Einführung in Java

Chris Weber, Kantonsschule Limmattal

EF Informatik 2023/24

1 Imperative und funktionale Programmierung

1.1 Hello World (Github, IDE, Terminal, erstes Programm)

1. Gehe auf https://github.com/KS-Limmattal/EF-Informatik-2022-23/ und nimm die Aufgabe 1 an. Klone den Code auf deinen Computer (bzw. Programmier-Stick), z.B. in einen Ordner `Projects`:

   - Arbeitest du auf einem eigenen Computer, solltest du Git installieren (https://git-scm.com/download). Dann kannst du in Visual Studio Code (VSC) auf Version Control klicken (drittoberstes Icon am linken Rand) und danach auf „Clone Repository“. Du musst dich in deinen Github-Account einloggen, danach kannst du dein Repository klonen.

   - Arbeitest du mit Programmierstick auf einem Schulcomputer, musst du leider das Repository von der Webseite herunterladen und entzippen, da auf den Schulcomputern kein Git läuft.

2. Du solltest nun einen Unterordner `23_01a_Grundlagen` haben, in dem bereits einige Dateien liegen. Diese kannst du im Moment ignorieren, du wirst sie in Abschnitt 1.3 brauchen. Stelle sicher, dass du den Unterordner `23_01a_Grundlagen` in VSC geöffnet hast (ggf. mit Ctrl&K, Ctrl&O öffnen).

3. Erstelle in VSC eine neue Datei mit dem Namen `Hello.java` (Gross- und Kleinschreibung beachten!) und fülle sie mit dem folgenden Code (Tabulator für Einrückungen):

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

   Speichere (Ctrl&S).

4. Öffne eine **Konsole** (Terminal/ New Terminal) und tippe darin die folgenden Befehle (jeweils mit Enter bestätigen):

   `javac Hello.java` (**Kompiliert** die Datei zu `Hello.class`)

   `java Hello` (führt die Datei `Hello.class` aus)

Kontrolliere mit `ls` oder `dir` (je nach Betriebssystem), welche Dateien im Verzeichnis sind.

weiterer Grundbefehl: `cd ..` bzw. `cd 01_Grundlagen` (*change directory*)

5. Eine „Vorschau“, in der die Schritte aus dem vorherigen Punkt im Hintergrund ausgeführt werden, ist der „Run“-Knopf, der über der `main()`-Methode erscheint (kann auch über die Taste F5 aufgerufen werden). Lösche die Datei `Hello.class` und probiere ihn aus. Was fällt auf?

   Antwort: Es wird keine neue Datei `Hello.class` erstellt! Die kompilierte Klasse wird also lediglich temporär im Arbeitsspeicher erstellt.

6. Mache auf Github einen Commit, der deine Datei `Hello.java` enthält:

   - Arbeitest du auf einem eigenen Computer, kannst du in VSC/Version Control „Commit“ (in dein lokales Repository) und danach „Sync Changes“ verwenden.
   - Arbeitest du auf einem Schulcomputer, lade die Datei direkt auf der Webseite hoch (im Repository unter „Add File“).

In unserem ersten Programm stehen sehr viele kryptische Schlüsselwörter, deren Bedeutung wir nach und nach kennenlernen werden. Ein paar erste Erklärungen:

- Die **Klasse** `Hello` enthält unser „Hello World“-Programm.
- die **Methode** `main()` wird ausgeführt, wenn `Hello` ausgeführt wird. Klassen können noch weitere Methoden enthalten.
- `System` ist eine bereits bestehende Klasse, auf die wir in unserem Programm zugreifen. Sie enthält ein Objekt `out`, das die Methode `println()` enthält, die einen übergebenen String (Text) auf der Konsole ausgibt.

1.2 Calc (Variablen, Datentypen int und String, Operationen, Syntax)

1. Erstelle eine Datei `Calc.java` mit dem Inhalt

```java
public class Calc {
  public static void main(String[] args) {
    int a = 34;
    int b = 7;
    System.out.println("sum=" + a + b);
  }
}
```

   Was erwartest du, dass das Programm macht? Versuche eine Theorie aufzustellen, was die Code-Zeilen bedeuten. Probiere es danach aus (kompilieren und ausführen im Terminal oder über den „Run“-Knopf). Hast du richtig geraten? Falls nein, hast du eine Erklärung dafür?

