Zadanie 4 *10, 12 listopada 2020 r.*

KURS JĘZYKA JAVA

DRZEWA WYRAŻEŃ I SEKWENCJE OBLICZEŃ

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

Zadanie 1.

W pakiecie `obliczenia` zdefiniuj interfejs `Obliczalny`, reprezentujący obiekty, na których można coś policzyć metodą `oblicz()`. Zadamiem tej metody ma być w klasach implementujących ten interfejs wykonanie obliczeń i zwrócenie wyniku jako wartości typu `int`.

W pakiecie tym zdefiniuj również publiczny interfejs `Wykonywalny`, reprezentujący obiekty, które umieją wykonać ciąg obliczeń metodą `wykonaj()`. Metoda ta nie powinna zwracać żadnej wartości.

Zadanie 2a.

W pakiecie `obliczenia` zdefiniuj abstrakcyjną klasę `Wyrazenie`, reprezentującą całkowitoliczbowe wyrażenie arytmetyczne. Klasa ta ma implementować interfejs `Obliczalny` (nie definiuj metody `oblicz()` w tej klasie, gdyż jeszcze nie wiadomo co należy policzyć) — będzie to klasa bazowa dla innych klas realizujących konkretne obliczenia określone w wyrażeniu. W klasie `Wyrazenie` umieść dwie statyczne metody ze zmienną liczbą argumentów, które będa realizowały zadanie sumowania i mnożenia wyrażeń:

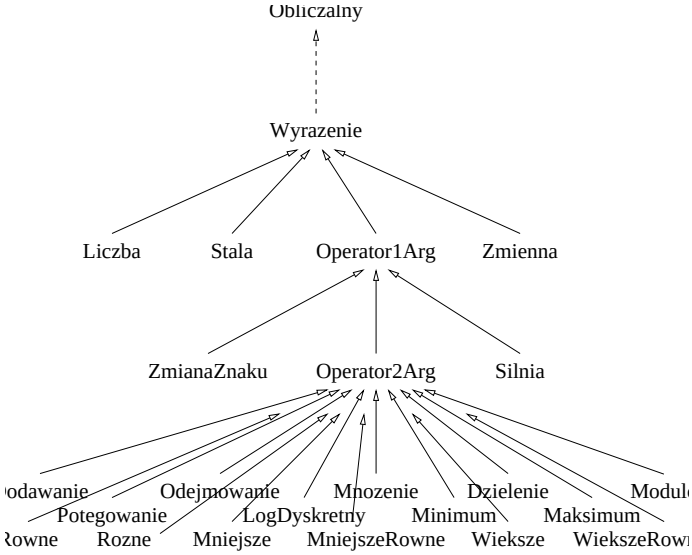
```
abstract class Wyrazenie {
    // ...
    /** metoda sumująca wyrażenia */
    public static double suma (Wyrazenie... wyr) {
        /* ... */ }
    /** metoda mnożąca wyrażenia */
    public static double iloczyn (Wyrazenie... wyr) {
        /* ... */ }
}
```

Następnie zdefiniuj klasy dziedziczące po klasie `Wyrazenie`, które będą reprezentowały kolejno: liczbę, stałą, zmienną, operacje arytmetyczne (dodawanie, odejmowanie, mnożenie, dzielenie i reszta z dzielenia oraz jednoargumentową operację zmiany znaku na przeciwny), operacje porównywania (równe, różne, mniejsze, mniejsze–równe, większe, większe–równe) dające w wyniku wartość 0 (fałsz) albo 1 (prawda) i popularne funkcje matematyczne (silnia, minimum, maksimum, potęgowanie, logarytm dyskretny, itp). Klasy te powinny być tak zaprojektowane, aby można z nich było zbudować drzewo wyrażenia: obiekty klas `Liczba`, `Stała` i `Zmienna` to

liście, a operatory i funkcje to węzły wewnętrzne w takim drzewie. We wszystkich klasach tych podefiniuj metody `toString()` oraz `equals(Object)`.

W klasie `Zmienna` zdefiniuj statyczne pole finalne do pamiętania zbioru wszystlich zmiennych w programie (pary identyfikator–liczba). Do przechowywania zmiennych możesz wykorzystać klasę `Zbior` z poprzedniego zadania. Odczytywanie wartości zmiennej ma polegać na zidentyfikowaniu pary w tym zbiorze i odczytaniu wartości związanej z identyfikatorem.

Klasa `Stala` ma reprezentować takie stałe wartości jak stała Archimedesa π, stała Eulera e czy stała złotego podziału ϕ, które są często używane w wyrażeniach arytmetycznych.



Zadanie 2b.

Uzupełnij swoje zadanie o krótki program testowy napisany poza pakietem `obliczenia`. Program ma rzetelnie sprawdzić działanie obiektów reprezentujących wyrażenie arytmetyczne.

W programie testowym skonstruuj drzewa obliczeń, wypisz je metodą `toString()` a potem oblicz i wypisz otrzymane wartości. Przetestuj swój program dla następujących wyrażeń:

```
3 + 5
-(2 - x) * 7
(3 * 11 - 1) / (7 + 5)
min((x + 13) * x, (1 - x) mod 2)
2 ^ 5 + x * log(2, y) < 20
```

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

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

Ustaw na początku programu testowego zmienną `x` na wartość `2`.

Zadanie 3a.

W pakiecie `obliczenia` zdefiniuj abstrakcyjną klasę `Instrukcja`, reprezentującą wykonanie jakiejś instrukcji w programie. Klasa ta ma implementować interfejs `Wykonywalny` (nie definiuj metody `wykonaj()` w tej klasie, gdyż jeszcze nie wiadomo co należy wykonać) — będzie to klasa bazowa dla innych klas realizujących konkretne instrukcje określone w programie.

Następnie zdefiniuj klasy dziedziczące po klasie `Instrukcja`, które będą reprezentowały kolejno: instrukcję blokową, deklarację zmiennej (zmienne inicjalizuj wartością 0), instrukcję przypisania wartości obliczonego wyrażenia do zmiennej), instrukcje warunkowe (takie jak instrukcje `if` oraz `if-else`), instrukcje pętli (takie jak instrukcje `while` oraz `do-while`), instrukcję czytania ze standardowego wejścia (odczytujemy tylko liczby całkowite) oraz instrukcję pisania na standardowe wyjście (piszemy tylko liczby całkowite). Warunek w instrukcjach warunkowych lub w pętlach jest wyrażeniem (warunek jest prawdziwy wtedy i tylko wtedy gdy wartością wyrażenia jest liczba różna od 0). Instrukcja blokowa powinna być inicjalizowana dowolną liczbą instrukcji wewnętrznych (konstruktor ze zmienną liczbą argumentów); poza tym instrukcja ta ma spamiętywać wszystkie zmienne, które zostały utworzone w tym bloku i na końcu ma je usunąć (nie wolno tworzyć zmiennych o takich samych nazwach). Konstruktory klas reprezentujących różne instrukcje powinny sprawdzać, czy ich argumenty są równe `null`, a jeśli tak, to należy zgłosić odpowiedni wyjątek.

Zadanie 3b.

Uzupełnij swoje zadanie o krótki program testowy napisany poza pakietem `obliczenia`. Program ma rzetelnie sprawdzić działanie obiektów reprezentujących instrukcje w programie.

W programie testowym sprawdź, czy wczytana liczba jest pierwsza. Program ten może działać według następującego schematu:

```
var n;
read n;
if (n < 2) write 0;
else
{
   var p;
   p = 2;
   var wyn;
   while (p * p <= n)
   {
      if (n mod p == 0)
      {
         wyn = p;
         p = n;
      }
      p = p + 1;
   }
   if (wyn > 0) write 0;
   else write 1;
}
```

Uwaga.

Program należy skompilować i uruchomić w zintegrowanym środowisku programistycznym *IntelliJ* (może być *NetBeans* albo *Eclipse*)!