Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 2 (4F2M) - 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. Un recurso que encapsula una cuenta bancaria compartida va a ser implementado con *Locks* y *Conditions*. La operación *Reintegro* se especifica de la manera siguiente:

CPRE: $q.Saldo >= c$
Reintegro(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) {
      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*.

(b) Ese código puede provocar inanición.

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

(d) Es un desbloqueo en *cascada* que sigue la metodología vista en clase.

(2 puntos) 2. A continuación se muestra una implementación de este recurso:

TIPO: Factoría $= \acute{A}rea \nrightarrow \mathbb{N}$
DONDE: $\acute{A}rea = \{0, 1\}$
INVARIANTE: $\forall f \in Factoría.\ (\forall\, a \in \acute{A}rea.\, f(a) \leq 2) \wedge \sum_{a \in \acute{A}rea} f(a) < 4$

INICIAL(f): $\sum_{a \in \acute{A}rea} f(a) = 0$

CPRE: $f(a) < 2 \wedge \sum_{a \in \acute{A}rea} f(a) < 3$
Entrar(f,a)
POST: $f = f^{pre} \oplus \{a \mapsto f^{pre}(a) + 1\}$

CPRE: Cierto
Salir(f,a)
POST: $f = f^{pre} \oplus \{a \mapsto f^{pre}(a) - 1\}$

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

```
class Factoria {
  private int dentro[] = {0, 0};
  private int total = 0;

  public Factoria() { }

  public synchronized void
    salir(int a) {
      dentro[a]--; total--;
      notify();
    }

  public synchronized void
    entrar(int a) {
      if (dentro[a]==2 || total==3) {
        try {
          wait();
        }
        catch (Exception e) { }
      }
      dentro[a]++; total++;
    }
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación correcta del recurso.

(b) Puede provocar interbloqueo.

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

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

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) Existe un riesgo de inanición de `p1`.

(b) Existe un riesgo de interbloqueo.

(c) Existe un riesgo de inanición de `p2`.

(d) Ninguna de las respuestas anteriores.

(4 puntos) 4. Un sistema concurrente controla la temperatura de una nave industrial. Para ello un proceso monitoriza la temperatura y notifica cambios en las mismas mientras que otros procesos detectan que dicha temperatura excede un determinado rango, diferente para cada proceso.

La comunicación entre los procesos se realiza mediante el siguiente recurso compartido:

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: $t = a^{pre} \wedge a = a^{pre}$

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 $\boxed{\text{COMPLETAR } i}$ (podría **no** ser necesario escribir código en todas ellas).

Apellidos: **Nombre:** **Matrícula:**

```
class AlarmaCSP
   implements CSProcess {

   // Canales de comunicación
   private
      Any2OneChannelInt chNotificar =
        Channel.any2oneInt();
   private Any2OneChannel chDetectar =
      Channel.any2one();

   public AlarmaCSP() {
   }

   // Ejecutado por el cliente
   public void notificar(int t) {
      chNotificar.out().write(t);
   }

   // Ejecutado por el cliente
   public int detectar(int min,
                       int max) {
      One2OneChannelInt sincro =
         Channel.one2oneInt();
      Object[] pet =
         {new Integer(min),
          new Integer(max),
          sincro};
      chDetectar.out().write(pet);
      return sincro.in().read();
   }
```

```
//  Código del servidor
public void run() {
   // Estado del recurso
   int a = 23;

   // Cola de bloqueados
   Queue<Object[]> esperanDetectar =
      new NodeQueue<Object[]>();

   // Preparando la recepción
   // no determinista
   final int NOTIFICAR = 0;
   final int DETECTAR  = 1;
   Guard[] entradas =
      {chNotificar.in(),
       chDetectar.in()};
   Alternative servicios =
      new Alternative(entradas);

   // Variables auxiliares
   int min, max;
   Object[] pet;
   One2OneChannel chResp;

   // Bucle principal del servidor
   while (true) {
      // Recepción no determinista
      switch (
         servicios.fairSelect()
      ) {
      case NOTIFICAR:

            COMPLETAR 1

         break;
      case DETECTAR:

            COMPLETAR 2

         break;
      }
      // Código de desbloqueo

            COMPLETAR 3

   }
}
```

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

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