Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 2 - Clave a

Concurrencia

2010-2011 - Primer semestre

Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **4 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**. 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

(2 puntos) 1. Dado el siguiente **CTAD** (sólo se muestran las partes necesarias):

TIPO: Factoría $= \acute{A}rea \nrightarrow \mathbb{N}$
DONDE: $\acute{A}rea = \{0, 1\}$
INVARIANTE: $\forall f \in Factoría.\ (\forall\, a \in \acute{A}rea.\, f(a) \leq 10) \wedge \sum_{a \in \acute{A}rea} f(a) < 20$

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

CPRE: $f(a) < 10 \wedge \sum_{a \in \acute{A}rea} f(a) < 19$
Entrar(f,a)
POST: $f = f^{pre} \oplus \{a \mapsto f^{pre}(a) + 1\}$

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

Suponemos un programa concurrente en el que los procesos respetan el siguiente protocolo (o esquemas de llamada): $Entrar(f, a);\ \ldots;\ Salir(f, a)$.

Lo que sigue es parte (cliente y servidor) de una implementación de este recurso usando paso de mensajes:

```
// Canales de comunicación
private Any2OneChannel chEntrar = Channel.any2one();
private Any2OneChannelInt chSalir = Channel.any2oneInt();
```

```
// Ejecutado por el cliente
public void entrar(int a) {
  One2OneChannel sincro =
    Channel.one2one();
  Object[] pet =
    {new Integer(a),sincro};
  chEntrar.out().write(pet);
  sincro.in().read();
}

public void salir(int a) {
  chSalir.out().write(a);
}

//  Código del servidor
public void run() {
  // Estado del recuros compartido
  int dentro[] = {0, 0};
  int total = 0;

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

  // Preparando la recepción
  // no determinista
  final int ENTRAR = 0;
  final int SALIR = 1;
  Guard[] entradas =
    {chEntrar.in(), chSalir.in()};
  Alternative servicios =
    new Alternative(entradas);
  boolean[] sincCond =
    new boolean[2];

  // Variables auxiliares
  int a;
  Object[] pet;
  One2OneChannel chResp;
```

```
  // Bucle principal del servidor
  while (true) {
    sincCond[ENTRAR] = total < 19;
    sincCond[SALIR] = true;

    // Aceptamos y almacenamos peticiones
    switch (servicios.fairSelect()) {
    case ENTRAR:
      pet = (Object[])chEntrar.in().read();
      a = ((Integer)pet[0]).intValue();
      chResp = ((One2OneChannel)pet[1]);
      if (dentro[a] < 10) {
        dentro[a]++; total++;
        chResp.out().write(null);
      }
      else
        esperanEntrar.enqueue(pet);
      break;
    case SALIR:
      a = chSalir.in().read();
      dentro[a]--;  total--;
      break;
    default:
      break;
    }
    int n = esperanEntrar.size();
    for (int i = 0; i < n; i++) {
      pet = esperanEntrar.front();
      a = ((Integer)pet[0]).intValue();
      chResp = ((One2OneChannel)pet[1]);
      if (dentro[a] < 10) {
        dentro[a]++; total++;
        chResp.out().write(null);
      }
      else {
        esperanEntrar.dequeue();
        esperanEntrar.enqueue(pet);
      }
    }
  }
}
```

Se pide señalar la respuesta correcta.

(a) La expresión `chResp.out().write(null)` no compila.

(b) Es una implementación correcta del recurso compartido.

(c) La operación `entrar` puede atenderse sin que se cumpla su *CPRE*.

(d) Pueden quedar procesos bloqueados en `esperanEntrar` a pesar de que su *CPRE* se cumple.

Apellidos: **Nombre:** **Matrícula:**

(2 puntos) 2. 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) 3. A continuación se muestra una implementación del recurso de la pregunta 1 utilizando los métodos *synchronized*, `wait` y `notify` de Java:

```
static 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]==10 || total==20) {
        try {
          wait();
        }
        catch (Exception e) { }
      }
      dentro[a]++; total++;
    }
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación correcta del recurso.

(b) Provoca interbloqueos.

(c) Provoca inanición.

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

(4 puntos) 4. Completa la implementación de la operación *insertar* del *Multibuffer* usando *locks* y *conditions*:

```
import net.datastructures.NodeQueue;
import net.datastructures.Queue;
import net.datastructures.Dictionary;
import net.datastructures.BinarySearchTree;
import net.datastructures.Entry;
import java.util.concurrent.locks.*;

/**
 * Las instancias de Multibuffer son buffers de tamanno
 * limitado cuyas operaciones pueden ser ejecutadas por varios threads
 * sin que se produzcan condiciones de carrera.
 */
public final class MultiBufferLocks<E> implements MultiBuffer<E> {
    // La exclusion mutua ya no la hacemos con synchronized
    final Lock mutex = new ReentrantLock();
    // La sincronizacion condicional va a ser explicita,
    // con dos variables "condition"
    final Condition haySitio = mutex.newCondition();
    final Condition hayDatos = mutex.newCondition();
    // Estado derivado del recurso compartido
    private Queue<E> cola = new NodeQueue<E>();
    private int maxDatos; // se establece en la constructora estandar

    // Estado para poder controlar la sincronizacion condicional.
    // Para ello es necesario recordar cada uno de los bloqueos con
    // los datos necesarios para provocar los desbloqueos
    // Usaremos una cola por cada tipo de peticion para
    // Realizar una politica de atencion por orden de llegada
    private Queue<Thread> esperanInsertar =
        new NodeQueue<Thread>();
    private Queue<Thread> esperanExtraer =
        new NodeQueue<Thread>();
    // Tambien necesitamos recordar el valor de las peticiones, por el
    // esquema de desbloqueo solidario
    private Queue<Integer> peticionesInsertar =
        new NodeQueue<Integer>();
    private Queue<Integer> peticionesExtraer =
        new NodeQueue<Integer>();

    private MultiBufferLocks () {
    }

    public MultiBufferLocks (int maxDatos) {
        this.maxDatos = maxDatos;
    }

    public int maxDatos() {
        // todas las operaciones en exclusion mutua
        mutex.lock();
        try {
            return maxDatos;
        } finally {
            mutex.unlock();
        }
    }

    public void insertar(int n, E[] datos) {
      // COMPLETAR
    }

    // [...]

}
```

Apellidos: **Nombre:** **Matrícula:**

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

(Página intencionadamente en blanco.)