Zadanie 5 *14–15 listopada 2018 r.*

KURS JĘZYKA JAVA

KALKULATOR ONP

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

Zadanie 1.

Zaprojektuj hierarchię klas, która umożliwi łatwe zapamiętywanie a potem obliczanie wyrażeń zapisanych w *Odwrotnej Notacji Polskiej*. 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` z wartością typu `double`) albo zmienne (klasa `Zmienna` z nazwą zmiennej — identyfikatorem pasującym do wzorca `"\\p{Alpha}\\p{Alnum}*"`). Funkcje to przede wszystkim dwuargumentowe operatory dodawania, odejmowania, mnożenia i dzielenia; należy też zaimplementować funkcje dwuargumentowe (`Min`, `Max`, `Log`, `Pow`), jednoargumentowe (`Abs`, `Sgn`, `Floor`, `Ceil`, `Frac`, `Sin`, `Cos`, `Atan`, `Acot`, `Ln`, `Exp`) oraz funkcje bezargumentowe pełniące rolę stałych (`E`, `Pi`, `Phi`) Żadna zmienna nie powinna mieć takiej samej nazwy jak funkcja.

Jaką funkcjonalność powinny mieć te klasy? Zarówno operandy (liczby i zmienne) jak i funkcje (bezargumentowe, jednoargumentowe i dwuargumentowe) powinny implementować interfejs `Obliczalny`:

```
public interface Obliczalny
{
    double oblicz() throws WyjątekONP;
}
```

Metoda `oblicz()` w odniesieniu do liczb i zmiennych powinna przekazywać pamiętane w operandach wartości a w odniesieniu do funkcji wyliczać wartość na podstawie przekazanych wcześniej argumentów. Funkcje powinny więc posiadać mechanizm umożliwiający przekazywanie im argumentów przed wykonaniem obliczenia. Można go zapisać w postaci interfejsu `Funkcyjny`:

```
public interface Funkcyjny extends Obliczalny
{
    int arność ();
    int brakującеArgumenty ();
    void dodajArgument (double) throws WyjątekONP;
}
```

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

```
while (fun.brakującеArgumenty() > 0) do
    fun.dodajArgument(...);
double wynik = fun.oblicz();
```

Gdy liczba dostarczonych argumentów jest niezgodna z arnością funkcji to wywołanie metody `oblicz()` powinno skutkować zgłoszeniem jakiegoś wyjątku `WyjątekONP`.

Pozostaje jeszcze pytanie: skąd i jak należy brać argumenty dla funkcji? Argumenty te będa nam potrzebne w trakcie obliczania wartości wyrażenia. Można więc zdefiniować klasę `Wyrażenie`, która będzie zawierała wyrażenie ONP w postaci kolejki symboli i stos z wynikami pośrednimi. To właśnie z tego stosu należy pobierać argumenty dla funkcji i operatorów. Należy jeszcze tak zaprojektować klasy związane z wyrażeniem, aby umożliwić dostęp do stosu symbolom z kolejki.

```
class Wyrażenie
{
    private Kolejka kolejka; // kolejka symboli wyrażenia ONP (elementy typu Symbol)
    private Stos stos; // stos z wynikami pośrednimi obliczeń (elementy typu Double)

    private Zbiór zmienne; // lista zmiennych czyli pary klucz-wartość (String-Double)

    public Wyrażenie (String onp, Lista zm) throws WyjątekONP {/*...*/}

    // ...
}
```

Klasa `Wyrażenie` powinna też mieć referencję do zbioru asocjacyjnego ze zmiennymi (będą one potrzebne w trakcie obliczania wartości wyrażenia). Referencję tą możesz przekazać do obiektu klasy `Wyrażenie` w konstruktorze.

Na koniec wyjątki. Zaprojektuj hierarchię klas wyjątków kontrolowanych przez kompilator, dziedziczących po wspólnej klasie `WyjątekONP`. Tylko te wyjątki powinny być używane w klasach reprezentujących wyrażenie ONP.

```
class WyjątekONP extends Exception {/*...*/}

class ONP_DzieleniePrzez0 extends WyjątekONP {/*...*/}
class ONP_NieznanySymbol extends WyjątekONP {/*...*/}
class ONP_BłędneWyrażenie extends WyjątekONP {/*...*/}
class ONP_PustyStos extends WyjątekONP {/*...*/}
// ...
```

Hierarchia twoich wyjątków powinna być co najmniej dwupoziomowa i składać się co najmniej pięciu klas.

Do zapamiętania wyrażenia ONP i do obliczenia jego wartości będą nam potrzebne trzy proste struktury danych: *lista*, *kolejka* i *stos* oraz *zbiór asocjacyjny*. Zaimplementuj je opakowując odpowiednie kolekcje ze standardowych pakietów Javy. Klasa `Lista` niech opakuje `LinkedList<String>`, klasa `Kolejka` niech wykorzysta kolekcję `ArrayDeque<Token>` a `Stos` niech wykorzysta kolekcję `ArrayDeque<Double>`. Kolekcja zmiennych `Zbiór` może się posłużyć klasą `TreeMap<String, Double>`.

Definicje wszytkich klas, interfejsów i wyjątków umieść w pakiecie `narzędzia` i dopisz do nich komentarze dokumentacyjne. Udokumentuj także cały pakiet `narzedzia` umieszczając komentarz dokumentacyjny w pliku `package-info.java`.

Zadanie 2.

Finalną częścią tego projektu będzie program *interaktywnego kalkulatora ONP*. Kalkulator ma interpretować i obliczać wyrażenia zapisane w postaci ONP. Program powinien odczytywać polecenia ze standardowego wejścia (każde polecenie w osobnym wierszu), wykonywać obliczenia i wypisywać wyniki na standardowe wyjście. Wszelkie komentarze i informacje o błędach program ma wysyłać na standardowe wyjście dla błędów.

Program powinien rozpoznawać tylko trzy rodzaje poleceń:

- `calc` *wyrażenieONP* (*zm* =)⋆
  Obliczenie wartości wyrażenia *wyrażenieONP* i wypisanie jej na standardowym wyjściu. Wyrażenie będzie zapisane w postaci postfiksowej (*Odwrotna Notacja Polska*). Czytając kolejne tokeny wyrażenia program powinien je zamieniać na obliczalne symbole i umieszczać w kolejce. Przy obliczaniu wartości wyrażenia należy się posłużyć stosem.
  W wersji rozszerzonej o nazwę zmiennej i znak przypisania, należy dodatkowo utworzyć nową zmienną *zm* i przypisać jej warości obliczonego wyrażenia *wyrażenieONP*. Jeśli zmienna *zm* była zdefiniowana już wcześniej, to należy tylko zmodyfikować zapisaną w niej wartość. Takich przypisań można zwobić kilka w jednym wyrażeniu. Przykłady:
  ```
  calc 7.5 r =
  calc r
  calc pi r 2 power *
  calc 2.5 x = y = z =
  ```

- `clear` (*zm*)⋆
  Usunięcie wskazanych zminnych z kolekcji asocjacyjnej.
  Jeśli w tym poleceniu nie występują żadne znazwy zmiennych, to wówczas należy usunąć wszystkie używane do tej pory zmienne z kolekcji. Przykłady:
  ```
  clear r
  clear
  ```

- `exit`
  Wyjście z programu. Alternatywą dla tego polecenia powinno być zamknięcie strumienia wejściowego. Przykład:
  ```
  exit
  ```

Jeśli w wyrażeniu ONP (polecenie `calc`) zostanie wykryty i zgłoszony błąd (źle sformułowane wyrażenie, błędna nazwa, błędny literał stałopozycyjny, czy nierozpoznany operator, funkcja lub zmienna) to należy wypisać stosowny komunikat o błędzie, ale nie przerywać działania programu.

Do swojego programu wstaw asercję, która zgłosi wyjątek `AssertionError` gdy użytkownik wpisze nieznaną komendę (inną niż `calc`, `clear`, `exit`).

Dodatkowo dopisz do programu logowanie każdego obliczanego wyrażenia (poprawnego i niepoprawnego) w dzienniku o nazwie `calc.log`.

Uwaga.

Program należy skompilować i uruchomić w zintegrowanym środowisku programistycznym *NetBeans* albo *IntelliJ*! Wygeneruj też na podstawie komentarzy dokumentacyjnych dokumentację do zawartości całego pakietu `narzędzia`.