Programación Concurrente -2018
Trabajo Práctico Concurrencia N° 1

1. Nombrar 5 (cinco) situaciones de la vida cotidiana que usted crea que, en caso de poder ser informatizadas, deberían serlo a través de la programación concurrente.

2. En Java, ejecutar el siguiente código **repetidas** veces:

```
public class Cliente extends Thread {
    public void run(){
        System.out.println(" Soy"+Thread.currentThread().getName());
        Recurso.uso();
        try {
            Thread.sleep(2000);
        }catch (InterruptedException e) {
        };
    };
}
```

```
public class Recurso {
   static void uso(){
        Thread t=Thread.currentThread();
        System.out.println("en Recurso:  Soy" + t.getName());
   }
}

public class testeoRecurso {
    public  static void main (String[] args){
        Cliente juan=new Cliente();
        juan.setName("Juan Lopez");
        Cliente ines=new Cliente ();
        ines.setName ("Ines Garcia");
        juan.start();
        ines.start();
    }
}
```

   a. Analice el funcionamiento del siguiente código
   b. ¿Cúal es la funcionalidad del método "uso" de Recurso?

Programación Concurrente -2018
Trabajo Práctico Concurrencia N° 1

3. Ejecutar el siguiente código en reiteradas oportunidades:

```
public class MiEjecucion extends Thread{
    public void run(){
        ir();
    }

    public void ir(){
        hacerMas();
    }

    public void hacerMas(){
        System.out.println("En la pila");
    }
}

class ThreadTesting{
    public static void main (String[] args){
      Thread miHilo=  new MiEjecucion();
      miHilo.start();
      System.out.println("En el main");
    }
}
```

a. ¿Cómo es el comportamiento de las diferentes ejecuciones?.
b. Se podría forzar las ejecuciones para que se comporte de una manera determinada? Realice las modificaciones pertinentes.

4. Teniendo en cuenta el ejercicio de teoria "Ping-Pong", representar, EJECUTAR Y ANALIZAR las siguientes situaciones:

a. El código sin la sentencia *sleep()*.
b. Dentro de la clase main realizar una iteración de una sentencia cualquiera con un numero grande de veces.
c. Ejecutar el código las veces que sea necesario hasta que la salida se vea modificada.
d. Dentro del bloque de iteraciones del inciso b) agregar una sentencia sleep().
e. De la ejecución de los puntos anteriores ¿Qué puede deducir?

f. Teniendo en cuenta el ejercicio anterior, agregar dos hilos "Pang" y "Pung". Ejecute varias veces y determine en cada momento el proceso que se está ejecutando.

5. Teniendo en cuenta el ejercicio anterior trate de visualizar el rendimiento de la CPU.
   a. En Linux, ir a "aplicaciones/herramientas del sistema/MATE System Monitor. Solapa "Recursos"".
   b. En Windows, ir a Inicio/Herramientas del sistema/Monitor de Recursos.
   c. ¿Por qué cree que varía? ¿En qué casos hay mayor uso de CPU?

6. Supongamos que debemos simular el proceso de cobro de un supermercado; es decir, unos clientes van con un carro lleno de productos y una cajera les cobra los productos, pasándolos uno a uno por el escáner de la caja registradora. En este caso la cajera debe procesar la compra cliente a cliente, es decir que primero le cobra al cliente 1, luego al cliente 2 y así sucesivamente. Para ello se debe definir una clase "Cajera" y una clase "Cliente" el cual tendrá un "array de enteros" que representarán los productos que ha comprado y el tiempo que la cajera tardará en pasar el producto por el escáner; es decir, que si tenemos un array con [1,3,5] significará que el cliente ha comprado 3 productos y que la cajera tardará en procesar el producto 1 '1 segundo', el producto 2 '3 segundos' y el producto 3 '5 segundos', con lo cual el tiempo total empleado por la cajera será de 9 segundos.

