Progetto 1: Valutatore di numeri interi

La classe si basa principalmente su tre metodi statici. Ho deciso di renderli statici perché non vi è la necessità di variabili d'istanza dato che l'unico dato esterno su cui operano i metodi è una stringa, o meglio, un tokenizzatore acceso su una stringa, che può essere passato come parametro da metodo a metodo. L'unico metodo pubblico è valutaOperando(StringTokenizer st). Nel javadoc sono tutti pubblici solo per mostrarli nel javadoc stesso, ma sul file gli altri due metodi sono privati.
I metodi principali sono i seguenti tre:

valutaOperando

```
public static java.lang.Integer valutaOperando(java.util.StringTokenizer st)
```

Parameters:
st - Lo StringTokenizer che scandisce la stringa
Returns:
Il risultato dell'espressione

valutaOperandoPar

```
public static java.lang.Integer valutaOperandoPar(java.util.StringTokenizer st)
```

Parameters:
st - Lo StringTokenizer che scandisce la stringa
Returns:
Il risultato dell'espressione (contenuta dentro una coppia di parentesi)

valutaEspressione

```
public static java.lang.Integer valutaEspressione(java.lang.Integer opn1,
                                                  java.lang.Integer opn2,
                                                  char op)
```

Parameters:
opn1 - é il primo operatore
opn2 - é il secondo operatore
op - é l'operatore in questione
Returns:
Calcola l'espressione matematica attraverso un'operazione aritmetica comandata dall'operatore. Torna il risultato a valutaOperando.

Il metodo valutaOperando(StringTokenizer st) opera sull'espressione per intero. Partendo dai suggerimenti dati nella traccia ho istanziato i due "stack" sotto forma di LinkedList, uno per gli operandi (LinkedList di Integer) e uno per gli operatori (LinkedList di Character). Il char opc, operatore corrente, servirà lungo tutto il metodo, quindi lo dichiaro già all'inizio. Il metodo scandisce ogni singolo carattere dell'espressione attraverso lo StringTokenizer in un ciclo comandato dalla condizione che esso abbia ancora dei token da prendere.

```
//Scandisco una sola volta la string, prendendo un carattere a ogni "passo"
while(st.hasMoreTokens() == true){
  String nt = st.nextToken();
```

Ora, con in mano il token, si presentano quattro situazioni in cui esso é:

1. Un operatore
2. Una parentesi aperta
3. Una parentesi chiusa
4. Un operando.

Nel **primo** caso confronto l'operatore corrente (opc) con la "testa" dello stack operatori, nel caso non sia vuoto, caso in cui inserisco a prescindere l'operatore nello stack.

Nel caso in cui l'espressione inizi con un segno, ovvero caso in cui lo stack operandi é vuoto, inserisco uno zero in quest'ultimo. Il confronto avviene con il metodo compare(Integer opn1, Integer opn2) tramite un oggetto, istanziato precedentemente nel metodo, istanza della classe Precedenza che implementa l'interfaccia Comparator<Character>, per cui vi è priorità negli operatori(^ > *,/,% > +,-).

```
//Caso in cui quello attuale é piú priotario del precedente, quindi devo inseri
if(pr.compare(opc,operatori.getFirst()) > 0) operatori.addFirst(opc);
//Caso in cui posso processare la attuale operazione
else{
  Integer opn2 = operandi.removeFirst(); Integer opn1 = operandi.removeFirst();
  char opr = operatori.removeFirst();
  //Processo l'espressione
  operandi.addFirst(valutaEspressione(opn1,opn2,opr));
  //Inserisco il risultato in cima allo stack
  operatori.addFirst(opc);
}
```

Figura 1

Nell'if, caso in cui va processato prima l'operatore corrente di quello in cima allo stack, inserisco opc in cima allo stack con addFirst(opc), mentre nell'else mi prendo i primi due operandi dallo stack (vanno presi al contrario vista la gestione lifo dello stack) e il primo operatore, dandoli in pasto a valutaEspressione(Integer opn1,Integer opn2,char opc) e inserendo il risultato in cima allo stack operandi per usarlo successivamente.

Nel **secondo** caso, visto che le espressioni dentro parentesi hanno priorità assoluta, affido la scannerizzazione a partire dal carattere dopo la parentesi aperta al metodo valutaOperandoPar(StringTokenizer st). Il tokenizzatore in questo metodo é sempre lo stesso. Tutta l'espressione, infatti, viene scandita una sola volta da un solo tokenizzatore. Il metodo scannerizza tutti i caratteri fino alla parentesi chiusa e passa la stringa risultante, sotto forma di StringTokenizer, a valutaOperando(StringTokenizer st), che ne calcolerà il risultato.
Il risultato verrà poi messa in cima allo stack.

```
String espr = "";
String nt = "";
//Costruisco una stringa che contiene tutta l'espressione contenuta nelle parentesi
while(!nt.equals(")") && st.hasMoreTokens()){
  nt = st.nextToken();
  espr += nt;
}
```

