Zusammenfassung Teil 1

1. Variablen

Variablen sind Speichermöglichkeiten für Daten.

Beschreibung	Syntax	Beispiele

Deklaration	*Datentyp* name;	`int x;`
		`double y;`
		`int a,b,c;`

Deklaration und Initialisierung	*Datentyp* name= *initialerWert*;	`int x= 1;`
		`double y= 7.5;`
		`int a=1,b=2,c=3;`

Zuweisung	name = *Operation*;
	Die Zusweisung ist immer von Rechts nach Links. Der Typ auf der rechten Seite muss der linken zuweisbar sein.
		`int x= 8;`
		`double y= 9;`
		`y= x;`
		`/* x= y; geht nicht*/`
		`x= 9+2;`

1.1. primitive Datentypen

Typ	Beschreibung	Grösse in Bit	Wertebereich
`boolean`	Wahr-/Falsch-Wert	8	`true` oder `false`
`char`	Zeichen	16	Unicode-Zeichen (siehe nächste Woche)
`byte`	Ganzzahl	8	-128 … +127 + -2^7 … $+(2^7-1)$
`short`	Ganzzahl	16	-32‘768 … +32‘767 + -2^{15} … $+(2^{15}-1)$
`int`	Ganzzahl	32	-2‘147‘483‘648 …+2‘147‘483‘647 bzw. -2^{31} … $+(2^{31}-1)$
`long`	Ganzzahl	64	-9‘223‘372‘036‘854‘775‘808…-9‘223‘372‘036‘854‘775‘807 bzw. -2^{63} … $+(2^{63}-1)$
`float`	Gleitkommazahl	32	+/- $3.40282347*10^{38}$
`double`	Gleitkommazahl	64	+/- $1.79769313486231570*10^{308}$

2. Operationen

2.1. Rechenoperationen

Operator	Rang	Typ	Beschreibung
++, --	1	arithmetisch	Inkrement und Dekrement
+, -	1	arithmetisch	unäres Plus und Minus
(Typ)	1	jeder	Cast
*, /, %	2	arithmetisch	Multiplikation, Division, Rest Wenn mehrere gleichartige Rechenoperationen nacheinander stehen, dann werden diese von links nach rechts abgearbeitet. A/B/C = (A/B)/C Achtung bei der ganzzahligen Division. Hier ist auch das Resultat eine ganze Zahl. (3/4 → 0)
+, -	3	arithmetisch	binärer Operator für Addition und Subtraktion
?:	12	jeder	Bedingungsoperator
=	13	jeder	Zuweisung
*=, /=, %=, +=,	14	arithmetisch	Zuweisung mit Operation

2.2. Logische Operationen

Operator	Rang	Typ	Beschreibung
!	1	boolean	logisches Komplement
<, <=, >, >=	5	arithmetisch	numerische Vergleiche
==, !=	6	primitiv	Gleich-/Ungleichheit von Werten
&&	10	boolean	logisches konditionales Und, Kurzschluss
||	11	boolean	logisches konditionales Oder, Kurzschluss

3. Kontrollstrukturen

3.1. Verzweigungen

Verzweigungen werden **einmal** getestet und dann wird ein Anweisungsblock ausgewählt, der

ausgeführt wird.

3.1.1. if-else-Verzweigung

Syntax	Beispiel

if (*Bedingung*) { … }	`if (5>4) {`
	`  System.out.println("Die Welt ist in Ordnung");`
	`}`

if (*Bedingung*) { … }
else { … }
	`if (5>4) {`
	`  System.out.println("Die Welt ist in Ordnung");`
	`} else {`
	`  System.out.println("WAS!");`
	`}`

if (*Bedingung*) { … }
else if (*Bedingung*) {…}
	`int x= // Zufallszahl;`
	`if (x>10) {`
	`  System.out.println("mehr als 10");`
	`} else if (x>5){`
	`  System.out.println("mehr als 5 weniger als 11");`
	`}`

if (*Bedingung*) {…}
else if (*Bedingung*){…}
else { … }
	`int x= // Zufallszahl;`
	`if (x>10) {`
	`  System.out.println("mehr als 10");`
	`} else if (x>5){`
	`  System.out.println("mehr als 5 weniger als 11");`
	`} else {`
	`  System.out.println("nicht grösser 5");`
	`}`

3.2. Schleifen

Schleifen wiederholen einen Anweisungsblock meist **mehrmals**. Alle Schleifen brauchen die folgenden Elemente, welche sich auf eine oder mehrere Variablen beziehen (auch Laufvariable(n) genannt):

- *init*: Die **Initialisierung** setzt einen oder mehrere Werte fest, bei dem/denen gestartet wird.
- *test*: Beim **Test** wird überprüft, ob die Schleife noch einmal ausgeführt werden soll. Wenn der Test `false` ergibt wird die Schleife abgebrochen.

- *update*: Das **Update** verändert die Variable(n). Falls das Update fehlt, ist der Test immer erfüllt, und die Schleife endet nie.

3.2.1. for-Schleife

Die for-Schleife wird meist dann benutzt, wenn bekannt ist, wie oft etwas wiederholt werden soll. Das kann sein genau 10 Mal oder auch für alle Elemente in einem Array.

Syntax	Beispiel
for (*init*; *test*; *update*) {...}	`for (int i=0; i<10; i++){ ...}` `for (int j=10; j>0; j--){ ...}`

3.2.2. while-Schleife

Die while-Schleife wird vor allem dann verwendet, wenn nicht klar ist, wie oft etwas wiederholt werden soll. Es kann auch sein, dass die Schleife gar nie ausgeführt wird.

Syntax	Beispiel

init;
while (*test*){
update;
}
	`int i=0;  // init`
	`while (i<10){  // test`
	`  i++;  // update`
	`}`
	`int j=20;  // init`
	`while (j>0){  // test`
	`  j= j-2;  // update`
	`}`
	`while (j>0){  /* jetzt ist j schon 0 und es passiert nichts mehr*/`
	`  j= j-2;`
	`}`

3.2.3. do-while-Schleife

Die do-while-Schleife wird vor allem dann verwendet, wenn nicht klar ist, wie oft etwas wiederholt werden soll, aber klar ist, dass die Schleife mindestens ein Mal ausgeführt werden soll. Ein typisches Beispiel ist eine Eingabe von der Konsole, welche wiederholt werden soll, wenn der Benutzer etwas Falsches eingegeben hat.

Syntax	Beispiel

init;
do {
update;
} while (*test*);
	`int i=0;  // init`
	`do {`
	`  i++;  // update`
	`} while(i<10);  // test`
	`int j=20;  // init`
	`do {`
	`  j= j-2;  // update`
	`} while (j>0);  // test`
	`do {  /* wird 1 Mal ausgeführt. Auch wenn j schon <=0. */`
	`  j= j-2;`
	`} while (j>0);`

4. Methoden

Methoden fassen Anweisungen zu einer sinnvollen Gruppe zusammen. Methoden haben einen Namen, können Parameter haben und können eine Rückgabe haben. In Java kann immer nur ein Element zurückgegeben werden.

```
public class Methoden{
    public static void main(String[] args){
        ausgabe();
        ausgabe("Hallo");
        int geheimnis= getNumber();
        double resultat= division(3,4);
        int[] arrayLeer= newArray(5);
    }
    public static void ausgabe(){
        System.out.println("die Methode wurde ausgeführt");
    }
    public static void ausgabe(String text){
        System.out.println("Text "+text);
    }
    public static int getNumber(){
        return 42;
    }
    public static doule division(int x, int y){
        return (double)x/y;
    }
    public static int[] newArray(int size){
        int[] array= new int[size];
        return array;
    }
}
```

5. Arrays

Was	Syntax	Beispiel

Deklaration	*Datentyp*[] name;
	Datentyp[][] name;
		`int[] values;`
		`double[][] matrix;`
		`String[] texts;`

Initialisierung	name = new *Datentyp*[7];
	name = new *Datentyp*[*variable*];
		`values = new int[8];`
		`matrix = new double[6][9];`
		`texts= new String[2];`
		`int x= 9;`
		`values = new int[x];`
		`matrix = new double[6][x];`
		`texts= new String[x];`

Deklaration & Initialisierung	*Datentyp*[] name= new *Datentyp*[7];
	Datentyp[][] name= new *Datentyp*[*variable*];
		`int x= 9;`
		`int[] values = new int[x];`
		`double[][] matrix = new double[6][x];`
		`String[] texts= new String[x];`

Werte Abspeichern	name[*Position*] = *Wert*;
	name[*Position*] = *Rechnung*;
	name[*Position1*] = name[*Position2*];
		`int[] values= new int[3];`
		`values[0]= 1;`
		`values[1]= 6;`
		`values[2]= 13;`
		`/* values[3] gibt es nicht */`
		`double[][] matrix= new double[3][4];`
		`matrix[2][0]= 4.5;`

Deklaration, Initialisierung & Werte Setzen	*Datentyp*[] name= {.., ..};
	Datentyp[][] name= {{},{}};
		`int[] values= {1,2,3,4};`
		`double[][] matrix = {{1.2, 4.3},{8.2, 3.4}};`
		`String[] texts= {"Hallo", "Du"};`

5.1. Beispiele 1Dimensionales Array

Was	Code

Werte im Array Speichern (mit Schleife)	`int[] values= new int[4];`
	`for (int i=0; i<4; i++) {`
	`  values[i]= i;`
	`}`
	`for (int i=0; i<values.length; i++) {`
	`  values[i]= i;`
	`}`

Werte aus dem Array Auslesen und die Summe bilden.	`int[] values= {1,2,3,4,5,6};`
	`int summe= 0;`
	`for (int i=0; i<values.length; i++) {`
	`  summe = summe + values[i];`
	`}`
	`/* oder */`
	`summe= 0;`
	`for (int x:values) {`
	`  summe = summe + x;`
	`}`

Etwas im Array suchen	`int[] values= {1,2,3,4,5,6};`
	`int wert= 7;`
	`for (int i=0; i<values.length; i++) {`
	`  if (wert==values[i]){`
	`  System.out.println("Juhu");`
	`  }`
	`}`

5.2. Beispiele 2Dimensionales Array

Was	Code

Werte im Array Speichern (mit Schleife)	`int[][] values= new int[4][5];`
	`for (int i=0; i<4; i++) {`
	`  for (int j=0; j<5; j++){`
	`  values[i][j]= i+j;`
	`  }`
	`}`
	`for (int i=0; i<values.length; i++) {`
	`  for (int j=0; j<values[i].length;`
	`j++){`
	`  values[i][j]= i+j;`
	`  }`
	`}`

Was	Code

Werte aus dem Array Auslesen und die Summe bilden.	`int[][] values= {{1,2,3,4},{5,6,7,8},{9,10,11,12}};`
	`for (int i=0; i<values.length; i++) {`
	`  for (int j=0; j<values[i].length; j++){`
	`  summe = summe + values[i][j];`
	`  }`
	`}`
	`/* oder */`
	`for (int[] row: values) {`
	`  for (int value:row){`
	`  summe= summe+value;`
	`  }`
	`}`

Etwas im Array suchen	`int[][] values= {{1,2,3,4},{5,6,7,8},{9,10,11,12}};`
	`for (int i=0; i<values.length; i++) {`
	`  for (int j=0; j<values[i].length; j++){`
	`  if (values[i][j] == 11){`
	`  System.out.println( "Position: "+i+"/"+j);`
	`  }`
	`  }`
	`}`

6. Klassen

6.1. String

Erstellen:

```
String text= new Strint("Hallo");
String textHallo= "Hallo";
int x= 1234;
String textX= "X: "+x;
String textNurZahl= ""+x;
```

Methode	Beschreibung	Beispiel (`s` ist `"Hello"`)
`length()`	Anzahl Zeichen im String	`s.length()` gibt `5`

Methode	Beschreibung	Beispiel (`s` ist `"Hello"`)
`charAt(index)`	Zeichen an bestimmter Stelle	`s.chatAt(1)` gibt `'e'`
`substring(start, ende)`	alle Zeichen von `start`-Index bis direkt vor `ende` -Index	`s.substring(1, 3)` gibt `"el"`
`startsWith(str)`	ob der String mit `str` beginnt	`s.startWith("a")` gibt `false`
`endsWith(str)`	ob der String mit `str` endet	`s.endsWith("llo") gibt `true`
`indexOf(str)`	Stelle, wo `str` zum ersten Mal vorkommt (`-1` falls nirgends)	`s.indexOf("l")` gibt `2`
`replace(s1, s2)`	ersetzt alle Vorkommnisse von `s1` mit `s2`	`s.replace("l", "yy")` gibt `"Heyyyyo"`
`toLowercase()`	neuer String mit lauter Kleinbuchstaben	`s.toLowerCase()` gibt `"hello"`
`toUpperCase()`	neuer String mit lauter Grossbuchstaben	`s.toUpperCase()` gibt `"HELLO"`

6.2. Scanner

Der Scanner wird zum Lesen von der Konsole verwendet.

Erstellen: `Scanner scan= new Scanner(System.in);`

Methode	Beschreibung	Beispiel
`next()`	liest das nächste *Token* (siehe unten) und gibt es als `String` zurück	`String text= scan.next();`
`nextDouble()`	liest das nächste Token und gibt es als `double` zurück	`double text= scan.nextDouble();`
`nextInt()`	liest das nächste Token und gibt es als `int` zurück	`int value= scan.nextInt();`
`nextLine()`	liest die ganze nächste Zeile und gibt sie als `String` zurück	`String text= scan.nextLine();`

6.3. Random

Um Pseudo-Zufallszahlen erzeugen zu können, kann man die Klasse Random verwenden.

Erstellen: `Random random = new Random();`

Methode	Beschreibung	Beispiel
`nextBoolean()`	Erzeugen eines zufälligen true/false-Werts.	`boolean jaNein= random.nextBoolean();`
`nextDouble()`	Erzeugen eines zufälligen Werts zwischen 0.0 und 1.0	`double zahl= random.nextDouble();`
`nextInt()`	Erzeugen eines zufälligen, ganzzahligen Werts.	`int zufall= random.nextInt();`

Methode	Beschreibung	Beispiel
`nextInt(int bound)`	Erzeugen eines zufälligen, ganzzahligen Werts zwischen 0 (inklusive) und dem bound-Wert (exklusive).	`int zufall= random.nextInt(100);`
`nextInt(int lower, int upper)`	Erzeugen eines zufälligen, ganzzahligen Werts zwischen lower (inklusive) und dem upper-Wert (exklusive).	`int zufall= random.nextInt(5, 10);`

6.4. Turtle

Befehl	Beschreibung	Beispiele
`forward`	Bewege dich um so viele Pixel vorwärts	`forward(100)` `forward(2.5)`
`back`	Bewege dich um so viele Pixel rückwärts	`back(50)` `back(200)`
`left`	Drehe dich um so viele ° nach links	`left(90)` `left(22.5)`
`right`	Drehe dich um so viele ° nach rechts	`right(45)` `right(120)`
`penUp`	Bewege dich ab jetzt ohne zu zeichnen	`penUp()`
`penDown`	Zeichne ab jetzt wieder beim Bewegen	`penDown()`

`setPenColor`	Verwende ab jetzt eine andere Stiftfarbe. Farben werden als Strings angegeben, entweder als englischer Name, als Hex-String oder mittels RGB- oder HSL-Format.	`setPenColor("blue")`
		`setPenColor("HotPink")`
		`setPenColor("#ff6688")`
		`setPenColor("rgb(255,102,136)")`
		`setPenColor("hsl(240,100%,100%)")`

`setPenWidth`	Verwende ab jetzt eine andere Stiftbreite. Die Breite wird in Pixel angegeben; der Standardwert ist 1.	`setPenWidth(3)` `setPenWidth(0.5)`
`setSpeed`	Bewege dich ab jetzt schneller oder langsamer. Die Geschwindigkeit wird in Pixel pro Sekunde angegeben; der Standardwert ist 100.	`setSpeed(50)` `setSpeed(1000)`

6.5. Math

Math ist zwar eine Klasse, aber im prinzip einfach eine Sammlung von mathematischen Methoden.

Befehl	Beschreibung	Beispiele
`abs(int a)`	Berechnet den absoluten Wert von der Zahl a.	`int positiv= Math.abs(-4); // positiv ist dann 4`

Befehl	Beschreibung	Beispiele
`ceil(double a)`	Die Zahl a wird auf die nächste ganze Zahl aufgerundet.	`double zwei= Math.ceil(1.2);`
`floor(double a)`	Die Zahl a wird auf die nächste ganze Zahl abgerundet.	`double zwei= Math.floor(2.8);`
`max(double a, double b)`	Berechnet, welche der Zahlen a und b grösser ist.	`double a= Math.max(2.8, 5.7);`
`min(double a, double b)`	Berechnet, welche der Zahlen a und b kleiner ist.	`double a= Math.min(2.8, 5.7);`
`pow(double a, double b)`	Berechnet a^b.	`double hoch= Math.pow(3, 2); // 3 hoch 2`

7. Klassen

Klassen können gesehen werden als Baupläne für Objekte. Sie definierten Folgendes:

- Attribute: Attribute sind Variablen. Sie definieren, was im Objekt für Daten gespeichert werden können.
- Konstruktor: Wie werden Objekte erstellt. Das heisst, was muss über das Objekt bekannt sein, damit es erstellt werden kann. Konstruktoren heissen gleich wie die Klasse und haben keinen Rückgabetyp. Wie die andere Methoden können Konstruktoren überladen sein und somit mehrfach mit unterschiedlichen Parameterlisten vorkommen.
- Methoden: Die Methoden definieren die Funktionalität eines Objekts. Also was man mit dem Objekt machen kann. Die Methoden können dabei auf die Attribute zugreifen.
- Getter/Setter: Getter und Setter können die Werte von Attributen zurückgeben bzw. setzen.

```
public class Bottle {
    double capacity;  // ml
    double content;  // ml
    // Default-Konstruktor für Standardflaschen mit einem Fassungsvermögen von 100 ml.
    public Bottle() {
        capacity = 100;
        content= 0;
    }
    // Konstruktor für beliebig grosse, leere Flaschen
    public Bottle(double capacity) {
        this.capacity = capacity;
        this.content= 0;
    }
    /* Konstruktor für Flaschen mit einer beliebigen Grösse und einem beliebigen
Startinhalt
    Falls die Flasche mit unsinnigen Werten erstellt werden soll wird der Konstruktor
abgebrochen
    indem eine IllegalArgumentException geworfen wird.
     */
```

```
    public Bottle(double capacity, double content) {
        if (capacity<0 || capacity<content) {
            throw new IllegalArgumentException();
        }
        this.capacity = capacity;
        this.content= content;
    }
    // Kopierkonstruktor so kann ich eine Flasche noch einmal erstellen
    public Bottle(Bottle b){
        this.capacity= b.capacity;
        this.content= b.content;
    }
    // Methode für das Füllen der Flasche
    public void fill(double amount) {
        if (amount > 0) {
            content += amount;
            if (content > capacity) {
                content = capacity;
            }
        }
    }
    // Methode die angibt, zu wie viel Prozent die Flasche gefüllt ist.
    public double contentInPercent() {
        return (content / capacity) * 100;
    }
    // Getter für capacity
    public double getCapacity(){ return capacity;}
    // Setter für capacity mit Zusatztest auf positive Werte
    public void setCapacity(double capacity) {
        if (capacity>0){
            this.capacity= capacity;
        }
    }
    public String toString(){
        return capacity/10+"dl-Flasche zu "+contentInPercent()+"% gefüllt.";
    }
}
```

8. Objekte

Objekte werden auf Basis einer Klasse erstellt. Anschliessend können die Instanzmethoden verwendet werden.

```
public class Beispiel {

    public static void main(String[] args){
        Bottle leer= new Bottle();
        Bottle menge500ml= new Bottle(500);
        Bottle literFlasche= new Bottle(1000);
```

```
        Bottle gefuellteFlasche= new Bottle(400, 400);

        leer.fill(100);
        System.out.println(leer.contentInPercent());

        System.out.println(literFlasche.toString());
    }
}
```