75.10 Tecnicas de diseño

Trabajo Práctico 1: validador de ofertas

Fecha de Entrega: 3/10/2018

Integrantes	Padron
Augusto Arturi	9....
Pablo inoriza	9....
Manuel, Llauro	9....
Sebastián Ezequiel Blanco	98539

Índice

1. Modelo del dominio

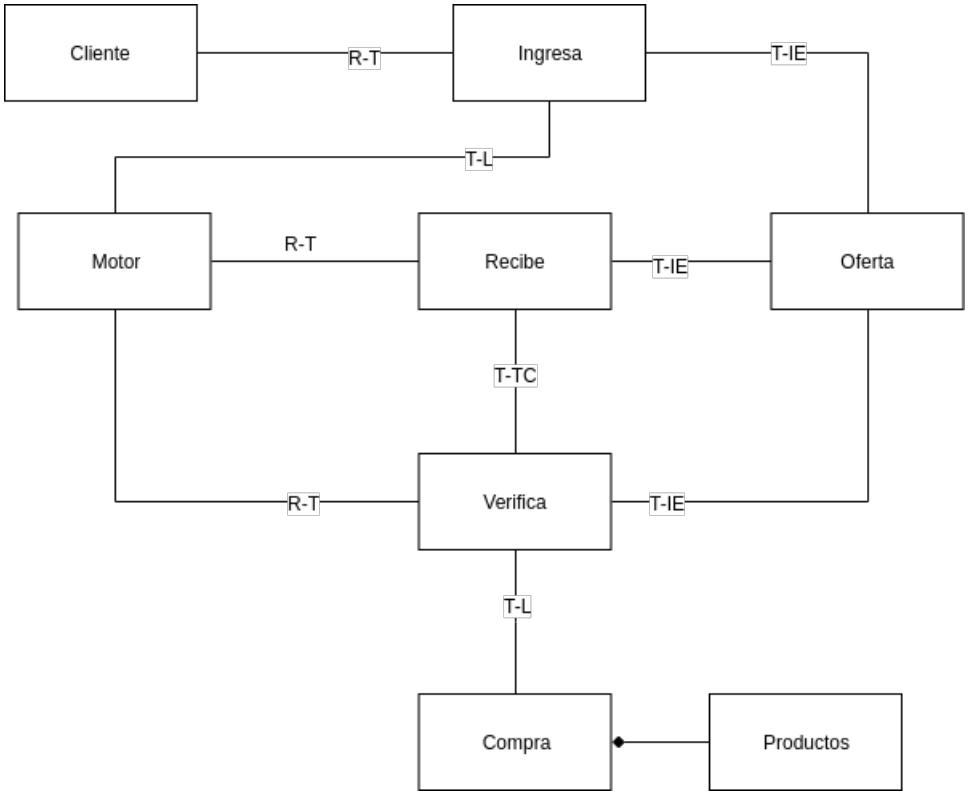

Figura 1: Modelo del dominio

2. Explicacion de los modulos

2.1. offer_processor

Este modulo es el principal del trabajo y contiene la interfaz publica que se provee al usuario, donde exiten dos funciones:

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

Listing 1: firmas de la interfaz publica

La primer funcion (initialize-offers) recibe una lista de ofertas y una lista de reglas y las almacena retornando un valor que representa una estado.
La segunda funcion (process-sale) recibe el estado que se obtubo de la funcion anterior y recibe una compra que contiene productos, un calendario y una forma de pago, retornando un vector donde cada elemneto representa una oferta que aplico a la compra con el respectivo descuento.

2.2. offer_applier

Este modulo tiene una sola funcion que obra de interfaz publica y es la siguiente:

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

Listing 2: firmas de la interfaz publica

Esta funcion recibe una unica oferta se la aplica a la respectiva compra (sale) retornando un vector de mapas donde cada uno tiene el sigueinte 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 modulo que resuelven otros porblemas 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 funcion 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 funcion para facilitar el diseño del codigo 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_porcentage' 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 funcion (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 explico anteriormente (ver Listing 3: mapa resultado de ofertas aplicadas).

