PASO DE MENSAJES

A continuación mostramos la especificación formal de un recurso para jugar al popular juego de mesa *Los nómadas que cantan*. Los desarrolladores han ideado un recurso compartido para representar las materias primas que tiene el jugador, siendo estas materias primas cereal, agua y madera. Se pide: Completar la implementación de este recurso mediante paso de mensajes siguiendo la metodología vista en clase.

C-TAD MateriasPrimas
OPERACIONES
ACCIÓN cargarCereal:
ACCIÓN cargarAgua:
ACCIÓN cargarMadera:
ACCIÓN avanzar:
ACCIÓN reparar:
SEMÁNTICA
DOMINIO:
TIPO: $MateriasPrimas = (cereal : \mathbb{N} \times agua : \mathbb{N} \times madera : \mathbb{N})$
INICIAL: $\text{self} = (0, 0, 0)$
INVARIANTE: $\text{self}.cereal + \text{self}.agua + \text{self}.madera < 10$

CPRE: $\text{self}.cereal + \text{self}.agua + \text{self}.madera + 1 < 10$
cargarCereal
POST: $\text{self}^{pre} = (c, a, m) \land \text{self} = (c + 1, a, m)$

CPRE: $\text{self}.cereal + \text{self}.agua + \text{self}.madera + 1 < 10$
cargarAgua
POST: $\text{self}^{pre} = (c, a, m) \land \text{self} = (c, a + 1, m)$

CPRE: $\text{self}.cereal + \text{self}.agua + \text{self}.madera + 1 < 10$
cargarMadera
POST: $\text{self}^{pre} = (c, a, m) \land \text{self} = (c, a, m + 1)$

CPRE: $\text{self}.cereal > 0 \land \text{self}.agua > 0$
avanzar
POST: $\text{self}^{pre} = (c, a, m) \land \text{self} = (c - 1, a - 1, m)$

CPRE: $\text{self}.agua > 0 \land \text{self}.madera > 0$
reparar
POST: $\text{self}^{pre} = (c, a, m) \land \text{self} = (c, a - 1, m - 1)$

A continuación mostramos la especificación formal del recurso compartido *Peligro*. Inmediatamente después mostramos una implementación del mismo utilizando JCSP:

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 PeligroCSP
  implements CSProcess {
  private Any2OneChannel cp =
    Channel.any2one();
  private Any2OneChannel ce =
    Channel.any2one();
  private Any2OneChannel cs =
    Channel.any2one();

  public PeligroCSP() {
    new ProcessManager(this).start();
  }

  public void avisarPeligro(boolean x)
  {
    cp.out().write(x);
  }

  public void entrar() {
    ce.out().write(null);
  }

  public void salir() {
    cs.out().write(null);
  }
```

```
  public void run() {
    Boolean p = false;
    Integer o = 0;
    Guard[] inputs =
      {cp.in(), ce.in(), cs.in()};
    Alternative services =
      new Alternative(inputs);
    while(true) {
      switch(services.fairSelect()) {
      case 0:
        p = (Boolean)cp.in().read();
        break;
      case 1:
        ce.in().read();
        if (!p && o < 5)
          o++;
        break;
      case 2:
        cs.in().read();
        if (o > 0)
          o--;
        break;
      }
    }
  }
}
```

Se pide marcar la afirmación correcta:
(a) Es una implementación correcta del recurso compartido
(b) Es una implementación incorrecta del recurso compartido

Dado el siguiente programa concurrente que ejecuta dos procesos JCSP con paso de mensajes síncrono:

```
static One2OneChannel c = Channel.one2one();
```

```
class A implements CSProcess {
  public void run() {
    System.out.print("1");
    c.out().write(null);
    System.out.print("2");
  }
}
```

```
class B extends Thread {
  public void run() {
    System.out.print("3");
    c.in().read();
    System.out.print("4");
  }
}
```

Se pide marcar la afirmación correcta:
(a) "1234" es una salida posible del recurso compartido
(b) "1234" no es una salida posible del recurso compartido

Considere el siguiente código:

```
class P1
  implements CSProcess{
  public void run(){
    ch1.out().write(null);
    System.out.print("A");
    ch3.in().read();
  }
}
```

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

```
class P3
  implements CSProcess{
  public void run(){
    System.out.print("B");
    ch2.in().read();
    ch3.out().write(null);
  }
}
```

```
    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 marcar la afirmación correcta:
(a) La salida del programa siempre será CAB
(b) Se imprimirá A tras lo cual el programa quedará bloqueado
(c) La salida del programa solo podrá ser ACB o CAB
(d) El programa siempre acabará pero no podemos saber con qué salida en concreto

Dado un recurso compartido implementado con JCSP.
Se pide decir cuál de las siguientes afirmaciones es la correcta:
(a) Cuando fairSelect(sincCond) devuelve un valor sel no sabemos si habrá alguna petición en el canal que ocupa la posición sel y la lectura sobre dicho canal se podrá quedar bloqueada.
(b) Cuando fairSelect(sincCond) devuelve un valor sel sabemos que hay una petición en el canal que ocupa la posición sel del array de alternativas y que sincCond[sel] == true.

Se define una clase servidor P y tres clases cliente P1, P2 y P3 que se comunican a través de los canales de comunicación petA, petB y petC (se muestran las partes relevantes del código para este problema).

```
class P implements CSProcess {
  public void run() {
    boolean q = true;
    boolean r = true;
    boolean s = false;
    final int A = 0;
    final int B = 1;
    final int C = 2;
    final Guard[] entradas =
        {petA.in(), petB.in(), petC.in()};
    final Alternative servicios =
        new Alternative (entradas);
    final boolean[] sincCond =
        new boolean[3];

    while (true) {
      sincCond[A] = (q && r);
      sincCond[B] = (r && s);
      sincCond[C] = (s && q);
      int sel = servicios.fairSelect(sincCond);
      switch (sel) {
      case A:
        petA.in().read();
        q = !q;
        s = !s;
        System.out.println("A");
        break;
      case B:
        petB.in().read();
        r = !r;
        q = !q;
        System.out.println("B");
        break;
      case C:
        petC.in().read();
        s = !s;
        r = !r;
        System.out.println("C");
        break;
      }
    }
  }
}
```

```
class P1 implements CSProcess {
  public void run() {
    while (true) {
      petA.out().write(null);
    }
  }
}

class P2 implements CSProcess {
  public void run() {
    while (true) {
      petB.out().write(null);
    }
  }
}

class P3 implements CSProcess {
  public void run() {
    while (true) {
      petC.out().write(null);
    }
  }
}
```

Dado un programa concurrente con tres procesos p, p1, p2 y p3 de las clases P, P1, P2 y P3.
Se pide marcar cuál de las siguientes afirmaciones es la correcta:
(a) Es seguro que los cuatro procesos van a ejecutar indefinidamente, pero no se puede saber con qué salida concreta.
(b) El programa imprimirá AB y los 4 procesos acabarán bloqueados.
(c) El programa se ejecutará indefinidamente y la salida por consola será ABCABCABC... indefinidamente.
(d) Ninguna de las otras respuestas.

Dado el siguiente **CTAD**

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

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

¿Cual de los siguientes códigos sería **una buena implementación** del método run() para este recurso si quisiéramos implementarlo en JCSP?

A

```
public void run()
{
  final int INC = 0;
  final int DEC = 1;
  int valor = 0;
  Guard[] ent =
    {petInc.in(),
     petDec.in()};
  Alternative ser =
    new Alternative (ent);
  boolean[] sc =
    new boolean[2];
  while(true){
    sc[INC] = valor < 1;
    sc[DEC] = true;
    int sel =
      ser.fairSelect(sc);
    ent[sel].read();
    switch (sel) {
      case INC:
        valor++;
        break;
      case DEC:
        valor = -1;
        break;}
  }}
```

