Programación Concurrente -2018
Trabajo Práctico Concurrencia N° 3

1.- Analice el siguiente código:

```
public class DualSynch {
        private Object syncObject = new Object();

        public synchronized void f() {
                for(int i = 0; i < 5; i++) {
                        print("f()");
                        Thread.yield();
                }
        }

        public void g() {
                synchronized(syncObject) {
                        for(int i = 0; i < 5; i++) {
                                print("g()");
                                Thread.yield();
                        }
                }
        }
}



public class SyncObject {
        public static void main(String[] args) {
                final DualSynch ds = new DualSynch();

// solo por cuestiones prácticas se trabaja de esta forma
                Thread hilo = new Thread() {
                                public void run() {
                                        ds.f();
                                }
                          }
                hilo.start();
                ds.g();
        }
}
```

1.- ¿cual es el efecto de Thread.yield()?

2.- ¿cual es el efecto de "synchronized (syncObject)"?

3.- ¿cual es el efecto de "synchronized void f()"?

4.- ¿cual es la diferencia entre el yield() y sleep()?

5.- Indique el funcionamiento general de lo presentado

2.- Considere la clase DualSynch modificada, ahora con un recurso compartido por los hilos

```
public class DualSynch {

    private Object syncObject = new Object();
    int dato=5;

    public synchronized void f() {
        for(int i = 0; i < 5; i++) {
            dato = dato * 4;
            print("f()" + dato);
            Thread.yield();
        }
    }

    public void g() {
      synchronized(syncObject) {
        for(int i = 0; i < 5; i++) {
            dato = dato + 20;
            print("g()" + dato);
            Thread.yield();
        }
      }
    }
}


public class SyncObject {
    public static void main(String[] args) {
        final DualSynch ds = new DualSynch();
        Thread hilo = new Thread() {
                    public void run() {
                        ds.f();
                    }
                }
        hilo.start();
        ds.g();
    }
}
```

1. cuantos hilos hay en ejecución
2. cual es el recurso compartido

3. la salida, es siempre la esperada?
4. es posible obtener la siguiente salida?
   g(), 25
   g(), 120
   g(), 140
   g(), 160
   f(), 160
   f(), 640
   f(), 2560
   f(), 10240
   f(), 40960
   g(), 40980

3.- Investigue la Interfez *Lock*, y su implementación *ReentrantLock* respecto a su uso para lograr la exclusión mútua. Considere sólo los métodos básicos necesarios

4.- Resuelva el ejercicio 6 del tp2 utilizando Locks como herramienta de sincronización

5.- Dado el siguiente código

a.
```
public class SynchronizedCounter {

        private int c = 0;

        public synchronized void increment() {c++;}

        public  void decrement() {c--;}

        public synchronized int value() {return c;}

    }
```

b.
```
public class SynchronizedObjectCounter {

        private int c = 0;

        public  void increment()
                synchronized (c) {
                        c++;
                }

            }
        public  void decrement() {
                synchronized (this) {
                        c--;
                }
            }

        public int value() {return c;}
```

```
}
```

- Verifique el funcionamiento del codigo del inciso a)
- Verifique el funcionamiento del codigo del inciso b)
- En caso de ser necesario realice las correcciones que crea convenientes.
- Compare entre usar, en este caso, métodos sincrozados con objetos sincronizados. (Ventajas y Desventajas)

6.- Considere un sistema formado por tres hilos fumadores que se pasan el día armando cigarrillos y fumando. Para armar y fumar un cigarrillo necesitan tres ingredientes: tabaco, papel y fósforos. Cada fumador dispone de un surtido suficiente (para el resto de su vida) de uno de los tres ingredientes. Cada fumador tiene un ingrediente diferente, es decir, un fumador tiene una cantidad infinita de tabaco, el otro de papel y el otro de fósforos. Hay también un hilo *agente* que pone dos de los tres ingredientes encima de una mesa. El agente dispone de unas reservas infinitas de cada uno de los tres ingredientes y escoge de forma aleatoria cuáles son los ingredientes que pondrá encima de la mesa. Cuando los ha puesto, el fumador que tiene el otro ingrediente puede armar su cigarrillo y fumar (los otros dos no). Para ello toma los ingredientes, se arma un cigarrillo y se lo fuma. Cuando termina de fumar vuelve a repetirse el ciclo. En resumen, el ciclo que debe repetirse es :

"agente pone ingredientes $\rightarrow$ fumador hace cigarro $\rightarrow$ fumador fuma $\rightarrow$ fumador termina de fumar $\rightarrow$ agente pone ingredientes $\rightarrow$ ..."

Es decir, en cada momento a lo sumo hay un fumador fumando un cigarrillo.

Considere el código siguiente e implemente la clase SalaFumadores, como recurso compartido entre fumadores y agente

```
public class Fumador implements Runnable{
 private int id;
 private SalaFumadores sala;
 public Fumador(int id, SalaFumadores sala){
  this.id = id;
  this.sala = sala;
 }
 public void run(){
   while(true){
    try {
      sala.entrafumar(id);
      System.out.println("Fumador "+id+" está fumando.");
      Thread.sleep(1000);
      sala.terminafumar();
    } catch (InterruptedException e) { e.printStackTrace(); }
  }
}

}


public class Agente implements Runnable {
 private SalaFumadores sala;
 private Random r;
 public Agente(SalaFumadores sala){
```

```
   this.sala = sala;
   r= new Random();
 }
 public void run (){
  while(true){
   sala.colocar(r.nextInt(3)+1);
  }
 }
}


public class DisparaSala {

    /**
     * @param args the command line arguments
     */
 public static void main(String[] args) {
   SalaFumadores sala = new SalaFumadores();
   Fumador f1 = new Fumador(1, sala);
   Fumador f2 = new Fumador(2, sala);
   Fumador f3 = new Fumador(3, sala);
   Agente ag  = new Agente(sala);

   Thread fumador1 = new Thread(f1);
   Thread fumador2 = new Thread(f2);
   Thread fumador3 = new Thread(f3);
   Thread agente   = new Thread(ag);

   fumador1.start();
   fumador2.start();
   fumador3.start();
   agente.start();
 }
}
```