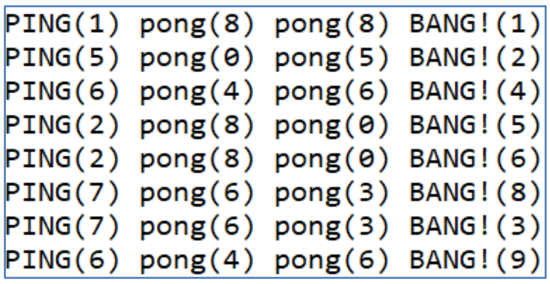
PARALLELISM AND CONCURRENCY
PRACTICAL ASSIGNMENT #1 WINTER TERM 2022
JAVA THREADS AND SYNCHRONIZATION

Introduction: The "PING pong^2 BANG!" problem

The "PING pong^2 BANG!" (aka PING double pong BANG!) problem is similar to other synchronization problems that you already know. In its easiest version, several threads compete to write PING, pong and BANG! The objects of Ping endlessly write the string "PING"; the objects of Pong endlessly write the string "pong" while the instances of Bang endlessly print "BANG!\n" The main goal is to give the threads a synchronization mechanism that strictly enforces an ordered writing so that the global effect is that the they endlessly print **`PING pong pong BANG!`** It is important to know that the Pong instances can only write "pong" once (they cannot write "pong pong" or iterate two times the writing of a single "pong", etc.)

```
Pong_i
loop
        pre-protocol-like operation(s)
        write "pong"
        post-protocol-like operation(s)
```

Pseudocode of the Pong process



```
PING(1) pong(8) pong(8) BANG!(1)
PING(5) pong(0) pong(5) BANG!(2)
PING(6) pong(4) pong(6) BANG!(4)
PING(2) pong(8) pong(0) BANG!(5)
PING(2) pong(8) pong(0) BANG!(6)
PING(7) pong(6) pong(3) BANG!(8)
PING(7) pong(6) pong(3) BANG!(3)
PING(6) pong(4) pong(6) BANG!(9)
```

Sample output of a well-synchronized "PING pong^2 BANG!"

Exercise 1: The Explicit Lock version [10%]

In this version of the problem, the actors share a single instance of an object that acts as a synchronizer. This object encapsulates a single instance of a fair `ReentrantLock` and as many instances of simple-typed variables as you deem necessary.

Exercise 2: The Compare & Set version [10%]

In this version of the problem, the actors share a single instance of an object that acts as a synchronizer. This object encapsulates a single instance of an `AtomicBoolean` and as many instances of simple-typed variables as you deem necessary (HINT: except for the fact that boolean TS-variables have no associated waiting-set, they can be made behave as if they were locks: the thread that succeeds in changing its value becomes "the owner")

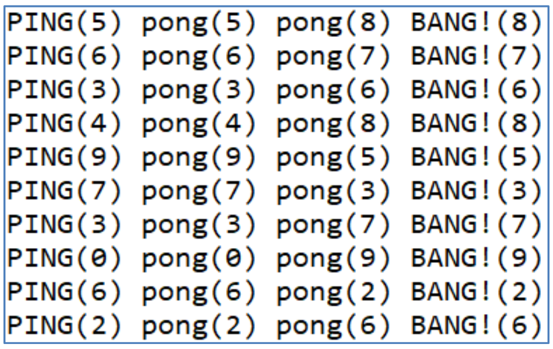
Exercise 3: The three binary SEMAPHORES version (10%)

Now your solution must be based **on three binary semaphores** (and as many simple-typed variables as you need). Notice that Java semaphores are not binary but general ones (counting semaphores). You will have to use them in such a way that never a semaphore holds more than one permit.

Exercise 4: The Implicit Lock version) (15%)

Now your solution must be based on the implicit lock contained in the synchronizer object. Also, the following restriction must be enforced

> In each line, the first pong must be written by a thread with the same ID that the thread that wrote ping; the second pong cannot be written by the same thread that wrote the previous one and the bang! must be written by a thread with the same ID that the thread that wrote the second pong.



```
PING(5) pong(5) pong(8) BANG!(8)
PING(6) pong(6) pong(7) BANG!(7)
PING(3) pong(3) pong(6) BANG!(6)
PING(4) pong(4) pong(8) BANG!(8)
PING(9) pong(9) pong(5) BANG!(5)
PING(7) pong(7) pong(3) BANG!(3)
PING(3) pong(3) pong(7) BANG!(7)
PING(0) pong(0) pong(9) BANG!(9)
PING(6) pong(6) pong(2) BANG!(2)
PING(2) pong(2) pong(6) BANG!(6)
```

Sample output produced by this version

Exercise 5: The SERIAL BANG-KILLER and a Single Semaphore version [30%]

