Autor: Christine Janischek
Version:1.0

Wirtschaftsinformatik

Unterrichtsdokumentation

Modulname:	Softwareentwicklung in Java

Stand: 30. Aug 2020

BSW Berufliches Schulzentrum Wangen

Inhaltsverzeichnis

1 Modell: Grundlagen der OOP....3
1.1 Objekte und Klassen....3
1.2 Grundgerüst einer Klasse....5
1.3 Übung: Die Fachklasse Geldkarte....7
1.4 Übung: Die Fachklasse Bmirechner....8
1.5 Übung: Die Fachklasse Taschenrechner....10
1.6 Übung: Die Fachklasse Währungsrechner....11
1.7 Überblick zu Primitiven Datentypen....13
1.8 Assoziationen....14

2 View: Benutzeroberflächen (Grafical User Interface, GUI)....15
2.1 Am Anfang steht ein Entwurf der benötigten Benutzeroberflächen....15
2.2 Einführung in die GUI-Programmierung....16

3 Algorithmik....17
3.1 Kontrollstrukturen....17
3.2 Einfache Datenstrukturen und Komplexe Datentypen....21
3.3 Vererbung....24

4 Unterrichtsbeispiele....27
4.1 Passwortgenerator....27
4.2 Wortspiele....29
4.3 Dienstwagennutzung....33

5 Projekt....39
5.1 Anforderungen....39
5.2 Projekt-Beispiele....39
5.3 Hinweise....40

6 Mündliche Prüfung....41

Datenbankstruktur optimieren:....42

7 Leistungskontrollen....44
7.1 Kurzarbeit 01....44
7.2 Kurzarbeit 02....45
7.3 Klassenarbeit 01....47
7.4 Kurzarbeit 03....52
7.5 Klassenarbeit 02....54
7.6 Kurzarbeit 04....58

1 Modell: Grundlagen der OOP

1.1 Objekte und Klassen

Klasse	Die Klasse ist ein Muster, eine Vorlage die eine ganze Menge an Objekten mit ihren Eigenschaften und Verhaltensweisen beschreibt (definiert = deklariert). Beispiel: Die Klasse Produkt beschreibt eine ganze Menge an möglichen Produkten die erzeugt werden können und denen dann Werte, also spezifische Eigenschaften zugewiesen werden können. Die Zuweisung expliziter Eigenschaftswerte erfolgt über die SET-Methode. `produkt1.setBezeichnung("iPhone 6plus");`

Objekt/
Assoziation
	Ein Objekt ist die Verwendung der Klasse für die konkrete Ausstattung einer Sache, eines „Dings". Die Eigenschaften bekommen konkreten Werte und die Verhaltensweisen können auf die Dinge angewendet werden. Der Methodenaufruf (Verhaltensweise) erfolgt immer am konkreten Objekt (z.B. am produkt1)
	
	Objekte erzeugen
	`//Erzeuge ein Objekt der Klasse Geldautomat`
	`Geldautomat automat1 = new Geldautomat();`
	
	Objekte nutzen
	`produkt1.produktAnzeigen();`

Datentyp	Der Datentyp legt die Art des Wertes fest der verarbeitet werden soll und reserviert den Speicherplatz für den Wert.
Zugriffsmodifikator	Werden für die Nutzung des Rechtesystems in Java genutzt. Beispiele: **public:** steht für öffentlich, von außen sichtbar und zugänglich. **private:** steht für privat, von außen nicht zugänglich (von außerhalb der Klasse).
Attribute → Attributname → Attributwert	Attributnamen sind Eigenschaften die eine Sache, ein Ding, nähe bezeichnen, mit konkreten Werten versehen. Sie werden in der Klasse selbst mit einem Datentyp und einem Namen ausgestattet. Dieser Vorgang heißt **Deklaration**. Ein oft verwendetes Synonym heißt Variable. Wird einer Eigenschaft ein

	konkreter Wert zugewiesen ergibt sich der Attriubutwert. Attributname: bezeichnung Attributwert: "iPhone 6plus"

Konstruktor	Der Default (Standard) Konstruktor ist parameterlos und leer!
	
	Deklaration in der Klasse selbst:
	`//Konstruktor (Standard, Default)`
	`public Geldautomat(){`
	`}`
	
	Wird benötigt um Objekte der Klasse zu erzeugen:
	`//Erzeuge ein Objekt der Klasse Geldautomat`
	`Geldautomat automat1 = new Geldautomat();`

Methoden	Methoden sind Verhaltensweisen die beschreiben wie etwas, eine Aufgabe, erledigt werden soll. Diese Beschreibung nennt sich auch **Implementierung**.

Main-Methode	`public static void main(String[] args ){`
	`}`
	
	Ist Startpunkt einer Java-Anwendung.

Kontrollstrukturen	Fester Bestandteil aller Programmiersprachen und damit fundamentales Wissen für jeden Programmierer. Dient der Implementierung von Business Logic → Fallunterscheidungen, Wiederholstrukturen → Automatisierung von Prozessen.
Prinzipien der Objekt-orientierten Programmierung (OOP)	1. Wiederverwendbarkeit 2. Vererbung 3. Kapselung 4. Zerlegung 5. Sicherheit 6. Polymorphie 7. Erweiterbarkeit 8. Machbarkeit → Testbarer Quellcode 9. Wiederholungen vermeiden 10. Trennung (MVC) von Modell – View - Controller

1.2 Grundgerüst einer Klasse

<table>
<tr><td>… den Quellcode (die Anweisung) für die Deklaration einer Klasse implementieren.
→ Großschreiben
→ Singular
→ jede Klasse in eine eigene Datei
→ identisch mit dem Dateiname</td><td><pre>public class Geldautomat{

}</pre></td></tr>
<tr><td>… die Datentypen für ganze Zahlen, Kommazahlen und Zeichenketten nennen.</td><td><pre>int, double, String</pre></td></tr>
<tr><td>… den Quellcode (die Anweisungen) um Variablen/Attribute zu deklarieren, implementieren.</td><td><pre>private double guthaben;
private String meldung;
private int ustSatz;</pre></td></tr>
<tr><td>… den Quellcode (die Anweisungen) um den Standardkonstruktor einer Klasse zu deklarieren.</td><td><pre>//Konstruktor (Standard, Default)
public Geldautomat(){
}</pre></td></tr>
<tr><td>… den Quellcode (die Anweisungen) für den Getter (Get-Methode) eines Attributs/Variablen implementieren.</td><td><pre>Getter für betrag:
public double getBetrag(){
  return this.betrag;
}

Getter für meldung:
public String getMeldung(){
  return this.meldung;
}</pre></td></tr>
<tr><td>… den Quellcode (die Anweisungen) für den Setter (Set-Methode) eines Attributs/Variablen implementieren.</td><td><pre>Setter für betrag:
public void setBetrag(double pBetrag){
  this.betrag = pBetrag;
}

Setter für meldung:
public void setMeldung(String pMeldung)
{
  this.meldung = pMeldung;
}</pre></td></tr>
<tr><td>… den Quellcode (die Anweisungen) für Methoden die einfache Berechnungen enthalten implementieren.

Rechenoperatoren:
<table>
<tr><td>Addieren</td><td>+</td></tr>
<tr><td>Subtrahieren</td><td>-</td></tr>
<tr><td>Dividieren</td><td>/</td></tr>
<tr><td>Multiplizieren</td><td>*</td></tr>
</table></td><td><pre>public void bezahlen(){
  this.guthaben = this.guthaben
                  - this.betrag;
}</pre></td></tr>
<tr><td>… den Quellcode (die Anweisungen) für Metho-</td><td><pre>public void bezahlen(){</pre></td></tr>
</table>

den die einfache Fallunterscheidungen (Kontrollstruktur → IF-ELSE) implementieren.	`if (this.guthaben >= this.betrag) { this.guthaben = this.guthaben - this.betrag; this.meldung = "Ihr neues Guthaben beträgt " + this.guthaben; } else{ this.meldung = "Zu geringes Guthaben!"; } }`

… den Quellcode (die Anweisungen) für die Erzeugung eines Objektes einer Klasse implementieren.	`// Erzeuge ein Objekt der Klasse // Geldautomat`
	
	`Geldautomat automat1 = new Geldautomat();`

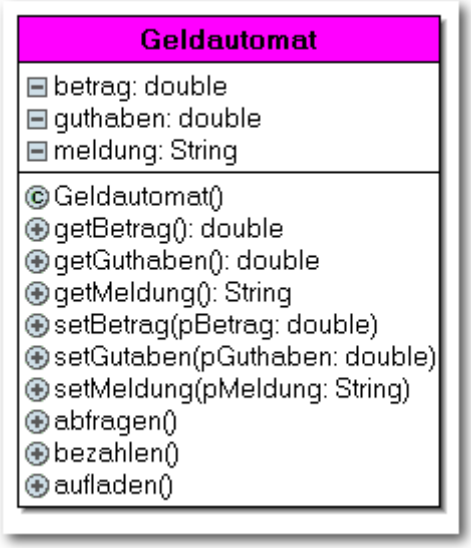
… den Quellcode (die Anweisungen) um Werte an ein bestehendes Objekt zu übermitteln, implementieren.	`automat1.setGutaben(150); automat1.setBetrag(25.0);`
… den Quellcode (die Anweisungen) um Werte eines bestehenden Objektes zu ermitteln, implementieren.	`automat1.getMeldung(); automat1.getGuthaben();`
… den Quellcode (die Anweisungen) für den Methodenaufruf eines bestehenden Objektes implementieren.	`automat1.abfragen();`
… den Quellcode (die Anweisungen) für die Erzeugung einer Ausgabe auf der Konsole implementieren.	`System.out.println("Meldung: " + automat1.getMeldung());`

… das EVA-Prinzip anhand der Main-Methode einer Startklasse erklären.	`//Eingabe: Wert an Objekt der Fachklasse übermitteln automat1.setGutaben(10.00); automat1.setBetrag(50.0);`
	
	`//Verarbeitung: Wert verarbeiten z.b. berechnen, prüfen automat1.abfragen();`
	
	`//Ausgabe: Erzeugt eine Ausgabe des Ergebnisses auf der Konsole System.out.println("Meldung: " + automat1.getMeldung()); System.out.println("Guthaben: " + automat1.getGuthaben());`

… Testfälle für eine bestehende Fachklasse for-	→ Unit tests

mulieren.	//##### IF-Fall testen //##### ELSE-Fall testen

1.3 Übung: Die Fachklasse Geldkarte



Fachklasse: Geldautomat

Methode abfragen():

```
public void abfragen(){

  this.meldung = "Ihr aktuelles Guthaben beträgt:" + this.guthaben + "!";
}
```

Methode bezahlen():

```
public void bezahlen(){

  if (this.guthaben >= this.betrag) {
    this.guthaben = this.guthaben - this.betrag;
    this.meldung = "Ihr neues Guthaben beträgt "+ this.guthaben;
  } else{
    this.meldung = "Zu geringes Guthaben!";
  }

}
```

