Programmieren in Java

Crashkurs für die Programmierung mit Java

Vorkurs Wintersemester 2023/2024

Inhalte

1. Hello World – eine einfache Ausgabe
2. Variablen und Datentypen
3. Operatoren
4. Interaktion mit dem Nutzer
5. Bedingte Verzweigungen
6. Schleifen
7. Methoden
8. Arrays

Hello World!
und einfache Ausgaben

Wir schreiben unser erstes Programm

Hello World!

Unser erstes Programm erblickt das Licht der Welt

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

Jedes Zeichen ist wichtig!
Geschweifte, eckige und runde Klammern, Semikolons … !

Die ersten Worte

Ausgaben auf der Konsole

- `System.out.print(…)`
- Gibt das Argument in Klammern auf der Konsole aus

- `System.out.println(…)`
- Gibt das Argument in Klammern auf der Konsole aus und fügt einen Zeilenumbruch ein

Variablen und Datentypen

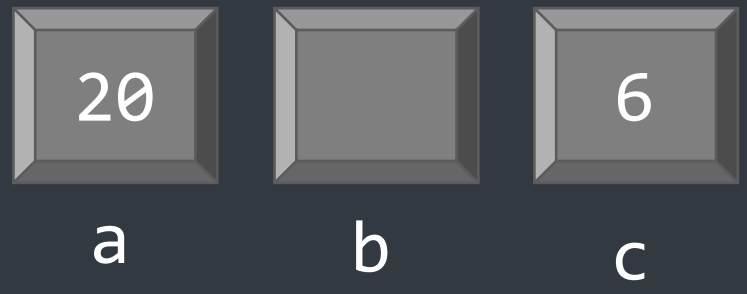
Inhalte speichern und verwenden

Variablen

Was sind Variablen?

- Variablen sind *Platzhalter* für Werte
- Variablen haben immer einen festen *Datentyp*
- Sie können dann auch nur Werte von diesem Datentyp annehmen

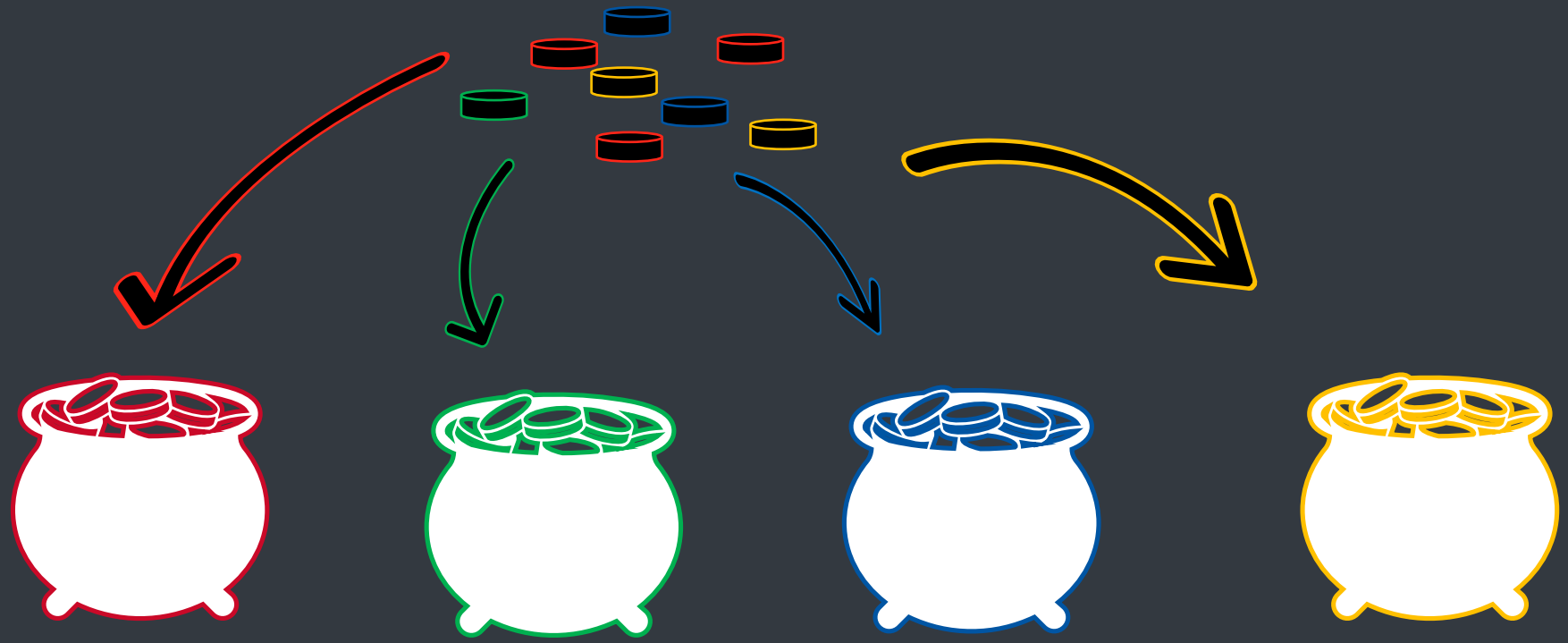
Wie in Mathe: $a^2 + b^2 = c^2$
Die Variablen sind Platzhalter für konkrete Werte.



Variablen

Was sind Datentypen?

- Jede Variable hat immer nur einen festen *Typ*
- Sie kann dann auch nur Werte dieses Typs annehmen



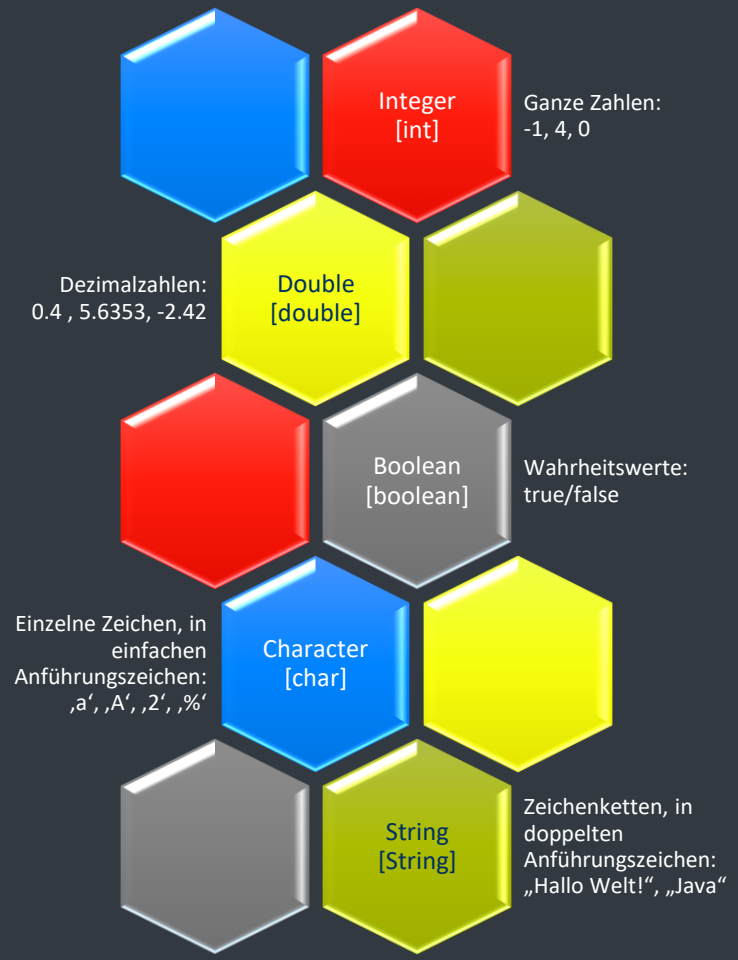
Variablen

Welche Datentypen gibt es?

Aufgabe:
Überlegen Sie, welche verschiedenen Wertetypen Sie bereits kennen!

Variablen

Welche Datentypen gibt es?



… und noch andere!

Variablen

Unsere ersten Variablen sagen hallo.

```
public class HelloVariables {
   public static void main(String[] args) {
      int anInteger = 6327;
      double aDouble = 4.7;
      boolean aBoolean = false;
      char aCharacter = 'A';
      String aString = "Hallo Welt!";

      System.out.println("Hallo von den Variablen!");
      System.out.println(anInteger);
      System.out.println(aDouble);
      System.out.println(aBoolean);
      System.out.println(aCharacter);
      System.out.println(aString);
   }
}
```

Achtung, String wird im Gegensatz zu den anderen Datentypen groß geschrieben!

Variablen

Der Sonderfall String

- String wird im Gegensatz zu den anderen bisherigen Datentypen groß geschrieben
- String ist ein *komplexer Datentyp*, kein *primitiver*
- Er ist dadurch mächtiger, und stellt *Methoden* bereit (mehr dazu später)

Variablen

Lebenszyklus

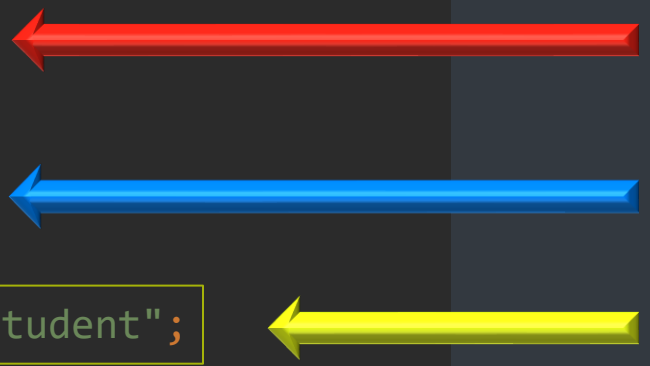
Variablen

Unterschied Deklaration und Definition

```
public class DeclarationDefinition {
   public static void main(String[] args) {
      int alter;
      double gewicht;

      alter = 25;
      gewicht = 84.0;

      String name = "A. Student";
   }
}
```



Variablen

Gültigkeitsbereich

- Variablen sind nur innerhalb ihres *Blocks* (meistens durch { } begrenzt) gültig
- Sie existieren auch in allen Blöcken *innerhalb* ihres aktuellen Blocks
- Variablennamen müssen immer *eindeutig* sein, es darf keine Duplikate geben
- Außerdem sollten Variablennamen *keine Sonderzeichen* wie Umlaute enthalten und dürfen nicht mit einer Zahl beginnen.

Wo haben wir bisher bereits einen Block verwendet?

Variablen

Was funktioniert und was nicht?

```
public class ValidVariables {
   public static void main(String[] args) {
      int alter = 28;
      String beruf = "Schauspieler";
      String name = "A. Bezeh";
      int fans = 213;

      {
         int filmJahr = 2020;
         double preis = 8.90;
         String name = "Rückkehr des Integers";

         fans = 410;
      }

      fans = 593;
      beruf = "Profi-Schauspieler";
      preis = 12.42;
      int filmJahr = 2021;
   }
}
```

Variablen

Was funktioniert und was nicht?

```
public class ValidVariables {
   public static void main(String[] args) {
      int alter = 28;
      String beruf = "Schauspieler";
      String name = "A. Bezeh";
      int fans = 213;

      {
         int filmJahr = 2020;
         double preis = 8.90;
         String name = "Rückkehr des Integers";

         fans = 410;
      }

      fans = 593;
      beruf = "Profi-Schauspieler";
      preis = 12.42;
      int filmJahr = 2021;
   }
}
```

„name“ wurde bereits deklariert!

„preis“ existiert nicht mehr!

Operatoren

Mit Variablen rechnen

Operatoren

Welche Operatoren gibt es?

Operator	Bedeutung	Verwendung
+	Addition oder Konkatenation (bei Strings)	`int a = b+c;` `String text2 = text1+“!“;`
-	Subtraktion	`int a = b-c;`
*	Multiplikation	`int a = b*c;`
/	Division bzw. Ganzzahldivision	`double d = e/f;` `int a = b%c;`
%	Modulo (Rest einer Division)	`int rest = a%b;`
++	Inkrement um 1, kurz für `a = a+1;`	`a++;`
--	Dekrement um 1, kurz für `a = a-1;`	`a--;`

Operatoren

Modulo im Detail

- Modulo gibt den *Rest* einer Ganzzahl-Division zurück

```
17 : 5 = 3 Rest 2

17 % 5 => 2

17 / 5  => 3
```

Wie kann man mit Modulo herausfinden, ob eine Zahl gerade oder ungerade ist?

