FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico Nº1: Validador de Reglas

_ Grupo Nº 07.

_Integrantes:

Fontana, Lucas - Padrón: 95814 -

Rodríguez, Rómulo - Padrón: 86816 -

_Fecha de entrega: 19/04/2018.

Comentarios iniciales:

1_ A partir del enunciado se pretende primer entender el problema, de qué trata el negocio.

Para empezar Martin Fowler en su libro UML gota a gota, hace mención a riesgos, en particular riesgos de habilidades y riesgos tecnológicos.

En los tecnológicos, empezamos con cero conocimiento de programación funcional (clojure). No es ajeno a otros alumnos.

Con el doble propósito de lograr un mayor dominio de clojure (programación funcional, ahondar enrecursividad y entender el
enunciado-Tests, se contruye un prototipo ,al estilo de lo mencionado en LeanUX, donde los supuestos hay que validarlos luego. De esta manera generar conocimiento validado.
(rama_de_pruebas_consultas).

2_ De este prototipo salen correcciones, reinterpretaciones, aclaraciones, que se ponen en común
con los docentes el Jueves previo a la entrega 12-04-2018.

3_ De aquí resulta una división de responsabilidades:
Lucas: de lo “troncal” (data_proccesor.clj)
Rómulo : de las funciones recursivas, las expresiones, diccionarios.

También por la clase sobre git: Ya se usa un editor para los mensajes del commit (el nano). Además de encontrar un plugin “gitk”. Que muestra de manera gráfica lo que expusieron en clases
(sobre el git log). Entre otras cosas.

4_En el avance del proyecto se tienen distintos módulos y tests respectivos.

5_A partir de la entrega y preparando la reentrega 19-04-2018, se hace una reasignación de responsabilidades.
Rómulo módulo utiles.operacionesRecursivas (utilizando multimétodos y resultando como impresión personal más legible el código). También los tests
Lucas: reimplementación data-proccesors (eliminación de defs globales, entre otros, funcionesEspeciales (counter-value,counter-step, current, past, etc.

