MONITORES

Supóngase que una condición de sincronización (*CPRE*) de una operación Op de un recurso compartido que 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 llamada a lo sumo por un único proceso.
(a) Es posible implementar la sincronización condicional de Op con una única variable Cond.
(b) Para implementar la sincronización condicional de Op es necesario crear una variable Cond por cada posible valor de x.

¿Debería permitirse a un thread invocar una operación de await sobre un objeto de clase Monitor.Cond generado a partir de un objeto de la clase Monitor sin previamente haber invocado el método enter?
(a) Sí
(b) No

Dado el siguiente **CTAD:**

TIPO: Contador = N
INICIAL: self = 0
INVARIANTE: -1 ≤ self ∧ self ≤ 1

CPRE: self < 1
inc()
POST: self = $self^{PRE}$ + 1

CPRE: cierto
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 cond = mutex.newCond();
  private int valor = 0;

  public void dec() {
    mutex.enter();
    this.valor = -1;
    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) Se trata de una implementación correcta del recurso
(b) Podría llegar a violarse la invariante
(c) Podría darse el caso de que hubiese hilos esperando en cond que, pudiendo ejecutarse, no se desbloqueen.

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: N[e]
ACCIÓN dec: N[e]

SEMÁNTICA
DOMINIO:
TIPO: MultiCont = N
INVARIANTE: $0 \leq self \wedge self \leq N$

INICIAL: self = 0

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

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

Completad el siguiente esqueleto:

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

class MultiCont {

  final static public int N = 20;
  private int multicont;

  private Monitor mutex;
  private Monitor.Cond espInc;
  private Monitor.Cond espDec;
  private Monitor.Cond espPre;

  public MultiCont() {
    multicont = 0;
    mutex = new Monitor();
    espInc = mutex.newCond();
    espDec = mutex.newCond();
  }

  public void inc(int n) {
    mutex.enter();
    // COMPROBAMOS PRE
    if(n <= 0 || n >= N/2)
      espPre.await();
    // COMPROBAMOS CPRE
    if(multicont + n > N)
      espInc.await();
    multicont = multicont + n;
    desbloqueSimple(n);
    mutex.leave();
  }

  public void dec(int n) {
    mutex.enter();
    // COMPROBAMOS PRE
    if(n <= 0 || n >= N/2)
      espPre.await();
    // COMPROBAMOS CPRE
    if(n > multicont)
      espDec.await();
    multicont = multicont - n;
    desbloqueSimple(n);
    mutex.leave();
  }

  private void desbloqueoSimple(int n) {
    if(n > 0 && n < N/2)
      espPre.signal();
    else if(multicont + n <= N &&
        espInc.waiting() > 0)
      espInc.signal();
    else if(n <= multicont &&
        espDec.waiting() > 0)
      espDec.signal();
  }
}
```

Dada la siguiente especificación formal de un recurso compartido *Peligro*. Se pide: Completar la implementación de este recurso mediante monitores:

C-TAD *Peligro*
OPERACIONES
ACCIÓN *avisarPeligro*: $\mathbb{B}[e]$
ACCIÓN *entrar*:
ACCIÓN *salir*:

SEMÁNTICA
DOMINIO:
TIPO: $Peligro = (p : \mathbb{B} \times o : \mathbb{N})$
INICIAL: $\text{self} = (false, 0)$

INVARIANTE: $\text{self}.o \leq 5$

CPRE: Cierto
***avisarPeligro*(x)**
POST: $\text{self}.p = x \wedge \text{self}.o = \text{self}^{pre}.o$

CPRE: $\neg\text{self}.p \wedge \text{self}.o < 5$
***entrar*()**
POST: $\neg\text{self}.p \wedge \text{self}.o = \text{self}^{pre}.o + 1$

CPRE: $\text{self}.o > 0$
***salir*()**
POST: $\text{self}.p = \text{self}^{pre}.p \wedge \text{self}.o = \text{self}^{pre}.o - 1$

```
class Peligro {
  // Estado del recurso (inicialización incluída)
  private boolean p = false;
  private int o = 0;
  // Monitores y conditions (inicialización incluída)
  private Monitor mutex = new Monitor();
  private Monitor.Cond espEntrar = mutex.newCond();
  private Monitor.Cond espSalir = mutex.newCond();

  public void Peligro() {  }

  public void avisarPeligro(boolean x) {
    mutex.enter();
    // CPRE: cierto
    p = x;
    // desbloqueos
    desbloquear();
    mutex.leave();
  }

  public void entrar() {
    mutex.enter();
    // CPRE:
    if(p || o >= 5)
      espEntrar.await();
    // POST:
    p = false;
    o++;
    // desbloqueos
    desbloquear();
    mutex.leave();
  }
```

