Uitrekenen van postfix-expressies

In deze opgave kijken we naar eenvoudige expressies, waar alleen de volgende elementen in voorkomen:
– positieve gehele getallen
– de operatoren +, -, * en /, waarbij / de gehele (integer) deling is.

In de "normale" notatie van expressies staan de operatoren tussen de operanden (de getallen). We noemen dat *infix-notatie*. In deze notatie zijn er soms haakjes nodig om duidelijk te maken hoe de expressie gelezen moet worden, bijvoorbeeld (15 + 5) * 2 is iets anders dan 15 + 5 * 2. Om een expressie goed te lezen is bovendien kennis nodig van prioriteiten (vermenigvuldigen gaat voor optellen).

Er zijn ook notaties waarin het gebruik van haakjes en kennis van prioriteiten niet nodig is. In deze opdracht kijken we naar één zo'n notatie, en wel de *postfixnotatie*. In die notatie staat de operator altijd achter de twee operanden (argumenten): in plaats van 5 + 7, schrijven we 5 7 +, in plaats van (15 - 5) * 2 schrijven we 15 5 - 2 *, en in plaats van 15 - 5 * 2 schrijven we 15 5 2 * -. De volgorde waarin de getallen en de operatoren staan, bepaalt de 'uitrekenvolgorde', dus kennis over prioriteiten en haakjes zijn niet nodig.

De volgende tabel toont nog meer voorbeelden van deze notatie.

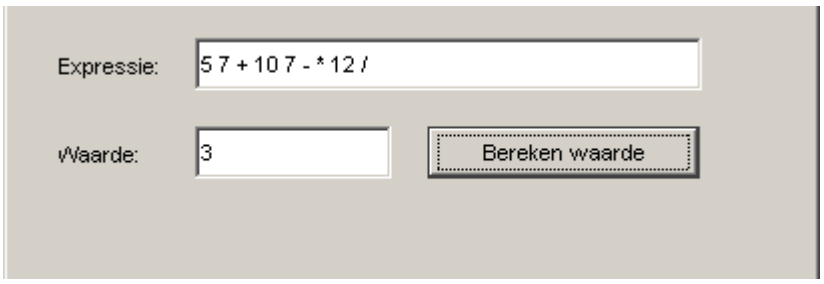
Expressie	*Postfixexpressie*	*Waarde*
12	12	12
12 - 5	12 5 -	7
(12 + 8) * 4	12 8 + 4 *	80
12 + 8 * 4	12 8 4 * +	44
12 + 8 * 4 / 2	12 8 4 * 2 / +	28
8 * 2 + 8 * 4 / 2	8 2 * 8 4 * 2 / +	32
((5 + 7) * (10 - 7)) / 12	5 7 + 10 7 - * 12 /	3
10 - 4 -2	10 4 - 2 -	4
10 - (4 - 2)	10 4 2 - -	8
100 / 10 / 2	100 10 / 2 /	5
100 / (10 / 2)	100 10 2 / /	20

Merk op, dat we 10 - 4 -2 moeten lezen als (10 - 4) - 2, en niet als 10 - (4-2). In postfixnotatie wordt 10 - 4 -2 dus 10 4 - 2 -. Voor / geldt iets dergelijks.

De waarde van een postfixexpressie kan gemakkelijk worden uitgerekend met behulp van een stack.

OPGAVE
Ga na hoe je een stack kunt gebruiken om de waarde van een expressie in postfixnotatie uit te rekenen. Gebruik de voorbeelden uit de tabel om dit uit te zoeken.

In deze opdracht ontwerpt en implementeert u een applicatie waarmee postfix expressies gelezen en uitgerekend worden. De volledige applicatie is getoond in figuur 1.



FIGUUR 1 Applicatie die postfixexpressies inleest en uitrekent

In het expressieveld wordt een expressie ingevoerd in postfixnotatie, waarbij getallen gescheiden moeten worden door spaties. Verder is het gebruik van spaties vrij (je mag ook rond een operator spaties zetten, maar het hoeft niet). Bij klikken op de knop Bereken waarde moet de waarde van de expressie getoond worden in het veld Waarde. Eventuele foutmeldingen verschijnen in een foutLabel onderin het frame.

Bouwstenen

Bij deze opdracht gaat het voornamelijk om het uitrekenen van de waarde van de expressie. De volgende bouwstenen zijn beschikbaar:
– Een onvolledige klasse ExpressieFrame met de gebruikersinterface. De event-handler voor de berekenKnop moet nog worden ingevuld.

– Een enumeratie Operation, die vier verschillende constanten bevat die de binaire rekenkundige operaties +, -, * en / representeren. Op een operator kan de apply(int x, int y) methode worden aangeroepen, die het resultaat berekent wanneer de operatie wordt toegepast met x en y als operanden. De methode toString en de statische methode fromString converteren een enumeratie constante naar zijn string representatie, en vice versa (!). Zie bijlage 1 voor verdere details van de interface.

– Gebruik de klasse java.util.StreamTokenizer om de gegeven expressie in String formaat op te splitsen in getallen en operatoren. Om te voorkomen dat deze klasse "-" en "/" als scheidingstekens beschouwd, kunt u de methode ordinaryChar gebruiken. Roep (herhaaldelijk) de nextToken aan om de expressie één voor één in getallen en operatoren te splitsen en uit te lezen. Het attribuut *ttype* geeft aan of het huidige element een getal bevat of een operator. Indien het een getal betreft staat de waarde van het getal in het attribuut *nval*:

```
StreamTokenizer tokenlezer =
                    new StreamTokenizer(new StringReader(expressie));
if(tokenlezer.ttype == StreamTokenizer.TT_NUMBER)
      int val = (int) tokenlezer.nval;
```

Aanwijzingen:
– U mag gebruik maken van de klasse java.util.Stack. U mag ook een eigen stack gebruiken, maar dan moet u ook echt een klasse Stack schrijven met de vereiste interface (push, pop, top, isEmpty en size).
– Schrijf eerst een versie die legale postfixexpressie kan uitrekenen, en test deze met behulp van de expressies uit de tabel. Voeg vervolgens foutafhandeling toe. Denk na wat er allemaal fout kan zijn!

Inleveren

Alle Java-klassen en projectbestanden die tot uw uitwerking behoren.

BIJLAGE 1: De enumeratieklasse Operation

```
public enum Operation
extends java.lang.Enum<Operation>
```

Representeert een binaire rekenkundige operatie.

Enum Constant Summary

Enum Constants

Enum Constant and Description
DELING Constante voor operator /
MAAL Constante voor operator *
MIN Constante voor operator -
PLUS Constante voor operator +

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Abstract Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`abstract int`	**apply**(int x, int y) Past deze binaire rekenkundige operatie toe op de meegegeven (integer) waarden.
`static Operation`	**fromString**(java.lang.String opStr) Converteert een string representatie van een operator naar zijn corresponderende enum constante.
`java.lang.String`	**toString**() Returnt het wiskundige symbool voor deze operatie als een string.
`static Operation`	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.
`static Operation[]`	**values**() Returns an array containing the constants of this enum type, in the order they are declared.