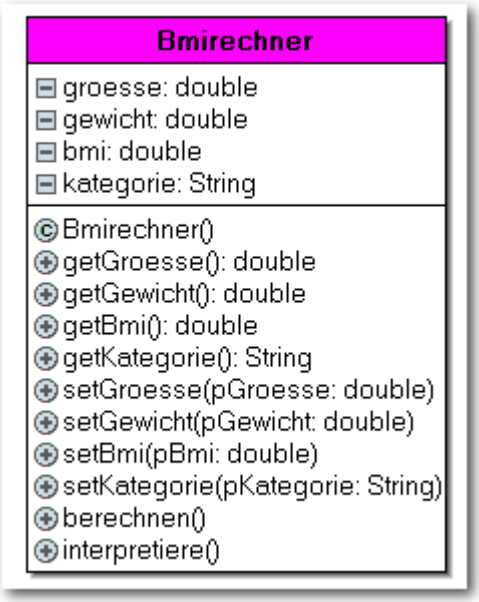
Methode aufladen():

```
public void aufladen(){
  if ((this.guthaben + this.betrag) > 200) {
    String newline = "\n";
    this.betrag = 200 - this.guthaben;
    this.meldung = "Ihr Guthaben würde den Maximalbetrag von 200 Euro überschreiten!"+newline;
    this.meldung = this.meldung + "Sie können maximal einen Betrag von "+ this.betrag + " aufladen!";
  }else{
    this.guthaben = this.guthaben + this.betrag;
    this.meldung = "Ihr neues Guthaben beträgt: "+this.guthaben;
  }
```

Geldautomat.java | Startklasse.java

```java
public class Startklasse{
  public static void main(String[]  args ){
    //Erzeuge ein Objekt der Klasse Geldautomat
    Geldautomat automat1 = new Geldautomat();

    //##### IF-Fall testen
    System.out.println("###### IF-Fall testen");

    //Eingabe: Wert an Objekt der Fachklasse übermitteln
    automat1.setGutaben(150);
    automat1.setBetrag(25.0);

    //Verarbeitung: Wert verarbeiten z.b. berechnen, prüfen
    automat1.abfragen();

    //Ausgabe: Erzeut eine Ausgabe des Ergebnisses auf der Konsole
    System.out.println("Meldung: " + automat1.getMeldung());
    System.out.println("Guthaben: " + automat1.getGuthaben());

    //##### ELSE-Fall testen
    System.out.println("###### ELSE-Fall testen");

    //Eingabe: Wert an Objekt der Fachklasse übermitteln
    automat1.setGutaben(10.00);
    automat1.setBetrag(50.0);

    //Verarbeitung: Wert verarbeiten z.b. berechnen, prüfen
    automat1.abfragen();

    //Ausgabe: Erzeut eine Ausgabe des Ergebnisses auf der Konsole
    System.out.println("Meldung: " + automat1.getMeldung());
    System.out.println("Guthaben: " + automat1.getGuthaben());

  }
}
```

1.4 Übung: Die Fachklasse Bmirechner



Fachklasse: Bmirechner

Methode berechne():

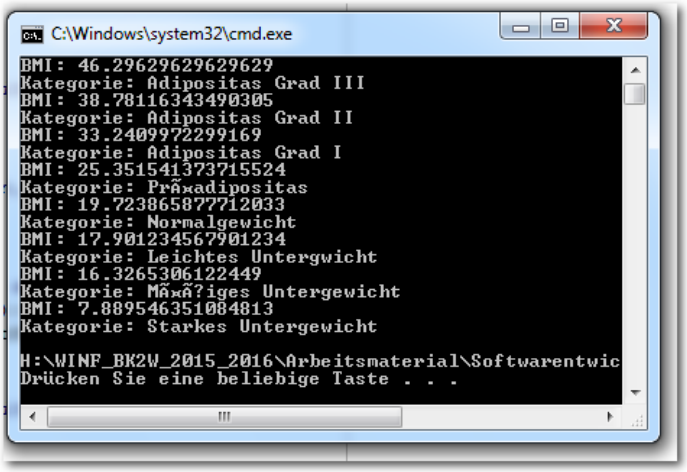
```java
public void berechnen(){

  this.bmi = this.gewicht / (this.groesse * this.groesse);
}
```

Methode interpretiere():

```
public void interpretiere(){

  if (this.bmi >= 40) {
    this.kategorie = "Adipositas Grad III";
  }else if(this.bmi >= 35 && this.bmi < 40){

    this.kategorie = "Adipositas Grad II";
  }else if(this.bmi >= 30 && this.bmi < 35){

    this.kategorie = "Adipositas Grad I";
  } else if(this.bmi >= 25 && this.bmi < 30){

    this.kategorie = "Präadipositas";
  } else if(this.bmi >= 18.5 && this.bmi < 25){

    this.kategorie = "Normalgewicht";
  } else if(this.bmi >= 17 && this.bmi < 18.5){

    this.kategorie = "Leichtes Untergwicht";
  } else if(this.bmi >= 16 && this.bmi < 17){
    this.kategorie = "Mäßiges Untergewicht";

  } else{
    this.kategorie = "Starkes Untergewicht";
  }
}
```

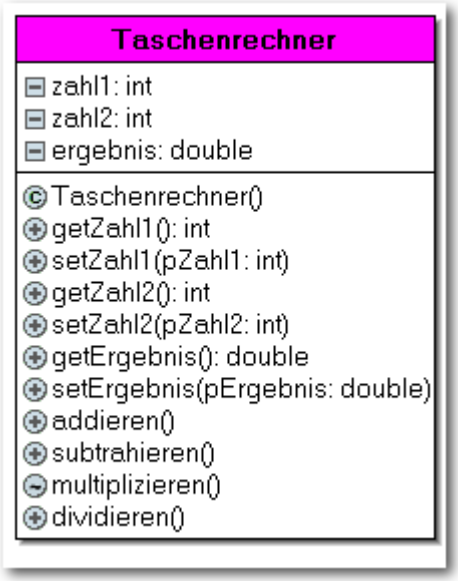
Startklasse mit Main-Methode

Prüfe 8 mögliche Fälle in einem Unit-Test:



Konsole

1.5 Übung: Die Fachklasse Taschenrechner



Methode addieren():

```
47   public void addieren(){
48     this.ergebnis = this.zahl1 + this.zahl2;
49   }
```

Methode subtrahieren():

```
51   public void subtrahieren(){
52     this.ergebnis = this.zahl1-this.zahl2;
53   }
```

Methode multiplizieren():

```
public void multiplizieren(){
  this.ergebnis = this.zahl1*this.zahl2;
}
```

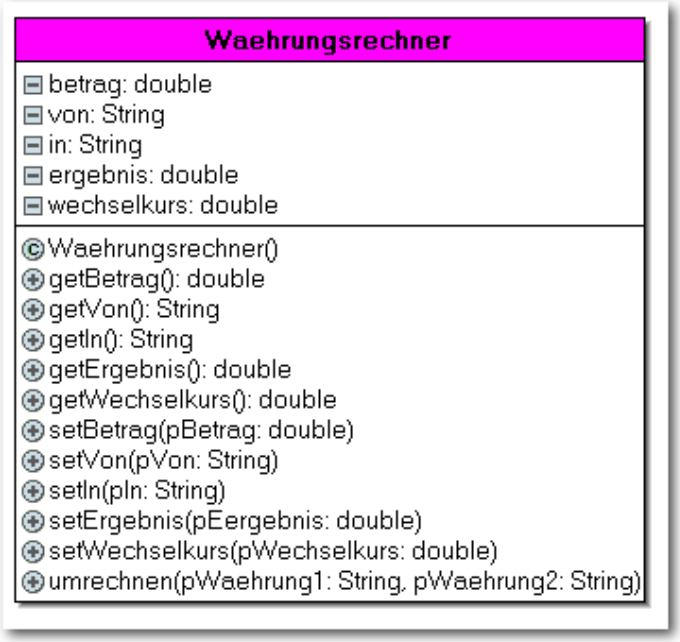
Methode dividieren():

```
public void dividieren() {
  if (this.zahl2 == 0) {
    this.meldung = "Divisionj durch null nicht möglich!";
  } else{

    this.ergebnis = this.zahl1/this.zahl2;
    }
}
```

Auszug: Unit-Test Division

```
//#####
System.out.println("##### dividieren:IF Fall");

//Eingabe: Wert an objekt der Klasse übermitteln
rechner1.setZahl1(150);
rechner1.setZahl2(0);

//Verarbeitung: Wert verarbeiten z.B. berechnen
rechner1.dividieren();

//Ausgabe: Ergebnis ausgeben
System.out.println("Ergebnis: "+ rechner1.getErgebnis());
System.out.println("Meldung: "+ rechner1.getMeldung());

//#####
System.out.println("##### dividieren:Else Fall");

//Eingabe: Wert an objekt der Klasse übermitteln
rechner1.setZahl1(150);
rechner1.setZahl2(50);

//Verarbeitung: Wert verarbeiten z.B. berechnen
rechner1.dividieren();

//Ausgabe: Ergebnis ausgeben
System.out.println("Ergebnis: "+ rechner1.getErgebnis());
```

1.6 Übung: Die Fachklasse Währungsrechner



Parameterlose Methode umrechnen():

```
public void umrechnen (){
  this.meldung = "Nicht Else";
  if ((this.von.equals( "Euro")) && (this.in.equals( "Yuan"))) {
    this.wechselkurs = 7.19422;
  } else if ((this.von.equals("Euro")) && (this.in.equals("Dollar"))) {
    this.wechselkurs = 1.13514;
```

...

```
  } else {
    this.meldung = "ELSE FALL";
  }
  this.ergebnis = this.betrag * this.wechselkurs ;
}
```

Parameterbehaftete Methode umrechnen(String pWaehrung1, String pWaehrung2):

```
63    // Höhere Methode: Methoden die mehr können als nur er- und übermitteln
64    public void umrechnen(String pWaehrung1, String pWaehrung2){
65        if(pWaehrung1.equals("Euro (EUR)") && pWaehrung2.equals("Euro (EUR)")){
66            //Euro(EUR) - Euro(EUR)
67            this.wechselkurs = 1.00000;
68            this.setVon(pWaehrung1);
69            this.setIn(pWaehrung2);
70        }else if(pWaehrung1.equals("Euro (EUR)") && pWaehrung2.equals("Britisches Pfund (GBP)")){
71            //Euro (EUR) - Britisches Pfund (GBP)
72            this.wechselkurs = 0.72085;
73            this.setVon(pWaehrung1);
74            this.setIn(pWaehrung2);
```

...

```
    }else{
        this.wechselkurs =    0.00000;
    }
    this.ergebnis =  Math.round((this.betrag * this.wechselkurs)*100)/ 100;
}
```

→ 16 Fälle müssen auf diese Weise behandelt werden.