Antwort:
`int a = 34;` macht drei Dinge: Es definiert eine **Variable** mit dem Namen `a`, definiert, dass sie vom Typ `int` (Ganzzahl, *integer*) sein soll, und speichert darin den Wert 34.
Naiv könnte man erwarten, dass das Programm die Summe aus `a` und `b` berechnet und danach `sum=41` auf der Konsole ausgibt. Das ist aber nicht der Fall.
`"sum="` ist Text, also vom Datentyp `String`. Wenn Java den Ausdruck `"sum=" + a` sieht, wandelt es den Inhalt von `a` in einen String um (das nennt man **automatische Typkonversion**) und hängt die beiden Strings zu einem neuen String `"sum=34"` zusammen. Danach passiert das Gleiche mit dem String `"sum=34"` und dem Inhalt von `b`, so dass am Schluss der String `"sum=347"` herauskommt.

2. Versuche, als Resultat die Summe von a und b angezeigt zu bekommen.

Mögliche Lösungen:
- Eine dritte Variable `int c = a + b;` definieren und danach `"sum=" + c` ausgeben.
- Klammern setzen: `System.out.println("sum=" + (a + b));`. Java verrechnet hier (mathematisch gut erzogen) zuerst die Dinge in der Klammer miteinander.

In beiden Fällen erreichen wir, dass die Operation `a + b` ausgeführt wird. Da `a` und `b` beides Zahlen vom Typ `int` sind, bedeutet `+` hier die ganz normale ganzzahlige Addition. Danach wird das Resultat (ebenfalls eine `int`-Zahl) mit der `String`-Operation `+` an `"sum="` angehängt.

3. Ersetze `+` der Reihe nach durch `-`, `*`, `/` und `%` und beobachte das Verhalten. Falls nötig, experimentiere mit anderen Werten für `a` und `b`, um zu verstehen, was passiert.

4. Sind die Einrückungen, Zeilenwechsel und Semikolons nötig für einen reibungslosen Ablauf des Programms? Probiere aus!

Antwort: Die Zeilenwechsel und Einrückungen haben keinen Einfluss auf die Funktion des Programms. Sie sind lediglich für uns Menschen da, um die Übersicht nicht zu verlieren. Das heisst, wir können auch in überlange Befehlszeilen Zeilenwechsel einfügen, so dass alles auf einer Bildschirmbreite Platz hat. Für den Java-Compiler ist das Semikolon das Zeichen für einen abgeschlossenen Befehl. Es ist deshalb unverzichtbar.

1.3 Powers (Methoden, Kommentare, boolean, Verzweigungen)

1. Studiere nun die Klasse `Powers.java`, die im geklonten Repository schon vorhanden war. Was verstehst du? Führe sie aus und überprüfe deine Überlegungen.

2. Ein paar Erläuterungen:

   - In der Datei `Powers.java` ruft die `main()`-Methode die Methode `square()` auf. Dabei übergibt sie den Integer `base` als Argument und bekommt als `return`-Wert wieder einen Integerwert zurück. Deshalb muss `int` (=der Typ des Rückgabewertes) vor der Definition der Methode `square()` stehen. Wir sagen auch, die Methode `square()` sei **vom Typ** `int`. Wenn eine Methode nichts zurückgibt (wie die `main()`-Methode), steht `void`.
   - `static` muss im Moment vor jeder Methode stehen, wir werden später sehen, wieso.
   - `//` und der `/** **/`-Block markieren **Kommentare** (ein- bzw. mehrzeilig), die vom Compiler ignoriert werden. Der `/** * **/`-Block ist in diesem Beispiel ein **Javadoc**-Kommentar. Er ist so formatiert, dass daraus automatisch Dokumentationsseiten für unser Java-Projekt erstellt werden könnten. Auch die Klassen aus der Java-Library sind auf diese Art dokumentiert: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/System.html Finde in der Dokumentation zur Klasse `System` die Beschreibung der Methode `println()`. Was bemerkst du?

     Antwort:
     - `out` ist eine Variable vom Typ `PrintStream`. Deshalb besitzt es auch die Methoden dieses Typs.
     - Es gibt nicht nur eine Methode `println()`, sondern deren zehn! Jede davon nimmt einen anderen Datentyp entgegen. Es ist in Java tatsächlich möglich, Methoden zu **überladen** (***method overloading***), d.h. mehrere Methoden mit dem gleichen Namen, aber unterschiedlicher Zahl oder Typ von Argumenten zu definieren.
     - `println()` ohne Argument gibt nur einen Zeilenwechsel aus. Die anderen Methoden sind aus `print()` mit dem gleichen Argumenttyp und einem Zeilenwechsel (argumentloses `println()`) zusammengesetzt.

3. Ergänze die Datei mit einer zweiten Methode `cube()`, die die dritte Potenz eines `int`-Arguments `base` zurückgibt. Teste sie mit einer Ausgabe in der Konsole.

4. Berechne `cube(10000)`. Hast du eine Idee, was da passiert? Probiere aus!

   Antwort: Offenbar gibt Ihre Methode für Argumente grösser als 8470 Quatsch aus. Das hat damit zu tun, wie `int`-Zahlen im Speicher dargestellt und verarbeitet werden. Mehr Details mündlich!
   Fortgeschrittenen-Aufgabe: Erklärе möglichst genau, warum und wie das falsche Resultat zustande kommt.

5. Wir wollen eine Warnung auf der Konsole ausgeben, falls `base` grösser als 8470 ist. Ergänze vor dem `return`-Befehl in der Methode `cube()` den Code

```
if (base > 8470) {
  System.out.println("Warning: The cube of " + base + " is outside the
    range of int.");
}
```

   Probiere aus.

6. Ersetze die Klammer `}` des `if`-Befehls durch

```
} else if (base == 0 || base == 1) {
  System.out.println("useless operation, but ok");
} else {
  System.out.println("Nothing to worry about :-)");
}
```

   und probiere aus. Du hast eine **Verzweigung** programmiert.

7. Die Bedingung `(...)` für das Ausführen der geschweiften Klammer ist vom Typ `boolean` (mögliche Werte: `true` oder `false`). Boolesche Werte können in Variablen gespeichert werden mittels z.B. `boolean a = true;` oder `boolean bedingung = (a >= 0);` Sie können dann benutzt werden in Ausdrücken wie `if a {...}`. Experimentiere damit!

8. **Boolesche Logik:**

`&&`	und (AND)	`==`	ist gleich	`<`	ist kleiner als
`||`	oder (OR)	`!=`	ist ungleich	`>=`	ist grösser/ gleich
`!a`	nicht (NOT) `a`	`>`	ist grösser als	`<=`	ist kleiner/ gleich

   Ausserdem kann mit Klammern gearbeitet werden:

```
(a > b && !(a <= 0)) || a == 0
```

   Es gelten die **Regeln von DeMorgan:**

```
!(a && b) = !a || !b
!(a || b) = !a && !b
```

9. Betrachte die Datei `Variablensichtbarkeit.java`. Wenn du in den kommentierten Stellen `// (1)` bis `// (5)` den Wert der Variablen `a` und `b` ausgibst, was erwartest du? Probiere es danach aus! Entsprechen die Resultate deinen Theorien? Wenn nicht, hast du Erklärungen dafür?

   Erklärung:
   Das Programm durchläuft die Kommentare in der angegebenen Reihenfolge. Es gibt zwei unterschiedliche Variablen `a`, nämlich in jeder der beiden Methoden eine. Jede Methode hat nur Zugriff auf die eigene **lokale Variable** `a`. Zum Zeitpunkt `// (3)` ist `a` undefiniert.
   `b` hingegen ist eine **globale Variable**, die in der ganzen Klasse definiert ist und von überall her verändert und zugegriffen werden kann.

10. Weisst du, was im Speicher geschieht, wenn `Variablensichtbarkeit` ausgeführt wird?