252-0027

Einführung in die Programmierung

Übungen

Woche 3: Einführung Java

Jonas Wetzel
Departement Informatik
ETH Zürich

Organisatorisches

- Mein Name: Jonas Wetzel
- Bei Fragen: jwetzel@ethz.ch
- **Mails bitte mit «[EProg24]» im Betreff**
- Neue Aufgaben: **Dienstag Abend** (im Normalfall)
- Abgabe der Übungen bis **Dienstag Abend (23:59)** Folgewoche
  - Abgabe immer via Git
  - Lösungen in separatem Projekt auf Git

- **Wir haben eine WhatsApp Gruppe**
- **Meine website: n.ethz.ch/~jwetzel**

Plan heute

- Nachbesprechung erste Übung
- Theorie
- Alte Prüfungsaufgaben
- Tipps für die nächste Übung
- Kahoot
- Outro

Nachbesprechung Übung 1

Weitere Fragen zu Eclipse oder Git?

Lösung Übung 1, Aufgabe 2

```
/*
 * Author: Maximiliana Muster
 * für Einführung in die Programmierung
 */
public class HelloProgrammer {

    public static void main(String[] args) {
        System.out.println("Hello, my name is Max!");
    }
}
```

Lösung Übung 1, Aufgabe 4

1. Erstellen Sie eine Beschreibung `<geradezahl>`, die als legale Symbole alle geraden Zahlen (d.h. Zahlen, die ohne Rest durch 2 teilbar sind) zulässt. Beispiele sind +02, 4, 10, −20.

Lösung Übung 1, Aufgabe 4

2. Zeigen Sie in einer Tabelle, dass Ihre Beschreibung das Symbol “28” als gerade Zahl erkennt.

Lösung Übung 1, Aufgabe 4

3. Erstellen Sie eine Beschreibung `<x2ygemischt>`, die als legale Symbole genau jene Wörter zulässt, in denen für jedes "`X`" zwei "`Y`" als Paar auftreten. Beispiele sind `XYY`, `YYX`, `XYYYYX`, `XXYYYY`. Die Beschreibung schliesst den leeren String nicht aus.

Lösung Übung 1, Aufgabe 4

Die folgenden EBNF-Beschreibungen sind nicht äquivalent. Finden Sie ein *kürzestmögliches* Symbol, das von der einen Beschreibung als legal erkannt wird, aber nicht von der anderen. (Fangen Sie mit einfachen Kombinationen von A und B an.)

EBNF-Beschreibung: `<beispiel1>`

`<beispiel1>` ⇐ `[A][B]`

EBNF-Beschreibung: `<beispiel2>`

`<beispiel2>` ⇐ `[A[B]]`

Lösung Übung 1, Aufgabe 4

5. Erstellen Sie eine EBNF Beschreibung `<doppelt>`, die als legale Symbole genau jene Wörter zulässt, in denen die doppelte Anzahl "`Y`" nach einer Folge von "`X`" auftritt. Beispiele sind `XYY`, `XXYYYY`, usw. Die Beschreibung schliesst den leeren String nicht aus.

Theorie

Aufbau einer Java-Klasse

```
public class ClassName {

    public static void main(String[] args) {
        // Anweisungen, die beim Starten des Programms ausgeführt
werden
        System.out.println("Hallo, Welt!");
        meineMethode();
    }


    public static void meineMethode() {
        System.out.println("Dies ist eine Methode!");
    }
```

Methoden in Java

Wieso?

1. Redundanz vermeiden
2. Lesbarkeit verbessern
3. Testbarkeit einzelner Funktionen → verbesserte Fehlersuche
4. …

Intuition: Jede Methode löst ein Teilproblem.

```java
public class LineareAusführung {

    public static void main(String[] args) {
        method1();
        method6();
        method4();
    }

    public static void method1() {
        method3();
        System.out.println("method1");
    }

    public static void method2() {
        method6();
        System.out.println("method2");
    }

    public static void method3() {
        System.out.print("3");
        System.out.println("method3");
    }
```

Was wird ausgegeben?

