Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 2)/clave: a (Solución)

Curso 2016–2017 - 2º Semestre (junio 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 **mismas hojas**. No olvidéis rellenar **apellidos, nombre y número de matricula** en cada hoja.

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. Considere el siguiente código:

```
class P1
  implements CSProcess{
  public void run(){
    System.out.print("C");
    ch1.out().write(null);
    ch3.in().read();
  }
}
```

```
class P2
  implements CSProcess{
  public void run(){
    ch1.in().read();
    System.out.print("A");
    ch2.out().write();
  }
}
```

```
class P3
  implements CSProcess{
  public void run(){
    ch2.in().read();
    ch3.write(null);
    System.out.print("B");
  }
}
```

```
    Any2OneChannel ch1 = Channel.any2one();
    Any2OneChannel ch2 = Channel.any2one();
    Any2OneChannel ch3 = Channel.any2one();
    public static final void main(final String[] args){
        new Parallel (new CSProcess[] {new P1(), new P2(), new P3()}).run();
    }
```

Se pide señalar la respuesta correcta.

(a) ☑ La salida del programa siempre será `CAB`.

(b) ☐ Se imprimirá `C` tras lo cual el programa quedará bloqueado.

(c) ☐ La salida del programa podrá ser `CAB` o `CBA`.

(d) ☐ El programa siempre acabará, pero no sabemos con qué salida en concreto.

(1 punto) 2. Dado el siguiente **CTAD**:

ACCIÓN op1: *d[e]*
ACCIÓN op2: *res[s]*

TIPO: MiCTAD $= (\mathbb{N} : dato \times \mathbb{B} : hay)$
INICIAL: $\neg$self.*hay*
INVARIANTE: *cierto*

CPRE: $\neg self.hay$
op1(d)
POST: self.*hay* $\wedge$ self.*dato* $= d$
CPRE: self.*hay*
op2(res)
POST: $\neg$self.*hay* $\wedge$ *res* $=$ self.*dato*

Se ha decidido implementarlo con monitores mediante el siguiente código:

```
public class MiCTAD{
  private Monitor mutex =
    new Monitor();
  private Monitor.Cond cond =
    mutex.newCond();
  private int dato = 0;
  private boolean hay = false;

  public void op1(int d){
    mutex.enter();
    if (hay) {cond.await();}
    hay = true;
    dato = d;
    desbloquear();
    mutex.leave();
}
```

```
  public int op2(int d){
    mutex.enter();
    if (!hay) {cond.await();}
    int res = dato;
    hay = false;
    desbloquear();
    mutex.leave();
    return res;
}

private void desbloquear () {
  if (cond.waiting() > 0) {
    cond.signal();
  }
}
```

Se pide marcar la afirmación correcta:

(a) ☑ Podrían ejecutarse operaciones cuya CPRE no se cumple.

(b) ☐ Podría darse el caso de que hubiese hilos esperando en cond que, pudiendo ejecutarse, no se desbloqueen.

(1½ puntos) 3. Dada esta otra implementación del CTAD del ejercicio anterior.

```
public class MiCTAD{
  private Monitor mutex =
    new Monitor();
  private Monitor.Cond c1 =
    mutex.newCond();
  private Monitor.Cond c2 =
    mutex.newCond();
  private int dato = 0;
  private boolean hay = false;

  public void op1(int d){
    mutex.enter();
    if (hay) {c1.await();}
    hay = true;
    dato = d;
    c2.signal();
    mutex.leave();
  }
```

```
public int op2(int d){
  mutex.enter();
  if (!hay) {c2.await();}
  int res = dato;
  hay = false;
  c1.signal();
  mutex.leave();
  return res;
}
```

Se pide señalar la respuesta correcta.

(a) ☐ Podría ejecutarse procesos cuya CPRE no se cumple

(b) ☑ Se trata de una implementación correcta del recurso compartido

(c) ☐ Podría darse el caso de que hubiese hilos esperando en c1 o en c2 que, pudiendo ejecutarse, no se desbloqueen.

(1 punto) 4. Supóngase que una condición de sincronización (*CPRE*) de una operación *Op* de un recurso compartido depende del estado del recurso y de un parámetro de entrada (x). Supóngase que dicho recurso va a ser implementado con monitores y que la operación va a ser invocada por múltiples procesos.

Se pide señalar la respuesta correcta:

(a) ☑ Es posible implementar la sincronización condicional de *Op* creando una variable Cond por cada posible valor de x.

(b) ☐ Es posible implementar la sincronización condicional de *Op* con una única variable Cond.

Apellidos: **Nombre:** **Matrícula:**

(1½ puntos) 5. Se define una clase *servidor* `P` y tres clases *cliente* `P1`, `P2` y `P3` que se comunican a través de los canales de comunicación `petA`, `petB` y `petC` (se muestran las partes relevantes del código para este problema).

```
class P implements CSProcess {
  public void run() {
    boolean q = false;
    boolean r = false;
    boolean s = false;
    final int A = 0;
    final int B = 1;
    final int C = 2;
    final Guard[] entradas =
        {petA.in(), petB.in(),petC.in()};
    final Alternative servicios =
        new Alternative (entradas);
    final boolean[] sincCond =
        new boolean[3];

    while (true) {
      sincCond[A] = !(q || r);
      sincCond[B] = !(q && r);
      sincCond[C] = !s;
      int sel = servicios.fairSelect(sincCond);
      switch (sel) {
      case A:
        petA.in().read();
        q = !q;
        break;
      case B:
        petB.in().read();
        r = !r;
        break;
      case C:
        petC.in().read();
        break;
      }
    }
  }
}
```

```
class P1 implements CSProcess {
  public void run() {
    while (true) {
      petA.out().write(null);
    }
  }
}

class P2 implements CSProcess {
  public void run() {
    while (true) {
      petB.out().write(null);
    }
  }
}

class P3 implements CSProcess {
  public void run() {
    while (true) {
      petC.out().write(null);
    }
  }
}
```

Dado un programa concurrente con tres procesos `p`, `p1`, `p2` y `p3` de las clases `P`, `P1`, `P2` y `P3`.

