Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 1)/clave: a

Curso 2015–2016 - 2º Semestre (Mayo 2016)
Grado en Ingeniería Informática / Grado en Matemáticas e Informática
UNIVERSIDAD POLITÉCNICA DE MADRID

NORMAS: Este es un cuestionario que consta de 7 preguntas. Todas las preguntas son de respuesta simple excepto la pregunta 7 que es una pregunta de desarrollo. La puntuación total del examen es de $9\frac{1}{2}$ puntos. La duración total es de una hora. El examen debe contestarse en las hojas de respuestas. No olvidéis rellenar apellidos, nombre y número de matrícula 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

(1 punto) 1. Suponiendo que hay tres procesos que ejecutan repetidamente las operaciones *uno*, *dos*, *tres* del CTAD que se especifica a continuación:

TIPO: MiCTAD $= (a : \mathbb{B} \times b : \mathbb{B})$
INICIAL: $self = (\text{Falso}, \text{Falso})$
INVARIANTE: $\neg self.a \vee \neg self.b$

CPRE: $\neg self.a$
uno()
POST: $self^{pre} = (a, b) \wedge self = (a, \neg b)$
CPRE: $self.a \vee self.b$
dos()
POST: $self^{pre} = (a, b) \wedge self = (\neg a, \neg b)$
CPRE: Cierto
tres()
POST: $self^{pre} = (a, b) \wedge self = (\text{Cierto}, b)$

Se pide marcar la afirmación correcta:

(a) En el programa no podría llegar a violarte el invariante

(b) En el programa podría llegar a violarse la invariante.

(Sugerencia: Dibuja aquí el grafo de los estados por los que puede pasar el recurso.)

(1 punto) 2. Dado el siguiente programa

```
static class Hilos {
  static class MiHilo
    implements Runnable {
    volatile int n = 0;
    public void run () {
      int t1 = n
      n = t1 + 1;
    }
  }
```

```
  public static final void
    main(final String[] args) {

      Thread t1 = new Thread(MiHilo);
      Thread t2 = new Thread(MiHilo);
      Thread t3 = new Thread(MiHilo);
      t1.start();
      t2.start();
      try {t1.join();}
        catch(InterruptedException e){}
      t3.start();
  }
}
```

¿Cuantos procesos puede haber a la vez en ejecución?

(a) 5

(b) 3

(c) 4

(1 punto) 3. Dado el programa concurrente de la pregunta 2.

Se pide marcar la afirmación correcta.

(a) El cuerpo del método `run()` es una sección crítica.

(b) El cuerpo del método `run()` no es una sección crítica.

(1 punto) 4. Dado un programa concurrente en la que dos *threads* instancias de las clases `A` y `B` comparten una variable `n`:

```
static int n = 0;
static Semaphore s1 = new Semaphore(1);
static Semaphore s2 = new Semaphore(0);
```

```
static class A extends Thread {
    public void run() {
        s2.await();
        n = 3 * n;
        s1.signal();
    }
}
```

```
static class B extends Thread {
    public void run() {
        s1.await();
        n = n + 2;
        s2.signal();
    }
}
```

¿Cuál es el valor de `n` tras terminar los dos threads?

(a) 2

(b) 6

(1 punto) 5. Si en el código anterior el semáforo `s2` se inicializa a 1:

(a) No está garantizada la terminación de ambas tareas.

(b) No está garantizada la exclusión mutua en el acceso a `n`.

Apellidos: **Nombre:** **Matrícula:**

(1½ puntos) 6. Se desea modelar con semáforos una barrera para hilos con la cual se bloquearán todos los hilos hasta que haya llegado el último, momento en el cual permitiremos seguir a todos los hilos.

```
static class Hilos{
  static final int MAX_HILOS = 5;
  static class MiHilo
  extends Thread{
    static volatile int cont = 0;
    static Semaphore mutex =
      new Semaphore(1);
    static Semaphore barrera =
      new Semaphore(0);
```

```
    public void run(){
      tarea1();

      mutex.wait();
      cont = cont + 1;
      if (cont == MAX)
        barrera.signal();
      barrera.wait();
      barrera.signal();
      mutex.signal();

      tarea2();
    }
  }
}
```

Supongamos que lanzamos `MAX_HILOS` hilos instancias de `MiHilo`. Asumiendo que los métodos `tarea1()` y `tarea2()` siempre terminan, **se pide** marcar la afirmación correcta.

(a) El programa implementa siempre la barrera tal como se pretendía.

(b) El programa siempre acabará en un interbloqueo.

(c) El programa se comporta como una barrera solo en algunas ejecuciones, dependiendo de las velocidades relativas de los procesos.

(3 puntos) 7. **Se pide** especificar un recurso compartido que mezcla dos secuencias ordenadas de números enteros para formar una única secuencia ordenada. En este recurso interactúan tres procesos: dos productores que van pasando números de sus secuencias de uno en uno y un consumidor que va extrayendo los números en orden.

Cada recurso será capaz de almacenar, como mucho, un dato de la secuencia que llamaremos "izquierda" y un dato de la secuencia "derecha". Cuando hay datos de ambas secuencias la operación *extraerMenor* tomará el menor de ambos y permitirá que se añada un nuevo dato de la secuencia correspondiente. La operación *insertarIzda(d)* inserta el dato *d* como parte de la secuencia izquierda. Bloquea hasta que el hueco para el dato izquierdo está disponible. La operación *insertarDcha* es análoga.

Por simplificar no hemos tenido en cuenta aquí la terminación de las secuencias, que requiere de operaciones y estado adicionales.

C-TAD OrdMezcla

OPERACIONES

ACCIÓN insertarIzda: $\mathbb{Z}$*[e]*
ACCIÓN insertarDcha: $\mathbb{Z}$*[e]*
ACCIÓN extraerMenor: $\mathbb{Z}$*[s]*

SEMÁNTICA
DOMINIO:

TIPO: *OrdMezcla* =

TIPO: *Lado* = *Izda* | *Dcha*

INICIAL:

CPRE:

insertarIzda(d)

POST:

CPRE:

extraerMenor(min)

POST: