laboratorium: zadanie 3 *termin: 29–30 października 2013 r.*

# KURS PROGRAMOWANIA W JAVIE

DRZEWA OBLICZEŃ

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

**Zadanie 1.**

Zdefiniuj klasę `Para`, która będzie przechowywać pary klucz–wartość, gdzie klucz jest identyfikatorem typu `String` a skojarzona z nim wartość to liczba całkowita typu `int`. Klucz powinien być polem publicznym ale niemodyfikowalnym, a wartość polem ukrytym, które można odczytać za pomocą gettera i zmodyfikować tylko za pomocą settera.

```
public class Para {
    public final String klucz;
    private int wartość;
    // ...
}
```

W klasie tej zdefiniuj metody `toString()` oraz `equals(Object)` — dwie pary są równe gdy mają takie same klucze.

Następnie zdefiniuj klasę `Zbior`, która będzie realizować operacje wyszukania wartości związanej z zadanym kluczem, wstawienia nowej pary do zbioru, zliczenia wszystkich elementów w zbiorze i usunięcia wszystkich par.

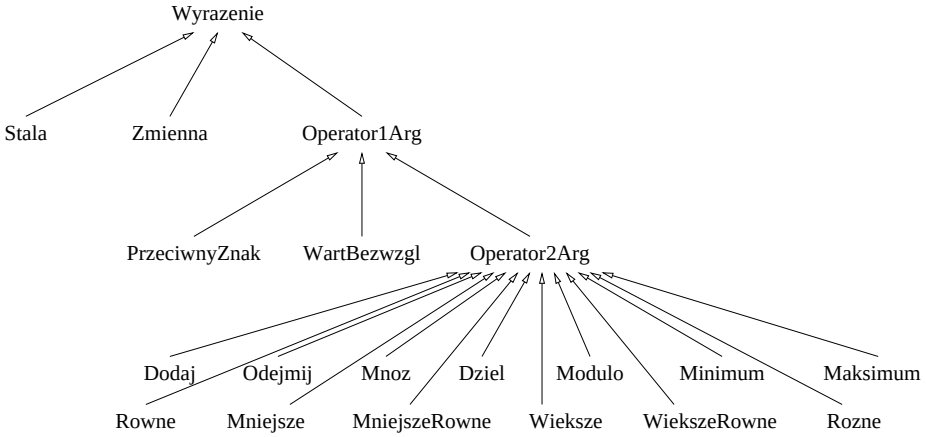
```
public class Zbior {
    // ...
    public void wstaw (String kl, int wart) throws IllegalArgumentException {
        /* ... */ }
    public int szukaj (String kl) throws IllegalArgumentException {
        /* ... */ }
    public int ile () {
        /* ... */ }
    public void czyść () {
        /* ... */ }
}
```

W klasie `Zbiór` przechowuj elementy typu `Para` w tablicy par (nie korzystaj z kolekcji standardowych). W klasie tej zdefiniuj metody `toString()` oraz `equals(Object)` — dwa zbiory są równe gdy zawieraja takie same zbiory kluczy.

**Zadanie 2.**

Zdefiniuj abstrakcyjną klasę bazową `Wyrazenie`, reprezentującą wyrażenie arytmetyczne. W klasie tej umieść deklarację abstrakcyjnej metody `oblicz()`, której zadaniem w klasach potomnych będzie obliczanie wyrażenia i przekazywanie wyniku jako wartości typu `int`.

Następnie zdefiniuj klasy dziedziczące po klasie `Wyrazenie`, które będą reprezentowały kolejno liczbę (stała całkowitoliczbowa), zmienną (wszystkie zmienne pamiętaj w polu statycznym typu `Zbior` w zdefiniowanym wcześniej zbiorze par klucz–wartość), operacje arytmetyczne (dodawanie, odejmowanie, mnożenie, dzielenie i modulo oraz jednoargumentowa operacja zmiany znaku na przeciwny), porównania (wynikiem porównania ma być jak w języku C liczba 0 albo 1 odpowiadająca wartościom logicznym *false* albo *true*), itp. Klasy te powinny być tak zaprojektowane, aby można z nich było zbudować drzewo wyrażenia: obiekty klas `Liczba` lub `Zmienna` to liście, a operatory to węzły wewnętrzne w takim drzewie. W klasach potomnych zdefiniuj metody `oblicz()` oraz `toString()`. W klasach tych podefiniuj metody `toString()` oraz `equals(Object)`.




Na koniec napisz krótki program testowy, sprawdzający działanie obiektów tych klas. W swoim programie skonstruuj drzewa obliczeń, wypisz je metodą `toString()` a potem oblicz i wypisz wartość dla następujących wyrażeń:

```
3+5
2+x*7
(3*11-1)/(7+5)
((x+13)*x)/2
17*x+19<0
```

Na przykład wyrażenie `7+x*5` należy zdefiniować następująco:

```
Wyrazenie w = new Wyrazenie(
    new Dodaj(
        new Liczba(7),
        new Mnoz(
            new Zmienna("x"),
            new Liczba(5)
        )
    )
);
```

Ustaw na początku programu testowego zmienną `x` na wartość `-3`.

**Zadanie 3.**

Zdefiniuj abstrakcyjną klasę bazową `Instrukcja`, reprezentującą instrukcję w programie. W klasie tej umieść deklarację abstrakcyjnej metody `wykonaj()`, której zadaniem w klasach potomnych będzie wykonywanie odpowiednich obliczeń.

Następnie zdefiniuj klasy dziedziczące po klasie `Instrukcja`, które będą reprezentowały kolejno deklarację zmiennej (zmienne zapamiętuj na stosie par klucz–wartość), instrukcję przypisania wartości wyrażenia do zmiennej, instrukcję warunkową (taką jak instrukcja *if/if-else* w języku C), instrukcję pętli (taką jak instrukcja *while/do-while* w języku C), przy czym warunki są wyrażeniami (warunek jest prawdziwy tylko wtedy gdy wartość wyrażenia jest różna od 0), instrukcje czytania ze standardowego wejścia i pisania na standardowe wyjście oraz instrukcję blokową (złożenie dwóch lub więcej instrukcji). Instrukcja blokowa powinna na zakończenie likwidować wszystkie zmienne, które zostały stworzone podczas jej wykonania. Klasy te powinny być tak zaprojektowane, aby można z nich było zbudować drzewo sterujące sekwencją obliczeń. W klasach potomnych zdefiniuj metody `wykonaj()` a także metody `toString()` oraz `equals(Object)`.

Na koniec napisz krótki program testowy, sprawdzający działanie obiektów tych klas. W swoim programie skonstruuj drzewo obliczeń dla programu obliczającego silnię:

```
var silnia;
silnia = 1;
var n;
read n;
if (n>1)
{
    var i;
```

```
        i = 2;
        while (i<=n)
        {
            silnia = silnia*i;
            i = i+1;
        }
    }
    write silnia;
```

Wydrukuj swój program metodą `toString()` a potem wykonaj go.

**Uwaga 1.**
Klasę `Zbior` zaimplementuj w jakiś prosty sposób, na przykład używając tablicy `Para[]` o ograniczonej wielkości.

**Uwaga 2.**
W swoich programach nie czytaj ani nie analizuj danych ze standardowego wejścia. Drzewa wyrażeń i drzewo obliczeń zdefiniuj na stałe w swoich programach testowych.

**Uwaga 3.**
Definicje klas `Wyrazenie` i `Instrukcja` oraz ich klas pochodnych umieść w pakiecie `narzedzia.obliczenia`.

**Uwaga 4.**
Program należy skompilować i uruchomić z wiersza poleceń!