2.3. rule_applier

Este modulo tiene una sola funcion que obra de interfaz publica y es la siguiente:

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

Listing 6: firmas de la interfaz publica

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

Luego tenemos funciones internas del modulo que resuelven otros porblemas 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 funcion (get_rule) recibe el codigo de una regla y devuelve la regla.
La segunda funcion (atomic_rule) es un multimetodo 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 multimetodo ya que no sabemos si la regla es atomica o es una regla con subreglas, por lo tanto si la regla es atomica ejecuta la funcion apply_atomic_rule, la cual recibe una regla atomica y un producto y retorna un boleano. Si la regla no es atomica llama a otro multimetodo que es multiple_rules, que recibe una regla no atomica y el producto. Si la regla no atomica esta compuesta por una unica subregla ejecuta apply_atomic_rule donde recbe la subregla y el producto. Si la regla no atomica esta compuesta por mas de una subregla ejecuta apply_multiple_rules donde recibe un vector de codigos de subreglas y el producto y retonra un vector de booleanos.

2.4. insertions

Este modulo tiene dos funciones que obran de interfaz publica y son las siguientes:

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

Listing 8: firmas de la interfaz publica

La primer funcion recibe un vector de ofertas y lo almacena en una variable global llamada offers_vector. Dentro de la funcion hace chequeos de datos y en caso de haber un dato invalido retorna excepcion.
La segunda funcion recibe un vector de reglas y lo almacena en una variable global llamada rules_vector. dentro de la funcion hace chequeos de datos y en caso de haber un dato invalido retorna excepcion.

2.5. operators

Este modulo tiene una funcion que obra de interfaz publica y es un multimetodo y es la siguiente:

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

Listing 9: firmas de la interfaz publica

Esta funcion recibe un string (op) que representa la operacion a realizar y dependiendo que tipo de operacion values y/o field tomas distintos formatos. Si se quiere ejecutar las operaciones AND, OR, NOT , values sera un vector de boleanos al cual de le apicara dicha operacion 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 operacion IN, values sera un vector de valores y field sera un valor singular.

2.6. fields

Este modulo tiene una sola funcion que obra de interfaz publica y es la siguiente:

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

Listing 10: firmas de la interfaz publica

Esta funcion 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 modulo que resuelven otros porblemas 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 funcion recibe una regla y retorna un vector de claves de un mapa en el cual se encuetra el valor buscar en el producto.
La segunda funcion es un multimetodo 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 modulo tiene cuatro funciones que obran de interfaz publica 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 publica

La primer funcion recibe un campo y un mensaje y si dicho campo es invalido, es decir tiene un valor que no esta permitido, retorna una excepcion con el mensage. En caso contrario retorna el campo.
La segunda funcion recibe un id que puede estar en formato json o no y si no contiene ningun dato o es un nill retorna excepcion. En caso contrario retorna el id.
La segunda funcion recibe un vector de reglas y si existen reglas no atomicas las cuales son ciclicas retorna una excepcion. En caso contrario retorna las reglas.
La segunda funcion recibe un vector ids que podrian ser tanto reglas como ofertas y si existen dos id con el campo code en el mismo valor retirna excepcion. En caso contrario retorna las reglas.

2.8. convertions

Este modulo tiene dos funciones que obran de interfaz publica y son las siguientes:

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

Listing 13: firmas de la interfaz publica

La primera funcion recibe un string (json) y lo retorna en la estructura de datos correspondiente de clojure.
La segunda funcion recibe una estructura de datos clojure y la convierte en string (json).

2.9. translations

Este modulo tiene una sola funcion que obra de interfaz publica que es un multimetodo y es la siguiente:

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

Listing 14: firmas de la interfaz publica

Esta funcion 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.