

# Java

## Wiederholung

---

Marcus Köhler

23. November 2018

Java-Kurs

1. Wiederholung: Datentypen
2. Wiederholung: OOP
3. Wiederholung: Vererbung
4. Wiederholung: Interfaces

# Wiederholung: Datentypen

---

Welcher Datentyp ist ideal für die folgenden Werte?

1337

Welcher Datentyp ist ideal für die folgenden Werte?

1337

int oder short

c

Welcher Datentyp ist ideal für die folgenden Werte?

1337

int oder short

c

char

4.2

## Quiz zu Datentypen

Welcher Datentyp ist ideal für die folgenden Werte?

1337

int oder short

c

char

4.2

float oder double

12.345.678.910

## Quiz zu Datentypen

Welcher Datentyp ist ideal für die folgenden Werte?

1337	int oder short
c	char
4.2	float oder double
12.345.678.910	long
Peter	



## Quiz zu Datentypen

Welcher Datentyp ist ideal für die folgenden Werte?

1337	int oder short
c	char
4.2	float oder double
12.345.678.910	long
Peter	String

# Wiederholung: OOP

---

Was macht ein Objekt aus?  
Wie passen Klassen dazu?

Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.

Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.
  - Attribute sollten nicht für die Außenwelt direkt zugänglich sein (-> Verkapselung).

Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.
  - Attribute sollten nicht für die Außenwelt direkt zugänglich sein (-> Verkapselung).
  - Attribute können sowohl primitive Datentypen als auch andere Objekte sein.

Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.
  - Attribute sollten nicht für die Außenwelt direkt zugänglich sein (-> Verkapselung).
  - Attribute können sowohl primitive Datentypen als auch andere Objekte sein.
- *Methoden*, die das sogenannte Verhalten des Objekts darstellen.

Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.
  - Attribute sollten nicht für die Außenwelt direkt zugänglich sein (-> Verkapselung).
  - Attribute können sowohl primitive Datentypen als auch andere Objekte sein.
- *Methoden*, die das sogenannte Verhalten des Objekts darstellen.
  - Methoden regeln, was & wie ein Objekt agiert.



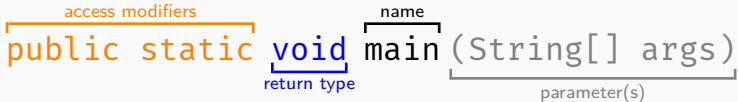
Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.
  - Attribute sollten nicht für die Außenwelt direkt zugänglich sein (-> Verkapselung).
  - Attribute können sowohl primitive Datentypen als auch andere Objekte sein.
- *Methoden*, die das sogenannte Verhalten des Objekts darstellen.
  - Methoden regeln, was & wie ein Objekt agiert.
  - Die *Signatur* einer Methode sieht folgendermaßen aus:

# Wiederholung: OOP

Ein *Objekt* besteht aus den folgenden Teilen:

- *Attribute*, welche den momentanen Zustand und die Eigenschaften des Objekts darstellen.
  - Attribute sollten nicht für die Außenwelt direkt zugänglich sein (-> Verkapselung).
  - Attribute können sowohl primitive Datentypen als auch andere Objekte sein.
- *Methoden*, die das sogenannte Verhalten des Objekts darstellen.
  - Methoden regeln, was & wie ein Objekt agiert.
  - Die *Signatur* einer Methode sieht folgendermaßen aus:

  
The diagram shows the method signature `public static void main(String[] args)` with annotations:   
- `public static` is bracketed and labeled "access modifiers" in orange.   
- `void` is bracketed and labeled "return type" in blue.   
- `main` is bracketed and labeled "name" in black.   
- `(String[] args)` is bracketed and labeled "parameter(s)" in black.

Eine *Klasse* dagegen ist sozusagen der Bauplan für ein Objekt.

- Sie definiert, welche Attribute ein Objekt haben kann und welchen Typ sie haben.

Eine *Klasse* dagegen ist sozusagen der Bauplan für ein Objekt.

- Sie definiert, welche Attribute ein Objekt haben kann und welchen Typ sie haben.
- Ebenso definiert sie die Methoden, die ein Objekt verwenden kann, kann sie aber nicht „selbst“ aufrufen.

Eine *Klasse* dagegen ist sozusagen der Bauplan für ein Objekt.

- Sie definiert, welche Attribute ein Objekt haben kann und welchen Typ sie haben.
- Ebenso definiert sie die Methoden, die ein Objekt verwenden kann, kann sie aber nicht „selbst“ aufrufen.
- Außerdem definiert sie den *Konstruktor*, eine spezielle Methode, die festlegt, wie ein Objekt erzeugt wird und welche Anfangswerte es benötigt.

