Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 1)/clave: b

Curso 2017–2018 - 2º semestre (mayo 2018)
Grado en Ingeniería Informática / Grado en Matemáticas e Informática /
Doble Grado en Ing. Informática y ADE
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. No olvidéis rellenar vuestros datos.

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**

C-TAD MiCTAD
TIPO: MiCTAD $=$ *Indice* $\rightarrow \mathbb{B}$
TIPO: *Indice* $= \{0, 1\}$
INICIAL: $\forall\, i \in$ *Indice* $\bullet\ \neg\text{self}(i)$
INVARIANTE: $\neg\text{self}(0) \vee \text{self}(1)$

CPRE: $\neg\text{self}(0) \wedge \neg\text{self}(1)$
uno()
POST: *self* $=$ self$^{pre} \oplus \{1 \mapsto$ *Cierto*$\}$

CPRE: $\neg\text{self}(0) \wedge \text{self}(1)$
dos()
POST: *self* $=$ self$^{pre} \oplus \{0 \mapsto$ *Cierto*$\}$

CPRE: self(1)
tres()
POST: *self* $=$ self$^{pre} \oplus \{0 \mapsto$ *Falso*$\} \oplus \{1 \mapsto$ *Falso*$\}$

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 punto) 2. Dado el siguiente programa

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

  public void main (String [] args) {
    Thread t = new MiHilo ();
    t.start();
    t.run();
    hacerAlgo(); // hace algo...
    try {t.join();}
    catch (InterruptedException e){}
    System.out.println(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 puede saber porque podría haber condiciones de carrera.

(c) 100

(1 punto) 3. Suponiendo que los threads de la pregunta 2 terminan normalmente,

(a) El número máximo de procesos ejecutando a la vez será 2 y el método `main` podrá terminar antes que el thread `t`.

(b) El número máximo de procesos ejecutando a la vez será 3 y el método `main` terminará siempre después que el thread `t`.

(c) El número máximo de procesos ejecutando a la vez será 2 y el método `main` terminará siempre después que el thread `t` .

($1\frac{1}{2}$ 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(1);
    static Semaphore s2 = new Semaphore(0);
```

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

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

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

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

(a) 2 o 16

(b) 4

(c) 4 o 16

(1 punto) 5. Si en el código anterior los semáforos `s1` y `s2` se inicializan 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 al problema de la exclusión mutua con *espera activa*:

```
static volatile boolean inc_quiere = false;
static volatile boolean dec_quiere = false;
static volatile int cont = 0;
```

```
class Incrementador extends Thread {
  public void run() {
    for (int i = 0; i < N_OPS; i++) {
      inc_quiere = true;
      while (dec_quiere) {}

      cont++;

      inc_quiere = false;
    }
  }
}
```

```
class Decrementador extends Thread {
  public void run() {
    for (int i = 0; i < N_OPS; i++) {
      dec_quiere = true;
      while (inc_quiere) {}

      cont--;

      dec_quiere = false;
    }
  }
}
```

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

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

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

(c) El programa no garantiza la propiedad de ausencia de interbloqueo.

(3 puntos) 7. Se está desarrollando una IA para competir en el campeonato del mundo del popular juego de mesa *Los nómadas que cantan*. Los desarrolladores han ideado un recurso compartido para representar las materias primas que tiene el jugador, siendo estas materias primas cereal, agua y madera.

Hay una operación no bloqueante por cada materia prima para que el jugador coja esa materia prima: **cargarCereal**, **cargarAgua** y **cargarMadera**. El jugador puede llevar a lo sumo una unidad de cada materia prima pero no puede llevar cereal y madera a la vez, si carga una pierde la otra.

Por otro lado, hay dos operaciones que consumen materias primas: **avanzar** (consume cereal y agua) y **reparar** (consume agua y madera). Estas operaciones son bloqueantes hasta que se disponga de las materias primas necesarias.

Se pide: especificar el recurso compartido teniendo en cuenta que bastaría un booleano por cada materia prima para indicar que el jugador tiene dicha materia (porque la ha cargado) o no tiene dicha materia (porque acaba de empezar el juego o la ha consumido avanzando o reparando).

C-TAD MateriasPrimas
OPERACIONES
ACCIÓN cargarCereal:
ACCIÓN cargarAgua:
ACCIÓN cargarMadera:
ACCIÓN avanzar:
ACCIÓN reparar:
SEMÁNTICA
DOMINIO:

TIPO: *MateriasPrimas* =

INICIAL:

INVARIANTE:

CPRE:

cargarCereal

POST:

CPRE:

cargarAgua

POST:

CPRE:

cargarMadera

POST:

CPRE:

avanzar

POST:

CPRE:

reparar

POST: