

# kurs języka Java

## wyrażenia arytmetyczne

Instytut Informatyki  
Uniwersytetu Wrocławskiego

Paweł Rzechonek

---

Zaprogramuj i przetestuj hierarchię klas, dzięki której będzie można zbudować drzewo wyrażenia arytmetycznego (liście to operandy a węzły wewnętrzne to operatory lub funkcje matematyczne). Drzewo wyrażenia można do używać obliczania wartości tego wyrażenia dla przygotowanych wcześniej zmiennych występujących w tym wyrażeniu.

### 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 rzeczywista typu `double`. Klucz ma być publicznym polem niemodyfikowalnym (klucz to niepusty napis, który powinien składać się tylko z małych liter alfabetu angielskiego; nie może też przyjmować wartości `null`). Wartość ma być polem prywatnym, które można odczytać za pomocą gettera i zmodyfikować za pomocą settera.

```
public class Para implements Cloneable {
    public final String klucz;
    private double wartość;
    // ...
}
```

W klasie tej nadpisz dwie metody: `toString()` do zaprezentowania pary w postaci napisu oraz `equals(Object)` do porównywania par (dwie pary są równe, gdy mają takie same klucze). Klasa `Para` ma także implementować interfejs `Cloneable` (zdefiniuj publiczną metodę `clone()`) do klonowania par oraz `Comparable<Para>` do porównywania par (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`. Pracując ze zbiorem par należy zachować niezmiennik, że 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 zadaniem 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 zadaniem kluczu; gdy pary o podanym kluczu nie ma w zbiorze metoda nic nie robi;
- `void` `czyszc()` – metoda ma usunąć wszystkie pary ze zbioru; po tej operacji zbiór staje się pusty;
- `boolean` `pusty()` – metoda ma sprawdzić czy zbiór par jest pusty;
- `int` `ile()` – metoda ma podać ile jest wszystkich par w zbiorze.

Część 3.

Na koniec w pakiecie struktury zdefiniuj klasę `ZbiorTablicowy` implementującą interfejs `Zbior`. Zbiór w tej klasie ma być zaimplementowany na zwykłej tablicy. Rozmiar tej tablicy powinien być określony w konstruktorze. Tablica ma się zapełniać elementami od początku, końcowe puste komórki będą do wykorzystania na nowo wstawiane elementy; próba wstawienia elementu do całkowicie zapełnionej tablicy ma skutkować zgłoszeniem wyjątku `IllegalStateException`.

```
public class ZbiorTablicowy implements Zbior, Cloneable {
    private Para[] zbiór;
    private int zapełnienie;
    // ...
}
```

Zaimplementowany zbiór ma być klonowalny, a więc powinien 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 w klasach implementujących ten interfejs ma być wykonanie obliczeń na liczbach rzeczywistych i zwrócenie wyniku jako wartości typu `double`.

Część 5.

Dalej w pakiecie `obliczenia` zdefiniuj publiczną abstrakcyjną klasę `Wyrazenie`, reprezentującą wyrażenie arytmetyczne pracujące na operandach (liczby, stałe i zmienne), operatorach i funkcjach rzeczywistych. Klasa ta ma implementować interfejs `Obliczalny` (nie definiuj metody `oblicz()` w tej klasie, ponieważ jeszcze nie wiadomo co i jak należy policzyć) – będzie to klasa bazowa dla innych klas realizujących konkretne obliczenia.

```

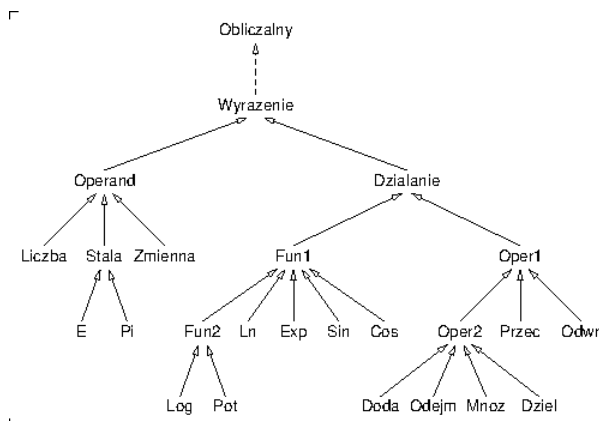
abstract class Wyrażenie {
    // ...
    /** metoda sumująca wyrażenia */
    public static int suma (Wyrażenie... wyr) { /* ... */ }
    /** metoda mnożąca wyrażenia */
    public static int iloczyn (Wyrażenie... wyr) { /* ... */ }
}

```

W klasie Wyrażenie umieść dwie statyczne metody ze zmienną liczbą argumentów, które będą realizowały zadanie sumowania i mnożenia wyrażeń.

### Część 6.

Na koniec w pakiecie obliczenia zdefiniuj klasy dziedziczące po klasie Wyrażenie, które będą reprezentowały operandy i działania arytmetyczne. Operandy to: liczba, stała i zmienna. Operatory arytmetyczne to: dodawanie, odejmowanie, mnożenie, dzielenie oraz jednoargumentowe operatory zmiany znaku na przeciwny ( $x \mapsto -x$ ) oraz odwrotności ( $x \mapsto 1/x$ ); popularne funkcje matematyczne to: sinus, cosinus, potęgowanie, logarytm itp. Klasy operandów i operatorów 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 napisz metody toString() oraz equals(Object).



Klasa Liczba ma przechowywać zwykłą wartość typu double. Klasa Stała ma reprezentować popularne stałe takie jak Pi ( $\pi \approx 3.13$ ) czy E ( $e \approx 2.72$ ), które są często używane w wyrażeniach arytmetycznych. W klasie Zmienna zdefiniuj statyczne pole finalne do pamiętania zbioru wszystkich zmiennych stworzonych w programie (pary identyfikator–liczba). Do przechowywania zmiennych powinieneś wykorzystać własną kolekcję ZbiorTablicowy. Odczytywanie wartości zmiennej ma polegać na zidentyfikowaniu pary w tym zbiorze i odczytaniu wartości związanej z identyfikatorem.

Operatory jednoargumentowe to operatory prefiksowe. W operatorach dwuargumentowych zaprojektuj system priorytetów, aby zminimalizować liczbę potrzebnych nawiasów przy wypisywaniu wyrażenia.

### Część 7.

Uzupełnij swoje zadanie o program testowy. 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:

$7 + 5 * 3 - 1$

$\sim (2 - x) * e$  # symbol  $\sim$  oznacza tutaj negację wyrażenia

$(3 * \pi - 1) / (! x + 5)$  # symbol  $!$  oznacza tutaj odwrotność wyrażenia

$\sin((x + 13) * \pi / (1 - x))$

$\exp(5) + x * \log(e, x)$

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

```
Wyrażenie w = new Dodaj(  
    new Liczba(7.2),  
    new Mnoz(  
        new Zmienna("x"),  
        new Liczba(2.4)  
    )  
);
```

Ustaw na początku programu testowego zmienną x na wartość 1.618.

Uwaga.

Wszystkie klasy z pakietu struktury spakuj do pliku struktury.jar a klasy z pakietu obliczenia spakuj do pliku obliczenia.jar. Obydwa pliki jar wykorzystaj w projekcie testowym.