Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 2 - Clave a

Concurrencia

2013-2014 - Primer semestre

Grado en Ingeniería Informática. Universidad Politécnica de Madrid

Normas

Este es un cuestionario que consta de **4 preguntas** en **5 páginas**. Las dos primeras preguntas son **de respuesta simple** y las otras dos son **preguntas de desarrollo**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora**. El examen debe contestarse en **estas mismas hojas**. No olvidéis rellenar **apellidos, nombre y número de matricula** en cada hoja.

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. Un recurso que encapsula una cuenta bancaria compartida va a ser implementado con métodos *synchronized*. La operación *Reintegro* se especifica de la manera siguiente:

CPRE: $self.Saldo >= c$
Reintegro(c)
POST: $self.Saldo = self^{pre}.Saldo - c$

La operación *Reintegro* se ha implementado mediante el código siguiente:

```
public synchronized void reintegro(int c) {
   // traduccion CPRE
   if (c > this.saldo) {
     try {
       wait();
     } catch (Exception e) {}
   }
   // traduccion POST
   this.saldo = this.saldo - c;
   // codigo desbloqueos
   notifyAll();
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación correcta de la operación *Reintegro* usando métodos *synchronized*, `wait()` y `notifyAll()`.

(b) Ese código puede provocar que se ejecute una operación cuya CPRE no se cumple.

(1 punto) 2. Supóngase que una condición de sincronización (*CPRE*) de una operación *Op* de un recurso compartido depende del estado del recurso y de un parámetro de entrada (x). Supóngase que dicho recurso va a ser implementado con monitores y que la operación va a ser llamada a lo sumo por un único proceso.

Se pide señalar la respuesta correcta:

(a) Para implementar la sincronización condicional de *Op* es necesario crear una variable `Cond` por cada posible valor de x.

(b) Es posible implementar la sincronización condicional de *Op* con una única variable `Cond`.

(4 puntos) 3. Supongamos un programa concurrente con procesos (al menos uno) que ejecutan repetidamente operaciones $r.inc(x)$ con x positivo y $x < N/2$ y procesos (al menos uno) que ejecutan repetidamente operaciones $r.dec(y)$ con y positivo e $y < N/2$, 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: self = 0

PRE: $n > 0 \wedge n < N/2$
CPRE: self + $n \leq N$
inc(n)
POST: self = self^{pre} + n

PRE: $n > 0 \wedge n < N/2$
CPRE: $n \leq$ self
dec(n)
POST: self + n = self^{pre}

Se pide implementar el recurso compartido en Java utilizando como mecanismo de concurrencia las clases `Monitor` y `Monitor.Cond` de la librería `es.upm.babel.cclib`.

Nota: se pueden ignorar los problemas de inanición a la hora de implementar, es decir, la implementación deberá bloquear a cualquier proceso que invoque una operación cuya CPRE no se cumple y al terminar de ejecutar una operación deberá desbloquearse un proceso (no importa cuál) siempre y cuando algún proceso pueda ser desbloqueado.

(Responder en esta y en la siguiente página).

Apellidos: **Nombre:** **Matrícula:**

(4 puntos) 4. En un programa concurrente hay dos tipos de procesos que acceden a una base de datos: los procesos lectores y los procesos escritores. Muchos procesos pueden leer de la base de datos simultáneamente pero cuando un proceso quiere escribir sobre la base de datos ningún otro proceso puede realizar ninguna operación sobre la misma.

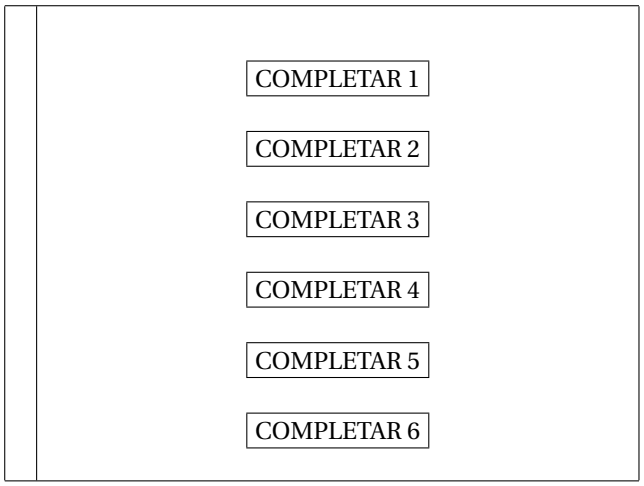
Para asegurar que el programa cumple esta propiedad, se ha diseñado un recurso compartido que controla el acceso de los procesos a la base de datos. Los procesos lectores ejecutan la operación *IL* antes de leer y la operación *FL* después de hacerlo mientras que los procesos escritores ejecutan la operación *IE* antes de escribir y la operación *FE* después de hacerlo.

La especificación formal de dicho recurso compartido es la siguiente:

TIPO: $LE = (l : \mathbb{Z} \times e : \mathbb{Z})$
INVARIANTE: $\forall\, r \in LE \bullet\ r.e \geq 0 \wedge r.l \geq 0 \wedge r.e \leq 1 \wedge ((r.e > 0 \Rightarrow r.l = 0) \wedge (r.l > 0 \Rightarrow r.e = 0))$
INICIAL: self $= (0,0)$
CPRE: self.$e = 0$
IL()
POST: self $= (\text{self}^{pre}.l + 1, \text{self}^{pre}.e)$
CPRE: self.$e = 0 \wedge$ self.$l = 0$
IE()
POST: self $= (\text{self}^{pre}.l, \text{self}^{pre}.e + 1)$
CPRE: *cierto*
FL()
POST: self $= (\text{self}^{pre}.l - 1, \text{self}^{pre}.e)$
CPRE: *cierto*
FE()
POST: self $= (\text{self}^{pre}.l, \text{self}^{pre}.e - 1)$

En la página siguiente se muestra una implementación incompleta del recurso utilizando los mecanismos de sincronización de JCSP. Dicha implementación se ha realizado siguiendo la metodología vista en clase.

Se pide completar las partes señaladas con cajas COMPLETAR i (podría **no** ser necesario escribir código en todas ellas).



(Puedes continuar aquí la solución a la pregunta de desarrollo.)