EXAMEN PARCIAL – Bloque Unidades Didácticas 1 a 6 (14/04/2014)

APELLIDOS:		NOMBRE:	
DNI:		FIRMA:	

Este bloque tiene una puntuación máxima de **10 puntos,** que equivalen a 2.5 puntos de la nota final de la asignatura. Indique, para cada una de las siguientes 50 afirmaciones, si éstas son verdaderas (**V**) o falsas (**F**). **Cada respuesta vale: correcta= 0.2, errónea= -0.2, vacía=0.**

1. Ventajas de la programación concurrente sobre la programación secuencial:

F	Los programas concurrentes son más fáciles de depurar. *JUSTIFICACIÓN: Justo al contrario. Los programas concurrentes son más difíciles de depurar, al no ser deterministas.*
F	Los programas concurrentes no generan condiciones de "carrera". *JUSTIFICACIÓN: Si no se aplican los controles adecuados, se pueden producir condiciones de carrera al acceder varios procesos/hilos a las mismas zonas de memoria.*
V	Los programas concurrentes suelen ser más eficientes; es decir, suelen proporcionar un menor tiempo de respuesta en aplicaciones interactivas. *JUSTIFICACIÓN: Los programas concurrentes mejoran la interactividad y la flexibilidad.*
F	Los programas concurrentes no requieren sincronización. *JUSTIFICACIÓN: Los programas concurrentes necesitan cooperar entre sí. Dicha cooperación se consigue mediante la comunicación y la sincronización (para establecer un determinado orden de ejecución, en casos concretos).*

2. Una solución correcta para el problema de la sección crítica:

V	Puede implantarse con un Semaphore "sem" iniciado a 1, utilizando como protocolo de entrada "sem.acquire()" y como protocolo de salida "sem.release()". *JUSTIFICACIÓN: Con el semáforo inicializado a 1 solamente se permite que un hilo pueda entrar en la sección crítica, garantizándose así la exclusión mutua.*
F	Debe cumplir las propiedades de exclusión mutua, retención y espera, no expulsión y espera circular. *JUSTIFICACIÓN: Debe cumplir las propiedades de exclusión mutua, progreso y espera limitada.*
V	Puede implantarse con un ReentrantLock "rl", utilizando como protocolo de entrada "rl.lock()" y como protocolo de salida "rl.unlock()". *JUSTIFICACIÓN: Un ReentrantLock es un tipo de lock y, como tal, permite proteger una sección crítica (SC). Para ello, se debe cerrar el lock a la entrada de la SC, con rl.lock(), y abrirlo a la salida de la SC, con rl.unlock().*
F	Puede implantarse con un CountDownLatch "cdl" iniciado a 1, utilizando como protocolo de entrada "cdl.await()" y como protocolo de salida "cdl.countDown()". *JUSTIFICACIÓN: Con cdl.await() todos los hilos se suspenderán al llamar a dicho método, por lo que ninguno de ellos podrá progresar (es decir, pasar a la sección crítica).*

3. Sobre la variante de monitores propuesta por Brinch Hansen:

F	Es la utilizada al combinar objetos ReentrantLock y Condition de la biblioteca java.util.concurrent. *JUSTIFICACIÓN: En dicha variante se obliga a que la llamada a notify, de requerirse, sea la última a realizar por un hilo antes de dejar el monitor. Al utilizar ReentrantLock y Conditions no se impone ninguna restricción sobre cuándo utilizar notify (o notifyAll).*
F	Los hilos quedan en la cola de entrada urgente del monitor al invocar el método "notify()" de una condición. *JUSTIFICACIÓN: Como en la variante Brinch Hansen se obliga a que si un hilo invoca el método notify() de una condición dicha invocación sea la última instrucción que haga, a continuación ese hilo sale del monitor, por lo que no va a ninguna cola de entrada urgente del monitor.*

V	A diferencia de otras variantes, en ésta no resulta necesario utilizar un esquema "**while** (expresión-lógica) cond.wait()"; basta con un "**if** (expresión-lógica) cond.wait()". *JUSTIFICACIÓN: Como el hilo que notifica abandona inmediatamente el monitor, el hilo reactivado (que estaba esperando en la condición sobre la que se acaba de notificar) siempre encontrará el estado del monitor que necesitaba, ya que ningún otro hilo lo habrá podido modificar.*
F	En esta variante no se permite definir múltiples atributos "condition" en un mismo monitor. *JUSTIFICACIÓN: Esta variante pertenece a la definición general de un monitor, en la que sí se pueden definir múltiples atributos "condition" en un mismo monitor.*

4. Sobre las condiciones de Coffman:

F	Se emplean para definir qué es una secuencia segura. *JUSTIFICACIÓN: Una secuencia segura es aquella en la que encontramos un orden en el que pueden terminar todos los hilos. No se requiere aplicar ninguna condición de Coffman para encontrarla.*
F	Son condiciones necesarias para que se ejecute la sección crítica de forma mutuamente excluyente. *JUSTIFICACIÓN: Son condiciones necesarias para que se produzcan interbloqueos.*
F	Si se cumplen todas ellas, entonces siempre se producirá un interbloqueo. *JUSTIFICACIÓN: Si se cumplen todas ellas existe un riesgo de interbloqueo. Son necesarias, pero no suficientes.*
F	Son utilizadas para diseñar algoritmos de detección de interbloqueos. *JUSTIFICACIÓN: Se utilizan para diseñar algoritmos en los que se previenen los interbloqueos, ya que en dichos algoritmos se rompe alguna de las condiciones y así no se podrán producirse interbloqueos.*

5. El siguiente código Java pretende implantar un monitor para controlar el acceso de hilos lectores (concurrentes) y escritores (de manera exclusiva) sobre un recurso compartido. Los escritores utilizarán writeStart() antes de acceder y writeEnd() tras haber accedido. Los lectores utilizarán readStart() antes de acceder y readEnd() tras haber accedido:

```
 1: public class ReadersWriters {
 2:     private int writersWaiting;
 3:     private boolean writing;
 4:     private int readers;
 5:
 5:     public ReadersWriters() {
 6:         writersWaiting=readers=0;
 7:         writing=false;
 8:     }
 9:
 9:     public synchronized void readStart() {
10:         while (writing || writersWaiting>0)
11:             try { wait(); }
12:             catch(Exception e) { };
13:         readers++;
14:     }
15:     public synchronized void readEnd() {
16:         readers--;
17:         notifyAll();
18:     }
19:     public synchronized void writeStart() {
20:         writersWaiting++;
21:         while (writing || readers>0)
22:             try { wait(); }
23:             catch(Exception e) { };
24:         writersWaiting--;
25:         writing=true;
26:     }
27:     public synchronized void writeEnd() {
28:         writing=false;
29:         notifyAll();
30:     }
31: }
```

F	Este código es erróneo, pues podrá haber múltiples escritores accediendo simultáneamente al recurso. *JUSTIFICACIÓN: Cuando un escritor desea acceder al recurso compartido, ejecuta writeStart. En dicho método, si hay algún otro escritor escribiendo (i.e. writing) o hay lectores leyendo (i.e. readers>0), el escritor se queda esperando. Por tanto, los escritores acceden al recurso en exclusión mutua.*
F	Este código es erróneo, pues podrá haber lectores y escritores accediendo simultáneamente al recurso. *JUSTIFICACIÓN: Como se ha explicado antes, los escritores acceden en exclusión mutua. Por tanto, no podrá haber un escritor accediendo al recurso a la vez que otro escritor ni otro lector.*

F	Este código es erróneo, pues se ha utilizado `wait()` y `notifyAll()` en lugar de `await()` y `signalAll()`. *JUSTIFICACIÓN: La utilización de wait() y notifyAll() es correcta en este código. Los métodos await() y signalAll() se emplean con los objetos Condition.*
F	Este código es erróneo, pues no admite que haya múltiples lectores accediendo simultáneamente. *JUSTIFICACIÓN: Cuando un lector desea acceder al recurso compartido, ejecuta readStart. En dicho método se comprueba si hay algún escritor accediendo al recurso (i.e. writing) o bien esperando a acceder (i.e. writersWaiting>0), y de ser así, el lector debe esperar. Pero, como no se comprueba si hay o no lectores accediendo, el hilo lector podrá acceder al recurso aunque hayan otros hilos lectores accediendo ya al mismo.*

6. Sobre el enunciado y código de la cuestión anterior:

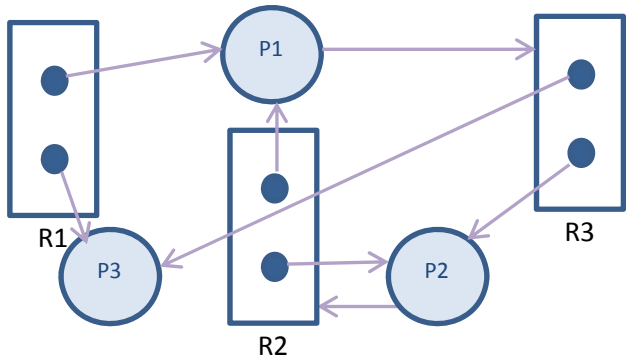
F	Este código otorga prioridad a los lectores. En caso de que haya lectores accediendo y sigan llegando lectores, los escritores tendrán que esperar. *JUSTIFICACIÓN: El código otorga prioridad a los escritores. Si hay lectores accediendo y llega un escritor (que se quedará a la espera, ya que debe ejecutarse en exclusión mutua), si llega un nuevo lector entonces este último quedará también a la espera, pues en su condición comprueba si hay escritores esperando a entrar en el recurso. Por tanto, una vez que llegue al menos un escritor, entonces todos los lectores que lleguen ya no podrán acceder al recurso y quedarán a la espera a que dicho escritor acceda al mismo (y termine el uso del recurso).*
F	Este código no otorga prioridad a ningún tipo de hilo. *JUSTIFICACIÓN: Como se ha explicado antes, este código otorga prioridad a los escritores.*
F	Este código es incorrecto, pues los métodos readStart() y readEnd() no deberían ser "**synchronized**", pues las lecturas jamás provocan condiciones de "carrera". *JUSTIFICACIÓN: Los métodos readStart() y readEnd() pertenecen al monitor y, por tanto, deben ejecutarse en exclusión mutua. Y para ello se requiere que sean" synchronized".*
F	Este código no implanta ningún monitor, pues en ningún momento se utiliza sincronización condicional. *JUSTIFICACIÓN: Sí que se utiliza sincronización condicional, ya que en los métodos readStart y readEnd se controla el acceso al recurso, de modo que el hilo que los ejecuta se queda a la espera (suspendido en la condición implícita del monitor) si la condición correspondiente no se cumple (ej: while (writing || writersWaiting>0). En los métodos readEnd y writeEnd, tras modificar el estado del monitor, se notifica a los hilos suspendidos en la condición, para que puedan reevaluar si la condición que requieren ya se cumple.*

7. Los objetos `CountDownLatch` de `java.util.concurrent` …:

F	...deben ser definidos llamando al método newCountDownLatch() de la clase ReentrantLock. *JUSTIFICACIÓN: Se definen como cualquier otro objeto, es decir, instanciando un objeto de su clase. Ejemplo: CountDownLatch cl= new CountDownLatch(5);*
-	...se utilizan para implantar sincronización condicional. *NOTA: Esta pregunta ha sido anulada, ya que podía ser tanto VERDADERA como FALSA, según se interpretase el concepto "sincronización condicional". En muchos trabajos, se utiliza este concepto solamente referido a la gestión de los monitores. En dicho caso, esta afirmación es falsa. En otros trabajos, la "sincronización condicional" se emplea para cualquier situación en la que se requiere que unos hilos esperen a otros por una determinada condición. En tal caso, se podría emplear CountDownLatch para gestionar dicha espera, por lo que la afirmación sería cierta.*
F	...garantizan que el método await() siempre suspenderá al hilo invocador. *JUSTIFICACIÓN: Si la barrera está cerrada, el método await() suspende al invocador. Sin embargo, una vez abierta la barrera (i.e. cuando el contador llega a cero), el método await() ya no suspende al hilo invocador.*
F	...son barreras que pueden reiniciarse utilizando su método reset(). *JUSTIFICACIÓN: Una vez abierta la barrera, ésta ya no puede volver a cerrarse.*
F	... se crean con un valor entero que indica el número de hilos que deberán quedarse esperando en la barrera para poder abrirla. *JUSTIFICACIÓN: El número de hilos que se pueden quedar esperando en la barrera es*

	independiente del valor que se le proporcione en la creación. Dicho valor entero se decrementa en uno cada vez que se ejecuta el método countDown() de este objeto.

8. Dado el siguiente grafo de asignación de recursos…:



F	... se da un interbloqueo pues no hay instancias libres de ningún recurso. *JUSTIFICACIÓN: Que no hayan instancias libres no garantiza que exista un interbloqueo.*
F	... para saber si hay o no interbloqueo en este GAR emplearemos un algoritmo de evitación. *JUSTIFICACIÓN: Lo que emplearemos es el análisis del GAR, viendo si hay un ciclo dirigido y si se puede encontrar una secuencia segura. Un algoritmo de evitación hace uso de un GAR para determinar si hay riesgo de interbloqueo y decidir así si otorgar o no una petición de recurso.*
F	... existe un interbloqueo entre P1 y P2, pues existe un ciclo dirigido de esperas: P1 -> R3 -> P2 -> R2 -> P1. *JUSTIFICACIÓN: Si existe un ciclo dirigido de esperas entonces hay un riesgo de interbloqueo, pero no podemos tener la certeza de la existencia de un interbloqueo. Para dicha certeza debemos analizar la existencia de secuencias seguras.*
F	... encontramos la secuencia de terminación <P3, P2, P1>. *JUSTIFICACIÓN: La única secuencia de terminación que se puede encontrar es <P3, P1, P2>.*

9. Sea el conjunto de tareas en un sistema de tiempo real descrito por la siguiente tabla:

Tarea	Periodo (T)	Cómputo (C)	Plazo (D)	Prioridad
A	5	2	4	1
B	15	5	12	2
C	20	8	16	3

Asuma una asignación de prioridades en la que la tarea con menor valor numérico será la más prioritaria.

F	Este sistema no supera el análisis de planificabilidad ya que la respuesta de la tarea A se produce fuera de su plazo. *JUSTIFICACIÓN: R_A=2 <=4 (D_A)*
F	Este sistema no supera el análisis de planificabilidad ya que la respuesta de la tarea B se produce fuera de su plazo. *JUSTIFICACIÓN: W_B^0=5+2=7 <=12; W_B^1=5+[7/5]*2=5+2*2=9 <=12; W_B^2=5+[9/5]*2=9* *Por tanto, R_B=9 <= 12 (D_B) está en plazo.*
V	Este sistema no supera el análisis de planificabilidad ya que la respuesta de la tarea C se produce fuera de su plazo. *JUSTIFICACIÓN: W_C^0=8+5+2=15<=16; W_C^1=8 + [15/15]* 5 + [15/5]*2= 8 + 5 + 3*2 =19 >16.* *R_C > 16. El sistema no es planificable.*
F	Este sistema sería planificable si el intervalo de cómputo de la tarea C fuera 5 unidades de tiempo (C_C=5). *JUSTIFICACIÓN: W_C^0=5+5+2=12 <= 16; W_C^1=5 + [12/15]*5+ [12/5]*2= 5 + 5+3*2= 16.* *W_C^2=5 + [16/15]*5+ [16/5]*2= 5 + 2*5+4*2= 23 > 16.* *R_C>16. Sigue sin ser planificable.*

10. Los protocolos de entrada/salida que controlan el acceso a una sección crítica deben siempre garantizar:

F	Que no se produzca la espera circular. *JUSTIFICACIÓN: Los protocolos de entrada/salida de una sección crítica deben garantizar la exclusión mutua, progreso y espera limitada.*
F	Que no se produzca la expropiación de los recursos. *JUSTIFICACIÓN: Misma respuesta que en la afirmación anterior.*
F	Que se respete retención y espera. *JUSTIFICACIÓN: Misma respuesta que en la primera afirmación.*
V	Que se respete exclusión mutua, progreso y espera limitada. *JUSTIFICACIÓN: Misma respuesta que en la primera afirmación.*

11. Dado el siguiente programa Java:

```
class Prueba extends Thread {
    private int i;
    public Prueba(int a) {i=a;}
    public void run() {
        for (int j=0; j<i; j++)
            new Prueba(j).start();
        System.out.println("i:"+ i);
    }
    static public void main(String args[]) {
        new Prueba(2).start();
    }
}
```

