laboratorium: zadanie nr 3 *23 października 2006 r.*

KURS PROGRAMOWANIA W JAVIE

OBLICZANIE WYRAŻEŃ ONP

Instytut Informatyki Uniwersytetu Wrocawskiego *Paweł Rzechonek*

Napisz program, który będzie obliczał i wypisywał wartość zadanej funkcji z jedną zmienną x dla kolejnych argumentów. Definicja funkcji ma być przekazana do programu w postaci wyrażenia ONP poprzez argumenty wywołania. W definicji tej funkcji oprócz liczb rzeczywistych, funcji standardowych, operatorów dodawania, odejmowania, mnożenia, dzielenia i potęgowania może także wystąpić zmienna x.

Program działający w pętli najpierw prosi użytkownika o podanie argumentu x (strumień `System.err`) a następnie oblicza i wypisuje (strumień `System.out`)wartość zadanej funkcji we wskazanym puncie. Czynność ta ma być powtarzana dotąd, dopóki użytkownik nie zamknie strumienia wejściowego `System.in`.

Rozwiązanie tego zadania będzie się składało z trzech części.

Zaprojektowanie i implementacja hierarchii klas:

Najpierw zaprojektuj hierarchię klas, która umożliwi łatwe zapamiętanie wyrażenia ONP a potem jego obliczenie. Wyrażenie ONP to ciąg symboli (abstrakcyjna klasa `Symbol`). Symbolami tymi mogą być albo operandy (klasa `Operand`) albo funkcje (klasa `Funkcja`). Operandy to liczby (klasa `Liczba`), stałe (klasa `Stala` i jej podklasy `E`, `Pi`, itp) lub zmienna x (klasa `X` z polem statycznym przechowującym aktualną wartość zmiennej). Funkcje to przede wszystkim dwuargumentowe operatory dodawania, odejmowania, mnożenia, dzielenia i potęgowania; pozostałe funkcje, które należy zaimplementować w postaci podklas to `Abs`, `Floor`, `Ceil`, `Frac`, `Min`, `Max`, `Sin`, `Cos`, `Atan`, `Acot`, `Ln`, `Ex`, itp.

Jaką funkcjonalność powinny mieć te klasy? Na pewno każda z nich powinna implementować interfejs `Wartosc`:

```
public interface Wartosc
{
    double obliczWartosc () throws IllegalStateException;
}
```

Metoda `obliczWartosc()` w odniesieniu do operandów powinna przekazywać pamiętane w nich wartości, a w odniesieniu do funkcji wyliczać wartość na podstawie przekazanych wcześniej argumentów.

Funkcje powinny więc posiadać mechanizm umożliwiający przekazywanie im argumentów w postaci interfejsu `Argumenty`:

```
public interface Argumenty extends Wartosc
{
    int arnosc ();
    int brakujaceArgumenty ();
    void dodajArgument (double) throws IllegalStateException;
}
```

Metoda `arnosc()` mówi o arności funkcji czy operatora. Metoda `brakujaceArgumenty()` informuje o liczbie brakujących argumentów, czyli argumentów które trzeba jeszcze dostarczyć do funkcji za pomocą `dodajArgument()`, zamin wywoła się metodę `obliczWartosc()`. Oto przykład wykorzystania tego interfejsu do obliczenia wartości funkcji:

```
while (fun.brakujaceArgumenty()>0) do
    fun.dodajArgument(...);
double wynik = fun.obliczWartosc();
```

Gdy liczba dostarczonych argumentów jest niezgodna z arnością funkcji to wywołanie metody `obliczWartosc()` powinno skutkować zgłoszeniem wyjątku `IllegalStateException`. W wyrażeniach ONP wygodnie jest dostarczać argumenty od końca.

I wreszcie klonowanie. Wszystkie elementy wyrażenia powinny implementować interfejs `Cloneable` umożliwiający poprawne klonowanie tych obiektów. Właściwość ta przyda się potem do wykonywania obliczeń.

Implementacja struktur danych:

Do pamiętania wyrażenia ONP i do wykonywania obliczeń będą nam potrzebne dwie proste struktury danych: kolejka i stos. Obie powinny dziedziczyć po klasie bazowej `Lista`, implementującej listę jednokierunkową przechowującą w węzłach obiekty implementujące interfejs `Wartosc`. Klasa `Lista` powinna być tylko opakowaniem dla homogenicznej dynamicznej struktury danych opartej na węzłach zagnieżdżonej klasy `Wezel`. Klasa `Lista` powinna być wyposażona w inteligentną metodę klonującą.

```
public class Lista implements Cloneable
{
    protected class Wezel implements Cloneable
    {
        protected Wartosc wart;
        protected Wezel nast;
        // ...
    }

    protected Wezel poczatek;

    public Object Clone () {/*...*/}

    // ...
}
```

W klasach `Lista` i `Wezel` zaimplementuj metody do wstawiania, usuwania i wyszukiwania zadanej wartości (użyj interfejsu `Cloneable`).

Implementacja logiki obliczeń ONP:

> Na początku program tworzy kolejkę z wyrażeniem ONP przekazanym do niego za pomocą argumentów wywołania. Następnie w pętli prosi użytkownika o podanie parametru x i wylicza wartość tego wyrażenia dla konkretnego argumentu (tak jak to zostało opisane na początku).
>
> Jak ma wyglądać proces samych obliczeń? Otóż program najpierw tworzy kopię wyrażenia i posyła ją do motoru obliczeniowego, który usuwa z niej kolejne elementy i za pomocą stosu dokonuje obliczeń. Na stos mają trafiać same wartości liczbowe. W czasie obliczeń należy wykorzystać mechanizm dostarczania argumentów (pobieranych ze stosu) do funkcji i operatorów w odwrotnej kolejności.

Programując to zadanie zdefiniuj w pakiecie `narzedzia` klasy interfejsy `Wartosc` i `Argumenty` oraz klasy `Lista`, `Kolejka` i `Stos`.

Uwaga!

Program powinien na początku sprawdzić poprawność definicji funkcji: zerujemy licznik i czytamy wyrażenie ONP od początku do końca; dodajemy 1 do licznika po napotkaniu liczby lub zmiennej i odejmujemy 1 po napotkaniu operatora; w trakcie tego procesu licznik nigdy nie powinien się wyzerować, a na końcu osiągnąć wartość 1.