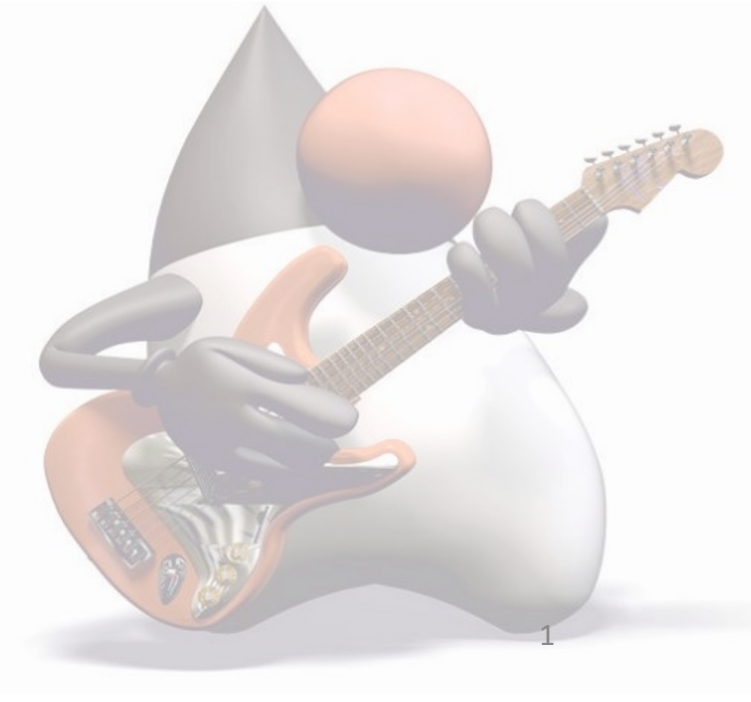
Java Fundamentals

Classes and Objects



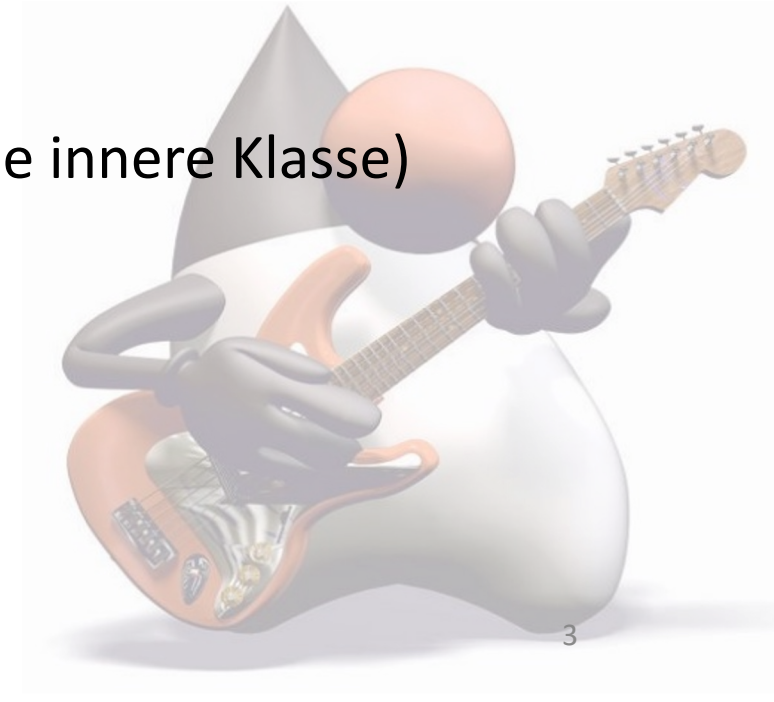
Java Klassen

- Java ist Objektorientiert
  - Object encapsulation principle
  - Erlaubt es dem Programmierer, den Speicherort und die Details wie dieser genutzt und modifiziert wird zu verstecken. Es muss nicht jeder wissen, wie etwas funktioniert, sondern nur dass es funktioniert.

- Klassen definieren Strukturen um Objekte zu beschreiben

Java Klassen

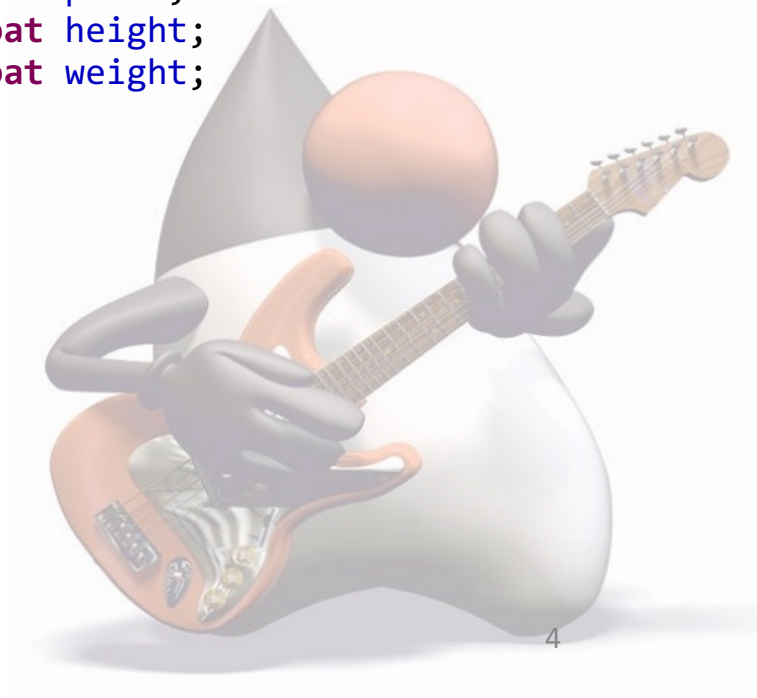
- Klassen sind eine Vorlage für Objekte
- Wird mit dem Keyword „class“ gefolgt vom Klassennamen definiert
- Der Klassenname entspricht dem Filenamen wenn die Klasse als public deklariert ist
  - Ausnahme Innere Klassen
  - Eine Java Datei kann nur eine public top-level class (keine innere Klasse) beinhalten



Klassendefinition

- Wenn man eine Klasse definiert, definiert man einen **Referenzdatentyp**
- Eine Klasse kann beliebig viele Attribute haben, die nicht an Methoden gebunden sind (Instanzvariablen)

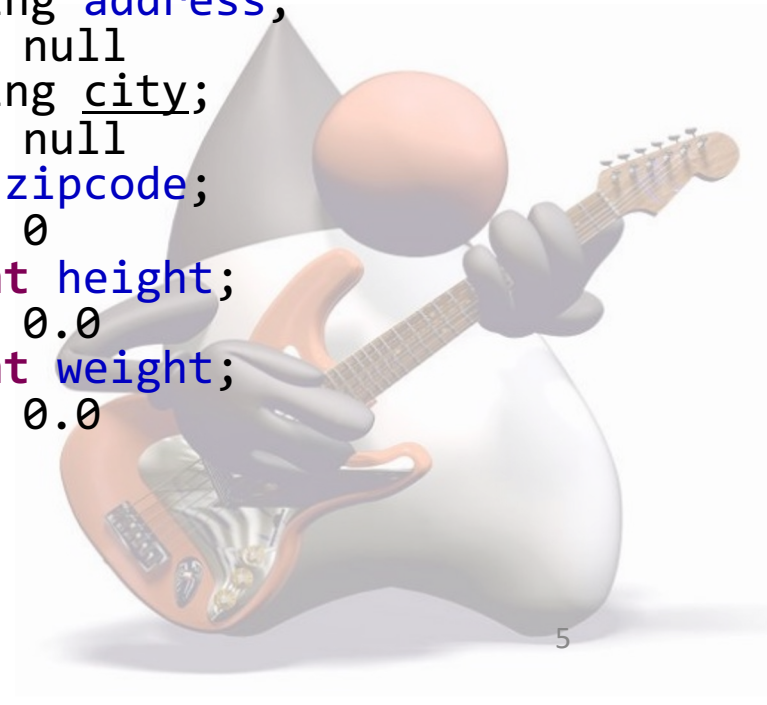
```
package at.java;

public class Customer {
        public String name;
        public String lastname;
        public String address;
        public String city;
        public int zipcode;
        public float height;
        public float weight;
}
```



Klassen

- Attribute werden initialisiert sobald ein Objekt instanziiert wird
- Primitive Datentypen werden mit ihrem Defaultwert initialisiert.
  - Z.B boolean = false, char = Unicode 0
- Referenzdatentypen werden immer mit NULL initialisiert.

```
package at.java;

public class Customer {
        public String name;
//initialized with null
        public String lastname;
//initialized with null
        public String address;
//initialized with null
        public String city;
//initialized with null
        public int zipcode;
//initialized with 0
        public float height;
//initialized with 0.0
        public float weight;
//initialized with 0.0
}
```



Klassen

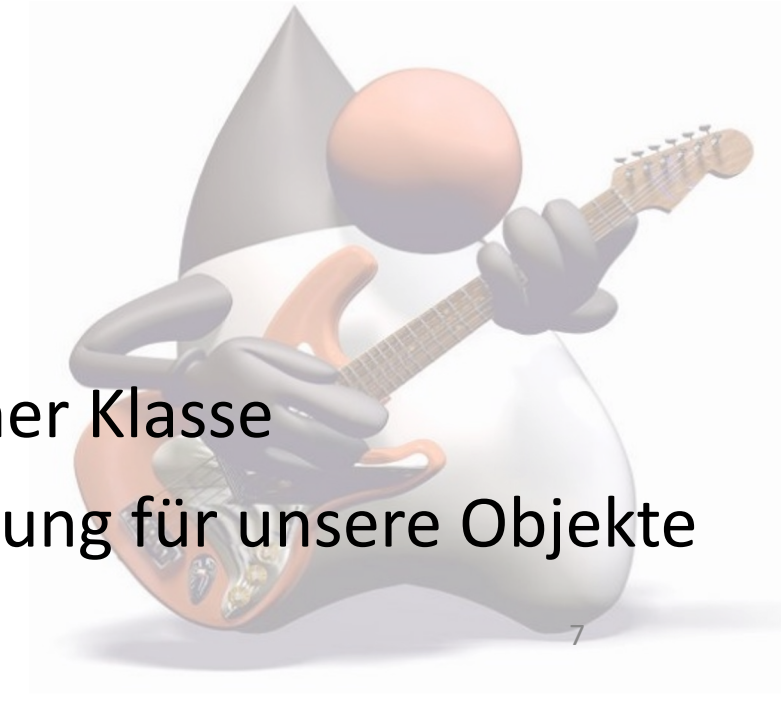
- Man kann Attribute explizit definieren.

```
public class Customer {
        public String name = "";
        public String lastname = "";
        public String address = "";
        public String city = "";
        public int zipcode = 0;
        public float height = 0.0F;
        public float weight= 0.0F;
}
```

Klassen verwenden

1. Eine Variable des entsprechenden Klassentyps definieren
2. Eine neue Instanz über den NEW Operator erstellen

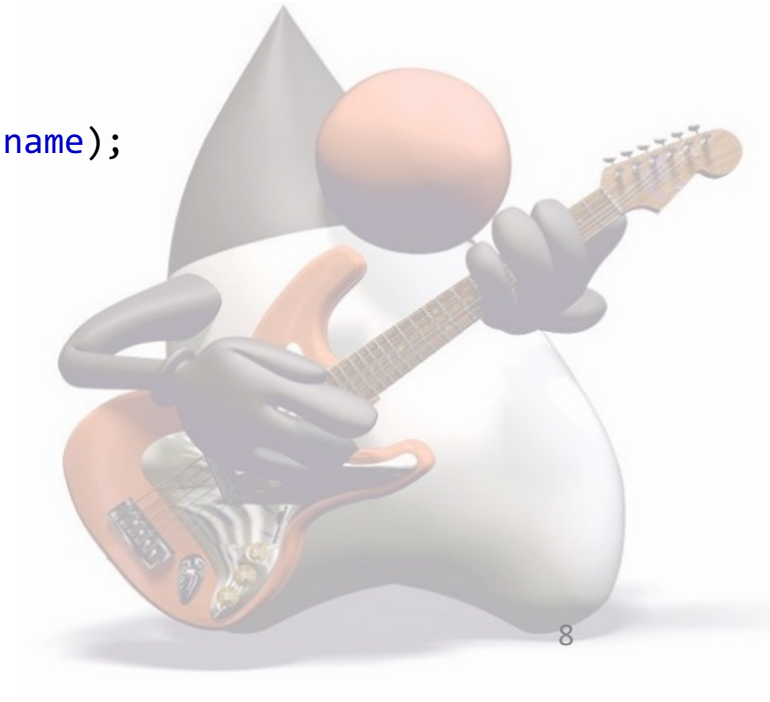
```
package at.java;

public class Main {
	public static void main(String[] args){
		Customer customer = new Customer();
	}
}
```



