Teil A

Welche Fehler begeht der Entwickler hier?

Ein Entwickler legt eine neue Java-Quelltextdatei mit dem Namen `MyApp.java` an und füllt sie mit folgendem Inhalt:

```
public class App {

}
```

Anschließend will der Entwickler seine Quelltextdatei mit dem Java Compiler `javac` übersetzen. Dazu gibt er folgendes Kommando ein:

```
javac MyApp.java
```

Der Compiler meldet einen Fehler. Woran kann das liegen? Welche Änderungen sind am Quelltext vorzunehmen, damit der Compiler ein Kompilat erzeugen kann?

Der Entwickler hat seinen Fehler im Quelltext behoben. Nun will er sein Programm mit Hilfe des Java Launchers `java` ausführen. Dazu tippt er folgendes Kommando ein:

```
java MyApp.class
```

Abermals erhält der Entwickler einen Fehler. Was hat er falsch gemacht? Wie müsste das Kommando stattdessen lauten?

Der Entwickler bemerkt seinen Fehler und ändert das Kommando entsprechend ab. Doch jetzt erhält er plötzlich einen weiteren Fehler. Angeblich findet der Java Launcher keine `main` Methode. Was hat es damit auf sich? Wie lässt sich das Problem beheben?

Welche Ergebnisse liefern die folgenden Zuweisungen?

Teste folgende Anweisungen in der JShell. Überlege dir vorher, welches Ergebnis zu erwarten ist und prüfe erst anschließend, ob deine Vermutung richtig war. Kannst du die Ergebnisse begründen?

Hinweis: Manche Anweisungen sind syntaktisch falsch. Welche sind es und warum schlagen sie fehl?

Bemerkung: Die JShell startest du mit dem Befehl `jshell`. Du kannst die JShell verlassen, indem du den Befehl `/exit` eintippst und mit Taste Enter bestätigst.

```
int a = 1703;
int b = 0703;
```

```
int c = 4 / 8;
int d = 0xCAFE;
int e = Integer.MIN_VALUE - 1;
int f = Integer.MAX_VALUE + 1;
int g = 07709;
int h = -2e3;
byte i = 120 + 9;
byte j = 0b1000_0000;
byte k = (byte)(0b1000_0000);
byte l = (byte)(0xFF);
byte m = 1;
byte n = 2;
byte o = m + n;
byte p = (byte)(m + n);
int q = 4 / 3;
int r = 5 / 2.5;
double s = 1 / 2;
double t = 1.0 / 2;
double u = (double)(1 / 2);
double v = 1 / 3.0;
float w = 1;
float x = 0.5;
float y = 0.5f;
float z = 3 * 5 / 2 + 2.5f;
float aa = 1.0 + 2.5f;
float ab = 1.0f + 2.5f;
double ac = 1.0 + 2.5f;
char ad = 'c';
char ae = "c";
char af = 0xff;
int ag = (int)('U');
char ah = 'ab';
char ai = '\n';
char aj = '"';
char ak = '\'';
boolean al = false;
boolean am = 1;
boolean an = 0;
boolean ao = (boolean)(1 + 1);
boolean ap = 4 > 3;
boolean aq = 4 == 4;
```

Wie lauten die Wrapper Klassen der primitiven Datentypen?

Finde heraus, wie die Wrapper-Klassen der primitiven Datentypen `byte`, `short`, `char`, `int`, `float`, `double` und `boolean` heißen. Nutze dazu die JDK API Documentation.

Teil B

Beachte folgende Hinweise:

- Falls dir für eine Aufgabe keine Lösung einfällt, widme dich einfach der nächsten Aufgabe. Vielleicht kommt dir später noch eine Idee.
- Ich rate dir dringend davon ab, eine KI einzusetzen, um diese Aufgaben zu lösen. Der Lerneffekt würde dadurch enorm sinken.
- Sofern nicht anders vorgegeben, ist pro Aufgabenstellung je ein Programm zu schreiben, das seine Eingabedaten über Kommandozeilenargumente erhält.
- Denke daran, dass Kommandozeilenargumente immer als Zeichenketten an das Java Programm übergeben werden. Argumente mit Leerzeichen sind auf der Kommandozeile in einfache oder doppelte Hochkommas zu setzen. Beispiel: `java MyApp argument1 argument2 "argument with spaces"`.
- In den Aufgabenstellungen heißt das Programm immer `MyApp`, aber du bist angehalten, passendere Namen zu wählen.
- Die in den Aufgabenstellungen genannten Tipps dienen nur als Denkanstoß und sind nicht verpflichtend. Meistens gibt es viele Lösungsmöglichkeiten für ein Problem.
- Wenn nicht anders vorgegeben, darfst du davon ausgehen, dass der Nutzer seine Argumente im korrekten Format angibt.
- Braucht ein Programm mindestens ein Argument, aber hat der Nutzer keines angegeben, soll das Programm den Nutzer über den korrekten Aufruf informieren.

Temperaturen in Grad Celsius und Fahrenheit umwandeln

Es ist ein Programm zu schreiben, das eine Temperatur in Grad Celsius (C) in Grad Fahrenheit (F) umrechnet. Das Programm soll außerdem in der Lage sein, eine Temperatur von Grad Fahrenheit (F) in Grad Celsius (C) umzuwandeln.

Programmaufruf: `java TemperatureConverter temperatur`. Beispiel: `java TemperatureConverter 20.5C` gibt `68.9F` aus. Beispiel 2: `java TemperatureConverter 68.9F` gibt `20.5C` aus.

Auf Teilbarkeit prüfen

Schreibe ein Programm, das prüft, ob eine ganze Zahl `z` durch eine andere ganze Zahl `t` teilbar ist. Es darf davon ausgegangen werden, dass der Nutzer nur positive Zahlen angibt.

Programmaufruf: `java MyApp z t`. Beispiel: `java MyApp 100 25` gibt `true` aus und `java MyApp 100 30` gibt `false` aus.

Im Anschluss ist das Program so zu erweitern, dass es eine Gleichung mit Teiler und ggf. auch mit Restwert (Modulo) ausgibt. Beispiel: `100 = 4 * 25` ist für `z = 100` und `t = 25` auszugeben. Bei `z = 100` und `t = 30` erscheint hingegen `100 = 3 * 30 + 10`.

Tipp: Verwende den Modulo-Operator `%` und die Integer-Division `/`.

Zeichenkette umkehren

Eine Zeichenkette `s` ist in umgekehrter Reihenfolge auszugeben.

Das Programm ist wie folgt aufzurufen: `java MyApp s`. Beispiel: `java MyApp "Hello World"` gibt `dlroW olleH` aus.

Tipp: Verwende eine herkömmliche for-Schleife.

Römische Ziffer in Dezimalzahl umwandeln

Römische Zahlen bestehen aus den Ziffern `i` (1), `v` (5), `x` (10), `l` (50), `c` (100), `d` (500) und `m` (1000). Es soll ein Programm geschrieben werden, das eine römische Ziffer in eine Dezimalzahl umwandelt.

Programmaufruf: `java MyApp ziffer`. Beispiel: `java MyApp v` gibt `5` aus und `java MyApp c` gibt 100 aus. Der Nutzer darf die Ziffer als Groß- oder Kleinbuchstabe angeben.

Das Programm soll keine vollständigen Römischen Zahlen wie `mcxx` in Dezimalzahlen umwandeln. Es geht hier nur um eine Ziffer wie `c` oder `m`. Siehe Zusatzaufgabe.

Vorgabe für Implementierung: Verwende eine moderne switch-Anweisung / switch-Expression.

Zusatzaufgabe 1

Erweitere das Program so, dass vollständige römische Zahlen übersetzt werden. Beispiel: `java MyApp mcxx` gibt `1120` aus. Hier zu sind folgende Regeln zu berücksichtigen:

- Steht eine kleinere Ziffer links von einer größeren Ziffer, so ist die kleinere Ziffer von der größeren abzuziehen. Andernfalls werden die Ziffern aufaddiert. Beispiel: `iv` ist 4 und `xc` ist 90. `cv` ist 105.
- Dieselbe Ziffer darf nicht mehr als dreimal nacheinander erscheinen. Beispiel: `iii` ist korrekt, aber `iiii` nicht, da die 4 als `iv` ausgedrückt wird.
- Folgende Ziffern dürfen grundsätzlich nicht wiederholt werden: `v`, `l` und `d`.
- Folgende Subtraktionen sind erlaubt: `iv` und `ix`, `xl` und `xc`, `cd` und `cm`.
- Die größtmögliche Zahl ist 3999, also `mmmcmxcix` (3000 + 900 + 90 + 9).

Zusatzaufgabe 2

Schreibe ein Programm, das eine Dezimalzahl in eine römische Zahl umwandelt. Es gelten dieselben Regeln wie in Zusatzaufgabe 1.

Schaltjahr bestimmen

Schreibe ein Programm, das bestimmt, ob ein eingegebenes Jahr ein Schaltjahr ist. Ein Jahr ist ein Schaltjahr, wenn es durch 4 teilbar ist. Falls das Jahr aber auch durch 100 teilbar ist, handelt es sich um kein Schaltjahr. Wenn das Jahr jedoch durch 400 teilbar ist, dann ist es trotzdem ein Schaltjahr.

Das Programm ist so aufzurufen: `java MyApp jahr`. Beispiel: `java MyApp 2024` gibt `true` aus. `java MyApp 1900` gibt `false` aus. `java MyApp 1600` gibt `true` aus.

Implementiere das Programm in zwei Varianten:

- Verwende `if` in Kombination mit `else if` und `else`.
- Verwende nur `if` und verknüpfe die Bedingungen mit `&&` und `||`.

Tipp: Benutze den Modulo-Operator `%` um die Teilbarkeit zu prüfen.

Schaltjahre in einem Bereich bestimmen

Es ist ein Programm zu schreiben, das alle Schaltjahre ausgibt, die sich in einem vorgegebenen Intervall befinden. Das Intervall wird durch Angabe eines Start- und eines Endjahres definiert.

Programmaufruf: `java MyApp start ende`. Beispiel: `java MyApp 1896 1915` gibt die Jahre 1896, 1904, 1908 und 1912 aus.

Tage eines Monats bestimmen

Es sind die Tage eines Monats auszugeben. Schaltjahre sind dabei zu berücksichtigen. Der Monat ist als Zahl anzugeben.

Programmaufruf: `java MyApp monat jahr`. Beispiel: `java MyApp 2 2024` gibt 29 aus und `java MyApp 2 2025` gibt 28 aus.

Vorgabe für Implementierung: Verwende ein modernes switch-Statement / switch-Expression.

Palindrom-Test

Es ist zu prüfen, ob eine Zeichenkette `s` ein Palindrom ist. Palindrome sind Wörter, die sowohl vorwärts als auch rückwärts gelesen dasselbe Wort ergeben. Bildlich gesprochen handelt es sich um Wörter, die gespiegelt sind. Beispiele: otto, anna, hannah, rentner, lagerregal.

Das Programm ist folgendermaßen aufzurufen: `java MyApp s`. Beispiel: `java MyApp "rentner"` gibt `true` und `java MyApp "java"` den Text `false` aus.

Vorgabe zur Implementierung: Verwende eine for-Schleife mit zwei Laufvariablen.

Zeichenketten vermischen

Zwei Zeichenketten `a` und `b` sollen nach dem Reißverschluss-Verfahren gemischt werden.

Das Programm ist wie folgt aufzurufen: `java MyApp a b`. Beispiel: `java MyApp "abc" "123"` gibt `a1b2c3` aus.

Falls sich die Längen von `a` und `b` unterscheiden, sind die verbleibenden Zeichen der längeren Zeichenkette am Ende auszugeben. Beispiel: `java MyApp "abcde" "123"` gibt `a1b2c3de` aus.

Tipp: Nutze eine herkömmliche for-Schleife.

Zusatzaufgabe

Erweitere das Programm so, dass es beliebig viele Zeichenketten miteinander mischt. Beispiel: `java MyApp "abc" "1234" "xy" "-+:"` gibt `a1x-b2y+c3:4` aus.

Zeichenketten auffüllen

Eine Zeichenkette `s`, die weniger als `n` Zeichen enthält, soll mit einem Füllzeichen `f` bis zur Länge `n` von rechts aufgefüllt werden.

Das Programm ist wie folgt aufzurufen: `java MyApp s n f`. Beispiel: `java MyApp "a b" 5 X` gibt `a bXX` aus.

Falls `s` schon lang genug ist, soll das Programm `s` unverändert ausgeben. Beispiel: `java MyApp "abcde" 3 X` gibt `abcde` aus.

Im Anschluss ist das obige Programm in zwei weiteren Varianten zu implementieren:

- Die Zeichenkette ist von links aufzufüllen.
- Die Zeichenkette ist beidseitig aufzufüllen. Beispiel: `java "abc" 7 X` gibt `XXabcXX` aus und `java MyApp "abc" 6 X` gibt `XXabcX` aus.

Tipp: Verwende `Integer.parseInt` zum Konvertieren von `String` nach `int`. Verwende den `+` Operator zum Verketten von Strings.

Zeichenketten abkürzen

Eine Zeichenkette `s`, die mehr als `n` Zeichen enthält, soll durch eine Ellipsis `...` (3 Punkte) so abgekürzt werden, dass sie genau die Länge `n` hat. Hat `s` höchstens `n` Zeichen, so ist `s` unverändert auszugeben.

Das Programm ist wie folgt aufzurufen: `java MyApp s n`. Beispiel: `java MyApp "Ein langer Text" 10` gibt `Ein lan...` aus.

Wenn `n <= 3` gilt und die Länge von s größer als `n` ist, sind `n` Punkte (`.`) auszugeben. Beispiel: `java MyApp "abcde" 3` gibt `...` aus und `java MyApp "abcde" 2` gibt `..` aus.

Im Anschluss ist das obige Programm in zwei weiteren Varianten zu implementieren:

- Die Zeichenkette ist am Beginn zu kürzen. Beispiel: `java MyApp "Ein langer Text" 10` gibt `...er Text` aus.
- Die Zeichenkette ist in der Mitte zu kürzen. Beispiel: `java MyApp "Ein langer Text" 10` gibt `Ein...Text` aus.

Tipp: Verwende die Instanzmethode `substring` der Klasse `String` und den `+` Operator zum Verketten. Du kannst auch einen `StringBuilder` verwenden.

Umwandlung von Tagen, Stunden, Minuten und Sekunden in Sekunden

Eine Dauer, die in Form von Tagen, Stunden, Minuten und Sekunden anzugeben ist, soll in Sekunden umgerechnet werden. Beispiel: 1 Tag, 2 Stunden, 5 Minuten und 20 Sekunden ergeben insgesamt 93920 Sekunden.

Das Programm ist wie folgt aufzurufen: `java MyApp 1d 2h 5m 20s`. Die Argumente dürfen vom Nutzer in *beliebiger* Reihenfolge angegeben werden! Außerdem soll es möglich sein, Argumente wegzulassen. Beispiel: `java MyApp 20s 1d`. Das Programm erkennt anhand der Suffixe `d` (Tage), `h` (Stunden), `m` (Minuten) und `s` (Sekunden), um welche Zeiteinheiten es sich handelt. Es darf davon ausgegangen werden, dass der Nutzer Argumente im korrekten Format angibt.

Taucht eine Zeiteinheit mehrfach auf, sind die Einheiten zu addieren. Beispiel: `java MyApp 20s 1d 60s` ist gleichbedeutend mit `java MyApp 80s 1d`. Steht vor der Zeiteinheit ein `-` (Minus), so sind die Zeiteinheiten zu subtrahieren. Beispiel: `java MyApp 1d 2h 5m 20s -2m` ergibt 93800 Sekunden.

Erhält das Programm gar keine Argumente, ist ein Text auszugeben, der den Nutzer darüber informiert, wie man das Programm korrekt aufruft - eine Synopsis.

Tipp: Du kannst hier Methoden der Klasse `String` verwenden, z.B. `endsWith`. Für die Umwandlung von Strings in Integer, steht dir die statische Methode `parseInt` der Klasse `Integer` zur Verfügung.

Umwandlung von Sekunden in Tage, Stunden, Minuten und Sekunden

Eine ganze Zahl soll in Tage, Stunden, Minuten und Sekunden zerlegt werden. Beispiel: 93920 Sekunden entsprechen 1 Tag, 2 Stunden, 5 Minuten und 20 Sekunden.

Tipp: Verwende den Modulo-Operator `%` und die Integer-Division `/`.

Arbeitsdauer aus Komm- und Gehzeit ermitteln

Anhand zweier Uhrzeiten im Format `hh:mm` soll die Arbeitszeit in Stunden und Minuten berechnet werden.

Das Programm ist wie folgt aufzurufen: `java MyApp hh:mm hh:mm`. Das erste Argument ist die Komm- und das zweite Argument die Gehzeit. Beispiel: `java MyApp 08:45 12:30` gibt `3h 45m` aus. Es darf davon ausgegangen werden, dass der Nutzer Argumente im korrekten Format angibt.

Das Program muss auch mit Tageswechseln umgehen können. Beispiel: `java MyApp 21:20 02:03` gibt `4h 43m` aus.

Wenn weniger als zwei Argumente angegeben werden, ist der Nutzer über die korrekte Verwendung des Programms zu informieren.

Tipp: Verwende `Integer.parseInt` zum Parsen. Die Uhrzeiten lassen sich zum Beispiel mit der Instanzmethode `split` der Klasse `String` zerlegen.