Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 2)/clave: a

Curso 2016–2017 - Convocatoria Extraordinaria (Julio 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

($1\frac{1}{2}$ puntos) 1. Considere el siguiente código:

```
class P1
  implements CSProcess{
  public void run(){
    String s = (String)
       ch1.in().read();
    System.out.print(s);
  }
}
```

```
class P2
  implements CSProcess{
  public void run(){
    String s = (String)
      ch2.in().read();
    ch1.out().write("A");
    ch3.out().write("B");
    System.out.print(s);
  }
}
```

```
class P3
  implements CSProcess{
  public void run(){
    ch2.out().write("C");
    String s = (String)
      ch3.in().read();
    System.out.print(s);
  }
}
```

```
    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 `ABC`.

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

(c) La salida del programa puede ser tanto `ABC` como `ACB`.

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

Utiliza el espacio para dibujar el grafo de procesos y canales.

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

TIPO: Contador $= \mathbb{N}$
INICIAL: *self* $= 0$
INVARIANTE: $-1 \leq$ *self* ≤ 1

CPRE: *self* < 1
inc()
POST: *self* $=$ *self*$^{pre} + 1$
CPRE: self > -1
dec()
POST: *self* $=$ *self*$^{pre} - 1$

Se ha decidido implementarlo con monitores mediante el siguiente código:

```
public class Contador{
  private Monitor mutex = new Monitor();
  private Monitor.Cond cInc = mutex.newCond();
  private Monitor.Cond cDec = mutex.newCond();
  private int valor = 0;

  public void inc(){
    mutex.enter();
    if (valor >= 1) {
      cInc.await(); /**/
    }
    valor++;
    desbloquear ();
    mutex.leave();
  }

  public void dec(){
    mutex.enter();
    if (valor <= -1) {
      cDec.await(); /**/
    }
    valor--;
    desbloquear ();
    mutex.leave();
  }

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

Se pide marcar la afirmación correcta:

(a) Se trata de una implementación correcta del recurso compartido.

(b) Podrían ejecutarse métodos cuya CPRE no se cumple.

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

(1 punto) 3. Cuando el método `inc` de la implementación del CTAD de la pregunta 2 invoca al método `await` en las líneas marcada con `/**/` se bloquea el proceso y se libera el mutex del `Contador` permitiendo que otros procesos obtengan dicho mutex.

(a) Sí.

(b) No.

(1 punto) 4. Dado el CTAD de la pregunta 2. Si existen múltiples procesos que invocan a `inc` y múltiples procesos que invocan a `dec`. Suponiendo que usamos la metodología vista en clase, **se pide** señalar la respuesta correcta:

(a) Se podría haber implementado con una única *condition*.

(b) Es necesario utilizar al menos dos *conditions*, tal y como se han hecho en la implementación dada.

Apellidos: **Nombre:** **Matrícula:**

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

```
class P implements CSProcess {
  public void run() {
    int n = 0;

    boolean p = true;
    boolean q = false;

    final int A = 0;
    final int B = 1;

    final Guard[] entradas =
      {petA.in(), petB.in()};
    final Alternative servicios =
      new Alternative (entradas);
    final boolean[] sincCond =
      new boolean[2];

    while (true) {
      sincCond[A] = p;
      sincCond[B] = q;

      int sel = servicios.fairSelect(sincCond);
      switch (sel) {
        case A:
          petA.in().read();
          n ++;
          p = !p;
          q = !q;
          break;
       case B:
         petB.in().read();
         n++;
         q = ((n % 2) != 0);
         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);
    }
  }
}
```

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

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

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

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

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

(d) Ninguna de las otras respuestas.

(1 punto) 6. El array **`boolean`** `sincCond []` en la implementación anterior nos sirve para:

Se pide señalar la respuesta correcta.

(a) Almacenar en qué canales tenemos peticiones esperando para indicar a `fairSelect` qué canales puede seleccionar.

(b) Almacenar la evaluación de cada una de las CPREs para indicar a `fairSelect` qué canales puede seleccionar.

(3 puntos) 7. **Se pide:** Implementar el siguiente CTAD usando como mecanismo de sincronización las clases `Monitor` y `Monitor.Cond` de la librería `es.upm.babel.cclib`.