```
  public void salir() {
    mutex.enter();
    // CPRE:
    if(o <= 0)
      espSalir.await();
    // POST:
    o--;
    // desbloqueos
    desbloquear();
    mutex.leave();
  }

  private void desbloquear() {
    if(!p && o < 5 && espEntrar.waiting() > 0)
      espEntrar.signal();
    else if(o > 0 && espSalir.waiting() > 0)
      espSalir.signal();
  }
}
```

A continuación mostramos la especificación formal de un recurso gestor *Misil*. Se pide: Completar la implementación de este recurso mediante monitores. 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 opción dejan en blanco el cuerpo del método desbloqueo. NOTA: Podéis usar el método Math.abs(x) para calcular el valor absoluto de un número, |x|:

C-TAD Misil
OPERACIONES
ACCIÓN notificar: $\mathbb{Z}[e]$
ACCIÓN detectarDesviacion: $TUmbral[e] \times \mathbb{Z}[s]$

SEMÁNTICA
DOMINIO:
TIPO: $TUmbral = [0..100]$
TIPO: $Misil = \mathbb{Z}$

INICIAL: self $= 0$

CPRE: *Cierto*
notificar(desv)
POST: self $= desv$

CPRE: $|$ self $|> umbral$
detectarDesviacion(umbral,d)
POST: $self = \text{self}^{pre} \wedge d = \text{self}^{pre}$

```
class Misil {
  // Estado del recurso
  private int misil;

  // Monitores y colas conditions
  private Monitor mutex;
  private Monitor.Cond espDesv;

  public void Misil() {
    misil = 0;
    mutex = new Monitor();
    espDesv = mutex.newCond();
  }
```

```
  public void notificar(int desv) {
    // acceso a la sección crítica y código de bloqueo
    mutex.enter();
    // CPRE: cierto

    // codigo de la operacion
    misil = desv;

    // codigo de desbloqueo y salida de la seccion critica
    desbloqueo();
    mutex.leave();
  }

  public int detectarDesviacion(int umbral) {
    int d;
    // acceso a la sección crítica y código de bloqueo
    mutex.enter();
    // CPRE:
    if(Math.abs(misil) <= umbral)
      espDesv.await();

    // codigo de la operacion
    d = misil;

    // codigo de desbloqueo y salida de la seccion critica
    desbloqueo(umbral);
    mutex.leave();
    return d;
  }

  private void desbloqueo(int umbral) {
    if(Math.abs(misil) > umbral && espDesv.waiting() > 0)
      espDesv.signal();
  }
}
```

A continuación mostramos la especificación formal de un recurso gestor de lectores/escritores:
C-TAD Gestor LE

OPERACIONES
ACCIÓN Iniciar_Lectura:
ACCIÓN Iniciar_Escritura:
ACCIÓN Terminar_Lectura:
ACCIÓN Terminar_Escritura:

SEMÁNTICA
DOMINIO:
TIPO: Gestor_LE = (NLect: N x Esc: B)
INVARIANTE: self.Esc → self.NLect = 0

INICIAL: ¬self.Esc ∧ self.NLect = 0

CPRE: ¬self.Esc
Iniciar_Lectura()
POST: self = $self^{PRE}$ \ self.NLect = 1 + $self^{PRE}$.NLect

CPRE: cierto
Terminar_Lectura()
POST: self = $self^{PRE}$ \ self.NLect = $self^{PRE}$.NLect - 1

CPRE: ¬self.Esc ∧ self.NLect = 0
Iniciar_Escritura()
POST: self = $self^{PRE}$ \ self.Esc

CPRE: cierto
Terminar_Escritura()
POST: self = $self^{PRE}$ \ self ≠ $self^{PRE}$.Esc

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 desbloqueo_generico

```
public class GestorLE_Mon {
    // estado del recurso
    private int nLect;
    private boolean esc;

    // declaración de monitores y colas de condición
    private Monitor mutex;
    private Monitor.Cond espEsc;
    private Monitor.Cond espLeer;

    public GestorLE_Mon() {
      nLect = 0;
      esc = false;
      mutex = new Monitor();
      espEsc = mutex.newCond();
      espLeer = mutex.newCond();
    }

    public void iniciar_lectura() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if(esc)
        espLeer.await();

      // código de la operación
      nLect++;

      // código de desbloqueo y salida de la sección crítica
      desbloqueo_generico();
      mutex.leave();
    }

    public void terminar_lectura() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE: cierto

