Apellidos: **Nombre:** **Matrícula:**

Prueba final, parte 1 - Clave a

Concurrencia

2011-2012 - Segundo semestre

Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **6 preguntas** en **3 páginas**. Todas las preguntas son **preguntas de respuesta simple** excepto la pregunta 6 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

(1½ puntos) 1. El siguiente código 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 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;
            while (en_sc_dec) {
               en_sc_inc = false;
               en_sc_inc = true;
            }
            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;
            while (en_sc_inc) {
               en_sc_dec = false;
               en_sc_dec = true;
            }
            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.

(1½ puntos) 2. Dado el siguiente programa:

```
class Hilo extends Thread {
   public static void main() {
      Thread t1 = new Hilo();
      Thread t2 = new Hilo();
      t1.start(); t2.start();
      t2.join(); t1.join();
   }
   public void run() {
      // tarda en ejecutar
      // al menos 1 minuto
   }
}
```

Se pide señalar la respuesta correcta:

(a) El programa tarda en ejecutar al menos dos minutos.

(b) El programa puede tardar en ejecutar menos de dos minutos.

(1½ puntos) 3. Supongamos un programa concurrente con procesos (al menos uno) que ejecutan repetidamente operaciones $r.inc(x)$ con x natural menor o igual que 10, y procesos (al menos uno) que ejecutan repetidamente operaciones $r.dec(y)$ con y natural menor o igual que 10, siendo r un recurso compartido del tipo especificado a continuación:

C-TAD MultiCont
OPERACIONES
ACCIÓN inc: $\mathbb{N}$*[e]*
ACCIÓN dec: $\mathbb{N}$*[e]*
SEMÁNTICA
DOMINIO:
TIPO: *MultiCont* = $\mathbb{N}$
INVARIANTE: self ≤ 10

INICIAL: self $= 0$

PRE: $n \leq 10$
CPRE: self $+ n \leq 10$
inc(n)
POST: self $=$ self$^{pre} + n$

PRE: $n \leq 10$
CPRE: $n \leq$ self
dec(n)
POST: self $=$ self$^{pre} - n$

Se pide señalar la respuesta correcta.

(a) El programa cumple la propiedad de ausencia de interbloqueo.

(b) El programa no cumple la propiedad de ausencia de interbloqueo.

(1½ puntos) 4. Dado el programa concurrente descrito en la pregunta 3. **Se pide** señalar la respuesta correcta.

(a) El programa puede violar la invariante de r.

(b) El programa no viola la invariante de r.

(1 punto) 5. Obsérvese la siguiente implementación del recurso compartido MultiCont especificado en la pregunta 3:

```
class MultiCont {
   private Semaphore permInc = new Semaphore(N);
   private Semaphore permDec = new Semaphore(0);
   private Semaphore atomicInc = new Semaphore(1);
   private Semaphore atomicDec = new Semaphore(1);

   public void inc(int n) {
      atomicInc.await();
      for (int i = 0; i < n; i++)
         permInc.await();
      atomicInc.signal();
   }
   public void dec(int n) {
      atomicDec.await();
      for (int i = 0; i < n; i++)
         permDec.await();
      atomicDec.signal();
   }
}
```

Asumiendo que se quiere atender a los procesos que quieren incrementos en estricto orden de llegada y a los que quieren realizar decrementos también en estricto orden de llegada, **se pide** señalar la respuesta correcta.

(a) Es una implementación correcta del recurso compartido.

(b) Es una implementación incorrecta del recurso compartido.

Apellidos: **Nombre:** **Matrícula:**

(3 puntos) 6. El objetivo de este ejercicio es especificar una *barrera* de *N threads*. La idea consiste en que los *threads* se van quedando bloqueados al llegar a la barrera hasta que han llegado *N threads*, momento en el cuál todos se debloquean. Para ello se decide crear un recurso compartido con dos operaciones ejecutadas secuencialmente por cada thread: *b.avisarLlegada(); b.continuar()*. Con la operación *avisarLlegada* el thread avisa de su llegada y con la operación *continuar* el thread se bloquea hasta que hayan llegado los *N*.

Se pide completar la especificación.

Nota: se sugiere que el dominio del recurso sean dos contadores, uno para el número de threads bloqueados (incrementado en *avisarLlegada* y decrementado en *continuar*) y otro para el número de threads desbloqueados (incrementado en *continuar* cuando el número de threads bloqueados sea mayor que 0 y puesto a 0 en *continuar* cuando el número de threads bloqueados ha llegado a 0). Se valorará esencialmente (2 puntos) que las condiciones de sincronización (CPREs) terminen de implementar el efecto de *barrera*.

C-TAD Barrera

OPERACIONES

ACCIÓN avisarLlegada:
ACCIÓN continuar:

SEMÁNTICA
DOMINIO:

TIPO: *Barrera* =

INVARIANTE:

INICIAL:

CPRE:

avisarLlegada()

POST:

CPRE:

continuar()

POST:

(Página intencionadamente en blanco, puede usarse como hoja en sucio).