

75.10 Técnicas de diseño

Trabajo Práctico 1: Validador de ofertas

Fecha de Entrega: 3/10/2018

Integrantes	Padrón
Augusto Arturi	97498
Pablo Inoriza	94986
Manuel Luis Llauro	95736
Sebastián Ezequiel Blanco	98539

Índice

1. Modelo del dominio	1
2. Explicacion de los modulos	2
2.1. offer_processor	2
2.2. offer_applier	3
2.3. rule_applier	5
2.4. insertions	6
2.5. operators	7
2.6. fields	8
2.7. exceptions	9
2.8. conversions	10
2.9. translations	11

1. Modelo del dominio

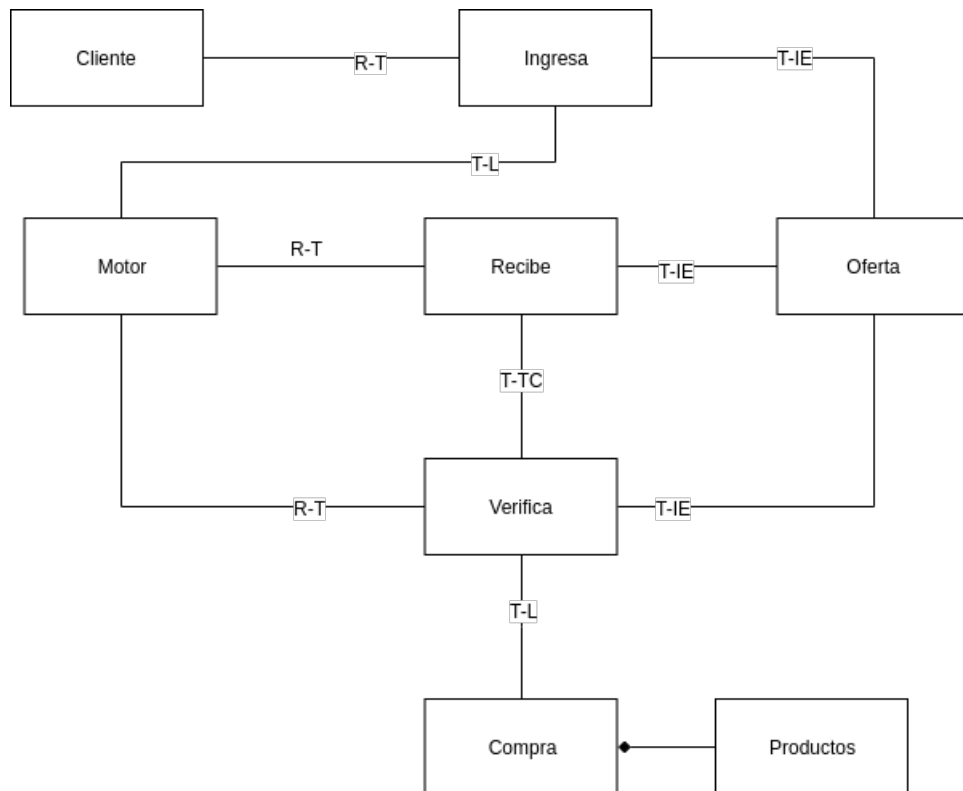


Figura 1: Modelo del dominio

2. Explicacion de los modulos

2.1. offer_processor

Este módulo es el principal del trabajo y contiene la interfaz pública que se provee al usuario, donde existen dos funciones:

```
1 [state] initialize-offers [offers rules]
2 [offers-applied] = process-sale [state, sale]
3
```

Listing 1: firmas de la interfaz pública

La primer función (initialize-offers) recibe una lista de ofertas y una lista de reglas y las almacena retornando un valor que representa una estado.

La segunda función (process-sale) recibe el estado que se obtuvo de la función anterior y recibe una compra que contiene productos, un calendario y una forma de pago, retornando un vector donde cada elemento representa una oferta que aplico a la compra con el respectivo descuento.

2.2. offer_applier

Este módulo tiene una sola función que obra de interfaz pública y es la siguiente:

```
1 [offers-applied] = apply_offer [offer sale]
2
```

Listing 2: firmas de la interfaz pública

Esta función recibe una única oferta que se la aplica a la respectiva compra (sale) retornando un vector de mapas donde cada uno tiene el siguiente formato:

```
1 {
2   "description" "una descripcion"
3   "offer_code" "un codigo"
4   "discount" "un valor"
5 }
6
```

Listing 3: mapa resultado de ofertas aplicadas

Cada oferta tiene un porcentaje de descuento y una compra tiene varios productos, por lo tanto en cada uno de estos mapas *"discount"* es el valor que se descuenta por cada producto que cumple con la oferta.

Luego tenemos funciones internas del módulo que resuelven otros problemas y las firmas son estas:

```
1 [products] get_products_that_met_the_rules [offer sale]
2 [offers-applied] get_offer_result [offer products]
3 [discount-value] apply_discount [offer price]
4
```

Listing 4: firmas de las funciones privadas

La primer función recibe una oferta y una compra y retorna una lista de productos los cuales cumplen con la oferta. Cada producto de este vector que retorna tiene un formato nuevo con un criterio propio. El formato de cada producto es cambiado dentro de esta función para facilitar el diseño del código y con un ejemplo mostramos como representamos un producto:

```
1 {
2   'products' {
3     'name' 'Leche Descremada 1L, la Calmisima'
4     'brand' {
5       'code' 'Z001ABC'
6       'name' 'La Calmisima'
7     }
8     'category' {
9       'code' 'X033AXX'
10      'name' 'Lacteo'
11    }
12    'price' 25.40
13    'iva_percentage' 10.5

```

```
14     'code' 'X033XXX'
15   }
16   'payment' {
17     'method' 'CASH'
18     'bank' 'CAPRO'
19   }
20   'purchase_date' {
21     'year' '2018'
22     'month' 'SEPTEMBER'
23     'day_number' 20
24     'week_day' 'Thursday'
25     'week_number' 4
26   }
27 }
28
```

Listing 5: formato de un producto

La segunda función (`get_offer_result`) recibe una oferta y una lista de productos (con el formato ya explicado) que cumplen con dicha oferta y retorna un vector de oferta aplicadas cuyo formato de cada oferta aplicada se explicó anteriormente (ver Listing 3: mapa resultado de ofertas aplicadas).

2.3. rule_applier

Este módulo tiene una sola función que obra de interfaz pública y es la siguiente:

```
1 [boolean-vector] = apply_rules [rules_codes prod]
2
```

Listing 6: firmas de la interfaz pública

Esta función recibe un vector de codigos de reglas y un producto con un formato explicado anteriormente (ver Listing 5: formato de un producto) y retorna un vector de booleano donde cada uno representa si el producto cumple o no con cada regla (true: cumple, false: no cumple).

Luego tenemos funciones internas del módulo que resuelven otros problemas y las firmas son estas:

```
1 [rule] get_rule [rule_code]
2 [boolean] atomic_rule [rule prod]
3 [boolean] apply_atomic_rule [rule prod]
4 [boolean-vector] multiple_rules [rules_codes prod]
5 [boolean-vector] apply_multiple_rules [rules_codes prod]
6
```

Listing 7: firmas de las funciones privadas

La primer función (get_rule) recibe el código de una regla y devuelve la regla. La segunda función (atomic_rule) es un multimétodo el cual recibe una regla y un producto con un formato explicado anteriormente (ver Listing 5: formato de un producto) y devuelve un booleano que nos dice si el producto cumple o no con la regla. Esto es un multimétodo ya que no sabemos si la regla es atómica o es una regla con subreglas, por lo tanto si la regla es atómica ejecuta la función apply_atomic_rule, la cual recibe una regla atómica y un producto y retorna un booleano. Si la regla no es atómica llama a otro multimétodo que es multiple_rules, que recibe una regla no atómica y el producto. Si la regla no atómica esta compuesta por una única subregla ejecuta apply_atomic_rule donde recibe la subregla y el producto. Si la regla no atómica esta compuesta por mas de una subregla ejecuta apply_multiple_rules donde recibe un vector de codigos de subreglas y el producto y retorna un vector de booleanos.