      // código de la operación
      nLect--;

      // código de desbloqueo y salida de la sección crítica
      desbloqueo_generico();
      mutex.leave();
    }
```

```
    public void iniciar_escritura() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE
      if(esc || nLect > 0)
        espEsc.await();

      // código de la operación
      esc = true;

      // código de desbloqueo y salida de la sección crítica
      desbloqueo_generico();
      mutex.leave();
    }

    public void terminar_escritura() {
      // acceso a la sección crítica y código de bloqueo
      mutex.leave();
      // CPRE: cierto

      // código de la operación
      esc = !esc;

      // código de desbloqueo y salida de la sección crítica
      desbloqueo_generico();
      mutex.leave();
    }

    private void desbloqueo_generico() {
      if(!esc && nLect == 0 && espEsc.waiting() > 0)
        espEsc.signal();
      else if(!esc && espLeer.waiting() > 0)
        espLeer.signal();
    }
}
```

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 GestorLE2

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

SEMÁNTICA
DOMINIO:
TIPO: GestorLE 2 = (leyendo : N ⇥ escribiendo : N ⇥ esc esperando : N)
INICIAL: self = (0, 0, 0)
INVARIANTE: self.leyendo · self.escribiendo = 0 ^ self.escribiendo 1

CPRE: Cierto
intencionEscribir()
POST: selfpre = (l, e,w) ^ self = (l, e,w + 1)

CPRE: self.leyendo = 0 ^ self.escribiendo = 0
permisoEscribir()
POST: selfpre = (l, e,w) ^ self = (0, e + 1,w − 1)

CPRE: Cierto
finEscribir()
POST: selfpre = (l, e,w) ^ self = (0, e − 1,w)

CPRE: self.escribiendo = 0 ^ self.esc esperando = 0
inicioLeer()
POST: selfpre = (l, e,w) ^ self = (l + 1, 0, 0)

CPRE: Cierto
finLeer()
POST: selfpre = (l, e,w) ^ self = (l − 1, 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.

```
public class GestorLE2_Mon extends GestorLE_Mon {
    // estado del recurso
    private int leyendo;
    private int escribiendo;
    private int esperando;

    // declaración de monitores y colas de condición
    private Monitor mutex;
    private Monitor.Cond espLeer;
    private Monitor.Cond espEsc;

    public GestorLE2_Mon() {
      leyendo = 0;
      escribiendo = 0;
      esperando = 0;
      mutex = new Monitor();
      espLeer = mutex.newCond();
      espEsc = mutex.newCond();
    }

    public void intencionEscribir() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE: cierto

      // código de la operación
      esperando++;

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }

    public void permisoEscribir() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if(leyendo > 0 || escribiendo > 0)
        espEsc.await();

      // código de la operación
      leyendo = 0;
      escribiendo++;
      esperando--;

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }
```

```
    public void finEscribir() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter()
      // CPRE: cierto

      // código de la operación
      leyendo = 0;
      escribiendo--;

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }

    public void incioLeer() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE
      if(escribiendo > 0 || esperando > 0)
        espLeer.await();

      // código de la operación
      leyendo++;
      escribiendo = 0;
      esperando = 0;

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }

    public void finLeer() {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE: cierto

      // código de la operación
      leyendo--;
      escribiendo = 0;

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }

    private void desbloqueoSimple() {
      if(leyendo == 0 && escribiendo == 0 && espEsc.waiting() > 0)
        espEsc.signal();
      else if(escribiendo == 0 && esperando == 0 && espLeer.waiting() > 0)
        espLeer.signal();
    }
}
```

El siguiente recurso compartido forma parte de un algoritmo paralelo de ordenación por mezcla. Permite mezclar dos secuencias ordenadas de números enteros para formar una única secuencia ordenada. En este recurso interactúan solo tres procesos: dos productores (izquierdo y derecho) que van pasando números de sus secuencias de uno en uno y un consumidor que va extrayendo los números en orden.

C-TAD: OrdMezcla

OPERACIONES:
ACCIÓN: insertar: Lado[e] x Z [e]
ACCIÓN: extraerMenor: Z[s]

SEMÁNTICA:
DOMINIO:
TIPO: OrdMezcla = { haydato: Lado → B x dato: Lado → Z }
TIPO: Lado = Izda | Dcha

INICIAL: $\forall i \in Lado \cdot \neg$self.hayDato(i)

CPRE: ¬self.hayDato(l)
insertar(l, d)
POST: $self^{PRE}$ = (hay, dat) ∧ self = <hay ⊕ {l → Cierto} ∧ dat ⊕ {l → d}>

CPRE: self.hayDato(Izda) ∧ self.hayDato(Dcha)
extraerMenor(min)
POST: $self^{PRE}$ = (hay, dat) ∧
(dat(Izda) ≤ dat(Dcha) ∧ min ⇒ dat(Izda) ∧ self = < hay ⊕ {Izda → Falso}, dat>) ∧
(dat(Dcha) ≤ dat(Izda) ∧ min ⇒ dat(Dcha) ∧ self = < hay ⊕ {Dcha → Falso}, dat>)

La operación insertar(lado, dato) inserta dato en el lado correspondiente, bloqueando si ese hueco no está disponible. Cuando hay datos de ambas secuencias la operación extraerMenor tomará el menor de ambos y permitirá que se añada un nuevo dato de la secuencia correspondiente. Por concisión, no hemos considerado el problema de la terminación de las secuencias.

Se pide: Completar la implementación de este recurso compartido mediante monitores que aparece a continuación 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.

