Informatik II
Prüfungsvorbereitungskurs

Tag 2, 21.6.2016

Giuseppe Accaputo

g@accaputo.ch

Themenübersicht

- 20.3: **Java**
- 21.3: **Objektorientierte Programmierung**
- 22.3: **Dynamische Datenstrukturen**
- 23.3: **Datenbanksysteme**

Programm für heute

- Repetition von gestrigen Themen
- Klassen und Objekte
- Instanzvariablen und -methoden
- Klassenvariablen und -methoden
- Speicherallokation
- Vererbung
- Polymorphie
- Abstrakte Klassen

Feedback

- Gibt es Feedback zur gestrigen Vorlesung?

Repetition: **Java**

Prüfung 08.2015 Aufgabe 1b

- Was wird auf der Konsole ausgegeben?

```
static double recursive(double x){
    if (x > 1)
        return ((int)x) * recursive(x-1);
    return x;
}


System.out.println(recursive(5.2));
```

Prüfung 02.2016 Aufgabe 1a

- Nur Pass-By-Value Teil
- Whiteboard

Aufgabe Zeichenkettenvergleiche

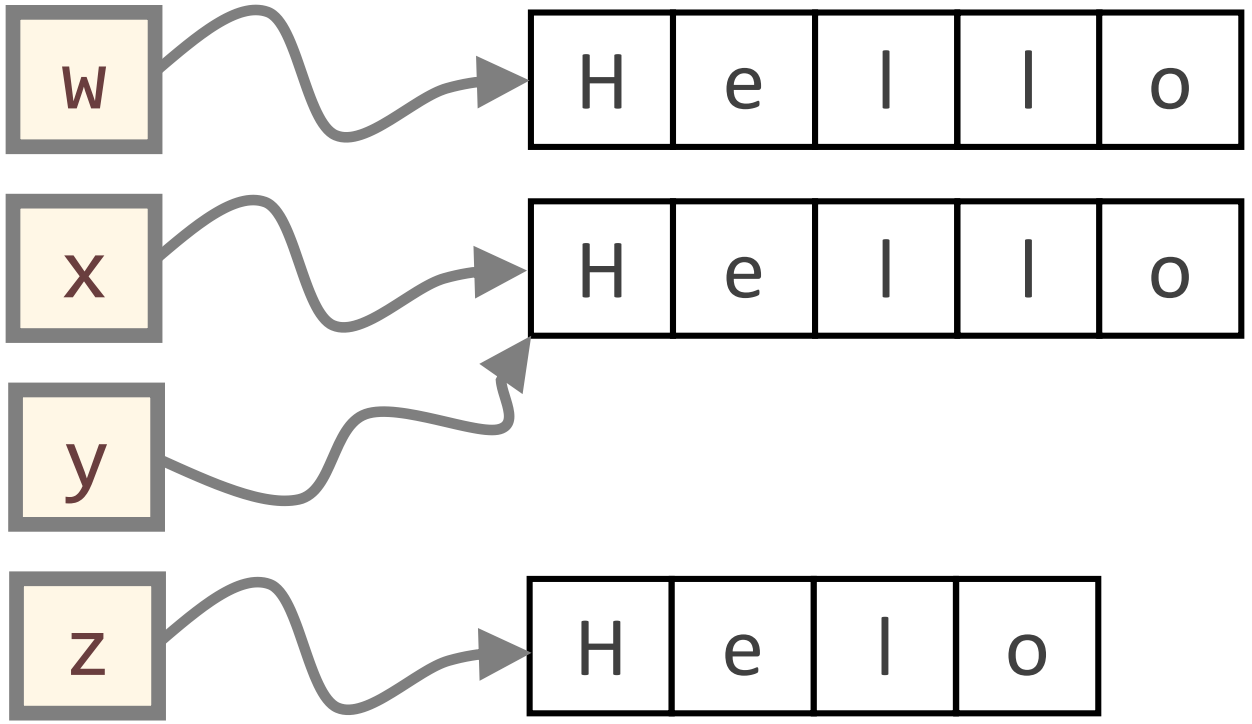
```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```

- Was ist das Ergebnis?