Ausschnitt UNIT-Test, mit Parameter:

```
System.out.println("#### Dollar - Japanischer Yen (JPY)");

//Eingabe: Wert an Objekt der Klasse übermitteln
rechner1.setBetrag (1);

//Verarbeitung: Wert verarbeiten z.B. berechnen
rechner1.umrechnen("Japanischer Yen (JPY)","US Dollar (USD");


//Ausgabe: Ergebnis ausgeben
System.out.println("Betrag: " + rechner1.getBetrag());
System.out.println("Wechselkurs: " + rechner1.getWechselkurs());
System.out.println("Von: " + rechner1.getVon());
System.out.println("In: " + rechner1.getIn());
System.out.println("Ergebnis: " + rechner1.getErgebnis());
System.out.println("Meldung: " + rechner1.getMeldung());
```

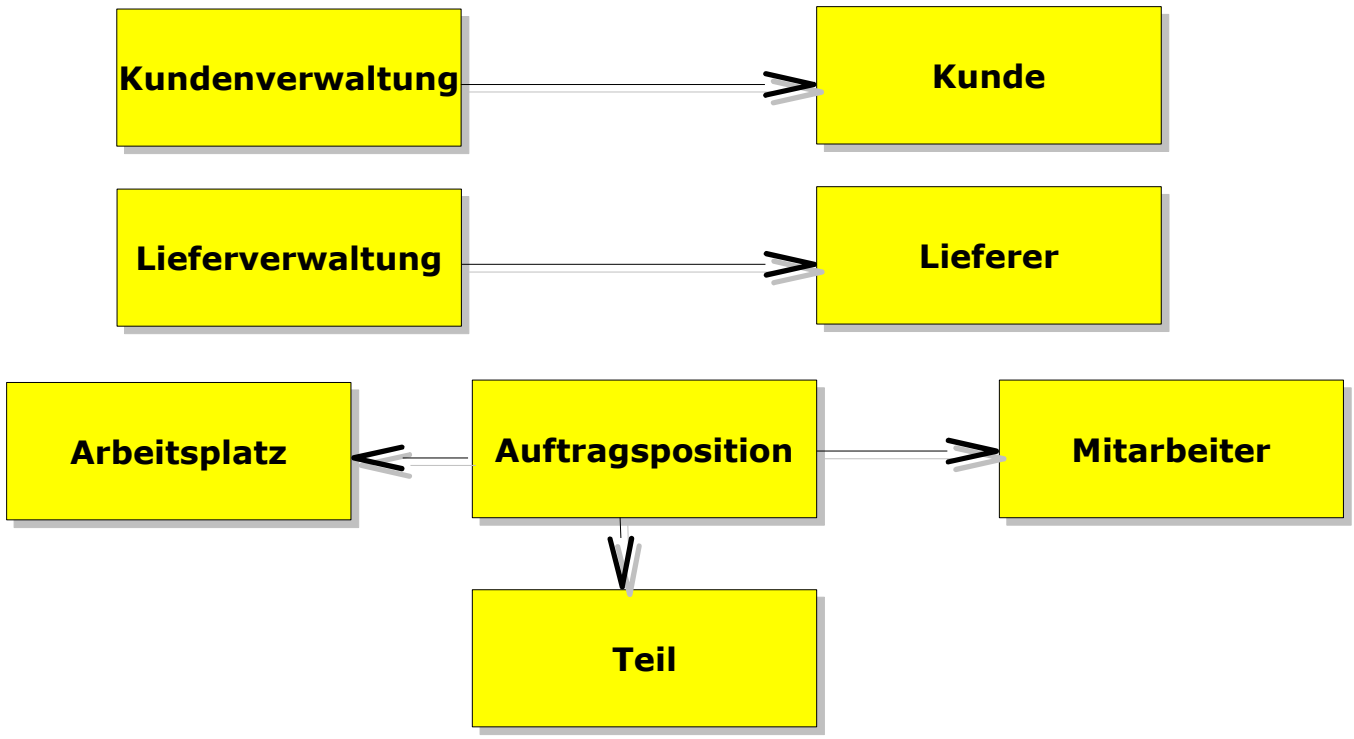
Ausschnitt UNIT-Test, ohne Parameter:

```
System.out.println("#### Dollar - Yuan");

//Eingabe: Wert an Objekt der Klasse übermitteln
rechner1.setVon("Dollar");
rechner1.setIn ("Yuan");
rechner1.setBetrag (1);

//Verarbeitung: Wert verarbeiten z.B. berechnen
rechner1.umrechnen();


//Ausgabe: Ergebnis ausgeben
System.out.println("Betrag: " + rechner1.getBetrag());
System.out.println("Wechselkurs: " + rechner1.getWechselkurs());
System.out.println("Von: " + rechner1.getVon());
System.out.println("In: " + rechner1.getIn());
System.out.println("Ergebnis: " + rechner1.getErgebnis());
System.out.println("Meldung: " + rechner1.getMeldung());
```

1.7 Überblick zu Primitiven Datentypen

Datentyp	**Speicherplatz** in Byte	**Wertebereich** von ... bis	**default** Standardwert	**Bemerkung**
byte	1 byte	Von -128 bis 127	0	Maßeinheit der Digitaltechnik (8 Bit = 1 byte)
short	2 byte	Von -32768 bis 32767	0	(kleine) Ganze Zahlen
int	4 byte	Von -2147483648 bis 2147483647	0	(mittel) Ganze Zahlen
long	8 byte	Von -9223372036854775808 Bis 9223372036854775807	0l	(große) Ganze Zahlen
float	4 byte	Von -3.40282347E+38 Bis 3.40282347E+38	0.0f	(einfache G.) Kommazahl
double	8 byte	Von -1.79769313486231570E+308 Bis 1.79769313486231570E+308	0.0f	(doppelte G.) Kommazahl
char	2 byte	Von \u0000 bis \uffff	\u0000	einzelnes Zeichen
boolean	1 bit	true false	false	Zweiwertiges Attribut

1.8 Assoziationen



Assoziationen	Sind Objekte anderer Klassen. Diese Objekte werden in Klassen implementiert die eine Beziehung pflegen, um die Eigenschaft und Verhaltensweisen dieser Objekte verwenden (nutzen) zu können.
Kundenverwaltung	`private Kunde k = new Kunde();`
Lieferverwaltung	`private Lieferer l = new Lieferer();`

Auftragsposition	`private Mitarbeiter m = new Mitarbeiter();`
	`private Teil t = new Teil();`
	`private Arbeitsplatz a = new Arbeitsplatz();`

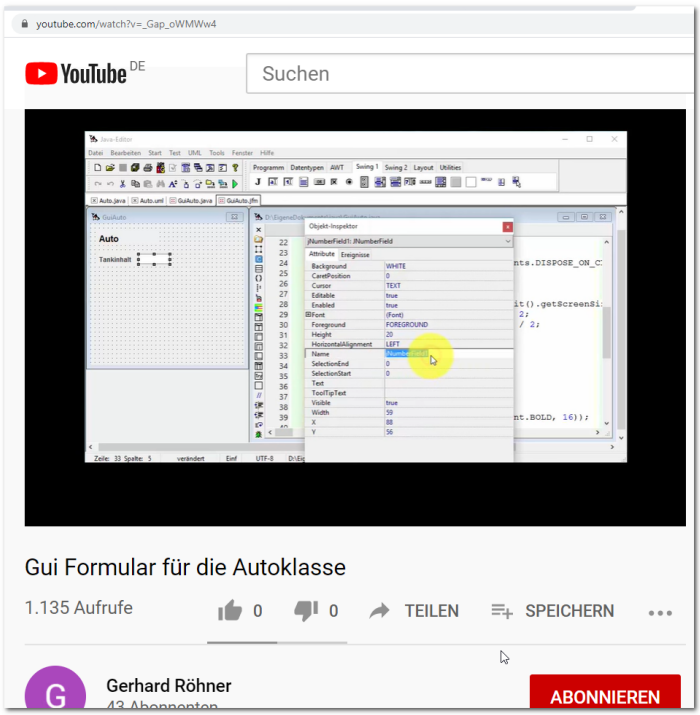
Siehe Projekt: Dienstwagen im Kapitel Algorithmik	

2 View: Benutzeroberflächen (Grafical User Interface, GUI)

2.1 Am Anfang steht ein Entwurf der benötigten Benutzeroberflächen

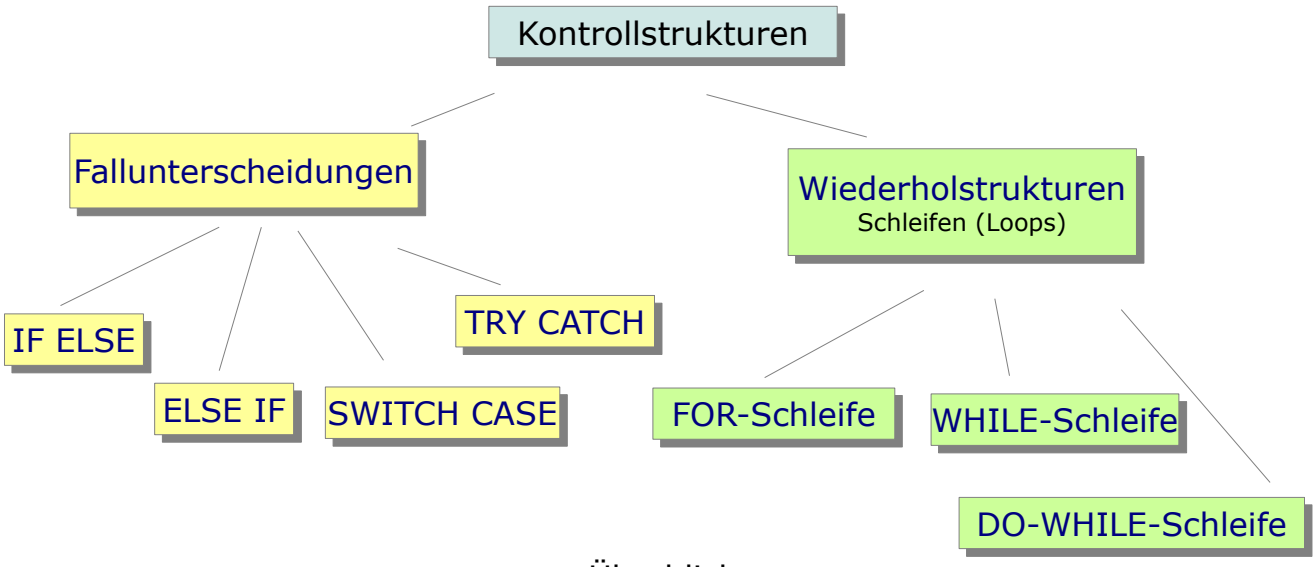
Hier beispielhaft mit dem Textverarbeitungsprogramm erstellt. Sieht schon gut aus, oder?

2.2 Einführung in die GUI-Programmierung

	https://www.youtube.com/watch?v=_Gap_oWMWw4

3 Algorithmik

3.1 Kontrollstrukturen



Überblick

Hinweis: Fallunterscheidungen sind Bestandteil von Verhaltensweisen (Methoden)	
Fallunterscheidungen	

IF ELSE
`if(){`
`}else{`
`}`	

ELSE IF

```
if(){

}else if(){

}else if(){

}else{

}
```

```
if(pKundenart == 1){
  this.setKundenart(pKundenart);
  return "ok";
}else if(pKundenart == 2){
  this.setKundenart(pKundenart);
  return "ok";
}else if(pKundenart == 3){
  this.setKundenart(pKundenart);
  return "ok";
}else{
  this.setKundenart(0);
  return "keiner";
}
```

SWITCH CASE

```
    switch () {
      case 1:
      break;

      case 2:
      break;

      case 3:
      break;

      default:

    }
```

```
switch (this.kundenart) {
  case 1:
  return "ok";

  case 2:
  return "ok";

  case 3:
  return "ok";

  default:
  return "keiner";
}
```

* wenn es sich um eine Methode mit Rückgabewert handelt können die „breaks" weggelassen werden.

Begründung:
Mit dem → break verlässt man die Kontrollstruktur. Da mit dem → return derselbe Effekt erreicht wird müssen wir auf den break verzichten.

Besonderheiten von SWITCH CASE:

- Übersichtlich
- Funktioniert nur bei Prüfung primitiver Datentypen
- Ein Attribut für die Prüfung in der SWTICH Bedingung, dessen Zustand (Wert) geprüft wird.

Hinweis: Wiederholstrukturen sind Bestandteil von Verhaltensweisen (Methoden)		
Wiederholstrukturen	Grundgerüst	Besonderheit

FOR-SCHLEIFE	`for (ausdruck1; ausdruck2; ausdruck3){`
	`  //Anweisung(en)`
	`}`
	
	Beispiel:
	`for(int i = 10;i > 0;i--) {`
	`  System.out.println(i);`
	`}`
		Ist die komplexeste Schleife.
		
		Ausdruck1 wird vor der Schleife ausgeführt und enthält in der Regel die Deklaration und Initatialisierung der Schleifenzählervariablen.
		
		Am Anfang eines jeden Schleifendurchlaufs wird die Anweisung in Ausdruck2 ausgeführt. Wenn diese → True ist wird die Schleife fortgesetzt und die darunterliegenden Anweisungen werden ausgeführt. Anderenfalls (→ False) wird der Vorgang abgebrochen.
		
		Am Ende eines jeden Schleifendurchlaufs wird die Anweisung in Ausdruck3 ausgeführt.

WHILE-SCHLEIFE	`while (bedingung){`
	`  //Anweisung(en)`
	`  //Abbruchbedingung`
	`}`
	
	Beispiel:
	`int i = 10;`
	`while(i > 0) {`
	`  System.out.println(i);`
	`  i--;`
	`}`
		Ist die einfachste Schleife.
		
		Die Anweisungen innerhalb der Schleife werden wiederholt ausgeführt, solange die Bedingung zutrifft, also zu → True evaluiert.
		
		Eine Abbruchbedingung z.B. ein Zähler stellt in der Regel sicher, dass die Schleife nicht endlos ausgeführt wird, die Bedingung zu gegebener Zeit zu → False evaluiert und der Vorgang

<table>
<tr><td></td><td></td><td>abgebrochen wird.</td></tr>
<tr><td>DO-WHILE-SCHLEIFE</td><td>

```
do{
	//Anweisung(en)
	//Abbruchbedingung
}while(bedingung);
```

Beispiel:

```
int i = 10;
do {
	System.out.println(i);
	i--;

} while (i > 0);
```

Beispiel 2 Unterricht:

```
public void teste2(int startwert){
  int x = startwert;
  int zaehler = 0;
  do{
    int iQuadrat = x*x;
    System.out.println(iQuadrat);
    x= x+1;
    zaehler++;
  } while (x <=12);
  System.out.println("Durchlauf: "+zaehler);
}
```

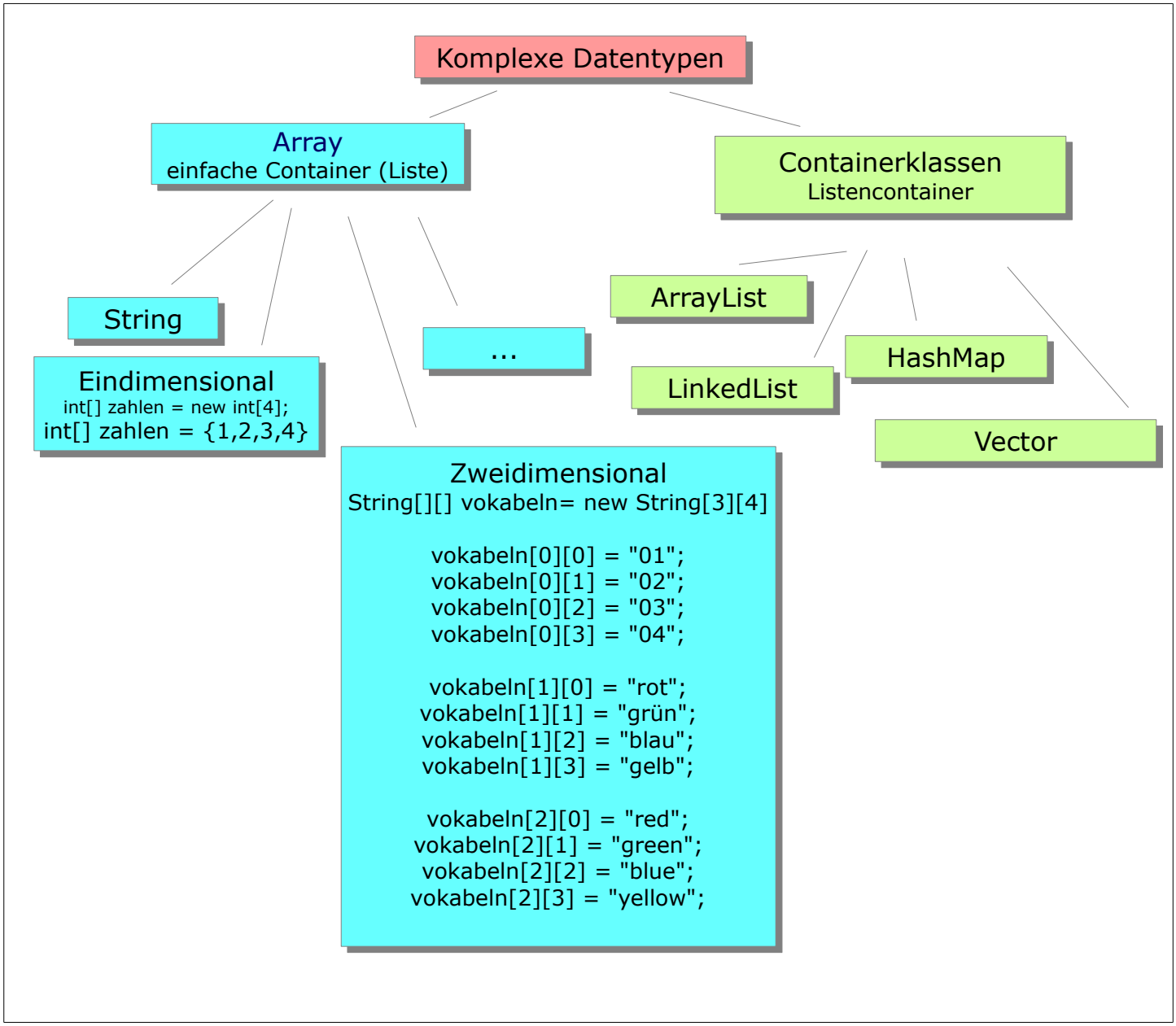
Konsolenausgabe:

```
######Teste Schleife 2########
0
1
4
9
16
25
36
49
64
81
100
121
144
Durchlauf: 13
----------------
100
121
144
Durchlauf: 3
----------------
169
Durchlauf: 1
----------------
```

</td><td>Ist der While-Schleife sehr ähnlich.

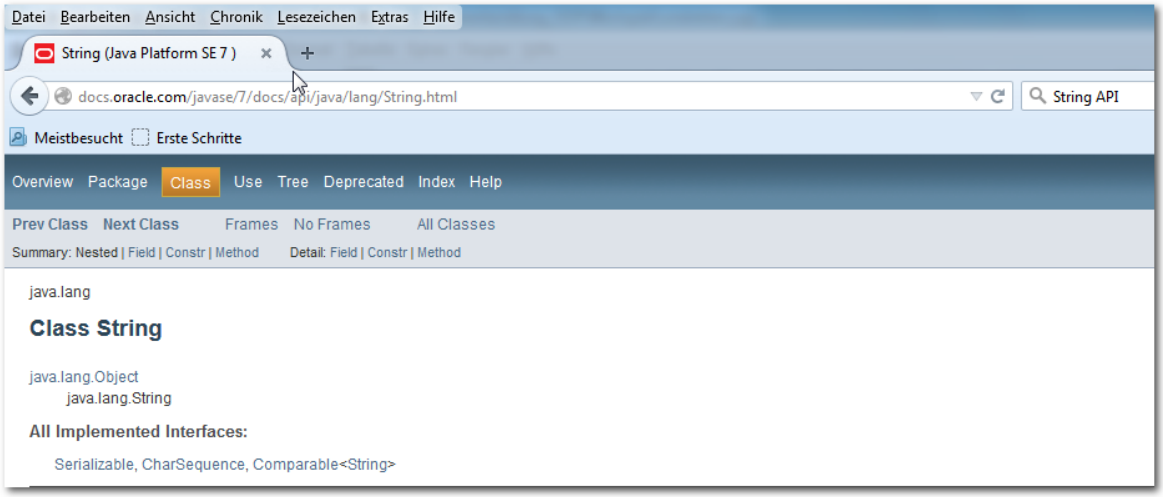
Der Unterschied liegt daran, dass die Bedingung erst am Ende des Schleifendurchlaufs geprüft wird.</td></tr>
</table>

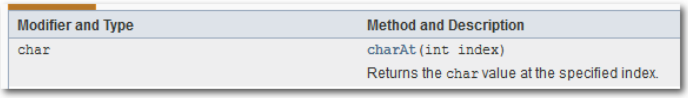
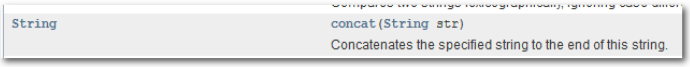
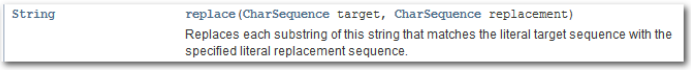
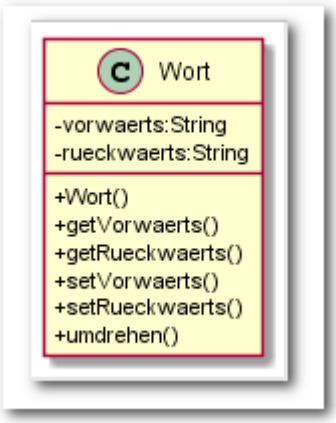
3.2 Einfache Datenstrukturen und Komplexe Datentypen



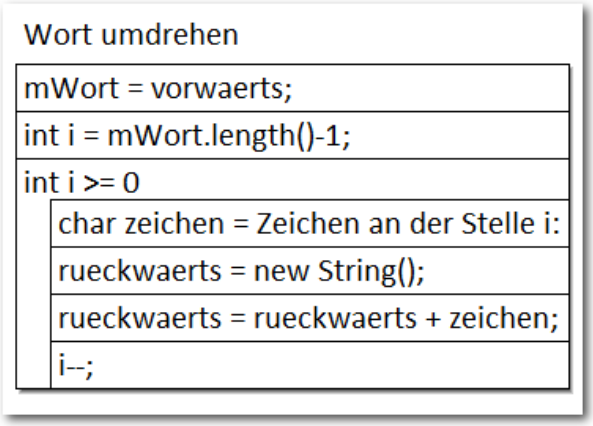
Stringverarbeitung

Ein String ist eine Zeichenkette (Array mit Characters) mit besonderen Eigenschaften und Verhaltensweisen.



charAt(int i)	
concat(String str)	
length()	
replace(char oldChar, char newChar)	
Übung	

Methode umdrehen()



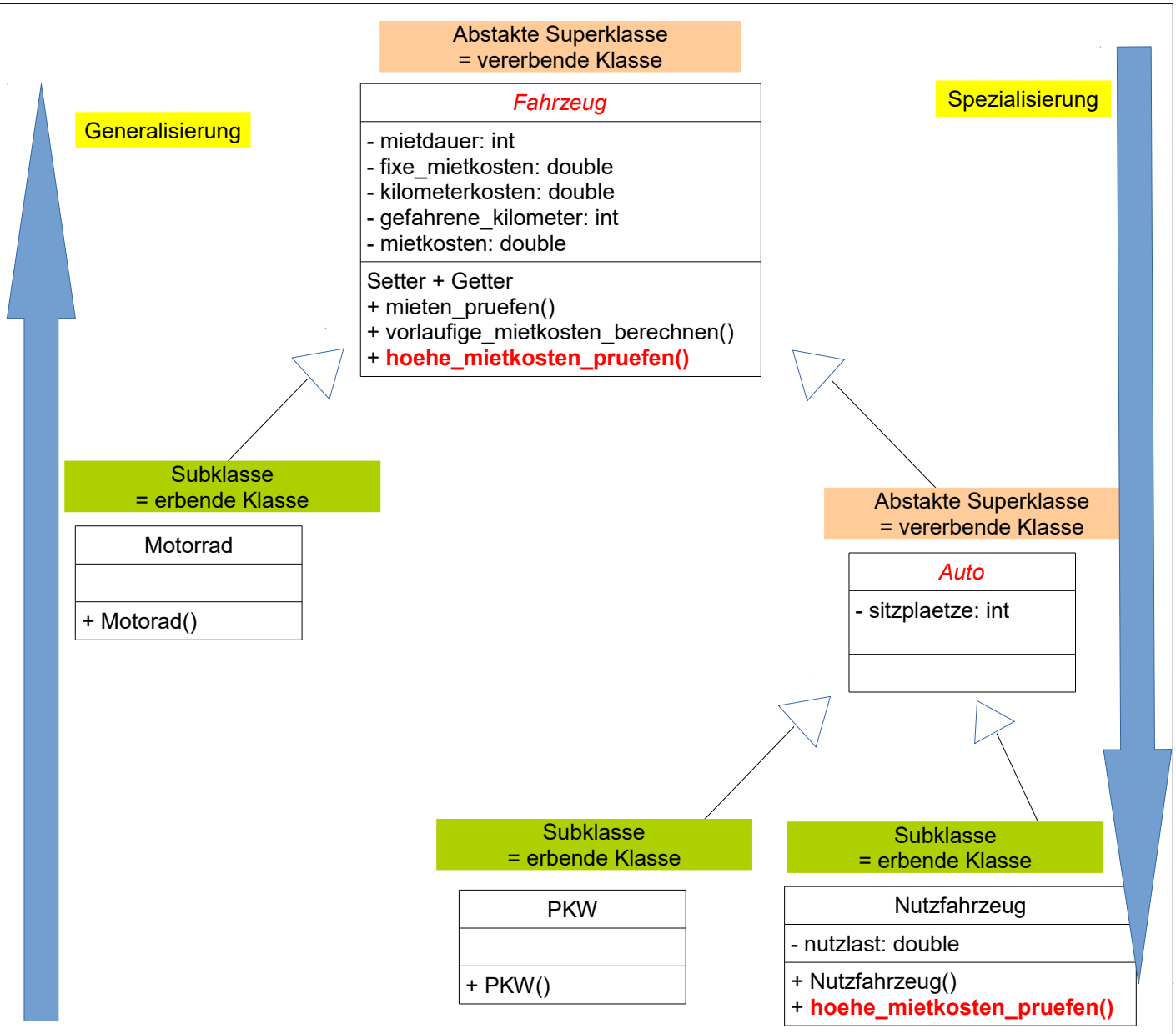
Hier gelöst mit einer WHILE-SCHLEIFE

```
public void umdrehen(){
  String mWort = this.getVorwaerts();
  String mWortRueckwaerts = new String();
  int i = mWort.length()-1;

  while(i >= 0){
    String zeichen = String.valueOf(mWort.charAt(i));
    mWortRueckwaerts = mWortRueckwaerts.concat(zeichen);
    this.setRueckwarts(mWortRueckwaerts);
    i--;
  }

}
```

Hier gelöst mit Hilfe einer FOR-SCHLEIFE

```
//Sonstige Methoden
public void umdrehen(){
  String mWort = this.vorwaerts;
  this.rueckwaerts = new String();
  for (int i = mWort.length()-1;i>=0 ; i--) {
    char zeichen = mWort.charAt(i)  ;
    this.rueckwaerts = this.rueckwaerts
    .concat(String.valueOf(zeichen));
  }
  System.out.println(this.rueckwaerts);
}
```

3.3 Vererbung



Signalwort: **abstract**

Ein Signalwort das dafür sorgt, dass von der Klasse selbst keine Objekte erzeugt werden können. Diese Klassen haben deshalb auch per Definition KEIN Konstruktor.

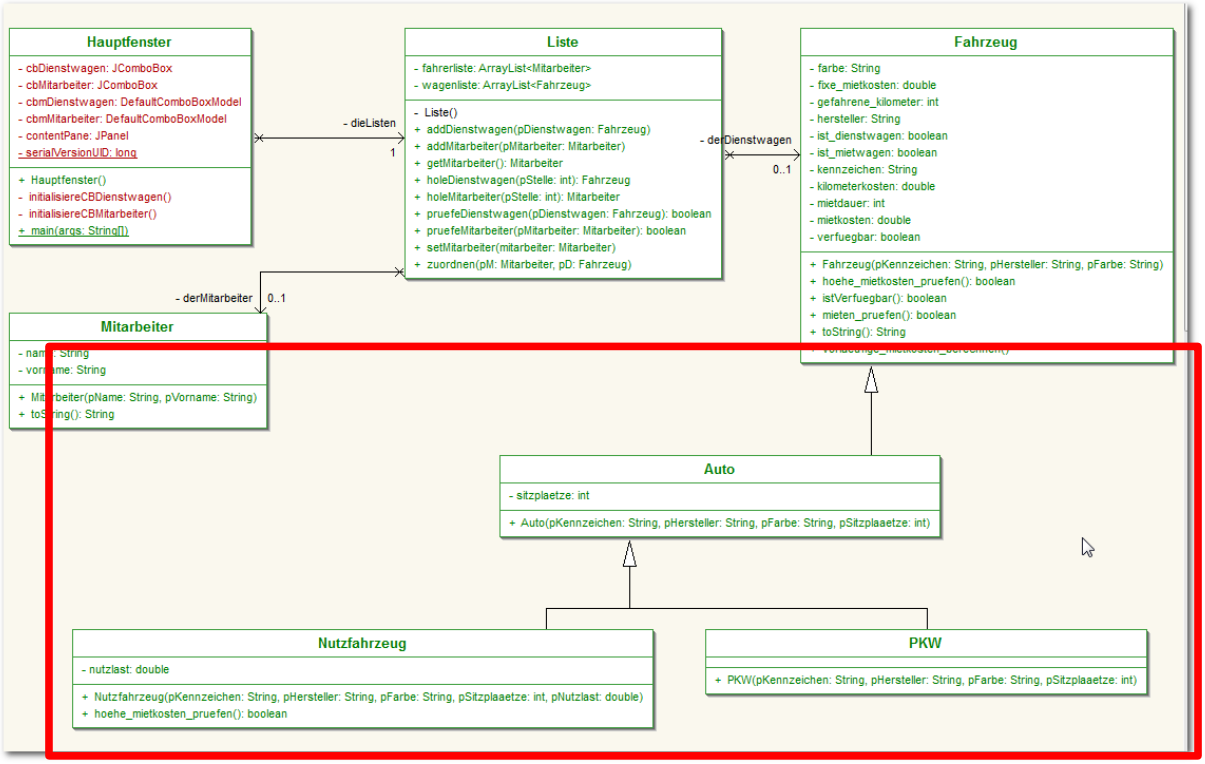
Im UML-Klassendiagramm werden die Klassennamen kursiv geschrieben.

→ Prinzip der Abstraktion

```
public abstract class Fahrzeug {...}
```

Signalwort: **extends**	Mit dem Signalwort erfolgt die Erweiterung einer Superklasse durch die Eigenschaften und Verhaltensweisen einer erbenden Klasse (Subklasse). → Prinzip der Vererbung `public class Nutzfahrzeug extends Auto {...}`

Beispiel:
Umsetzung des Konzepts der Vererbung am Dienstwagen-Projekt:



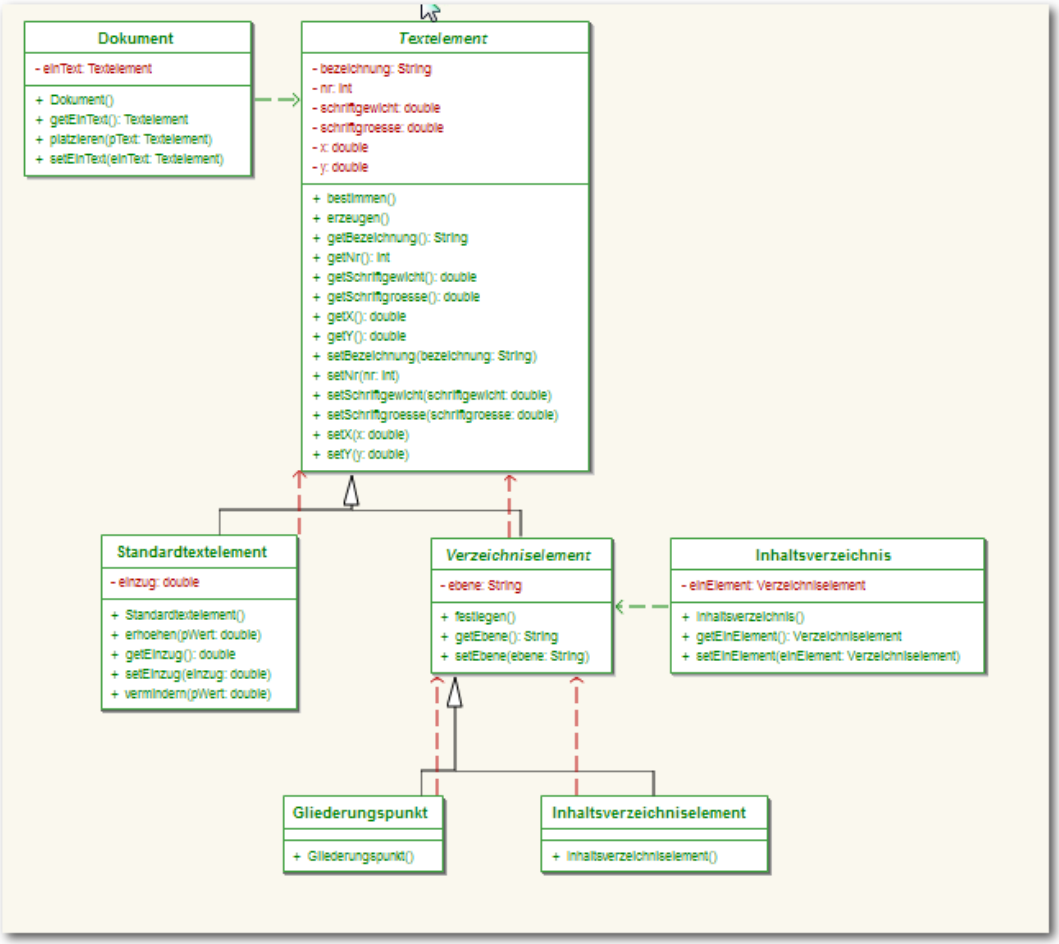
Differenzierung:

Vererbungsbeziehungen sind IST-EINE-Beziehungen
Assoziationen sind HAT-Beziehungen

Übung: Textverarbeitungsfirma

Ein Softwarehersteller möchte eine neue Testverarbeitungssoftware auf dem Markt etablieren. Die Software soll dem Nutzer die Möglichkeit bieten Textelemente zu erzeugen und anhand der Textelementkoordinaten x und y im Dokument zu platzieren. Jedes Textelement hat eine Nummer und eine Bezeichnung. Für alle Textelemente kann u.a. die Schriftgröße und das Schriftgewicht bestimmt werden. Der Nutzer kann nur spezielle Textelemente erzeugen! Im Speziellen kann der Nutzer dazu u.a. Verzeichniselemente und Standardtextelemente erzeugen. Speziell bei den Verzeichniselementen kann der Nutzer die Ebene selbst festlegen. Für ein Standardtextelement kann der Nutzer u.a. den Einzug vermindern oder erhöhen.
Verzeichniselemente können in Inhaltsverzeichniselemente und Gliederungspunkte eingeteilt werden. Verzeichniselemente zeichnen sich dadurch aus, dass sie Bestandteil des Inhaltsverzeichnisses sind.

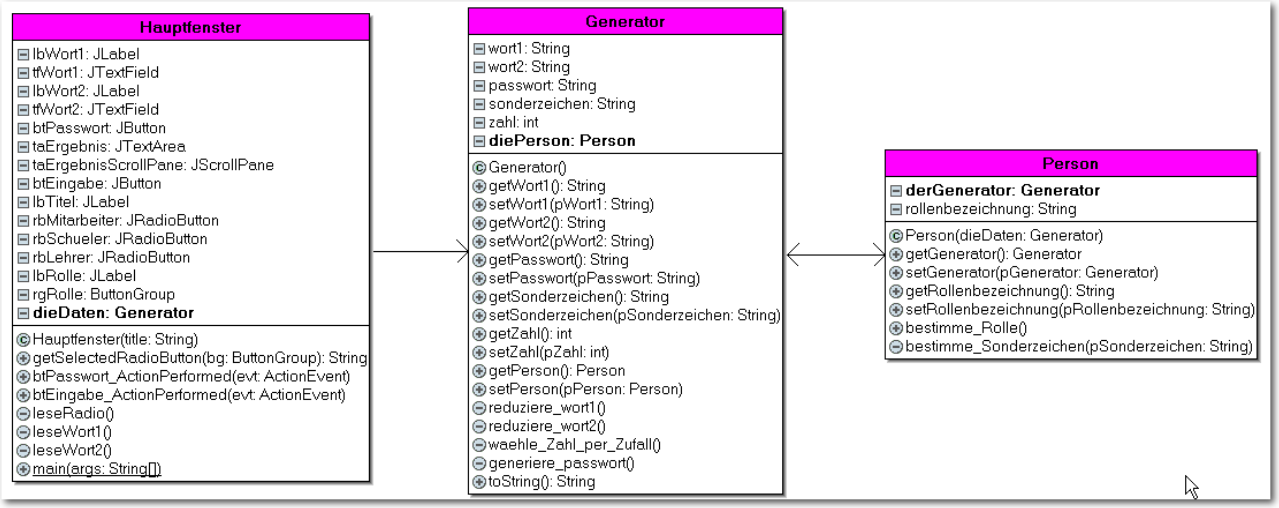
Leisten Sie im ersten Schritt die notwendige Textarbeit und ermitteln Sie dazu Substantive, Verben, Adjektive.	• Substantive sind → Klassen • Adjektive und Substantive die Hauptworten zugeordnet werden können → Attribute • Verben sind → Methoden

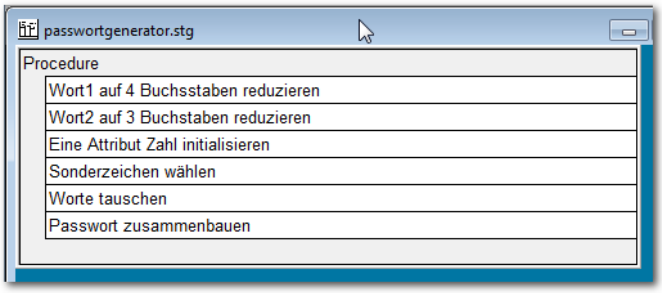


4 Unterrichtsbeispiele

4.1 Passwortgenerator

View

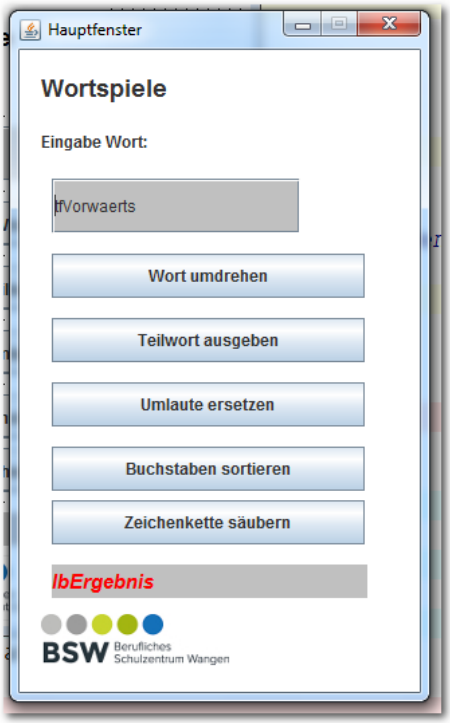


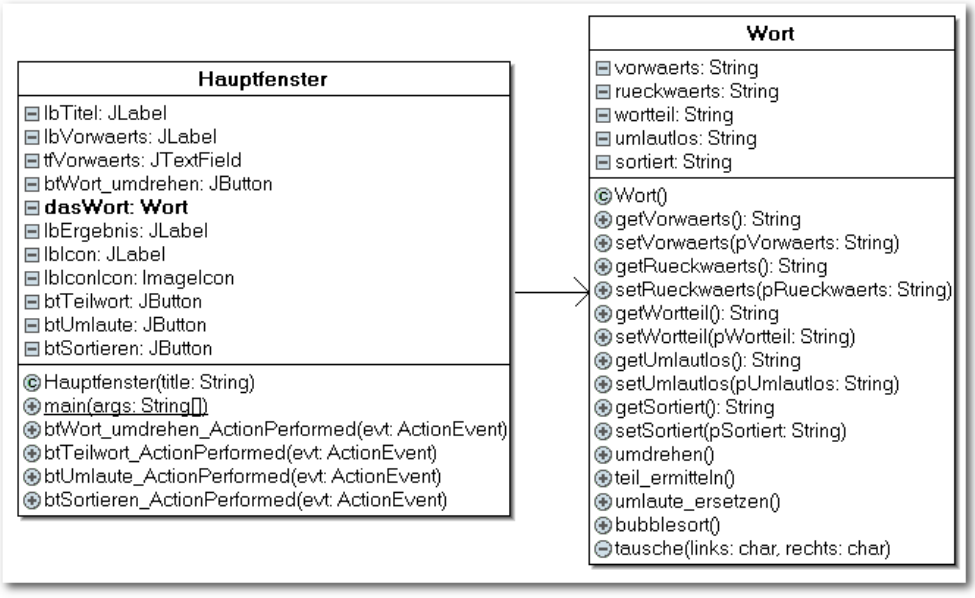


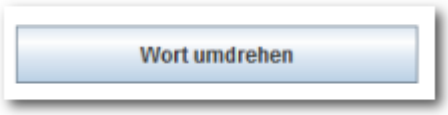
Algorithmik
umfasst die Teilprobleme die es zu lösen gilt, um das Gesamtproblem zu lösen.

	Hinweis: Algorithmik wurde modifiziert (siehe in folgendem) `public void generiere_passwort(){ this.reduziere_wort1(); this.reduziere_wort2(); this.waehle_Zahl_per_Zufall(); this.getPerson().bestimme_Rolle(); this.passwort = this.wort2 + this.zahl + this.wort1 + this.sonderzeichen; }`
`//Anfang Methoden` `private void reduziere_wort1(){ String mWort1 = this.wort1; this.wort1 = mWort1.substring(0,4); }`	Methode enthält Algorithmus: Reduziere Wort 1
`private void reduziere_wort2(){ String mWort2 = this.wort2; this.wort2 = mWort2.substring(0,3); }`	Methode enthält Algorithmus: Reduziere Wort 2
`private void waehle_Zahl_per_Zufall(){ //Liste (Container) mit ganzen Zahlen int[] szliste; szliste = new int[] {1,2,3,4}; Random rand = new Random(); for (int i = 0;i < szliste.length ;i++){ this.zahl = szliste [rand.nextInt(szliste.length)]; } // end of for }`	Methode enthält Algorithmus: Zahl per Zufall wählen
`public void bestimme_Rolle(){ if(this.rollenbezeichnung.equals("Lehrer")){ this.bestimme_Sonderzeichen("!"); }else if(this.rollenbezeichnung.equals("Schüler")){ this.bestimme_Sonderzeichen(":"); }else if(this.rollenbezeichnung.equals("Mitarbeiter")){ this.bestimme_Sonderzeichen("%"); }else{ this.rollenbezeichnung = "Gast"; this.bestimme_Sonderzeichen("?"); } }`	Methode enthält Algorithmus: Bestimme Rolle
`private void bestimme_Sonderzeichen(String pSonderzeichen){ derGenerator.setSonderzeichen(pSonderzeichen); }`	Private Hilfsmethode Bestimme Sonderzeichen

4.2 Wortspiele







Modell - In der Fachklasse:

```
//Sonstige Methoden
public void umdrehen(){
  String mWort = this.vorwaerts;
  this.rueckwaerts = new String();
  for (int i = mWort.length()-1;i>=0 ; i--) {
    char zeichen = mWort.charAt(i)  ;
    this.rueckwaerts = this.rueckwaerts.concat(String.valueOf(zeichen));
  }
  System.out.println(this.rueckwaerts);
}
```

Controller – Ereignissteuerung:

```
public void btWort_umdrehen_ActionPerformed(ActionEvent evt) {
  //Eingabe
  String mVorwaerts = tfVorwaerts.getText();

  //Verarbeitung
  dasWort.setVorwaerts(mVorwaerts);
  dasWort.umdrehen();

  //Ausgabe
  lbErgebnis.setText(dasWort.getRueckwaerts());

} // end of btWort_umdrehen_ActionPerformed
```

Modell - In der Fachklasse:

```
/*Die Methode soll die ersten drei
Buchstaben in einem Wort ermitteln*/
public void teil_ermitteln(){
  String mWort = this.vorwaerts;
  this.wortteil =  mWort.substring(0,3);
}
```

Controller – Ereignissteuerung:

```
public void btTeilwort_ActionPerformed(ActionEvent evt) {
  //Eingabe
  String mVorwaerts = tfVorwaerts.getText();

  //Verarbeitung
  dasWort.setVorwaerts(mVorwaerts);
  dasWort.teil_ermitteln();

  //Ausgabe
  lbErgebnis.setText(dasWort.getWortteil());
} // end of btTeilwort_ActionPerformed
```

Modell - In der Fachklasse:

```
/*Die Methode soll das Wort durchlaufen
und Umlaute ermitteln und ersetzen
ü --> ue
ä --> ae
ö --> oe
ß --> ss*/

public void umlaute_ersetzen(){
  String mWort = this.vorwaerts;
  this.umlautlos = new String();
  for (int i =0 ;i<=mWort.length()-1 ; i++) {
    char zeichen = mWort.charAt(i)  ;
```

```
    if (zeichen == 'ü') {
      String mErsatz = "ue";
      this.umlautlos = this.umlautlos;
      .concat(String.valueOf(mErsatz));
    }else if(zeichen == 'ö'){
      String mErsatz = "oe";
      this.umlautlos = this.umlautlos;
      .concat(String.valueOf(mErsatz));
    }else if(zeichen == 'ä'){
      String mErsatz = "ae";
      this.umlautlos = this.umlautlos;
      .concat(String.valueOf(mErsatz));
    }else if(zeichen == 'ß'){
      String mErsatz = "ss";
      this.umlautlos = this.umlautlos;
      .concat(String.valueOf(mErsatz));
    }else{
      this.umlautlos = this.umlautlos;
      .concat(String.valueOf(zeichen));
    }
  }
}
```

Controller – Ereignissteuerung:

```
public void btUmlaute_ActionPerformed(ActionEvent evt) {
  //Eingabe
  String mVorwaerts = tfVorwaerts.getText();

  //Verarbeitung
  dasWort.setVorwaerts(mVorwaerts);
  dasWort.umlaute_ersetzen();

  //Ausgabe
  lbErgebnis.setText(dasWort.getUmlautlos());

} // end of btUmlaute_ActionPerformed
```

Modell - In der Fachklasse:

```
/*Sortieren von Buchstaben: Vergleichen und
austauschen - Bubblesort*/
public void bubblesort(){
  char[] mUnsortiert = this.vorwaerts.toCharArray();
  //Durchlaufe die Liste
  for (int i = mUnsortiert.length; i>1;i-- ) {

    //Ermittle zwei Elemente benachbarte Elemente der Liste
    for (int j=0; j<i-1; j++){
      //Prüfe ob das linke Element größer ist als das rechte Element
      if (mUnsortiert[j] > mUnsortiert[j+1]){

        //Beispiel für die Kapselung
        this.tausche(mUnsortiert[j],mUnsortiert[j+1]) ;
      } // ende if
    }
  }
  this.sortiert =  String.valueOf(mUnsortiert);
  System.out.println(this.sortiert);
}
```

Controller – Ereignissteuerung:

```
public void btSortieren_ActionPerformed(ActionEvent evt) {
  //Eingabe
  String mVorwaerts = tfVorwaerts.getText();

  //Verarbeitung
  dasWort.setVorwaerts(mVorwaerts);
  dasWort.bubblesort();

  //Ausgabe
  lbErgebnis.setText(dasWort.getSortiert());

} // end of btSortieren_ActionPerformed
```

Modell - In der Fachklasse:

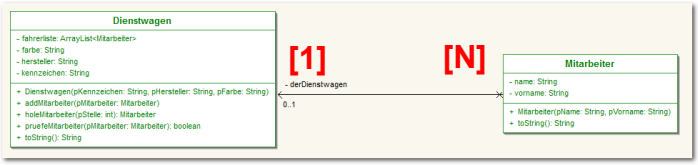
```
private void tausche(char links, char rechts){
  char mTemp;
  mTemp = links;
  links =  rechts;
  rechts= mTemp;
}
```

```
public void saubermachen(){
  String mZeichenkette = this.vorwaerts.trim();
  this.sauber = new String();
  int i = 0;
  while (i < mZeichenkette.length()){
    String mZeichen =  mZeichenkette.substring(i,i+1);

    if (mZeichen.equals("#")) {
      mZeichen = "";
    }if(mZeichen.equals(" ")){
      mZeichen = "";
    }
    this.sauber= this.sauber.concat(mZeichen);
    i++;
  }
}
```

```
public void btSaubermachen_ActionPerformed(ActionEvent evt) {
  //Eingabe
  String mVorwaerts = tfVorwaerts.getText();

  //Verarbeitung
  dasWort.setVorwaerts(mVorwaerts);
  dasWort.saubermachen();

  //Ausgabe
  lbErgebnis.setText(dasWort.getSauber());
} // end of btSaubermachen_ActionPerformed
```

4.3 Dienstwagennutzung

Entwurf der Benutzeroberfläche



Realisierung im Quellcode...

Assoziation in Mitarbeiter:

```
private Dienstwagen derDienstwagen;
```

Assoziation in Dienstwagen:

```
private ArrayList<Mitarbeiter> fahrerliste
    = new ArrayList<Mitarbeiter>();
```

Verhaltensweisen für die Fahrerliste:

```
//sonstige Methoden
public void addMitarbeiter(Mitarbeiter pMitarbeiter){
  this.fahrerliste.add(pMitarbeiter);
}

public Mitarbeiter holeMitarbeiter(int pStelle){
  return this.fahrerliste.get(pStelle);
}

public boolean pruefeMitarbeiter(Mitarbeiter pMitarbeiter){
  if (this.getFahrerliste().contains(pMitarbeiter)) {
    return true;
  }else{
    return false;
  } // end of if
}
```

Assoziation:
Wir formulieren dazu die Eine-Sätze:

1. Ein Mitarbeiter fährt **genau einen Dienstwagen.[1]**
2. Ein Dienstwagen wird von einem oder **mehreren Mitarbeitern** gefahren.**[N]**

Im Kontext Java heißt der mengenmäßige Zusammenhang zwischen den Objekten zweier Klassen Multiplizität. Diese Betrachtung entspricht der Kardinalität im Kontext Relationaler Datenbanken.

Anwendungsfall 1:
Fünf Mitarbeiter nutzen insgesamt drei Dienstwagen.

Mitarbeiter:

Name	Vorname
Müller	Lisa
Maier	Kurt
Keller	Fritz
Mauser	Karl
Hauser	Fred

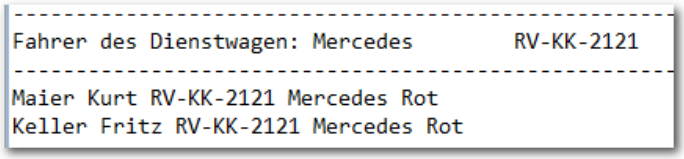
Dienstwagen:

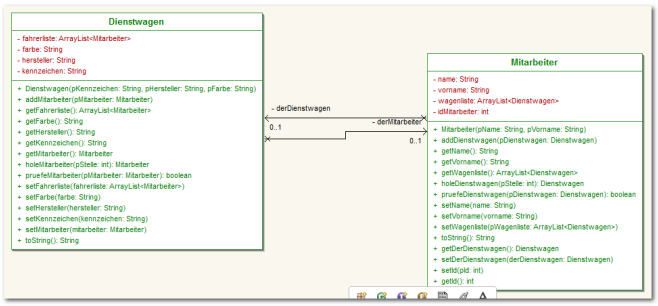
Kennzeichen	Hersteller	Farbe
RV-LM-2002	Audi	Blau
RV-KK-2121	Mercedes	Rot
RV-KF-0404	VW	Gelb

Zusammenhang:

```
------------------------------------------------------------
Mitarbeiter     fahrt mit       Dienstwagen
------------------------------------------------------------
Müller Lisa RV-LM-2002 Audi Blau
Maier Kurt RV-KK-2121 Mercedes Rot
Keller Fritz RV-KK-2121 Mercedes Rot
Mauser Karl RV-KF-0404 VW Gelb
Hauser Fred RV-KF-0404 VW Gelb
```

Zusammenhang für ein Dienstwagen (→ Mercedes):

	 Fahrer des Dienstwagen: Mercedes RV-KK-2121 Maier Kurt RV-KK-2121 Mercedes Rot Keller Fritz RV-KK-2121 Mercedes Rot

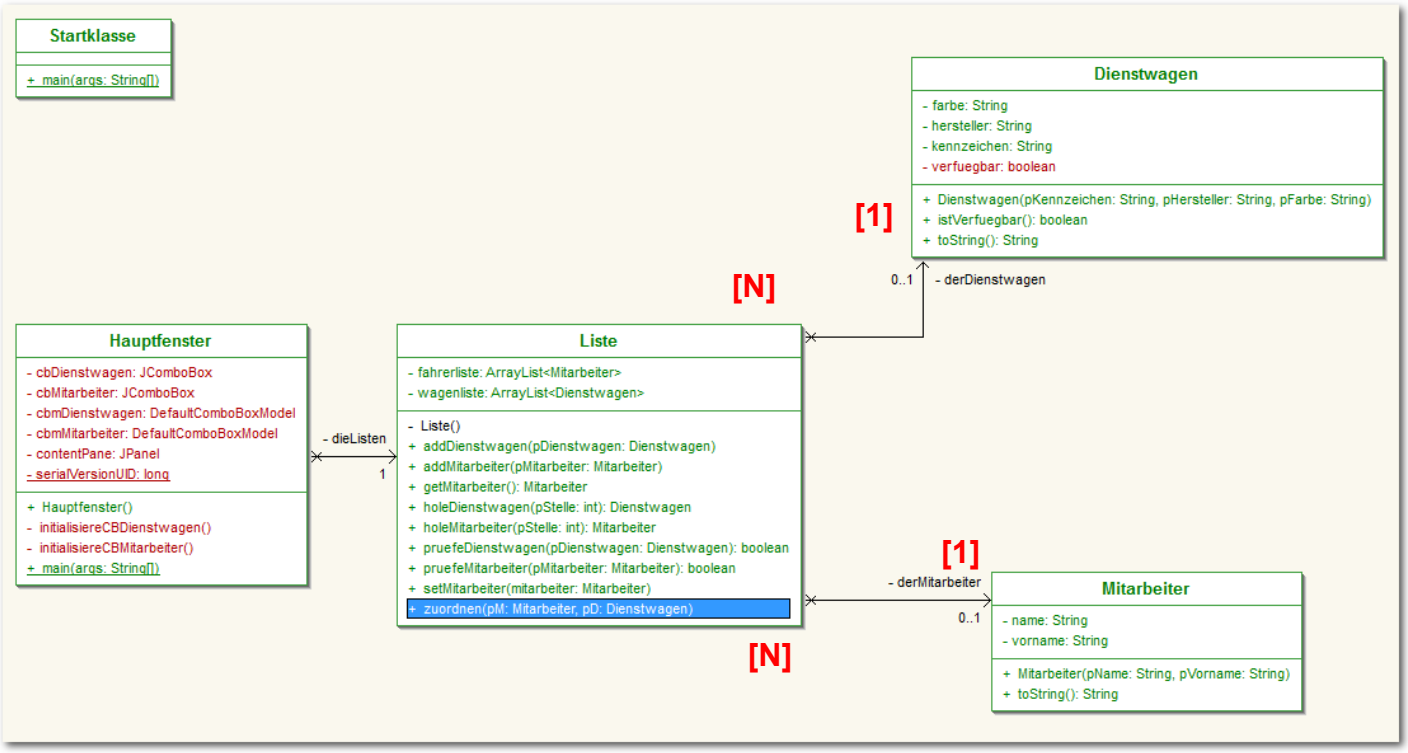


Änderung der Anforderung:
Zukünftig soll jeder Mitarbeiter, jeden freien Dienstwagen nutzen können.

Assoziation:
Wir formulieren dazu die Eine-Sätze:

1. Ein Mitarbeiter fährt **einen oder mehrere Dienstwagen**.**[N]**
2. Ein Dienstwagen wird von einem oder **mehreren Mitarbeitern** gefahren.**[M]**

Musterlösung:

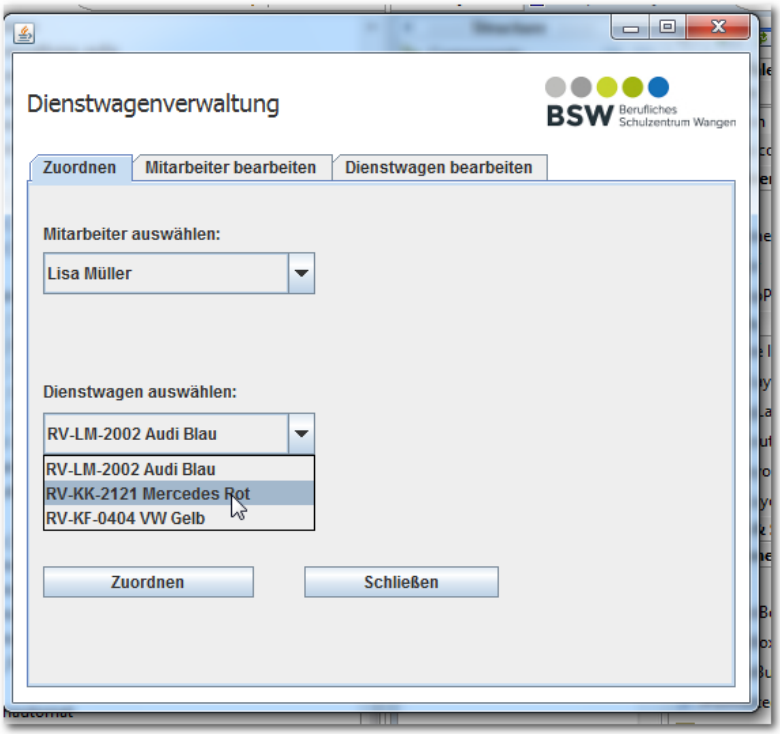


Aus N:M **wird 1:N und N:1**

Was ist Anders?