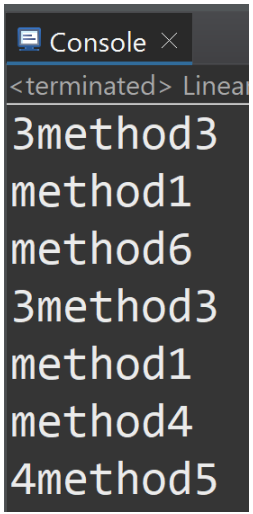
```java
public static void method4() {
    System.out.println("method4");
    System.out.print("4");
    method5();
}

public static void method5() {
    System.out.println("method5");
}

public static void method6() {
    System.out.println("method6");
    method1();
}
```

```java
public class LineareAusführung {

    public static void main(String[] args) {
        method1();
        method6();
        method4();
    }

    public static void method1() {
        method3();
        System.out.println("method1");
    }

    public static void method2() {
        method6();
        System.out.println("method2");
    }

    public static void method3() {
        System.out.print("3");
        System.out.println("method3");
    }
```

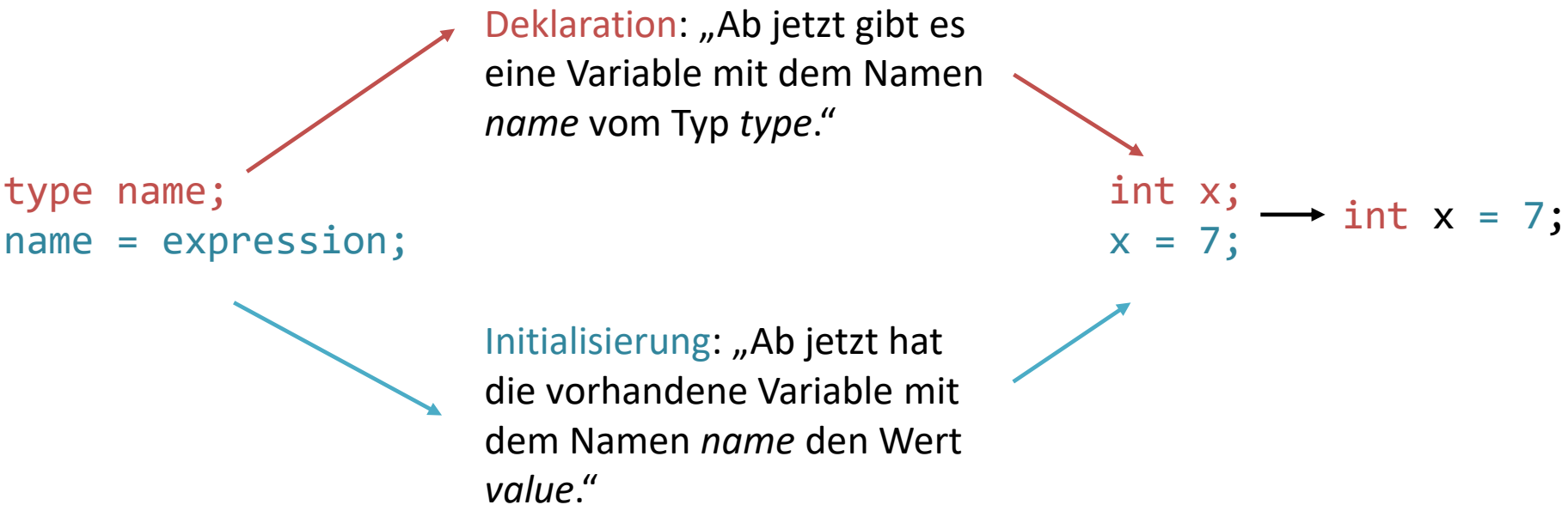
```java
public static void method4() {
    System.out.println("method4");
    System.out.print("4");
    method5();
}

public static void method5() {
    System.out.println("method5");
}

public static void method6() {
    System.out.println("method6");
    method1();
}
```

Was wird ausgegeben?

Output:



Variablen in Java



Keine Standard-Initialisierung in Java

Primitive („einfache“) Datentypen in Java

Name	Beschreibung	Beispiele
`int`	ganze Zahlen	`-2147483648, -3, 0, 42, 2147483647`
`long`	grosse ganze Zahlen	`42, -3, 0, 9223372036854775807`
`double`	reelle Zahlen	`3.1, -0.25, 9.4e3`
`char`	(einzelne) Buchstaben	`'a', 'X', '?', '\n'`
`boolean`	Wahrheitswerte	`true, false`

Was passiert bei 2147483647+1?

