Apellidos: **Nombre:** **Matrícula:**

Prueba objetiva 2 - Clave b

Concurrencia

2011-2012 - Segundo semestre

Lenguajes, Sistemas Informáticos e Ingeniería de Software

Normas

Este es un cuestionario que consta de **7 preguntas** en **7 páginas**. Todas las preguntas son **preguntas 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 DNI** 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.

C-TAD CuentaBancaria
OPERACIONES
ACCIÓN reintegro: $\mathbb{N}$*[e]*
ACCIÓN ingreso: $\mathbb{N}$*[e]*
SEMÁNTICA
DOMINIO:
TIPO: *CuentaBancaria* = $\mathbb{N}$
INICIAL: self $= 0$

CPRE: $c \leq$ self
reintegro(c)
POST: self $=$ self$^{pre} - c$

CPRE: Cierto
ingreso(n)
POST: self $=$ self$^{pre} + n$

Figura 1: Especificación formal del recurso *CuentaBancaria*.

C-TAD LectoresEscritores
OPERACIONES
ACCIÓN InicioLectura:
ACCIÓN FinLectura:
ACCIÓN InicioEscritura:
ACCIÓN FinEscritura:
SEMÁNTICA
DOMINIO:
TIPO: *LectoresEscritores* = *(nLect:* $\mathbb{N}\times$ *esc:* $\mathbb{B}$*)*
INVARIANTE: self.*esc* $\rightarrow$ self.*nLect* $= 0$

INICIAL: $\neg$self.*esc* $\wedge$ self.*nLect* $= 0$

CPRE: $\neg$self.*esc*
InicioLectura()
POST: self $=$ self$^{pre} \setminus$ self.*nLect* $= 1 +$ self^{pre}.*nLect*

CPRE: *cierto*
FinLectura()
POST: self $=$ self$^{pre} \setminus$ self.*nLect* $=$ self^{pre}.*nLect* $- 1$

CPRE: $\neg$self.*esc* $\wedge$ self.*nLect* $= 0$
InicioEscritura()
POST: self $=$ self$^{pre} \setminus$ self.*esc*

CPRE: *cierto*
FinEscritura()
POST: self $=$ self$^{pre} \setminus \neg$self.*esc*

Figura 2: Especificación formal del recurso *LectoresEscritores*.

```
class CuentaBancaria1 {
    private volatile int saldo;
    private volatile int aplazado;
    private volatile boolean enespera;
    private Monitor mutex;
    private Monitor.Cond cSaldo;
    public CuentaBancaria1() {
        saldo = 0;
        aplazado = 0;
        enespera = false;
        mutex = new Monitor();
        cSaldo = mutex.newCond();
    }
    public void reintegro(int c) {
        mutex.enter();
        if (saldo < c || enespera) {
            aplazado = c;
            enespera = true;
            cSaldo.await();
            enespera = false;
        }
        saldo -= c;
        mutex.leave();
    }
    public void ingreso(int c) {
        mutex.enter();
        saldo += c;
        if (saldo >= aplazado) {
            cSaldo.signal();
        }
        mutex.leave();
    }
}
```

```
class CuentaBancaria2 implements CSProcess {
    private Any2OneChannel chReintegro;
    private Any2OneChannel chIngreso;
    private Any2OneChannel chRe2;
    public CuentaBancaria2() {
        chReintegro = Channel.any2one();
        chIngreso = Channel.any2one();
        chRe2 = Channel.any2one();
    }
    public void reintegro(int c) {
        chReintegro.out().write(c);
        chRe2.out().write(null);
    }
    public void ingreso(int c) {
        chIngreso.out().write(c);
    }
    public void run() {
        // preparando recepcion alternativa
        Guard[] entradas =
            {chReintegro.in(), chIngreso.in()};
        Alternative servicios =
            new Alternative (entradas);
        final int REINTEGRO = 0;
        final int INGRESO = 1;
        final boolean[] sincCond = new boolean[2];
        // estado
        int saldo = 0;
        boolean enespera = false;
        int aplazado = 0;
        // bucle principal
        while (true) {
            // refresco de las guardas
            sincCond[REINTEGRO] = !enespera;
            sincCond[INGRESO] = true;
            switch (servicios.fairSelect(sincCond)) {
            case REINTEGRO:
                aplazado =
                    (Integer)chReintegro.in().read();
                enespera = true;
                break;
            case INGRESO:
                saldo +=
                    (Integer)chIngreso.in().read();
                break;
            }
            // tratamos la peticion aplazada,
            // si se puede
            if (enespera && aplazado <= saldo) {
                saldo -= aplazado;
                enespera = false;
                chRe2.in().read();
            }
        }
    }
}
```

