Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 1)/clave: b

Curso 2016–2017 - 2º Semestre (mayo 2017)
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 10 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

(1½ puntos) 1. Dado el siguiente **CTAD**[1]

TIPO: Mi-CTAD $= (a : \mathbb{N} \times b : \mathbb{B})$
INICIAL: $self = (0, \text{Cierto})$
INVARIANTE: $0 \leq \text{self}.a \leq 1 \wedge (\text{self} = 1 \vee self.b)$

CPRE: Cierto
uno()
POST: $self^{pre} = (c, d) \wedge self = (c, \text{Cierto})$

CPRE: $self.a = 0 \wedge self.b$
dos()
POST: $self^{pre} = (c, d) \wedge self = (c + 1, \neg d)$

CPRE: self.b
tres()
POST: $self^{pre} = (c, d) \wedge self = (0, d)$

Asumiendo que tenemos tres procesos que invocan repetidamente las operaciones *uno*(), *dos*() y *tres*() del recurso compartido. Se pide marcar la afirmación correcta:

(a) Podría llegar a violarse la invariante.

(b) El sistema podría quedarse bloqueado.

(c) El sistema nunca quedará bloqueado y siempre se cumplirá la invariante.

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

[1] Recordamos a nuestros alumnos del Grado en Matemáticas e Informática que en nuestra notación los naturales incluyen al cero.

(1 punto) 2. Dado el siguiente programa

```
class Hilos {
  static class MiHilo extends Thread {
    private int n;
    public void run () {
      for (int i=0; i<100; i++) {
        n ++;
      }
    }
  }

  public void main (String [] args) {
    Thread t = new MiHilo ();
    t.run();
    t.start();
    System.out.println("Ejecutando main");
    try {t.join();}
    catch (InterruptedException e){}
    System.out.println("n: " + t.n);
    }
} // Fin clase Hilos
```

¿Cuál será la salida por consola al ejecutar el `main`? **Se pide** marcar la afirmación correcta.

(a) 200

(b) No se sabe, porque podría haber condiciones de carrera.

(c) 100

(1 punto) 3. Dado el programa concurrente de la pregunta 2. ¿Cuantos procesos puede haber simultáneamente en ejecución? **Se pide** marcar la afirmación correcta.

(a) 1

(b) 2

(c) 3

(1½ puntos) 4. Dado un programa concurrente en la que tres *threads* instancias de las clases `A`, `B` y `C` comparten una variable `n`:

```
static int n = 0;
static Semaphore s1 = new Semaphore(0);
static Semaphore s2 = new Semaphore(1);
static Semaphore s3 = new Semaphore(0);
```

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

```
static class B
  extends Thread {
  public void run() {
    s3.await();
    n = n * n;
    s2.signal();
  }
}
```

```
static class C
  extends Thread {
  public void run() {
    s2.await();
    n = n + 2;
    s1.signal();
    s3.signal();
  }
}
```

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

(a) 8

(b) 2

(c) 4 u 8

(1 punto) 5. Si en el código anterior los semáforos `s1` y `s2` se inicializan a 0 y el semáforo `s3` a 1, **se pide** marcar la afirmación correcta.

(a) No está garantizada la terminación de las tres tareas.

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

Apellidos: **Nombre:** **Matrícula:**

(1 punto) 6. Dada la siguiente implementación de una solución con *espera activa*.

```
static final int INC = 1;
static final int DEC = 0;
static volatile int cont = 0;
static volatile int turno = INC;
```

```
class Incrementador extends Thread {
  public void run() {
    for (int i = 0; i < N_OPS; i++) {
      while (turno != INC) {};

      // SECCION CRITICA
      cont ++;

      turno = DEC;
    }
  }
}
```

```
class Decrementador extends Thread {
  public void run() {
    for (int i = 0; i < N_OPS; i++) {
      while (turno != DEC) {};

      // SECCION CRITICA
      cont --;

      turno = INC;
    }
  }
}
```

Suponiendo que tenemos un proceso de tipo `Incrementador` y otro proceso `Decrementador`, **se pide** marcar la afirmación correcta.

(a) El programa no garantiza la propiedad de ausencia de inanición.

(b) El programa no garantiza la exclusión mutua en el acceso a la sección crítica (`cont++` y `cont--`).

(c) El programa no garantiza la propiead de ausencia de esperas innecesarias.

(3 puntos) 7. En la especificación formal de un recurso gestor de *lectores/escritores* hay riesgo de inanición de los procesos escritores ya que los lectores podrían monopolizar el acceso al documento compartido adquiriendo y liberando permisos de lectura indefinidamente, sin permitir acceder a ningún escritor.

Se pide: modificar la especificación formal del gestor de lectores/escritores para forzar a que el número de lectores no crezca cuando hay un escritor en espera. Más concretamente, se ha dividido la operación *inicioEscribir* en dos: una primera que declara la intención de escribir por parte de un proceso escritor (*intencionEscribir*) y una segunda que realmente solicita el acceso (*permisoEscribir*). Tendréis que extender el estado del recurso para llevar cuenta de los escritores en espera y modificar el resto de la especificación de acuerdo con estos cambios.

C-TAD GestorLE_2
OPERACIONES
ACCIÓN intencionEscribir:
ACCIÓN permisoEscribir:
ACCIÓN finEscribir:
ACCIÓN inicioLeer:
ACCIÓN finLeer:
SEMÁNTICA
DOMINIO:

TIPO: *GestorLE_2* =

INICIAL:

INVARIANTE:

CPRE:

intencionEscribir()

POST:

CPRE:

permisoEscribir()

POST:

CPRE:

finEscribir()

POST:

CPRE:

inicioLeer()

POST:

CPRE:

finLeer()

POST: