Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. C. Böhm, G. Zhang

Übungen zu Einführung in die Informatik

Hinweis: Bewertet. Nur für Diplom-BWL-Studenten. Abgabe von Montag, den 07.01.08 um 12.00 bis zum Montag, den 14.01.08 um 12.00

Aufgabe 11-1 **Farben Sortieren** Bewertet

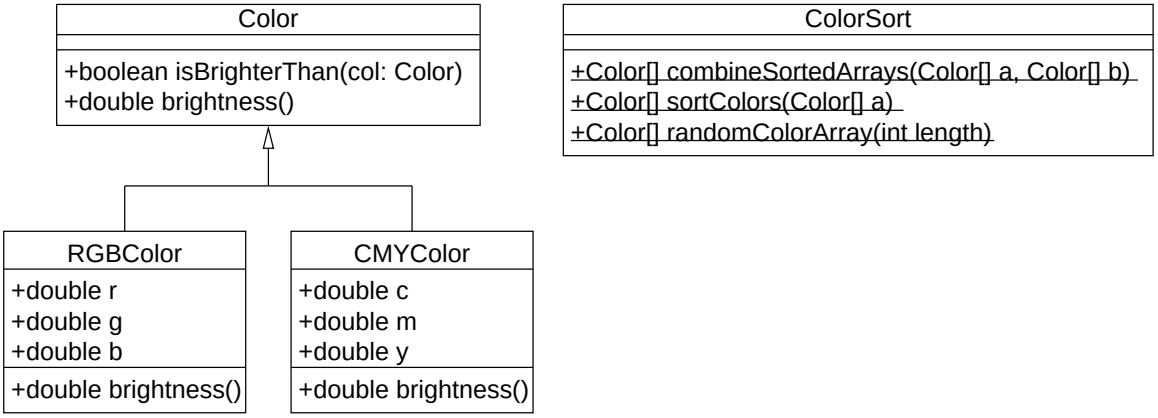

Abbildung 1: Klassendiagramm: Farben

a) Implementieren Sie bitte gemäß dem UML Diagramm (Abbildung 1) die Klassen `RGBColor` und `CMYColor`, und die Methode `isBrighterThan` in der Klasse `Color`, die Sie von unserer Web-Seite herunterladen können..

Die Klasse `RBGColor` implementiert eine Farbe im klassischen additiven Farbmodell. Im additiven Farbmodell werden alle Farben durch ihren Rot-, Grün- und Blauanteil repräsentiert – r, g und b. Dies sei durch `double`-Werte zwischen 0 und 1.0 realisiert. Die Helligkeit einer Farbe ist durch den Mittelwert der drei Farbanteile (also $\frac{1}{3}(r+g+b)$) gegeben – implementieren Sie bitte die `brightness`-Methode entsprechend, sie soll auch einen Wert zwischen 0 und 1.0 zurückgeben.

Die Klasse `CMYColor` implementiert eine Farbe im klassischen subtraktiven Farbmodell. Im subtraktiven Farbmodell werden alle Farben durch ihren Cyan-, Mangenta- und Zitronengelbanteil repräsentiert – c, m und y[1]. Dies sei durch `double`-Werte zwischen 0 und 1.0 realisiert. Die Schwärze einer Farbe ist durch den Mittelwert der drei Farbanteile (also $\frac{1}{3}(c+m+y)$) gegeben, keine Schwärze entspricht voller Helligkeit, volle Schwärze keiner Helligkeit – implementieren Sie bitte die `brightness`-Methode entsprechend, sie soll auch einen Wert zwischen 0 und 1.0 zurückgeben (wenn die Schwärze s von 0 bis 1.0 geht, dann ist die Helligkeit $1-s$).

b) Laden Sie die Klasse `ColorSort` von unserer Web-Seite herunter und implementieren Sie die drei statischen Methoden. Sie können diese Klasse dann mit dem `ColorArrayViewer.java` von unserer Web-Seite testen – einfach mit `javac ColorArrayViewer.java <IhreJavaDateien>` compilieren und mit `java ColorArrayViewer` testen.[2]

Die drei Methoden sollen folgendes leisten:

- **combineSortedArrays** bekommt zwei sortierte Color-Arrays der Länge n und m als Eingabe und liefert einen sortierten Color-Array der Länge $n+m$. Als Sortierkriterium dient dabei die Helligkeit der Farben der Eingabe-Arrays. Das Ergebnis-Array wird dadurch konstruiert, dass von den beiden Eingabe-Arrays jeweils das kleinere Element von links, welches noch nicht in das Ergibnis eingefügt wurde, dort eingefügt wird. Wenn ein Array komplett in das Ergebnis-Array übertragen wurde, so kann der Rest des verbleibenden Arrays komlett an das Ende des Ergebnis-Arrays gestellt werden. Würde man als nach diesem Verfahren Integers sortieren, würde man für `[1,3,6]` und `[2,4,5]` also `[1,2,3,4,5,6]` bekommen.
- **sortColors** ist ein rekursiver Algorithmus, der folgendermassen definiert ist:
  - Wenn der Array die Länge kleiner gleich 1 hat, so ist er sortiert und wird zurückgegeben.
  - Wenn der Array grösser ist, so wird er halbiert, die beiden hälften werden rekursiv sortiert und die beiden Teilergebnisse werden mit `combineSortedArrays` als Ergebnis zurückgegeben.

[1] y steht dabei für yellow.

[2] In der Praxis können ähnlich helle Farben von Monitor zu Monitor auch recht unterschiedlich hell wirken, nehmen Sie also bitte die Visualisierung nicht zu ernst...

- **randomColorArray** erzeugt ein Array der Gewünschten Länge (die Länge wird als Argument übergeben). Das Array wird abwechselnd mit einer `RBGColor`-Instanz und einer `CMYColor`-Instanz gefüllt. Die drei Farbanteile werden dabei mit Zufallszahlen zwischen 0 und 1.0 gesetzt (z.B. mit `Math.random()`).