Apellidos: **Nombre:** **Matrícula:**

Prueba final, parte 2 - Clave a

Concurrencia

2011-2012 - Segundo semestre

Dpto. de Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **5 preguntas** en **4 páginas**. Todas las preguntas son **preguntas de respuesta simple** excepto la pregunta 5 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, además de la **clave** (a o b).

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

(1½puntos) 1. Supóngase un objeto `c` de una clase `C` en la que se han definido los métodos `o` y `p`, declarados como **`synchronized`**. Supóngase también que un thread `t` entra en un bucle infinito *mientras* ejecuta el código de `c.o()`.

Se pide señalar la respuesta correcta.

(a) Cualquier otro thread distinto de `t` que intente ejecutar `c.p()` quedará bloqueado para asegurar la exclusión mutua.

(b) Cualquier otro thread distinto de `t` que intente ejecutar `c.p()` no quedará bloqueado porque la exclusión mutua afecta sólo a las llamadas a `c.o()`.

(c) Cualquier otro thread distinto de `t` que intente ejecutar `c.p()` no quedará bloqueado porque la exclusión mutua no se tiene en cuenta en el caso de un bucle infinito.

(1½puntos) 2. Supóngase que dos threads ejecutan los siguientes códigos:

```
while (true) {
   S11;
   ch1.in.read();
   S12;
   ch2.out.write(null);
   S13;
}
```

```
while (true) {
  ch1.out.write(null);
  ch2.in.read();
  S2;
}
```

Suponemos que no hay otros threads que afecten al problema aparte de los mostrados. **Se pide** señalar la respuesta correcta.

(a) `S11` y `S2` no pueden ejecutar simultáneamente.

(b) `S12` y `S2` no pueden ejecutar simultáneamente.

(c) `S13` y `S2` no pueden ejecutar simultáneamente.

(1½puntos) 3. ¿Debería permitirse a un *thread* invocar una operación de `await` sobre un objeto de clase `Monitor.Cond` generado a partir de un objeto de la clase `Monitor` sin previamente haber invocado el método `enter`?

(a) Sí.

(b) No.

($1\frac{1}{2}$puntos) 4. Partimos de la siguiente especificación formal de un recurso compartido:

C-TAD Binario

OPERACIONES

ACCIÓN A:
ACCIÓN B:
ACCIÓN C:

SEMÁNTICA
DOMINIO:
TIPO: *Binario* = $(p : \mathbb{B} \times q : \mathbb{B})$

INICIAL: $self.p \wedge self.q$

CPRE: Cierto
A()
POST: $self.p \wedge (self.q = self^{pre}.q)$

CPRE: Cierto
B()
POST: $\neg self.p \wedge self.q$

CPRE: $self.p \wedge self.q$
C()
POST: $\neg self.p \wedge \neg self.q$

Alguien ha decidido implementarlo mediante monitores de la siguiente manera:

```
class Binario {
    private boolean p = true;
    private boolean q = true;
    private Monitor mutex = new Monitor();
    private Monitor.Cond cond_p = mutex.newCond();
    private Monitor.Cond cond_q = mutex.newCond();

    void a() {
        mutex.enter();
        p = true;
        cond_p.signal();
        mutex.leave();
    }
    void b() {
        mutex.enter();
        p = false;
        q = true;
        cond_q.signal();
        mutex.leave();
    }
    void c() {
        mutex.enter();
        if (!p) { cond_p.await(); }
        if (!q) { cond_q.await(); }
        p = false;
        q = false;
        mutex.leave();
    }
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación correcta del recurso compartido.

(b) El código puede provocar que se ejecute `c()` en un estado en el que no se cumple su CPRE.

(c) El código de `c` está incompleto: falta el código de desbloqueos.

Apellidos: **Nombre:** **Matrícula:**

(4puntos) 5. Una pequeña variación del típico problema del *productor-buffer-consumidor* es el llamado "buffer de pares e impares" que tenéis en los apuntes de especificación de recursos. La idea es que en la operación de extracción se permite decir si queremos retirar un número par o un número impar.

Se pide implementar el recurso compartido usando paso de mensajes síncrono, mediante la librería JCSP, a partir de la siguiente especificación:

C-TAD BufferPI

OPERACIONES
ACCIÓN Poner: *Tipo_Dato[e]*
ACCIÓN Tomar: *Tipo_Paridad[e]* $\times$ *Tipo_Dato[s]*

SEMÁNTICA
DOMINIO:
TIPO: *Tipo_Buffer_PI* = Secuencia*(Tipo_Dato)*
Tipo_Paridad = *par*|*impar*
Tipo_Dato = $\mathbb{N}$
INVARIANTE: $\forall\, b \in$ *Tipo_Buffer_PI* $\bullet$ Longitud$(b) \leq$ *MAX*

DONDE: $MAX = ...$
INICIAL: Longitud$(self) = 0$

***CPRE*:** *El buffer no está lleno*
CPRE: Longitud$(self) <$ *MAX*
Poner(d)
***POST*:** *Añadimos un elemento al buffer*
POST: $l =$ Longitud$(self^{pre}) \wedge$ Longitud$(self) = l+1 \wedge self(l+1) = d^{pre} \wedge self(1..l) = self^{pre}$

***CPRE*:** *El buffer no está vacío y el primer dato preparado para salir es del tipo que requerimos*
CPRE: Longitud$(self) > 0 \wedge$ *Concuerda*$(self(1), t)$
DONDE: *Concuerda*$(d,t) \equiv (d \bmod 2 = 0 \leftrightarrow t = par)$
Tomar(t, d)
***POST*:** *Retiramos el primer elemento del buffer*
POST: $l =$ Longitud$(self^{pre}) \wedge self^{pre}(1) = d \wedge self = self^{pre}(2..l)$

La solución consistirá en una clase `Gestor_PI` con la siguiente estructura:

```
static class Gestor_PI implements CSProcess {
  private static final int   PAR = 0;
  private static final int IMPAR = 1;
  // A: declaraciones
  public Gestor_PI() {
    // B: inicializacion
  }
  public void poner(int d) {
    // C: op. poner
  }
  public int tomar(int t) {
    // D: op. tomar
  }
  public void run() {
    // E: cod. servidor
  }
}
```

(Escribe aquí la solución a la pregunta de desarrollo.)