Apellidos: **Nombre:** **DNI/NIE:**

Prueba objetiva 2 - Clave b

Concurrencia

2012-2013 - Segundo semestre

Universidad Politécnica de Madrid

Normas

Este es un cuestionario que consta de **5 preguntas** en **5 páginas**. Todas las preguntas son **preguntas de respuesta simple** excepto la pregunta 5 que es una **pregunta de desarrollo**. La puntuación total del examen es de **10 puntos**, habiéndose de conseguir un **mínimo de 1 punto** en la pregunta de desarrollo. La duración total es de **una hora**. El examen debe contestarse en las **hojas de respuestas**. No olvidéis rellenar **apellidos, nombre y DNI o NIE** en cada hoja de respuesta, asi como la clave (A o B) en el formulario óptico.

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: *TSaldo[e]*
ACCIÓN ingreso: *TSaldo[e]*
SEMÁNTICA
DOMINIO:
TIPO: *CuentaBancaria* = $\mathbb{N}$
TIPO: *TSaldo* = $\{1 \ldots MAX\}$
INICIAL: self = 0

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

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

Figura 1: Especificación formal del recurso *CuentaBancaria*.

Cuestionario

($1\frac{1}{2}$ puntos) 1. La figura 1 muestra la especificación de un recurso para gestionar una cuenta bancaria compartida. La parte izquierda de la figura 2 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 si nos da igual el orden en que se atiendan los reintegros.

($1\frac{1}{2}$ puntos) 2. La parte derecha de la figura 2 muestra otra implementación de la cuenta bancaria usando JCSP.

Se pide señalar la respuesta correcta:

(a) Es una implementación incorrecta del recurso compartido.

(b) Se trata de una implementación correcta del recurso asumiendo que se intenta favorecer a los reintegros de menor cuantía.

($1\frac{1}{2}$ puntos) 3. Dado el código de la figura 3, **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`.

```
class CuentaBancaria3
  implements CSProcess {
  private Any2OneChannel chReintegro;
  private Any2OneChannel chIngreso;
  private int a_retirar = 0;
  public CuentaBancaria3() {
    chReintegro = Channel.any2one();
    chIngreso = Channel.any2one();
    a_retirar = 0;
  }
  public void reintegro(int c) {
    this.a_retirar = c;
    chReintegro.out().write(null);
  }
  public void ingreso(int c) {
    chIngreso.out().write(c);
  }
  public void run() {
    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];
    int saldo = 0;
    while (true) {
      sincCond[REINTEGRO] =
        a_retirar < saldo;
      sincCond[INGRESO] = true;
      switch
        (servicios.fairSelect(sincCond))
      {
      case REINTEGRO:
        chReintegro.in().read();
        saldo -= a_retirar;
        break;
      case INGRESO:
        saldo +=
          (Integer)chIngreso.in().read();
        break;
      }
    }
  }
}
```

```
class CuentaBancaria4
  implements CSProcess {
  private Any2OneChannel chReintegro;
  private Any2OneChannel chIngreso;
  private Any2OneChannel chRe2;
  public CuentaBancaria4() {
    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() {
    Guard[] entradas =
      {chReintegro.in(), chIngreso.in()};
    Alternative servicios =
      new Alternative (entradas);
    final int REINTEGRO = 0;
    final int INGRESO = 1;
    int saldo = 0;
    int a_retirar = 0;
    int[] aplazados = new int[MAX+1];
    for (int i = 1; i <= MAX; i++) {
      aplazados[i] = 0;
    }
    while (true) {
      switch (servicios.fairSelect()) {
      case REINTEGRO:
        a_retirar =
          (Integer)chReintegro.in().read();
        aplazados[a_retirar]++;
        break;
      case INGRESO:
        saldo +=
          (Integer)chIngreso.in().read();
        break;
      }
      for (int c = 1;
           c <= MAX && c <= saldo;
           c++) {
        while (c <= saldo
               && aplazados[c] > 0) {
          saldo -= c;
          aplazados[c]--;
          chRe2.in().read();
        }
      }
    }
  }
}
```

Figura 2: Dos implementaciones de la cuenta bancaria usando JCSP.

Apellidos: **Nombre:** **DNI/NIE:**

```
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 3: Un sistema de dos threads que interactúan con paso de mensajes síncrono.

(1½ puntos) 4. Supongamos que, a pesar de los problemas inherentes al uso de métodos *synchronized* y el *notifyAll* hemos decidido implementar la cuenta bancaria usando estos mecanismos. La operación *Reintegro* se ha implementado mediante el código siguiente:

```
public synchronized void reintegro(int c) {
   // traduccion CPRE
   if (c > this.saldo) {
     try {
       wait();
     } catch (Exception e) {}
   }
   // traduccion POST
   this.saldo = this.saldo - c;
   // codigo desbloqueos
   notifyAll();
}
```

Se pide señalar la respuesta correcta:

(a) Ese código puede provocar que se ejecute una operación cuya CPRE no se cumple.

(b) Es una implementación correcta de la operación *Reintegro* usando métodos *synchronized*, `wait()` y `notifyAll()`.

(4 puntos) 5. 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]* × *Tipo_Dato[s]*

SEMÁNTICA
DOMINIO:
TIPO: $Tipo\_Buffer\_PI = \mathsf{Secuencia}(Tipo\_Dato)$
$Tipo\_Paridad = par|impar$
$Tipo\_Dato = \mathbb{N}$
INVARIANTE: $\forall\, b \in Tipo\_Buffer\_PI \bullet \mathsf{Longitud}(b) \leq MAX$
DONDE: $MAX = ...$
INICIAL: $\mathsf{Longitud}(self) = 0$

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

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

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