(3 puntos) 7. A continuación mostramos una 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.

C-TAD Juego
OPERACIONES
ACCIÓN cargar: *materia[e]*
ACCIÓN avanzar:
ACCIÓN reparar:
SEMÁNTICA
DOMINIO:
TIPO: $materia = \{cereal, agua, madera\}$
TIPO: $Juego = materia \to \mathbb{N}$
INICIAL: $\text{self} = \{cereal \mapsto 0, agua \mapsto 0, madera \mapsto 0\}$
INVARIANTE: $\text{self}(cereal) + \text{self}(agua) + \text{self}(madera) < 10$

CPRE: $\text{self}(cereal) + \text{self}(agua) + \text{self}(madera) + 1 < 10$
cargar(m)
POST: $\text{self} = \text{self}^{pre} \oplus \{m \mapsto \text{self}^{pre}(m) + 1\}$

CPRE: $\text{self}(cereal) > 0 \land \text{self}(agua) > 0$
avanzar
POST: $\text{self} = \text{self}^{pre} \oplus \{cereal \mapsto \text{self}^{pre}(cereal) - 1\} \oplus \{agua \mapsto \text{self}^{pre}(agua) - 1\}$

CPRE: $\text{self}(agua) > 0 \land \text{self}(madera) > 0$
reparar
POST: $\text{self} = \text{self}^{pre} \oplus \{agua \mapsto \text{self}^{pre}(agua) - 1\} \oplus \{madera \mapsto \text{self}^{pre}(madera) - 1\}$

Se pide: implementar este recurso compartido usando JCSP y codificando las CPREs como condiciones de la recepción alternativa (`fairselect(sinccond)`).

```
public class JuegoCSP implements CSProcess {
    // valores simbolicos para las materias primas
    final int CEREAL = 0;
    final int AGUA   = 1;
    final int MADERA = 2;

    // Canales para pedir al servidor
    private Any2OneChannel chCargar;
    private Any2OneChannel chAvanzar;
    private Any2OneChannel chReparar;

    private JuegoCSP () {
        this.chCargar  = Channel.any2one();
        this.chAvanzar = Channel.any2one();
        this.chReparar = Channel.any2one();
    }

    public void cargar(int m) {
```

```
    }

    public void avanzar() {
```

```
    }

    public void reparar() {
```

```
    }
```

Apellidos: **Nombre:** **Matrícula:**

```
    public void run() {
        // declaramos aqui el estado del recurso
        int[] materias = {0, 0, 0};

        // soporte para recepcion alternativa condicional
        // Nombres simbolicos para los indices de servicios
        final int CARGAR  = 0;
        final int AVANZAR = 1;
        final int REPARAR = 2;
        // Entradas de la select
        final AltingChannelInput[] entradas =
            {chCargar.in(), chAvanzar.in(), chReparar.in()};
        // Recepcion alternativa
        final Alternative servicios =  new Alternative (entradas);
        // Sincronizacion condicional en la select
        final boolean[] sincCond = new boolean[3];

        // el servidor ejecuta un bucle de servicio infinito:
        while (true) {
            // Preparacion de las precondiciones
```

```
            switch (servicios.fairSelect(sincCond)) {
             case CARGAR:
```

```
                break;
            case AVANZAR:
```

```
                break;
            case REPARAR:
```

```
                break;
            }
        } // fin bucle servidor
    } // fin servidor
}
```

```
    public void cargar(int m) {
        chCargar.out().write(m);
    }

    public void avanzar() {
        chAvanzar.out().write(null);
    }

    public void reparar() {
        chReparar.out().write(null);
    }

    public void run() {
        // declaramos aqui el estado del recurso
        int[] materias = {0, 0, 0};

        // soporte para recepcion alternativa condicional
        // Nombres simbolicos para los indices de servicios
        final int CARGAR  = 0;
        final int AVANZAR = 1;
        final int REPARAR = 2;
        // Entradas de la select
        final AltingChannelInput[] entradas = {chCargar.in(), chAvanzar.in(), chReparar.in()};
        // Recepcion alternativa
        final Alternative servicios =  new Alternative (entradas);
        // Sincronizacion condicional en la select
        final boolean[] sincCond = new boolean[3];

        // el servidor ejecuta un bucle de servicio infinito:
        while (true) {
            // Preparacion de las precondiciones
            sincCond[CARGAR]  = materias[CEREAL] + materias[AGUA] + materias[MADERA] < 9;
            sincCond[AVANZAR] = materias[CEREAL] > 0 && materias[AGUA]   > 0;
            sincCond[REPARAR] = materias[AGUA]   > 0 && materias[MADERA] > 0;;

            switch (servicios.fairSelect(sincCond)) {
            case CARGAR:
                int queMateria = (Integer) chCargar.in().read();
                materias[queMateria]++;
                break;
            case AVANZAR:
                chAvanzar.in().read();
                materias[CEREAL]--;
                materias[AGUA]--;
                break;
            case REPARAR:
                chReparar.in().read();
                materias[AGUA]--;
                materias[MADERA]--;
                break;
            }
        } // fin bucle servidor
    } // fin servidor
}
```