Tarea 6
CC4101 - Lenguajes de Programación

Éric Tanter
Auxiliar: Richard Ibarra

Fecha de Entrega. Viernes 9 de Julio *

En cada parte, provea todos las casos de tests necesarios para demostrar el buen funcionamiento y completitud de su propuesta.

1. Preparación: extienda el FAE con `define`, booleanos, más todas las primitivas que (re)quiera[1].

2. (2ptos) Extienda el FAE con objetos y delegación (OFAE). Una diferencia fundamental entre lo visto en clase y lo que tiene que implementar es que en el OFAE, al igual que en JavaScript, no hay distinción entre campos y metodos: soló hay *propiedades*.

```
{define o {object root
            {x 5}
            {set-x {fun {nx} {@! x nx}}}
            {get-x {fun {} {@? x}}}}}
> {@ o x}
5
> {{@ o set-x} 10}
> {{@ o get-x}}
10
```

   Como pueden observar:

   - se usa `@` para acceder a una propiedad de objeto desde afuera.
   - No es posible en este lenguaje cambiar una propiedad desde afuera de un objeto: tiene que proveer un *setter* (como `set-x`) para poder hacer eso.
   - `@?` y `@!` permiten acceder a las propiedades del objeto actual (`self`).
   - `{@? x}` es lo mismo que `{@ self x}`.
   - un metodo es solamente una propiedad cuyo valor es una función. Pueden definir `->` como azucar sintáctico para invocar un método: `{-> o set-x 10}` es equivalente a `{{@ o set-x} 10}`

*Ver las reglas de entrega y evaluación en `http://pleiad.cl/teaching/cc4101/reglas`.
[1]http://pleiad.cl/teaching/cc41a/primitivas

3. (2ptos) Extienda el OFAE para soportar *contratos* sobre slots: al igual que en el PLAI, un contrato es un predicado que retorna `true` si el valor pasado como argumento es correcto, y `false` sino. Settear un valor incorrecto en un campo provoca un error.

```
;; contract for positive numbers
{define positive-number
  {fun {v} {and {number? v}
                {> v 0}}}}

{define make-point
  {fun {init-x}
    {object root
       {x init-x : positive-number}
       {set-x {fun {nx} {@! x nx}}}
       {get-x {fun {} {@? x}}}}}}

{define p {make-point 10}}

> {{@ p set-x} -3}
error: value -3 violates field contract for x
```

Declarar contratos sobre propiedades debe ser opcional. Además, solamente tiene que soportar contratos para propiedades que son valores simples, es decir, no para métodos.

Pregunta opcional: (1pto) Extienda su mecanismo de contratos para soportar contratos para métodos. Un contrato de método incluye un contrato para el argumento, y un contrato para el resultado.

4. (2ptos) Extienda su lenguaje para soportar una noción explicita de *interfaces* para objetos. Una interfaz es un conjunto de nombres de propiedades:

```
{define i-point {interface {} get-x set-x}}
```

El primer argumento de `interface` es la lista de las interfaces de la cual la interfaz extiende.

Al definir un objeto uno ahora puede especificar la lista de interfaces que el objeto implementa. Si un objeto no define todas las propiedades de las interfaces mencionadas, se debe generar un error *al momento de construir el objeto*.

```
{define make-point
  {fun {init-x}
    {object root {i-point}
       {x init-x : positive-number}
       {get-x {fun {} {@? x}}}}}}

> {make-point 2}
error: object does not implement interface member set-x
```

Finalmente, defina una primitiva `implements?` que permite determinar si un objeto implementa una interfaz:

```
> {implements? p i-point}
#t
```

Ilustre su uso definiendo una fábrica de objetos linea, cuyos contratos aseguran que sus propiedades son puntos.