Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 2)/clave: b

Curso 2015–2016 - Convocatoria Extraordinaria (Julio 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 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 = "A"
    ch1.out().write(s);
    System.out.print("B");
    ch3.in().read();
  }
}
```

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

```
class P3
  implements CSProcess{
  public void run(){
    String s = (String)
        ch2.in().read();
    ch3.write(null);
    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 puede ser tanto `CBA` como `CAB`.

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

(c) La salida del programa siempre será `CBA`.

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

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

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

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

Se ha decidido implementarlo con monitores mediante el siguiente código:

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

  public  void dec(){
    mutex.enter();
    this.valor = -1;
    for(int i=0;i<cond.waiting();i++) {
      cond.signal();
    }
    mutex.leave();
  }
```

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

Se pide marcar la afirmación correcta:

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

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

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

(a) No.

(b) Sí.

(1 punto) 4. Dadas las siguientes implementaciones de `inc` y `dec` utilizando **while** en la evaluación de la CPRE para el CTAD de la pregunta 2:

```
void inc() {
  mutex.enter();
  while (valor >= 1) {
    cond.await();
  }
  valor++;
  desbloqueo();
  mutex.leave();
}
```

```
void dec() {
  mutex.enter();

  valor = -1;
  desbloqueo();
  mutex.leave();
}
```

```
void desbloqueo() {

  if(valor < 1 &&
     cond.waiting()>0) {
    cond.signal();
  }
}
```

Se pide señalar la respuesta correcta:

(a) Un thread que durante la ejecución de `inc()` se ha bloqueado en la condition `cond`, comprobará innecesariamente la CPRE tras ser despertado por el método `desbloqueo`, pero no volverá a bloquearse.

(b) Un thread que durante la ejecución de `inc()` se ha bloqueado en la condition `cond`, podría quedarse nuevamente bloqueado tras ser despertado por el método `desbloqueo`.

Apellidos: **Nombre:** **Matrícula:**

(1½ puntos) 5. Dada la siguiente implementación mediante JCSP del CTAD de la pregunta 2.

```
public class C implements CSProcess {

  private Any2OneChannel chInc =
      Channel.any2one();
  private Any2OneChannel chDec =
      Channel.any2one();

  private int valor = 0;

  static final int INC = 0;
  static final int DEC = 1;

  public void inc() {
    chInc.out().write(null);
    valor ++;
  }

  public void dec() {
    chDec.out().write(null);
    valor = -1;
  }
```

```
 public void run() {

   final Guard[] puertos =
      new AltingChannelInput[2];
   puertos[INC] = chInc.in();
   puertos[DEC] = chDec.in();
   final Alternative servicios =
      new Alternative(puertos);
   boolean[] sincCond = new boolean[2];
   int petIndex;
   while (true) {

     sincCond[INC] = valor < 1; // inc
     sincCond[DEC] = true;      // dec
     petIndex =
         servicios.fairSelect(sincCond);

     switch(petIndex) {
     case INC:
       chInc.in().read();
       break;
     case DEC:
       chDec.in().read();
       break;
   }
 }
}
```

Se pide marcar la afirmación correcta:

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

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

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

Se pide señalar la respuesta correcta.

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

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

(3 puntos) 7. El siguiente recurso compartido permite gestionar un sistema de backups que dispone de varios discos duros para realizar las copias de seguridad. En este recurso interactúan dos tipos de procesos: unos que se encargan de *hacer backups* y otros que se encargan de ir *reponiendo los discos* a medida que éstos se van llenando.

C-TAD SistemaBackup

OPERACIONES

ACCIÓN hacerBackup: *PosDisco[e], $\mathbb{Z}$[e]*
ACCIÓN reponerDisco: *PosDisco[e]*

SEMÁNTICA
DOMINIO:
TIPO: *SistemaBackup* $= \langle discos : PosDisco \to \mathbb{Z} \rangle$
TIPO: *PosDisco* $= 0..NUM\_DISCOS - 1$

INICIAL: $\forall\, i \in PosDisco \bullet discos(i) = 0$

PRE: $size < MAX\_BYTES$
CPRE: $self.discos(i) <= MAX\_BYTES$
hacerBackup(i,size)
POST: $self.discos = \text{self}^{pre}.discos \oplus \{i \mapsto \text{self}^{pre}.discos(i) + size\}$

CPRE: $self.discos(i) > MAX\_BYTES$
reponerDisco(i)
POST: $\text{self}.discos = \text{self}^{pre}.discos \oplus \{i \mapsto 0\}$

El sistema de backups dispone de *NUM_DISCOS* discos duros con capacidad *MAX_BYTES*. La operación $hacerBackup(i, size)$ solicita la realización de un backup de tamaño *size* bytes en el disco *i*. En caso de se intente realizar un backup y el disco solicitado ya haya superado su capacidad, el proceso deberá bloquearse hasta que vuelva a haber espacio en el disco correspondiente. Cuando un disco se haya llenado, $reponerDisco(i)$ deberá reponer el disco *i* indicando que el espacio ocupado del disco *i* es de 0 bytes.

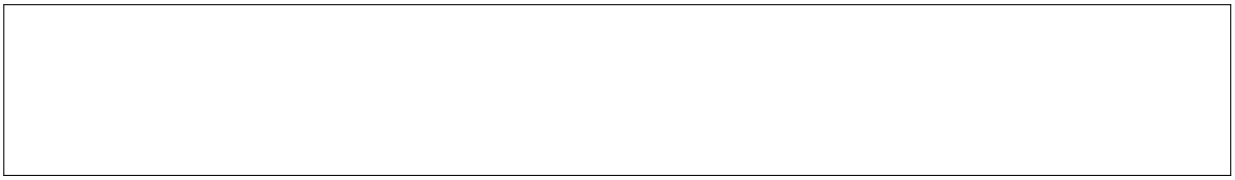
Se pide: Completar la implementación de este recurso mediante monitores que aparece a continuación y en la página siguiente. En cuanto al código de desbloqueos podéis optar tanto por un método de desbloqueo genérico como por tener código de desbloqueo especializado en los métodos `hacerBackup` y `reponerDisco`.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