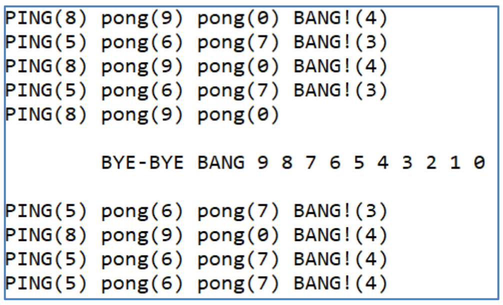
Using a single semaphore (and as many single-typed variables as you deem necessary) your solution must:

a) Enforce the following restriction

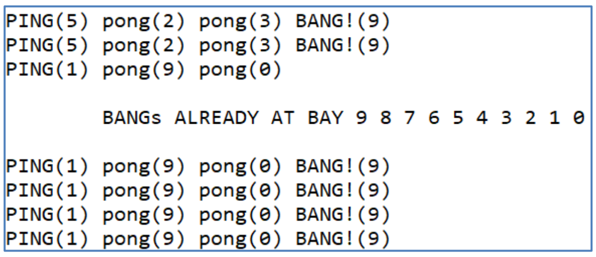
> Pongs in the same line must be written by threads with consecutive ids.

b) Add to the Bang class a `syncStop` method that, when invoked, makes the bang thread stop without causing any trouble to the other threads executing.

c) Include a new "actor": a serial bang-killer. A serial bang-killer is a thread that every half a second gets ready to kill a bang thread invoking its `syncStop` method and writes BYE-BYE BANG and a countdown from 9 to 0 (with a delay, between numbers, of 100 ms). When the bang killer has murdered all bangs but one it just writes BANGs ALREADY AT BAY (also every half a second and also with a countdown from 9 to 0). The serial bang-killer can only kill and write after a second pong written by a thread with id 0 (its accomplice…). Right after the murder, no bang is written. Instead, a new line is started.



```
PING(8) pong(9) pong(0) BANG!(4)
PING(5) pong(6) pong(7) BANG!(3)
PING(8) pong(9) pong(0) BANG!(4)
PING(5) pong(6) pong(7) BANG!(3)
PING(8) pong(9) pong(0)

        BYE-BYE BANG 9 8 7 6 5 4 3 2 1 0

PING(5) pong(6) pong(7) BANG!(3)
PING(8) pong(9) pong(0) BANG!(4)
PING(5) pong(6) pong(7) BANG!(4)
PING(5) pong(6) pong(7) BANG!(4)
```



```
PING(5) pong(2) pong(3) BANG!(9)
PING(5) pong(2) pong(3) BANG!(9)
PING(1) pong(9) pong(0)

        BANGs ALREADY AT BAY 9 8 7 6 5 4 3 2 1 0

PING(1) pong(9) pong(0) BANG!(9)
PING(1) pong(9) pong(0) BANG!(9)
PING(1) pong(9) pong(0) BANG!(9)
PING(1) pong(9) pong(0) BANG!(9)
```

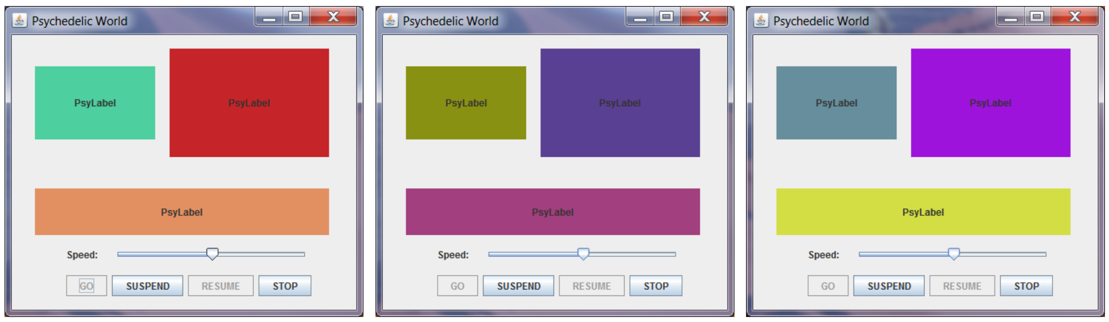
Sample outputs produced by this version. After killing a bang (left) and after all bangs but one have been killed (right).

Exercise 6: A GUI-BASED “DIVERTIMENTO”. PSYCHEDELIA [25%]

A psychedelicLabel is an object from a subclass of JLabel that endlessly changes its background colour.

```java
package Psychedelia;

public class PsychedelicLabel extends JLabel implements Runnable {

   private int speed = 50;
   private Random alea;
   private Color color;

   public PsychedelicLabel () {
      this.setOpaque(true);
      this.alea = new Random();
      this.setText("PsyLabel");
   }

   public void setSpeed (int speed) {this.speed=speed;}


   public void run () {

          /* Endlessly generate a random colour and give it to the JLabel
            (change its background). Sleep a time inversely proportional to speed */
   }
}
```

Write a PsychedelicWorld program that displays three or more psychedelic labels. Provide a JSlider object to change the speed of change. Also provide suspend, resume and stop buttons.



WARNING: fixedly looking at active psychedelic objects may have undesirable effects on the watcher (e.g. they can make you feel sick...)

SUBMISSION

Due date: Check eCampus

Instructions:

Solve the exercises in the given project and then rename the folder that contains it. Its name must be
PA1_Surname1Surname2_Surname1Surname2_Surname1Surname2

Compress the folder and upload it.