

KURS PROGRAMOWANIA W JAVIE

DRZEWA BINARNYCH POSZUKIWAŃ

Instytut Informatyki Uniwersytetu Wrocławskiego

Paweł Rzechonek

Zadanie 1.

W pakiecie **narzedzia** zdefiniuj publiczną klasę **Para**, która będzie przechowywać pary *klucz–wartość*, gdzie klucz jest typu **String** a wartość typu **Double**. Klucz powinien być polem publicznym ale niemodyfikowalnym a wartość polem ukrytym, które można odczytać za pomocą gettera i zmodyfikować tylko za pomocą settera. Zdefiniowana przez Ciebie klasa ma implementować interfejs **Comparable<Para>** (porównanie względem kluczy).

Zadanie 2.

W pakiecie **narzedzia** zdefiniuj publiczny interfejs **Słownik**, który będzie reprezentować zbiór obiektów typu **Para** z podstawowymi operacjami słownikowymi: wyszukiwanie (metoda **szukaj()**), wstawianie (metoda **wstaw()**), usuwanie (metoda **usuń()**) i policzenie wszystkich elementów w zbiorze (metoda **ile()**).

Zadanie 3.

W pakiecie **narzedzia** zdefiniuj publiczną klasę **Drzewo**, która będzie *drzewem binarnych poszukiwań* przechowującym w węzłach wartości typu **Para**. Klasa ta ma implementować interfejsy **Słownik** i **Cloneable**.

Klasa **Drzewo** ma być klasą opakowującą homogeniczną strukturę drzewa BST zbudowaną z węzłów. Pojedynczy węzeł natomiast, czyli klasa **Węzeł**, powinien być klasą wewnętrzną w klasie **Drzewo**. W klasie wewnętrznej **Węzeł** zaimplementuj rekurencyjne metody do wstawiania par klucz–wartość oraz wyszukiwania i usuwania par na podstawie zadanego klucza.

Zadanie 4.

Napisz program testujący, który będzie manipulował początkowo pustymi drzewami. Program ma działać interaktywnie na konsoli: użytkownik wpisuje polecenie z parametrami, a program polecenie to interpretuje i wykonuje. Do poleceń tych muszą należeć operacje słownikowe (wstawianie, usuwanie i wyszukiwanie) na wskazanym drzewie, klonowanie drzew, wyświetlenie ich całej zawartości oraz wyjście z programu. Nazwy poszczególnych drzew (i jednocześnie ich ilość) mają być przekazane do programu poprzez parametry wywołania. Oto przykładowe wywołanie tego programu i praca użytkownika:

```
user@computer:~/myprograms> java ManipulacjaDrzewami a b c
komenda: insert a x 11
komenda: insert a y 19
komenda: insert a z 17
komenda: insert a u 13
komenda: print a
      {(u 13), (x 11), (y 19), (z 17)}
komenda: bla bla bla
      ??? nie znane polecenie !!!
komenda: clone a c
komenda: delete c x
komenda: print c
      {(u 13), (y 19), (z 17)}
komenda: insert b m 13
komenda: insert b k 29
komenda: insert b l 23
komenda: insert b n 7
```

```
komenda: delete b p
komenda: delete b k
komenda: search b n
    yes
komenda: search b q
    no
komenda: print b
    {(1 23), (m 13), (n 7)}
komenda: exit
```

Twój program powinien sprawdzać poprawność wpisywanych przez użytkownika komend i ich parametrów.

Wskazówka.

Do odczytania danych ze standardowego wejścia wykorzystaj klasę `BufferedReader` z pakietu `java.io`:

```
BufferedReader we = new BufferedReader(new InputStreamReader(System.in));
```

Wywołując na obiekcie `we` metodę `readLine()` odczytasz cały wiersz z danymi wpisanymi z klawiatury. Odczytany wiersz można podzielić na fragmenty według zadanego separatora (ciąg białych znaków) metodą `split()`:

```
String[] tab = we.readLine().trim().split("\\s+");
```

Pamiętaj też, że łańcuchy znakowe porównujemy metodą `equals()` lub `compareTo()`.