Eine *Klasse* dagegen ist sozusagen der Bauplan für ein Objekt.

- Sie definiert, welche Attribute ein Objekt haben kann und welchen Typ sie haben.
- Ebenso definiert sie die Methoden, die ein Objekt verwenden kann, kann sie aber nicht „selbst“ aufrufen.
- Außerdem definiert sie den *Konstruktor*, eine spezielle Methode, die festlegt, wie ein Objekt erzeugt wird und welche Anfangswerte es benötigt.
- Eine Klasse kann auch eigene Methoden und Variablen haben; diese sind mit dem Keyword `static` markiert.

-> Code

# Wiederholung: Vererbung

---



Was ist Vererbung?  
Wozu braucht man Vererbung?

Vererbung wird verwendet, um gemeinsame Attribute und Methoden von mehreren Klassen in eine gemeinsame *Superklasse* auszulagern.

Dies hat die folgenden Auswirkungen:

- Jede Instanz(d.h. ein Objekt) einer erbenden Klasse ist automatisch auch eine Instanz ihrer Superklasse.

Vererbung wird verwendet, um gemeinsame Attribute und Methoden von mehreren Klassen in eine gemeinsame *Superklasse* auszulagern.

Dies hat die folgenden Auswirkungen:

- Jede Instanz(d.h. ein Objekt) einer erbenden Klasse ist automatisch auch eine Instanz ihrer Superklasse.
- Attribute und Methoden, die in der Superklasse als `protected` oder `public` markiert sind, sind auch in jeder erbenden Klasse („Subklasse“) verfügbar.

Vererbung wird verwendet, um gemeinsame Attribute und Methoden von mehreren Klassen in eine gemeinsame *Superklasse* auszulagern.

Dies hat die folgenden Auswirkungen:

- Jede Instanz(d.h. ein Objekt) einer erbenden Klasse ist automatisch auch eine Instanz ihrer Superklasse.
- Attribute und Methoden, die in der Superklasse als `protected` oder `public` markiert sind, sind auch in jeder erbenden Klasse („Subklasse“) verfügbar.
- Wenn in der Superklasse ein Konstruktor definiert ist, *muss* dieser im Konstruktor der Subklasse mittels `super(...)`; aufgerufen werden.

-> Code

# Wiederholung: Interfaces

---

Wofür braucht man Interfaces?  
Was macht ein Interface aus?

Ein *Interface* ist sozusagen eine Vereinbarung, dass alle Objekte, die ein Interface implementieren, mindestens die im Interface vorgegebenen Methoden umsetzt. Ein Interface ist durch die folgenden Eigenschaften gekennzeichnet:

- Es definiert Methodens**ignaturen**, die von den implementierenden Klassen „mit Funktionalität versehen werden müssen“.



Ein *Interface* ist sozusagen eine Vereinbarung, dass alle Objekte, die ein Interface implementieren, mindestens die im Interface vorgegebenen Methoden umsetzt. Ein Interface ist durch die folgenden Eigenschaften gekennzeichnet:

- Es definiert Methodens**ignaturen**, die von den implementierenden Klassen „mit Funktionalität versehen werden müssen“.
- Es definiert Konstanten(d.h. Variablen mit dem Keyword `final`), die von allen Objekten der implementierenden Klassen geteilt werden.

Ein *Interface* ist sozusagen eine Vereinbarung, dass alle Objekte, die ein Interface implementieren, mindestens die im Interface vorgegebenen Methoden umsetzt. Ein Interface ist durch die folgenden Eigenschaften gekennzeichnet:

- Es definiert Methodens**ignaturen**, die von den implementierenden Klassen „mit Funktionalität versehen werden müssen“.
- Es definiert Konstanten(d.h. Variablen mit dem Keyword `final`), die von allen Objekten der implementierenden Klassen geteilt werden.
- (Seit Java 8) Es *kann* Methoden eine „Fallback“-Definition geben, falls eine Klasse sie nicht selbst definiert(mittels `default`).

# Wiederholung: Polymorphie

Es ist auch möglich, einer Referenz einer „übergeordneten“ Klasse bzw. Interface mit einer „untergeordneten“ Klasse zu belegen:

```
1 public class Student extends Person {...}
2 public class Video implements Streamable {...}
3 [...]
4 Person student = new Student(...);
5 Streamable stream = new Video(...);
```

*Hinweis:* Mit Polymorphie beschränkt man die „Fähigkeiten“ des Objekts auf diejenigen, die in der „übergeordneten“ Klasse bzw. Interface definiert sind und kann nicht mehr auf eventuelle zusätzlich definierte Methoden in den Unterklassen zugreifen.

-> Code