

Apellidos:

Nombre:

La duración máxima del examen será de 2:30 horas.

Ejercicio 1) Test (3p)

- Sólo una respuesta es correcta.
- Las preguntas contestadas correctamente sumarán 0,6 puntos y las que se contesten de forma errónea restarán 0,2 puntos. Con estas puntuaciones en el test se obtendrá una nota sobre 3 puntos.
- Marque con una X las respuestas a las preguntas del test en la tabla que se encuentra debajo.
- Sólo hay una respuesta correcta.

Respuestas:

	1	2	3	4	5
a					
b					
c					

Preguntas:

P1.- Los atributos volatile en Java...

- a) ... son aquellos atributos de tipos primitivos (int, boolean...) compartidos entre varios hilos de ejecución. No es posible compartir un atributo en Java a menos que sea marcado como volatile.
- b) ... son atributos que no se suelen usar de forma habitual en Java porque cuando se usan herramientas de sincronización de procesos no son necesarios.
- c) ... deben usarse siempre en un programa concurrente para evitar que el compilador o la JVM los elimine en el proceso de optimización del código.

P2.- Cuando se necesita finalizar la ejecución de un hilo antes de que termine por si mismo...

- a) Se debe utilizar el método stop() de la clase Thread. Esto asegura de que el hilo finaliza de forma inmediata sin esperas para el usuario.
- b) Lo más habitual es utilizar las herramientas de sincronización como Semáforos, Monitores o las herramientas de sincronización condicional de Java para la finalización de procesos.
- c) Es aconsejable utilizar las interrupciones, que pueden ser consultadas por los hilos que las reciben o desbloquear con una excepción la mayoría de los métodos bloqueantes.

P3.- Respecto a las activaciones inesperadas en Java:

- a) Como las JVM se implementan usando primitivas del sistema operativo y en los sistemas POSIX se pueden producir desbloques inesperados, entonces todos los métodos bloqueantes en Java sufren de este problema (Semáforos, Monitores, Colecciones bloqueantes, etc...)
- b) Incomprensiblemente, afectan únicamente a los monitores en Java, pero no al resto de herramientas como Exchanger, CountdownLatch, etc...
- c) Ya no se producen en Java debido a los avances en las herramientas de sincronización de Java.

P4.- La palabra reservada synchronized ...

- a) ... permite especificar el cerrojo (lock) que controla la exclusión mutua.
- b) ... se pone en los atributos que van a ser compartidos entre varios hilos.
- c) ... sólo permite delimitar una única sección crítica para la exclusión mutua. Por eso se desarrollaron los Locks.

P5.- En relación a la implementación de programas concurrentes en un lenguaje de programación:

- a) Cualquier lenguaje de programación permite el desarrollo de aplicaciones concurrentes, basta con encontrar la librería adecuada.
- b) No todos los lenguajes de programación soportan la programación concurrente, pero los que lo hacen, siempre tienen una sintaxis específica para los aspectos concurrentes (sincronización, compartición de variables, etc...)
- c) En algunos lenguajes de programación el desarrollo de aplicaciones concurrentes depende en gran medida del sistema operativo utilizado.

Ejercicio 2) Preguntas Cortas (3p)

Pregunta 1: Describe en qué se parecen y diferencian la sincronización con *synchronized* y con objetos *Lock*. ¿Cuándo usar una u otra?

Pregunta 2: Describe las diferentes tecnologías o estándares para implementar un programa con JavaScript.

Ejercicio 3) Programa (4p)

Se pide implementar en Java (sin utilizar SimpleConcurrent) el programa concurrente *ProductoresConsumidores*.

Este programa debe cumplir los siguientes requisitos:

- Está formado por varios hilos consumidores (NUM_CONSUMIDORES) y dos hilos productores.
- Los productos generados por los hilos productores son números enteros. Un productor genera números pares y el otro productor genera números impares.
- Los productores generan constantemente números (pares e impares) y los insertan en unos buffers para que estén disponibles para los consumidores. Un objeto de la clase *Buffers* representará la estructura de datos que almacena los números hasta que son procesados por los consumidores.
- Un objeto *Buffers* puede almacenar un máximo de 10 números pares y 10 impares.
- Los consumidores primero sacan el número par de los buffers, luego el impar y posteriormente procesan ambos números multiplicándolos y mostrando el resultado por pantalla.
- Los productores no pueden insertar más números si los buffers están llenos.
- Los consumidores no pueden sacar números si los buffers están vacíos.
- Los consumidores deben acceder al buffer bajo exclusión mutua, no puede haber dos consumidores sacando números a la vez.
- Cuando un consumidor ha recogido su número, tendrán preferencia para acceder al buffer recién liberado aquellos consumidores pares frente a los impares. Es decir, un consumidor esperando un número impar no debería acceder a los buffers si hay un consumidor esperando un número par, aunque haya llegado después a la exclusión mutua.

Para la implementación del programa, se ofrece la clase principal *ProductoresConsumidores*. El alumno deberá implementar completamente la clase *Buffers* respetando las cabeceras de los métodos usados en esta clase principal.

```
public class ProductoresConsumidores {

    private static int NUM_CONSUMIDORES = 5;

    private static Buffers buffers;

    private static int generarNumero(boolean par) throws InterruptedException {
        Thread.sleep((long) (Math.random() * 1000));
        int numero = (int) (Math.random() * 10);
        if (numero % 2 != (par ? 0 : 1)) {
            numero++;
        }
        System.out.println(Thread.currentThread().getName() + ": " + numero);
        return numero;
    }

    private static void procesarParImpar(int par, int impar) throws InterruptedException {
        Thread.sleep((long) (Math.random() * 500));
        System.out.println(Thread.currentThread().getName() + ": " + (par * impar));
    }

    public static void productor(boolean par) throws InterruptedException {
        while (true) {
            int numero = generarNumero(par);
            buffers.insertarNumero(numero);
        }
    }

    public static void consumidor() throws InterruptedException {
        while (true) {
            int par = buffers.sacarNumeroPar();
            int impar = buffers.sacarNumeroImpar();
            procesarParImpar(par, impar);
        }
    }

    public static void main(String[] args) {

        buffers = new Buffers();

        //Iniciar NUM_CONSUMIDORES hilos ejecutando consumidor()
        //Iniciar un hilo ejecutando productor(true)
        //Iniciar un hilo ejecutando productor(false)
    }
}
```

Solución: