CTAD CON PASO DE MENSAJES

Parking con CSP:

C-TAD: Parking
OPERACIONES:
ACCION: entrar
ACCION: salir

DOMINIO
TIPO: Parking = $\mathbb{N}$
DONDE: CAP = $\mathbb{N}$
INVARIANTE: $0 \leq self \leq CAP$
INICIAL: self = 0

CPRE: self < CAP
entrar()
POST: self = $self^{PRE}+1$

CPRE: cierto
salir()
POST: self = $self^{PRE} - 1$

```
public class ParkingCSP implements CSProcess {
	private final int CAP;

	// Canales
	private Any2OneChannel chEntrar;
	private Any2OneChannel chSalir;

	public ParkingCSP (int capacidad) {
		CAP = capacidad;

		// Inicializamos los canales:
		chEntrar = Channel.any2one();
		chSalir = Channel.any2one();
	}

	public void entrar() {
		while(true) // SUPONEMOS QUE NOS PIDEN QUE SE HAGA ETERNAMENTE
			chEntrar.out().write(null);
	}
```

```
	public void salir() {
		while(true)
			chSalir.out().write(null);
	}

	public void run() {
		// Inicializamos los atributos del recurso
		int nElems = 0;

		// soporte para recepcion alternativa condicional
		// Nombres simbolicos para los indices de servicios
		final int ENTRAR = 0;
		final int SALIR = 1;

		// Entradas de la select
		final AltingChannelInput[] entradas =
				{chEntrar.in(), chSalir.in()};

		// Recepcion alternativa
		final Alternative servicios = new Alternative(entradas);

		// Sincronizacion condicional en la select
		boolean[] sincCond = new boolean[2];
		sincCond[SALIR] = true;

		// el servidor ejecuta un bucle de servicio infinito
		while(true) {
			// Estudio de las CPREs
			sincCond[ENTRAR] = nElems < CAP;

			switch(servicios.fairSelect(sincCond)) {
			case ENTRAR:
				chEntrar.in().read();
				// POST
				nElms++;
				break;

			case SALIR:
				chSalir.in().read();
				// POST
				nElems--;
				break;
			} // switch
		} // while
	}
}
```

Almacén de un dato con CSP:

C-TAD: Almacen1Dato
OPERACIONES:
ACCION: almacenar: Tipo_Dato[e]
ACCION: extraer: Tipo_Dato[s]

DOMINIO
TIPO: Almacen1Dato = (Dato: Tipo_Dato x HayDato: B)
INVARIANTE: cierto
INICIAL: ¬self.HayDato

CPRE: ¬self.HayDato
almacenar(e)
POST: self.Dato = e ∧ self.HayDato

CPRE: self.HayDato
extraer(s)
POST: s = $self^{pre}$.Dato ∧ ¬self.HayDato

```
public class Almacen1DatoCSP implements CSProcess {

      // Canales
      private Any2OneChannel chAlmacenar;
      private Any2OneChannel chExtraer;

      public Almacen1DatoCSP () {
            // Inicializamos los canales:
            chAlmacenar = Channel.any2one();
            chExtraer = Channel.any2one();
      }

      public void almacenar(Producto producto) {
            chAlmacenar.out().write(producto);
      }

      public Producto extraer() {
            return (Producto)chExtraer.in().read();
      }
```

```
	public void run() {
		// Inicializamos los atributos del recurso
		Producto dato = null;
		boolean hayDato = false;

		// soporte para recepcion alternativa condicional
		// Nombres simbolicos para los indices de servicios
		final int ALMACENAR = 0;
		final int EXTRAER = 1;

		// Entradas de la select
		final AltingChannelInput[] entradas =
				{chAlmacenar.in(), chExtraer.in()};

		// Recepcion alternativa
		final Alternative servicios = new Alternative(entradas);

		// Sincronizacion condicional en la select
		boolean[] sincCond = new boolean[2];

		// el servidor ejecuta un bucle de servicio infinito
		while(true) {
			// Estudio de las CPREs
			sincCond[ALMACENAR] = !hayDato;
			sincCond[EXTRAER] = hayDato;

			switch(servicios.fairSelect(sincCond)) {
			case ALMACENAR:
				dato = (Producto) chAlmacenar.in().read();
				hayDato = true;
				break;

			case EXTRAER:
				chExtraer.out().write(dato);
				hayDato = !hayDato; // hayDato = false;
				break;

			} // switch
		} // while
	}
}
```

Dada la siguiente especificación formal del recurso compartido de Lectores/Escritores. Se pide: Completar la implementación de este recurso mediante paso de mensajes:

TIPO: $LE = (l : \mathbb{Z} \times e : \mathbb{Z})$
INVARIANTE: $\forall r \in LE \bullet\ r.e \geq 0 \wedge r.l \geq 0 \wedge r.e \leq 1 \wedge$
$((r.e > 0 \Rightarrow r.l = 0) \wedge (r.l > 0 \Rightarrow r.e = 0))$
INICIAL: $\text{self} = (0, 0)$