C-TAD MultiCont
OPERACIONES
ACCIÓN inc: $\mathbb{N}$*[e]*
ACCIÓN dec: $\mathbb{N}$*[e]*
SEMÁNTICA
DOMINIO:
TIPO: *MultiCont* $= \mathbb{N}$
INVARIANTE: $0 \leq \text{self} \wedge \text{self} \leq N$

INICIAL: $\text{self} = 0$

PRE: $n > 0 \wedge n < N/2$
CPRE: $\text{self} + n \leq N$
inc(n)
POST: $\text{self} = \text{self}^{pre} + n$

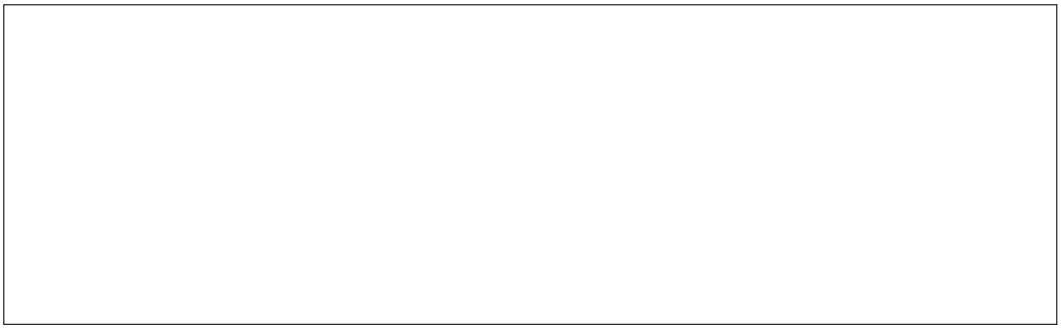
PRE: $n > 0 \wedge n < N/2$
CPRE: $n \leq \text{self}$
dec(n)
POST: $\text{self} = \text{self}^{pre} - n$

Completad el siguiente esqueleto:

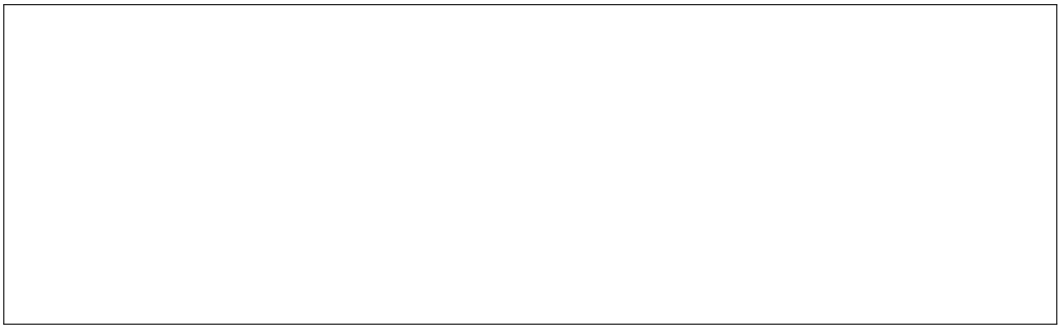
```
import es.upm.babel.cclib.Monitor;

class MultiCont {
   final static public int N = 20;

   private int multiCont;
```

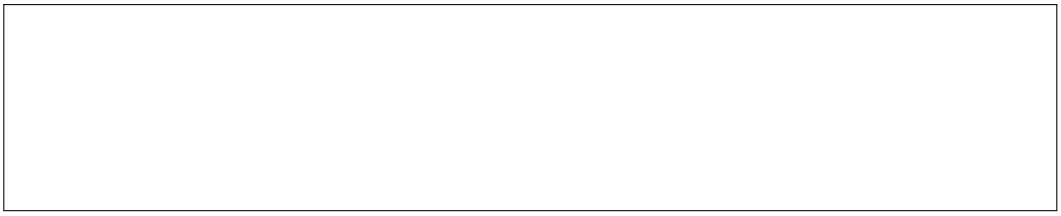


```
  public MultiCont() {
```



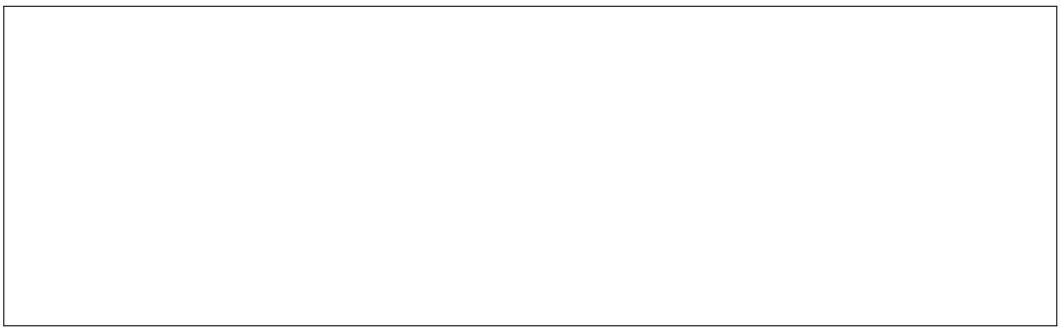
```
   }
```

```
  public void inc(int n) {
```



```
  }

  public void dec(int n) {
```



```
  }

  private void desbloqueoSimple () {
```

```
  }
}
```

(Página intencionadamente en blanco, puede usarse como hoja en sucio).