Figura 3: Dos códigos para la cuenta bancaria, el de la izquierda usando monitores y el de la derecha mediante CSP.

```
class LectoresEscritores1 {
    private volatile int lectores;
    private volatile int escritores;
    private Monitor mutex;
    private Monitor.Cond cLeer;
    private Monitor.Cond cEscribir;
    public LectoresEscritores1(){
        mutex.enter();
        lectores = 0;
        escritores = 0;
        mutex = new Monitor();
        cLeer     = mutex.newCond();
        cEscribir = mutex.newCond();
        mutex.leave();
    }
    public void inicioLeer() {
        mutex.enter();
        if (escritores > 0) {
            cLeer.await();
        }
        lectores++;
        cLeer.signal();
        mutex.leave();
    }
    public void finLeer() {
        mutex.enter();
        lectores--;
        if (lectores+escritores == 0
            && cEscribir.waiting()>0) {
            cEscribir.signal();
        } else {
            cLeer.signal();
        }
        mutex.leave();
    }
    public void inicioEscribir() {
        mutex.enter();
        if (lectores+escritores > 0) {
            cEscribir.await();
        }
        escritores++;
        mutex.leave();
    }
    public void finEscribir() {
        mutex.enter();
        escritores--;
        // se cumple escritores == 0 &&
        //           lectores == 0
        if (cLeer.waiting() > 0) {
            cLeer.signal();
        }
        else if (cEscribir.waiting() > 0) {
            cEscribir.signal();
        }
        mutex.leave();
    }
}
```

```
class LectoresEscritores2 implements CSProcess{
    private volatile int lectores;
    private volatile int escritores;
    private Any2OneChannel chIniLeer;
    private Any2OneChannel chFinLeer;
    private Any2OneChannel chIniEscribir;
    private Any2OneChannel chFinEscribir;
    public LectoresEscritores2(){
        lectores = 0;
        escritores = 0;
        chIniLeer = Channel.any2one();
        chFinLeer = Channel.any2one();
        chIniEscribir = Channel.any2one();
        chFinEscribir = Channel.any2one();
    }
    public void inicioLeer() {
        chIniLeer.out().write(null);
        lectores++;
    }
    public void finLeer(int c) {
        chFinLeer.out().write(null);
        lectores--;
    }
    public void inicioEscribir() {
        chIniEscribir.out().write(null);
        escritores++;
    }
    public void finEscribir() {
        chFinEscribir.out().write(null);
        escritores--;
    }
    public void run() {
        Guard[] entradas = {
            chIniLeer.in(), chFinLeer.in(),
            chIniEscribir.in(), chFinEscribir.in()
        };
        Alternative servicios =
            new Alternative (entradas);
        final int INI_LEER = 0;
        final int FIN_LEER = 1;
        final int INI_ESCRIBIR = 2;
        final int FIN_ESCRIBIR = 3;
        final boolean[] sincCond = new boolean[4];
        while (true) {
            sincCond[INI_LEER] = escritores == 0;
            sincCond[FIN_LEER] = true;
            sincCond[INI_ESCRIBIR] =
                escritores+lectores == 0;
            sincCond[FIN_ESCRIBIR] = true;
            switch (servicios.fairSelect(sincCond)) {
            case INI_LEER:
                chIniLeer.in().read();
                break;
            case FIN_LEER:
                chFinLeer.in().read();
                break;
            case INI_ESCRIBIR:
                chIniEscribir.in().read();
                break;
            case FIN_ESCRIBIR:
                chFinEscribir.in().read();
                break;
            }
        }
    }
}
```

Figura 4: Dos códigos para *lectores/escritores*, el de la izquierda usando monitores y el de la derecha mediante CSP.

```
static class LectoresEscritores3 {
  private int escritores = 0;
  private int lectores = 0;

  public LectoresEscritores3() { }

  public synchronized void finLeer() {
    lectores--;
    notify();
  }

  public synchronized void finEscribir() {
    escritores--;
    notify();
  }

  public synchronized void inicioLeer() {
    if (escritores > 0) {
      try {wait();} catch (Exception e) {}
    }
    lectores++;
  }

  public synchronized void inicioEscribir() {
     if (escritores + escritores > 0) {
      try {wait();} catch (Exception e) {}
    }
    escritores++;
  }
}
```

Figura 5: Un lectores/escritores con métodos *synchronized*, *wait* y *notify*.

```
static private Any2OneChannel c1 = Channel.any2one();
static private Any2OneChannel c2 = Channel.any2one();
```

```
static class A implements CSProcess {
    public void run() {
        c1.out().write("A");
        String s =
            (String) c2.in().read();
        System.out.print(s);
    }
}
```

```
static class B implements CSProcess {
    public void run() {
        c2.out().write("B");
        String s =
            (String) c1.in().read();
        System.out.print(s);
    }
}
```

```
    // Programa principal
    CSProcess sistema = new Parallel(new CSProcess[] {new A(), new B()});
    sistema.run();
```

Figura 6: Un sistema de dos threads que interactúan con paso de mensajes síncrono.