1. Ein Objekt ist eine Instanz einer Klasse
2. Eine Klasse ist eine Beschreibung für unsere Objekte

Nach der Instanziierung können wir auf die Attribute unseres Objekts mit dem Punkt-Operator zugreifen (lesen und schreiben)

```
public class Main {
        public static void main(String[] args){
                Customer customer = new Customer();
                customer.name = "Silvia";
                customer.lastname = "Gruber";
                customer.weight = 46F;
                customer.height = 165.F;

                System.out.println("Our customers name is: " + customer.name);
        }
}
```



Wie sieht das im RAM aus?

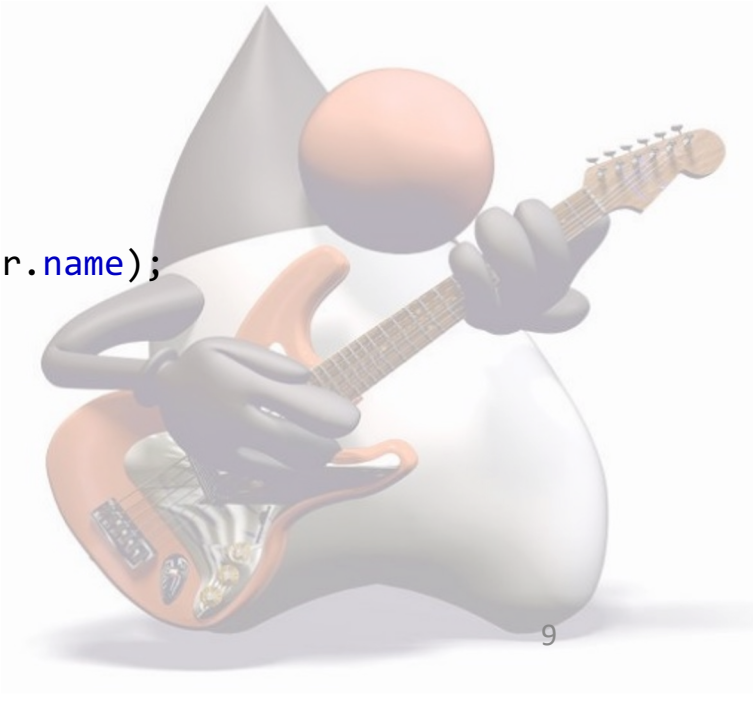
customer

0xa4f3

:Customer

memory address 0xa4f3

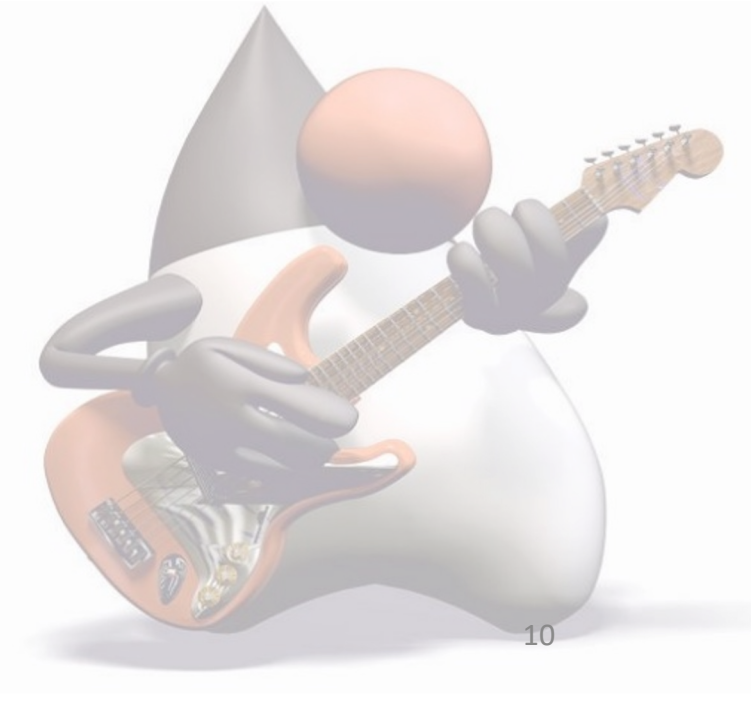
```
public class Main {
        public static void main(String[] args){
                Customer customer = new Customer();
                customer.name = "Silvia";
                customer.lastname = "Gruber";
                customer.weight = 46F;
                customer.height = 165.F;

                System.out.println("Our customers name is: " + customer.name);
        }
}
```



Man kann also sagen, der Wert einer Referenz Variable ist die Referenz Adresse des Objektes im Speicher!

Das führt zu folgender Beobachtung:

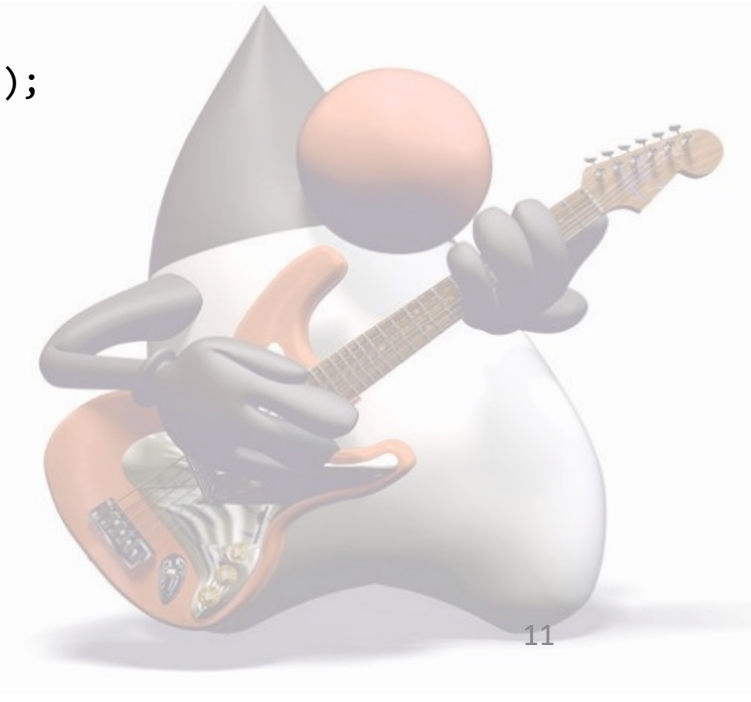
```
public class Main {
        public static void main(String[] args){
                Customer c1 = new Customer();
                c1.name = "Silvia";
                Customer c2 = c1;
                c2.name = "Sonja";
                System.out.println(c1.name);
        }
}

Output: Sonja
```



Wenn man den Punkt Operator verwendet, darf die Referenz Variable nicht NULL sein, sonst wird eine Nullpointer Exception geworfen

```
public class Main {
        public static void main(String[] args){
                Customer customer = null;
                customer.name = "Silvia";
                customer.lastname = "Gruber";
                customer.weight = 46F;
                customer.height = 165.F;

                System.out.println("Our customers name is: " + customer.name);
        }
}
```

```
Exception in thread "main" java.lang.NullPointerException
        at at.java.Main.main(Main.java:6)
```



Methoden

- Ausführbarer Code, der es erlaubt
  - Attribute zu verwenden und sie zu manipulieren
  - Berechnungen durchzuführen
- Methoden Namen
  - … sollten ein Verb enthalten z.B loadDataForEntity(Entity entity);
  - … werden CamelCase geschrieben
- Alle Java Methoden brauchen einen „Return-Type“, wenn es keinen gibt muss die Methode void returnieren.
- Es muss kein Parameter übergeben werden

Attribute und Methoden können kombiniert werden

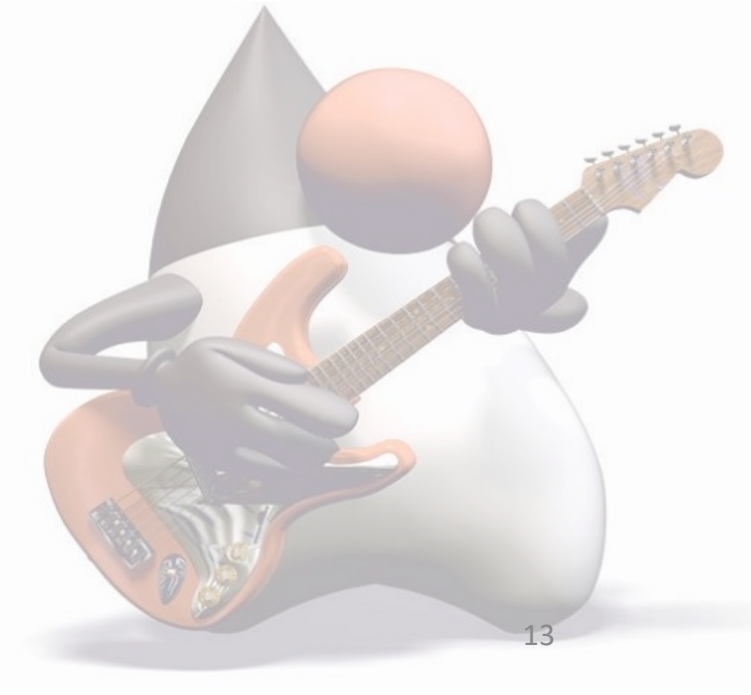
```
package at.java;

public class Customer {
        public String name;
        public String lastname;
        public String address;
        public String city;
        public int zipcode;
        public float height;
        public float weight;

        public double getBMI(){
                return weight / (Math.pow(height, 2));
        }
}
```