CPRE: $\text{self}.e = 0$
inicioLeer()
POST: $\text{self} = (\text{self}^{pre}.l + 1, \text{self}^{pre}.e)$
CPRE: $\text{self}.e = 0 \wedge \text{self}.l = 0$
inicioEscribir()
POST: $\text{self} = (\text{self}^{pre}.l, \text{self}^{pre}.e + 1)$
CPRE: *cierto*
finLeer()
POST: $\text{self} = (\text{self}^{pre}.l - 1, \text{self}^{pre}.e)$
CPRE: *cierto*
finEscribir()
POST: $\text{self} = (\text{self}^{pre}.l, \text{self}^{pre}.e - 1)$

```
public class GestorLECSP implements CSProcess {

     // Canales
     private Any2OneChannel chIL;    // inicio leer
     private Any2OneChannel chFL;    // fin leer
     private Any2OneChannel chIE;    // inicio escribir
     private Any2OneChannel chFE;    // fin escribir

     public GestorLECSP () {
          // Inicializamos los canales:
          chIL = Channel.any2one();
          chFL = Channel.any2one();
          chIE = Channel.any2one();
          chFE = Channel.any2one();
     }

     public void inicioLeer() {
          chIL.out().write(null);
     }

     public void inicioEscribir() {
          chIE.out().write(null);
     }
```

```
public void finLeer() {
      chFL.out().write(null);
}

public void finEscribir() {
      chFE.out().write(null);
}

public void run() {
      // Inicializamos los atributos del recurso
      double l = 0; // numero lectores
      double e = 0; // numero escritores

      // soporte para recepcion alternativa condicional
      // Nombres simbolicos para los indices de servicios
      final int IL = 0;
      final int IE = 1;
      final int FL = 2;
      final int FE = 3;

      // Entradas de la select
      final AltingChannelInput[] entradas =
                {chIL.in(), chIE.in(), chFL.in(), chFE.in()};

      // Recepcion alternativa
      final Alternative servicios = new Alternative(entradas);

      // Sincronizacion condicional en la select
      boolean[] sincCond = new boolean[4];
```

```
        // el servidor ejecuta un bucle de servicio infinito
        while(true) {
              // Estudio de las CPREs
              sincCond[IL] = e == 0;
              sincCond[IE] = e == 0 && l == 0;
              sincCond[FL] = true;
              sincCond[FE] = true;

              switch(servicios.fairSelect(sincCond)) {
              case IL:
                    chIL.in().read();
                    l++;
                    break;

              case IE:
                    chIE.in().read();
                    e++;
                    break;

              case FL:
                    chFL.in().read();
                    l--;
                    break;

              case FE:
                    chFE.in().read();
                    e--;
                    break;

              } // switch
        } // while
    }
}
```

Se pide implementar el siguiente CTAD usando como mecanismo de sincronización el paso de mensajes:

C-TAD MultiCont

OPERACIONES
ACCIÓN inc: N[e]
ACCIÓN dec: N[e]

SEMÁNTICA
DOMINIO:
TIPO: MultiCont = N
INVARIANTE: $0 \leq self \wedge self \leq N$

INICIAL: self = 0

PRE: $n > 0 \wedge n < N/2$
CPRE: $self + n \leq N$
inc(n)
POST: $self = self^{PRE} + n$

PRE: $n > 0 \wedge n < N/2$
CPRE: $n \leq self$
dec(n)
POST: $self = self^{PRE} - n$

```
public class MultiContCSP implements CSProcess {

	private final int N;

	// Canales
	private Any2OneChannel chINC;
	private Any2OneChannel chDEC;

	public MultiContCSP (int capacidad) {
		N = capacidad;

		// Inicializamos los canales:
		chINC = Channel.any2one();
		chDEC = Channel.any2one();
	}

	public void inc(int n) {
		chINC.out().write(n);
	}

	public void dec(int n) {
		chDEC.out().write(n);
	}
```

```
	public void run() {
		// Inicializamos los atributos del recurso
		int multicont = 0;

		// Inicializamos los atributos auxiliares
		int n = 0;

		// soporte para recepcion alternativa condicional
		// Nombres simbolicos para los indices de servicios
		final int INC = 0;
		final int DEC = 1;

		// Entradas de la select
		final AltingChannelInput[] entradas = {chINC.in(), chDEC.in()};

		// Recepcion alternativa
		final Alternative servicios = new Alternative(entradas);

		// el servidor ejecuta un bucle de servicio infinito
		while(true) {
			switch(servicios.fairSelect()) {
			case INC:
				n = (int)chINC.in().read();
				// PRE
				if(n <= 0 || n >= N/2)
					throw new IllegalArgumentException();
				// CPRE
				if( multicont + n <= N)
					multicont += n; // POST
				break;

			case DEC:
				n = (int) chDEC.in().read();
				// PRE
				if(n <= 0 || n >= N/2)
					throw new IllegalArgumentException();

				// CPRE
				if(n <= multicont)
					multicont -= n; // POST
				break;

			} // switch
		} // while
	}
}
```