Technische Universität Berlin **Fachgebiet für Neuronale Informationsverarbeitung**

Übungsblatt 11

Prof. Dr. Klaus Obermayer und Ivo Trowitzsch

J

Abstrakte Kunst

Verfügbar ab:	**18.01.2019**
Abgabe bis:	**27.01.2019**

Aufgabe 1: Java Syntax

2.5 Punkte

1. Betrachten Sie die folgenden Deklarationen:

```
interface I {...}
abstract class A {...}
class C {...}
```

Welche der folgenden Anweisungen sind stets unzulässig:

- ◯ `I x = new I();`
- ◯ `A y = new A();`
- ◯ `C z = new C();`
- ◯ `I[] u = new I[3];`
- ◯ `A[] v = new A[3];`
- ◯ `C[] w = new C[3];`

2. Betrachten Sie die folgenden Deklarationen:

```
interface I {...}
abstract class A implements I {...}
class C extends A implements I {...}
```

Welche der folgenden Anweisungen sind stets unzulässig:

- ◯ `I w = new A();`
- ◯ `I x = new C();`
- ◯ `A y = new C();`
- ◯ `C z = new C();`

3. Betrachten Sie die folgenden Deklarationen:

```
interface I1 {...}
interface I2 {...}
abstract class A1 {...}
abstract class A2 {...}
class C {...}
```

Welche der folgenden Deklarationen sind stets unzulässig:

- ◯ `interface I implements I1, I2 {...}`
- ◯ `abstract class A extends A1, A2 {...}`
- ◯ `abstract class A extends A1 implements I1, I2 {...}`
- ◯ `class D extends C implements I1 {...}`
- ◯ `class D extends C implements I1, I2 {...}`
- ◯ `class D extends A1 implements I1, I2 {...}`

4. Weshalb ist der folgende Quelltext nicht kompilierbar?

```
interface I {
    public int n;
    public void f();
}
```

5. Weshalb ist der folgende Quelltext nicht kompilierbar?

```
interface I {
    int x = 1;
}

class C extends I {
    void f() {
        x = 2;
    }
}
```

Aufgabe 2: Monster **7 Punkte**

In dieser Aufgabe wird ein kleines Monsterspiel unter Nutzung von Vererbung erstellt.

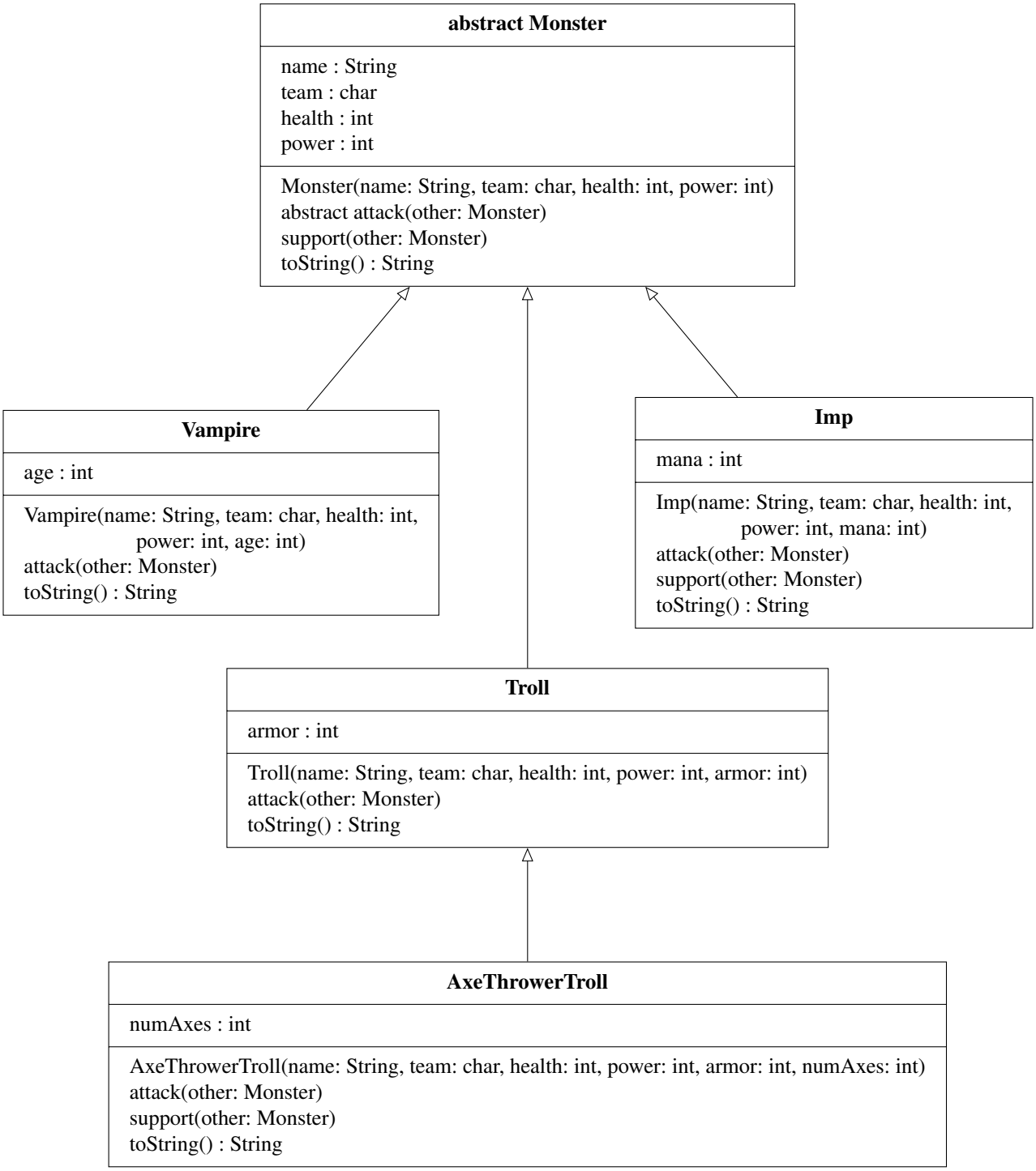
Im Spiel existieren Monster (Vampire, Imp, Troll und AxeThrowerTroll), die in Teams eingeteilt werden und dann gegen Monster der anderen Teams kämpfen (Methode `attack()`), wobei der angegriffene Gesundheit (`health`) verliert. Alternativ können auch verbündete Monster unterstützt werden (Methode `support()`). Diese Methoden sind je nach Monstertyp unterschiedlich. Die Monster sind als Klassen zu implementieren. Die gegebene *battleground.java* erstellt Monster und lässt diese miteinander kämpfen, bis ein Gewinner feststeht.

Hinweis: Es darf kein "einfaches" Monster existieren, nur die Klassen Vampire, Imp, Troll und AxeThrowerTroll dürfen erzeugt werden.

Überlegen Sie sich als Vorbereitung Antworten auf die folgenden Fragestellungen (keine Abgabe der Antworten):

1. Was ist Polymorphie?
2. Was bedeutet Überschreiben (nicht Überladen) von Methoden?
3. Was ist der Zweck des Schlüsselwortes `abstract`?

Betrachten Sie nun folgendes Klassendiagramm:



Im Folgenden sollen diese Klassen implementiert werden. Beachten Sie beim Definieren der Methoden und Attribute stets das Klassendiagramm. Wählen Sie zur Vereinfachung ausnahmsweise das Sichtbarkeitslevel **`public`** für Attribute.

1. Erstellen Sie die Klasse `Monster.java`
   a) Der Konstruktor weist allen Klassenattributen die übergebenen Parameter zu.
   b) Der Konstruktor prüft im Konstruktorrumpf die Eingabe des Teams. Das Nullliteral (`\u0000`) soll abgefangen und stattdessen die Ziffer '0' als Teamname verwendet werden. Der Nutzer soll eine Warnung auf der Konsole erhalten, die erklärt, was passiert ist.
   c) Der Konstruktor gibt unter Verwendung des Namens eine Meldung auf der Konsole aus, dass ein

Monster erschaffen wurde.

d) Die `String toString()`-Methode erstellt einen String, der alle Attribute lesbar zusammenfasst.