```
!y.Equals(z) →
        w == x →
        x == y →
```

Lösung Zeichenkettenvergleiche

```
String w = "Hello";
String x = "Hello";
String y = x;
String z = "Helo";
```




```
!y.Equals(z) → true
      w == x → false
      x == y → true
```

Repetition der Flashcards aus der Vorlesung

- Flashcard 1 (Whiteboard)
- Flashcard 2 (Whiteboard)
- Flashcard 3 (Whiteboard)

Part 2:
Objektorientierte Programmierung

Klassen und Objekte

Pascal	Java
• RECORDs in Pascal sind reine Datenobjekte. Auf ihnen wird mit Prozeduren operiert • Wertsemantik: Instanzen werden *in place* alloziert	• Klassen in Java beherbergen Daten und Code • Referenzsemantik: Instanzen müssen mit **new** alloziert werden • Instanzen heissen auch Objekte.

Java Klassen

- Java-Programm besteht aus mindestens einer Klasse
- Java-Programm hat eine Klasse mit `main`-Methode

```
public class BeispielKlasse {
   public static void main(String[] args) {
      // Code
   }
}
```

- Java Virtual Machine führt `main`-Methode bei Programmstart aus

Aufbau einer Klasse

```
public class MeineKlasse {
     public MeineKlasse(){…}

     private int i;

     public int setI(int i){…}

     public static int s;

     public static void setS(int es){…}
}
```

Konstruktor

Instanzvariablen

Instanzmethoden

Klassenvariablen

Klassenmethoden

Konstruktor

- Spezielle Methoden, die den Namen der Klasse tragen und keinen Rückgabetyp haben
- Kann Parameter haben und daher auch überladen werden
- Wird beim Aufruf mit **new** wie eine Funktion aufgerufen
- Wird kein passender Konstruktor gefunden, gibt Compiler eine Fehlermeldung aus

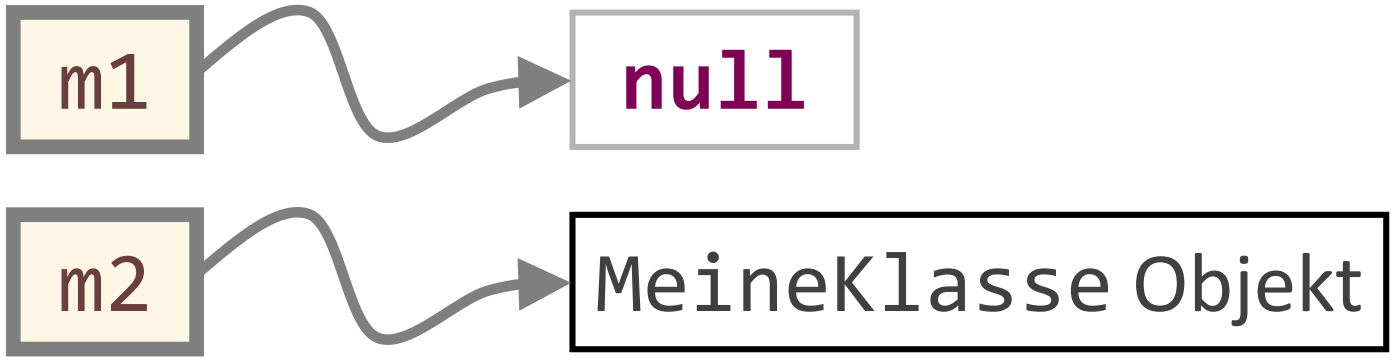
Beispiel: Konstruktor

```
public class MeineKlasse {
    private int i;
    public MeineKlasse(){…}
}


MeineKlasse m1 = new MeineKlasse();
```

Speicherallokation mit **new**

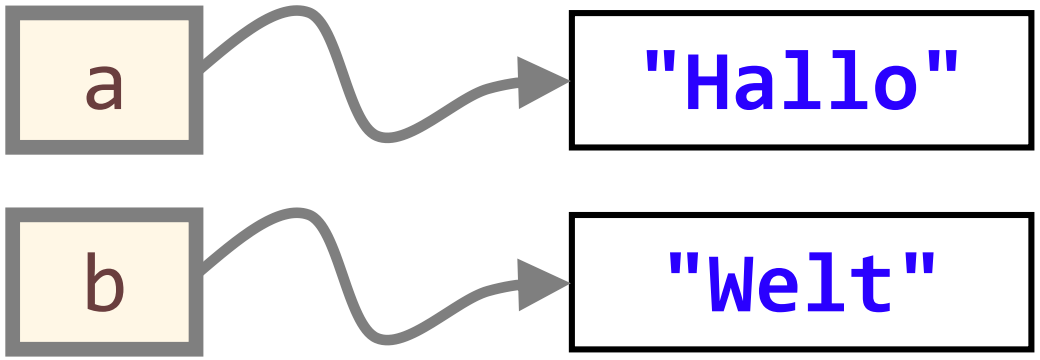
- Für die Nutzung von Klassen benötigt man dynamischen Speicher, also Speicher den man *explizit* anfordern muss
- Speicherallokation in dyn. Speicher erfolgt mittels **new**:

```
MeineKlasse m1;
MeineKlasse m2 = new MeineKlasse();
```



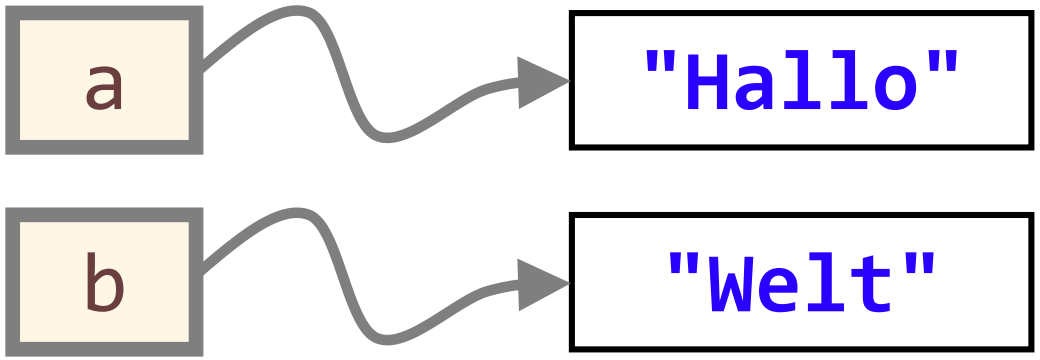
Beispiel: dynamische Speicherallokation

```
String a = new String("Hallo");
String b = new String ("Welt");
```



Beispiel: dynamische Speicherallokation

```
String a = new String("Hallo");
String b = new String ("Welt");
System.out.println(a + " " + b);
```

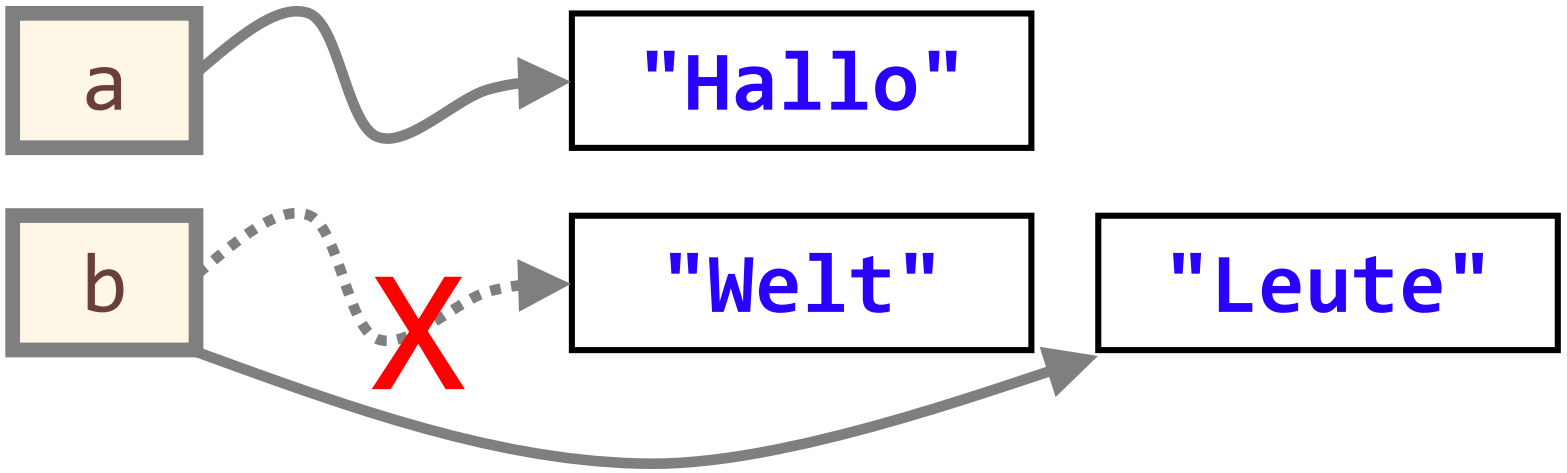


Konsole:

```
Hallo Welt
```

Beispiel: dynamische Speicherallokation

```
String a = new String("Hallo");
String b = new String("Welt");
System.out.println(a + " " + b);
b = new String("Leute");
```

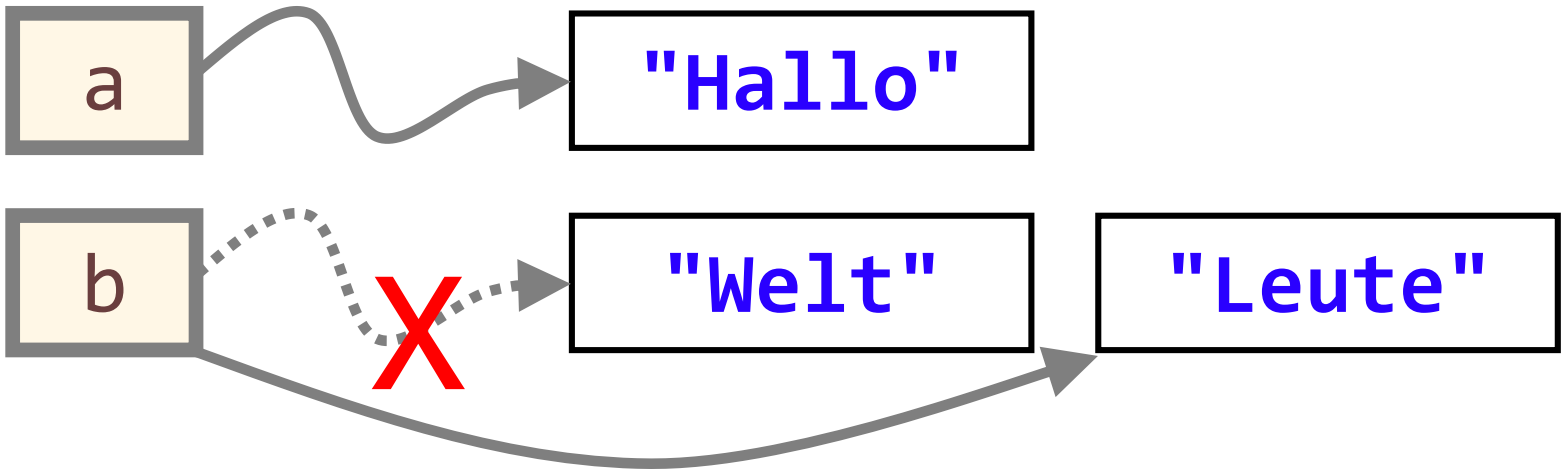


Beispiel: dynamische Speicherallokation

```
String a = new String("Hallo");
String b = new String ("Welt");
System.out.println(a + " " + b);
b = new String("Leute");
System.out.println(a + " " + b);
```

Konsole:

```
Hallo Leute
```



Überladene Funktion

- Möglich, mehrere Methoden des gleichen Namens zu definieren
- Unterscheidung erfolgt über Signatur einer Methode, welche durch Art, Anzahl und Reihenfolge der Argumente bestimmt ist:

```
static int func(int x){}
static long func(double x){}
```

- Compiler wählt automatisch Funktion aus, welche am besten passt

Beispiel: Quadrat einer Zahl berechnen

```
static int sq(int x){…}
static double sq(double x){…}
static Complex sq(Complex x){…}
```

Beispiel: Überladener Konstruktor

```
public class MeineKlasse {
    private int i;
    public MeineKlasse(){…}
    public MeineKlasse(int i){ this.i = i }
}


MeineKlasse m1 = new MeineKlasse();
MeineKlasse m2 = new MeineKlasse(10);
```

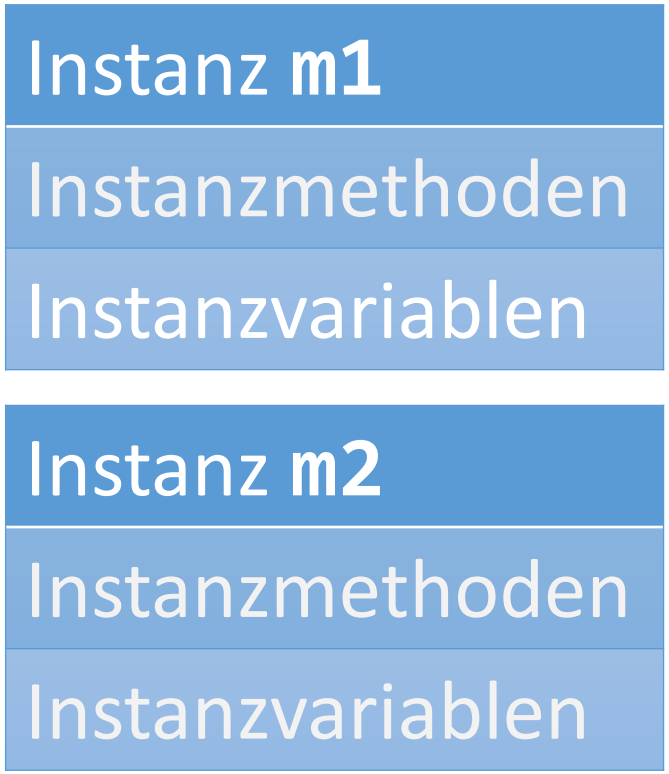
Instanzen einer Klasse