1. Wir nutzen eine extra Klasse → hier: Liste
2. Auslagerung der Listenoperationen
3. dadurch: Reduzierung der Fachklasse auf wesentliche Eigenschaften und Verhaltensweisen.
4. Wiederholungen im Quellcode vermeiden → Combobox-Modell wird mit der Liste verkünpft. Dazu werden die Verhaltensweisen Methoden im Hauptfenster integriert [initialisiereCBDienstwagen() und initialisiereCBMitarbeiter()] → verweisen auf die fahrerliste bzw. wagenliste.

Umsetzung im Quellcode:



Benutzeroberflächenklasse Hauptfenster:

Für die ComboBox Dienstwagen:

1. Deklaration der ComboBox

```
private JComboBox cbDienstwagen;
```

2. Objekt der Klasse DefaultComboBox-Model

```
private DefaultComboBoxModel cbmDienstwagen
    = new DefaultComboBoxModel();
```

3. Initialisierung der ComboBox im Konstruktor des Hauptfensters

```
cbDienstwagen = new JComboBox();
```

4. Zuordnung des benötigten Combo-Box-Modells

```
cbDienstwagen.setModel(cbmDienstwagen);
```

5. Implementierung einer Methode für die Initialisierung der ComboBox

```
private void initialisiereCBDienstwagen(){
    for(int i=0; i <dieListen.getWagenliste().size();i++){
        Dienstwagen x = dieListen.holeDienstwagen(i);
        cbmDienstwagen.addElement(x);
    }
}
```

6. Nutzung der Methode (Methoden-Aufruf) im Konstruktor

```
initialisiereCBDienstwagen();
```

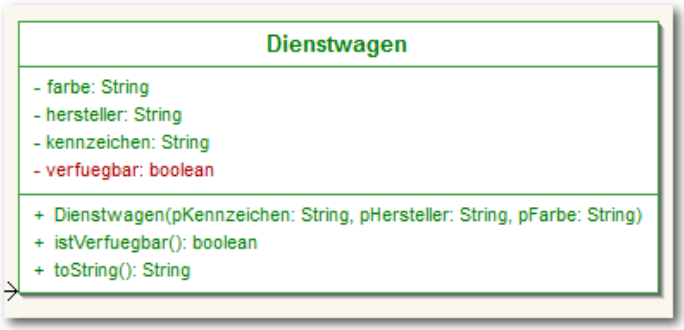
Fachklasse Liste:

Alle Listenobjekte werden in eine zusätzliche Klasse → Liste ausgelagert.

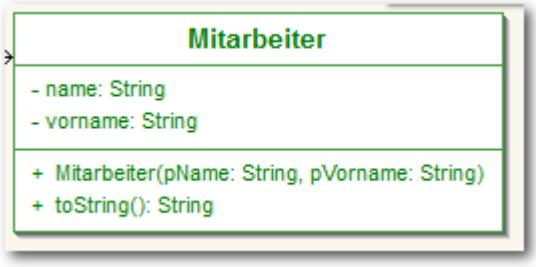
Alle Listenbezogenen Eigenschaften (Objekte → ArrayList) und deren Verhaltensweisen werden in die Klasse ausgelagert.

Es entsteht eine Art „Steuerungs- bzw. Verwaltungsklasse"

Fachklasse Dienstwagen



Fachklasse Mitarbeiter



organisieren. Hierzu soll es möglich sein dass Lehrer unterschiedliche Räume nutzen können. Die Software soll sicherstellen dass es bei der Raumbelegung nicht zu Überschneidungen (Doppelbelegung) kommen kann. Beziehen Sie den Zeitpunkt in Ihre Überlegungen mit ein.	gehensweise.
Übung 2: Assoziation Für die Terminvereinbarung der Elternteile mit Lehrkräften soll künftig mit Hilfe einer Software möglich sein. Diese soll für Eltern die Möglichkeit bieten Termine mit Lehrer zu reservieren. Lehrer und Elternteile sollen sich Ihre Terminliste anzeigen lassen können.	Aufgabe: 1. Erzeugen Sie das Datenbankmodell. 2. Erzeugen Sie die Softwarearchitektur. 3. Begründen Sie jeweils Ihre Vorgehensweise.

5 Projekt

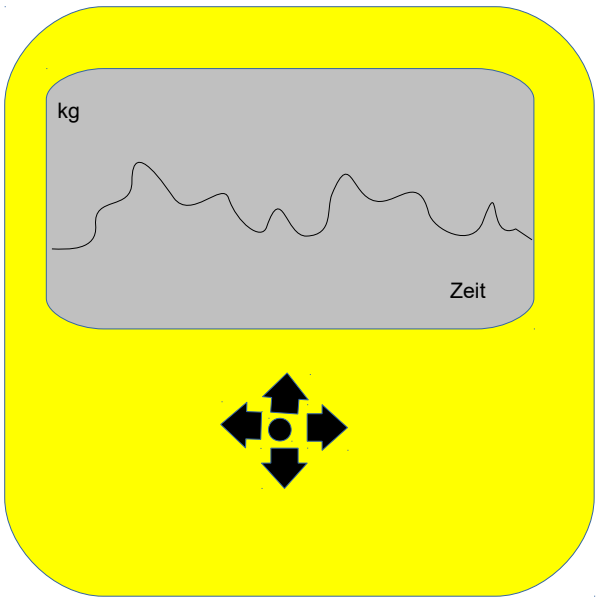
5.1 Anforderungen

1. Freie Themenwahl mit Genehmigung des Fachlehrers.
2. Entwurf und Screenshots der Benutzeroberfläche(n)
3. Anforderungen, Funktionalitäten
4. Quellcode der Ereignissteuerung für die Schaltflächen inklusive Implementierung der verwendeten Hilfsmethoden.
5. Quellcode der implementierten Fachklasse(n).
6. UML-Klassendiagramm der Anwendung.
7. Unit-Test mit Testdaten und den Ergebnissen.
8. Protokoll: Probleme und Lösungen/Lösungsansätze
9. Abgabe Projekt: 26.03.2020 per E-Mail

5.2 Projekt-Beispiele

Bezeichnung	**Bemerkung**	**Umfang**
Primzahlen-Checker	Schülerprojekt	gerrechtfertigt
Torjäger-App	Schülerprojekt	gerrechtfertigt
Promille-Rechner	Schülerprojekt	gerrechtfertigt
Tic-Tac-Toe-App	Unterrichtsprojekt	umfangreich
Vokabeltrainer	Unterrichtsprojekt	umfangreich
Chat	Unterrichtsprojekt	sehr umfangreich
Zeitzonen-Rechner	Schülerprojekt	gerrechtfertigt

6 Mündliche Prüfung



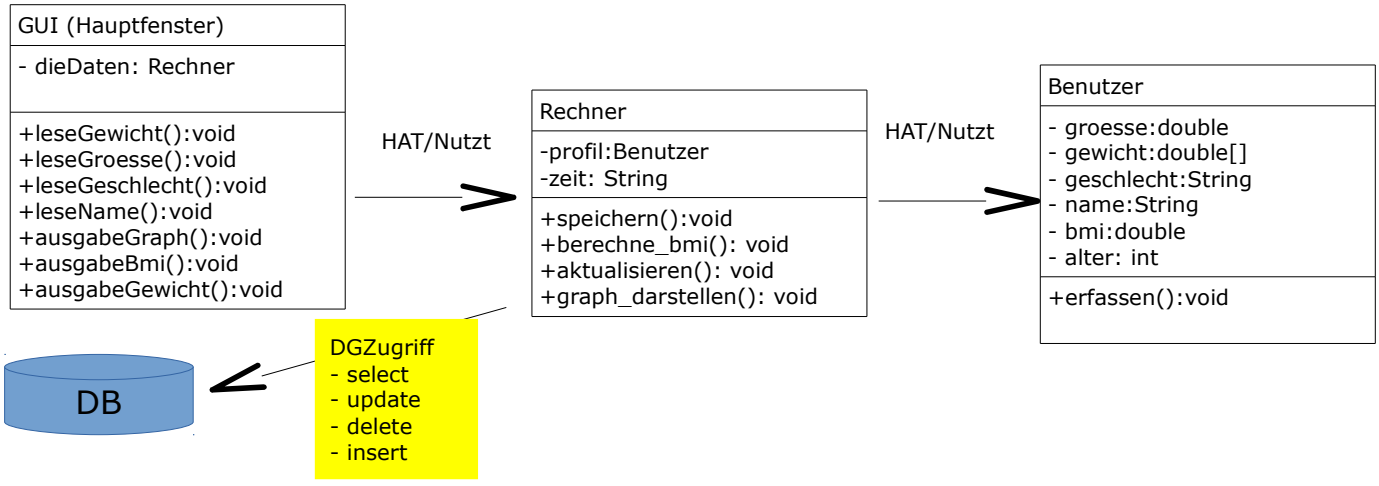
Die digitale Funkwaage

Die digitale Funkwaage soll in der Lage das Profil eines Benutzers anhand des Namens zu erfassen. Außerdem sollen das Gewicht, die Größe und das Geschlecht des Benutzers dauerhaft gespeichert werden. Für den Fall, dass der Benutzer die Waage nutzt soll dann das Gewicht aktualisiert werden, der aktuelle BMI errechnet werden und abschließend der Gewichtsverlauf über die Zeit grafisch auf einem einfachen Display dargestellt werden.

Aufgabe:

Welche Systemvoraussetzungen müssen Sie schaffen?

UML-Klassendiagramm: Unified Modelling Language (Auszeichnungssprache)



Entwurf der Datensammlung:

Nachteil:

- Wiederholungen
- Nullfeldern (keine Werte)
- Expansion einer Datenbanktabelle nach rechts (Spalten kommen nachträglich hinzu)

name	groesse	gewicht	geschlecht	zeit	alter
chrissi	1.70	54	w	2016-06-27	42
chrissi	1.70	55	w	2016-06-26	42
chrissi	1.70	53.5	w	2016-06-25	42
chrissi	1.70	54.5	w	2016-06-24	42
chrissi	1.70	53.5	w	2016-06-25	42

Datenbankstruktur optimieren:

name	groesse	geschlecht	alter
chrissi	1.70	w	42
markus	1.85	m	45

Tabelle: Benutzer

zeitstempel	gewicht	**name_fk**
2016-06-27 - 07:00:01	54	chrissi
2016-06-26 - 07:30:01	55	chrissi
2016-06-25 - 07:21:01	53.5	chrissi
2016-06-24 - 07:05:01	54.5	chrissi
2016-06-25 - 07:17:01	53.5	chrissi
2016-06-27 - 07:12:01	67	markus
2016-06-26 - 08:08:01	68	markus
2016-06-25 - 06:00:01	65.6	markus
2016-06-24 - 05:15:01	65.3	markus
2016-06-25 - 07:00:01	66	markus

Tabelle: Rechner

Bestimmung der Kardinalität:
Mengenmässiger Zusammenhang

Eine-Sätze:

1. Ein Rechner erfasst ein oder mehrere Benutzer [N]

2. Ein Benutzer wird von genau einem Rechner erfasst [1]

ERM: Entity Relationship Modell