return type

method name

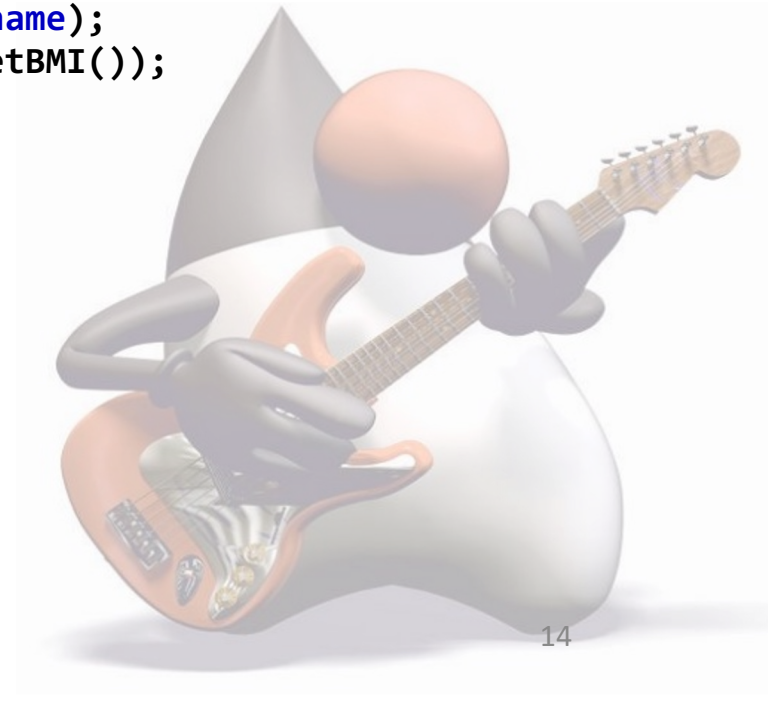
access modifier

method parameters



Aufrufen der getBMI() Method

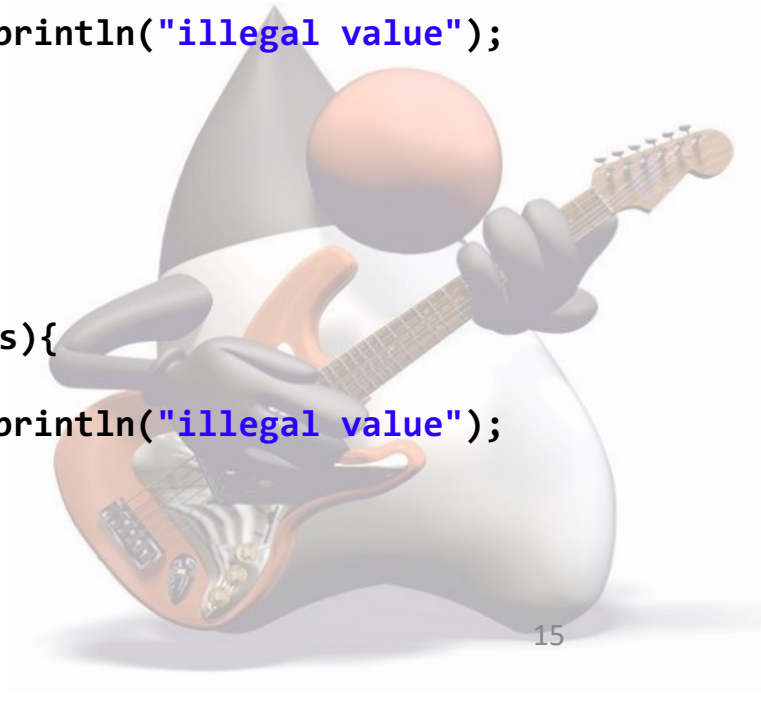
```
public class Main {
        public static void main(String[] args){
                Customer customer = new Customer();
                customer.name = "Silvia";
                customer.lastname = "Gruber";
                customer.weight = 46F;
                customer.height = 1.65F;

                System.out.println("Our customers name is: " + customer.name);
                System.out.println("The customers BMI is: " + customer.getBMI());
        }
}
```



Passing parameters

```
Customer c = new Customer();
c.weight=120F;
c.height=1.83F;
double bmi = c.getBMI();
```

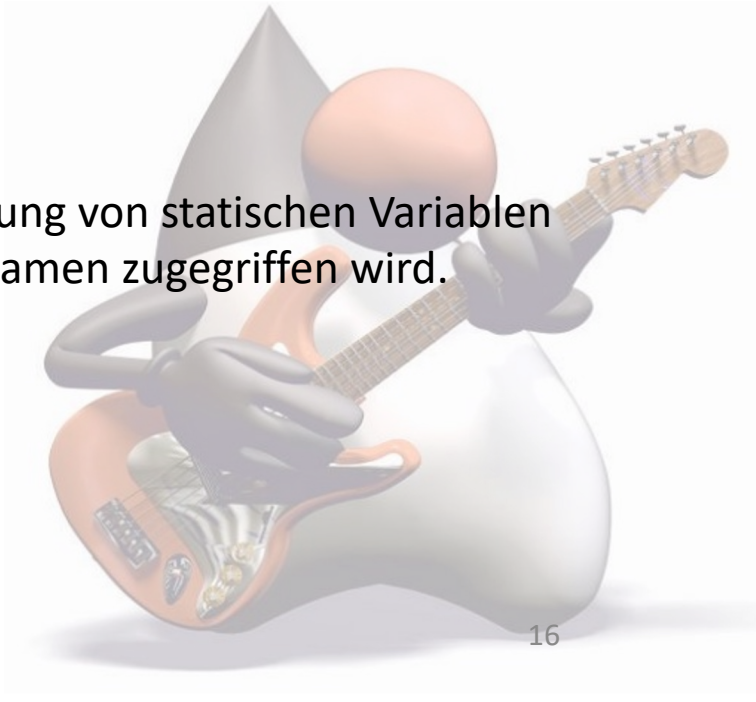
```
public class Customer {
	public String name = "";
	public String lastname = "";
	public String address = "";
	public String city = "";
	public int zipcode = 0;
	public float height = 0.0F;
	public float weight= 0.0F;

	public double getBMI(){
		return weight / (Math.pow(height, 2.));
	}

	public void eat(int kilos){
		if (kilos <= 0){
			System.out.println("illegal value");
			return;
		}
		weight += kilos;

	}

	public void excercise(int hours){
		if (hours < 0) {
			System.out.println("illegal value");
			return;
		}
		weight -= hours;
	}
}
```



Statische Variablen

```
public class StaticFields {

    public static int number;

    public static void main(String[] args) {
        System.out.println(StaticFiels.number); //static fields are initialized
        StaticFields.number = 1; //static field is overwritten
        System.out.println(StaticFields.number);
    }

}
```

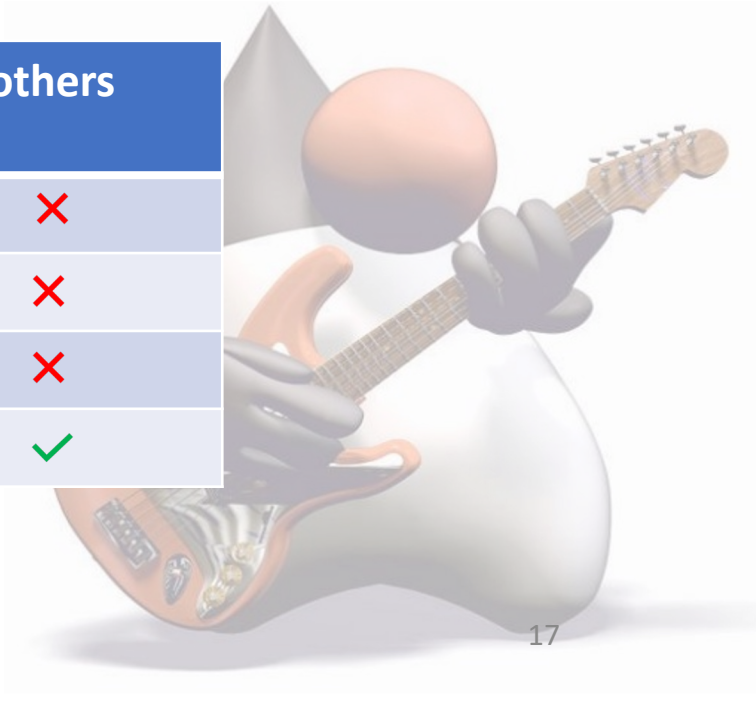
Aufgabe: Suche in dem Code, den Du bisher programmiert hast nach der Verwendung von statischen Variablen Und statischen Methoden. Du erkennst Sie daran, dass auf sie über einen Klassennamen zugegriffen wird.



Encapsulation and Access Modifiers

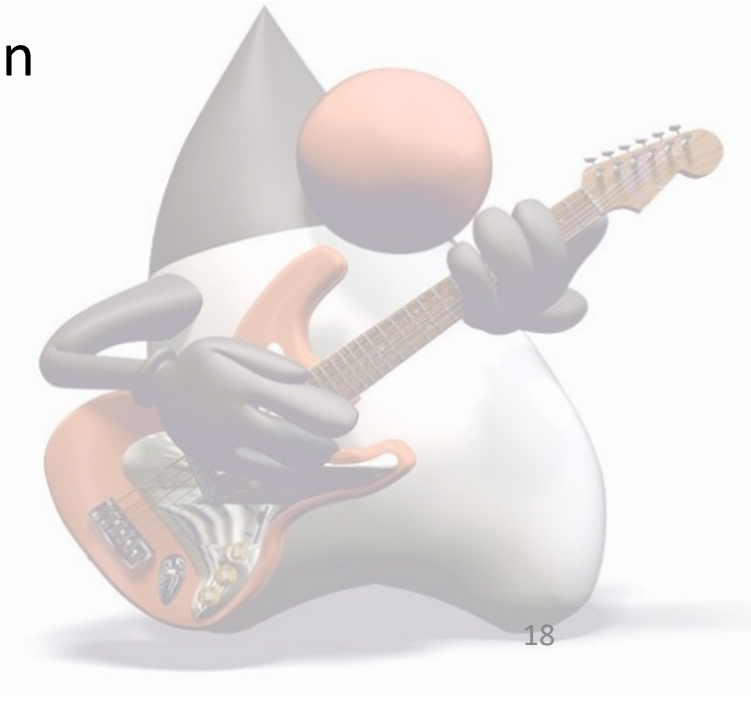
- Die interne Repräsentation eines Objekts sollte nicht bekannt sein
- Konzept: ***Encapsulation***
  - Java verwendet die Access Modifiers um encapsulation zu gewährleisten.

access modifier	same class	package	subclass	others
private	✓	✗	✗	✗
	✓	✓	✗	✗
protected	✓	✓	✓	✗
public	✓	✓	✓	✓



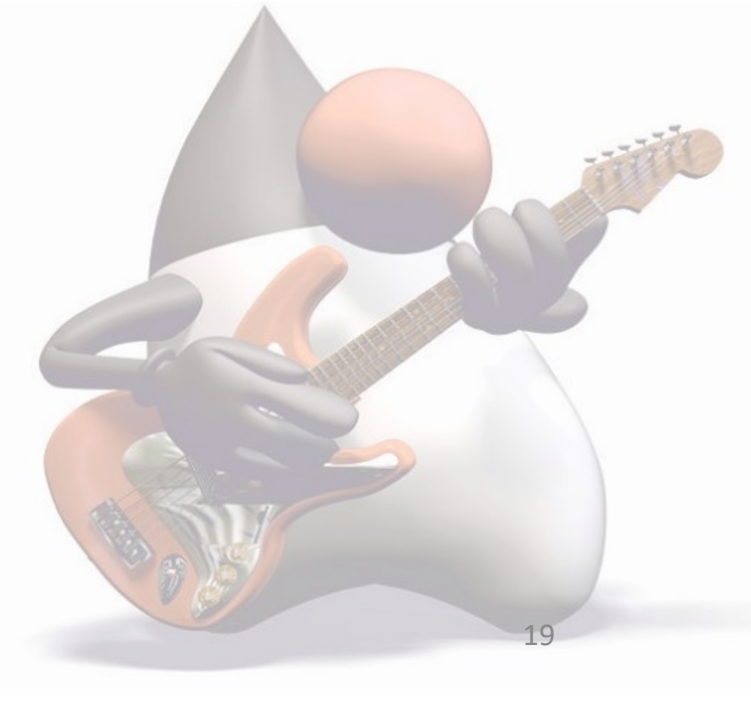
Encapsulation

- In der Regel werden die Attribute einer Klasse vor der aufrufenden Klasse verborgen
- Um auf Attribute zuzugreifen, werden Methoden verwendet
  - Ermöglicht Validierung bzw. Read-Only Attribute zu realisieren
  - Man nennt diese Methoden Getter- und Setter-Methoden
  - Dieses Konzept heißt Accessor/Mutator pattern



Attribute werden mit dem private Access Modifier vor direktem Zugriff geschützt.

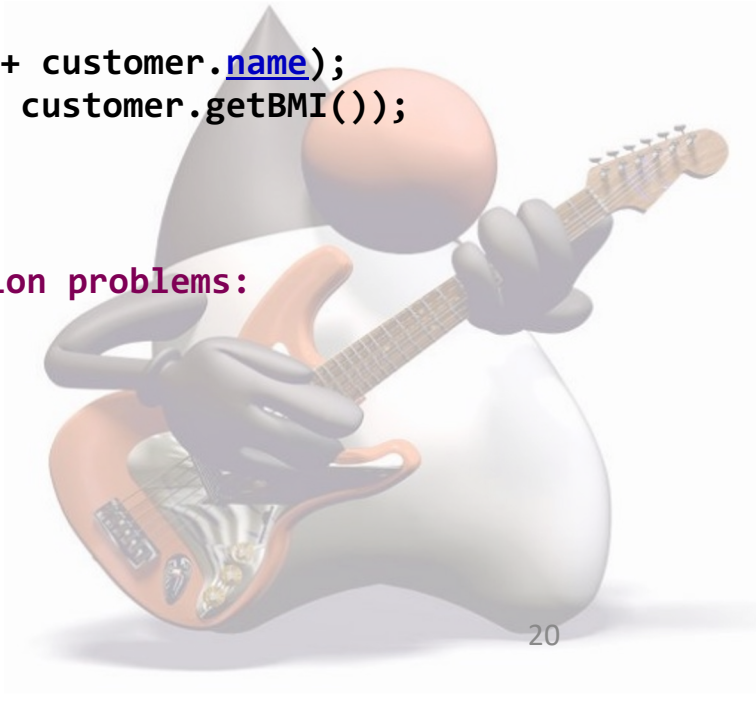
```
public class Customer {
          private String name = "";
          private String lastname = "";
          private String address = "";
          private String city = "";
          private int zipcode = 0;
          private float height = 0.0F;
          private float weight= 0.0F;

          // ...
}
```



Der Versuch von außerhalb der Klasse direkt auf die Attribute zuzugreifen führt zu einem Compile-Fehler.

```
public class Main {
	public static void main(String[] args){
		Customer customer = new Customer();
		customer.name = "Silvia";
		customer.lastname = "Gruber";
		customer.weight = 46F;
		customer.height = 1.65F;

		System.out.println("Our customers name is: " + customer.name);
		System.out.println("The customers BMI is: " + customer.getBMI());
	}
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	The field Customer.name is not visible
	The field Customer.lastname is not visible
	The field Customer.weight is not visible
	The field Customer.height is not visible
	The field Customer.name is not visible
```

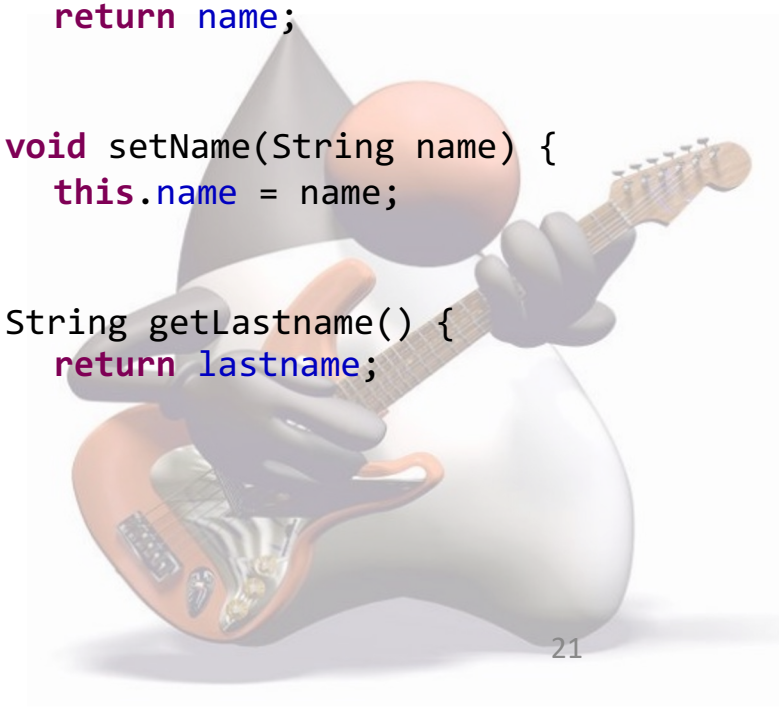


Der Zugriff auf die Attribute werden jetzt über getXXX bzw. setXXX Methode realisiert.

Hinweis:

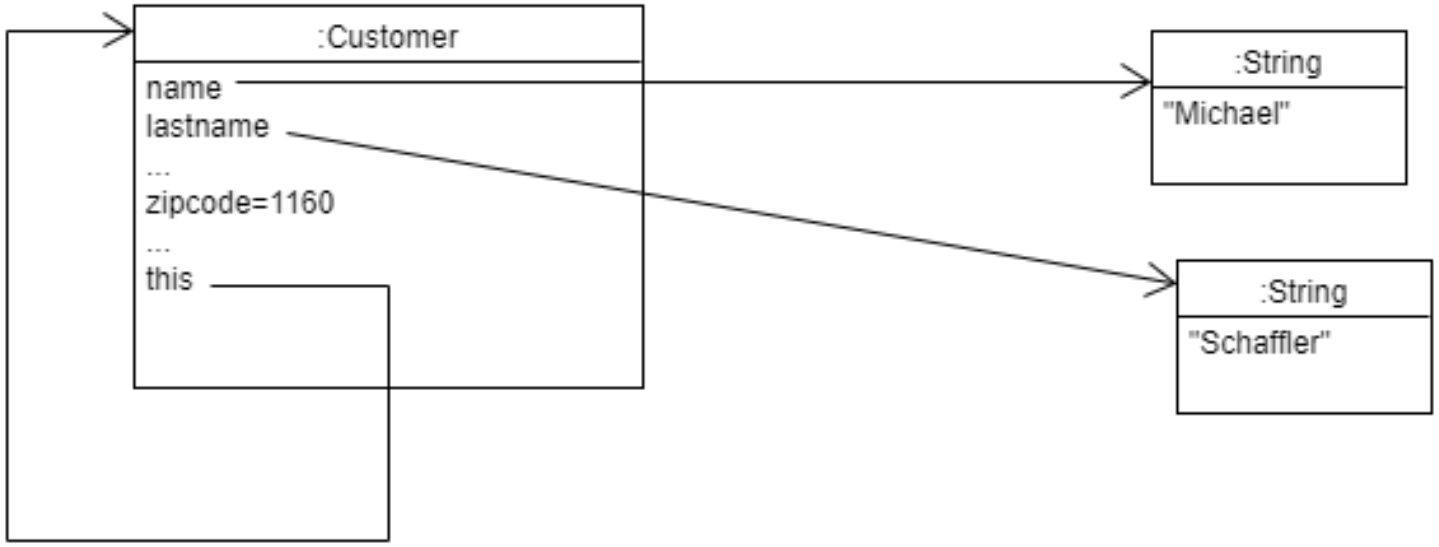
this ist eine Variable, in der die Referenz des Objektes selbst enthalten ist.

```
public class Customer {
        private String name = "";
        private String lastname = "";
        private String address = "";
        private String city = "";
        private int zipcode = 0;
        private float height = 0.0F;
        private float weight= 0.0F;


        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getLastname() {
                return lastname;
        }
// ...
```



Hinweis:

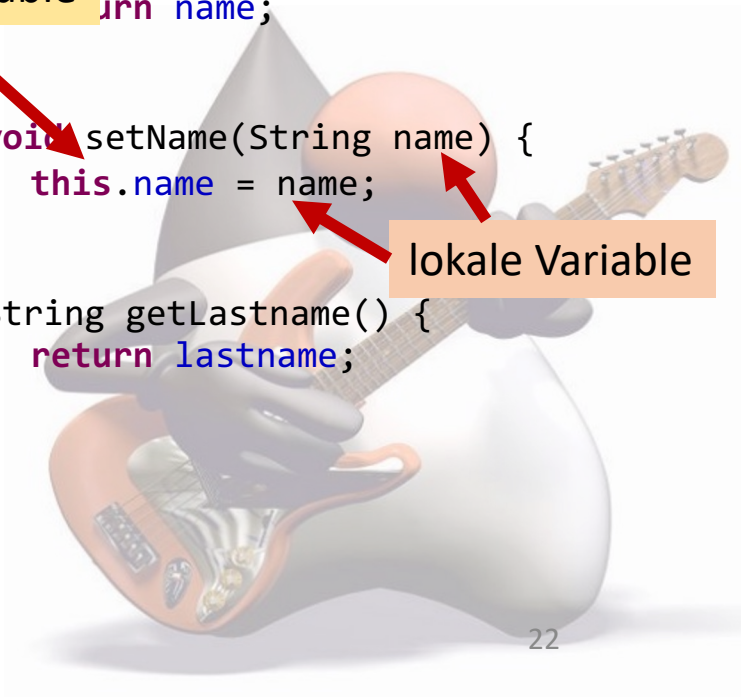
this ist eine Variable, in der die Referenz des Objektes selbst enthalten ist. Über sie kann auf Instanzvariablen zugegriffen werden, wenn diese durch eine lokale Variable verdeckt werden.




```
public class Customer {
        private String name = "";
        private String lastname = "";
        private String address = "";
        private String city = "";
        private int zipcode = 0;
        private float height = 0.0F;
        private float weight= 0.0F;


    ; getName() {
    urn name;
    }

    public voi setName(String name) {
            this.name = name;
    }

    public String getLastname() {
            return lastname;
    }
```