```
public class OrdMezclaMon {
    // estado del recurso
    private boolean hayDatoIzq;
    private boolean hayDatoDer;
    private int datoIzq;
    private int datoDer;

    // declaración de monitores y colas de condición
    private Monitor mutex;
    private Monitor.Cond espInsertar;
    private Monitor.Cond espExtraer;

    public OrdMezclaMon() {
      hayDatoIzq = false;
      hayDatoDer = false;
      datoIzq = 0;
      datoDer = 0;
      mutex = new Monitor();
      espInsertar = mutex.newCond();
      espExtraer = mutex.newCond();
    }

    public void insertar(int lado, int dato) {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if((lado == 0 && hayDatoIzq) || (lado == 1 && hayDatoDer))
        espInsertar.await();

      // código de la operación
      if(lado == 0) {
        hayDatoIzq = true;
        datoIzq = dato;
      }

      else {
        hayDatoDer = true;
        datoDer = dato;
      }

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }
```

```
    public void extraerMenor() {
      int result;
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if(!hayDatoIzq || !hayDatoDer)
        espExtraer.await();

      // código de la operación
      if(datoIzq < datoDer) {
        result = datoIzq;
        hayDatoIzq = false;
      }
      else if(datoDer > datoIzq) {
        result = datoDer;
        hayDatoDer = false;
      }
      else {
        result = datoDer;
        hayDatoDer = false;
        hayDatoIzq = false;
      }

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
      return result;
    }

    private void desbloqueoSimple() {
      if(!hayDatoDer && !hayDatoIzq && espInsertar.waiting() > 0)
        espInsertar.signal();
      else if(hayDatoIzq && hayDatoDer && espExtraer.waiting() > 0)
        espExtraer.signal();
    }
}
```

C-TAD Buffer

OPERACIONES
ACCIÓN Poner: Tipo_Dato*[e]*
ACCIÓN Tomar: Tipo_Dato*[s]*

SEMÁNTICA
DOMINIO:
TIPO: Buffer = Secuencia:Tipo_Dato
INVARIANTE: Longitud(self) ≤ MAX

DONDE: MAX = …
INICIAL: Longitud(self) = 0

CPRE: *El buffer no está lleno*
CPRE: Longitud(self) < MAX
Poner(d)
POST: *Añadimos un elemento al buffer*
POST: l = Longitud($self^{PRE}$) ∧ Longitud(self) = l + 1 ∧ self(l + 1) = d^{PRE} ∧ self(1..l) = $self^{PRE}$

CPRE: *El buffer no está vacío*
CPRE: Longitud(self) > 0
Tomar(d)
POST: *Retiramos un elemento del buffer*
POST: l = Longitud($self^{PRE}$) ∧ Longitud(self) = l - 1 ∧ $self^{PRE}$(1) = d ∧ self = self(2..l)

```
public class BufferMon {
       // estado del recurso
       private final static int MAX = 20;
       private Object[] secuencia;
       private int nElem;

       // declaración de monitores y colas de condición
       private Monitor mutex;
       private Monitor.Cond espPoner;
       private Monitor.Cond espTomar;

       public BufferMon() {
         secuencia = new Object[MAX];
         nElem = 0;
         mutex = new Monitor();
         espPoner = mutex.newCond();
         espTomar = mutex.newCond();
       }
```

```
    public void poner(Object d) {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if(nElem == MAX)
        espPoner.await();

      // código de la operación
      secuencia[nElem] = d;
      nElem++;

