Apellidos: **Nombre:** **Matrícula:**

Concurrencia (parte 2)/clave: a

Curso 2014/2015 - 2º semestre (junio 2015)
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

(1 punto) 1. Sea un CTAD que incluye dos métodos, `inc()` y `dec()`, y que es especificado como sigue:

C-TAD IncDec **OPERACIONES** **ACCIÓN** inc: **ACCIÓN** dec: **SEMÁNTICA** **DOMINIO:** **TIPO:** *IncDec* $= \mathbb{Z}$ **INICIAL:** self $= 0$	**CPRE:** self ≤ 0 **inc()** **POST:** self $=$ self$^{pre} + 1$ **CPRE:** self ≥ 0 **dec()** **POST:** self $=$ self$^{pre} - 1$

Alguien, al implementar el recurso mediante monitores, ha decidido implementar la comprobación de la CPRE de `inc()` y `dec()` con un **`while`** como se muestra a continuación:

```
private int self;

void inc() {
  mutex.enter();
  while (self > 0) {
    cInc.await();
  }
  self++;
  desbloqueo();
  mutex.leave();
}
```

```
void dec() {
  mutex.enter();
  while (self < 0) {
    cDec.await();
  }
  self--;
  desbloqueo();
  mutex.leave();
}
```

```
void desbloqueo() {
  if (self<=0 &&
  cInc.waiting()>0) {
    cInc.signal();
  }
  else
  if(self>=0 &&
  cDec.waiting()>0){
    cDec.signal();
  }
}
```

Se pide señalar la respuesta correcta.

(a) Un thread ejecutando `inc()` y bloqueado en la *condition* `cInc` podría quedarse bloqueado nuevamente tras ser despertado.

(b) Un thread ejecutando `inc()` y bloqueado en la *condition* `cInc` comprobará innecesariamente la CPRE tras ser despertado, pero no volverá a bloquearse.

(1 punto) 2. Suponed el CTAD e implementación de la pregunta anterior, y que solo un thread ejecuta la operación `dec()`.

Se pide señalar la respuesta correcta.

(a) Un thread ejecutando `dec()` puede sufrir inanición.

(b) Todas las invocaciones tanto de `inc()` como de `dec()` terminarán, en todas las ejecuciones.

(c) Ninguna de las anteriores.

(1 punto) 3. Dada la siguiente clase que implementa el CTAD de la pregunta 1 usando JCSP:

```
public class C implements CSProcess {

  private Any2OneChannel chInc;
  private Any2OneChannel chDec;
  private int self = 0;

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

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

  public void dec() {
    chDec.out().write(null);
    self --;
  }
```

```
  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] = self <= 0; // inc
      sincCond[DEC] = self >= 0; // dec

      petIndex =
          servicios.fairSelect(sincCond);

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

Se pide señalar la respuesta correcta.

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

(b) Se trata de una implementación incorrecta del recurso compartido

(1½ puntos) 4. Considere tres threads ejecutando el siguiente código:

```
while(true) {
  ch1.out().write(null);
  ch2.in().read();
}
```

```
while(true) {
  ch1.in().read();
  ch2.out().write(null);
  ch1.in().read();
  ch3.out().write(null);
}
```

```
while(true) {
  ch1.out().write(null);
  ch3.in().read();
}
```

Considere que no hay ningún otro thread ejecutando. (Sugerencia: usa la última página para dibujar el grafo de canales y threads.)

Se pide señalar la respuesta correcta.

(a) El programa no sufre interbloqueos.

(b) El programa puede sufrir interbloqueos en algunas ejecuciones.

(c) El programa se bloqueará necesariamente en todas las ejecuciones.

Apellidos: **Nombre:** **Matrícula:**

(1½ puntos) 5. Partimos de la especificación formal de un recurso compartido mostrada en la izquierda, donde alguien ha decidido implementarlo mediante monitores con el código de la derecha:

C-TAD Turno

OPERACIONES

ACCIÓN cambia: *Turno[e]*

SEMÁNTICA
DOMINIO:
TIPO: *Turno = 0 | 1*

INICIAL: self $= 0$

CPRE: t $=$ self
cambia(t)
POST: *self* $= (self^{pre} +_2 1)$

