-vs-extends-when-to-use-whats-the-difference>

**Implement together:** simpleThreads class

**Lab 1:**

After doing this lab you should understand how threads work, and how they can communicate with each other.

Crete a separate file for each exercise.

* Use threads to implement a race between the rabbit and the turtle.
  + The rabbit is fast but is also lazy. This means that the rabbit will run 500 to 1000 **(random)** feet and then rest for a while for 100 or 200 mili-seconds.
  + The turtle on the other hand Is always walking at a slow pace between 3 to 10 **(random)** feet. But the turtle never sleeps.
  + Whenever either of them reaches the goal, the other one must stop.
  + Implement several versions of them where you change:
    - The length of the race.
    - The sleeping time of the rabbit
    - The speed of the turtle.
  + Call the different versions from the main class and put the necessary outputs that will enable the user to see how each race behaves.
  + <https://www.facebook.com/animalkindstories/videos/551806765177508/>

**Reading Material 2:**

Read and make notes of the sections: “Synchronization” and “Liveness” in the following link [Synchronization](http://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html)

* What would happen if there was no synchronization?
* What would happen if the whole program was synchronized?

**During Class:**

* How do other parallel programs work?
* (**show** Counter, Synchronized Counter)
* How do they handle these type of errors?
* Example of error of memory consistency
* Example of error with a structure
* Java example of **threadSafe** Structures [ConcurrentHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html) vs [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html).
* Structures that are safe to use (already synchronized)

**(compare:** SynchronizedRGB **and** ImmutableRGB**)**

**(implement:** SynchExamples**)**

**Lab 2**

After doing this lab you should understand the problems that might arise from working with several threads, in particular race conditions, and how to avoid them.

* Mexico City’s botanical garden is open to the public. Any person can go in through one of two doors. East door and West door. Management wishes to know how many people are inside the botanical garden at any given time. Consider that people can go in and out through either door at any given time. At the end of the day we expect the people to leave the garden and we must compare the number of people leaving with the number of people that entered the garden to check for oddities.

Create a program that represents each door with a thread and provide an accurate count of the people inside the garden in real time. Use simulation to test the number of people that go in and out of the garden.

* + The program should have more than one thread and they should all work on the same object. Your threads should receive the number of people that will going in through each door during the day.
  + The people should stay an number of random milliseconds in the garden and the leave through either door.
  + Implement the necessary outputs so we know:
    - how many people are in the garden at any given time,
    - through which door the come in through which door they go out and
    - how many are left when the garden closes (this last one should be 0)

**Reading Material 3:**

Read and make notes of the following links: [Executors](http://docs.oracle.com/javase/tutorial/essential/concurrency/executors.html) and [Fork Join](http://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html)

* What are these structures and why are they useful?
* Provide two examples of programs you have made which might be improved using “Fork Join”, how would they improve?

**During Class:**

* Example of parallel computing
* **(Implement sequentialPi, ParallelPi)**
* Recursion
* Benchmark of parallel <https://togototo.wordpress.com/2013/08/23/benchmarks-round-two-parallel-go-rust-d-scala-and-nimrod/>
* Example of Fork Join Recursive Action
* Example of Fork Join Recursive Task
* Throughput
* Benchmark fork/join <http://blog.takipi.com/forkjoin-framework-vs-parallel-streams-vs-executorservice-the-ultimate-benchmark/>

**Lab 3:**

After this lab you should understand the gains of time that can be obtained from parallelism in non-sequential tasks.

* The file Gray.java changes a regular image to grey scale, it does so using a single thread. Use this file to generate your own version of the program which parallelizes the task of turning the image into grey scale. The task must run faster than the sequential version.
* Compare both times the sequential version vs the parallel version.
* Time only the segment of code that do the conversion, not the loading and writing of the image.
* Output images must be in grey scale or apply some filter that changes the whole image in a correct way.
* You can use these images to test your program [Owl.jpg](file:///C:\Users\NG\Documents\Tec\Cursos%20Impartidos\Lenguajes%20de%20Programacion\Dise%C3%B1o%202016\Black%20Board\calendar\imagenes\Owl.jpg),[zelda.jpg](file:///C:\Users\NG\Documents\Tec\Cursos%20Impartidos\Lenguajes%20de%20Programacion\Dise%C3%B1o%202016\Black%20Board\calendar\imagenes\zelda.jpg), [natalie.png](file:///C:\Users\NG\Documents\Tec\Cursos%20Impartidos\Lenguajes%20de%20Programacion\Dise%C3%B1o%202016\Black%20Board\calendar\imagenes\natalie.png)

**Ideas for project:**

**Parallel:** Make one of your previously developed apps run in parallel to improve performance.

**Concurrence:** Create an online server that handles real queries in a parallel way, such that it can handle several requests at the same time.

**Concurrence:** Create a chat service.

Use the ideas seen in this paradigm to implement a concurrent o parallel application in a language of your choice.

**Example of commented code for labs:**

**/\*\***

**Benjamin Valdes Aguirre A00882900**

**Java example of PI**

**9 / 10 /2019**

**\*/**

**package classExamples;**

**/\***

**Calculate PI by area aproximation, using multiple processors**

**\*/**

**public class PPi implements Runnable{**

**long inicio;**

**long fin;**

**double area;**

**double width;**

**public PPi(long ini, long fi, double width){**

**inicio = ini;**

**fin = fi;**

**this.width = width;**

**}**

**@Override**

**public void run() {**

**double mid, height;**

**double sum = 0;**

**for(long i = inicio; i < fin; i++){**

**mid = (i + 0.5)\*width;**

**height = 4.0/(1.0+mid\*mid);**

**sum+=height;**

**}**

**this.area = sum;**

**}**

**public static void main(String args[]) throws InterruptedException{**

**//setup for the computation**

**long num= 1000\_000\_000;**

**double width = 1.0/(double)num;**

**int p = Runtime.getRuntime().availableProcessors();**

**System.out.println("number of cores " + p);**

**//create the Runnable objects splitting the calculation in 2**

**PPi ppi1= new PPi(0, num/2, width);**

**PPi ppi2= new PPi(num/2,num, width);**

**//assing each object to a thread**

**Thread th1 = new Thread(ppi1);**

**Thread th2 = new Thread(ppi2);**

**//start the threads**

**th1.start();**

**th2.start();**

**//wait for threads to end**

**th1.join();**

**th2.join();**

**// add results of each thread and print**

**double pi\_value = (ppi1.area + ppi2.area)\*width;**

**System.out.println("ppi1 value is "+ppi1.area);**

**System.out.println("ppi2 value is "+ppi2.area);**

**System.out.println("total pi is "+pi\_value);**

**}**

**}**