Teil 1:

Aufgaben

1. Lesen Sie die folgenden Erläuterungen genau durch.
2. Setzen Sie die angegebenen Beispiele (hellblaue Rechtecke) in ein oder mehrere eigenständige Java-Programme um. Geben Sie, wo immer dies möglich ist, den Inhalt deklarierter Variablen auf die Standardausgabe aus.
3. Probieren Sie aus was passiert, wenn Sie die Maximal-/Minimalwerte von Datentypen (`int`, `long`), die ganze Zahlen repräsentieren um 1 erhöhen / -1 verkleinern.
4. Probieren Sie aus was passiert, wenn Sie zu den großen Zahlen bei `float` oder `double` große Zahlen hinzuaddieren.
5. In Variablen vom Typ `char` befindet sich in Wirklichkeit ein ganzzahliger positiver Wert, der den Code eines Zeichens darstellt. Untersuchen Sie, ob und/oder wie man damit rechnen kann.

Datentöpfe – Datentypen

Um Information in Programmen verarbeiten zu können, müssen Computer-Programme grundsätzlich in der Lage sein, Information zu speichern. Um beispielsweise eine Zahl, die der Anwender eingibt, im Programm speichern zu können, muss es eine Art Topf geben, in den Sie diese Zahl hineinlegen können. Wird später vom Programm eine Berechnung durchgeführt, in der auf diese Zahl zugegriffen werden muss, dann wird der gespeicherte Zahlenwert einfach aus dem Topf herausgeholt. Diese Töpfe heißen in den Programmiersprachen **Variablen**. Eine Variable ist also ein Topf, der eine ganz bestimmte Information speichern kann.
In jeden Topf - also in jede Variable - passt jeweils genau eine Information hinein. Wollen Sie eine Zahl speichern, benötigen Sie einen Topf für diese Zahl. Wollen Sie eine zweite Zahl in Ihrem Programm speichern, benötigen Sie einen zweiten Topf. Legen Sie die zweite Zahl nämlich in den Topf hinein, in den Sie bereits die erste Zahl hineingelegt haben, geht die erste Zahl verloren. Wollen Sie also in Ihrem Programm mehrere Informationen gleichzeitig speichern, benötigen Sie für jede zu speichernde Information einen Topf, also eine Variable.
Wo Sie Variablen herbekommen, wie Sie Informationen in Variablen speichern und was Sie sonst noch im Umgang mit Variablen in Java berücksichtigen müssen, das erfahren Sie in diesem Kapitel.

Primitive Datentypen

Für jede zu speichernde Information benötigen Sie einen Topf - das ist aber noch nicht alles. Abhängig von der Art der Information, die Sie speichern möchten, brauchen Sie jeweils einen anderen Topf. Das heißt, wollen Sie eine Zahl speichern, brauchen Sie einen Topf, der Zahlen speichern kann. Wollen Sie hingegen einen Buchstaben speichern, brauchen Sie einen Topf, der Buchstaben speichern kann. Wollen Sie eine Buchstabenkette - also ein Wort oder einen Satz speichern - brauchen Sie wiederum einen anderen Topf, der Buchstabenketten speichern kann. Sie benötigen also jeweils die richtige Art von Topf, um die entsprechende Information speichern zu können. Man spricht davon, dass Variablen jeweils den richtigen **Datentyp** benötigen. Der Datentyp beschreibt, welche Art von Information in einer Variablen gespeichert werden kann.

Java bietet wie auch andere Programmiersprachen eine Handvoll **primitiver** Datentypen an. Eine andere Bezeichnung lautet intrinsische Datentypen. Es handelt sich um Datentypen, die in die Programmiersprache Java direkt eingebaut sind und die Sie ohne besondere Vorkehrungen verwenden können. Es ist zum Beispiel nicht notwendig, primitive Datentypen über eine import-Anweisung in Ihr Programm einzubinden, um sie dem Java-Compiler bekannt zu machen. Der Java-Compiler kennt von Haus aus alle primitiven Datentypen. Genau das zeichnet diese Datentypen ja aus.

```
boolean b;
char c;
byte y;
short s;
int i;
long l;
float f;
double d;
```

Die in Java existierenden primitiven Datentypen sind **boolean**, **char**, **byte**, **short**, **int**, **long**, **float** und **double**. Jede oben definierte Variable besitzt einen anderen Datentyp, kann also eine andere Art von Information speichern.

Um Variablen in Java anzulegen, geben Sie zuerst einen Datentyp an und dahinter durch ein Leerzeichen getrennt einen Variablennamen. Variablendefinitionen werden in Java am Zeilenende zusätzlich mit einem Semikolon abgeschlossen. Die obigen Code-Zeilen legen also die Variablen ***b***, ***c***, ***y***, ***s***, ***i***, ***l***, ***f*** und **d** an. Jede dieser Variablen hat einen anderen Datentyp. Ihnen gemeinsam ist jedoch, dass jede dieser Variablen auf einem primitiven Datentyp basiert.

Während bei der Definition der Variablen der Datentyp angegeben werden muss, erfolgen Zugriffe auf Variablen nach der Definition nur mehr über den Variablennamen. Der Datentyp wird also für eine Variable genau einmal angegeben, und zwar immer nur bei der Definition. Achten Sie in den folgenden Beispielen darauf, dass Zugriffe auf Variablen ausschließlich über den Variablennamen stattfinden.

```
boolean b;
b = true;
b = false;
```

Variablen vom Datentyp **boolean** können entweder den Wahrheitswert **true** oder den Wahrheitswert **false** speichern. Der Datentyp **boolean** eignet sich also für Variablen, die genau einen von zwei Zuständen speichern sollen. Ein typisches Beispiel ist also ein Lichtschalter, der entweder an oder aus ist. Den Zustand des Lichtschalters könnte man in Java in einer Variablen vom Typ **boolean** speichern, wobei **true** bedeutet, dass das Licht brennt, und **false**, dass das Licht aus ist.

Man spricht bei einem Speichervorgang, der mit Hilfe des Operators „**=**“ erfolgt, von einer **Zuweisung**. Das, was rechts vom Operator „**=**“ steht, wird dem, was links steht, zugewiesen. Die Zuweisung findet immer von rechts nach links statt. Obige Zeilen weisen also einmal den Wahrheitswert **true**, einmal den Wahrheitswert **false** der Variablen ***b*** zu.

Der **Zuweisungsoperator** „**=**“ findet nicht nur bei Variablen vom Typ **boolean** Anwendung, sondern ist für Variablen jeden Datentyps definiert.

```
char c;
c = 'A';
```

Der Datentyp **char** wird benötigt, um einzelne Buchstaben in Variablen speichern zu können. Genaugenommen können nicht nur Buchstaben, sondern beliebige Zeichen in Variablen vom Datentyp **char** gespeichert werden. Das zu speichernde Zeichen muss hierbei immer in einfache Anführungszeichen eingeschlossen werden. Pro Variable vom Datentyp **char** kann nur ein Zeichen gespeichert werden. Probieren Sie in diesem Zusammenhang auch folgendes aus (und geben die Ergebnisse sofern möglich auch aus):

```
c = 'A' + 1;
int ci = 'A' + 1;
ci = c;
c = ci;
```

Beim Speichern von Zahlen sind vor allem auch die Verhaltensweisen in Grenzbereichen interessant.

```
byte y;
short s;
int i;
long l;

y = -128;
y = 127;
s = -32768;
s = 32767;
i = -2147483648;
i = 2147483647;
l = -9223372036854775808;
l = 9223372036854775807;
```

Variablen vom Typ **byte, short, int** oder **long** ermöglicht das Speichern von ganzen Zahlen. Die Datentypen unterscheiden sich insofern, als dass sie unterschiedliche Bandbreiten besitzen. Während in Variablen vom Typ **byte** nur Zahlen von -128 bis +127 gespeichert werden können, können Variablen vom Typ **long** Zahlen von -9223372036854775808 bis +9223372036854775807 speichern. Je größer die Bandbreite, umso mehr Speicherplatz verschlingt der Datentyp. In dem Beispiel oben ist jeweils Minimal- und Maximalwert zugewiesen.