Primitive („einfache“) Datentypen in Java

Name	Beschreibung	Beispiele
`int`	ganze Zahlen	`-2147483648, -3, 0, 42, 2147483647`
`long`	grosse ganze Zahlen	`42, -3, 0, 92233720368547758O7`
`double`	reelle Zahlen	`3.1, -0.25, 9.4e3`
`char`	(einzelne) Buchstaben	`'a', 'X', '?', '\n'`
`boolean`	Wahrheitswerte	`true, false`

Was passiert bei 2147483647+1? → Overflow

Arithmetische Operatoren

Operator	Beschreibung	Kurzbeispiel
+	Addition	`int antwort = 40 + 2;`
-	Subtraktion	`int antwort = 48 - 6;`
*	Multiplikation	`int antwort = 2 * 21;`
/	Division	`int antwort = 84 / 2;`
%	Teilerrest, Modulo-Operation, errechnet den Rest einer Division	`int antwort = 99 % 57;`
+	positives Vorzeichen	`int j = +3;`
-	negatives Vorzeichen	`int minusJ = -j;`

Was passiert bei 42 / 0?
Was passiert bei 42 / 0.0?

Arithmetische Operatoren

Operator	Beschreibung	Kurzbeispiel
+	Addition	`int antwort = 40 + 2;`
-	Subtraktion	`int antwort = 48 - 6;`
*	Multiplikation	`int antwort = 2 * 21;`
/	Division	`int antwort = 84 / 2;`
%	Teilerrest, Modulo-Operation, errechnet den Rest einer Division	`int antwort = 99 % 57;`
+	positives Vorzeichen	`int j = +3;`
-	negatives Vorzeichen	`int minusJ = -j;`

Was passiert bei 42 / 0? → Error
Was passiert bei 42 / 0.0? → Infinity

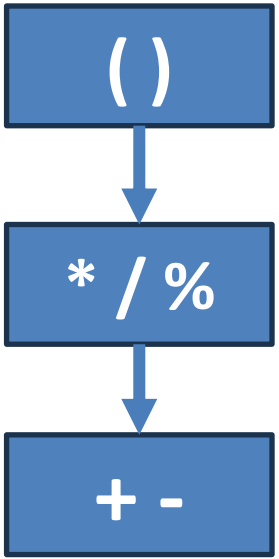
Assoziativität

- **Die Assoziativität («associativity») eines Operators bestimmt, wie ein Operand verwendet wird**

- **Ist ein Operator ⊗**
  - **Linksassoziativ** («**left-associative**») → $X \otimes Y \otimes Z$ gleich $(X \otimes Y) \otimes Z$
  - **Rechtsassoziativ** («**right-associative**») → $X \otimes Y \otimes Z$ gleich $X \otimes (Y \otimes Z)$

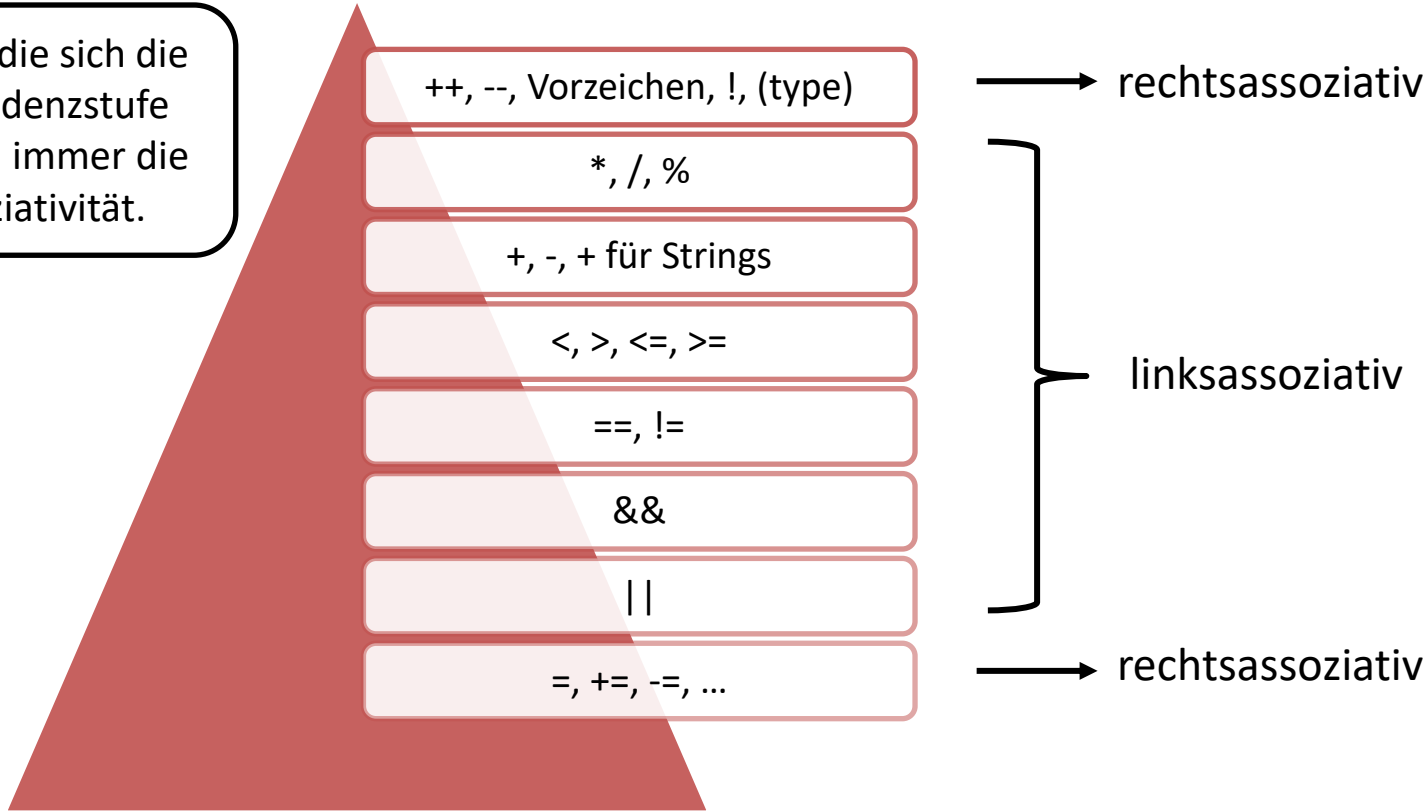
Auswertung – Ordnung

- Operand wird vom Operator mit höherer Rang Ordnung (“precedence”) verwendet
- Wenn zwei Operatoren die selbe Rang Ordnung haben, dann entscheidet die Assoziativität
- Wenn zwei Operatoren die selbe Rang Ordnung und Assoziativität haben, dann werden die (Teil)Ausdrücke von links nach rechts ausgewertet.
- Wenn etwas anderes gewünscht wird: Klammern verwenden!



Auswertung - detaillierter

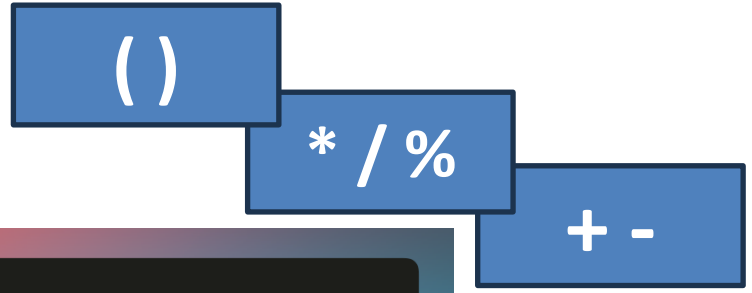
Operatoren, die sich die gleiche Präzedenzstufe teilen, haben immer die gleiche Assoziativität.



Assoziativitätskonflikte?

- In Java gibt es keine Situation, in der zwei Operatoren mit derselben Präzedenz (precedence) unterschiedliche Assoziativitäten (d.h. einer ist linksassoziativ und der andere rechtsassoziativ) haben.
- Und die Moral von der Geschichte: Wenn man sich nicht sicher ist, setzt man einfach Klammern.

Auswertung – Beispiele




```java
public class Main {
    public static void main(String[] args) {
        System.out.println(2/3+4.5/3);
    }
}
```

Output: 1.5

Auswertung – Beispiele


```java
public class Main {
    public static void main(String[] args) {
        System.out.println(8%3+2.5+4/3);
    }
}
```

Output: 5.5

Auswertung – Casting


```java
public class Main {
    public static void main(String[] args) {
        System.out.println((double)5/2);
    }
}
```

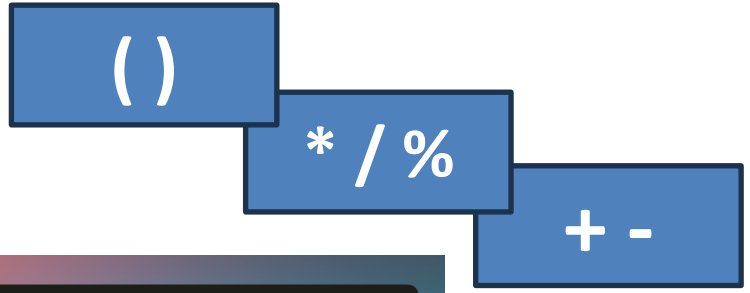
Output: 2.5

Auswertung – String


```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Wir schreiben das Jahr "+2023/2*2.0);
    }
}
```

Output: Wir schreiben das Jahr 2022.0

Auswertung – Prüfungsbeispiel




```java
public class Main {
    public static void main(String[] args) {
        System.out.println(12/4*3+("X"+7%3)+18%5*2);
    }
}
```

Output: 9X16

Auswertung – Prüfungsbeispiel


```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Wir schreiben das Jahr " + 2000 + 24);
    }
}
```

Output: Wir schreiben das Jahr 200024

Aufgabe 1

```java
class MyClass{
    public static void main(String[] args){
        System.out.println(9 / 3 * 2 + 1 + "" + 2 * 3 + (4 * 3) % 3);
    }
}
```

760

Aufgabe 2

```java
class MyClass{
    public static void main(String[] args){
        System.out.println(8 / 5 + 0.5 + 5 / 2 + (8 % 3) * 1.0);
    }
}
```

5.5

Kurzschlussauswertung («short-circuit evaluation»)

Bei Berechnung von `p && q` und `p || q`

- Java wertet zuerst den linken Operanden `(p)` und dann den rechten `(q)` aus
- die Auswertung wird beendet, sobald das Ergebnis feststeht.
  - Bei `p && q`: Falls `p == false`, dann ist `p && q == false`
  - Bei `p || q`: Falls `p == true`, dann ist `p || q == true`

p	q	p && q
true	true	true
true	false	false
false	true	false
false	false	false

p	q	p || q
true	true	true
true	false	true
false	true	true
false	false	false

In diesen Fällen ist keine Auswertung von q nötig!

Prüfungsaufgaben

Rechnung - Prüfungsaufgabe

1. `System.out.println(36 / (3 * 12     4));`

   `//Output:  4`

2. `System.out.println(64 % (4      8 + 1) / 4);`

   `//Output: 7`

3. `System.out.println(32    (2     6));`

   `//Output -8`

Bestimmen Sie für jede Anweisung die fehlenden Operatoren so, das die Anweisung die gezeigte Ausgabe erzeugt. Mögliche Operatoren sind +, -, *, / und %.

Rechnung - Prüfungsaufgabe

```
1. System.out.println(36 / (3 * 12  /  4));

   //Output:  4


2. System.out.println(64 % (4  *  8 + 1) / 4);

   //Output: 7


3. System.out.println(32  / (2  -  6));

   //Output -8
```

Bestimmen Sie für jede Anweisung die fehlenden Operatoren so, das die Anweisung die gezeigte Ausgabe erzeugt. Mögliche Operatoren sind +, -, *, / und %.

- System.out.println(11 + 16 / 4 * 2 + (4 + ">") + 4 * 2);

Solution

- System.out.println(11 + 16 / 4 * 2 + (4 + ">") + 4 * 2);
- 194>8

- First, 11 + 16 / 4 * 2 evaluates to 19. (4 + ">") evaluates to 4>, and afterwards 4*2 (so 8) must be concatenated, since 4> is not an integer.

- System.out.println(20 / 10 % 6 + 3 / 2 + (double) 5 / 4 + 3 / 4);

Solution

- System.out.println(20 / 10 % 6 + 3 / 2 + (double) 5 / 4 + 3 / 4);
- 4.25

- 20/10 = 2 ... %6 **= 2**
- + 3/2 = 1 **= 3**
- + 5.0/4 = 1.25 **= 4.25**
- +3/4 = 0 **= 4.25**

- System.out.println((11 % 4) > 2 && 9 > (16 / (2 / 4)) && 1 % 8 < 0);
- Achtung, tricky. Teilen durch 0 geht nicht!

- System.out.println((11 % 4) > 2 && 9 > (16 / (2 / 4)) && 1 % 8 < 0);
- Achtung, tricky. Teilen durch 0 geht nicht.

- Das würde einen Fehler produzieren, da wir nicht durch 0 teilen können. In Java gebe es eine sogenannte Runtime Exception, also während das Programm gelaufen ist, ist etwas schiefgelaufen, nämlich DivideByZero

- System.out.println((5 + 4) / 2 * (8 / (4 * 4) + 2.0));

Solution

- System.out.println((5 + 4) / 2 * (8 / (4 * 4) + 2.0));

- 8.0
- ((5 + 4) / 2 = 4 und (8 / (4 * 4) + 2.0) = 2.0))

- System.out.println(1452 % 10 + " = " + 1 + 4 / 5 + "<" + 2 * 2);

Solution

- System.out.println(1452 % 10 + " = " + 1 + 4 / 5 + “<" + 2 * 2);

- 2 = 10 < 4

- Denke daran, dass nach der ersten Zeichenkette ' = ' die weiteren + ebenfalls eine Verkettung bedeuten und keine Addition.

- System.out.println((4 * 4) == 16.0 || (5 / (6 / 7)) < 1);

Solution

- System.out.println((4 * 4) == 16.0 || (5 / (6 / 7)) < 1);

- true
- Der zweite Teil würde eine Division durch Null enthalten, aber da (4 * 4) == 16.0 wahr ist, wird dieser Teil überhaupt nicht ausgewertet.

Kahoot

Vorbesprechung Übung 2

Aufgabe 1: Fehlerbehebung

Finden und beheben Sie alle Fehler im Programm “FollerVehler.java”. Eclipse hilft Ihnen dabei, indem es anzeigt, wo die Fehler sind (und eine mehr oder weniger hilfreiche Fehlermeldung dazu ausgibt), aber Sie müssen selber herausfinden, was das Problem ist und wie Sie es beheben können. Wenn Sie alle Fehler behoben haben, sollte das Programm folgendes ausgeben:

```
Hello world
Gefällt Ihnen dieses Programm?

Ich habe es selbst geschrieben.
```

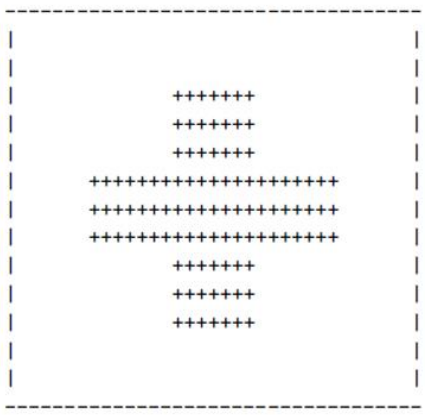
Aufgabe 2: Schweizerfahne

Statt eine Vorlage zu benützen, schreiben Sie in dieser Aufgabe ein Programm von Grund auf selbst. Eclipse wird Ihnen allerdings etwas Schreibarbeit abnehmen.

1. Erstellen Sie eine neue Java-Datei "SwissFlag.java". Wählen Sie dafür im Menü *File* → *New* → *Class* oder klicken Sie auf das Symbol in der Symbolleiste. Geben Sie im Dialog bei *Name* "SwissFlag" ein und drücken Sie "Finish".

2. Erstellen Sie zuerst eine leere `main`-Methode, so wie Sie es bei anderen Programmen gesehen haben. (Tipp: Sie können in Zukunft auch die Option *public static void main(String[] args)* im Dialog für neue Java-Klassen auswählen, um sich ein wenig Arbeit zu sparen.)

3. Erweitern Sie das Programm so, dass es die Schweizerfahne in der Konsole ausgibt. Die Fahne könnte ungefähr wie folgt aussehen, Sie dürfen aber auch eine grössere oder schönere Version entwerfen:



Teilen Sie das Programm in mehrere Methoden auf, welche von der `main`-Methode aufgerufen werden. Damit sorgen Sie dafür, dass weniger Wiederholungen von Code-Stücken vorkommen, was das Ändern des Programms deutlich einfacher macht.

Aufgabe 3: Eingabe und Zufall

Aufgabe 3: Eingabe und Zufall

In dieser Aufgabe arbeiten Sie mit der Eingabe und Ausgabe von Java und lernen die Klassen `Scanner` und `Random` näher kennen.

1. Schreiben Sie ein Programm "Adder.java", welches zwei ganze Zahlen einliest und die Summe davon ausgibt. Sie sollen dafür die `Scanner`-Klasse verwenden, wie in der Vorlesung gezeigt. Das Programm soll nach der ersten Zahl fragen:

   ```
   Geben Sie Zahl 1 ein:
   ```

   dann nach der zweiten Zahl:

   ```
   Geben Sie Zahl 2 ein:
   ```

   und schliesslich, wenn Sie zum Beispiel "4" und "1999" eingeben, folgendes ausgeben:

   ```
   4 + 1999 = 2003
   ```

   Sie können davon ausgehen, dass nur ganze Zahlen eingegeben werden. Testen Sie das Programm mit verschiedenen Zahlen.

2. Schreiben Sie ein Programm `Wuerfel.java`, das einen Würfel simuliert. Hierbei soll eine positive ganze Zahl `N` eingelesen werden, welche die Anzahl der Seiten des Würfels repräsentiert. Der übliche Würfel hat 6 Seiten, jedoch existieren auch Würfel mit 12, 16, 20 (siehe Abbildung 1) und mehr Seiten. Jede Seite trägt eine unterschiedliche Zahl, die von 1 bis `N` (inklusive `N`) reicht.

   Das Programm soll den Wurf simulieren, indem es die Klasse `Random` verwendet. Ein möglicher Ablauf des Programmes könnte folgendermassen aussehen:

   ```
   Wie viele Seiten hat Ihr Würfel?
   ```

   Der Benutzer gibt eine Zahl ein, z.B. 20, danach wird gewürfelt:

   ```
   Es wurde eine 17 gewürfelt!
   ```

Einlesen mit Scanner

- **Code Beispiel: scanner und random**

3. Schreiben Sie ein Programm `ChatGTP.java`, welches die Interaktion mit einem AI-Chatbot simuliert. Das Programm soll den Benutzer begrüssen, Sie nach Ihrem Namen und Alter fragen, und dem Benutzer Ihre Glückszahl verraten. Die Glückszahl soll zufällig gewählt werden, in dem Sie die `Random` Klasse benutzen. Ein möglicher Ablauf des Programmes könnte folgendermassen aussegen:

```
Guten Tag!  Ich bin ChatGTP, der beste Chatbot, denn es gibt!  Wie heissen Sie?
```

Ein Namen wird eingegeben, z.B. Alice, danach antwortet der Chatbot auf diesen Input:

```
Sehr erfreut Alice!  Wie alt sind Sie?
```

Ein Alter wird eingegeben, z.B. 23, danach antwortet der Chatbot auf diesen Input:

```
Mittels dieser Informationen habe ich Ihre Glückszahl gefunden!  Die Glückszahl
lautet 42.
```

Aufgabe 4: Berechnung

2. Vervollständigen Sie "SharedDigit.java". In der Main-Methode sind zwei int Variablen `a` und `b` deklariert und mit einem Wert zwischen 10 und 99 (einschliesslich) initialisiert. Das Programm soll einer int Variablen `r` einen bestimmten Wert zuweisen. Wenn `a` und `b` eine Ziffer gemeinsam haben, dann wird `r` die gemeinsame Ziffer zugewiesen (wenn `a` und `b` beide Ziffern gemeinsam haben, dann kann eine beliebige Ziffer zugewiesen werden). Wenn es keine gemeinsame Ziffer gibt, dann soll -1 zugewiesen. Sie brauchen für dieses Programm keine Schleife.

   Beispiele:

   - Wenn `a`: 34 und `b`: 53, dann ist `r`: 3
   - Wenn `a`: 10 und `b`: 22, dann ist `r`: -1
   - Wenn `a`: 66 und `b`: 66, dann ist `r`: 6
   - Wenn `a`: 34 und `b`: 34, dann ist `r`: 3 oder 4

   Testen Sie Ihre Loesung mit `a` gleich 34 und `b` gleich 43. Was liefert Ihr Programm?

2. Vervollständigen Sie “SumPattern.java”. In der Main-Methode sind drei int Variablen `a`, `b`, und `c` deklariert und mit irgendwelchen Werten initialisiert. Wenn die Summe von zwei der Variablen die dritte ergibt, nehmen wir an dass `a + c == b`, so soll die Methode “`Moeglich. a + c == b`” ausgeben (wobei die Werte für `a`, `b`, und `c` einzusetzen sind). Wenn das nicht der Fall ist, dann soll die Methode “`Unmoeglich.`” ausgeben.

   Beispiele:

   - Wenn `a`: 4, `b`: 10, `c`: 6, dann wird “`Moeglich.  4 + 6 == 10`” oder “`Moeglich.  6 + 4 == 10`” ausgegeben.
   - Wenn `a`: 2, `b`: 12, `c`: 0, dann wird “`Unmoeglich.`” ausgegeben.

3. Vervollständigen Sie “AbsoluteMax.java”. In der Main-Methode sind drei int Variablen `a`, `b`, und `c` deklariert und mit irgendwelchen Werten initialisiert. Das Programm soll einer int Variable `r` den grössten absoluten Wert von `a`, `b`, und `c` zuweisen.

Aufgabe 5: EBNF

Aufgabe 5: EBNF

In dieser Aufgabe erstellen Sie erneut verschiedene EBNF-Beschreibungen. Speichern Sie diese wie gewohnt in der Text-Datei "EBNF.txt", welche sich in Ihrem "u02"-Ordner bzw. im "U02 <*N-ETHZ-Account*>"-Projekt befindet. Sie können die Datei direkt in Eclipse bearbeiten.

1. Erstellen Sie eine Beschreibung `<pyramid>`, welche als legale Symbole genau jene Wörter zulässt, welche aus einer Folge von strikt aufsteigenden, gefolgt von einer Folge von strikt absteigenden Ziffern bestehen. Beispiele sind: 14, 121, 1221, 1341.
   Sie dürfen annehmen, dass das leere Wort auch zugelassen wird. (Als Challenge können Sie probieren, das leere Wort auszuschliessen.)

Aufgabe 5: EBNF

2. Erstellen Sie eine Beschreibung `<digitsum>`, welche als legale Symbole genau jene natürlichen Zahlen zulässt, deren Quersumme eine gerade Zahl ist.

3. Erstellen Sie eine Beschreibung `<xyz>`, welche genau alle Wörter zulässt, die aus X, Y und Z bestehen und bei welchen jedes X mindestens ein Y im Teilwort links und rechts von sich hat. Beispiele sind: Z, YXY, YXXY, ZYXYY.

4. Erstellen Sie eine Beschreibung `<term>`, welche als legale Symbole genau alle wohlgeformten arithmetischen Terme, bestehend aus positiven ganzen Zahlen, Variablen (x, y, z), Addition und Klammern zulässt. Geklammerte Terme müssen mindestens eine Addition enthalten. Beispiele sind: 1 + 4, (1 + 4), 1 + (3 + 4), (1 + 1) + x + 5.

Outro

- **Nutzt das Studycenter für Eprog**
- **Wenn ihr möchtet das ich bestimmte Aufgaben nochmal bespreche, dann schreibt mir gerne eure Wünsche**
- **Gerade kann alles noch etwas überwältigend sein, aber bleibt dran, ihr schafft das**