Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 1 - Clave a

Concurrencia

2011-2012 - Segundo semestre

Lenguajes, Sistemas Informáticos e Ingeniería de Software

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. Asumiendo que la variable `t` contiene una referencia a un primer *thread* que ya ha terminado y que un segundo *thread* ejecuta `t.join()`, **se pide** señalar la respuesta correcta.

(a) El segundo *thread* se parará en la ejecución de `t.join()` hasta que el *thread* referenciado por `t` le envíe una señal.

(b) El segundo *thread* no se parará en la ejecución de `t.join()`

(1½ puntos) 2. 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) { }
            n++;
            en_sc_inc = false;
         }
      }
   }
```

```
   static class Decrementador
      extends Thread {
      public void run () {
         for (int i = 0;
              i < N_PASOS;
              i++) {
            while (en_sc_inc) { }
            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.

(1 punto) 3. Supongamos un programa concurrente con procesos (al menos uno) que ejecutan repetidamente operaciones $r.reintegro(x)$ y procesos (al menos uno) que ejecutan repetidamente operaciones $r.ingreso(y)$, siendo r un recurso compartido del tipo especificado a continuación:

C-TAD CuentaBancaria
OPERACIONES
ACCIÓN reintegro: $\mathbb{N}$*[e]*
ACCIÓN ingreso: $\mathbb{N}$*[e]*
SEMÁNTICA
DOMINIO:
TIPO: *CuentaBancaria* = $\mathbb{N}$
INICIAL: self $= 0$

CPRE: $c \leq$ self
reintegro(c)
POST: self $=$ self$^{pre} - c$

CPRE: Cierto
ingreso(n)
POST: self $- n =$ self^{pre}

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 3. **Se pide** señalar la respuesta correcta.

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

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

(1½ puntos) 5. Obsérvese la siguiente implementación del recurso compartido CuentaBancaria especificado en la pregunta 3:

```
class CuentaBancaria {
   private Semaphore saldo = new Semaphore(0);
   private Semaphore atomic = new Semaphore(1);

   public void reintegro(int c) {
      atomic.await();
      for (int i = 0; i < c; i++)
         saldo.await();
      atomic.signal();
   }

   public void ingreso(int c) {
      for (int i = 0; i < c; i++)
         saldo.signal();
   }
}
```

La idea principal consiste en que el semáforo `saldo` represente el valor interno (de tipo $\mathbb{N}$) del recurso. Asumiendo que se quiere atender a los procesos que quieren realizar reintegros 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.

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

(a) El acceso a un atributo no estático y privado de tipo **`int`** de un *thread* desde su método `run` nunca es una sección crítica.

(b) El acceso a un atributo no estático y privado de tipo **`int`** de un *thread* desde su método `run` puede ser una sección crítica.

[1] En clase se emitió una nota aclaratoria para avisar que no habría otros métodos más allá del método `run`

Apellidos: **Nombre:** **Matrícula:**

(3 puntos) 7. 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: *b.dejarDato(d); b.esperarExtraer();*. Para la extracción de datos se mantiene una única operación a la que se llama de esta forma: *b.extraer(d)*.

Se pide completar la especificación.

Nota: en el dominio, además de una componente para almacenar el dato y un booleano para saber si hay o no hay dato, será necesario mantener como información si el dato ha sido o no extraído.

C-TAD BufferSinc

OPERACIONES

ACCIÓN dejarDato: *Dato[i]*
ACCIÓN esperarExtraer:
ACCIÓN extraer: *Dato[o]*

SEMÁNTICA
DOMINIO:

TIPO: *BufferSinc* =

INVARIANTE:

INICIAL:

CPRE:

dejarDato(d)

POST:

CPRE:

extraer(d)

POST:

CPRE:

esperarExtraer()

POST:

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