Programación de Servicios y Procesos. 2019. Primera Evaluación.
Cálculo de números primos

Deseamos calcular todos los números primos que hay entre 0 y una variable [**int** maxProcesos] que va a coincidir con el número máximos de hilos a lanzar. Cada hilo determinará si su número de proceso es primo. Para ello, tenemos un array de hilos (Clase Procesador).

```
Procesador[] procesador = new Procesador[maxProcesos];
```

Cuando todos los hilos se hayan creado se iniciarán concurrentemente (sincronizamos mediante una cyclicbarrier)

Dado que no queremos sobrecargar la CPU, vamos a limitar el número de hilos que se pueden ejecutar concurrentemente mediante un semáforo (Semaphore concurrencia) a un valor, por ejemplo 10 [**int** maxProcesosCpu = 10]

Así, una posible implementación del constructor **Procesador** podría ser:

```
Procesador(nproceso,iniciaProceso,esperaFinProceso,concurrencia,emisor,accesoPipeCounter);
```

Donde:

- **nproceso** = número de proceso
- **maxProcesos** = número máximo de procesos a generar, 1 por cada número que queremos determinar si es primo
- **iniciaProceso** = cyclicbarrier para sincronizar que todos los hilos se inician simultáneamente
- **esperaFinProceso** = countdownlatch para esperar que todos los hilos terminen
- **concurrencia** = semáforo para limitar el número de hilos que se ejecutan concurrentemente y no cargar la CPU, limitado a maxProcesosCpu
- **emisor** = pipedwriter donde los hilos van a enviar los números primos encontrados
- **accesoPipeCounter** = semáforo para controlar el acceso concurrente a la variable global **static int** *datosPipe* de la clase principal CalculaPrimos. Esta variable debe registrar la cantidad de primos encontrada. Los hilos incrementarán *datosPipe* cuando encuentren un primo, pero el acceso estará limitado a un hilo simultáneamente

El funcionamiento de la clase **Procesador** es la siguiente:

1. Espera que todos los hilos estén inicializados
2. En un while(true) intenta entrar en la sección crítica, limitada a maxProcesosCpu, si no lo consigue le haremos esperar un valor aleatorio de 0 a 300ms [**long** aleatorio = rand.nextInt(300)]
3. Una vez tenga acceso a la sección crítica determinará si su número de proceso es primo (ver función boolean esprimo(valor) al final de la hoja)
4. Si es primo lo envía por el pipe emisor e incrementa la variable estática *datosPipe,* controlando el acceso concurrente mediante el semáforo accesoPipeCounter

Simultáneamente a la clase **Procesador**, tendremos otro hilo perteneciente a la clase **OrdenaDatos**. Un ejemplo de constructor sería

```
OrdenaDatos(esperaFinProceso,esperaFinOrdena,receptor);
```

Donde:
esperaFinProceso es el countdownlatch compartido con la clase Procesador
esperaFinOrdena es otro countdownlatch para que la clase main espere a que se ordenen e impriman los primos encontrados
receptor = pipedreader enlazado con el emisor para leer los números primos generados

El funcionamiento de la clase **OrdenaDatos** es el siguiente:

1. Espera que todos los hilos de la clase Procesador finalicen.
2. Lee todos los datos del pipe y los mete en un array. Podemos utilizar un bucle while o for, ya que en la variable estática *datosPipe* de la clase principal CalculaPrimos tenemos el número de primos encontrados (valores = **new int**[CalculaPrimos.*datosPipe*]) y por tanto metidos en el pipe (leemos tantos como metemos)
3. Ordena el array (Arrays.*sort*(valores)). Importamos de java.util.Arrays
4. Imprime los números primos encontrados "Primo i-ésimo > número-primo"
5. Notifica al main de la clase principal **CalculaPrimos** que hemos terminado

Nuestra aplicación constará de la siguientes clases (no son necesarias más clases):

1. **CalculaPrimos**, que es la aplicación principal (tiene el main[]) que inicia todos los hilos y espera que termine todo el proceso, imprimiendo un mensaje de aplicación finalizada
2. **Procesador**, que es la clase que determina si un número es primo y lo envía por el pipe. De ejecuta como hilo, tantos hilos como números a verificar
3. **OrdenaDatos**. Recibe los primos por un pipe, los mete en un array, los ordena e imprime. Se ejecuta como un hilo

Criterios de evaluación:

1. Las clases se ha implementado adecuadamente, realizan la función sugerida y los parámetros de las clases son coherentes con su funcionalidad. (2 puntos)
2. La sincronización de la toda la simulación se ha realizado adecuadamente, utilizando las clases de concurrencia que java proporciona, y ejecutando la sincronización tal y como se ha descrito. (2 puntos)
3. Las secciones críticas se han implementado correctamente, tanto el acceso a la variable global, como el límite de hilos ejecutándose concurrentemente. (2 puntos)
4. La aplicación presenta una lógica coherente y funciona correctamente. (2 puntos)

Entrega del ejercicio

El ejercicio deberá ser entregado en un archivo comprimido con el nombre y apellidos y todos los archivos fuente utilizados.

Seguidamente se muestra la función esprimo(valor) que determina si un número es primo (divisible únicamente entre 1 y el mismo número)

```
private boolean esprimo(int valor) {
            boolean primo = false;
            switch(valor) {
            case 0: primo = false;
                  break;
            case 1: primo = false;
                  break;
            case 2: primo = true;
                  break;
            default:
                  int contador = 0;
                  for(int i = 1; i <= valor; i++)
                  {
                        if((valor % i) == 0)
                        {
                              contador++;
                        }
                  }//end-for
                  if(contador <= 2) {
                        //es primo
                        primo = true;
                  }
                  else {
                        primo = false;
                  }//end-if-else
            }//end-case
            return primo;
      }//end-esprimo
```