Giorgio Ubbriaco – matricola:209899 – 2° anno Ingegneria Informatica formativo

Progetto - Valutazione di una espressione aritmetica intera con le priorità degli operatori della matematica

Il progetto è composto da sei file:

* Stack
* StackAstratto
* StackConcatenato
* ApplicationEspressione
* GetRisultato
* RiconoscitoreEspressione

I primi tre file sono collocati nel package poo.stack sviluppato durante le lezioni dal professore Nigro.

Gli ultimi tre file, invece, si trovano nel package denominato progetto.espressione.

La classe ApplicationEspressione è composta da un main al cui interno viene creato un nuovo JFrame tramite la inner class CalcolaEspressione la quale estende per l’appunto JFrame. Quest’ultima classe appena citata è composta da un JLabel contenente una stringa che spiega all’utente di inserire un’espressione aritmetica. Tanto è vero che nella stessa finestra troviamo subito sotto (al JLabel) un JTextField dove l’utente potrà inserire l’espressione e successivamente cliccando “OK” verrà calcolato il valore e mostrato in un’altra finestra (la classe ValutaEspressione che estende JFrame). Ovviamente, l’espressione inserita dovrà essere un’espressione valida, il quale controllo verrà effettuato tramite la classe “RiconoscitoreEspressione”. Se il controllo non dovesse andare a buon fine, verrebbe visualizzata dall’utente una finestra-errore (la classe “ExprMalformata”). Inoltre, se l’utente decidesse di annullare l’operazione, verrebbe visualizzata una finestra-“uscita” (la classe “Uscita” che estende JFrame) che chiederà all’utente se è veramente intenzionato ad uscire dall’applicazione.

La classe RiconoscitoreEspressione permette di verificare se un’espressione è valida tramite una regex denominata “regex” ottenuta dalla composizione di altre regex. Viene verificato inizialmente se è presente un operando (regex “op”) che in caso affermativo farebbe pensare subito ad un operatore successivo ad esso (condizione non necessaria, poiché l’espressione potrebbe essere composta anche soltanto da un solo operando che corrisponderebbe al risultato dell’espressione stessa). Oltretutto, viene verificata la presenza delle parentesi: sia all’inizio che successivamente agli operatori, viene verificata la presenza di una parentesi tonda aperta, mentre subito dopo gli operandi e alla fine la presenza di una parentesi tonda chiusa. La regex finale sarà la concatenazione delle regex più piccole appena citate. Pertanto, l’utente potrà inserire una qualsiasi tipo di espressione aritmetica con tutti i possibili operatori e con quante parentesi tonde meglio crede purché vengano rispettati i vincoli imposti dalla regex.

La classe GetRisultato permette di calcolare il risultato dell’espressione aritmetica inserita dall’utente nel JTextField che poi verrà visualizzato in un’apposita finestra (la classe ValutaEspressione). Essa è composta da 5 metodi di cui due strettamente collegati per ricorsione, mentre gli altri 3 sono metodi di “appoggio” per un corretto funzionamento dei primi due. Nel metodo pubblico valutaEspressione() viene dichiarato uno StringTokenizer il quale verrà passato come argomento nel metodo richiamato subito dopo (valutaEspressione(StringTokenizer st)). Il metodo privato valutaEspressione(StringTokenizer st) è il metodo principale che effettua, grazie all’aiuto di due Stack (dichiarati all’inizio insieme ad altre due stringhe utili nell’algoritmo), il calcolo dell’espressione aritmetica proposta dall’utente. I due Stack (operandi e operatori) saranno fondamentali nel metodo appena citato poiché gestiranno gradualmente le stringhe ottenute tramite il metodo “nextToken()” dello StringTokenizer. Tanto è vero che la stringa i-esima, prima di essere inserita nell’apposito Stack, viene valutata se essere un operatore o un operando. Il tutto facilitato da un metodo privato “isOperatore” che verifica tramite una piccola regex se siamo di fronte ad un operatore o meno. Se il metodo ritorna true la stringa i-esima viene inserita all’interno dello Stack operatori, altrimenti inserita nello Stack operandi. Per quanto riguarda invece la gestione delle parentesi, la loro “possibile” presenza viene verificata inizialmente (cioè prima della verifica “isOperatore”). Ovviamente, ogni volta che si aggiunge un operatore allo Stack operatori bisognerà controllare se tale Stack sia vuoto o meno. Nel caso in cui esso sia vuoto allora si può direttamente aggiungere l’operatore nello Stack, altrimenti bisogna verificare la priorità di tale operatore rispetto a quella dell’operatore in cima allo Stack operatori (ottenuto tramite il metodo peek()). Se l’operatore corrente considerato risulta essere più prioritario dell’operatore in cima allo Stack (ciò significa che l’operatore corrente è un elevamento a potenza e l’operatore in cima allo Stack è un operatore qualsiasi oppure l’operatore corrente è una “moltiplicazione” o una “divisione” o un “resto” e l’operatore in cima allo Stack è una “addizione” o una “sottrazione”) allora il metodo priorita(String opCor, String operatore) ritornerà true e di conseguenza l’operatore corrente considerato sarà aggiunto in cima allo Stack operatori, altrimenti nei casi rimanenti il metodo ritornerà false e verrà calcolato il valore dell’espressione aritmetica avente come operatore l’operatore in cima allo Stack operatori e come operandi e operatori i rispettivi Stack omonomi; infine in cima allo Stack operatori verrà aggiunto l’operatore corrente. Il metodo calcola(String operatore, Stack<Integer> operandi, Stack<Character> operatori), pertanto, riceve come parametri un operatore e i due Stack. Inizialmente verranno memorizzati in due variabili intere, denominate “operando2” e “operando1”, i due operandi in cima allo Stack operandi tramite il metodo pop() che li andrà oltretutto a rimuovere. Successivamente verrà rimosso e memorizzato in una variabile String l’operatore in cima allo Stack operatori (che prima era stato soltanto memorizzato in una variabile stringa e non rimosso tramite il metodo peek()) tramite il metodo pop(). Quindi, viene fatto lo switch dell’operatore considerato così da poter capire che tipo di operazione dovrà essere svolta. Dopo che aver trovato la giusta corrispondenza dell’operatore, verrà aggiunto in cima allo Stack operandi il risultato dell’operazione corrispondente ottenuta dai due operandi precedentemente rimossi e aggiunti alle due variabili intere all’inizio del metodo. Infine, si effettua un break per far terminare il ciclo e far continuare a calcolare l’espressione aritmetica rimanente. Nel caso in cui lo Stack operandi dovesse avere una dimensione diversa da 1, farebbe sollevare un’eccezione unchecked segnalando all’utente una situazione di espressione malformata (esempio: 2+3(4\*6) , dove praticamente manca all’appello un operatore prima della parentesi tonda). Nel caso in cui la stringa i-esima corrisponde ad una parentesi tonda aperta allora si inserisce all’interno dello Stack operandi il risultato dell’espressione interna alle parentesi richiamando ricorsivamente il metodo valutaEspressione(StringTokenizer st). Nel caso in cui la stringa i-esima corrisponde ad una parentesi tonda chiusa allora si interrompe il ciclo e di conseguenza aggiungendo il risultato ottenuto delle operazioni tra le due parentesi, nel caso in cui ci sia stata una richiamata ricorsiva. Infine, dopo che l’algoritmo finisce di calcolare qualsiasi tipo di operazione all’interno dell’espressione aritmetica proposta dall’utente rimuoverà l’operando in cima allo Stack operandi e lo ritornerà come risultato dell’espressione aritmetica.