## Progetto Valutazione Espressione

Il progetto Valutazione Espressione si compone di due package: *poo.util*, munito di 3 classi, necessarie per l'utilizzo della struttura dati Stack Concatenato e *poo.valutazioneEspressione*, composto dalle classi: *"ValutazioneEspressioneGUI.java"* , che implementa l'interfaccia grafica, e dalla classe *"EspressioneUtil.java"* , che rappresenta una classe di utilità contentente i metodi adibiti al controllo della validità dell'espressione e della sua valutazione.

***Premessa***

La valutazione dell'espressione viene realizzata rispettando le usuali precedenze della matematica, le operazioni supportate ed ordinate secondo priorità decrescente sono:

^ > *, /, % > +,- .

A parità di priorità si assume l'associatività a sinistra: per alterare le priorità è possibile l'utilizzo delle parentesi.

A tal proposito, si munisce la classe *EspressioneUtil*, di un oggetto comparatore: si definisce una innerClass denominata "*Gerarchia"*, finalizzata a delineare le priorità tra gli operatori.

*public static class Gerarchia implements Comparator<Character>*

La classe appena citata implementa l'interfaccia *Comparator*, prevede quindi il metodo *compare* che riceve come parametri due operatori: "*cor"*, ovvero l'operatore appena estratto dall'espressione e "*opCima"*, ovvero l'operatore in cima allo stack. Il metodo restituisce 0 in caso di uguale priorità; restituisce -1, se l'operatore *cor* possiede minore priorità di *opCima*; e restituisce 1 se l'operatore *cor* possiede maggiore priorità di *opCima*.

***Algoritmo di valutazione***

Per la memorizzazione dell'espressione si utilizzano due stack: uno di *operandi* ed uno di *operatori*.

*public static int valuta(String espressione)*

Il metodo *valuta* riceve in input una stringa *"espressione"* già sottoposta alla verifica di condizione sufficiente (con il metodo *espressioneValida* che verrà trattato in seguito). Si utilizza sulla stringa uno String Tokenizer *st* che usa come delimitatori (inclusi), gli operatori e le parentesi.

Il metodo restituirà il risultato finale dell'espressione che verrà visualizzato nella console apposita.

*public static int costruzioneEspressione(String tokenizer st)*

Il metodo "*valuta"* memorizza il risultato all'interno della variabile *"ris"*, invocando il metodo "*costruzioneEspressione"* , all'interno del quale vengono dichiarati i due stack prima citati.

Il metodo prevede due cicli più esterni:

**Il primo ciclo** itera fin quando il tokenizer passato contiene elementi.

Si preleva un operando (attraverso il metodo "*gestioneOperando"* trattato in seguito), successivamente, si preleva l'operatore corrente dallo stack di operatori, se quest'ultimo è vuoto oppure l'operatore corrente ha maggiore priorità di quello in cima allo stack di operatori, l'operatore corrente viene aggiunto allo stack.

Altrimenti, se lo stack degli operatori non è vuoto ed allo stesso tempo l'operatore corrente non ha maggiore priorità di quello in cima, si entra in un ciclo annidato, all'interno del quale si preleva l'operatore in cima e si invoca il metodo gestioneOperatore.

*public static Integer gestioneOperatore(Stack<Integer> operandi, Stack<Character> operatori, char opCima)*

Il metodo *gestioneOperatore* riceve come parametri lo stack di operandi, lo stack di operatori, e l'operatore in cima allo stack appena prelevato.

Il metodo *gestioneOperatore* preleva gli ultimi due operandi ed applica, supportato dal metodo eseguiOperazione, l'operatore estratto, ed inserisce il risultato dell'operazione appena effettuata nello stack di operandi.

public static int eseguiOperazione(int o1, int o2, char op)

Il metodo *eseguiOperazione* sfrutta uno switch al fine di applicare la corretta operazione associata all'operatore e restituisce il risultato dell'operazione.

Una volta terminato il metodo *gestioneOperatore*, il ciclo annidato viene eseguito fin quando lo stack di operatori non è vuoto ed allo stesso tempo l'operatore corrente non ha maggiore priorità di quello in cima.

