Język Java

Zestaw 5

2013 / 2014

Proszę napisać tylko **jedno** z zadań.
Klasy proszę umieścić w pakiecie `generics`.

1. **Kolor, Mieszadlo**
   Zapoznać się ze stroną

   `http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html`
   `http://en.wikipedia.org/wiki/RGB_color_model`

   Napisać typ wyliczeniowy `Kolor`. Powinien on zawierać przynajmniej takie przedmioty jak

   `BLACK, WHITE, GREY, RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA`

   Każdy przedmiot jest charakteryzowany przez trzy liczby typu `double` (od 0 do 1) `r, g, b` (odpowiednie składowe kolorów podstawowych RGB). Proszę napisać odpowiedni konstruktor. Napisać metodę (wykonywaną na przedmiocie)

   ```
   boolean porownaj(double r, double g, double b);
   ```

   która zwraca `true` gdy argumenty są dokładnie takie jak parametry koloru. Napisać metodę statyczną

   ```
   public static Kolor nazwij(double r, double g, double b);
   ```

   która zwraca przedmiot, który dokładnie odpowiada składowym r, g, b. Gdy taki przedmiot nie istnieje zwraca `null`. Użyć metody statycznej `values()` i konstrukcji

   ```
   for(Kolor a : values()){ ...}
   ```

   Napisać typ wyliczeniowy `Mieszadlo`. Powinien on zawierać rożne rodzaje mieszania kolorów np.

   ```
   ADD (addtywne)
   MUL (multiplikatywne)
   AVER (średnia)
   ```

   itd. Zadeklarować abstrakcyjną metodę

   ```
   abstract Kolor mieszaj(Kolor a, Kolor b);
   ```

A każda operacja (przedmiot) ma ją zaimplementować. Metoda ta zwraca wynik mieszania kolorów `a` i `b` lub `null`. Dodać metodę (statyczną) `main`, która wypisuje wszystkie możliwości mieszania. Pominąć wyniki dające `null`. Przykładowy rezultat dla kolorów `RED, GREEN, YELLOW, BLACK` i operacji `ADD, MUL` zamieściłem w dodatku.

2. **BinarySearchTree**

   Napisać klasę `BinarySearchTree<E extends Comparable<? super E>>` implementującą `Collection<E>` oraz klasę `Node<E extends Comparable<? super E>>`. Klasy te mają obsługiwać dynamiczną strukturę danych - **binarne drzewo poszukiwań**. W szczególności zdefiniować metody `add` (wstawianie elementu), `toArray` (wstawianie elementu) i `iterator`. Elementy struktury danych mają być dodawane dynamicznie w taki sposób aby przechodzenie drzewa zwracało je w posortowanej kolejności. Ostatnia metoda ma zwracać obiekt klasy
   `BinaryIterator<E extends Comparable<? super E>>`, która implementuje interfejs `Iterator<E>`. Zaimplementować metody iteracyjne (na potrzeby `iterator`) i rekurencyjne (na potrzeby `toArray`) do przechodzenia drzewa (*in-order*).

   Napisać interfejs `Pair<K, V>`, który posiada metody `K getKey()` i `V getValue()`. Napisać klasę `OrderedPair<K extends Comparable<? super K>, V>` implementującą interfejsy `Comparable <OrderedPair<K, V>>` i `Pair<K, V>`. Zdefiniować konstruktor oraz metody `compareTo`, `toString`, `getKey` i `getValue`.

   Stworzyć obiekt `list` klasy `BinarySearchTree` i
   kilka obiektów klasy `OrderedPair<String, String>`. Sprawdzić działanie iteratora wypisując posortowane elementy przy użyciu pętli `for(Object e : list)`.

3. **MergeSort**
   Zaimplementować algorytm sortowania przez scalanie z wykorzystaniem wątków.

Andrzej Görlich
andrzej.goerlich@uj.edu.pl
http://th.if.uj.edu.pl/~atg/Java