e) Deklarieren Sie die Klasse sowie die Methode `void attack(Monster other)` als `abstract`.

f) Nicht alle Monster können andere konstruktiv unterstützen. Die Methode `void support(Monster other)` soll daher in der Grundfassung das zu unterstützende Monster nur mit offenem Mund anstarren. Sorgen Sie für eine entsprechende Konsolenausgabe. Manche der spezielleren Monster verhalten sich geschickter. Dazu später mehr.)

g) Kompilieren Sie nun ihre Monster-Implementierung, bis sie ohne Warnungen und Fehler kompiliert: *javac Monster.java*.

h) Überlegen Sie, warum es sinnvoll ist, die Methode `attack` als abstrakte Methode zu deklarieren, nicht jedoch die Methode `support`. **Schreiben Sie die Antwort als Kommentar** über die jeweiligen Methoden der Klasse Monster.

2. Schauen Sie sich nun die bereitgestellte fertige Klasse `Vampire.java` an (und speichern Sie sie im selben Verzeichnis wie `Monster.java`.

a) Kompilieren Sie diese mit *javac Vampire.java*, *um zu prüfen, dass Sie kompatibel zu ihrer Monster-Implementierung ist*.

b) Werden Fehler oder Warnungen angezeigt, muss die Monster-Implementierung korrigiert werden.

3. Erstellen Sie die Klasse `Troll`.

a) Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu. Verwenden Sie den Elternkonstruktor der Klasse `Monster`.

b) Die `String toString()`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben, damit auch das Attribut `armor` ausgegeben wird. Wenn Sie möchten, können Sie diese auf der Elternmethode aufbauen.

c) Die `void attack(Monster other)`-Methode soll die abstrakte Methode der Elternklasse implementieren[1]. Der Troll fügt dem Gegner Schaden in Höhe seiner **doppelten** `power` abzüglich seiner `armor` zu (da letztere ihn behindert). Es soll jedoch mindestens ein Schaden mit dem Wert 1 zugefügt werden, auch wenn die Rüstung viel zu schwer ist. Sorgen Sie für eine Konsolenausgabe der Namen und des Schadens.

d) Kompilieren Sie nun mit *javac Troll.java* ihre Troll-Implementierung, bis sie ohne Warnungen und Fehler kompiliert.

4. Erstellen Sie die Klasse `AxeThrowerTroll.java`.

a) Achten Sie darauf, von `Troll` statt von `Monster` zu erben!

b) Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu. Verwenden Sie den Elternkonstruktor der Klasse `Troll`.

c) Die `toString()`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben, damit auch das Attribut `numAxes` ausgegeben wird. Überlegen Sie, welcher Datentyp zurückgegeben werden muss. Wenn Sie möchten, können Sie auf der Elternmethode aufbauen.

d) Die `attack(Monster other)`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben. Der Troll schaut nach, ob er noch über Äxte verfügt. Wenn ja, wirft er eine Axt (diese geht verloren) und fügt dabei Schaden in Höhe des **dreifachen** seiner `power` zu. Hat er keine Äxte mehr, verhält er sich wie ein normaler Troll (rufen Sie dann die Elternmethode auf).

e) Die `support(Monster other)`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben. Der Troll verstärkt durch einen motivierenden Kampfschrei die `power` des anderen Monsters um den Wert 1. Sorgen Sie für eine angemessene Konsolenausgabe.

f) Kompilieren Sie nun ihre AxeThrowerTroll-Implementierung, bis Sie ohne Warnungen und Fehler

[1] Schauen Sie sich die `attack()`-Methode der Vampir Klasse an, falls Sie Schwierigkeiten haben.

kompiliert mit *javac AxeThrowerTroll.java*.

5. Erstellen Sie die Klasse `Imp.java`.
   a) Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu. Verwenden Sie den Elternkonstruktor der Klasse `Monster`.
   b) Die `toString()`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben, damit auch das Attribut `mana` ausgegeben wird. Wenn Sie möchten, können Sie auf der Elternmethode aufbauen.
   c) Die `attack(Monster other)`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben. Falls der Imp `mana` mit dem Wert 2 oder höher hat, wirft er einen Feuerball auf den Gegner, der Schaden in Höhe des `dreifachen` seiner Power verursacht (und dabei Mana mit dem Wert 2 verbraucht). Hat er kein Mana, regeneriert dieser Mana mit dem Wert 1. Sorgen Sie erneut für eine entsprechende Konsolenausgabe.
   d) Die `support(Monster other)`-Methode soll die gleichnamige geerbte Methode der Elternklasse überschreiben. Falls der Imp Mana hat, heilt er das andere Monster um das **doppelte** seiner `power` und verbraucht dabei Mana mit dem Wert 1. Falls es sich beim anderen Monster ebenfalls um einen Imp handelt (Stichwort: **`instanceof`**), wird bei diesem zusätzlich Mana mit dem Wert 1 generiert. Hat der aktive Imp kein Mana, regeneriert er Mana mit dem Wert 1. Sorgen Sie für eine angemessene Konsolenausgabe.
   e) Kompilieren Sie mit *javac Imp.java* nun ihre Imp-Implementierung, bis Sie ohne Warnungen und Fehler kompiliert.
6. Speichern Sie das bereitgestellte fertige Programm `battleground.java` im selben Verzeichnis.
   a) Kompilieren Sie es mit: *javac Battleground.java*
   b) Anschließend starten Sie das Programm mit: *java Battleground*

Die Ausgabe Ihres Testprogramms könnte z.B. so aussehen:

```
Vorbereitung...
Jaraxxus erschaffen.
Vampy erschaffen.
Leeroy Jenkins erschaffen.
Dracularius erschaffen.
Trollopa erschaffen.
Trolline erschaffen.
Wichtelantius erschaffen.
Berserkerus erschaffen.
Wichtelontias erschaffen.
Kukundi erschaffen.
Mögen die Spiele beginnen...
Jaraxxus wirft eine Axt nach Vampy an und verursacht 3 Schaden.
Vampy beisst Jaraxxus an und verursacht 2 Schaden , heilt sich um 1 Gesundheit und altert um 1.
Leeroy Jenkins wirft eine Axt nach Vampy an und verursacht 30 Schaden.
Vampy ist am Ende seiner Kraefte.
Dracularius starrt Trollopa nutzlos mit offenem Mund an.
Trollopa greift Jaraxxus an und verursacht 1 Schaden.
Trolline greift Jaraxxus an und verursacht 1 Schaden.
Wichtelantius nutzt 1 Mana und heilt Berserkerus um 8 Gesundheit.
Berserkerus wirft eine Axt nach Jaraxxus an und verursacht 3 Schaden.
Wichtelontias nutzt 1 Mana und heilt Dracularius um 8 Gesundheit.
Kukundi wirft eine Axt nach Jaraxxus an und verursacht 3 Schaden.
Jaraxxus wirft eine Axt nach Dracularius an und verursacht 3 Schaden.
Leeroy Jenkins wirft eine Axt nach Dracularius an und verursacht 30 Schaden.
Dracularius ist am Ende seiner Kraefte.
Trollopa greift Jaraxxus an und verursacht 1 Schaden.
Trolline greift Jaraxxus an und verursacht 1 Schaden.
Wichtelantius regeneriert 1  Mana.
Berserkerus laesst einen Kampfschrei erklingen, der Wichtelantius anspornt und ihm 1 Power gewaehrt.
Wichtelontias regeneriert 1 Mana.
Kukundi greift Jaraxxus an und verursacht 1 Schaden.
Jaraxxus greift Trollopa an und verursacht 1 Schaden.
```

```
Leeroy Jenkins wirft eine Axt nach Trollopa an und verursacht 30 Schaden.
Trollopa ist am Ende seiner Kraefte.
Trolline greift Jaraxxus an und verursacht 1 Schaden.
Wichtelantius regeneriert 1  Mana.
Berserkerus wirft eine Axt nach Jaraxxus an und verursacht 3 Schaden.
Wichtelontias nutzt 1 Mana und heilt Trolline um 8 Gesundheit.
Kukundi greift Jaraxxus an und verursacht 1 Schaden.
Jaraxxus greift Trolline an und verursacht 1 Schaden.
Leeroy Jenkins greift Trolline an und verursacht 1 Schaden.
Trolline greift Jaraxxus an und verursacht 1 Schaden.
Wichtelantius nutzt 2 Mana, wirft einen Feuerball auf Jaraxxus und verursacht 15 Schaden.
Berserkerus laesst einen Kampfschrei erklingen, der Wichtelantius anspornt und ihm 1 Power gewaehrt.
Wichtelontias regeneriert 1  Mana.
Kukundi greift Jaraxxus an und verursacht 1 Schaden.
Jaraxxus greift Trolline an und verursacht 1 Schaden.
Leeroy Jenkins greift Trolline an und verursacht 1 Schaden.
Trolline starrt Wichtelontias nutzlos mit offenem Mund an.
Wichtelantius regeneriert 1  Mana.
Berserkerus wirft eine Axt nach Jaraxxus an und verursacht 3 Schaden.
Jaraxxus ist am Ende seiner Kraefte.
Wichtelontias regeneriert 1  Mana.
Kukundi laesst einen Kampfschrei erklingen, der Wichtelantius anspornt und ihm 1 Power gewaehrt.
Leeroy Jenkins sieht keinen Verbuendeten mehr und ist am verzweifeln.
Trolline greift Leeroy Jenkins an und verursacht 1 Schaden.
Leeroy Jenkins ist am Ende seiner Kraefte.
Wichtelantius regeneriert 1  Mana.
Berserkerus greift Trolline an und verursacht 1 Schaden.
Wichtelontias nutzt 2 Mana, wirft einen Feuerball auf Wichtelantius und verursacht 12 Schaden.
Wichtelantius ist am Ende seiner Kraefte.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline greift Berserkerus an und verursacht 1 Schaden.
Berserkerus greift Trolline an und verursacht 1 Schaden.
Wichtelontias regeneriert 1  Mana.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline greift Berserkerus an und verursacht 1 Schaden.
Berserkerus greift Trolline an und verursacht 1 Schaden.
Wichtelontias nutzt 1 Mana und heilt Trolline um 8 Gesundheit.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline greift Berserkerus an und verursacht 1 Schaden.
Berserkerus greift Trolline an und verursacht 1 Schaden.
Wichtelontias regeneriert 1  Mana.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline greift Berserkerus an und verursacht 1 Schaden.
Berserkerus laesst einen Kampfschrei erklingen, der Kukundi anspornt und ihm 1 Power gewaehrt.
Wichtelontias regeneriert 1  Mana.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline starrt Wichtelontias nutzlos mit offenem Mund an.
Berserkerus greift Trolline an und verursacht 1 Schaden.
Wichtelontias nutzt 2 Mana, wirft einen Feuerball auf Berserkerus und verursacht 12 Schaden.
Berserkerus ist am Ende seiner Kraefte.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline greift Kukundi an und verursacht 1 Schaden.
Wichtelontias regeneriert 1  Mana.
Kukundi greift Trolline an und verursacht 1 Schaden.
Trolline greift Kukundi an und verursacht 1 Schaden.
Kukundi ist am Ende seiner Kraefte.
Wichtelontias moechte angreifen, findet aber keinen Gegner mehr.

Es gewinnt das Team B.

Sieger:
Monster:name="Trolline" team="B" health="8" power="2" armor="1"
Monster:name="Wichtelontias" team="B" health="2" power="4" mana="1"
```

Wenn Sie möchten, können Sie natürlich gerne weitere Monsterklassen implementieren und/oder ein weiteres Testprogramm `battleground_custom.java` erstellen. Zusatzpunkte werden keine vergeben.

