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(){
    Channel0utput resp =
      (Channel0utput)
        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 lo 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

Suponed que en el código anterior cambiamos los canales ch1 y ch2 por canales de tipo One2OneChannel haciendo todos los cambios que sean necesarios para ello. Se pide señalar la respuesta correcta:

(a) Nos dará un error cuando lo compilemos
(b) Seguirá funcionando igual que antes
(c) Nos dará un error en tiempo de ejecución

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$^{pre} + c$

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.

ingreso(3)
reintegro(10) → NO SE HA LLEGADO A EJECUTAR NUNCA
ingreso(5)
reintegro(2) → SE EJECUTA
Luego, es una implementación incorrecta del recurso

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.

(b) Es una implementación correcta, se hacen de menor a
reintegro(3); → INICIO: aplazados = {0,0,0,0,0,0,0,0}; → COMO QUEDA: aplazados={0,0,0,1,0,0,0,0}
reintegro(2); → INICIO: aplazados = {0,0,0,1,0,0,0,0}; → COMO QUEDA: aplazados={0,0,1,1,0,0,0,0}
ingreso(2); → reintegro(2) tiene efecto quedando aplazados={0,0,0,1,0,0,0,0}
ingreso(2);

```
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.

```
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.

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`.

(c) La salida es AB o BA (se hace paralelamente y los dos tienen 3 sentencias que ejecutar)

Se define una clase *servidor* `S` y dos clases *cliente* `C1` y `C2` que se comunican a través de los canales de comunicación `pet1` y `pet2` (se muestran las partes relevantes del código para este problema).

```
class S implements CSProcess {
  public void run() {
    boolean x = true;
    boolean y = false;
    final int PET1 = 0;
    final int PET2 = 1;
    final Guard[] entradas =
        {pet1.in(), pet2.in()};
    final Alternative servicios =
        new Alternative (entradas);
    final boolean[] sincCond =
        new boolean[2];
    while (true) {
      sincCond[PET1] = x;
      sincCond[PET2] = y;
      int sel = servicios.fairSelect(sincCond);
      switch (sel) {
      case PET1:
        pet1.in().read();
        x = !x;
        y = !y;
        break;
      case PET2:
        pet2.in().read();
        y = !y;
        break;
      }//end case
    }//end while
  }//end run()
```

```
class C1 implements CSProcess {
  public void run() {
    while (true) {
      pet1.out().write(null);
      System.out.print("A");
    }
  }
}

class C2 implements CSProcess {
  public void run() {
    while (true) {
      pet2.out().write(null);
      System.out.print("B");
    }
  }
}
```

Dado un programa concurrente con tres procesos `s`, `c1` y `c2` de las clases `S`, `C1` y `C2`, **se pide:** marcar cuál de las siguientes afirmaciones es la correcta:

La salida del programa es `BA` y los tres procesos se quedarán bloqueados.

La salida del programa es `AB` y los tres procesos se quedarán bloqueados.

La salida del programa es `ABABABAB`... indefinidamente.

Ninguna de las respuestas anteriores.

Imprimo A, pongo x a false e y a true
Imprimo B, pongo y a false (x sigue siendo false)
Al tener tanto x como y a false, el proceso S se queda bloqueado y el write(null) de los procesos C1 y C2 se quedan bloqueados también porque no hay ningún read asociado a ellos.

AB y los tres procesos se bloquean

Se define una clase *ContadorCSP* que implementa un recurso compartido *Contador* usando JCSP. Dicho recurso dispone de 2 operaciones, `inc()` y `dec()`, cuya *CPRE = cierto*:

```
public class ContadorCSP
       implements CSProcess {

  private Any2OneChannel chInc =
       Channel.any2One();
  private Any2OneChannel chDec =
       Channel.any2one();
  private int contador = 0;

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

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

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

```
  public void run() {

    Guard[] puertos =
       new AltingChannelInput[2];
    puertos[INC] = chInc.in();
    puertos[DEC] = chDec.in();

    final Alternative servicios =
       new Alternative(puertos);

    while (true) {

      int petIndex = servicios.fairSelect();

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

Se trata de una implementación incorrecta del recurso compartido

Se trata de una implementación correcta del recurso compartido.

Dado que los dos procesos tienen la CPRE como true, entonces no tenemos por qué poner condiciones de sincronización, se puede hacer de manera aleatoria sin problema alguno. Es una implementación correcta del recurso.