```
MeineKlasse m1 = new MeineKlasse();
MeineKlasse m2 = new MeineKlasse(10);
```

- `m1, m2` sind *Instanzen* der Klasse `MeineKlasse` und werden mittels **`new`**-Operator *instanziert*

Instanzen einer Klasse

```
MeineKlasse m1 = new MeineKlasse();
MeineKlasse m2 = new MeineKlasse(10);
```



- Jede Instanz hat eigene Kopie von Instanzmethoden und -variablen
- Ändert Instanz Wert der eigenen Instanzvariablen, so ist die Änderung nur für die Instanz geltend

Aufgabe Instanzvariablen ändern

- Was wird auf der Konsole ausgegeben?

```
public class InstVar {
    public int a = 10;
}
…
InstVar s1 = new InstVar();
InstVar s2 = new InstVar();
s2.a = 100;
System.out.println(s1.a);
```

Lösung Instanzvariablen ändern

```
public class InstVar {
    public int a = 10;
}
…
InstVar s1 = new InstVar();
InstVar s2 = new InstVar();
s2.a = 100;
System.out.println(s1.a);
```

Konsole:

```
10
```

Datenkapselung

- Fundamentales Konzept der objektorientierten Programmierung (**Prüfungsrelevant**)
- Wir verbergen interne Daten und Strukturen vor dem Zugriff von aussen
  - Wie die Datenfelder (z.B. Instanzvariablen) einer Klasse repräsentiert werden, sollte für den Benutzer nicht sichtbar sein

Klasse ohne Datenkapselung

```
public class BspKaps {
    public int a;
    public double b;
}
```

- **Fehlende Datenkapselung**: Benutzer können direkt auf die Variablen zugreifen (`public`)

Klasse mit Datenkapselung

```
public class BspKaps {
	private int a;
	public int getA() {
		return a;
	}
	public void setA(int a) {
		this.a = a;
	}

	private double b;
	public double getB() {
		return b;
	}
	public void setB(double b) {
		this.b = b;
	}
}
```

- **Datenkapselung**: Getter und Setter Methoden werden verwendet, um Zugriffe auf die Variablen zu kontrollieren

Prüfung 08.2014 Aufgabe 2

- Whiteboard

Beispiel Datenkapselung bei `Schloss`

```
public class Schloss {
    public int code = 10;
}
```

- **IST**: jedermann kann Code vom Schloss ändern, da Instanzvariable **`public`** ist
- **SOLL**: Nur wer das Masterpasswort hat, darf Schloss-Code ändern

Beispiel Datenkapselung bei Schloss

```
class Schloss {
     private int code = 10;
     private String mpw = "jl1923ikx";

     public void setCode(int code, String pw){
          if(pw.equals(mpw))
               this.code = code;
          else
               System.out.println("Falsches PW");
     }
}
```

Modifizierer für Datenkapselung

	Klasse	Paket	Sub-Klasse	Global
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	x
(keiner)	✓	✓	x	x
`private`	✓	x	x	x

- Klasse: Zugriff innerhalb Klasse, z.B. Methode auf Variable
- Paket: Zugriff zwischen Klassen innerhalb des gleichen Pakets
- Sub-Klasse: Zugriff von Sub-Klasse auf Basisklasse

Aufgabe Instanzvariablen ändern

- Kompiliert dieser Code?

