

M12: Objektorientierte Programmierung

WS 21/22 – Prof. Dr. Klaus Möllenhoff

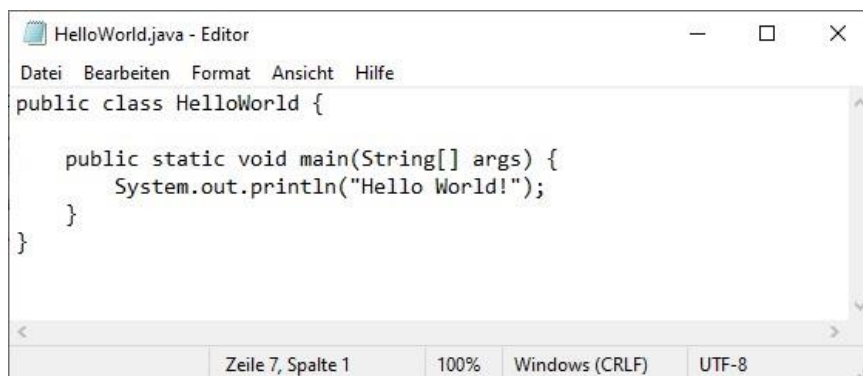
Übungsaufgaben

A-001

Beschreiben Sie, was mit Plattformunabhängigkeit gemeint ist und wodurch Java diese erreicht.

A-002

Tippen Sie das folgende Java-Programm ab und speichern es in einer Datei „HelloWorld.java“.



```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Führen Sie **an der Eingabeaufforderung** (cmd.exe) folgendes aus: Übersetzen Sie das Programm mit dem Aufruf „javac HelloWorld.java“ und beschreiben Sie, was dadurch passiert ist. Führen Sie es anschließend mit „java HelloWorld“ aus.

A-003

Erweitern Sie das „Hello World“-Programm derart, dass es unter der ersten Ausgabe zusätzlich in einer zweiten Zeile „Hallo HSBund!“ ausgibt. Führen Sie es erneut **an der Eingabeaufforderung** aus.

A-004

Ersetzen Sie im „Hello World-Programm“ die Implementierung der main-Methode durch folgendes Code-Stück:

```
String alter = "16";  
if (alter < 18) {  
    System.out.println("Werd erwachsen!");  
}
```

Versuchen Sie das Programm zur Ausführung zu bringen. Wieso kommt es zu einem Fehler? Handelt es sich um einen Compile- oder Laufzeitfehler?

A-005

Erstellen Sie in Ihrer IDE ein neues Java-Projekt mit Namen „hello-ide“. Fügen Sie dort eine Klasse HelloIntelliJ hinzu, die sie analog zu Aufgabe A-002 implementieren. Starten Sie das Programm und kopieren sie dessen Ausgabe in die Zwischenablage.

A-006

Erstellen Sie eine neue Klasse HelloDebugger mit folgendem Inhalt:

```
public class HelloDebugger {  
    public static void main(String[] args) {  
        int a = 6;  
        int b = 7;  
  
        // calculate the product  
        int product = 6 + 7;  
  
        // print the result  
        System.out.println(a + " * " + b + " = " + product);  
    }  
}
```

Das gegebene Programm enthält leider einen Bug, den Sie mittels Debugging finden sollen.

Erzeugen Sie hierzu einen Breakpoint in der ersten Zeile der main()-Methode und starten anschließend das Programm im Debugger. Machen Sie sich mit den Debugger-Funktionen vertraut, indem Sie zeilenweise durch die Methode schreiten und die jeweils aktuellen Variablenbelegungen inspizieren.

Sobald sie den Fehler gefunden haben, korrigieren Sie ihn und führen das Programm erneut aus.

A-007 (eingeschränkt prüfungsrelevant)

Prüfen Sie, welche der folgenden Zeichenketten in Java keine gültigen Bezeichner sind und erläutern Sie, weshalb.

- HelloWorld
- helloWörlD
- class
- _foo
- %foo
- X0
- xO
- frieda
- else
- \$
- three€
- final
- hu🤖hu
- hello-mom
- hello_mOm

A-008

Schreiben Sie ein ausführbares Java-Programm, welches in einer Klasse DeepThought die folgende Anweisung ausführt:

```
System.out.println(42);
```

Schreiben Sie das Programm aus dem Kopf und benutzen Sie keine Hilfsmittel.

A-009

Schreiben Sie ein Programm `TestOutput`, das die Zeichenkette „Hu hu!“ auf der Standardausgabe und die Zeichenkette „Oops!“ auf der Fehlerausgabe ausgibt.

Führen Sie das Programm zuerst an der Eingabeaufforderung und erst dann in Ihrer IDE aus. Vergleichen Sie die Ausgaben.

Führen Sie `TestOutput` an der Eingabeaufforderung aus und leiten Sie dessen Standardausgabe in eine Datei `out.txt` und die Fehlerausgabe in eine Datei `err.txt` um.

Hinweis: An der Eingabeaufforderung lassen sich die Standardausgabe und Fehlerausgabe separat in Dateien umleiten. Hierzu muss hinter dem Programmaufruf ein „>“ (Standardausgabe) bzw. „2>“ (Fehlerausgabe) gefolgt vom Namen der Ausgabedatei angegeben werden. Mit einem einzigen Aufruf können auch beide Ausgaben umgeleitet werden.

A-010

Machen Sie sich mit der Benutzung der API-Dokumentation der Java Class Library vertraut. Schlagen Sie dazu die Dokumentation der folgenden Elemente sowohl in der Online-Doku als auch in der Doku innerhalb Ihrer IDE nach:

- Klasse `System`
- Attribute `System.out` und `System.err`
- Methode `println()`

Ändern Sie nach Studieren der Dokumentation zu `println(String)` das HelloWorld-Programm derart ab, dass es in der ersten Anweisung lediglich die Zeichenkette „Hello“ ohne Zeilenumbruch ausgibt, und in einer zweiten Anweisung dann die Zeichenkette „world“.

Hinweise:

- Die Klasse `System` befindet sich im Standard-package „`java.lang`“ im Modul „`java.base`“.
- IntelliJ: Experimentieren Sie mit `Strg+Q` (einmaliges und mehrmaliges Drücken; Cursor dabei auf „passenden Elementen“)

A-011

Bennen Sie alle Zeilen, die zu Compiler-Fehlern führen:

```
public class VariableScoping {
    public static void main(String[] args) {
        int a = 6;
        {
            int b = 23;
            {
                int c = 42;
                System.out.println(a);
                System.out.println(b);
                System.out.println(c);
            }
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

A-012

Führen Sie die folgende Anweisung aus und erläutern Sie das möglicherweise überraschende Ergebnis.

```
System.out.println(2.0 - 1.1);
```

A-013

Führen Sie die folgende Anweisung aus und erläutern Sie das möglicherweise überraschende Ergebnis. Wie ließe sich der Fehler korrigieren?

```
System.out.println(3 * 1_000_000_000);
```

A-014

Geben Sie das Ergebnis und den Ergebnistyp der folgenden Ausdrücke an. Notieren Sie das Ergebnis dabei derart, dass es ein zulässiges Literal des entsprechenden Ergebnistyps ist. Die Variablen a, b, c sind vom Typ `int` und mit 42 belegt.

- $(3 / 4) + 7 * 5$
- $(1777 \% 2) + (1000 / 20 / 2)$
- $26 / 4. + 6$
- $26 / -4 + 6.0$
- $a++ \% 2$
- $--b \% 2 - b$
- $c++ / 3 + c$
- $27 \geq (42 / 3)$
- $27 != 63 - 36$

- `(4 > 3) && (27 < 42)`
- `(2 > 42) || true`
- `true ^ (42 < 2)`
- `!(27 < 42) ^ (true & (2 < 42))`
- `"ab" + "d"`
- `"ab" + 6 + 'd'`
- `"ab" + 42 / 2 + "d"`
- `"ab" + 6 + 36 + 'd'`
- `6 + 36 + "ab"`

A-015

Geben ist folgender Ausdruck:

```
42 > 42 + 160 / 8 / 2 ^ 42 != 42 && 27 > 6 || true != 4 < 4
```

Erläutern Sie, was an ihm problematisch ist und wie man das Problem vermeiden könnte.

A-016

Weisen Sie die folgenden Zahlen bzw. Berechnungen jeweils in einem einzelnen Ausdruck einer Variablen vom Typ `int` zu. Geben Sie anschließend beide Variablen aus und erläutern Sie, wie die Ergebnisse zustande kommen.

- `39 + (3.000.000.000 / 1.000.000.000)`
- Die Fläche einer Pizza mit 28 cm Durchmesser (möglichst genau in Quadratzentimeter)

Hinweis: Die Fläche eines Kreises mit Radius r berechnet sich zu $\pi * r^2$.

A-017

Deklarieren Sie eine Variable vom Typ `int` und initialisieren Sie diese mit einem beliebigen Wert. Genau dann wenn die Variable durch 3 teilbar ist, geben Sie „durch 3 teilbar“ aus. Im Anschluss geben Sie in jedem Fall aus: „Die Zahl war: <VARIABLENWERT>“.

Für <VARIABLENWERT> ist dabei der tatsächliche Wert unter Benutzung der Variablen einzusetzen.

A-018

Deklarieren Sie eine Variable vom Typ `int` und initialisieren Sie diese mit einem beliebigen Wert. Geben Sie dann entweder „liegt im Intervall“ oder „liegt außerhalb“ aus, abhängig davon, ob die Variable im Intervall `[0; 6]` liegt oder nicht. Im Anschluss geben Sie in jedem Fall aus: „Die Zahl war: <VARIABLENWERT>“.

Benutzen Sie eine `if-else`-Verzweigung.

A-019

Deklarieren Sie eine Variable `month` vom Typ `int` und initialisieren diese vorerst mit dem Wert 2. Weiter deklarieren Sie eine Variable `isLeapYear` zur Speicherung von Wahrheitswerten und initialisieren diese zunächst mit dem Wert für Wahr.

Berechnen Sie nun in einer Variable `days` die Anzahl der Tage des Monats - ausgehend von dem gegebenen Monat `month` im Intervall `[1; 12]` und der Information, ob wir uns momentan in einem Schaltjahr befinden (`isLeapYear`). Geben Sie anschließend die berechnete Anzahl aus.

- Benutzen Sie geschachtelte `if-else`-Verzweigungen.
- Benutzen Sie eine Mehrfachverzweigung (und ggf. weitere Verzweigungen). Was passiert, wenn Sie einzelne `break`-Anweisungen weglassen?

A-020

Versehen Sie Ihr Programm aus Übung A-019 a) in der IDE mit folgenden (mehr oder weniger) sinnvollen Kommentaren:

- Zeilenkommentare in eigener Zeile
- Zeilenkommentare in einer Zeile mit weiterem Programmcode
- Blockkommentare in eigenen Zeilen
- Kommentieren Sie den Teil zur Sonderbehandlung für Schaltjahre derart aus, dass der Februar immer 28 Tage besitzt. Fügen Sie dabei keine zusätzlichen Zeilenumbrüche ein.
- Kommentieren Sie zusätzlich die gesamte Sonderbehandlung für den Februar aus.

A-021

Zählen Sie in einer Schleife von 100 bis 40 in Zehnerschritten herunter und geben dabei jeden gezählten Wert aus.

- a) Benutzen Sie eine while-Schleife.
- b) Benutzen Sie eine do-while-Schleife.
- c) Benutzen Sie eine for-Schleife.

A-022

Implementieren Sie eine sogenannte „Endlosschleife“, die eine beliebige Ausgabe produziert.

- a) Benutzen Sie eine while-Schleife.
- b) Benutzen Sie eine do-while-Schleife.
- c) Benutzen Sie eine for-Schleife.

A-023

Geben Sie die 20 kleinsten Quadratzahlen aus. Benutzen Sie eine Schleife Ihrer Wahl.

A-024

Geben Sie in zehn Reihen das kleine Einmaleins aus. Das Ergebnis soll ungefähr so aussehen:

```
1: 1×1=1, 2×1=2, ... 10×1=10
2: 1×2=2, 2×2=4, ... 10×2=20
3: 1×3=3, 2×3=6, ... 10×3=30
...
```

A-025

Geben Sie alle Zahlenpaare (a, b) aus, für die gilt: $a + b$ ist Vielfaches von 40, $a, b \in [0; 100]$, $a \leq b$.

A-026

Implementieren Sie ein Programm, welches unentwegt Ganzzahlen vom Benutzer einließt und nach jeder eingegebenen Zahl die aktualisierte Summe ausgibt. Negative Zahlen sollen ignoriert werden. Bei Eingabe von 0 soll die Bearbeitung abbrechen. Benutzen Sie `continue`- und `break`-Anweisungen.

Hinweise:

- Eine Ganzzahl können Sie folgendermaßen einlesen:

```
int number = new java.util.Scanner(System.in).nextInt();
```
- Um nach Programmausführung in IntelliJ IDEA eine Eingabe tätigen zu können, muss das „Run“-Tab im unteren Bereich den Fokus haben (einmal reinklicken).

A-027

Deklariere Sie eine Array-Variable zur Speicherung von `double`-Werten und versuchen Sie diese anschließend auf der Standardausgabe auszugeben. Was ist das Ergebnis?

Initialisieren Sie diese Variable nun mit einem Array der Größe 7 und geben Sie sie anschließend aus. Wie sieht die Ausgabe aus?

Der Methodenaufruf `java.util.Arrays.toString(<ARRAY_VARIABLE>)` ist ein Ausdruck, der bei Auswertung eine String-Repräsentation des von der Variable referenzierten Arrays als Ergebnis liefert. Benutzen Sie diese Methode, um das Array auszugeben. Wie sieht jetzt die Ausgabe aus?

A-028

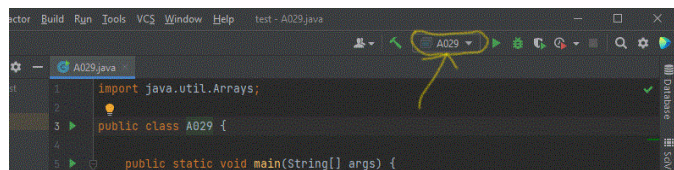
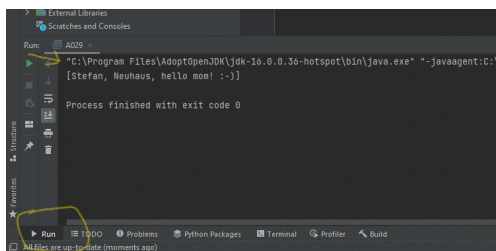
Deklariere Sie eine Array-Variable und initialisieren Sie das Array in derselben Anweisung mit den folgenden Werten: ihr Vorname, ihr Nachname, DACS-Kursbezeichnung.

Benutzen Sie dann eine Schleife, um alle Elemente des Arrays auszugeben.

A-029

Implementieren Sie die `main`-Methode derart, dass alle Programmargumente ausgegeben werden.

Starten Sie das Programm. Im „**Run Tool Window**“ im unteren Bildschirmbereich sehen Sie nun in der ersten Zeile die ausgeführte Kommandozeile zum Start der JVM. Kopieren Sie diese in eine leere Textdatei.



Beim Start des Programms wurde automatisch eine sogenannte „**Run/Debug Configuration**“ angelegt. Über diese können Sie zahlreiche Einstellungen für die Ausführung anpassen. U.a. finden Sie dort ein Texteingabefeld für die (Leerzeichen-separierten) „**Program arguments**“. Fügen Sie als Argumente Ihren Vor- und Nachnamen ein und führen Sie das Programm anschließend über diese Configuration aus. Vergleichen Sie die ausgeführte Kommandozeile des JVM-Starts mit der des vorherigen Starts. Worin besteht der Unterschied?

A-030

Erzeugen Sie ein Array der Größe 100 zur Speicherung von Gleitkommazahlen. Füllen Sie das Array dann mit Zufallszahlen aus dem Intervall `[0.0; 1.0[`.

Berechnen Sie nun den Mittelwert der Zufallszahlen und geben diesen aus.

- Benutzen Sie für die Mittelwertberechnung eine `for`-Schleife.
- Benutzen Sie für die Mittelwertberechnung eine `for-each`-Schleife.

A-031

Erzeugen Sie ein mehrdimensionales Array der Größe 20.000×10.000 zur Speicherung von Ganzzahlen. Füllen Sie das Array derart, dass an Stelle (i, j) das Produkt $i * j$ gespeichert wird.

Benutzen Sie dann dieses vorgefüllte Array, um den Wert von $12345 * 9876$ auszugeben.

Wieviel Hauptspeicher belegt das Array ungefähr?

Hinweis: Gehen Sie davon aus, dass die JVM das Array als einen zusammenhängenden Block im Speicher ablegt, in dem die Array-Werte alle direkt hintereinanderstehen.

A-032

Erzeugen Sie ein Array der Größe 40 namens `random` und füllen es mit zufälligen Ganzzahlen aus dem Intervall $[0; 100[$. Geben Sie das Array aus.

Sortieren Sie das Array und geben es erneut aus.

Erzeugen Sie ein weiteres Array mit den Werten 6, 17, 101, 23, 42 und prüfen Sie in einer Schleife für jeden dieser Werte, ob er im Array `random` vorkommt und geben eine entsprechende Meldung aus. Benutzen Sie dazu die Hilfsmethode `java.util.Arrays.binarySearch`. Machen Sie sich für die korrekte Benutzung zuvor mit der API-Dokumentation der Methode vertraut.

Erzeugen Sie folgende (Teil-)Kopien von `random`:

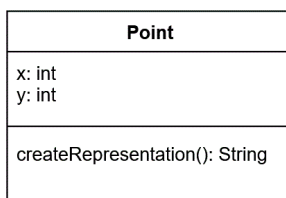
- Kopie der Referenz
- Kopie des Arrays
- Array der Größe 45: alle Werte aus `random`, Rest mit 0 aufgefüllt
- Array der Größe 10 mit den ersten 10 Zahlen aus `random`

Geben Sie die Kopien jeweils aus und prüfen mittels einer geeigneten Hilfsmethode, ob sie „gleich“ wie `random` sind.

A-033

Implementieren Sie eine Klasse für Punkte im zweidimensionalen Raum gemäß gegebenem Klassendiagramm. Die Methode `createRepresentation()` soll dabei eine Darstellung des Punktes in der üblichen Koordinatendarstellung liefern, z.B.: `"(6, 17)"`.

Schreiben Sie eine Main-Klasse, die zwei `Point`-Objekte erzeugt, deren Referenzen jeweils einer Variablen zuweist und die Darstellungen der beiden Punkte ausgibt.



A-034

Implementieren Sie eine Klasse für einen Taschenrechner, der die vier Grundrechenarten beherrscht. Die „Punkt vor Strich“-Regel kennt er jedoch nicht. Setzen Sie dazu das untenstehende Klassendiagramm um. Das Attribut `intermediateResult` soll dabei das Zwischenergebnis speichern, welches nach jeder Rechenoperation fortgeschrieben wird. Vor der ersten Benutzung soll der Taschenrechner den Wert 0 haben. Damit sinnvolle Ergebnisse berechnet werden können, muss dieses Zwischenergebnis gegebenenfalls initial auf einen Wert ungleich 0 gesetzt werden.

Schreiben Sie eine Main-Klasse, die mithilfe eines Taschenrechners folgende Berechnung durchführt und das Ergebnis ausgibt:

$$\left(\left((2 + 4) / 2 \right) * 17,5 \right) - 10,5$$

Calculator
intermediateResult: double
plus(double): void minus(double): void multiply(double): void divide(double): void solve(): double

A-035

Ändern Sie die Implementierung des Taschenrechners so ab, dass er sich für die Verwendung von *Method Chaining* eignet.

Berechnen Sie mit dem Taschenrechner unter Benutzung von Method Chaining das Ergebnis der Formel und geben Sie es aus:

$$\left(\left((2 + 4) / 2 \right) * 17,5 \right) - 10,5$$

A-036

Ändern Sie die Implementierung des Taschenrechners aus Aufgabe A-035 so ab, dass sich bei der Erzeugung eines Calculator-Objektes angeben lässt, mit welchem Startwert/Zwischenergebnis der Taschenrechner initialisiert werden soll. Implementieren Sie es so, dass die Angabe optional ist. Wenn kein Startwert angegeben wird, soll implizit vom Startwert 0 ausgegangen werden.

Erzeugen Sie zwei Taschenrechner um die folgenden beiden Formeln zu berechnen und auszugeben:

- $\left(\left((2 + 4) / 2 \right) * 17,5 \right) - 10,5$
- $\left(\left((0 - 4) * 3 \right) + 46 \right) / 2$

A-037

Schreiben Sie Programm, welches so lange wiederholt einen Text vom Benutzer abfragt, bis der eingegebene Text mindestens 10 Zeichen lang ist. Benutzen Sie dabei `javax.swing.JOptionPane.showInputDialog("Langer Text bitte: ")`, um den Text vom Benutzer einlesen. Machen Sie sich ggf. mit der API-Dokumentation vertraut.

Ein Abbruch/Schließen des Eingabe-Dialogfensters soll als unzureichend langer Text gewertet werden und darf nicht zum Programmabbruch führen.

Geben Sie nach Eingabe eines genügenden Textes die Länge und den Text selbst aus, z.B. in der Form:

Eingabe ist 27 Zeichen lang: Hallo Welt, hallo DACS! :-)

Hinweis: Die Klasse `String` bietet eine Methode `length()`.

A-038

Für ein Banken-Informationssystem soll eine Klasse `BankAccount` zur Repräsentation eines Bankkontos entwickelt werden. Ein Bankkonto besteht dabei aus folgenden Dingen:

- Eindeutige ID
- Kontoinhaber („*account holder*“)
- Kontostand („*balance*“)

Die Erzeugung von Konten soll mit folgenden Angaben möglich sein:

- Kontoinhaber und Kontostand
- nur Kontostand
- nur Kontoinhaber
- weder Kontoinhaber noch Kontostand

Bei fehlenden Angaben soll der Kontoinhaber "n.n." bzw. der Kontostand 0 EUR sein.

Mit der Erzeugung eines neuen Kontos soll automatisch eine zufällige ID für dieses Konto erzeugt werden und anschließend eine Meldung in folgender Form ausgegeben werden:

Konto 3a8fb0cb-f004-482d-84e5-72b2ebd18ff1 wurde für Klaus angelegt: 6.17 EUR

Wählen Sie passende Typen für die Attribute und zeichnen Sie ein Klassendiagramm für die Klasse.

Implementieren Sie die Klasse. Achten Sie dabei darauf, möglichst wenig duplizierten Code zu erstellen.

Hinweis: Die Klasse `java.util.UUID` wird typischerweise zur Repräsentation eines *universally unique identifier* (UUID) verwendet. Mittels `java.util.UUID.randomUUID()` lässt sich eine zufällige UUID erzeugen.

A-039

Schreiben Sie eine Methode, die ein Array von `int`-Zahlen als Parameter entgegennimmt und den größten Wert als Ergebnis zurückliefert. Sie dürfen davon ausgehen, dass das Array mindestens ein Element enthält.

A-040

Schreiben Sie eine Methode, die drei `int`-Zahlen als Parameter entgegennimmt und diese aufsteigend sortiert ausgibt.

A-041

Implementieren Sie den Euklid'schen Algorithmus zur Bestimmung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen p und q :

1. Bestimme den Rest r der ganzzahligen Division von p durch q .
2. Falls $r = 0$: q ist der gesuchte ggT
3. Sonst setze p gleich q , setze $q = r$ und gehe zu Schritt 1.

Testen Sie Ihre Implementierung:

- $ggT(1, 1) = 1$
- $ggT(17, 23) = 1$
- $ggT(24, 42) = 6$
- $ggT(42, 24) = 6$
- $ggT(1024, 320) = 64$

A-042

Schreiben Sie eine Methode `int crossSum(long number)`, die die Quersumme einer Zahl berechnet und als Ergebnis zurückliefert. Sie dürfen davon ausgehen, dass die übergebene Zahl nicht-negativ ist.

A-043

Die Fibonacci-Zahlen sind folgendermaßen rekursiv definiert:

- $fibonacci(0) = 0$
- $fibonacci(1) = 1$
- $fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)$, für $n \geq 2$

Implementieren Sie eine Methode `long fibonacci(int n)`, die die Fibonacci-Zahl zu einer gegebenen Zahl n berechnet und zurückgibt. Sie dürfen davon ausgehen, dass n nicht-negativ ist.

A-044

Schreiben Sie eine Methode `String decimalToBinary(int decimal)`, die eine Dezimalzahl als Eingabe bekommt die Binärdarstellung der Zahl als String zurückgibt.

Schreiben Sie eine Methode `String decimalToHex(int decimal)`, die eine Dezimalzahl als Eingabe bekommt die Hexadezimaldarstellung der Zahl als String zurückgibt.

Sie dürfen jeweils davon ausgehen, dass die Dezimalzahlen nicht-negativ sind.

A-045

Schreiben Sie ein Programm, um die folgende Fragestellung zu beantworten:

10000 Leute stehen durchnummeriert (beginnend mit 0) im Kreis. Nacheinander muss jeder Dritte gehen. Welche zwei Personen bleiben übrig?

Hinweis: Prüfen Sie die Korrektheit Ihrer Implementierung am Beispiel von 10 Personen.

A-046

Schreiben Sie eine Methode

```
double[][] createMatrix(int rows, int columns)
```

die eine Matrix mit entsprechend vielen Zeilen und Spalten erzeugt. Die Matrix soll Zufallszahlen aus dem Intervall $]-100; 100[$ enthalten.

Schreiben Sie zudem eine Methode, die die Daten des übergebenen Arrays zeilenweise ausgibt:

```
void printMatrix(double[][] matrix)
```

A-047

Das Sieb des Eratosthenes ist ein Algorithmus zur Bestimmung von Primzahlen. Der Algorithmus berechnet die Primzahlen bis zu einer vorzugebenden Obergrenze n :

1. Schreibe die Zahlen von 1 bis n auf.
2. Streiche die Zahl 1.
3. Für jede Zahl $2 \leq x \leq n$: Streiche alle Vielfachen von x , beginnend bei $2x$.
4. Alle Zahlen, die nach dem Streichen übrig bleiben sind Primzahlen.

Schreiben Sie ein Programm, welches alle Primzahlen von 1 bis 10.000 ausgibt.

Hinweis: Mit der Methode `java.util.Arrays.fill(boolean[] array, boolean value)` können Sie ein initialisiertes Array mit einem bestimmten Wert füllen.

A-048

Das Sortierverfahren Selection-Sort arbeitet nach folgendem Prinzip:

1. Der Index des kleinsten Elements wird bestimmt.
2. Dieses kleinste Element wird mit dem ersten Element vertauscht (d.h. das erste Element ist nun bereits das Kleinste).
3. Danach wird, beginnend mit dem 2. Element, wiederum das Minimum bestimmt und dieses Element mit dem 2. Element vertauscht (jetzt sind bereits die ersten zwei Elemente in der richtigen Reihenfolge).
4. Die Minimum-Suche und das Vertauschen werden solange sukzessive mit den folgenden Elementen wiederholt, bis das vorletzte Element erreicht wurde (dieses kann dann ggf. mit dem letzten Element vertauscht werden).

Implementieren Sie eine Methode `void selectionSort(int[] a)`, die das übergebene Array von int-Zahlen mittels Selection-Sort aufsteigend sortiert.

A-049

Implementieren Sie basierend auf den Methoden aus der vorherigen Aufgabe A-046 die folgenden Matrix-Operationen. Die Operationen sollen dabei die Eingaben nicht verändern, sondern immer eine neue Matrix mit dem Ergebnis liefern. Prüfen Sie jeweils, ob die Dimensionen der Matrizen zueinander passen. Geben Sie andernfalls eine Fehlermeldung aus.

Multiplikation einer Matrix mit einem Skalar

```
double[][] multMatrix(double[][] m, double x)
```

Addition zweier Matrizen

```
double[][] addMatrices(double[][] m1, double[][] m2)
```

Multiplikation zweier Matrizen

```
double[][] multMatrices(double[][] a, double[][] b)
```

Hinweis: Wenn a die Dimension $m \times n$ hat, muss b die Dimension $n \times p$ haben. Das Ergebnis c hat dann die Dimension $m \times p$. Die einzelnen Werte von c bilden sich folgendermaßen:

$$c_{ik} = \sum_{j=1}^m a_{ij} \cdot b_{jk}$$

Beispiel:

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 3 \cdot 1 + 2 \cdot 0 + 1 \cdot 4 & * \\ * & * \end{pmatrix} = \begin{pmatrix} 7 & * \\ * & * \end{pmatrix}$$

A-050

Das Pascal'sche Dreieck ist rekursiv definiert: Jede Zeile $z \geq 1$ enthält genau z ganze Zahlen. Die erste und die letzte Zahl einer Zeile ist immer 1. Die weiteren Zahlen (ab Zeile 3) ergeben sich aus der Summe der beiden Zahlen, die in der Zeile $z - 1$ über dem gesuchten Wert stehen:

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
```

Beispiel: Die 3. Zahl der 6. Zeile (10) ergibt sich aus der Summe der 2. (4) und 3. Zahl (6) der 5. Zeile.

Schreiben Sie ein Programm, das die ersten n Zeilen des Pascal'schen Dreiecks in einem zweidimensionalen int-Array berechnet und ausgibt.

Hinweis: Die Ausgabe muss nicht die passend ausgerichtete Dreiecksform haben. Eine Ausgabe für $n = 7$ könnte beispielsweise folgendermaßen aussehen:

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
```

A-051

Schreiben Sie eine Methode `String nameOfNumber(int number)`, die das Zahlwort der übergebenen int-Zahl zurückliefert.

Beispiel: 1234567 → „einemillionzweihundertvierunddreißigtausendfünfhundertsiebenundsechzig“

Hinweise:

- Zerlegen Sie die Aufgabe in Teile, z.B. Zahlen bis 9 ausgeben, Zahlen bis 99 ausgeben, Zahlen bis 999 ausgeben etc. und verwenden Sie diese Teile, um möglichst wenig doppelt zu programmieren.
- Die korrekte Deklination des Zahlwortes „ein“ nach Genus können Sie ignorieren.

A-052

Entwerfen und implementieren Sie eine Klasse `DynamicArray`, die eine Array-basierte Datenstruktur mit automatischer Größenanpassung zur Speicherung von long-Werten umsetzt. Das dynamische Array soll die folgenden Operationen bieten:

- Erzeugen eines leeren Arrays (mit initial Platz für eine sinnvolle Anzahl an Elementen)
- Erzeugen eines leeren Arrays mit initial Platz für n Elemente
- Zurückliefern der Anzahl der gespeicherten Elemente
- Zurückliefern einer String-Repräsentation der Elemente
- Anhängen des Elements x am Ende
- Entfernen des i -ten Elements (falls bereits mindestens i Werte gespeichert sind)
- Zurückliefern des i -ten gespeicherten Elements
- Setzen des i -ten Elements auf den Wert x (falls bereits mindestens i Werte gespeichert sind)

A-053

Nehmen Sie die Klasse `BankAccount` aus Aufgabe A-038 als Grundlage:

```
class BankAccount {  
    java.util.UUID id;  
    String accountHolder;  
    long balance;  
  
    BankAccount() {  
        // CTOR chaining: rely on more specific CTOR  
        this("n.n.");  
    }  
  
    BankAccount(String accountHolder) {  
        // CTOR chaining: rely on more specific CTOR  
        this(accountHolder, 0);  
    }  
  
    BankAccount(long balance) {  
        // CTOR chaining: rely on more specific CTOR  
        this("n.n.", balance);  
    }  
  
    // the most specific CTOR implements all the initialization code  
    BankAccount(String accountHolder, long balance) {  
        id = java.util.UUID.randomUUID();  
        this.accountHolder = accountHolder;  
        this.balance = balance;  
    }  
}
```

Fügen Sie der Klasse die folgenden Methoden hinzu:

- `deposit`: erhöht den Kontostand um einen bestimmten Betrag
- `withdraw`: hebt einen Betrag vom Konto ab

Testen Sie die Klasse mit einem Programm namens `BankAccountTest`. Das Programm soll ein Konto mit einem Kontostand von 1000 EUR anlegen. Zur Kontrolle soll der Kontostand ausgegeben werden. Anschließend sollen 500 EUR eingezahlt und 1750,50 EUR ausgezahlt werden. Zur Kontrolle wird danach nochmals der Kontostand ausgegeben.

A-054

Schreiben Sie zur Klasse `BankAccount` aus der vorherigen Übung eine Unterklasse `BankAccountWithLimit` für Girokonten, die über ein zusätzliches Attribut `limit` verfügt, welches den Kreditrahmen repräsentiert, der Kunden für das Überziehen zur Verfügung steht.

Passen Sie die `withdraw`-Methode derart an, dass im Fall von Girokonten vor der Auszahlung geprüft wird, ob die geforderte Auszahlung ohne Überschreitung des Kreditlimits möglich ist. Falls dies nicht möglich ist, soll eine Fehlermeldung ausgegeben und die Auszahlung verweigert werden.

Die Klasse soll über einen Konstruktor mit zwei Parametern für den Besitzer und den Kreditrahmen verfügen.

Schreiben Sie ein Programm `BankAccountWithLimitTest`, welches die Funktionalität testet.

Zeichnen Sie ein Klassendiagramm für die Klassen `BankAccount` und `BankAccountWithLimit`.

Hinweis: Implementieren Sie die Prüfung der `withdraw`-Methode innerhalb der Klasse `BankAccount` unter Benutzung von Typprüfung und Typumwandlung.

A-055

Erzeugen Sie ein Paket namens `de.hsbund.dacs.oop.a055.bank` und „kopieren“ Sie die Klassen `BankAccount` und `BankAccountWithLimit` aus Aufgabe A-054 in dieses Paket.

Um den Zugriff von außerhalb des Pakets zu ermöglichen, müssen wir die Klassen und die Konstruktoren, sowie ggf. die Attribute und Methoden noch „public machen“. Schreiben Sie dazu das Schlüsselwort `public` jeweils vor Schlüsselwort `class`, die Konstruktoren, die Methoden- und die Attributdeklarationen (mehr zum Thema *Sichtbarkeit* später).

Legen Sie eine Main-Klasse im Paket `de.hsbund.dacs.oop.a055` an. Wie dies geht, ist möglicherweise nicht ganz offensichtlich. Es gibt verschiedene Wege, versuchen Sie kreativ eine Lösung zu finden.

Implementieren Sie die `main`-Methode derart, dass ein `BankAccount`- und ein `BankAccountWithLimit`-Objekt angelegt werden.

A-056

Stellen Sie sicher, dass die drei Klassen aus der vorhergehenden Aufgabe mit Imports arbeiten und keine voll qualifizierten Klassennamen verwenden. Überarbeiten Sie Ihre Klassen entsprechend (sofern Ihre IDE dies nicht von vorneherein entsprechend angelegt hat).

Bei der Wahl des Paketnamens ist leider ein Fehler passiert. Das `bank`-Paket sollte eigentlich `de.hsbund.dacs.oop.a056.bank.domain` heißen.

Korrigieren Sie den Fehler auf möglichst elegante Art mit der IDE. Welche Änderungen müssen dazu (manuell oder automatisch durch die IDE) durchgeführt werden?

A-057

Die bisherige Kontenimplementierung funktioniert zwar grundsätzlich, sie enthält aber einige Punkte, die als schlechter Stil gelten und meist vermeidbar sind:

- Die Methode `BankAccount.withdraw()` verwendet Typumwandlungen (*Casts*), um die Sonderbehandlung für Girokonten umzusetzen.
- Die Klassen `BankAccount` und `BankAccountWithLimit` verstoßen gegen das Geheimnisprinzip.
- Die Klasse `TestBankAccounts` enthält duplizierten Code zur formatierten Ausgabe des Kontostandes. Passen Sie die Implementierung derart an, dass sich der Kontostand über die Anweisung `System.out.println(account)` in der bekannten Form ausgeben lässt.

Beispiel: `-1250.5 EUR`

Beheben Sie die oben aufgeführten Kritikpunkte.

Zeichnen Sie das Klassendiagramm inklusive Sichtbarkeiten.

A-058

Implementieren Sie eine Klasse `Exponentiation` zur Repräsentation von Potenzen. Eine Potenz besteht aus einer Basis `base` und einem Exponenten `power`, die beide reellwertig sein sollen. Die Anzahl der von der Klasse erzeugten Objekte sollen mitgezählt werden.

Beispiel: Die Potenz $1764^{0,5}$ würde repräsentiert durch ein Objekt, dessen Basis den Wert 1764 und dessen Exponent den Wert 0.5 besitzt.

A-059

Ziel ist die Implementierung einer Klasse `Utilities`, die im Laufe der Zeit um diverse Hilfsmethoden angereichert werden soll. Da es sich um Hilfsmethoden handelt, benötigt die Klasse selbst keinen Zustand.

Implementieren Sie eine Hilfsmethode namens `lengthOf(...)`, die zu einem übergebenen String dessen Länge zurückliefert. Eine übergebene null-Referenz soll als ein String der Länge 0 gelten.

Implementieren Sie eine Hilfsmethode namens `isBlankString(...)`, die zu einem übergebenen String zurückliefert, ob dieser die Länge 0 hat oder ausschließlich aus Whitespace besteht (d.h. Leerzeichen, Tabs etc.). Eine übergebene null-Referenz soll als *blank* gelten.

Stellen Sie sicher, dass Aufrufer nicht irrtümlicherweise eine Instanz der Klasse erzeugen können.

Testen Sie Ihre Implementierung.

Stellen Sie die Klasse in einem UML-Klassendiagramm dar.

Hinweise:

- Die Objektmethode `String.length()` liefert die Länge eines String-Objektes zurück.
- Die Objektmethode `String.trim()` liefert von einem String-Objekt als Ergebnis den String, der durch Entfernung von führendem und abschließendem Whitespace entsteht.

Beispiel: `" hello world! ".trim()` liefert `"hello world!"`.

A-060

Für ein Computerspiel entwickeln Sie eine Komponente, welche die Richtungsänderungen der Spielfigur protokollieren soll. Die Spielfigur kann sich hoch, runter, nach links oder rechts bewegen.

Implementieren Sie einen Aufzählungstyp `Direction` für die möglichen Richtungen.

Implementieren Sie außerdem die Klasse `MovementLog`, welche vorerst nur über eine Methode `public void logMovement(...)` verfügt. Die Methode verfügt über einen einzigen Parameter, nämlich eine konkrete Richtung. Die Methode soll für den übergebenen Parameter die entsprechende Richtung auf der Standardausgabe ausgeben.

Beispiel: `neue Richtung: hoch`

A-061

Benutzen Sie für ein Werkstatt-Informationssystem die `Vehicle`-Klasse aus Übung B-017, die bereits Argumentprüfungen in den Settern umsetzt. Implementieren Sie dann eine Klasse `Garage` gemäß folgendem Klassendiagramm:

Garage
+ Garage()
+ replaceEngine(vehicle: Vehicle, mileage: int, model: string): void

Die Methode `replaceEngine()` setzt das Austauschen des Motors um. Nach dem Austausch soll die Kilometerleistung auf den neuen übergebenen Wert gesetzt werden. Da der Austausch auch Auswirkungen auf die Modellbezeichnung haben kann, soll diese auch auf den neuen Wert gesetzt werden. Benutzen Sie dazu jeweils die Setter der `Vehicle`-Klasse.

Die Methode `replaceEngine()` soll Fehler beim Setzen der neuen Attributwerte erkennen und dem Aufrufer in Form einer `Checked Exception` werfen. Benutzen Sie dazu die Klasse `GarageException` aus Übung B-018. Damit die `Exception` für die Fehleranalyse möglichst wertvoll ist, soll sie neben einer sprechenden Fehlerbeschreibung auch den Grund der `Exception` enthalten.