Operatoren

Richtig dividieren

- Eine Ganzzahl-Division selbst (also int geteilt durch int) ergibt auch wieder eine *Ganzzahl*!
- Ist als Ergebnis eine Dezimalzahl gewünscht, muss zumindest der Dividend eine Dezimalzahl sein!

```
double b = 12.0;
int c = 5;
double a = b/c;
```

Operatoren

Was gibt dieses Programm aus und warum?

```java
public class Operators {
   public static void main(String[] args) {
      int a = 35;
      double b = 3.0;
      double c = 16.5;
      int d = 8;

      double ergebnis1 = a-b;
      double ergebnis2 = b*c;
      double ergebnis3 = a/c;
      double ergebnis4 = a/b;
      int ergebnis5 = a/d;
      int ergebnis6 = a%d;

      System.out.println("ergebnis1 = " + ergebnis1);
      System.out.println("ergebnis2 = " + ergebnis2);
      System.out.println("ergebnis3 = " + ergebnis3);
      System.out.println("ergebnis4 = " + ergebnis4);
      System.out.println("ergebnis5 = " + ergebnis5);
      System.out.println("ergebnis6 = " + ergebnis6);
   }
}
```

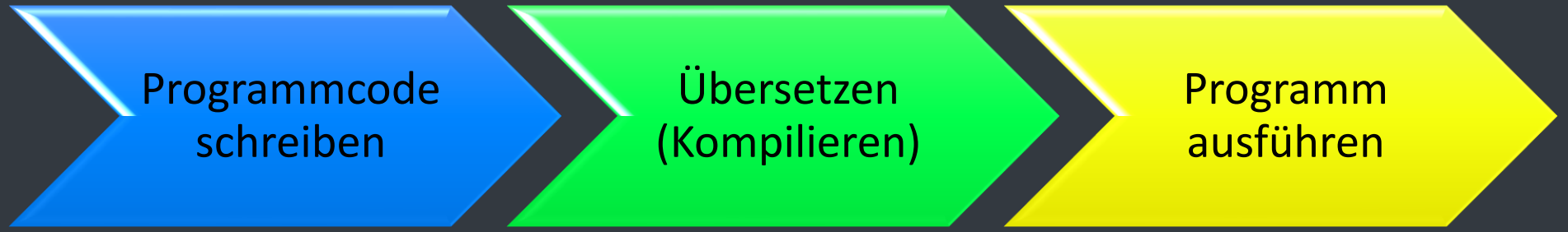
Interaktion mit dem Nutzer

Wie man Eingaben zur Laufzeit verwertet

Eingaben

Unterschied zwischen Kompilierzeit und Laufzeit

- Alles, was wir fest in unseren *Programmcode* eingeben, wird in *Maschinencode* übersetzt
- Wird das Programm ausgeführt, so werden diese festen Werte verwendet
- Damit bei der Ausführung des Programms Werte von extern verwendet werden können, müssen wir Eingaben des Benutzers möglich machen
- Diese Werte werden dann auch erst bei der *Ausführung* des Programms *ausgewertet*, es können also *keine Annahmen* darüber getroffen werden



Eingaben

Andere Arten von Eingaben

Aufgabe:

Überlegen Sie, welche anderen Arten von Werten erst zur Laufzeit bekannt sind.
Gibt es andere Orte, außer der Tastatur, von der Eingaben kommen könnten?

Eingaben

Wir sagen unserem Programm Hallo

```java
import java.util.Scanner;

public class HelloProgram {
    public static void main(String[] args) {
        Scanner myScanner = new Scanner(System.in);

        System.out.println("Hallo, Mensch!");
        String eingabe = myScanner.nextLine();
    }
}
```

Die import-Zeile steht noch oberhalb der Klassenzeile und wird benötigt, damit der Computer weiß, welchen Scanner wir meinen.

„myScanner“ ist ein beliebiger Variablenname. Diese Zeile wird nur einmal benötigt, danach können wir beliebig viel damit einlesen.

An dieser Stelle wird dann bei der Programmausführung auf die Eingabe des Benutzers gewartet

Eingaben

Methoden des Scanners

- Je nach gewünschtem Datentyp der Eingabe bietet der Scanner verschiedene *Methoden*

Methode	Datentyp	Verwendung
`nextLine()`	`String`	Liest eine gesamte Zeile ein
`next()`	`String`	Liest ein Wort (bis zum Leerzeichen) ein
`nextInt()`	`int`	Liest eine Ganzzahl ein
`nextDouble()`	`double`	Liest eine Dezimalzahl ein Achtung: Die Eingabe ist landestypisch. Hier wird die Dezimalzahl im Gegensatz zum Code dann mit Komma und nicht mit Punkt geschrieben

Bedingte Verzweigungen

Variablen vergleichen und Fallunterscheidungen treffen

Bedingte Verzweigungen

Wozu braucht man Verzweigungen?

