Complicando el estado

Dante Zanarini

12 de Abril de 2016

Necesitamos más información en el estado

- Pensemos en el ejercicio 2 de la práctica 2
- Por qué no permitimos que el objeto se mueva horizontalmente?
- 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,
  - ...

Una estructura bien simple

- Volvamos al ejercicio 2 de la práctica.
- Necesitamos representar una posición en el plano

Una estructura bien simple

- Volvamos al ejercicio 2 de la práctica.
- 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 2 de la práctica.
- Necesitamos representar una posición en el plano
- Cómo representamos esta información?
- Echémosle un vistazo a la estructura **`posn`**
- Usando **`posn`** podemos repensar el ejercicio 2, para mover el objeto en ambas direcciones...

De dónde sale posn?

- *Racket* nos permite definir

  clases de estructuras

De dónde sale posn?

- *Racket* nos permite definir
  clases de estructuras
- En el caso de **`posn`**:

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

De dónde sale posn?

- *Racket* nos permite definir
  clases de estructuras
- 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 una nueva clase de valores
- Luego indicamos el nombre para esta nueva clase, (**`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 instancias de la clase.
  - Un *selector* por cada campo. Permiten acceder a las componentes de una instancia.
  - 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 instancias de la clase.
  - Un *selector* por cada campo. Permiten acceder a las componentes de una instancia.
  - 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.