Aufgabe 3: Kaufhaus 5.5 Punkte

- Schreiben Sie ein Interface `MehrwertBesteuerbar`, welches
  - ein Array für verschiedene Mehrwertsteuersätze enthält:
    `public double[] steuersaetze = {1.0, 1.09, 1.19};`
  - eine Methode `double preisMitMwstBerechnen()` deklariert.
- Schreiben Sie eine abstrakte Klasse `Artikel`, welche
  - das Interface `MehrwertBesteuerbar` implementiert, aber ohne `preisMitMwstBerechnen()` tatsächlich zu implementieren. (Erst in den Unterklassen – deswegen ist `Artikel` abstrakt.)
  - die Attribute `preis` (ohne Mehrwertsteuer) und `idNr` besitzt.
  - einen Konstruktor besitzt, der diese Attribute initialisiert.
  - eine `toString`-Methode besitzt, deren Rückgabestring den Preis ohne und mit Mehwertsteuer und Id des Artikels enthalten soll.
  - außerdem das Interface `Comparable` und damit die Methode `int compareTo(Object o)` implementiert. Artikel sollen anhand ihres mehrwertbesteuerten Preises verglichen werden. Achten Sie darauf, dass Preisunterschiede < 1 Euro nicht irrtümlich zu angeblicher Gleichheit führen.
- Schreiben Sie eine Klasse `Buch`, die
  - von `Artikel` erbt.
  - die zusätzlichen Attribute `titel` und `autor` besitzt.
  - einen Konstruktor enthält, der den Konstruktor von `Artikel` aufruft und die weiteren Attribute initialisiert, sowie eine `toString`-Methode besitzt, deren Rückgabestring den der Elternklasse um die weiteren Attribute ergänzt.
  - Bücher werden mit dem ermäßigten (zweiten) Mehrwertsteuersatz besteuert. Greifen Sie bei der Preisberechnung auf den im Array gespeicherten Satz zu.
- Schreiben Sie eine Klasse `TShirt`, welche
  - von `Artikel` erbt.
  - die zusätzlichen Attribute `farbe` und `groesse` besitzt.
  - einen Konstruktor enthält, der den Konstruktor der Oberklasse aufruft und die zusätzlichen Attribute initialisiert, sowie eine `toString`-Methode besitzt, deren Rückgabestring den der Elternklasse um die weiteren Attribute ergänzt.
  - T-Shirts werden mit dem normalen (dritten) Mehrwertsteuersatz besteuert. Greifen Sie bei der Preisberechnung auf den im Array gespeicherten Satz zu.
- Implementieren Sie eine Testklasse `Kaufhaus`:
  - Erstellen Sie zwei Bücher und zwei T-Shirts; speichern Sie diese Objekte zusammen in einem Array `einkaufswagen`.
  - Berechnen Sie den mehrwertbesteuerten Gesamtpreis der Waren im Einkaufswagen und geben Sie ihn aus. Bitte nutzen Sie dafür eine for-each-Schleife (https://www.google.com/search?q=java+for+each).
  - Schreiben Sie eine statische Methode `vergleichen`, die zwei Artikel mit Hilfe der `compareTo`-Methode miteinander vergleicht und eine Ausgabe auf dem Bildschirm macht, welcher Artikel um wie viel Euro teurer ist.
  - Rufen Sie ihre Vergleichsmethode mindestens zwei Mal in der `main`-Methode auf. Setzen Sie die Preise von zweien der Artikel auf unter einen Euro, dass Sie testen können, ob Ihre Implementation von `compareTo` damit umgehen kann.

Die Ausgabe von Kaufhaus sollte beispielsweise so aussehen:

```
ivo@lubi:/media/sf_host-projekte/ppr/inftech/aufgabendb/de4r/Code$ java Kaufhaus
Im Einkaufswagen befindet sich:
Artikel 100 für 9.95 Euro (10.85 Euro mit MwSt) -- Buch: Die unendliche Geschichte von Michael Ende
Artikel 101 für 14.24 Euro (15.52 Euro mit MwSt) -- Buch: Tintenherz von Cornelia Funke
Artikel 201 für 33.33 Euro (39.66 Euro mit MwSt) -- T-Shirt: Größe 36 in der Farbe blau
Artikel 202 für 33.50 Euro (39.86 Euro mit MwSt) -- T-Shirt: Größe 40 in der Farbe rot
Preis der Waren im Einkaufswagen mit Mehrwertsteuer beträgt 105.89 Euro.

Artikel 101 ist 4.68 Euro teurer als Artikel 100.
Artikel 201 ist 28.82 Euro teurer als Artikel 100.
Artikel 202 ist 0.20 Euro teurer als Artikel 201.
```