Falls Sie sich ein wenig mehr mit Speichergrößen auskennen, ist folgende Information hilfreich für Sie: Der Datentyp **byte** benötigt 1 Byte, **short** 2 Bytes, **int** 4 Bytes und **long** 8 Bytes. Diese Speichergrößen gelten für die Datentypen auf jedem beliebigen physikalischen Rechner - egal, ob mit 32-Bit- oder 64-Bit-Prozessor. Der Datentyp **boolean** belegt übrigens 1 Byte, während der Datentyp **char** im Gegensatz zu anderen Programmiersprachen 2 Bytes belegt. Grund ist, dass Java nicht mit dem ASCII-Zeichensatz oder einem der ISO-8859-Zeichensätze arbeitet, sondern komplett auf Unicode basiert, der per definitionem (mindestens) 2 Bytes für jedes Zeichen reserviert.

```
float f;
double d;

f = -3.40282347e38;
f = 3.40282347e38;
d = -1.79769313486231570e308;
d = 1.79769313486231570e308;
```

Mit den Datentypen **float** und **double** ist das Speichern von Kommazahlen möglich. **float** speichert Kommazahlen mit einfacher Genauigkeit (4 Bytes, ca. 7 Dezimalstellen), **double** mit doppelter Genauigkeit (8 Bytes, ca. 15 Dezimalstellen). Für Sie als Programmierer sind wiederum die Bandbreiten der beiden Datentypen am interessantesten, deren Grenzwerte im obigen Beispiel-Code angegeben sind.

Nochmal zur Verdeutlichung: Um Kommazahlen in Variablen zu speichern, muss die Variable vom Typ **float** oder **double** sein. Es ist nicht möglich, Kommazahlen in Variablen vom Typ **byte**, **short**, **int** oder **long** zu speichern. Diese Datentypen ermöglichen ausschließlich das Speichern von Ganzzahlen.

Beachten Sie obige Schreibweise der Kommazahlen (Dezimalpunkt). Hinter dem Buchstaben **e** kann ein Exponent für die Kommazahl angegeben werden; d.h. der angegebene Wert ist mit 10^e multipliziert.

Probieren Sie u.a. auch mal aus was passiert, wenn Sie die oben gezeigten **float** oder **double**-Zahlen mit 10 multiplizieren und ausgeben.

Referenzvariable

Eine Klasse wird genauso verwendet wie ein Datentyp. Sie können also Klassen unter anderem als Datentyp betrachten. Genauso wie Sie Variablen von primitiven Datentypen anlegen können, können Sie Variablen vom Typ einer Klasse anlegen. Man spricht in diesem Fall von **Referenzvariablen**. Referenzvariablen sind also Variablen vom Typ einer Klasse. Die offizielle Java-Klassenhierarchie enthält eine ganze Menge an Klassen. Sie werden in naher Zukunft Referenzvariablen anlegen, die als Datentyp die eine oder andere Klasse verwenden. Aktuell verwenden wir die Klasse **`String`**, die sich hervorragend dafür eignet, Zeichenketten zu speichern. Die Operation „+“ spielt im Zusammenhang mit **`String`** -Variablen oder **`String`**-Literalen eine wichtige Rolle. Zeichenketten werden zusammengefügt, Werte werden zu Zeichenketten konvertiert und zusammengefügt

```
String string;

string = " ich bin ein String";
string = 123 + string;
string = string + "\n\tund ein \"Referenztyp\"";
```

Teil 2:

Fingerübung 1

Entwickeln Sie eine Java-Anwendung, die drei Variablen vom Typ **`int`** besitzen soll. Setzen Sie die erste Variable auf den Wert 10 und lesen Sie für die zweite und dritte Variable einen Wert ein. Addieren Sie die Werte der beiden ersten Variablen miteinander und multiplizieren Sie dann das Ergebnis mit dem Wert der dritten Variablen. Geben Sie daraufhin das Ergebnis der Berechnung auf die Standardausgabe aus. Geben Sie außerdem die Rechnung auf die Standardausgabe aus, mit der das Ergebnis zustande kam, damit der Anwender die Rechnung und das Ergebnis nachvollziehen kann. Beispiel einer Ein- und Ausgabe:

```
Variable2> 13                          //Eingabe
Variable3> 17                          //Eingabe
Ergebnis: (10 + 13) * 17 = 391         //Ausgabe
```

Hinweis: für eine elegante Ausgabe siehe auch die Unterlage „Ausgabeformatierung“ (ILIAS, Lehrmittelordner).

Fingerübung 2

Schreiben Sie ein Programm, das folgende Quadratgleichung mit einer Variablen x vom Typ **`double`** berechnet:

$$3x^2 - 8x + 4$$

Weisen Sie der Variablen **`x`** einen Wert zu. Schreiben Sie eine Anweisung, die den Wert der Quadratfunktion berechnet und das Ergebnis in eine andere Variable vom Typ **`double`** schreibt. Lassen Sie das Ergebnis ausgeben, ungefähr in der Form:

```
Bei x = 4.0 ergibt die Quadratgleichung den Wert 20.0
```

Führen Sie das Programm für verschiedene Werte von x aus und betrachten Sie das Ergebnis. Die Quadratgleichung sollte bei **`x = 2.0`** und bei **`x = 2.0/3.0`** Null ergeben. Testen Sie für diese Werte von x. Ist das Ergebnis richtig?

Legen Sie das Programm so an, das es die verschiedenen Ergebnisse für **`x=0`**, **`x=2`**, **`x=2.0/3.0`** und **`x=4`** nacheinander berechnet und ausgibt.

Machen Sie in einem nächsten Schritt die Variable **`x`** einlesbar. Verwenden Sie dann Dezimalwerte, große und kleine Werte, negative Werte und Nullwerte.

Teil 3: Syntaktische Fehler, der Umgang damit bzw. deren Vermeidung

Die Datei Addition.java hat folgenden Inhalt:

```
public class Addition
{
  public static void main(String[] args)
  {
        System.out.print("Dieses Programm addiert drei Zahlen. ),
        int x = 3;
        y = 5;
        int z = 1.5;
        System.out.print("Die Summe von " + x + ", " + y + " und " + z " ist: ");
        System.out.println(x+y+z)
  }
}
```

Das Programm enthält sechs syntaktische Fehler. Korrigieren Sie alle syntaktischen Fehler, so dass das Programm kompilierbar ist. Die einfachste Möglichkeit ist natürlich, das Programm zu kompilieren, die Fehlermeldungen anzuschauen und den Quellcode entsprechend zu editieren. Die Fehler sind behoben, wenn der Compiler keine mehr meldet. Das richtige Vorgehen hierbei ist, immer den ersten Fehler zu korrigieren, den der Compiler meldet, und dann von neuem zu kompilieren. Der Grund hierfür ist, dass die weiteren Fehler, die der Compiler nach dem ersten Fehler meldet, nicht mehr in jedem Fall verlässlich sind.

Kopieren Sie das Programm also nicht einfach nach Eclipse sondern erstellen Sie es systematisch neu.

- Erst die Klasse erstellen
- Dann die main-Methode dareinsetzen.
- Dann Zeile für Zeile den Inhalt erfassen; immer wenn ein Fehler auftaucht erst diesen korrigieren, dann erst weitermachen.
- Anmerkung: Sollte Sie trotzdem alles auf einmal kopieren, wird erstmal nur ein Fehler angezeigt!

In Zukunft dann immer so arbeiten. Erst Fehler bearbeiten bevor Sie weitermachen. Je mehr Fehler Sie gleichzeitig haben, umso schwieriger wird es sonst, den einzelnen Fehler zu lokalisieren.