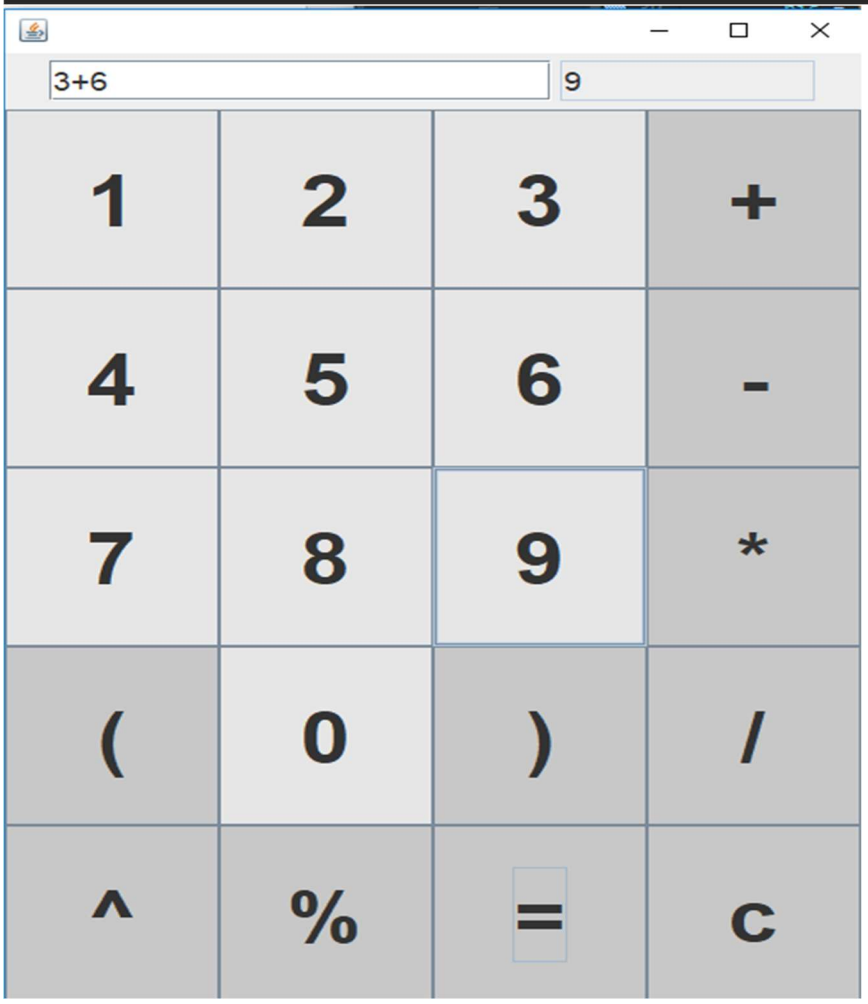
Nel **terzo** caso non dovrà avvenire nulla perché il controllo di chiusura delle parentesi viene fatto già in valutaOperandoPar(StringTokenizer st).

Nel **quarto** caso inserisco l'intero, in seguito a un valueOf sul token, in testa allo stack operandi.

Vengono, infine, processate tutte le operazioni finché lo stack operatori non è vuoto, prendendo sempre due operandi dalla cima e un operatore richiamando ogni volta valutaEspressione, così come in figura 1 e viene restituita la testa dello stack operandi, che sarà il risultato di tutta l'espressione.

Per quanto riguarda l'interfaccia grafica, essa si presenta come una calcolatrice, con i bottoni in gridLayout. L'ascoltatore del bottone = costituisce in pratica il main della classe Valutatore.
In esso, infatti, istanzio lo StringTokenizer sull'espressione con i relativi separatori e inserisco valutaOperando(StringTokenizer st) in un try-catch per catturare tutte le eccezioni e lanciare nel caso fosse necessario l'eccezione "Espressione Malformata". La NumberFormatException, ad esempio, può sollevarsi durante il valueOf(Integer a) nel quarto ramo if di valutaOperando(StringTokenizer st), mentre la seconda eccezione può sollevarsi quando ad esempio c'è un segno di troppo o un operando che manca e, di conseguenza, lo stack operandi risulterebbe vuoto prematuramente, per cui si solleva la NoSuchElementException.

```
// Ascoltatore per il bottone = . Una volta premuto torna il risultato.
class ugListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        Integer risultato = null;
        String espressione = espr.getText();
        StringTokenizer st = new StringTokenizer(espressione,"*^/-+%()",true);
        try{
            risultato = Valutatore.valutaOperando(st);
          }catch(NumberFormatException | NoSuchElementException ex){
            ris.setText("Espr. Malformata");
          }
        if(risultato != null) ris.setText(risultato.toString());
    }
}
```



Flavio Maiorana – Matricola 182611