

# testJunio.pdf



Spoderman



Programación Concurrente y de Tiempo Real



2º Grado en Ingeniería Informática



Escuela Superior de Ingeniería  
Universidad de Cádiz



**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.



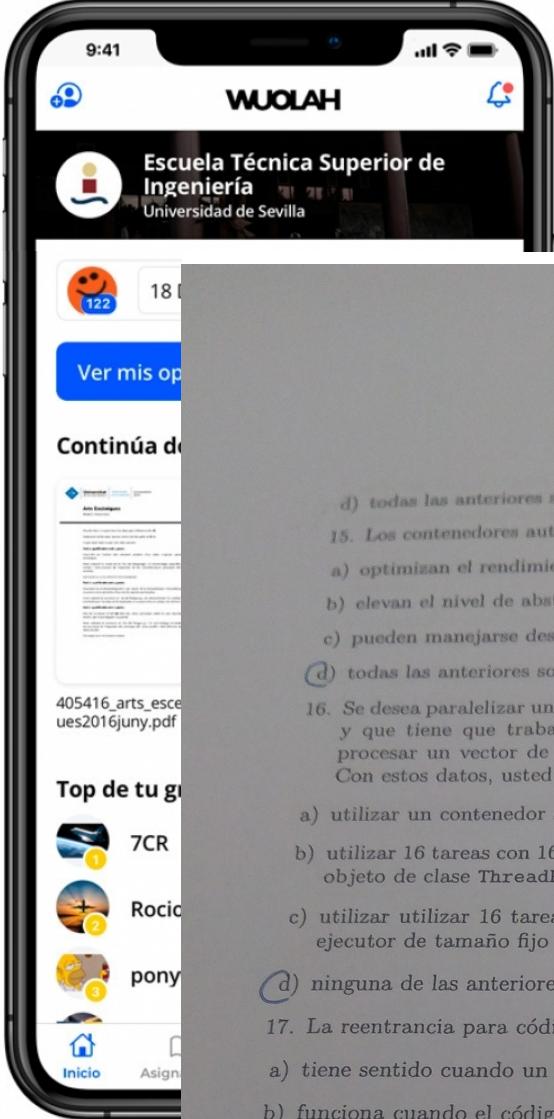


**KEEP  
CALM  
AND  
ESTUDIA  
UN POQUITO**

1. Suponga que 500 hebras comparten el acceso a un objeto común cada una a través de su propia referencia, y ejecutan el acceso a un método `inc()` de ese objeto que incrementa un contador que inicialmente vale cero. El programador protege el acceso haciendo que cada hebra ejecute la llamada a `inc()` envolviendo la misma en un bloque de la forma `synchronized(this){...miReferencia.inc();...}`. El valor final del contador es:
  - a) 500.
  - b) 499.
  - c) un número indeterminado entre 0 y 499.
  - d)** un número indeterminado entre 1 y 500.
2. En Java, los cerrojos de clase `ReentrantLock` se utilizan:
  - a) Para tener regiones críticas de grano muy grueso.
  - b) Para no utilizar regiones `synchronized`.
  - c)** Para poder utilizar variables de condición si hacen falta.
  - d) Para optimizar los accesos a los datos que protegen.
3. Una hebra
  - a) Puede estar detenida en un punto de su código por un bloqueo de exclusión mutua, mientras ejecuta código de una zona diferente.
  - b) Puede acceder a varios objetos diferentes de forma concurrente.
  - c)** Puede ser procesada a través de un ejecutor.
  - d) Ninguna de las anteriores es correcta.
4. Usted debe desarrollar una aplicación bajo el *framework* RMI que controla un sistema de reserva de billetes de avión, y sabe:
  - a)** Que el sistema deberá ser concurrente.
  - b) Que habrá que utilizar necesariamente control de exclusión mutua explícito en el lado del servidor.
  - c) Que deberá crear una hebra de servicio por cada petición procedente de un cliente.
  - d) Que utilizando generación dinámica de resguardos puede prescindir de utilizar un DNS local a la máquina que ejecuta el servidor.
5. Considere un objeto en Java que dispone de tres métodos `m1`, `m2` y `m3`. El programador decide que deben ejecutarse en exclusión mutua, y para ello dispone todo el código de los métodos `m1` y `m3` dentro de un bloque `synchronized(this)`. Para el tercer método, utiliza un objeto de clase `Semaphore` que inicializa a cero, y engloba todo el código del mismo bajo el par de instrucciones `acquire()` y `release()`. La sección crítica en todos los casos implica incrementar una variable de tipo `int` que inicialmente vale 3. Tres hebras externas A, B y C activadas dentro de una co-rutina ejecutan lo siguiente: A.`m1`, C.`m3` y B.`m2`. Terminada la co-rutina, el programa principal imprime el valor de la variable y este es:

- a) 3, ya que las tres hebras quedan bloqueadas y no lo modifican.
- b) 6, pues las tres hebras hacen su incremento de modo segura.
- c) 5, porque dos hebras hacen su incremento y otra no, ya que queda bloqueada.
- d) el programa realmente no imprime nada porque queda bloqueado a causa de las hebras.
6. Se desea implementar un sistema de control de una válvula de presión en una central térmica, y usted determina emplear para ello el API de JRTS. Dado lo anterior, estima que necesita controlar de forma periódica la presión en la válvula, mientras que ocasionalmente inyecta *fuel-oil* en las turbinas de generación. En consecuencia, deberá utilizar
- a) hebras de T.R. periódicas y gestores de eventos asíncronos ligados a hebras de T.R. esporádicas.
- b) hebras de T.R. esporádicas y gestores de eventos asíncronos ligados a hebras de T.R. periódicas.
- c) únicamente gestores de eventos asíncronos.
- d) Ninguna de la anteriores es correcta, y basta utilizar hilos de T.R.
7. Es conocido que la anidación de bloqueos en lenguaje Java sobre dos recursos compartidos puede llevar a dos hebras, en el peor de los casos, a una situación de *deadlock*. Para evitar lo anterior podemos
- a) evitar el anidamiento de bloqueos mediante un monitor.
- b) anidar transacciones.
- c) verificar de manera formal las propiedades de corrección de nuestro código.
- d) utilizar el API de alto nivel del lenguaje.
8. Cuando se utiliza programación CUDA, el programa principal que lanza el *kernel CUDA*:
- a) está sincronizado con parte de los *kernels*, pero no con todos.
- b) no puede ejecutar código paralelo y actuar como coprocesador de la GPU.
- c) puede tener hebras sobre la CPU, pero deben estar en exclusión mutua con los *kernels* de la GPU.
- d) nada de lo anterior es cierto.

9. El uso de ejecutores en lenguaje Java supone
- a) optimizar siempre el rendimiento de la aplicación.
  - (b) procesar indistintamente tareas **Callable** o **Runnable** a través de ellos.
  - c) no controlar la sincronización, pues son los ejecutores quienes controlan el ciclo de vida de las tareas.
  - d) asegurar la exclusión mutua de las tareas sobre los datos compartidos.
10. El uso de variables **volatile** en el ámbito de la concurrencia
- a) nos garantiza que los datos son accedidos de forma segura.
  - b) sirven solo para implementar algoritmos de control de exclusión mutua con espera ocupada.
  - c) únicamente tiene sentido cuando esas variables son estáticas.
  - d) ninguna de las anteriores es cierta.
11. El resultado de paralelizar una aplicación sobre una máquina de ocho **cores** ofrece un *speedup* de 2.75. Esto significa que
- (a) el código original es inherentemente paralelo.
  - b) el coeficiente de bloqueo de la aplicación está próximo a cero.
  - c) no se han utilizado ejecutores en el procesamiento de las hebras paralelas.
  - d) ninguna de las anteriores es correcta.
12. Cuando se utiliza STM en Java sobre Clojure:
- a) desaparece la necesidad de controlar la exclusión mutua.
  - b) los interbloqueos siguen sin poder evitarse.
  - (c) tenemos garantizada la consistencia de la información.
  - d) tenemos escrituras concurrentes y lecturas no concurrentes.
13. En el lenguaje C++ el desarrollo de un monitor exige:
- a) el uso de sincronización **call\_once**.
  - b) el uso exclusivo de **recursive\_mutex**.
  - (c) utilizar **unique\_lock** y variables de condición si son necesarias.
  - d) ninguna de las anteriores es correcta.
14. La implementación del concepto teórico de región crítica en Java
- (a) exige que los métodos sean **synchronized**.
  - b) exige utilizar un objeto auxiliar que actúe como cerrojo.
  - c) exige acotar con **synchronized(this)** la región.



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

d, c, d, a, d

b, c, d, b ~~c~~

~~d~~, c, c, c, d

d, d, c, d ~~b~~

misma se inicializó a cero.  
ita anteriormente, y el valor

Descarga la app de Wuolah desde tu store favorita

WUOLAH