```
void cambia(int t)
{
  mutex.enter();

  if(self != t)
    cond[t].await();

  self = (self + 1) % 2;

  senalEnviada = false;
  for(int i = 0; i < 2
      && !senalEnviada; i ++)
    if(self != i)
    {
      cond[i].signal();
      senalEnviada = true;
    }

  mutex.leave();
}
```

Se pide señalar la respuesta correcta:

(a) El recurso podría quedar bloqueado, ya que puede hacer *signals* sobre *conditions* que no lo necesitan.

(b) Se podrían hacer *signals* que despertasen hilos cuya CPRE no se cumple.

(c) Ambas son ciertas

(1 punto) 6. Supóngase un objeto `c` de una clase `C` en la que se han definido los métodos `o` y `p`, declarados como **`synchronized`**. Supóngase también que un thread `t` entra en un bucle infinito *mientras* ejecuta el código de `c.o()`. En cada iteración de este bucle infinito se llama `wait()`.

Se pide señalar la respuesta correcta.

(a) Cualquier otro thread distinto de `t` que intente ejecutar `c.p()` quedará bloqueado para asegurar la exclusión mutua.

(b) Cualquier otro thread distinto de `t` que intente ejecutar `c.p()` no quedará bloqueado porque la exclusión mutua afecta sólo a las llamadas a `c.o()`.

(c) Cualquier otro thread distinto de `t` que intente ejecutar `c.p()` no quedará bloqueado porque el `wait()` del bucle infinito soltará el mutex del objeto `c`.

(3 puntos) 7. A continuación os proporcionamos la especificación de un recurso compartido que implementa la adquisición simultánea de pares de tokens adyacentes escogidos de un vector circular de tokens. Las operaciones *adquirir* y *soltar* reciben un índice válido del vector de tokens. La operación *adquirir(i)* corresponde a la adquisición de los tokens en las posiciones i e $i+1$ (o 0 si $i+1$ es igual al número total de tokens). La operación *soltar(i)* libera los tokens de las posiciones i e $i+1$. Un proceso que invoca *adquirir* tendrá que bloquearse si alguno de los tokens necesarios está asignado.

C-TAD AnilloTokens
OPERACIONES
ACCIÓN adquirir: *Indice[e]*
ACCIÓN soltar: *Indice[e]*

SEMÁNTICA
DOMINIO:
TIPO: $\mathit{AnilloTokens} = \mathit{Indice} \nrightarrow \mathbb{B}$
TIPO: $\mathit{Indice} = \{0 \ldots N-1\}$

INVARIANTE: $| \{i \in \mathit{Indice} \bullet \mathit{self}(i)\} | \bmod 2 = 0$

INICIAL: $\forall\, i \in \mathit{Indice} \bullet \neg \mathit{self}(i)$

CPRE: $\neg \text{self}(i) \wedge \neg \text{self}((i+1) \bmod N)$
adquirir(i)
POST: $\text{self} = \text{self}^{pre} \oplus \{i \mapsto \text{Cierto}\} \oplus \{(i+1) \bmod N \mapsto \text{Cierto}\}$

PRE: $\text{self}(i) \wedge \text{self}((i+1) \bmod N)$
CPRE: Cierto
soltar(i)
POST: $\text{self} = \text{self}^{pre} \oplus \{i \mapsto \text{Falso}\} \oplus \{(i+1) \bmod N \mapsto \text{Falso}\}$

Se pide: completar la implementación por monitores de este recurso. (En la siguiente página.)
IMPORTANTE: No implementar el chequeo de las PRE.

```java
public class AnilloTokensMonitor implements AnilloTokens {
  private final int N;                          // N: para el tamanno del anillo
  private boolean anillo[];
  private Monitor mutex;
  private Cond cAdquirir[];

  private AnilloTokensMonitor () {}
  public AnilloTokensMonitor (int n) {          // n: tamanno del anillo
    [                                        ]
  }

  public void adquirir (int i) {                // No implementar la PRE!
    [                                  ]        // accedemos en
                                                // exclusion mutua
    if ( [                        ] ) {         // CPRE
      [                                   ]
    }
    anillo[i]        = true;
    anillo[(i+1) % N] = true;
    desbloqueo();                               // desbloqueo generico
    [                                  ]        // liberamos
                                                // exclusion mutua
  }
  public void soltar (int i) {                  // No implementar la PRE!
    [                                  ]        // accedemos en
                                                // exclusion mutua

                                                // CPRE: cierto
    anillo[i]        = false;
    anillo[(i+1) % N] = false;
    desbloqueo();                               // desbloqueo generico
    [                                  ]        // liberamos
                                                // exclusion mutua
  }
  private void desbloqueo () {
    [                                  ]
    for ( [                            ] ) {
      if ( [                        ] ) {
        [                                  ]
      }
    }
  }
}
```

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