Zadanie 4 4, 9 listopada 2021 r.

kurs języka Java

drzewa wyrażeń

Instytut Informatyki
Uniwersytetu Wrocławskiego Paweł Rzechonek

Część 1.

W pakiecie `struktury` definiuj 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 ma być publicznym polem niemodyfikowalnym; klucz to niepusty napis, który powinien składać się tylko z małych liter alfabetu angielskiego (nie może to być wartość `null`). Wartość ma być polem prywatnym, które można odczytać za pomocą gettera i zmodyfikować tylko za pomocą settera.

```
public class Para implements Cloneable {

    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. Klasa `Para` ma także implementować interfejs `Cloneable` (zdefiniuj publiczną metodę `clone()`) oraz `Comparable<Para>` (zdefiniuj metodę porównującą `compareTo(Para p)`).

Część 2.

Dalej w pakiecie `struktury` zdefiniuj interfejs `Zbior`, który będzie zawierać niezbędne narzędzia do pracy na zbiorze obiektów typu `Para` (w zbiorze nie mogą wystąpić dwie pary o takim samym kluczu).

```
public interface Zbior {

    // ...

}
```

W interfejsie tym powinny się znaleźć przynajmniej następujące metody:

- `Para szukaj (String k)` – metoda ma wyszukać parę z zadanym kluczem; metoda zwraca `null`, gdy nie znajdzie pary o podanym kluczu;

- `void wstaw (Para p)` – metoda ma wstawić do zbioru nową parę; gdy para o podanym kluczu już jest w zbiorze, metoda dokonuje aktualizacji wartości w znalezionej parze;
- `void usuń(String k)` – metoda ma usunąć ze zbioru parę o zadanym kluczu; gdy pary o podanym kluczu nie ma w zbiorze metoda nic nie robi;
- `void czysc()` – metoda ma usunąć wszystkie pary ze zbioru; po tej operacji zbiór staje się pusty;
- `int ile()` – metoda ma podać ile jest wszystkich par w zbiorze.

Część 3.

Na koniec w pakiecie `struktury` zdefiniuj klasę `ZbiorListowy` implementującą interfejs `Zbior`. Sam zbiór ma być reprezentowany przez listę jednokierunkową złożoną z węzłów. Klasa węzła `Wezel` niech będzie prywatną klasą wewnętrzną w `ZbiorListowy`.

```
public class ZbiorListowy implements Zbior, Cloneable {
    private class Wezel {
        private Para para;
        private Wezel nast;
        // ...
    }
    private Para lista;
    // ...
}
```

Klasa `ZbiorListowy` ma także implementować interfejs `Cloneable` (zdefiniuj publiczną metodę `clone()`).

Część 4.

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

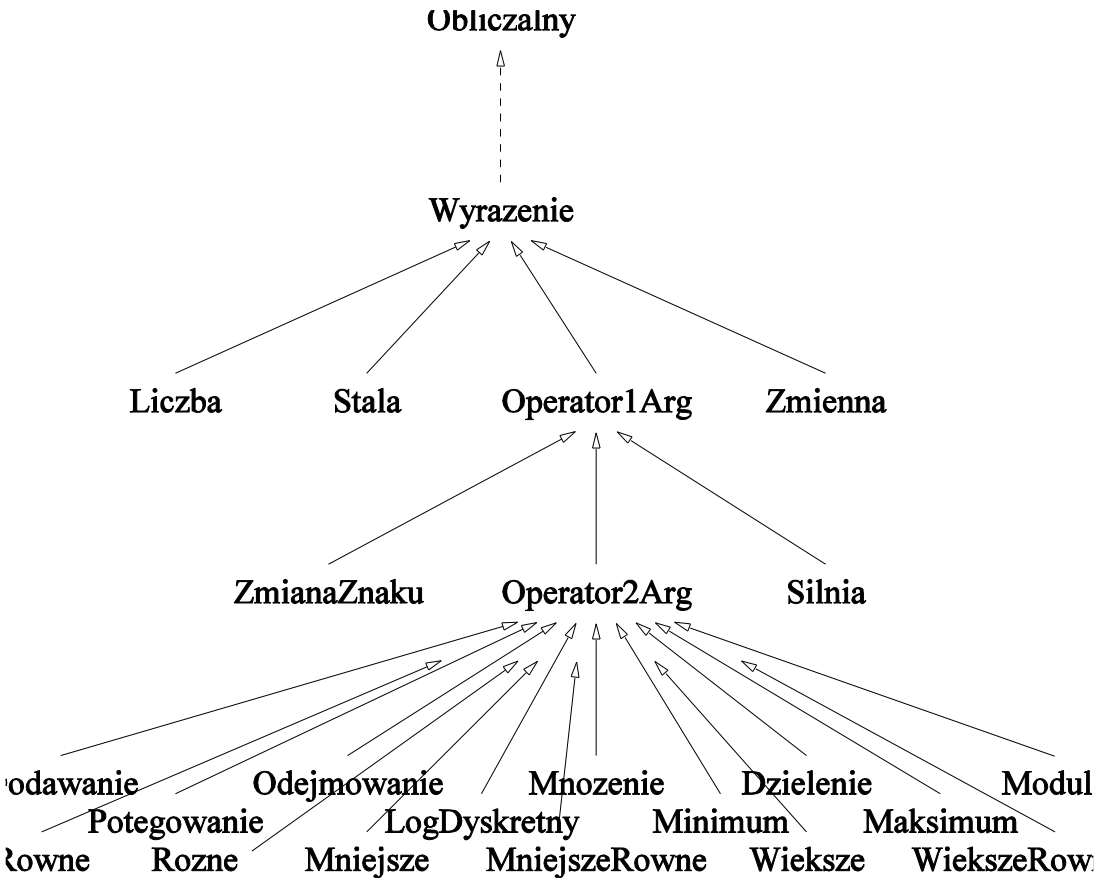
Część 5.

Dalej w pakiecie `obliczenia` zdefiniuj publiczną 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 będą realizowały zadanie sumowania i mnożenia wyrażeń:

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

Część 6.

Wreszcie 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 operandy czyli liście w drzewie wyrażenia, natomiast operatory i funkcje to węzły wewnętrzne w takim drzewie. We wszystkich klasach zdefiniuj metody `toString()` oraz `equals(Object)`.



W klasie Zmienna zdefiniuj statyczne pole finalne do pamiętania zbioru wszystkich zmiennych w programie (pary identyfikator–liczba). Do przechowywania zmiennych możesz wykorzystaj zmodyfikowany wcześniej `ZbiorListowy`. 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 zero, jeden oraz minus jeden, które są często używane w wyrażeniach arytmetycznych. Klasa `Liczba` ma przechowywać wartość typu `int`.

Część 7.

Uzupełnij swoje zadanie o program testowy napisany w pakiecie `testy`. Program ma rzetelnie sprawdzić działanie klonowania zbiorów oraz 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) * y
(3 * 11 - 1) / (y + 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 a zmienną y na 7.