```
class SistemaBackups {
  static final int MAX_BYTES = ...;
  static final int NUM_DISCOS = ...;
  // estado del recurso
  private int [] discos;

  // Declaramos monitores y colas condition
  Monitor mutex;
  Monitor.Cond[] cReponer;
  Monitor.Cond[] cDiscos;

  public SistemaBackup() {
```

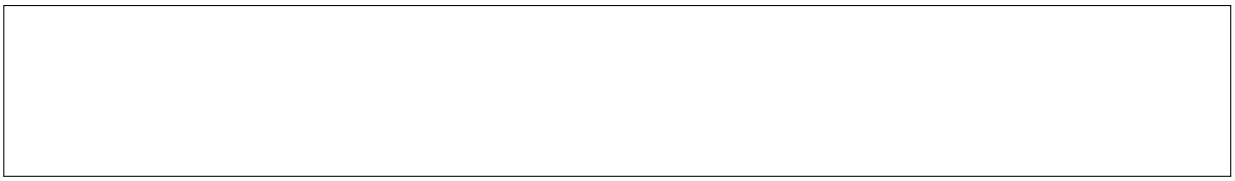
```

```

```
  }
```

```
  public void hacerBackup(int i, int size) {
    // acceso a la seccion critica y codigo de bloqueo
```



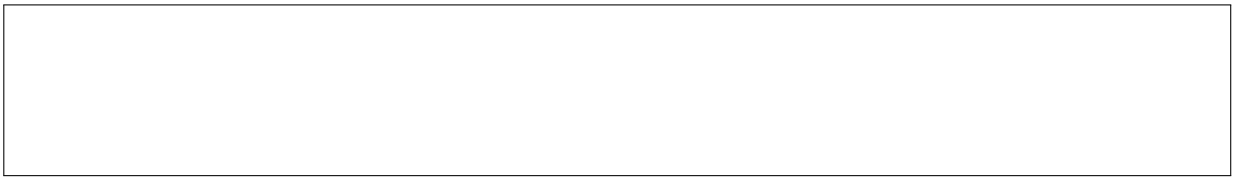
```
    // codigo de la operacion
```



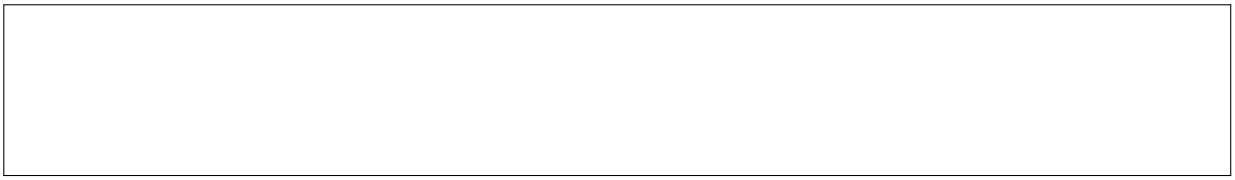
```
    // codigo de desbloqueo y salida de la seccion critica
```

```
  }

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

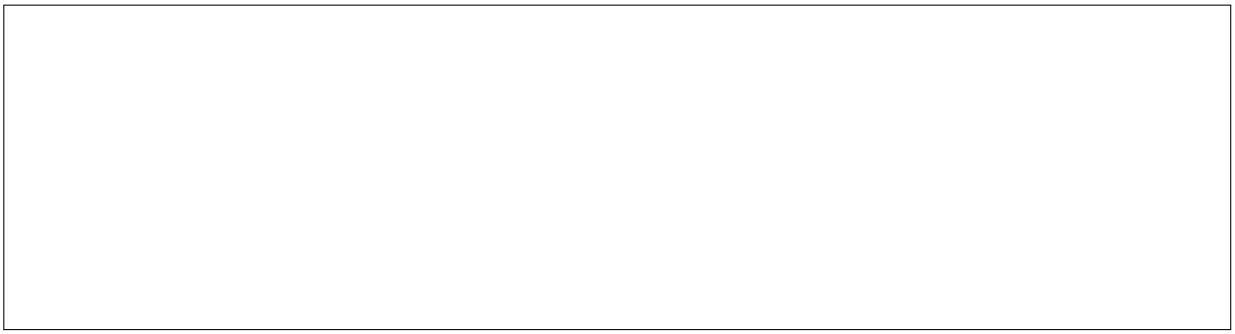
```
    // codigo de la operacion
```



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



```
  }

  private void desbloqueoSimple() {
```



```
  }
}
```