Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 1 - Clave a

Concurrencia

2011-2012 - Primer semestre

Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **8 preguntas** en **4 páginas**. Todas las preguntas son **preguntas de respuesta simple** excepto la pregunta 8 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. El siguiente código pretende garantizar la exclusión mutua en el acceso a las secciones críticas `n++;` y `n--;` desde, respectivamente, dos *threads*:

```
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) No se garantiza la propiedad de ausencia de interbloqueo.

(c) No se garantiza la propiedad de ausencia de inanición.

(d) Ninguna de las otras respuestas es correcta.

(1 punto) 2. Supongamos un programa concurrente con procesos (al menos uno) que ejecutan repetidamente operaciones $r.inc(x)$ con $x < N/2$ y procesos (al menos uno) que ejecutan repetidamente operaciones $r.dec(y)$ con $y < N/2$, 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: $0 \leq \text{self} \wedge \text{self} \leq N$

INICIAL: $\text{self} = 0$

PRE: $n < N/2$
CPRE: $\text{self} + n \leq N$
inc(n)
POST: $\text{self} = \text{self}^{pre} + n$

PRE: $n < N/2$
CPRE: $n \leq \text{self}$
dec(n)
POST: $\text{self} + n = \text{self}^{pre}$

Se pide señalar la respuesta correcta.

(a) El programa puede violar la invariante del recurso compartido.

(b) El programa no viola la invariante del recurso compartido.

(1 punto) 3. Dado el programa concurrente descrito en la pregunta 2. **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.

(1 punto) 4. Dado el programa concurrente descrito en la pregunta 2. **Se pide** señalar la respuesta correcta.

(a) La especificación de la operación de decremento (*dec*) es incorrecta.

(b) La especificación de la operación de decremento (*dec*) es correcta.

(1 punto) 5. Obsérvese la siguiente implementación del recurso compartido MultiCont especificado en la pregunta 2:

```
class MultiCont {
   private Semaphore permInc = new Semaphore(N);
   private Semaphore multicont = new Semaphore(0);

   public void inc(int n) {
      for (int i = 0; i < n; i++)
         permInc.await();
      for (int i = 0; i < n; i++)
         multicont.signal();
   }

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

La idea principal consiste en que el semáforo `multicont` represente el valor interno (de tipo $\mathbb{N}$) del recurso.

Se pide señalar la respuesta correcta.

(a) Es una implementación correcta del recurso compartido.

(b) Es una implementación incorrecta del recurso compartido.

(1 punto) 6. **Se pide** señalar la respuesta correcta.

(a) El acceso a un atributo de un *thread* desde su método `run` nunca es una sección crítica.

(b) El acceso a un atributo de un *thread* desde su método `run` puede ser una sección crítica.

(1 punto) 7. **Se pide** señalar la respuesta correcta.

(a) Un *thread* inicia la ejecución del método `run` en el momento en el que se crea el *thread* con **`new`**.

(b) Un *thread* inicia la ejecución del método `run` en el momento en el que se ejecuta el método `start` de dicho *thread*.

Apellidos: **Nombre:** **Matrícula:**

(3 puntos) 8. Una factoría tiene dos naves no adyacentes: la nave 0 y la nave 1. Por dichas naves circulan robots encargados de realizar diversas tareas. Para que los robots puedan maniobrar necesitan espacio dentro de las naves. Dicho espacio no está garantizado cuando en una misma nave hay más de *MaxN* robots. Fuera de las naves el espacio es también reducido por lo que no puede haber más de *MaxF* robots (donde $MaxF < 2 * MaxN$) fuera de las mismas. Al arrancar los sistemas cada mañana se debe asumir que hay *MaxN* robots en cada nave.

Se desea crear un recurso compartido para gestionar el movimiento de robots por la factoría. Los *threads* que controlan los robots ejecutan la operación $entrar(n)$ cuando quieren entrar en la nave *n* y la operación $salir(n)$ cuando quieran salir de ella.

Dichas operaciones serán ejecutadas por dichos *threads* antes de entrar y antes de salir de las naves.

Se pide completar la especificación de un recurso compartido *ControlFactoria* que controle el acceso de los robots a las naves:

C-TAD ControlFactoria

OPERACIONES

ACCIÓN entrar: *TipoNave [e]*
ACCIÓN salir: *TipoNave [e]*

SEMÁNTICA
DOMINIO:

TIPO: *ControlFactoria* =

TIPO: *TipoNave* = $\{0, 1\}$

INVARIANTE:

INICIAL:

PRE:

CPRE:

entrar(n)

POST:

PRE:

CPRE:

salir(n)

POST:

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