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
    }

    public Object tomar() {
      Object result;
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if(nElem == 0)
        espTomar.await();

      // código de la operación
      nElem--;
      result = secuencia[nElem];

      // código de desbloqueo y salida de la sección crítica
      desbloqueoSimple();
      mutex.leave();
      return result;
    }

    public void desbloqueoSimple() {
      if(nElem < MAX && espPoner.waiting() > 0)
        espPoner.signal();
      else if(nElem > 0 && espTomar.waiting() > 0)
        espTomar.signal();
    }
}
```

C-TAD BufferPI

OPERACIONES

ACCIÓN Poner: Tipo_Dato*[e]*

ACCIÓN Tomar: Tipo_Dato*[s]* x Tipo_Paridad*[e]*

SEMÁNTICA

DOMINIO:

TIPO: BufferPI = Secuencia(Tipo_Dato)
Tipo_Paridad = par|impar
Tipo_Dato = N

INVARIANTE: Longitud(self) ≤ MAX

DONDE: MAX = …

INICIAL: Longitud(self) = 0

CPRE: *El buffer no está lleno*

CPRE: Longitud(self) < MAX

Poner(d)

POST: *Añadimos un elementos al buffer*

POST: l = Longitud($self^{PRE}$) ∧ Longitud(self) = l + 1 ∧ self(l + 1) = d^{PRE} ∧ self(1..l) = $self^{PRE}$

CPRE: *El buffer no está vacío y el primer dato preparado para salir es del tipo que requerimos*

CPRE: Longitud(self) > 0 ∧ Concuerda(self(1),t)

DONDE: Concuerda(d,t) ≡ (d mod 2 = 0 ↔ t = par)

Tomar(d, t)

POST: *Retiramos el primer elemento del buffer*

POST: l = Longitud($self^{PRE}$) ∧ $self^{PRE}$(1) = d ∧ self = $self^{PRE}$(2..l)

```
public class BufferPIMon {
     // estado del recurso
     private final static int MAX = 20;
     private int[] secuencia;
     private int nElem;

     // declaración de monitores y colas de condición
     private Monitor mutex;
     private Monitor.Cond espPoner;
     private Monitor.Cond espTomar;

     public BufferPIMon() {
       secuencia = new int[MAX];
       nElem = 0;

       mutex = new Monitor();
       espPoner = mutex.newCond();
       espTomar = mutex.newCond();
     }
```

```
    public void poner(int d) {
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if(nElem == MAX)
        espPoner.await();

      // código de la operación
      secuencia[nElem] = d;
      nElem++;

      // código de desbloqueo y salida de la sección crítica
      if(nElem < MAX && espPoner.waiting() > 0)
        espPoner.signal();
      mutex.leave();
    }

    public int tomar(boolean par) {
      int result;
      // acceso a la sección crítica y código de bloqueo
      mutex.enter();
      // CPRE:
      if((nElem == 0) && ((par && secuencia[0] % 2 != 0) ||
      (!par && secuencia[0] % 2 == 0)))
        espTomar.await();

      // código de la operación
      result = secuencia[0];
      nElem--;
      for(int i=0; i<nElem; i++)
        secuencia[i] = secuencia[i+1];

      // código de desbloqueo y salida de la sección crítica
      if(nElem > 0 && ((par && secuencia[0] % 2 == 0) ||
      (!par && secuencia[0] % 2 != 0)))
        espTomar.signal();
      mutex.leave();
      return result;
    }
}
```

C-TAD MultiBuffer

OPERACIONES
ACCIÓN Poner: Tipo_Secuencia*[e]*
ACCIÓN Tomar: Tipo_Secuencia*[s]* x N*[e]*

SEMÁNTICA
DOMINIO:
TIPO: MultiBuffer = Secuencia(Tipo_Dato)
Tipo_Secuencia = Tipo_MultiBuffer
INVARIANTE: Longitud(self) ≤ MAX
DONDE: MAX = …

INICIAL: self = <>

PRE: n ≤ [MAX/2]
CPRE: *Hay suficientes elementos en el multibuffer*
CPRE: Longitud(self) ≥ n
Tomar(self, s, n)
POST: *Retiramos elementos*
POST: n = Longitud(s) ∧ $self^{PRE}$ = s + self

PRE: Longitud(s) ≤ [MAX/2]
CPRE: *Hay sitio en el buffer para dejar la secuencia*
CPRE: Longitud(self + s) ≤ MAX
Poner(self, s)
POST: *Añadimos una secuencia al buffer*
POST: self = $self^{PRE} + s^{PRE}$