Quando finisce il primo ciclo e non vi sono più elementi nell'espressione (o nella sottoespressione se stiamo analizzando una parentesi tonda) si entra nel **secondo ciclo**, all'interno del quale si applicano a ripetizione gli operatori presenti nello stack sfruttando nuovamente il metodo *gestioneOperatore*.

Se non si verificano errori dovute ad uno scorretto inserimento delle parentesi, il metodo restituisce il risultato dell'espressione, ed in questo caso si avrà il risultato finale, oppure della sottoespressione, che sarà operando di un' espressione più grande.

***Gestione delle parentesi tonde***

All'interno del metodo "*costruzioneEspressione"* si memorizza l'operando, di tipo Integer, attraverso il metodo di supporto *"gestioneOperando".*

*public static Integer gestioneOperando(StringTokenizer st)*

il metodo gestioneOperando, se trova un numero intero, restituisce un valore Integer; altrimenti, se trova una parentesi tonda, invoca ricorsivamente il metodo *costruzioneEspressione*.

Se nel corso del metodo *costruzioneEspressione*, viene trovata una parentesi chiusa, il ciclo termina, si ha la fine della sottoespressione (o dell'espressione) e si torna all'area dati precedente.

***Malformazioni***

*public static boolean espressioneValida(String espressione)*

Attraverso il metodo *"espressioneValida"*, che restituisce un booleano, si pone una condizione necessaria ma non sufficiente, affinchè si possano identificare da subito alcune evidenti malformazioni quali la presenza di caratteri diversi da numeri interi o dagli operatori.

La corretta chiusura o apertura della parentesi ed altri casi specifici si valuteranno nel corso della costruzione dell'espressione.

*Corretta apertura e chiusura parentesi*

Si definisce una variabile static *"apertura" inizializzata a false.* All'interno del metodo *costruzioneEspressione* si definiscono due variabili booleane, *parentesiChiuse* (inizializzata a false) e *parentesiAperte* (che sfrutta la variabile statica): se si verifica un'apertura nell'area dati precedente *parentesiAperte* viene inizializzata true, altrimenti false.

Se per ogni sottoespressione avremo (parentesiAperte == parentesiChiuse) l'espressione è ben formata.

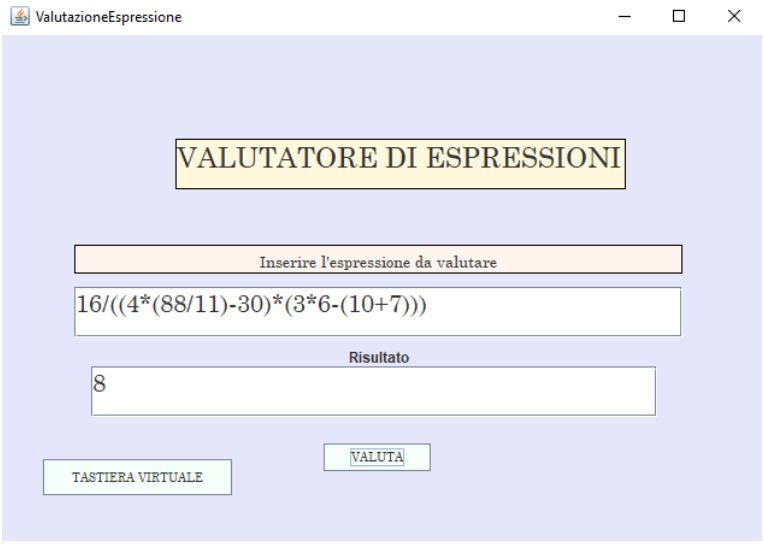
**GUI di interazione**

Il progetto è dotato di un' interfaccia GUI implementata all'interno della classe *"ValutazioneEspressioneGUI.java".*

La classe si compone di un frame principale *"FrameGUI"* che rimanda, se selezionato, al frame "*TastieraVirtuale"*:

*class FrameGUI extends JFrame implements ActionListener*

Il frame principale è dotato di area inserimento e di console(provviste entrambe di scroll-Panel), e varie Jlabel indicative.




*private class TastieraVirtuale extends JFrame implements ActionListener*

È possibile inoltre sfruttare una tastiera virutale che consente l'inserimento dell'espressione da valutare, unicamente attraverso il click del mouse