Apellidos: **Nombre:** **DNI/NIE:**

Prueba final, parte 2 - Clave a

Concurrencia

2013-2014 - Segundo semestre

Universidad Politécnica de Madrid

Normas

Este es un cuestionario que consta de **6 preguntas** en **5 páginas**. Todas las preguntas son **preguntas de respuesta simple** excepto la pregunta 6 que es una **pregunta de desarrollo**. La puntuación total del examen es de **10 puntos**, habiéndose de conseguir un **mínimo de 1 punto** en la pregunta de desarrollo. 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. No olvidéis rellenar **apellidos, nombre y DNI o NIE** 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 dividida por el número de alternativas ofrecidas en la misma.

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 = $\mathbb{Z}$
INVARIANTE: $0 \leq \text{self} \wedge \text{self} < 50$

INICIAL: $self = 23$

PRE: $0 \leq t \wedge t < 50$
CPRE: Cierto
Notificar(t)
POST: $self = t$

PRE: $0 \leq min \wedge min \leq max \wedge max < 50$
CPRE: $min \leq self \wedge self \leq max$
Detectar(min,max,t)
POST: $t = self^{pre} \wedge self = self^{pre}$

Figura 1: Especificación formal del recurso *Alarma*.

Cuestionario

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

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

  public AlarmaSync() { }

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

  public synchronized int
     detectar(int min, int max) {
        while (a < min || a > max) {
           try {
              wait();
           } catch (Exception e) { }
        }
        return a;
     }
}
```

Se pide señalar la respuesta correcta. Asumiendo que los procesos que llaman a los métodos `notificar` y `detectar` respetan siempre sus respectivas PREs:

(a) Es una implementación correcta del recurso.

(b) No es una implementación correcta del recurso.

(1 punto) 2. Supóngase un recurso compartido con una operación cuya sincronización condicional (CPRE) depende de un parámetro de entrada. Se quiere programar dicho recurso con monitores y forzar una atención en estricto orden de llegada para dicha operación. Es decir, si un proceso invoca la operación y su CPRE no se cumple quedará bloqueado y si otros procesos intentan ejecutar la misma operación también quedarán bloqueados (independientemente de que se cumpla su CPRE o no).

Se pide señalar la respuesta correcta:

(a) No es posible implementar dicha operación utilizando una única *condition*.

(b) Es posible implementar dicha operación utilizando una única *condition*.

(1 punto) 3. A continuación se muestra una implementación del recurso de la figura 1 utilizando monitores. Como podéis ver, el programador se ha saltado nuestra metodología y ha programado un bucle de signals:

```
class AlarmaMon {
  private Monitor mutex =
    new Monitor();
  private Monitor.Cond c[][] =
    new Monitor.Cond[50][50];
  private int a = 23;
  public AlarmaMon() {
    for (int min = 0; min < 50; min++)
      for (int max = 0; max < 50; max++)
        c[min][max] = mutex.newCond();
  }
  public void notificar(int t) {
    mutex.enter();
    a = t;
    for (int min = 0; min < 50; min++)
      for (int max = 0; max < 50; max++)
        if (min <= a && a <= max)
          c[min][max].signal();
    mutex.leave();
  }
```

```
  public int
    detectar(int min, int max) {
      int t;
      mutex.enter();
      if (a < min || a > max) {
        try {
          c[min][max].await();
        } catch (Exception e) { }
      }
      t = a;
      mutex.leave();
      return t;
  }
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación *insegura*: procesos pueden ejecutar operaciones sin cumplirse su CPRE.

(b) Es una implementación que sufre de *esperas innecesarias*: procesos cuya CPRE se cumple pueden esperar más tiempo de lo necesario.

(1 punto) 4. Dado el siguiente código de dos threads que interactúan con paso de mensajes síncrono:

```
static private Any2OneChannel c1 = Channel.any2one();
```

```
static class A implements CSProcess {
  public void run() {
    c1.out().write("A");
    System.out.print("1");
    c1.out().write("B");
    System.out.print("2");
  }
}
```

```
static class B implements CSProcess {
  public void run() {
    String s;
    s = (String) c1.in().read();
    System.out.print(s);
    s = (String) c1.in().read();
    System.out.print(s);
  }
}
```

```
  // Programa principal
  CSProcess sistema = new Parallel(new CSProcess[] {new A(), new B()});
  sistema.run();
```

Se pide: señalar la respuesta correcta:

(a) `12AB` es una salida posible del programa.

(b) `12AB` no es una salida posible del programa.

Apellidos: **Nombre:** **DNI/NIE:**

(1 punto) 5. A continuación se muestra una implementación del recurso de la figura 1 utilizando JCSP:

```
class AlarmaCSP implements CSProcess {
  private Any2OneChannelInt chNotificar;
  private Any2OneChannel[][] chDetectar;
  public AlarmaCSP() {
    chNotificar = Channel.any2oneInt();
    chDetectar =
      new Any2OneChannel[50][50];
    for (int min = 0; min < 50; min++)
      for (int max = min; max < 50; max++)
        chDetectar[min][max] =
          Channel.any2one();
  }
  public void notificar(int t) {
    chNotificar.out().write(t);
  }
  public int detectar(int min, int max) {
    One2OneChannelInt c =
      Channel.one2oneInt();
    chDetectar[min][max].out().write(c);
    return c.in().read();
  }
```

```
  public void run() {
    Guard[] entradas = new Guard[50*50+1];
    for (int min = 0; min < 50; min++)
      for (int max = min; max < 50; max++)
        entradas[50*min+max] =
          chDetectar[min][max].in();
    entradas[50*50] = chNotificar.in();
    Alternative servicios =
      new Alternative (entradas);
    int a = 23;
    while (true) {
      boolean[] sincCond =
        new boolean[50*50+1];
      for (int min = 0; min < 50; min++)
        for (int max = min; max < 50; max++)
          sincCond[50*min+max] =
            min <= a && a <= max;
      sincCond[50*50] = true;
      int entrada =
        servicios.fairSelect(sincCond);
      switch (entrada) {
      case 50*50:
        a = chNotificar.in().read();
        break;
      default:
        int min = entrada / 50;
        int max = entrada % 50;
        One2OneChannel c =
          (One2OneChannel)
          chDetectar[min][max].in().read();
        c.out().write(a);
        break;
      }
    }
  }
}
```

Se pide señalar la respuesta correcta:

(a) Es una implementación correcta del recurso.

(b) No es una implementación correcta del recurso.

(5 puntos) 6. Se pide implementar con monitores, siguiendo la metodología enseñada en clase (sin ejecutar más de un await/signal por llamada a método, etc.), el recurso compartido de la figura 1.

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