Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 2 (4F1M) - Clave a

Concurrencia

2010-2011 - Segundo semestre

Dpto. de Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **3 preguntas de respuesta simple** y **una pregunta de desarrollo** en **6 páginas**. La puntuación total del examen es de **10 puntos**. La duración total es de **una hora**. El examen debe contestarse en las **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

(2 puntos) 1. A continuación se muestra una implementación del recurso
C-TAD Alarma

OPERACIONES
ACCIÓN Notificar: $Alarma[io] \times \mathbb{Z}[i]$
ACCIÓN Detectar: $Alarma[io] \times \mathbb{Z}[i] \times \mathbb{Z}[i] \times \mathbb{Z}[o]$

SEMÁNTICA
DOMINIO:
TIPO: Alarma $= \mathbb{Z}$
INICIAL(a)**:** $a = 23$

CPRE: Cierto
Notificar(a,t)
POST: $a = t$

PRE: $min + 1 < max$
CPRE: $min < a \wedge a < max$
Detectar(a,min,max,t)
POST: $t = a^{pre} \wedge a = a^{pre}$

utilizando los métodos *synchronized*, `wait` y `notify` de Java:

```
class Alarma {
  private int a = 23;

  public Alarma() { }

  public synchronized void
    notificar(int t) {
      a = t;
      notify();
    }
```

```
  public synchronized int
     detectar(int min, int max) {
        if (!(min < a && a < max)) {
           try {
              wait();
           } catch (Exception e) { }
        }
        return a;
     }
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación correcta del recurso.

(b) Puede provocar interbloqueo.

(c) Hay que proteger el acceso a *a* mediante un semáforo binario.

(d) Puede provocar la ejecución de una operación cuando su condición de sincronización (*CPRE*) es falsa.

(2 puntos) 2. Un recurso que encapsula una cuenta bancaria compartida va a ser implementado con *locks* y *conditions*. Las operaciones son *Reintegro* e *Ingreso*:

CPRE: $q.saldo >= c$
Reintegro(q, c)
POST: $q.saldo = q^{pre}.saldo - c$

CPRE: *Cierto*
Ingreso(q, c)
POST: $q.saldo = q^{pre}.saldo + c$

Al depender la *CPRE* del parámetro de entrada se ha optado por declarar una clase *Peticion* que contiene la cantidad pedida y una variable *condition* y usar una cola para guardar las peticiones pendientes.

El código de la operación *Reintegro* es el siguiente (dejando aparte el manejo de la exclusión mutua):

```
public void reintegro(int c) {
   // traduccion CPRE
   if (c > this.saldo || !pendientes.isEmpty();) {
      Peticion pet = new Peticion(c);
      pendientes.enqueue(pet);
      pet.condition.await();
      pendientes.dequeue();
   }
   // traduccion POST
   this.saldo = this.saldo - c;
   // codigo desbloqueos
   if (!pendientes.isEmpty() &&
      (pendientes.front().cantidad <= this.saldo)) {
           pendientes.front().condition.signal();
   }
}
```

Se pide señalar la respuesta correcta:

(a) Bastaba haber usado una sola variable *condition* si lo que se quería era atender los reintegros por estricto orden de llegada.

(b) Ese código puede provocar inanición de las peticiones de reintegro.

(c) El problema hay que implementarlo con métodos *synchronized* de Java.

(d) La implementación mostrada permitiría ejecutar llamadas a *Reintegro* en casos en que su CPRE no se cumple.

Apellidos: **Nombre:** **Matrícula:**

(2 puntos) 3. Se define una clase *servidor* `P` y dos clases *cliente* `P1` y `P2` que se comunican a través de los canales de comunicación `petA` y `petB` (se muestran las partes relevantes del código para este problema).

```
class P implements CSProcess {
  public void run() {
    // Estado del recurso
    boolean q = true;
    boolean r = true;

    final int A = 0;
    final int B = 1;

    final Guard[] entradas =
      {petA.in(), petB.in()};

    final Alternative servicios =
      new Alternative (entradas);

    final boolean[] sincCond =
      new boolean[2];

    while (true) {
      sincCond[A] = q || r;
      sincCond[B] = q && r;

      switch (servicios.fairSelect(sincCond)) {
        case A:
          petA.in().read();
          q = !q;
          break;
        case B:
          petB.in().read();
          r = !r;
          break;
      }
    }
  }
}
```

```
class P1 implements CSProcess {
  public void run() {
    while (true) {
      petA.out().write(null);
    }
  }
}

class P2 implements CSProcess {
  public void run() {
    while (true) {
      petB.out().write(null);
    }
  }
}
```

Dado un programa concurrente con tres procesos `p`, `p1` y `p2` de las clases `P`, `P1` y `P2`.

Se pide decir cuál de las siguientes afirmaciones es la correcta:

(a) Es seguro que los tres procesos se van a bloquear.

(b) Es seguro que `p2` acabará bloqueándose, pero `p` y `p1` podrían seguir ejecutando indefinidamente.

(c) Es posible que `p1` se bloquee pero en ese caso `p` y `p2` podrían seguir ejecutando indefinidamente.

(d) Ninguna de las respuestas anteriores.

(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(r): $r = (0, 0)$

CPRE: $r.e = 0$
IL(r)
POST: $r = (r^{pre}.l + 1, r^{pre}.e)$

CPRE: $r.e = 0 \wedge r.l = 0$
IE(r)
POST: $r = (r^{pre}.l, r^{pre}.e + 1)$

CPRE: *cierto*
FL(r)
POST: $r = (r^{pre}.l - 1, r^{pre}.e)$

CPRE: *cierto*
FE(r)
POST: $r = (r^{pre}.l, r^{pre}.e - 1)$

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).

Apellidos: **Nombre:** **Matrícula:**

```
class LECSP
   implements CSProcess {

   // Canales de comunicación
   private Any2OneChannel chIL =
      Channel.any2one();
   private Any2OneChannel chFL =
      Channel.any2one();
   private Any2OneChannel chIE =
      Channel.any2one();
   private Any2OneChannel chFE =
      Channel.any2one();

   public LECSP() {
   }

   // Ejecutado por el cliente
   public void il() {
      chIL.out().write(null);
   }

   // Ejecutado por el cliente
   public void fl() {
      chFL.out().write(null);
   }

   // Ejecutado por el cliente
   public void ie() {
      chIE.out().write(null);
   }

   // Ejecutado por el cliente
   public void fe() {
      chFE.out().write(null);
   }
```

```
//  Código del servidor
public void run() {
   // Estado del recurso
   int l = 0;
   int e = 0;

   // Preparando la recepción
   // no determinista
   final int IL = 0;
   final int FL = 1;
   final int IE = 2;
   final int FE = 3;
   Guard[] entradas =
      {chIL.in(),
       chFL.in(),
       chIE.in(),
       chFE.in()};
   Alternative servicios =
      new Alternative(entradas);
   boolean[] sincCond =
      new boolean[4];

   // Bucle principal del servidor
   while (true) {
      // Establecer sincCond para
      // implementar las CPRE
      sincCond[FL] = true;
      sincCond[FE] = true;

            COMPLETAR 1


      // Recepción no determinista
      switch (
         servicios.fairSelect(sincCond)
      ) {
      case IL:

            COMPLETAR 2

         break;
      case FL:

            COMPLETAR 3

         break;
      case IE:

            COMPLETAR 4

         break;
      case FE:

            COMPLETAR 5

         break;
      }
      // Código de desbloqueo

            COMPLETAR 6

   }
}
```

(Escribe en la siguiente página la solución a la pregunta de desarrollo.)

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