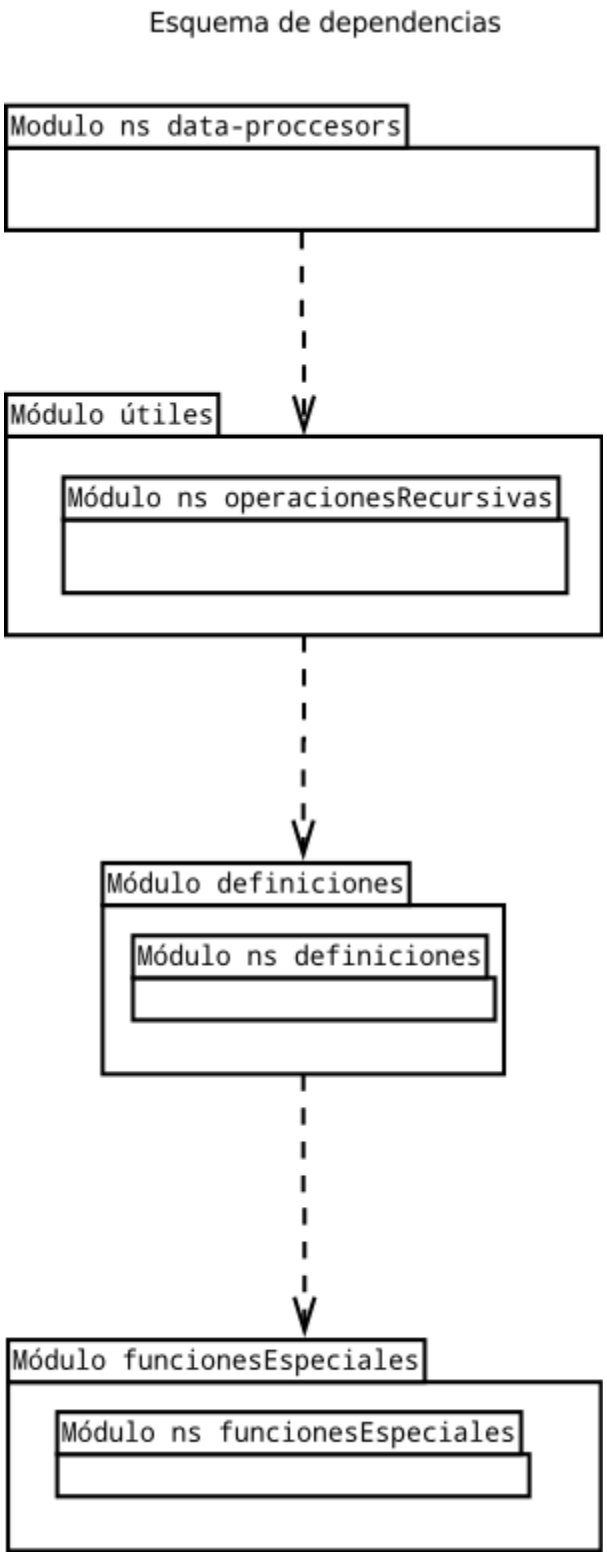
•

Estructura Proyecto:

```
└── TDD-2018-01-G07
    ├── CHANGELOG.md
    ├── doc
    │   ├── Enunciado
    │   │   ├── 2018-1C - TP-1.1 - Validador de Reglas - AceptaciOn.pdf
    │   │   ├── 2018-1C - TP-1.1 - Validador de Reglas.pdf
    │   │   ├── 7510-functional-rule-validator_acceptance_test.pdf
    │   │   └── Informe.pdf
    │   └── intro.md
    ├── LICENSE
    ├── project.clj
    ├── README.md
    ├── src
    │   ├── data_processor.clj
    │   ├── definiciones
    │   │   └── definiciones.clj
    │   ├── funcionesEspeciales
    │   │   └── funcionesEspeciales.clj
    │   └── utiles
    │       └── operacionesRecursivas.clj
    └── test
        ├── acceptance_test.clj
        ├── extension_test.clj
        └── utiles
            └── operacionesRecursivas_test.clj

9 directories, 17 files
```

Estructura de dependencia de módulos.



Estructura de estado:
(tipo map)

```
{       :Contadores {},
        :ContadorSteps {},
        :Sennales (:Sennales state),
        :DatosPasados pasado}
```

Definición de constantes (“def” tipo)

- (def cero_inicial [0])
- **Diccionarios**:
  - (def diccionario_op_igualdades {'= = 'not= not=})
  - (def diccionario_op_logicas {'and #'operar_con_AND 'or #'operar_con_OR})
  - (def diccionario_op_comparativas {'< < '> > '<= <= '>= >=})
  - (def diccionario_op_aritmeticas {'+ + '- - '* * '/ / 'quot quot})
  - (def diccionario_op_NOT {'not not});casos particulares
  - (def diccionario_op_MOD {'mod mod});casos particulares
  - (def diccionario_op_Func_especiales {'counter-value #'counter-value 'current #'current 'past #'past})
  - (def diccionario_op_strings {'concat str 'includes? clojure.string/includes? 'starts-with? clojure.string/starts-with? 'ends-with? clojure.string/ends-with?})

Módulo data_proccesors

PRINCIPALES

(defn initialize-processor [rules])
Recorre las nuevas reglas una vez por cada tipo de regla y produce un mapa por cada tipo de regla Usando el nombre como llave de mapa y los datos de la regla como los datos del mapa. Guarda todo en State, el cual es mapa.

(defn process-data [state new-data])
Recibe el dato y lo pasa por todas las reglas. Revisando en cada una si debe actuar o no. Luego guarda el dato en un mapa de datos pasados sin repeticiones.