Aufgabe:
Überlegen Sie sich, weshalb Fallunterscheidungen in einem guten Programm notwendig sind

Bedingte Verzweigungen

Die Einweg-Verzweigung

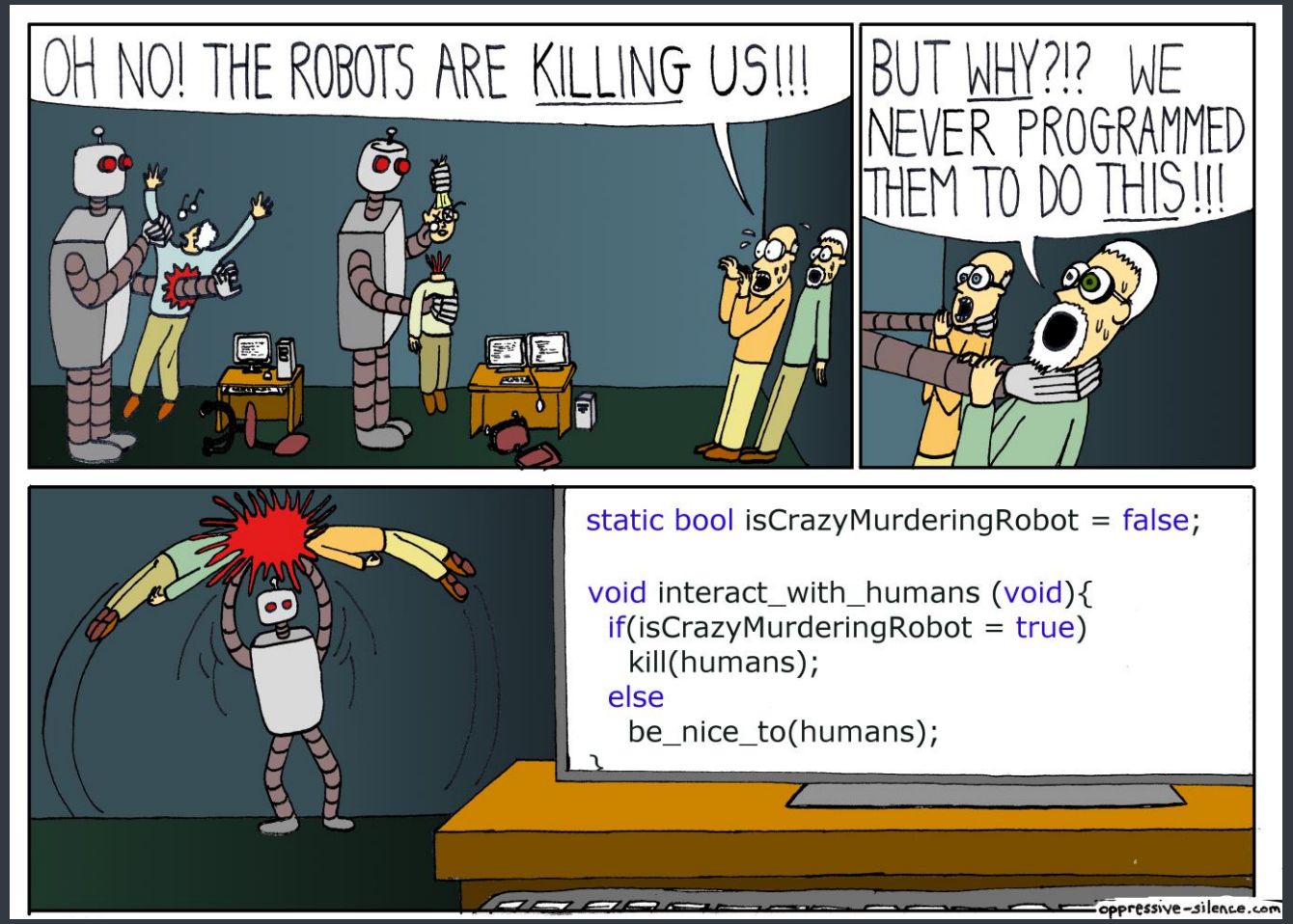
```
public class SimpleIf {
   public static void main(String[] args) {
      boolean esRegnet = true;
      if(esRegnet == true){
         System.out.println("Es regnet!");
      }

      int zahl = 5;
      if(zahl > 10){
         System.out.println("Die Zahl ist größer als 10!");
      }
   }
}
```

Wieso muss ein Vergleich immer mit == und nicht mit = geschrieben werden?

Bedingte Verzweigungen

== für den Vergleich, = für die Zuweisung.



http://oppressive-silence.com/comic/oh-no-the-robots

Bedingte Verzweigungen

Die Zweiwege-Verzweigung

```
public class IfElse {
   public static void main(String[] args) {
      int zahl = 4;
      if(zahl >= 5){
         System.out.println("Die Zahl ist größer oder gleich 5!");
      }else{
         System.out.println("Die Zahl ist kleiner 5!");
      }
   }
}
```

Wenn die Bedingung des If-Teils nicht erfüllt ist, dann wird der else-Zweig ausgeführt.

Bedingte Verzweigungen

Die Mehrwege-Verzweigung

```
public class MultipleIfs {
   public static void main(String[] args) {
      int zahl = 8;
      if(zahl < 5){
         System.out.println("Die Zahl ist kleiner als 5!");
      }else if(zahl > 5){
         System.out.println("Die Zahl ist größer als 5!");
      }else{
         System.out.println("Die Zahl ist gleich 5!");
      }
   }
}
```

Hiermit können beliebig viele Bedingungen *der Reihe nach* abgefragt werden.

Bedingte Verzweigungen

Bedingungen und Vergleiche

Vergleichsoperator	Bedeutung	Beispiel
`==`	Prüfen auf Gleichheit	`if(a == 3)`
`<, >, <=, >=`	Größer (gleich), kleiner (gleich)	`if(a>=4)`
`!=`	Ungleich	`if(a!=2)`
`!(…)`	Negation der Klammer, `true` wird zu `false` und `false` zu `true`	`if(!(a<3))`
`.equals(…)`	(nur bei Strings und anderen komplexen Datentypen): Prüfen auf Gleichheit	`if(myString.equals(„Hallo“))`

Bedingte Verzweigungen

Logische Verknüpfungen

UND – Verknüpfung: &&

- Nur erfüllt, wenn *beide* Seiten wahr sind
- Ist eine oder beide Seiten falsch, so ist der gesamte Ausdruck falsch

```
if((a<10) && (a>5)){
    ...
}
```

ODER – Verknüpfung: ||

- *Mindestens eine* der beiden Seiten muss wahr sein
- Es können auch beide Seiten wahr sein

```
if((a==4) || (a==5)){
    ...
}
```

Bedingte Verzweigungen

Was ist der Unterschied zwischen den beiden Programmen?

```java
boolean istHungrig = false;
boolean istDurstig = true;

if (istHungrig == true) {
   System.out.println("Iss etwas!");
}

if (istDurstig == true) {
   System.out.println("Trink etwas!");
}
```

```java
boolean istHungrig = false;
boolean istDurstig = true;

if (istHungrig == true) {
   System.out.println("Iss etwas!");

   if (istDurstig == true) {
        System.out.println("Trink etwas!");
   }
}
```

Bedingte Verzweigungen

Was ist der Unterschied zwischen den beiden Programmen?

```java
boolean istHungrig = false;
boolean istDurstig = true;

if (istHungrig == true) {
   System.out.println("Iss etwas!");
}

if (istDurstig == true) {
   System.out.println("Trink etwas!");
}
```

Beide Bedingungen werden der Reihe nach ausgewertet

```java
boolean istHungrig = false;
boolean istDurstig = true;

if (istHungrig == true) {
   System.out.println("Iss etwas!");

   if (istDurstig == true) {
        System.out.println("Trink etwas!");
   }
}
```

Die zweite Bedingung wird nur ausgewertet, wenn die erste wahr ist!

Schleifen

Wie man mit wenigen Codezeilen dem Computer viel Arbeit gibt

Schleifen

Alle Studenten begrüßen

Aufgabe:

Schreiben Sie ein Programm, das 60 Studenten begrüßt. Die Studenten sind hierbei von 1-60 durchnummeriert.

Schleifen

Alle Studenten begrüßen – bisherige Lösung

```
public class HelloStudentsIterative {
   public static void main(String[] args) {
      System.out.println("Hallo, Student 1!");
      System.out.println("Hallo, Student 2!");
      System.out.println("Hallo, Student 3!");
      System.out.println("Hallo, Student 4!");
      // ...
      System.out.println("Hallo, Student 60!");
   }
}
```

Sehr mühsam!
Und was, wenn der Begrüßungstext später einmal geändert werden soll?
Er müsste dann an 60 Stellen angepasst werden.

Schleifen

Was sind Schleifen allgemein?

- Eine Schleife ist ein Konstrukt, das *sich wiederholende Codeabschnitte* zusammenfasst und kompakt darstellt
- Der Schleifeninhalt wird dabei so lange wiederholt, bis eine definierte *Bedingung* eingetreten ist
- Alle Schleifen sind gleich *mächtig*, das bedeutet, dass sie alle dieselben Programme formulieren können

Schleifen

Die While-Schleife

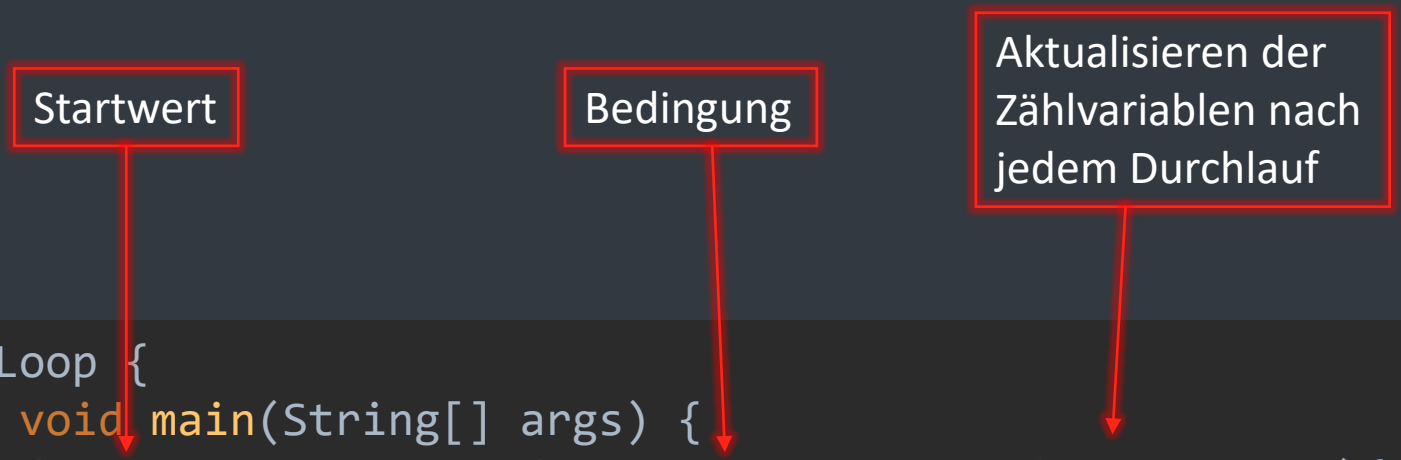
```
public class WhileLoop {
   public static void main(String[] args) {
      int studentNummer = 1;
      while(studentNummer <= 60){
         System.out.println("Hallo, Student "+studentNummer);
         studentNummer++;
      }
   }
}
```

Solange die Bedingung gilt, wird der Block ausgeführt. Achten Sie darauf, die Bedingung auch zu aktualisieren, da man sonst leicht eine Endlosschleife programmiert.

Erinnerung:
`studentNummer++;`
ist das gleiche wie
`studentNummer = studentNummer+1;`

Schleifen

Die for-Schleife




```
public class ForLoop {
   public static void main(String[] args) {
      for(int studentNummer = 1; studentNummer<=60; studentNummer++){
         System.out.println("Hallo, Student "+studentNummer);
      }
   }
}
```

Schleifen

While-Schleife vs For-Schleife

`while`

- Eignet sich gut für komplexere logische Bedingungen
- Wird oft verwendet, wenn die genaue Anzahl der Wiederholungen *unbekannt* ist
- Die Bedingung kann in einem *anderen Programmabschnitt* aktualisiert werden

`for`

- Gut für eine *feste und bekannte Anzahl* an Wiederholungen
- Oft zum Iterieren (=Durchlaufen) von *Arrays* (dazu später mehr) verwendet
- Die Zählvariable sollte nur *innerhalb* der Schleife aktualisiert werden

Aufgabe

Formulieren Sie dieses Programm als While-Schleife

```
public class TransformLoops {
   public static void main(String[] args) {
      for(int i = 1; i<=32;i=i*2){
         System.out.println(i);
      }
   }
}
```

Was ist das besondere an den ausgegeben Zahlen?

Methoden aufrufen

Wie wir komplexen Datentypen Aufgaben erteilen können

Methoden aufrufen

Was ist eine Methode?

- Viele komplexe Datentypen stellen *Methoden* bereit
- Es handelt sich hierbei um *aufrufbare Funktionen*, die bestimmte Dinge erledigen
- *Primitive Datentypen* (`int`, `double`, `char`, `boolean` …) stellen selbst *keine* Methoden zur Verfügung
- Methoden können auch selbst definiert werden
- Eine Methode kann einen Wert eines bestimmten Typs *zurückgeben*
- Manchen Methoden muss man *Argumente* übergeben

Wie in Mathe: y = f(x)
Die Funktion gibt einen Wert zurück und benötigt selbst ein Argument.

Methoden aufrufen

Das Beispiel String

```
public class StringMethods {
   public static void main(String[] args) {
      String text = "Hallo Welt, ich lerne gerade Javaprogrammierung.";
      char buchstabe = text.charAt(6);
      int laenge = text.length();
      String klein = text.toLowerCase();
      String gross = text.toUpperCase();
      String ausruf = text.replace('.','!');
      boolean istGleich = text.equals("Hier spricht Python.");
      boolean enthaelt = text.contains("Java");
   }
}
```

Methoden aufrufen

Methoden von String

Methode	Wirkung
`length()`	Gibt die **Anzahl** der Zeichen (auch Leerzeichen) des Strings zurück
`charAt(index)`	Gibt das Zeichen an der **Stelle** zurück (erstes Zeichen hat den index 0)
`toLowerCase()`	Wandelt alle Buchstaben in **Kleinbuchstaben** um, und gibt einen neuen String zurück
`toUpperCase()`	Gibt einen neuen String zurück, bei dem alle Buchstaben **großgeschrieben** sind
`replace(old, new)`	**Ersetzt** alle Vorkommen des chars durch einen neuen char
`equals(text)`	Prüft, ob beide Strings **identisch** sind
`contains(text)`	Prüft, ob der String einen anderen String **enthält**

Wenn Sie nach der Eingabe des Punktes kurz warten, zeigt Ihnen die IDE alle Methoden an, die Sie aufrufen können!

Methoden aufrufen

Methoden des Scanners

Aufgabe:
Finden Sie heraus, welche andere Methoden `Scanner` hat!
Hierzu benötigen Sie neben dem Import wieder diese Zeile:

```
Scanner myScanner = new Scanner(System.in);
```

Arrays

Ein Regal mit vielen Paketen

Arrays

Herkömmliche Implementierung

Aufgabe:
Schreiben Sie ein Programm, das alle Vornamen Ihres Kurses speichert, und jeden Namen begrüßt.

Arrays

Was ist ein Array?

- Arrays sind *Felder*, die mehrere Werte *eines bestimmten Datentyps* aufbewahren
- Man kann Sie sich wie Regale vorstellen, in denen viele Werte-Pakete stehen
- Die Größe ist *begrenzt* und muss *zu Beginn* festgelegt werden
- Die Nummerierung der einzelnen Fächer beginnt bei *0*
- Einzelne Fächer können auch *leer* sein

Arrays

Wie werden Arrays formuliert?

```
public class Arrays {
    public static void main(String[] args) {
        int[] zahlen = new int[6];
        zahlen[0] = 9;
        zahlen[1] = 3;
        zahlen[3] = 1;
        zahlen[5] = 14;
    }
}
```

Bei der Definition des Arrays muss gleich auch die Größe angegeben werden

Die einzelnen Elemente werden über ihren Index angesprochen. Dieser geht von `0` bis `(size-1)`

Arrays

Ein Array mit einer for-Schleife durchiterieren

```
public class ArrayFor {
   public static void main(String[] args) {
      double[] zahlen = new double[4];
      zahlen[0] = 4.5;
      zahlen[1] = 1.3;
      zahlen[2] = zahlen[0]*zahlen[1];
      zahlen[3] = zahlen[2]-zahlen[1];

      for(int i = 0;i<zahlen.length;i++){
         System.out.println(zahlen[i]);
      }
   }
}
```

Die for-Schleife zählt von `0` solange, wie der Index kleiner als die Größe (`length`) des Arrays ist

Sollten Sie bei der Programmausführung eine `ArrayIndexOutOfBounds Exception` erhalten, so haben Sie einen Index angefordert, der nicht existiert!

Arrays

Die Aufgabe vom Anfang intelligent gelöst

Aufgabe:
Definieren Sie ein Array, das alle Vornamen Ihres Kurses enthält. Begrüßen Sie anschließend jeden Namen mithilfe einer for-Schleife.

Sie können Arrays auch in einer Zeile initialisieren:

```
int[] values = new int[]{5,1,20,2};
```

Die Größe des Arrays wird hierbei dann nicht angegeben.