```
public class InstVar2{
    public int a = 10;
    private double b = 1.12;
}
…
InstVar2 k1 = new InstVar2();
k1.a = (int)100.01023;
k1.b = (int)4.412;
```

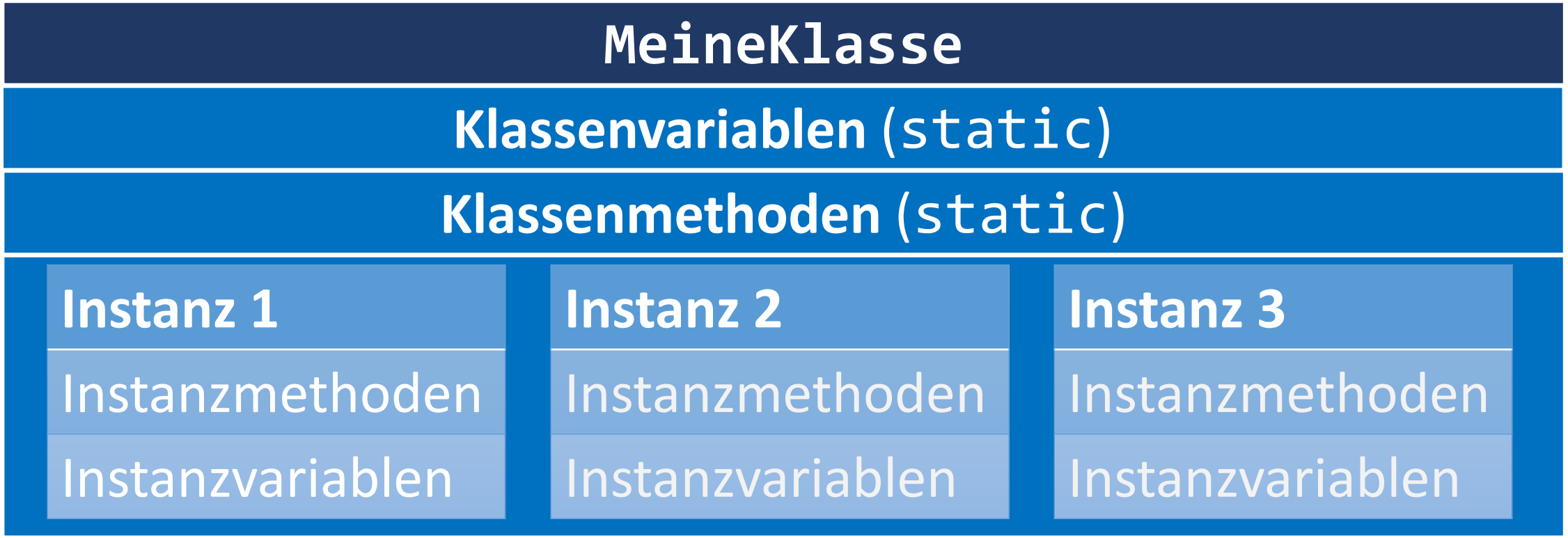
Lösung Instanzvariablen ändern

Kompilierfehler: The field b is not visible

```
public class InstVar2{
    public int a = 10;
    private double b = 1.12;
}
…
InstVar2 k1 = new InstVar2();
k1.a = (int)100.01023;
k1.b = (int)4.412; // Fehler!
```

Klassenvariablen und Klassenmethoden

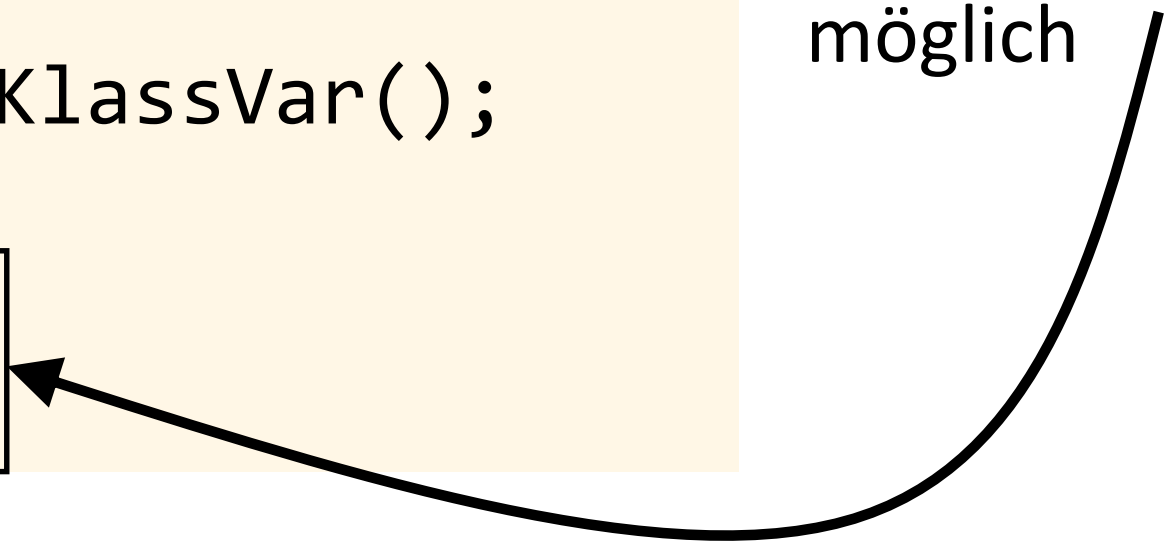
- Klassenvariablen und Klassenmethoden sind über alle Instanzen verfügbar (sofern Zugriff gewährleistet)



Zugriff auf Klassenvariablen und -methoden