Apellidos: **Nombre:** **Matrícula:**

Cuestionario

(1 punto) 1. La figura 1 muestra la especificación de un recurso para gestionar una cuenta bancaria compartida. La parte izquierda de la figura 3 muestra una posible implementación de dicho recurso usando monitores. **Se pide** señalar la respuesta correcta:

   (a) Es una implementación insegura en la que se pueden ejecutar operaciones sin cumplirse su CPRE.

   (b) Se trata de una implementación correcta del recurso asumiendo atención de reintegros por estricto orden de llegada.

(1 punto) 2. La parte derecha de la figura 3 muestra una posible implementación del recurso de la cuenta bancaria usando JCSP. **Se pide** señalar la respuesta correcta:

   (a) Es una implementación insegura en la que se pueden ejecutar operaciones sin cumplirse su CPRE.

   (b) Se trata de una implementación correcta del recurso asumiendo atención de reintegros por estricto orden de llegada.

(1 punto) 3. La figura 2 muestra la especificación de un recurso para gestionar dos modos de acceso a un fichero. Permitimos que varios threads accedan simultáneamente al fichero en modo *lectura*, pero un acceso en modo *escritura* debe realizarse en exclusión con cualquier otro acceso, sea de lectura o de escritura. Para ello, cuando un thread desea acceder en modo lectura, llama a *inicioLeer()*, después accede al fichero, y al terminar llama a *finLeer()*. Análogamente, para ejecutar un acceso en modo escritura, primero llama a *inicioEscribir()*, después accede al fichero, y al terminar llama a *finEscribir()*.

   La parte izquierda de la figura 4 muestra una posible implementación de dicho recurso usando monitores. **Se pide** señalar la respuesta correcta:

   (a) Es una implementación insegura en la que puede violarse la invariante del recurso.

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

(1 punto) 4. La parte derecha de la figura 4 muestra una posible implementación del recurso de lectores/escritores usando JCSP. **Se pide** señalar la respuesta correcta:

   (a) Es una implementación insegura en la que puede corromperse el estado del recurso.

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

(1 punto) 5. La figura 5 muestra una posible implementación del recurso de lectores/escritores usando métodos *synchronized*, *wait* y *notify*. **Se pide** señalar la respuesta correcta:

   (a) En el caso de un lector y un escritor, el sistema adolecería de alternancia estricta.

   (b) Puede provocar un interbloqueo.

   (c) Es una implementación correcta del recurso.

   (d) Puede provocar la ejecución de una operación cuando su condición de sincronización (*CPRE*) es falsa.

(1 punto) 6. Dado el código de la figura 6, **se pide:** señalar la respuesta correcta:

   (a) La salida del programa es siempre `BA`.

   (b) Se produce un interbloqueo y el programa no produce salida alguna.

   (c) La salida del programa es `AB` o `BA`.

   (d) La salida del programa es siempre `AB`.

(4 puntos) 7. Una pequeña variación del típico problema del *productor-buffer-consumidor* es el llamado "buffer de pares e impares" que tenéis en los apuntes de especificación de recursos. La idea es que en la operación de extracción se permite decir si queremos retirar un número par o un número impar.

Se pide implementar el recurso compartido usando monitores, a partir de la siguiente especificación:

C-TAD BufferPI

OPERACIONES
ACCIÓN Poner: *Tipo_Dato[e]*
ACCIÓN Tomar: *Tipo_Paridad[e]* $\times$ *Tipo_Dato[s]*

SEMÁNTICA
DOMINIO:
TIPO: *Tipo_Buffer_PI* = Secuencia*(Tipo_Dato)*
Tipo_Paridad = *par*|*impar*
Tipo_Dato = $\mathbb{N}$
INVARIANTE: $\forall\, b \in$ *Tipo_Buffer_PI* $\bullet$ Longitud$(b) \leq$ *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$(self^{pre}) \wedge$ Longitud$(self) = l+1 \wedge self(l+1) = d^{pre} \wedge self(1..l) = self^{pre}$

***CPRE*:** *El buffer no está vacío y el primer dato preparado para salir es del tipo que requerimos*
CPRE: Longitud$(self) > 0 \wedge$ *Concuerda*$(self(1), t)$
DONDE: *Concuerda*$(d, t) \equiv (d \bmod 2 = 0 \leftrightarrow t =$ *par*$)$
Tomar(t, d)
***POST*:** *Retiramos el primer elemento del buffer*
POST: $l =$ Longitud$(self^{pre}) \wedge self^{pre}(1) = d \wedge self = self^{pre}(2..l)$

Apellidos: **Nombre:** **Matrícula:**

(Escribe aquí la solución a la pregunta de desarrollo.)