   a. El siguiente código simula la operación de cobro con dos Clientes con un solo proceso (que es lo que se suele hacer normalmente), teniendo en cuenta que se procesará primero la compra del Cliente 1 y después la del Cliente 2, con lo cual se tardará el tiempo del Cliente 1 + Cliente 2. Completar y ubicar en la clase que corresponda la implementación del método: *esperarXsegundos*.

```
public class Cajera {
    private String nombre;
    // Agregar Constructor, y métodos de acceso

    public void procesarCompra(Cliente cliente, long timeStamp) {

        System.out.println ("La cajera " + this.nombre +
                    " COMIENZA A PROCESAR LA COMPRA DEL CLIENTE " +
                     cliente.getNombre() + " EN EL TIEMPO: " +
                    (System.currentTimeMillis() - timeStamp) / 1000 + "seg");

        for (int i = 0;  i < cliente.getCarroCompra().length;  i++) {
                this.esperarXsegundos(cliente.getCarroCompra()[i]);
```

Programación Concurrente -2018
Trabajo Práctico Concurrencia N° 1

```
                System.out.println("Procesado el producto " + (i + 1) + " ->Tiempo: " +
                (System.currentTimeMillis() - timeStamp) / 1000 + "seg");
        }

        System.out.println("La cajera " + this.nombre +
        " HA TERMINADO DE PROCESAR " + cliente.getNombre() + " EN EL TIEMPO: " +
                                (System.currentTimeMillis() - timeStamp) / 1000 +
                        "seg");
    }
}
```

```
public class Cliente {
        private String nombre;
        private int[] carroCompra;
        // Constructor y métodos de acceso
}
```

```
public class Main {
    public static void main(String[] args) {
        Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5, 2, 3 });
        Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1, 1 });
        Cajera cajera1 = new Cajera("Cajera 1");
        // Tiempo inicial de referencia
        long initialTime = System.currentTimeMillis();
        cajera1.procesarCompra(cliente1, initialTime);
        cajera1.procesarCompra(cliente2, initialTime);
    }
}
```

Salida:
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO:
0seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg

Programación Concurrente -2018
Trabajo Práctico Concurrencia N° 1

La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO:
15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg

Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
GENERACIÓN CORRECTA (total time: 26 seconds)

b. ¿Y si en vez de procesar primero un cliente y después otro, procesásemos los dos a la vez?, ¿Cuánto tardaría el programa en ejecutarse?. Si en vez de haber solo una Cajera (es decir un solo hilo), hubiese dos Cajeras (es decir dos hilos o threads) podríamos procesar los dos clientes a la vez y tardar menos tiempo en ejecutarse el programa. Complete el siguiente código a fin de representar el escenario descripto.

```
public class CajeraThread extends Thread {
        private String nombre;
        private Cliente cliente;
        private long initialTime;
        // Constructor, y metodos de acceso

        public void run() {
                System.out.println("La cajera " + this.nombre +
                " COMIENZA A PROCESAR LA COMPRA DEL CLIENTE "
                + this.cliente.getNombre() + " EN EL TIEMPO: "
                + (System.currentTimeMillis() - this.initialTime) / 1000 + "seg");

                for (int i = 0;  i < this.cliente.getCarroCompra().length;  i++) {
                        this.esperarXsegundos(cliente.getCarroCompra()[i]);
                        System.out.println("Procesado el producto " + (i + 1) + " del cliente " +
                                        this.cliente.getNombre() + "->Tiempo: " +
                                          (System.currentTimeMillis()  -  this.initialTime)  /  1000  +
                        "seg");
                }

                System.out.println("La  cajera"  +  this.nombre  +  "HA  TERMINADO  DE
                PROCESAR"
```

```
                + this.cliente.getNombre() + " EN EL TIEMPO: " +
                (System.currentTimeMillis() - this.initialTime) / 1000 +
            "seg");
        }
}

public class MainThread {
public static void main(String[] args) {
Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5, 2, 3 });
Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1, 1 });
.....
}  }
```

Posible salida:

run:
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
GENERACIÓN CORRECTA (total time: 15 seconds)

7. Realice los ejercicios 2, 3, y 6b utilizando la *interfaz Runnable*