Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 1 (4F1M) - Clave a

Concurrencia

2010-2011 - Segundo semestre

Dpto. de Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **3 preguntas de respuesta simple** y **una pregunta de desarrollo** en **4 páginas**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora**. El examen debe contestarse en las **mismas hojas**. No olvidéis rellenar **apellidos, nombre y número de matricula** en cada hoja.

Sólo hay una respuesta válida a cada pregunta de respuesta simple. Toda pregunta en que se marque más de una respuesta se considerará incorrectamente contestada. Toda pregunta incorrectamente contestada restará del examen una cantidad de puntos igual a la puntuación de la pregunta dividido por el número de alternativas ofrecidas en la misma.

Cuestionario

(2 puntos) 1. Dado el siguiente **CTAD** (sólo se muestran las partes necesarias):

TIPO: $LE = (l : \mathbb{Z} \times e : \mathbb{Z})$
INVARIANTE: $\forall\, r \in LE \bullet\ r.e \geq 0 \wedge r.l \geq 0 \wedge r.e \leq 1 \wedge ((r.e > 0 \Rightarrow r.l = 0) \wedge (r.l > 0 \Rightarrow r.e = 0))$

INICIAL(r): $r = (0, 0)$

CPRE: $r.e = 0$
IL(r)
POST: $r = (r^{pre}.l + 1, r^{pre}.e)$

CPRE: $r.e = 0 \wedge r.l = 0$
IE(r)
POST: $r = (r^{pre}.l, r^{pre}.e + 1)$

CPRE: *cierto*
FL(r)
POST: $r = (r^{pre}.l - 1, r^{pre}.e)$

CPRE: *cierto*
FE(r)
POST: $r = (r^{pre}.l, r^{pre}.e - 1)$

Se pide marcar la afirmación correcta:

(a) El recurso puede pasar por un máximo de cuatro estados.

(b) Ninguna operación de las cuatro puede violar la invariante del recurso, con independencia del orden en que se invoquen.

(c) Si hay *nl* procesos que respetan el esquema de llamadas $IL(r);\ \ldots;\ FL(r)$ y *ne* procesos que respetan el esquema $IE(r);\ \ldots;\ FE(r)$, el recurso puede pasar por $nl + 2$ estados, y no más.

(d) Si hay *nl* procesos que respetan el esquema de llamadas $IL(r);\ \ldots;\ FL(r)$ y *ne* procesos que respetan el esquema $IE(r);\ \ldots;\ FE(r)$, el recurso puede pasar por $1 + nl + ne$ estados, y no más.

PISTA: Dibujad el grafo de estados en este hueco:

(2 puntos) 2. Dado el siguiente programa concurrente en el que `Dato` es una clase cuyos atributos internos son de tipo **`int`** y `m` un método de los objetos de dicha clase que sólo trabaja con dichos atributos:

```
static Dato x = new Dato();
```

```
static class T extends Thread {
   private Dato y;
   public T (Dato y) {
      this.y = y;
   }
   public void run() {
      Dato z = new Dato();
      x.m(); y.m(); z.m();
   }
}
```

```
// Programa principal
Dato w1 = new Dato();
Dato w2 = new Dato();
Thread[] threads = new Thread[] {new T(w1), new T(w2)};
threads[0].start(); threads[1].start();
threads[0].join(); threads[1].join();
```

Se pide marcar la afirmación correcta.

(a) `x.m()` es una sección crítica.

(b) `y.m()` es una sección crítica.

(c) `z.m()` es una sección crítica.

(d) Ninguna de las otras respuestas es correcta.

(2 puntos) 3. Dadas las clases de threads `T1`, `T2` y `T3`, que comparten una variable `n`, se lanzan sendas instancias `t1`, `t2` y `t3`:

```
static int n = 3;
static Semaphore s1 =
   new Semaphore(1);
static Semaphore s2 =
   new Semaphore(1);
static Semaphore s3 =
   new Semaphore(0);

static class T1 extends Thread {
   public void run() {
      s1.acquire();
      s2.acquire();
      n = n + 2;
   }
}
```

```
static class T2 extends Thread {
   public void run() {
      s2.acquire();
      s3.acquire();
      n = n * 2;
      s2.release();
   }
}

static class T3 extends Thread {
   public void run() {
      s1.acquire();
      n = n * n;
      s1.release();
      s2.release();
      s3.release();
   }
}
```

Se pide marcar la respuesta correcta:

(a) No está garantizada la exclusión mutua en el acceso a la variable `n`.

(b) Está asegurada la terminación de la tarea `t3`.

(c) Si las tres tareas terminan y no se viola la exclusión mutua, la variable `n` puede tomar el valor 100.

(d) Si las tres tareas terminan y no se viola la exclusión mutua, la variable `n` puede tomar el valor 38.

Apellidos: **Nombre:** **Matrícula:**

(4 puntos) 4. Se pide especificar un *buffer síncrono* con capacidad para un dato, es decir, un buffer en el que la operación de almacenar bloquea hasta que la operación de extraer es ejecutada. Para ello, durante la etapa de diseño, ha sido necesario desdoblar la operación de almacenar en dos de forma que un proceso que quiera almacenar un dato deberá ejecutar las dos operaciones de forma consecutiva respetando el siguiente esquema de llamada: *DejarDato(b,d); EsperarExtraer(b);*. La interfaz es la siguiente:

C-TAD BufferSinc
OPERACIONES
ACCIÓN DejarDato: *BufferSinc[io]* $\times$ *Dato[i]*
ACCIÓN EsperarExtraer: *BufferSinc[io]*
ACCIÓN Extraer: *BufferSinc[io]* $\times$ *Dato[o]*

Se pide especificar en esta página el dominio (tipos e invariante), estado inicial y las CPREs y POSTs de las operaciones *DejarDato*, *EsperarExtraer* y *Extraer*.

(Utiliza esta página para trabajar en sucio si lo necesitas.)