Übungen zu Algorithmen und Datenstrukturen für Medizintechnik
Sommersemester 2021
René Raab, Robert Richer

Lehrstuhl für Maschinelles Lernen und Datenanalytik
Department Artificial Intelligence in Biomedical Engineering (AIBE)
(https://www.mad.tf.fau.de)

Aufgabenblatt 4 vom 09. Mai 2021, Abgabe am 30. Mai 2021, 22:00 Uhr

---

Aufgabe 4.1: Theorie

Punkte siehe StudOn
Zahlensysteme

Aufgabenstellung und Abgabe (individuell, nicht als Gruppe!) im StudOn.

Aufgabe 4.2: AuD-Calculator

25 Punkte
Zahlensysteme, Methoden

In dieser Aufgabe sollen Sie Teile eines einfachen wissenschaftlichen Taschenrechners implementieren, der Zahlen von verschiedenen Eingabe-Zahlensystemen einlesen, einfache Rechenoperationen ausführen und das Ergebnis in einem anderen Zahlensystem ausgeben kann. Der Rechner soll einfache Ganzzahlarithmetik mit *positiven* Zahlen beherrschen und Zahlen im

- Dezimal-,
- Binär-,
- Oktal- und
- Hexadezimalsystem

einlesen und ausgeben können.

1. Legen Sie ein neues Java-Project `04-AuDCalculator` mit einer Klasse `AuDCalculator` und einer `main`-Methode an.

2. Laden Sie das Zusatzmaterial aus dem StudOn (`04-material.zip`) herunter, und kopieren Sie die Datei `CalculatorGUI.java`, die die GUI für den Taschenrechner bereitstellt, in Ihr Projektverzeichnis. Die Teilaufgaben sollen in der Datei `AuDCalculator.java` implementiert werden.

3. Fügen Sie die Zeile

   ```
   new CalculatorGUI();
   ```

   in die `main`-Methode der Klasse `AuDCalculator` hinzu, um die GUI des Taschenrechners anzuzeigen, wenn Sie Ihr Programm starten (siehe Abbildung 1).

   Hinweis: Kommentieren Sie diese Codezeile am Anfang noch aus und testen Sie die ersten Methoden, indem Sie sich die Ergebnisse auf die Konsole ausgeben lassen.

   Die Taschenrechner-GUI bricht mit einer Fehlermeldung ab, falls die Methoden `convertToInt()`, `convertToNumberSystem()` und `computeArithmetic()` noch nicht implementiert sind.

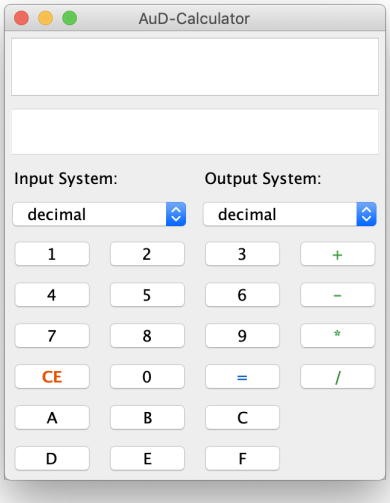

Abbildung 1: Grafische Benutzeroberfläche des *AuD-Calculator*.

Achtung: Die folgenden Teilaufgaben sollen gelöst werden, **ohne** Methoden aus der Java-API, die bereits Zahlen zwischen verschiedenen Zahlensystemen umrechnen, zu verwenden! Insbesondere dürfen Sei keine Methoden der Klassen `Byte`, `Short`, `Integer` und `Long` verwenden!

Methoden der Klasse `String` (wie z. B. `String.charAt(i)` oder `String.substring(i)`) sind aber erlaubt und dürften im weiteren Verlauf sehr hilfreich sein!