V	Veremos al menos 2 líneas que contengan "i:0". *JUSTIFICACIÓN: Al lanzar el siguiente programa a ejecución, se creará y ejecutará en primer lugar un hilo (llamado Prueba), con parámetro 2. Dicho hilo, en su método run, crea otros dos hilos Prueba (llamémosles A y B), con parámetro j=0 y j=1 respectivamente, lanzándolos también a ejecución. Además, el hilo inicial imprime "i:2". Cada uno de los dos nuevos hilos creados (A y B) ejecutará su método run. Así, el hilo A (con valor i=0) no entrará en su bucle for, por lo que no crea ningún hilo más, e imprimirá la línea "i:0". Por su parte, el hilo B (con valor i=1) sí que entrará en su bucle for, generando un nuevo hilo (llamémosle hilo C) con parámetro j=0. Y se imprimirá la línea "i:1". Finalmente, el hilo C (con valor i=0) no creará ningún hilo más, al no entrar en su bucle for, y se imprimirá la línea "i:0". En definitiva, se imprimirán dos líneas con "i:0".*
F	Veremos al menos 2 líneas que contengan "i:1". *JUSTIFICACIÓN: Como se ha comentado antes, solamente se imprimirá una línea con valor "i:1".*
F	No se ejecutará ningún hilo (aparte del principal). *JUSTIFICACIÓN: Se han ejecutado 3 hilos aparte del principal.*
V	Se crearán y se ejecutarán al menos 3 hilos. *JUSTIFICACIÓN: Se han ejecutado 3 hilos aparte del principal.*

12. Sea el conjunto de tareas en un sistema de tiempo real descrito por la siguiente tabla:

Tarea	Periodo (T)	Cómputo (C)	Plazo (D)	Prioridad
A	5	2	4	1
B	15	4	10	2
C	20	3	16	3

Asumiendo una asignación de prioridades en la que la tarea con menor valor numérico será la más prioritaria, y considerando que dichas tareas utilizan dos semáforos S1 y S2 de la siguiente manera:

Tarea	Semáforo	Duración de la sección crítica
A	S1	1
B	S2	3
C	S1	2
C	S2	1

F	El techo del semáforo S1 es 1 y el techo del semáforo S2 también es 1.

	JUSTIFICACIÓN: El semáforo S1 es utilizado por las tareas A y C. La más prioritaria es la tarea A, por los que su techo es igual a la prioridad de A, es decir, 1. *El semáforo S2 es utilizado por las tareas B y C, siendo B la tarea más prioritaria. Por tanto, el techo de S2 es igual a la prioridad de B, es decir, 2.*
V	El factor de bloqueo máximo para la tarea A es 2. *JUSTIFICACIÓN: Las tareas con menor prioridad que A utilizan los semáforos S1 y S2. Solamente S1 tiene techo con valor igual o superior a la prioridad de A (en este caso, techo(S1)=prio(A)). Dicho semáforo lo emplea la tarea C con una duración de la sección crítica igual a 2. Por tanto, el factor de bloqueo máximo para A es 2.*
V	Si se aplica el protocolo de techo de prioridad inmediato, el tiempo de respuesta en el peor caso para la tarea B es 10. *JUSTIFICACIÓN: El factor de bloqueo para B es 2, que se corresponde con la duración de la sección crítica de C usando el semáforo S1, que tiene techo > prio(B). Si añadimos dicho factor de bloqueo a la fórmula del cálculo del tiempo de respuesta tenemos que:* *W_B^0=2+4+2=8<=10; W_B^1=2+4+[8/5]*2=2+4+2*2=10<=10; W_B^2=2+4+[10/5]*2=1+4+2*2=10<=10* *Por tanto, R_B=10.*
F	Haciendo uso del protocolo del techo de prioridad inmediato, el sistema no es planificable. *JUSTIFICACIÓN: Con la afirmación anterior hemos visto que R_B=10. Falta calcular R_A y R_C.* *R_A= 2 + 2 = 4 <= 4 (D_A), pues su factor de bloqueo B_A=2.* *Para R_C debemos calcular primero su factor de bloqueo, que en este ejemplo es 0 ya que es la tarea con menor prioridad. Por ello, el cálculo de su tiempo de respuesta es igual que para el análisis de planificabilidad sin aplicar el protocolo de techo de prioridad inmediato.* *W_C^0=3+4+2=9 <=16; W_C^1=3+[9/15]*4+[9/5]*2=3+1*4+2*2=11 <=16;* *W_C^2=3+[11/15]*4+[11/5]*2=3+1*4+3*2=13 <=16; W_C^3=3+[13/15]*4+[13/5]*2=13 <=16* *Por tanto, R_C=13.* *Como R_A<= D_A , R_B<= D_B y R_C<= D_C, el sistema sí es planificable.*
V	Si se aplica el protocolo de techo de prioridad inmediato, el tiempo de respuesta en el peor caso para la tarea C es 13. *JUSTIFICACIÓN: Como se ha visto en la respuesta de la afirmación anterior, aplicando dicho protocolo se obtiene que R_C=13 (pues su factor de bloqueo es 0).*