Se pide marcar cuál de las siguientes afirmaciones es la correcta:

(a) ☐ Es seguro que los cuatro procesos se van a bloquear.

(b) ☑ Es seguro que `p1` y `p2` acabarán bloqueándose, pero `p` y `p3` podrían seguir ejecutando indefinidamente.

(c) ☐ Es posible que `p3` se bloquee pero en ese caso `p`, `p1` y `p2` podrían seguir ejecutando indefinidamente.

(d) ☐ Ninguna de las otras respuestas.

(1 punto) 6. Dado el sistema de la pregunta 5 implementado con JCSP. **Se pide** decir cuál de las siguientes afirmaciones es la correcta:

(a) ☐ El array `sincCond` contiene la evaluación de las CPREs de las operaciones `A`, `B` y `C`.

(b) ☐ Cuando `fairSelect` devuelve un valor `sel` se cumple que `sincCond[sel]` == true.

(c) ☑ Ambas son correctas

(3 puntos) 7. A continuación mostramos una modificación de la especificación formal de un recurso gestor de *lectores/escritores* para evitar el riesgo de inanición de escritores. 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*). La primera incrementa el contador de *escritores en espera*, de modo que si este contador es distinto de 0, no dejamos que entren más lectores.

C-TAD GestorLE_2

OPERACIONES
ACCIÓN intencionEscribir:
ACCIÓN permisoEscribir:
ACCIÓN finEscribir:
ACCIÓN inicioLeer:
ACCIÓN finLeer:

SEMÁNTICA
DOMINIO:
TIPO: *GestorLE_2* = $(\mathit{leyendo} : \mathbb{N} \times \mathit{escribiendo} : \mathbb{N} \times \mathit{esc\_esperando} : \mathbb{N})$
INICIAL: self $= (0, 0, 0)$
INVARIANTE: self.*leyendo* $\cdot$ self.*escribiendo* $= 0 \wedge$ self.*escribiendo* ≤ 1

CPRE: Cierto
intencionEscribir()
POST: self$^{pre} = (l, e, w) \wedge$ self $= (l, e, w + 1)$

CPRE: self.*leyendo* $= 0 \wedge$ self.*escribiendo* $= 0$
permisoEscribir()
POST: self$^{pre} = (0, e, w) \wedge$ self $= (0, e + 1, w - 1)$

CPRE: Cierto
finEscribir()
POST: self$^{pre} = (0, e + 1, w) \wedge$ self $= (0, e, w)$

CPRE: self.*escribiendo* $= 0 \wedge$ self.*esc_esperando* $= 0$
inicioLeer()
POST: self$^{pre} = (l, 0, 0) \wedge$ self $= (l + 1, 0, 0)$

CPRE: Cierto
finLeer()
POST: self$^{pre} = (l + 1, 0, w) \wedge$ self $= (l, 0, w)$

Se pide: Completar la implementación de este recurso mediante monitores que aparece en la página siguiente. En cuanto al código de desbloqueos podéis optar tanto por un método de desbloqueo genérico como por tener código de desbloqueo especializado en los distintos métodos. Si optáis por la segunda posibilidad dejad en blanco el cuerpo del método `desbloqueoSimple`.

Apellidos: **Nombre:** **Matrícula:**

```
class GestorLE2Mon implements GestorLE2 {
  // estado del recurso
```

```
  // Declaramos monitores y colas condition
```

```
  public GestorLE2Mon() {
```

```
  }

  public void intencionEscribir() {
    // acceso a la seccion critica y codigo de bloqueo
```

```
    // codigo de la operacion
```

```
    // codigo de desbloqueo y salida de la seccion critica
```

```
  }

  public void permisoEscribir() {
    // acceso a la seccion critica y codigo de bloqueo
```

```
    // codigo de la operacion
```

```
    // codigo de desbloqueo y salida de la seccion critica
```

```
  }

 public void finEscribir() {
    // acceso a la seccion critica y codigo de bloqueo
```

```
    // codigo de la operacion
```

```
    // codigo de desbloqueo y salida de la seccion critica
```

```
    }

  public void inicioLeer() {
    // acceso a la seccion critica y codigo de bloqueo
```

```
    // codigo de la operacion
```

```
    // codigo de desbloqueo y salida de la seccion critica
```

```
  }

  public void finLeer() {
    // acceso a la seccion critica y codigo de bloqueo
```

```
    // codigo de la operacion
```

```
    // codigo de desbloqueo y salida de la seccion critica
```

```
  }

  private void desbloqueoSimple() {
```

```
  }
}
```