(defn query-counter [state counter-name counter-args])
revisa en los contadores si hay uno con nombre "counter-name" y sino se fija en los ContadoresSteps.
Si esta devuelve el dato. Sino esta en ninguno devuelve cero.
Si un Contador y ContadorStep se llaman igual. Devuelve el Contador. (No llamarlos igual)

Internas

(defn identificarReglas [rules])
Hace todo el proceso de initialize-processor

(defn past [clave pasdo])
(defn current [clave actual])
reciben un mapa "pasado/actual" y devuelven el valor de la clave

(defn counter-value [counter-name counter-args state])
Hace todo el proceso de query-counter

(defn validarCondiciones [condiciones new-data state])
Valida la condición. Con todas las posibilidades de past.

(defn ejecutarSenneal [clave_Sennales new-data state])
Ejecuta la señal y devuelve el resultado.

(defn ejecutarFuncionRecursiva [funcion state pasdo actual])
Ejecuta una función en String. El State, pasdao y actual es para las funciones past, current y counter-value

(defn CargarPast [dato pasado])
recibe un mapa "pasado" y un dato. Pone el dato en el mape. Usandolo de lalve y poniendo Tru como valor.

(defn incrementar [contador cuanto])
Recibe un vector y incrementa su elemento en "cuánto". Lo devuelve como un vector de un elemento

(defn validarSennales [clave_Sennales new-data state])
se fija que las condiciones se cumple mediante. "validarCondiciones" si se cumple llama a ejecutarSenneal. Sino devuelve Nil

(defn aplicar-reglas-Sennales [state new-data])
Recorre el mapa de Seniales llamando a Validar Sennal con cada una de ellas

(**defn incrementar_Contador_simple [counter-name argFN reglas])**
Incrementa el contador .“argFN” es cuanto se incremanta

(defn incrementar_Contador_mapa [counter-name new-data argFN reglas]
Incrementa la opcion del contador que cumple con los pareamtros “argFN” es cuanto se incremanta

(defn incrementar_Contador [counter-name new-data argFN state])
(defn incrementarContadorStep [counter-name new-data argFN state])
Se fija si el contador tiene o no parametros y llama a la funcion para incrementarlo.

(defn validarContador [clave_contador new-data state])
(defn validarContadorSteps [clave_contador new-data state])
se fija que las condiciones se cumple mediante.
“incrementar_Contador/incrementarContadorStep” si se cumple llama a ejecutarSenneal. Sino deveulve el contador sin incrementar

(**defn aplicar-reglas-Contador [state new-data])**
(**defn aplicarReglasContadorSteps [state new-data])**
Recorre el mapa de contadores/contadoresStep llamando a validarContador/validarContadorSteps con cada una de ellas.

Módulo Utiles.operacionesRecursivas

Principal

(defn resolver_operacion [coleccion]
"Retorna el resultado de la operación que se recibe tipo List.
Ejemplos:
'(+ 5 3)
'(mod 5 3)
'(and true false 5)
'(<= 10 9 8 (+ 5 2))
'(str Hola Mundo)
'(= 2 (mod 5 3)).
Considerando que hay un argumento adicional para ejecutar_operación:
recibe (arg1 arg2). Donde arg1 es la expresión (coleccion en si misma de operaciones) y arg2 es un valor, 'estado o condición', de ser necesario como en el caso de funcionesEspeciales (counter, current, etc)."

Internas

;**********Funciones para diccionarios**********

(defn obtener_operador_var_de_symbol
"Se recupera el caracter var de un operador symbol.Si no lo encuentra retorna nil."
[operador_symbol]
;*********Funciones recursivas requeridas para MULTIMETODO*********
(defn iniciar_recursion_en_operaciones_logicas
"Ocurre la recursión de la función."
[coleccion estado_condicionado]
;**********

(defn iniciar_recursion_en_operaciones_igualdad
"Ocurre la recursión de la función. Para todo valor.Caso = y not=."
[coleccion estado_condicionado]
;**********

