# RELAZIONE PROGETTO VALUTATORE ESPRESSIONE

La classe creata si chiama “ValutaExpr”.

Essa ha una sola variabile d’istanza e consiste in una stringa expr, che appunto sarà quella che verrà presa in esame dall’algoritmo. La classe contiene di conseguenza un

* **ValutaExpr(String s) :** costruttore che richiede come parametro appunto la stringa da inserire, e la assegna come variabile d’istanza.
* **risolvi() :** ritorna un int (soluzione) e ha il compito di aprire uno StringTokenizer sulla stringa expr, con delimitatori i simboli corrispondenti alle operazioni matematiche di: addizione, sottrazione, divisione, moltiplicazione, resto/modulo, elevamento a potenze e chiusura/apertura delle parentesi tonde. Ho previsto anche il terzo parametro del costruttore dello StringTokenizer ovvero “true”, in modo da restituire come token anche i delimitatori. Successivamente ritorna il metodo valutaEspressione(StringTokenizer st) che a sua volta ritorna un int, passandogli come parametro lo StringTokenizer appena aperto.
* **valutaEspressione(StringTokenizer st) :** ritorna anch’esso un int e ha come parametro uno StringTokenizer. Come prima cosa vengono istanziati i due stack locali operandi e operatori, che rispettivamente conterrano gli operandi (numeri) e gli operatori che lo StringTokenizer troverà sulla stringa (variabile d’istanza). A questo punto un while, che opera fin quando st avrà prossimi token, li scorre e in base se essi corrispondono al pattern matching al quale corrispondono:

**-** se è un operando, il numero viene subito posto in testa allo stack a lui dedicato con una operazione di push().

**-** se è un operatore: se lo stack corrispondente è vuoto o l’operatore in questione ha maggiore priorità matematica di quello presente in testa a questo stack, esso viene posto in testa con una operazione di push(). \*Altrimenti si procede con prendere due operandi presenti in testa (testa e testa-1) con due operazioni di pop() e un operatore sempre con una operazione di pop() dallo stack corrispondente. Successivamente si esegue l’operazione operando1 (operatore) operando2. Il risultato viene messo in testa allo stack “operand” e l’operatore originariamente incontrato dallo StringTokenizer inserito in testa allo stack “operatori”. In questa fase dell’algoritmo è stato inserito un try..catch in caso le operazioni appena citate facciano sorgere un’eccezione, andando così a riportare che la stringa è malformata.

**-** nel caso in cui il token sia equivalente a una parentesi tonda aperta “(“ viene richiamato ricorsivamente il metodo valutaEspressione(st) e il risultato prodotto dalla richiamata ricorsiva di tale metodo, viene posto in testa dello stack ”operandi”.

**-** nel caso in cui il token sia equivalente alla parentesi tonda chiusa “(“ il metodo chiamato ricorsivamente viene fermato con una operazione i break.

**-** se il token non rientra in nessuna delle alternative elencate viene sollevata una eccezione “stringa malformata” in quanto in essa è stato inserito un carattere non valido.

Quando i token sono stati esplorati tutti, nell’ultima fase vengono svolte le operazioni matematiche conclusive con il medesimo algoritmo precedente\*.

Il metodo ritorna l’ultimo e unico int rimasto sullo stack “operandi”, il quale sarà il risultato dell’intera espressione.

* **primoMaggSec(Character a, Character b) :** il metodo ha una funzione molto importante all’interno dell’intero algoritmo, infatti grazie ad esso viene valutata la priorità degli operatori. Esso ritorna “true” se il primo operatore (a) è ha maggiore priorità matematica del secondo(b), “false” altrimenti.

ValutaExprGUI

La classe è stata creata per implementare una semplice GUI di interazione. All’Interno di essa è presente un’altra classe FinestraGUI nel cui costruttore è definita appunto la finestra grafica di interazione.

* **FinestraGUI() :** Viene definito il titolo della finestra, i suoi bordi, la dimensione, la visibilità, l’operazione di uscita di default se si clicca sulla ‘x’ di uscita. Viene aggiunto il JPanel principale sulla finestra, all’interno del quale verranno aggiunti tutti i componenti grafici. Esso è impostato con un BorderLayout. Nella parte superiore del JPanel principale viene posizionato in centro il titolo grafico dell’applicazione. Successivamente viene creato un secondo JPanel che conterrà il JTextField per immettere la stringa da sottoporre a valutazione e un JLabel alla sua destra di descrizione. Esso viene addizionato al pannello principale nella sua parte sinistra. L’applicazione per funzionare necessità di un JButton dove poter cliccare per dare inizio alla valutazione dell’espressione, e di una JLabel al di sotto di esso, per mostrare il risultato. Al JButton viene associato un ActionListener attraverso una LambdaExpression: in esso, all’interno di una struttura try..catch, viene invocata la valutazione dell’espressione sulla stringa presente all’interno del JTextField. Il risultato del metodo risolvi() viene impostato come testo della JLabel. Se in questa fase viene generata un’eccezione, essa viene controllata dal catch, il quale mostra un messaggio di stringa malformata.

Nel main viene inizializzata la FinestraGUI().

Andrea Dell’Osso, mat. n°209865.