Instanzvariable

lokale Variable



Records (neu seit Java 16)

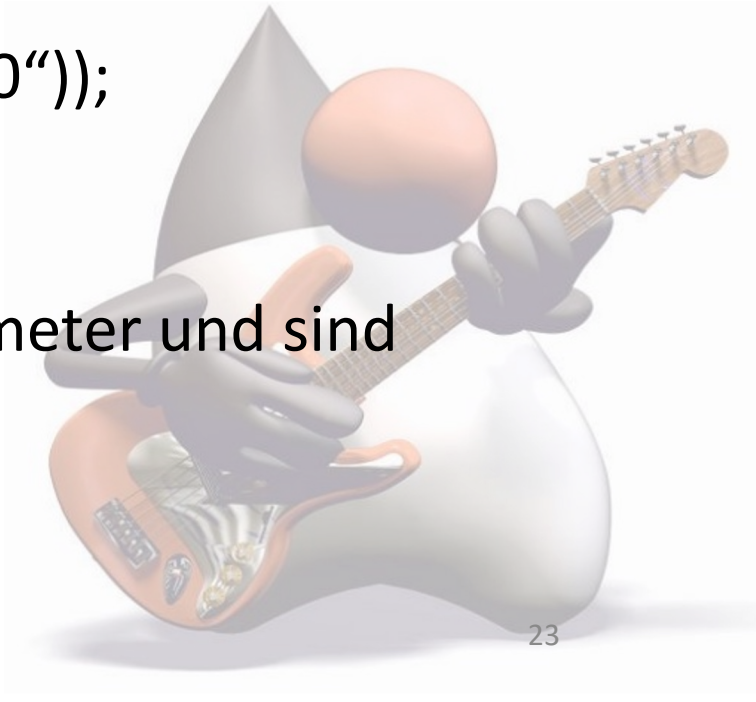
Mit Records kann man einfach immutable Objekte Erzeugen:

```
public record Article(int id, String name, BigDecimal price) {
}
```

```
Articel a = new Article(1,“T-Shirt“,new BigDecimal(„100.00“));
String name = a.name();
```

… die Parameter des Records bilden die Konstruktorparameter und sind zugleich finale Attribute (Instanzvariablen) der Klasse.

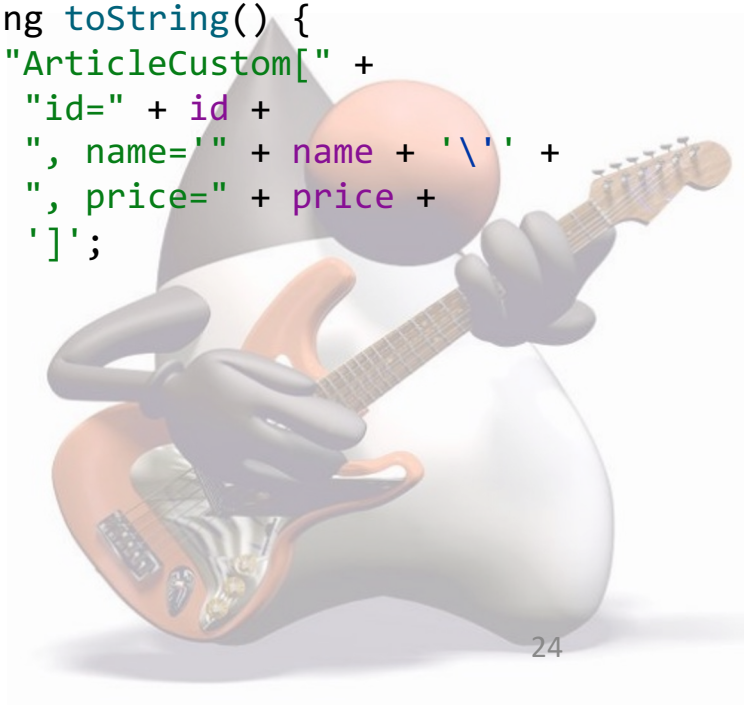
… entspricht der Implementierung ->



Records (neu seit Java 16)

```
public class ArticleCustom{
    private int id;
    private String name;
    private BigDecimal price;

    public ArticleCustom(int id, String name, BigDecimal price)
{
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public int id() {
        return this.id;
    }

    public String name() {
        return this.name;
    }

    public BigDecimal price() {
        return this.price;
    }
```

```
@Override
    public boolean equals(Object o) {
        //…
     }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, price);
    }

    @Override
    public String toString() {
        return "ArticleCustom[" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", price=" + price +
                ']';
    }
}
```



Übung 4

- Vervollständigen Sie die Customer Klasse indem sie für alle Attribute getter und setter Methoden hinzufügen.
- Erstellen sie eine Klasse „Order“ mit folgenden Attributen:
  - Order date (String)
  - itemPrice (int)
  - Item (String)
  - Quantity (int)
- Erstelle alle getter und setter Methoden für die Klasse Order
- Schreibe eine Methode „getTotalPrice()“ in der Klasse Order
- Füge ein Attribute „order“ vom Typ Order in die Klasse Customer ein
- Schreibe eine Getter und Setter Methode für dieses Attribute
- Schreibe eine Methode Customer.printCustomer welche einen String retourniert mit allen Details von Customer und seiner Order