Estructuras

Dante Zanarini

15 de mayo de 2020

Necesitamos más información en el estado

- Pensemos en los ejercicios de la práctica 3
- ¿Por qué los objetos no se muevan en dos direcciones?
- ¿Y si queremos que el objeto cambie de color?
- Repasemos los otros ejemplos y veamos qué nos gustaría **agregarle al estado**

Necesitamos agrupar información

- Más allá de los programas interactivos, hay otras situaciones en las que necesitamos **agrupar datos.**
- Supongamos que queremos representar:
  - Una agenda,
  - el catálogo de una biblioteca,
  - colores,
  - puntos en el plano,
  - ...

Una estructura bien simple

- Volvamos al ejercicio del círculo en la práctica 3.
- Necesitamos representar una posición en el plano

Una estructura bien simple

- Volvamos al ejercicio del círculo en la práctica 3.
- Necesitamos representar una posición en el plano
- Cómo representamos esta información?
- Echémosle un vistazo a la estructura **`posn`**

Una estructura bien simple

- Volvamos al ejercicio del círculo en la práctica 3.
- Necesitamos representar una posición en el plano
- Cómo representamos esta información?
- Echémosle un vistazo a la estructura **`posn`**
- Usando **`posn`** podemos repensar el ejercicio 4, para mover el objeto en ambas direcciones...

De dónde sale posn?

- *Racket* nos permite definir

  nuevos tipos de datos, agrupando los ya conocidos

De dónde sale posn?

- *Racket* nos permite definir

  nuevos tipos de datos, agrupando los ya conocidos

  - En el caso de **`posn`**:

```
(define-struct posn [x y])
```

De dónde sale posn?

- *Racket* nos permite definir
  nuevos tipos de datos, agrupando los ya conocidos
  - En el caso de **`posn`**:

```
(define-struct posn [x y])
```

  - En general:

```
(define-struct Nombre [Campo1 ... CampoN])
```

- La palabra clave **`define-struct`** indica que estamos definiendo un nuevo tipo de datos
- Luego indicamos su nombre, (**`Nombre`**)
- Finalmente, una lista con los nombres de los campos que incluye la estructura

Qué hacemos cuando definimos una estructura?

```
(define-struct Nombre [Campo1 ... CampoN])
```

- **Se incorporan varias funciones:**
  - Un *constructor* que permite crear elementos en el nuevo tipo.
  - Un *selector* por cada campo. Permiten observar el valor de cada uno.
  - Un *predicado* que distingue instancias de la clase creada de otros objetos.

Qué hacemos cuando definimos una estructura?

```
(define-struct Nombre [Campo1 ... CampoN])
```

- **Se incorporan varias funciones:**
  - Un *constructor* que permite crear elementos en el nuevo tipo.
  - Un *selector* por cada campo. Permiten observar el valor de cada uno.
  - Un *predicado* que distingue instancias de la clase creada de otros objetos.
- Repasemos todo esto con **`posn`**

Otras estructuras...

```
(define-struct contacto [nombre tel mail])
; contacto es (make-contacto String String String)
```

- *constructor:* **`make-contacto`**
  - Ejemplo:
    ```
    (make-contacto "Juan" "3416-342356" "jj@gmx.net")
    ```
- *selectores:* **`contacto-nombre`**, **`contacto-tel`** y **`contacto-mail`**
  - Ejemplo:
    ```
    (contacto-mail (make-contacto "Juan" "3416-342356" "jj@gmx.net"))
    ```
- *predicado:* **`contacto?`**. Funciona como los predicados para los tipos de datos conocidos.

Leyes de evaluación...

- Las leyes de evaluación para estructuras relacionan el constructor con los selectores
- Por ejemplo, para la estructura **`posn`** :

```
(posn-x (make-posn a b))
== ley 1
a

(posn-y (make-posn a b))
== ley 2
b
```

- Observemos que un elementos de la forma **`(make-posn a b)`** ya está en forma canónica y no requiere ser evaluado.