4. **Hilfsmethoden**

   a) Legen Sie zunächst die Hilfsmethode `public static boolean isKnownNumberSystem(int inputSystem)` an. Sie soll überprüfen, ob das übergebene Zahlensystem `inputSystem` vom AuD-Calculator unterstützt wird und soll das entsprechende Ergebnis als `boolean`-Wert an den Aufrufer zurückgeben.

   b) Implementieren Sie eine weitere Hilfsmethode `isInNumberSystem`, die die beiden Parameter `char number` und `int inputSystem` besitzt und einen `boolean`-Wert zurückgibt. Diese Methode soll überprüfen, ob das übergebene Zeichen (in Großbuchstaben) korrekt im Eingabesystem eingegeben wurde, wie z. B.:

```
isInNumberSystem('0', 2) => true
isInNumberSystem('A', 2) => false
isInNumberSystem('9', 8) => false
isInNumberSystem('9', 10) => true
isInNumberSystem('B', 16) => true
isInNumberSystem('Y', 10) => false
```

5. **Umrechnung $X \rightarrow$ Dezimalsystem**
   Implementieren Sie die Methode `public static int convertToInt(String inputNumber, int inputSystem)`. Diese Methode soll eine Zahl, eingegeben als `String` im Zahlensystem `inputSystem`, in einen `int` (also in das Dezimalsystem) konvertieren.

   a) Falls das `inputSystem` nicht vom AuD-Calculator unterstützt wird, soll die Methode `convertToInt` -1 zurückgeben.

   b) Erstellen Sie eine Schleife, die den String `inputNumber` Zeichen für Zeichen einliest und

überprüft, ob jedes Zeichen gültig ist. Falls nicht, soll die Methode -1 zurückgeben.

c) Falls die Eingabe korrekt ist, soll die Zahl mithilfe des *Horner-Schemas* in das Dezimalsystem konvertiert und zum Schluss zurückgegeben werden.

Testen Sie `convertToInt` in der `main`-Methode ausgiebig! Legen Sie Strings mit Zahlen in unterschiedlichen Zahlensystemen an, konvertieren Sie die Zahlen und überprüfen Sie das Ergebnis.

6. **Umrechnung Dezimalsystem $\rightarrow$ *X***
Nun soll Ihr AuD-Calculator Zahlen vom Dezimal- in ein anderes Zahlensystem umwandeln und ausgeben können. Implementieren Sie hierfür die Methode `public static String convertToNumberSystem(int number, int outputSystem)`. Diese Methode soll die `int`-Zahl `number` in das Zahlensystem `outputSystem` umrechnen. Im Anschluss soll die String-Repräsentation der konvertierten Zahl zurückgegeben werden.

   a) Falls es sich um ein Zahlensystem handelt, das nicht vom AuD-Calculator unterstützt wird, soll die Methode den String `"error"` zurückgeben.

   b) Legen Sie anschließend einen leeren Ergebnis-String an, der von der Methode zurückgegeben werden soll. Erstellen Sie eine Schleife, in der jede Ziffer der Zahl `number` im neuen Zahlensystem `outputSystem` berechnet wird und fügen Sie den Ergebnis-String in der Schleife zusammen.

   Falls in das Hexadezimalsystem umgerechnet werden soll, müssen Sie die Zahlen $10 - 15$ natürlich durch die Großbuchstaben $A - F$ ersetzen.

   c) Stellen Sie nach der Schleife sicher, dass die korrekten Präfixe für die Zahlensysteme verwendet werden. Verwenden Sie `"b"` für das Binärsystem, `"0"` (»Null«, nicht »O«!) für das Oktalsystem und `"0x"` für das Hexadezimalsystem. Geben Sie den Ergebnis-String anschließend zurück.

Testen Sie `convertToNumberSystem` in der `main`-Methode ausgiebig! Konvertieren Sie verschiedene `int`-Werte in die unterschiedlichen Zahlensysteme und überprüfen Sie die Ergebnisse.

7. Der nächste Schritt ist es, die arithmetischen Rechenoperationen des AuD-Calculators zu implementieren:

   a) Legen Sie zuerst die Methode `public static String computeArithmetic(int firstNumber, int secondNumber, String operator, int outputSystem)` an.

   b) Berechnen Sie in der Methode das Ergebnis der Rechenoperation, abhängig vom eingegebenen Operator (z. B.: `firstNumber + secondNumber`).

   c) Falls eine ungültige arithmetische Operation vorliegt (beispielsweise eine Division durch 0), soll die Methode den String `"Calculation Error"` zurückgeben.

   d) Ebenso soll der Taschenrechner nur für positive Zahlen funktionieren. Falls das Ergebnis der arithmetischen Operation negativ ist, soll die Hilfsmethode ebenfalls `"Calculation Error"` zurückgeben.

   e) Ansonsten soll die das Ergebnis einer *gültigen* arithmetischen Operation in das Zahlensystem `outputSystem` umwandeln und an den Aufrufer zurückgeben.

Hinweis: Sie können Sie ab jetzt mit der grafischen Benutzeroberfläche, indem Sie die auskommentierte Zeile in der `main`-Methode wieder einkommentieren.

Testen Sie Ihr Programm ausgiebig! Achten Sie dabei insbesondere darauf, unterschiedliche Zahlensysteme für die Ein- und Ausgabe (wie z. B. in Abbildung 2) zu testen und überprüfen Sie, ob alle Rechenoperationen funktionieren.

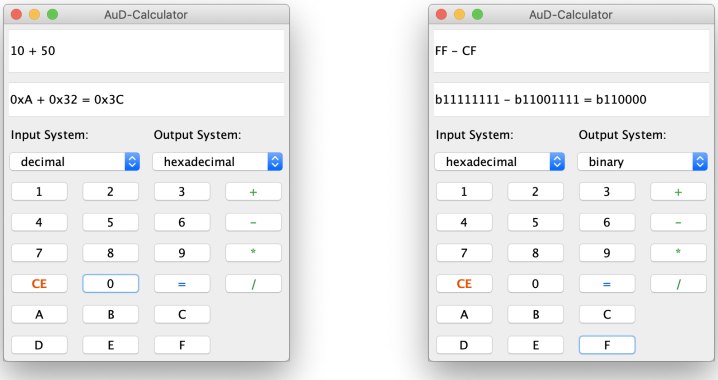

Abbildung 2: Beispiele für verschiedene Rechenoperationen mit dem AuD-Calculator.

8. Geben Sie die Datei `AuDCalculator.java` mit allen implementierten Methoden ab!

Aufgabe 4.3: Farben 18 Punkte

OOP, Zahlensysteme, Bitweise Operatoren

Farben werden am Monitor als additive Mischung der Grundfarben *rot*, *grün* und *blau* dargestellt (der sog. RGB-Farbraum). Die Farbanteile der einzelnen Grundfarben werden meistens als ganze Zahl zwischen 0 und 255 angegeben. 0 bedeutet, dass die Grundfarbe nicht, 255, dass die Grundfarbe voll zur Mischfarbe beiträgt.

Einige Beispiele, angegeben als Tripel (rot, grün, blau):

- `BLACK`: $(0, 0, 0)$
- `WHITE`: $(255, 255, 255)$
- `GRAY`: $(128, 128, 128)$
- `RED`: $(255, 0, 0)$
- `GREEN`: $(0, 255, 0)$
- `BLUE`: $(0, 0, 255)$
- `ORANGE`: $(255, 160, 0)$
- `PEACHPUFF`: $(255, 218, 185)$
- `VIOLET`: $(238, 130, 238)$

Im Speicher werden diese Tripel normalerweise als eine einzige Integer-Zahl dargestellt. Für den Wertebereich von 0 bis 255 werden genau 8 Bit benötigt, die drei Farbkanäle brauchen zusammen also 24 Bit. In die ersten (*most significant*) 8 Bit davon schreibt man den Rot-Wert, danach kommen 8 Bits für Grün und die untersten (*least significant*) 8 Bit sind schließlich für den Blau-Kanal [1].

$\text{RGB} = (255, 160, 0) = (11111111_2, 10100000_2, 00000000_2)$

R:	00000000	11111111	00000000	00000000
G:	00000000	00000000	10100000	00000000
B:	00000000	00000000	00000000	00000000
RGB:	00000000	11111111	10100000	00000000
⇒	00000000	11111111	10100000	00000000

[1] Die verbleibenden obersten 8 Bit eines 32-Bit-Integer-Wertes werden häufig für den sog. *Alpha*-Kanal verwendet, in dem die Deckkraft einer Farbe angegeben werden kann. Diesen werden wir in dieser Aufgabe jedoch nicht verwenden

Die Farbe Orange wird also als Integer-Wert 11111111 10100000 00000000_2 dargestellt.

Diese 24-stellige Binärzahl ist leider eher unhandlich und lang – man könnte sie stattdessen beispielsweise als Dezimalzahl $16\,752\,640_{10}$ angeben, um eine kürzere Darstellung zu erhalten. Dann kann man allerdings nicht mehr direkt die Werte der einzelnen Farbkanäle erkennen. Abhilfe schafft hier das Hexadezimalsystem: Zwei Hexadezimalziffern zusammen entsprechen exakt 8 Bit, die gleiche Zahl lässt sich also schreiben als $(\text{FF A0 00})_{16}$. Diese Darstellung wird z. B. auch in HTML[2] oder in Bildbearbeitungsprogrammen verwendet und dort mit einem Rautezeichen (»`#`«) als Präfix notiert, also `#FFA000`.

In dieser Aufgabe sollen Sie eine Java-Klasse `Color` zur Darstellung eines Farbwertes schreiben, die diese unterschiedlichen Darstellungsarten beherrscht. Zudem

1. Legen Sie ein neues Projekt `04-Color` und eine Java-Klasse `Color` an.

2. Die Klasse soll ein Attribut `rgb` vom Typ `int` haben. Darin soll die `int`-Darstellung der Farbe wie oben beschrieben gespeichert werden. Stellen Sie sicher, dass dieses Attribut von außen nicht sichtbar ist!

3. Das Attribut `rgb` muss nun noch initialisiert werden. Dafür soll die Klasse mehrere öffentliche Konstruktoren erhalten:

   - Einen Konstruktor, der direkt die `int`-Darstellung der Farbe übergeben bekommt und sie in `rgb` speichert.
   - Einen Konstruktor, der drei `int`-Werte `red`, `green` und `blue` übergeben bekommt. Überprüfen Sie zunächst, ob die übergebenen Werte für die jeweiligen Farbkanäle gültig sind, sich also im Interval $[0, 255]$ befinden. Falls nicht, soll eine entsprechende Fehlermeldung auf `stderr` ausgegeben werden und der ungültige Wert auf die jeweilige Intervallgrenze (0 oder 255) gesetzt werden.

     Verwenden Sie die Operatoren `<<` (*left shift*) und `|` (*bitweises Oder*), um aus diesen drei Werten zwischen 0 und 255 wie oben beschrieben den korrekten Wert für `rgb` zu berechnen!
   - Einen Standard-Konstruktor ohne Parameter, in dem der Variablen `rgb` die Farbe Schwarz zugewiesen werde soll.

4. Erstellen Sie eine öffentliche *get*-Methode `getRgb()`, mit der der Wert von `rgb` von außen gelesen werden kann!

5. Legen Sie zum Testen der bisher implementierten Funktionen eine `main`-Methode an. Erstellen Sie dort einige `Color`-Objekte und überprüfen Sie, ob Sie die Konstruktoren korrekt implementiert haben!

6. Um die einzelnen Farbkanäle (aus `rgb`) auslesen zu können, soll die `Color`-Klasse drei öffentliche *getter*-Methoden `getRed()`, `getGreen()` und `getBlue()` bekommen.

   Verwenden Sie den Operator `>>` (*right shift*), um die relevanten Bits an die richtige Stelle zu verschieben, und `&` (*bitweises Und*), um alle für den jeweiligen Kanal nicht benötigten Binärziffern auf 0 zu setzen, sodass die Methoden den jeweiligen Farbkanal-Wert als `int`-Zahl im Bereich von 0 bis 255 zurückgeben können.

   Achtung: Das Attribut `rgb` darf dabei natürlich nicht verändert werden!

7. Die öffentliche Methode `getHex()` soll die 6-stellige Hexadezimaldarstellung (mit führenden Nullen) von `rgb`, zusammen mit dem `#`-Präfix als `String` in Großbuchstaben zurückgeben.

[2] Eine Auflistung von gängigen HTML-Farben finden Sie hier: https://www.w3schools.com/colors/colors_names.asp

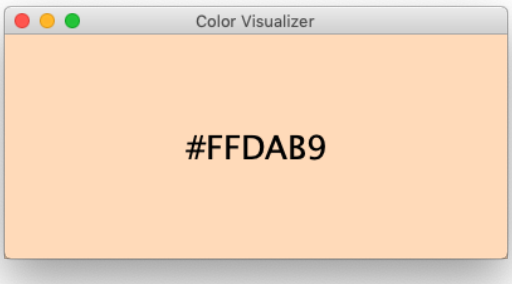

Abbildung 3: Farbdarstellung durch den `ColorVisualizer` für die Farbe *PeachPuff*.

Rechnen Sie dazu den Wert im Attribut `rgb` (der natürlich im Dezimalsystem gespeichert ist) ins Hexadezimalsystem um.

Hinweis: In der Aufgabe *AuD-Calculator* sollten Sie die beiden Methoden, um Zahlen aus einem Zahlensystem ins Dezimalsystem (und umgekehrt) umzuwandeln, implementieren, **ohne** Hilfsmethoden zur Umwandlung aus der Java-API zu verwenden. In dieser Aufgabe dürfen Sie diese nun verwenden, vor allem Methoden der Klasse `Integer` dürften hierbei hilfreich sein. Informieren Sie sich in der Java-Dokumentation, wie Sie die entsprechenden Methoden korrekt anwenden.

8. Um die verschiedenen Farben visualisieren zu können stellen wir Ihnen die Klasse `ColorVisualizer` im als Zusatzmaterial im StudOn (`04-material.zip`) zur Verfügung:

   a) Laden Sie diese Datei herunter, entpacken Sie sie und kopieren Sie die Datei `ColorVisualizer.java` in das Verzeichnis, in das sich auch Ihre Datei `Color.java` befindet.

   b) Die Klasse besitzt einen Konstruktor `ColorVisualizer(Color color)`, der das `Color`-Objekt der Farbe übergeben bekommt und die Farbe – wie in Abbildung 3 – darstellt.

   Hinweis: Solange die Methode `getHex()` in der Klasse `Color` noch nicht implementiert ist, bricht der *ColorVisualizer* mit einer Fehlermeldung ab!

9. Ergänzen Sie nun einen weiteren öffentlichen Konstruktor, der die Hexadezimaldarstellung der Farbe als `String` mit `#`-Präfix (also z. B. `#FFA000`) übergeben bekommt. Nach dem Entfernen des Präfixes soll die Zahl in das Dezimalsystem umgewandelt werden und im Attribut `rgb` gespeichert werden. Sie können davon ausgehen, dass nur gültige Hexadezimalziffern vorkommen.

10. Überschreiben Sie die `toString()`-Methode der Klasse `Color`, sodass die Methode die Hexadezimaldarstellung des Farbwertes mit `#`-Präfix zurückgibt.

11. Nun soll die Farbklasse noch ein paar Funktionalitäten hinzubekommen, um Komplementärfarben zu erzeugen oder Farben zu mischen:

    a) *Komplementärfarbe:* Erstellen Sie die öffentliche Methode `Color complementaryColor()`, die die Komplementärfarbe zum aktuellen `Color`-Objekt erzeugen soll (siehe Beispiel in Abbildung 4). Die Komplementärfarbe lässt sich berechnen, indem pro Farbkanal die Differenz zu 255 gebildet wird. Erzeugen in der Methode ein entsprechendes Farbobjekt und geben Sie dieses zurück.

    b) *Farben mischen:* Implementieren Sie eine öffentliche Methode `mixColor`, die als Parameter ein `Color`-Objekt namens `color` übergeben bekommt, die Farbe mit dem aktuellen Farbwert

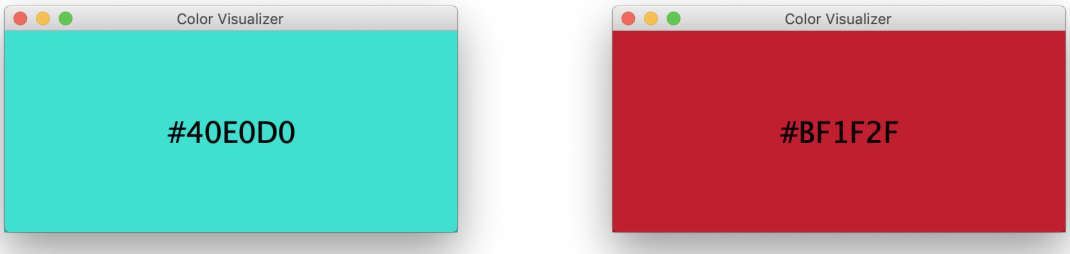

Abbildung 4: Darstellung der Farbe *Turquoise* (links) und Komplementärfarbe (rechts)

mischen und ein `Color`-Objekt mit der neuen Farbe $C_{neu} = (r_{neu}, g_{neu}, b_{neu})$ zurückgeben soll. Verwenden Sie dafür folgende Formel, wobei r_i, g_i und b_i die entsprechenden Farbkanäle der zu mischenden Farben sind:

$$r_{neu} = (r_1 + r_2)/2$$
$$g_{neu} = (g_1 + g_2)/2$$
$$b_{neu} = (b_1 + b_2)/2$$

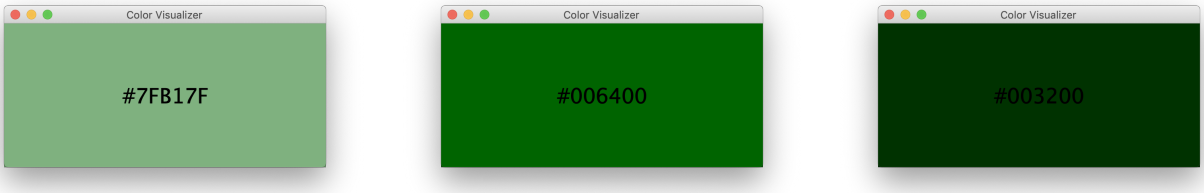

Abbildung 5: Darstellung der Farbe *DarkGreen* (mitte), jeweils gemischt mit der Farbe *Weiß* (links) und *Schwarz* (rechts)

12. Zu guter Letzt soll die Klasse noch einige häufig verwendeten Farben bereitstellen. Legen Sie die folgenden Farben daher als öffentliche, *nicht mehr veränderliche* `Color`-Objekte an:

    - `BLACK`: $(0, 0, 0)$
    - `WHITE`: $(255, 255, 255)$
    - `GRAY`: $(128, 128, 128)$
    - `RED`: $(255, 0, 0)$
    - `GREEN`: $(0, 255, 0)$
    - `BLUE`: $(0, 0, 255)$

    Damit sie auch ohne eine Instanz der Klasse zur Verfügung stehen, müssen diese Attribute natürlich *statisch* sein.

13. Ergänzen Sie weitere Tests in der `main`-Methode, sodass auch die neuen Methoden (insbesondere die zur Erstellung der Komplementärfarbe und zum Mischen von Farben) getestet werden! Suchen Sie sich dazu mindestens vier weitere, bisher noch nicht verwendete Farbwerte aus dieser Farbtabelle (https://www.w3schools.com/colors/colors_names.asp) aus und erstellen Sie dazu entsprechende `Color`-Objekte.

14. Geben Sie die Datei `Color.java` ab!

Sollte Ihr Programm nicht übersetz- bzw. ausführbar sein, wird die Lösung mit 0 Punkten bewertet. Stellen Sie also sicher, dass IntelliJ IDEA keine Fehler in Ihrem Programm anzeigt, Ihr Programm übersetz- und ausführbar ist sowie die in der Aufgabenstellung vorgegebenen Namen und Schnittstellen *exakt* eingehalten werden. Geben Sie am Schluss die Dateien `AuDCalculator.java` und `Color.java` über die EST-Webseite ab. Wenn Sie die Aufgabe zusammen mit einem

Übungspartner bearbeitet haben, geben Sie im EST unbedingt dessen Gruppenabgabe-Code an! Kontrollieren Sie, ob Ihre Namen am Anfang aller Dateien angegeben sind – schreiben Sie im Quellcode Ihre Angaben in einen Kommentar. Im EST-Abgabesystem können Sie modifizierte Dateien mehrfach abgeben. Nur die zuletzt hochgeladene Version wird bewertet.