TITLE
Laboratory 3. Concurrent Computation II

PREREQUISITES

- OOP basic knowledge
- Java programming language basic knowledge
- Algorithms

RESOURCES

- Course slides
- Java Tutorial
- Java Concurrency

LABORATORY INSTRUCTIONS

Thread scheduler in java is the part of the Java Virtual Machine (JVM) that decides which thread should run.
An important thing to know is that only one thread can run on a single process at a time and there is no guarantee on which runnable thread will be chosen to run by the thread scheduler. The thread scheduler mainly uses two approaches when it comes to selecting a thread from the pool:

- **Preemptive scheduling** - highest priority task executes until it enters the waiting or dead states or a higher priority task is created and starts running.
- **Time slice scheduling** - a thread is running for a predefined period of time and then reenters the pool of ready threads, then the thread scheduler selects the thread that should execute next based on priority and other factors.

!!! DO NOT USE the *stop()*, *resume()* and *suspend()* deprecated methods to stop/resume a running thread !!!

The sleep() method

There are two sleep methods in Java for a Thread object:

- public static void **sleep(long milliseconds)** throws **InterruptedException**
- public static void **sleep(long milliseconds, int nanos)** throws **InterruptedException**

Try the example code below. What is the output of the program?

```
class MyThread extends Thread{
 public void run(){
  for(int i = 1; i < 10; i++){
    try{Thread.sleep(1000);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  MyThread th1 = new MyThread();
  MyThread th2 = new MyThread();

  t1.start();
  t2.start();
 }
}
```

The join() method

Allows one thread to wait until another thread completes its execution.

- public final void **join()** throws **InterruptedException**
- public final void **join(long milliseconds, int nanos)** throws **InterruptedException**
- public final void **join(long milliseconds)** throws **InterruptedException**

```
class ThreadJoin extends Thread{
 public void run(){
  for(int i =1;i <= 5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 ThreadJoin t1 = new ThreadJoin ();
 ThreadJoin t2 = new ThreadJoin ();
 ThreadJoin t3 = new ThreadJoin ();
```

```
 t1.start();
 try{
  t1.join();
 }catch(Exception e){System.out.println(e);}
 t2.start();
 t3.start();
 }
}
```

LABORATORY TASKS

1. Create a new Java project and then create a MyThread class. In the main() method try to start the instance of that thread twice. What is the output?
2. Using the project above, what happens if we call the run() method instead of the start method()?
3. Implement the *NumberCanvas* example from Course 1. What do you notice?
4. Implement the prime numbers exercise (exercise 1, from Homework section in Course 1). What is the difference between the 2 implementations requested?
5. There is a horse track where 20 horses are scheduled to start at the same time. Imagine each horse as being a separate instance of HorseThread:
   - The main program will wait until all horses will be finalized
   - Each horse makes a move, prints its move, and then sleeps for 10 ms

   Which horse will win after making 20 moves? What if you increase the sleep period? How about if you run the program multiple times or more than 20 moves? Do you observe any difference?
6. If you check out the Thread class on Java's web page, what other methods do you observe in addition to:
   - Start, run, join, sleep, suspend, resume?
   - Anything noticeable? What about wait and notify/notifyAll?