Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 1 - Clave a

Concurrencia

2010-2011 - Primer 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 y cuarto**. Las preguntas de respuesta simple deben contestarse en las **hojas de respuestas**. No olvidéis rellenar **apellidos, nombre y DNI** en cada hoja de respuesta.

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: T_Cuatro = $(a : \mathbb{B} \times b : \mathbb{B})$

INICIAL(r): $r = (falso, cierto)$

CPRE: *cierto*
S(r)
POST: $r^{pre} = (pe, se) \wedge r = (ps, \neg se) \wedge ps = (\neg pe \wedge se) \vee (pe \wedge \neg se)$

CPRE: $r \neq (cierto, falso)$
M(r)
POST: $r = (\neg r^{pre}.a, r^{pre}.b)$

Se pide marcar la afirmación correcta:

(a) El recurso puede pasar, a lo sumo, por 3 estados diferentes.

(b) El recurso podría estar en un estado dado y, tras ejecutarse una sola de sus acciones, continuar en el mismo estado.

(c) Si el sistema solo contiene (además de un recurso de este tipo) un único proceso que intenta ejecutar S indefinidamente, el sistema acaba en interbloqueo.

(d) Si el sistema consta de un recurso de este tipo, un proceso que invoca a S una sola vez y otro que intenta ejecutar M indefinidamente, el sistema puede terminar en interbloqueo.

PISTA: Dibujar el grafo de estados en este hueco:

(2 puntos) 2. Dado el siguiente programa concurrente

```
static int x = 0;
```

```
static class T extends Thread {
    private int y;
    public T (int y) {
        this.y = y;
    }
```

```
    public void run() {
        int z = y;
        z = z + y; x = x + z;
    }
}
```

```
    // Programa principal
    Thread[] t = new Thread[] {new T(1), new T(2)};
    t[0].start(); t[1].start(); t[0].join(); t[1].join();
```

Se pide marcar la afirmación correcta.

(a) Es necesario asegurar exclusión mutua en el acceso a la variable `x`.

(b) Es necesario asegurar exclusión mutua en el acceso al atributo `y`.

(c) Es necesario asegurar exclusión mutua en el acceso a la variable `z`.

(d) Ninguna de las otras respuestas es correcta.

(2 puntos) 3. Dado un programa concurrente en la que tres *threads* instancias de las clases `C`, `D` y `E` comparten una variable `n`:

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

static class C extends Thread {
    public void run() {
        s2.acquire();
        s1.acquire();
        n = 3 * n;
        s1.release();
    }
}
```

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

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

Se pide marcar el conjunto que contiene únicamente los posibles valores de la variable `n` tras la terminación de los tres threads.

(a) $\{2, 12, 36\}$

(b) $\{12, 36\}$

(c) $\{6, 12, 36\}$

(d) $\{2, 6, 12\}$

Apellidos: **Nombre:** **Matrícula:**

(4 puntos) 4. Una pequeña variación del típico problema del *productor-buffer-consumidor* es el llamado "buffer de pares e impares" que tenéis en los apuntes de especificación de recursos. La idea es que en la operación de extracción se permite decir si queremos retirar un número par o un número impar.

Se pide completar la especificación formal del recurso.

C-TAD BufferPI

OPERACIONES
ACCIÓN Poner: *Tipo_Buffer_PI[es]* × *Tipo_Dato[e]*
ACCIÓN Tomar: *Tipo_Buffer_PI[es]* × *Tipo_Dato[s]* × *Tipo_Paridad[e]*

SEMÁNTICA
DOMINIO:
TIPO: *Tipo_Buffer_PI* = Secuencia *(Tipo_Dato)*
Tipo_Paridad = *par*|*impar*
Tipo_Dato = ℕ
INVARIANTE: …
DONDE: *asumir la existencia de una constante MAX*
INICIAL(b)**:** …

***CPRE*:** *El buffer no está lleno*
CPRE: …
Poner(b, d)
***POST*:** *Añadimos un elemento al buffer*
POST: …

***CPRE*:** *El buffer no está vacío y el primer dato preparado para salir es del tipo que requerimos*
CPRE: …
Tomar(b, d, t)
***POST*:** *Retiramos el primer elemento del buffer*
POST: …

(Escribe aquí la solución a la pregunta de desarrollo.)