```
public class KlassVar {
    public int i = 1;
    public static int s = 2;
}
…
KlassVar a = new KlassVar();
```

```
a.s = 100;
KlassVar.s = 100;
```



- Zugriff auf Klassenvariablen und -methoden über Instanz oder Klassennamen möglich

Aufgabe Klassenvariablen ändern

- Was wird auf der Konsole ausgegeben?

```
public class KlassVar {
      public int i = 1;
      public static int s = 2;
}
…
KlassVar a = new KlassVar();
KlassVar b = new KlassVar();
a.s = 100; b.s += 10;
b.i -= 10; a.s -= 100;
System.out.println(b.i);
System.out.println(b.s);
```

Lösung: Klassenvariablen ändern

```
public class KlassVar {
      public int i = 1;
      public static int s = 2;
}
…
KlassVar a = new KlassVar();
KlassVar b = new KlassVar();
a.s = 100; b.s += 10;
b.i -= 10; a.s -= 100;
System.out.println(b.i);
System.out.println(b.s);
```

Konsole:

```
-9
10
```

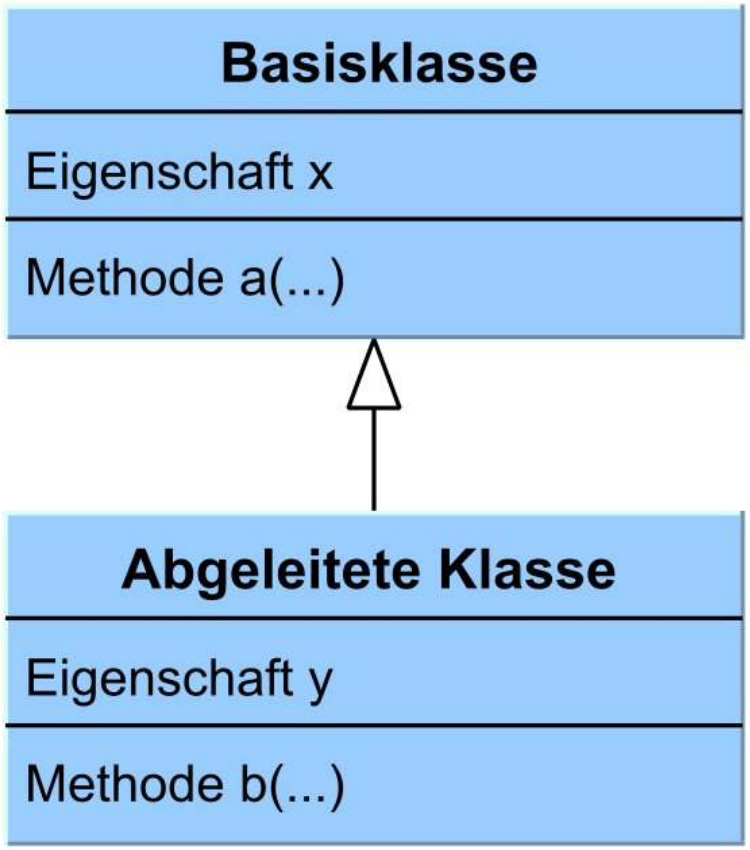
Zugriffsregeln zwischen Variablen und Methoden einer Klasse

1. Instanzmethoden können auf Instanzmethoden und Instanzvariablen direkt zugreifen
2. Instanzmethoden können auf Klassenmethoden und Klassenvariablen direkt zugreifen
3. Klassenmethoden können auf Klassenmethoden und Klassenvariablen direkt zugreifen