B

```
public void run()
{
  final int INC = 0;
  final int DEC = 1;
  int valor = 0;
  Guard[] ent =
    {petInc.in(),
     petDec.in()};
  Alternative ser =
    new Alternative (ent);
  while(true){
    int sel =
      ser.fairSelect();
    ent[sel].read();
    switch (sel) {
      case INC:
        valor++;
        break;
      case DEC:
        valor = -1;
        break;}
  }}
```

C

```
public void run()
{
  final int INC = 0;
  final int DEC = 1;
  int valor = 0;
  Guard[] ent =
    {petInc.in(),
     petDec.in()};
  Alternative ser =
    new Alternative (ent);
  boolean[] sc =
    new boolean[2];
  while(true){
    sc[INC] = true;
    sc[DEC] = valor < 1;
    int sel =
      ser.fairSelect(sc);
    ent[sel].read();
    switch (sel) {
      case INC:
        valor++;
        break;
      case DEC:
        valor = -1;
        break;}
  }}
```

Se define una clase *servidor* `P` y dos clases *cliente* `P1` y `P2` que se comunican a través de los canales de comunicación `petA` y `petB` (se muestran las partes relevantes del código para este problema).

```
class P implements CSProcess {
  public void run() {
    int n = 0;

    boolean p = true;
    boolean q = false;

    final int A = 0;
    final int B = 1;

    final Guard[] entradas =
      {petA.in(), petB.in()};
    final Alternative servicios =
      new Alternative (entradas);
    final boolean[] sincCond =
      new boolean[2];

    while (true) {
      sincCond[A] = p;
      sincCond[B] = q;

      int sel = servicios.fairSelect(sincCond);
      switch (sel) {
        case A:
          petA.in().read();
          n ++;
          p = !p;
          q = !q;
          break;
       case B:
         petB.in().read();
         n++;
         q = ((n % 2) != 0);
         break;
      }
    }
  }
}
```

```
class P1 implements CSProcess {
  public void run() {
    while (true) {
      petA.out().write(null);
    }
  }
}

class P2 implements CSProcess {
  public void run() {
    while (true) {
      petB.out().write(null);
    }
  }
}
```

Dado un programa concurrente con tres procesos p, p1 y p2 de las clases P, P1 y P2.
Se pide marcar cuál de las siguientes afirmaciones es la correcta:
(a) Es seguro que los tres procesos se van a bloquear
(b) Es seguro que p1 acabará bloqueándose, pero p y p2 podrían seguir ejecutando indefinidamente.
(c) Es posible que p2 se bloquee, pero en ese caso p y p1 podrían seguir ejecutando indefinidamente.
(d) Ninguna de las otras respuestas.

Considere el siguiente código:

```
class P1
  implements CSProcess{
  public void run(){
    String s = (String)
       ch1.in().read();
    System.out.print(s);
  }
}
```

```
class P2
  implements CSProcess{
  public void run(){
    String s = (String)
      ch2.in().read();
    ch1.out().write("A");
    ch3.out().write("B");
    System.out.print(s);
  }
}
```

```
class P3
  implements CSProcess{
  public void run(){
    ch2.out().write("C");
    String s = (String)
      ch3.in().read();
    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 siempre será `ABC`.

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

(c) La salida del programa puede ser tanto `ABC` como `ACB`.

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

Considere el siguiente código:

```
class P1
  implements CSProcess{
  public void run(){
    One2OneChannel chResp =
      Channel.one2one();
    ch1.out().write(
      chResp.out());
    chResp.in().read();
    System.out.print("A");
  }
}
```

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

```
class P3
  implements CSProcess{
  public void run(){
    ChannelOutput resp =
      (ChannelOutput)
        ch2.in().read();
    resp.write(null);
    System.out.print("C");
  }
}
```

```
    Any2OneChannel ch1 = Channel.any2one();
    Any2OneChannel ch2 = Channel.any2one();
    public static final void main(final String[] args){
        new Parallel (new CSProcess[] {new P1(), new P2(), new P3()}).run();
    }
```

Se pide marcar la respuesta correcta:

(a) La salida del programa siempre será ABC

(b) Se imprimirá C tras o 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