(defn iniciar_recursion_en_operaciones_aritmeticas
"Ocurre la recursión de la función. Operaciones aritmeticas."
[coleccion estado_condicionado]
;**********

(defn iniciar_recursion_en_operaciones_comparativas
"Ocurre la recursión de la función."
[coleccion estado_condicionado]
;**********

(defn iniciar_recursion_en_operaciones_mod
"Ocurre la recursión de la función mod"
[coleccion estado_condicionado]
;**********Funciones de desestructuracion**********

```
(defn desestructurar
  "Desestructura coleccion en  algunos elementos útiles para otras funciones."
  [coleccion estado_condicionado]
;**************************************************************
(defn desestructurar_coleccion_y_calcular_funciones_de_igualdad
;**************************************************************

(defn desestructurar_coleccion_y_calcular_funciones_logicas
  "Desestructura la colección en operador, operandos y resto de la colección. Para
operaciones lógicas."
  [coleccion estado_condicionado]
;**************************************************************

(defn desestructurar_coleccion_y_calcular_Funcion_MOD
  "Desestructura la colección en operador, operandos y resto de la colección. Operador
mod que recibe dos argumentos."
  [coleccion estado_condicionado]
;
**************************************************************************
**
(defn desestructurar_coleccion_y_calcular_funciones_comparativas
  "Desestructura la coleccion en operador, operandos y resto de la coleccion."
  [coleccion estado_condicionado]

;
**************************************************************************
**
(defn desestructurar_coleccion_y_calcular_funciones_aritmeticas
 "Desestructura la colección en operador, operandos y resto de la colección. Control de
la division por cero"
 [coleccion estado_condicionado]
**************************************************************************
*
(defn desestructurar_coleccion_y_calcular_Funcion_NOT
        "Desestructura la colección en operador, operandos y resto de la colección.
Operador NOT que recibe UN argumento."
        [coleccion estado_condicionado]
;
**********************MULTIMETODO**********************************
*
(defn determinar_operacion [colec estado_condicionado]

;Definicion de MULTIMETODO.
(defmulti ejecutar_operacion determinar_operacion)

;**************************METODOS**********************************

(defmethod ejecutar_operacion :Igualdad resolver_operaciones_de_igualdad [coleccion
estado_condicionado]
```

```
(defmethod ejecutar_operacion :Aritmetica resolver_operacion_aritmetica [coleccion
estado_condicionado]
(defmethod ejecutar_operacion :Comparativa resolver_operacion_comparativa
[coleccion estado_condicionado]
(defmethod ejecutar_operacion :Logica resolver_operacion_logica [coleccion
estado_condicionado]
(defmethod ejecutar_operacion :MOD resolver_operacion_mod [coleccion
estado_condicionado]
(defmethod ejecutar_operacion :NOT resolver_operacion_not [coleccion
estado_condicionado]

(defmethod ejecutar_operacion :Funcion_Especial resolver_operaciones_especiales
[coleccion (defmethod ejecutar_operacion :Funcion_Para_Strings
resolver_operaciones_en_strings
  ;"Funciones requeridas para strings. Abarca: includes?, starts-with?, ends-with?"
   [coleccion estado_condicionado]
(defmethod ejecutar_operacion :Coleccion_Vacia retornar_coleccion_Vacia [coleccion
estado_condicionado]
  ;"Función para cuando se recibe una coleccion sin elementos. retorna ()."
(defmethod ejecutar_operacion nil operacion_anulada [coleccion estado_condicionado]

(defmethod ejecutar_operacion :default operacion_Argumentos_Incorrectos [coleccion
estado_condicionado]
**********FIN MÉTODOS**********
```

Modulo funcionesEspeciales.funcionesEspeciales:

(defn operar_con_AND [coleccion]
(encapusla la funcionalidad de función and).
(defn operar_con_OR [colección]
(encapsula la funcionalidad de OR).
(defn past [clave pasdo]
(defn current [clave actual]
(defn counter-value [counter-name counter-args state]