laboratorium: zadanie 4 *termin: 6, 8 listopada 2012 r.*

KURS PROGRAMOWANIA W JAVIE

DRZEWA BINARNYCH POSZUKIWAŃ

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

Zadanie 1.

W pakiecie `narzedzia` zdefiniuj publiczną klasę `Para`, która będzie reprezentować niemodyfikowalną parę *klucz–wartość*, gdzie klucz będzie typu `String` a wartość typu `Double`. Klucz i wartość mają być polami publicznymi i finalnymi, inicjalizowanymi w konstruktorze.

Zdefiniowana przez Ciebie klasa ma implementować interfejs `Comparable<Para>` (porównanie względem kluczy). Implementując metodę `compareTo()` sprawdź, czy typy obu obiektów są identyczne.

Zadanie 2.

W pakiecie `narzedzia` zdefiniuj publiczny interfejs `Słownik`, który będzie opisywać funkcjonalnie 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 `DrzewoBST`, która będzie *drzewem binarnych poszukiwań* przechowującym w węzłach wartości typu `Para`. Klasa ta ma implementować interfejsy `Słownik` i `Cloneable`.

Klasa `DrzewoBST` ma być klasą opakowująca dla homogenicznej struktury drzewa BST zbudowanej z węzłów. Pojedynczy węzeł, zdefiniowany jako klasa `Węzeł`, powinien być niepubliczną klasą wewnętrzną w `DrzewoBST`. W klasie `Węzeł` zaimplementuj rekurencyjne metody do wstawiania par klucz–wartość oraz wyszukiwania i usuwania par na podstawie zadanego klucza.

Pamiętaj też o nadpisaniu metody klonującej `clone()` zarówno w węźle jak i w drzewie BST.

Zadanie 4.

Dopisz komentarze dokumentacyjne do wszystkich (nie tylko publicznych) klas, interfejsów, pól i metod; dopisz także komentarz dokumentacyjny do całego pakietu `narzędzia` (plik `package-info.java`). Na koniec wygeneruj dokumentację dla całego pakietu za pomocą programu `javadoc`.

Zadanie 5.

Napisz program testujący (poza pakietem `narzędzia`), 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 takiego programu o nazwie `ManipulacjaDrzewami` i praca użytkownika z tym programem:

```
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
    {(l 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()`.

Uwaga.
Program należy napisać, skompilować i uruchomić w środowisku zintegrowanym *NetBeans*, przeznaczonym do pracy z językiem Java!