2.4. insertions

Este módulo tiene dos funciones que obran de interfaz pública y son las siguientes:

```
1  [ nil ] add_offer [o]  
2  [ nil ] add_rule [r]  
3
```

Listing 8: firmas de la interfaz pública

La primer función recibe un vector de ofertas y lo almacena en una variable global llamada `offers_vector`. Dentro de la función hace chequeos de datos y en caso de haber un dato inválido retorna excepción.

La segunda función recibe un vector de reglas y lo almacena en una variable global llamada `rules_vector`. Dentro de la función hace chequeos de datos y en caso de haber un dato inválido retorna excepción.

2.5. operators

Este módulo tiene una función que obra de interfaz pública y es un multimétodo y es la siguiente:

```
1 [boolean] apply_op [op values field]
2
```

Listing 9: firmas de la interfaz pública

Esta función recibe un string (op) que representa la operación a realizar y dependiendo que tipo de operación values y/o field tomas distintos formatos. Si se quiere ejecutar las operaciones AND, OR, NOT , values sera un vector de booleanos al cual de le aplicara dicha operación y field sera nil, ya que no se usara. Si se quiere ejecutar las operaciones LOWER, HIGHER, EQUALS values sera un valor singular y field tambien. Si se quiere ejecutar la operación IN, values sera un vector de valores y field sera un valor singular.

2.6. fields

Este módulo tiene una sola función que obra de interfaz pública y es la siguiente:

```
1 [field] get_field [product rule]
2
```

Listing 10: firmas de la interfaz pública

Esta función recibe un producto con un formato explicado anteriormente (ver Listing 5: formato de un producto) y una regla y retorna el campo del producto el cual la regla especifica.

Luego tenemos funciones internas del módulo que resuelven otros problemas y las firmas son estas:

```
1 [path] rule_field_path [rule]
2 [key] translate [value])
3
```

Listing 11: firmas de las funciones privadas

La primer función recibe una regla y retorna un vector de claves de un mapa en el cual se encuentra el valor buscar en el producto.

La segunda función es un multimétodo y recibe una clave que representa parte del camino (path) para encontrar el field y lo traduce a el nombre que el mapa usa, ya que por ejemplo las reglas usan la palabra CALENDAR para referirse al campo purchase_date de un producto.

2.7. exceptions

Este módulo tiene cuatro funciones que obran de interfaz pública que son multimetodos y son las siguientes:

```
1  [field/Exception] check_field [field msg]
2  [id/Exception] check_unknown_id [id]
3  [rules/Exception] check_cycle_id [rules]
4  [ids/Exception] check_duplicate_codes [ids code]
5
```

Listing 12: firmas de la interfaz pública

La primer función recibe un campo y un mensaje y si dicho campo es inválido, es decir tiene un valor que no esta permitido, retorna una excepción con el mensaje. En caso contrario retorna el campo.

La segunda función recibe un id que puede estar en formato json o no y si no contiene ningun dato o es un nill retorna excepción. En caso contrario retorna el id.

La segunda función recibe un vector de reglas y si existen reglas no atomicas las cuales son ciclicas retorna una excepción. En caso contrario retorna las reglas.

La segunda función recibe un vector ids que podrian ser tanto reglas como ofertas y si existen dos id con el campo code en el mismo valor retorna excepción. En caso contrario retorna las reglas.

2.8. conversions

Este módulo tiene dos funciones que obran de interfaz pública y son las siguientes:

```
1  [structure] json_to_map [j]  
2  [json]    map_to_json [j]  
3
```

Listing 13: firmas de la interfaz pública

La primera función recibe un string (json) y lo retorna en la estructura de datos correspondiente de clojure.

La segunda función recibe una estructura de datos clojure y la convierte en string (json).

2.9. translations

Este módulo tiene una sola función que obra de interfaz pública que es un multimétodo y es la siguiente:

```
1 [month] translate [month]  
2
```

Listing 14: firmas de la interfaz pública

Esta función recibe un mes en español o ingles y lo retorna en español. Esto se debe a que la veces se termina comparando los mismos meses en distintos idiomas y la comparacion da falso, por lo tanto no aseguramos de tener todo en un mismo idioma.