Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 1 - Clave a

Concurrencia

2012-2013 - Primer semestre

Dpto. de Lenguajes, Sistemas Informáticos e Ingeniería de Software
Universidad Politécnica de Madrid

Normas

Este es un cuestionario que consta de **7 preguntas** en **4 páginas**. Todas las preguntas son **preguntas de respuesta simple** excepto la pregunta 7 que es una **pregunta de desarrollo**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora**. El examen debe 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

(1 punto) 1. ¿Cuál es el método de los objetos de la clase `java.lang.Thread` que pone en marcha un thread?

(a) `run`.

(b) `start`.

(1 punto) 2. El siguiente código, una variación del algoritmo de Peterson, pretende garantizar la exclusión mutua en el acceso a las secciones críticas `n++;` y `n--;` desde, respectivamente, dos *threads*:

```
static class MutexEA {
   static final int N_PASOS = 1000000;

   // Variable compartida
   volatile static int n = 0;

   // Variables para asegurar mutex
   volatile static boolean turno_inc;
   volatile static boolean en_sc_inc =
      false;
   volatile static boolean en_sc_dec =
      false;

   static class Incrementador
      extends Thread {
      public void run () {
         for (int i = 0;
              i < N_PASOS;
              i++) {
            en_sc_inc = true;
            turno_inc = true;
            while (en_sc_dec
                   && !turno_inc) { }
            n++;
            en_sc_inc = false;
         }
      }
   }
```

```
   static class Decrementador
      extends Thread {
      public void run () {
         for (int i = 0;
              i < N_PASOS;
              i++) {
            en_sc_dec = true;
            turno_inc = false;
            while (en_sc_inc
                   && turno_inc) { }
            n--;
            en_sc_dec = false;
         }
      }
   }

   public static final void
      main(final String[] args)
      throws InterruptedException {
      Thread t1 = new Incrementador();
      Thread t2 = new Decrementador();

      t1.start();
      t2.start();
      t1.join();
      t2.join();
   }
}
```

Se pide señalar la respuesta correcta.

(a) No se garantiza la propiedad de exclusión mutua.

(b) Se garantiza la propiedad de exclusión mutua.

(1 punto) 3. Dada la siguiente clase *thread* en la que `Dato` es una clase con atributos no estáticos de tipo **`int`** y `m` un método que accede y modifica dichos atributos:

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

Se pide marcar la afirmación correcta.

(a) `y.m()` nunca es una sección crítica.

(b) `y.m()` puede ser una sección crítica.

(1½ puntos) 4. Supongamos un programa concurrente con cuatro tipos de procesos T_0, T_1, T_2 y T_3 que ejecutan repetidamente operaciones *r.Cero*, *r.Uno*, *r.Dos* y *r.Tres*, respectivamente. Suponiendo que existe al menos un proceso de cada tipo y que *r* es un recurso compartido del tipo especificado a continuación[1]:

C-TAD Circular
TIPO: *Circular* = *(a :* $\mathbb{B} \times$ *b :* $\mathbb{B}$*)*
INVARIANTE: self.$a \vee$ *self*.b

INICIAL: self $=$ (Cierto, Cierto)

CPRE: Cierto
Cero
POST: self $=$ (self^{pre}.a, self^{pre}.b)

CPRE: self.$a \wedge$ self.b
Uno
POST: self $=$ ($\neg$self^{pre}.a, self^{pre}.b)

CPRE: Cierto
Dos
POST: self $=$ (self^{pre}.a, $\neg$self^{pre}.b)

CPRE: $\neg$self.$a \vee \neg$self.b
Tres
POST: self $=$ ($\neg$self^{pre}.a, $\neg$self^{pre}.b)

Se pide señalar la respuesta correcta.

(a) En el programa se puede violar la invariante de *r*.

(b) En el programa no se puede violar la invariante de *r*.

(1½ puntos) 5. Si eliminamos los procesos de tipo T_2 (aquellos que ejecutaban *r.Dos*) del programa concurrente descrito en la pregunta 4. **Se pide** señalar la respuesta correcta.

(a) El número de estados posibles del recurso *r* es 2.

(b) El número de estados posibles del recurso *r* es 3.

(c) El número de estados posibles del recurso *r* es 4.

(1 punto) 6. Supóngase un programa concurrente con un semáforo inicializado a 0. Dos procesos invocan el método `await` sobre dicho semáforo y posteriormente un tercer proceso invoca el método `signal` sobre el mismo semáforo. Supóngase que el programa se detiene sin realizar más invocaciones de métodos sobre dicho semáforo.

Se pide marcar la afirmación correcta.

(a) El valor del semáforo es 1 al detener el programa.

(b) El valor del semáforo es 0 al detener el programa.

[1] No se muestra la declaración de operaciones.

Apellidos: **Nombre:** **Matrícula:**

(3 puntos) 7. Se pide especificar un *buffer replicador* con capacidad para un dato. En la operación de inserción se especifica el número de veces que el dato puede ser *extraído*. De esta forma, si un proceso ejecuta, por ejemplo, $b.insertar(5, x)$, será necesario ejecutar 5 veces la operación $b.extraer(d)$ antes de que se pueda volver a realizar una llamada a *insertar*. Esas 5 extracciones del ejemplo extraen el dato *x* en el parámetro de salida *d*.

Se pide completar la especificación.

C-TAD BufferX

OPERACIONES

ACCIÓN insertar: $\mathbb{N}$*[i]* $\times$ *Dato[i]*
ACCIÓN extraer: *Dato[o]*

SEMÁNTICA
DOMINIO:

TIPO: *BufferX* =

INVARIANTE:

INICIAL:

CPRE:

insertar(n,d)

POST:

CPRE:

extraer(d)

POST:

(Página intencionadamente en blanco. Puede ser usada como hoja en sucio.)