Representación de problemas con clases e instancias en CLIPS

1. El mismo problema pero con clases

Recordemos el problema que veíamos en la parte 1 del tutorial:
Representar el siguiente problema por medio de un sistema de producción. Tenemos dos jarras, una de 4 litros y otra de 3 litros. Ninguna de ellas tienen marcas que permitan identificar cuanta agua hay en ellas. Hay un grifo que permite llenar las jarras. El problema consiste en acabar teniendo exactamente 2 litros de agua en la jarra de 4 litros.

Supongamos ahora que se nos enuncia el problema de una manera ligeramente diferente:
En el mundo que vamos a representar existen varias clases de jarras, la más común es la mediana aunque también las puede haber grandes (4 litros), pequeñas (2 litros) y medianas (3 litros). Ninguna de ellas tiene marcas que permitan identificar cuanta agua hay en ellas. Hay un grifo que permite llenar las jarras. El problema consiste en acabar teniendo exactamente 2 litros de agua en alguna de las jarras.

2. La ontología o jerarquía de clases

En el enunciado nos aparece un tipo de entidad (la jarra), de la que sabemos que hay varios tipos: pequeñas, medianas y grandes, pero todas comparten la característica general de ser una jarra. En CLIPS este tipo de jerarquías con clases de objetos generales y otras clases mas especificas, las representamos con clases (o marcos) y sus instancias correspondientes (también llamadas objetos). De hecho la programación orientada a objetos comparte muchas características con los marcos. Nuestro caso quedaría representado con el siguiente código CLIPS:

```
(defclass JARRA (is-a INITIAL-OBJECT)
(slot litros
(type INTEGER)
(default 0))
(slot capacidad
(type INTEGER)
(default 3))
)
(defclass JARRA_PEQUEÑA (is-a JARRA)
(slot capacidad
(default 2))
)
(defclass JARRA_GRANDE (is-a JARRA)
(slot capacidad
(default 4))
)
```

Nos encontramos con las siguientes características:
- Hemos definido una clase general llamada JARRA con dos slots: litros y capacidad. Puesto que sabemos que frecuentemente las jarras van a ser medianas, el valor por omisión de litros es 3.
- Obsérvese que el segundo argumento de defclass es (is-a INITIAL-OBJECT). Esto es así porque todas las clases nuestras clases que estén en el nivel mas alto (no tienen super-clase de la que heredar atributos/slots) deben derivar de INITIAL-OBJECT.
- En la terminología clásica de marcos, a cada una de las propiedades de un slot se las denomina facetas (o “facets”). En nuestro caso, a la clase JARRA le hemos definido 2 facetas: el valor por omisión (default), el tipo del slot (type).

- Hemos definido dos clases más: JARRA PEQUEÑA y JARRA GRANDE. Ambas derivan de la clase genérica JARRA, como puede verse en (is-a JARRA). Al derivar de ella, heredan todos los slots de la clase madre. Por tanto, la clase JARRA GRANDE tendrá dos slots: litros y capacidad. El valor por omisión de litros sera 0, puesto que también se hereda de la clase madre. Sin embargo, el valor por omisión de capacidad se re-especifica en la clase hija, por lo que pasa a ser 4. No hemos definido una clase JARRA MEDIANA porque consideramos que ese tamaño represent la jarra normal, representada con la clase JARRA.

3. Las instancias iniciales

Además tenemos que especificar los objetos iniciales de nuestra base de hechos. Usaremos para ello la instrucción definstances, la cual, en cuanto tecleemos (RESET) añaadirá a la base de hechos las instancias indicadas en el definstances (además del initial-object). A continuación se muestra un ejemplo:

```
(definstances estado_inicial_jarras
(of JARRA_PEQUEÑA)
(of JARRA_GRANDE (litros 0))
)
```

Si quisiéramos identificar dichas instancias como j1 y j2 pondríamos:

```
(definstances estado_inicial_jarras
(j1 of JARRA_PEQUEÑA)
(j2 of JARRA_GRANDE (litros 0))
)
```

4. Las reglas

A continuación se muestra una de las reglas apropiadas para este problema. El alumno debería transformar el resto de ellas comparandolas con las que usábamos cuando no utilizábamos objetos.

```
; Llena jarra
(defrule llena-jarra
?jarra <- (object (is-a JARRA) (capacidad ?c) (litros ?l))
(test (< ?l ?c))
=>
(modify-instance ?jarra (litros ?c)))
```

Las dos diferencias que tienen esta regla que maneja instancias con respecto a la que solo manejaban hechos es:

- Se usa make-instance para crear instancias, modify-instance para modificarlas y unmakeinstance para borrarlas.
- La equiparación de instancias en la condición de la regla se hace con el patrón (object (is-a <la clase>) ...).

Por último, como sabemos que si estamos utilizando una jarra pequeña (2l) entonces la solución es inmediata: solo tenemos que llenar la jarra y ya tendremos dos litros en alguna jarra, podríamos representar esta regla de la siguiente manera:

```
(defrule solucion-rapida
?jarra <- (object (is-a JARRA_PEQUEÑA) (capacidad ?c))
=>
(modify-instance ?jarra (litros ?c))
)
```

5. Ejercicios

1. Construir el resto de las reglas que resuelve el problema de las jarras planteado, utilizando clases (y sin prioridades).

2. Construir una ontología o jerarquía de clases que clasifique animales. Deberían figurar las clases de: animal, mamífero, ave, hombre, albatros y pingüino. Los atributos que habría que especificar son: nombre, piel (posibles valores: pelo o plumas), vuela (si o no), razona (si o no).
3. Partiendo del ejercicio anterior, hacer que las instancias iniciales sean un hombre que se llame Pepe, un albatros que se llame Alf y un pingüino que se llame Chilly.
4. Escribir una regla que imprima todos los nombres de los pingüinos que haya en la base de hechos.

6. Entrega

Subir a AG los siguientes tres ficheros: la ontología (con las clases), las instancias (definstances) y las reglas del problema de las jarras por un lado y del de los animales por otro.