4. Klassenmethoden *können nicht direkt* auf Instanzmethoden und Instanzvariablen zugreifen

Vererbung

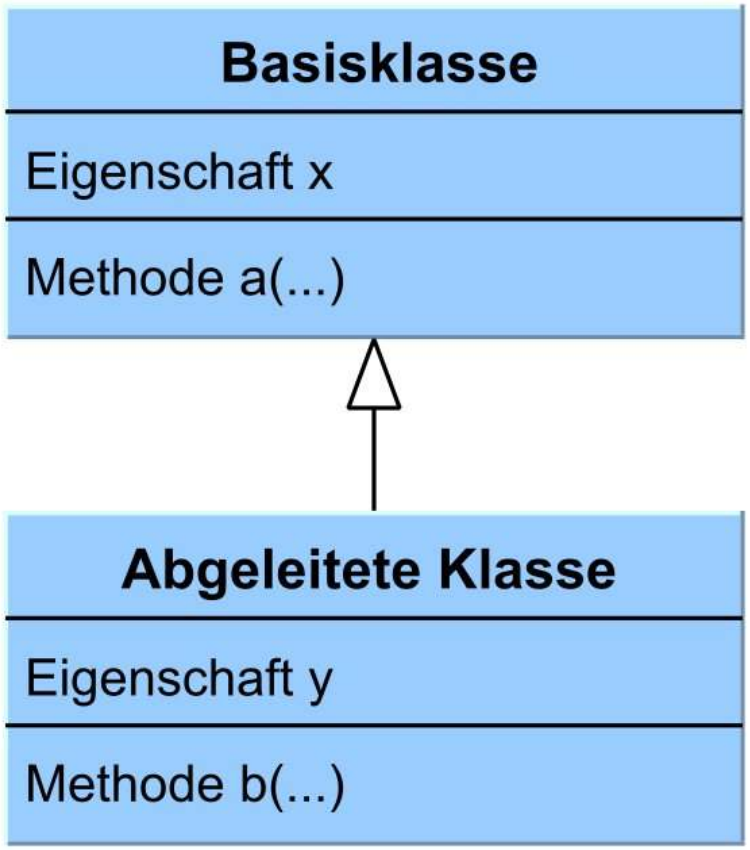
- Weiteres fundamentales Konzept der objektorientierten Programmierung (**Prüfungsrelevant**)
- Ziel: aufbauend auf existierenden Klassen neue zu schaffen, wobei Beziehung zwischen den Klassen dauerhaft ist
- Geometrische Figuren: Rechteck, Kreis, Dreieck, etc.
  - Gemeinsame Eigenschaften: Randfarbe, Füllfarbe, etc.
  - Gemeinsame Operationen: Füllen, Zeichnen, etc.
  - Unterschiede: Repräsentation, Flächenberechnung, etc.

Vererbung: Klassendiagramm



- Vererbung als Klassendiagramm dargestellt (Wikipedia)
- Vererbende Klasse wird meist *Basisklasse* genannt
- Erbende Klasse wird meist *abgeleitete Klasse* genannt

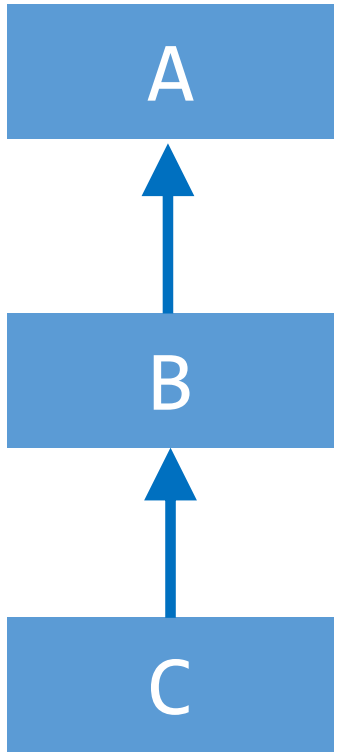
Vererbung: Klassendiagramm



- Abgeleitete Klasse und Basisklasse stehen in einer «ist-ein»-Beziehung untereinander
  - Basisklasse: Tier
  - Abgeleitete Klasse: Hund
  - Hund «ist-ein» Tier
- Gemeinsame Eigenschaften und Operationen können also zusammengefasst werden

Vererbung: Nomenklatur

```
class A{…}

class B extends A{…}

class C extends B{…}
```



Basisklasse
(Superklasse)

Abgeleitete Klasse
(Subklasse)

«B und C *erben* von A»
«C *erbt* von B»

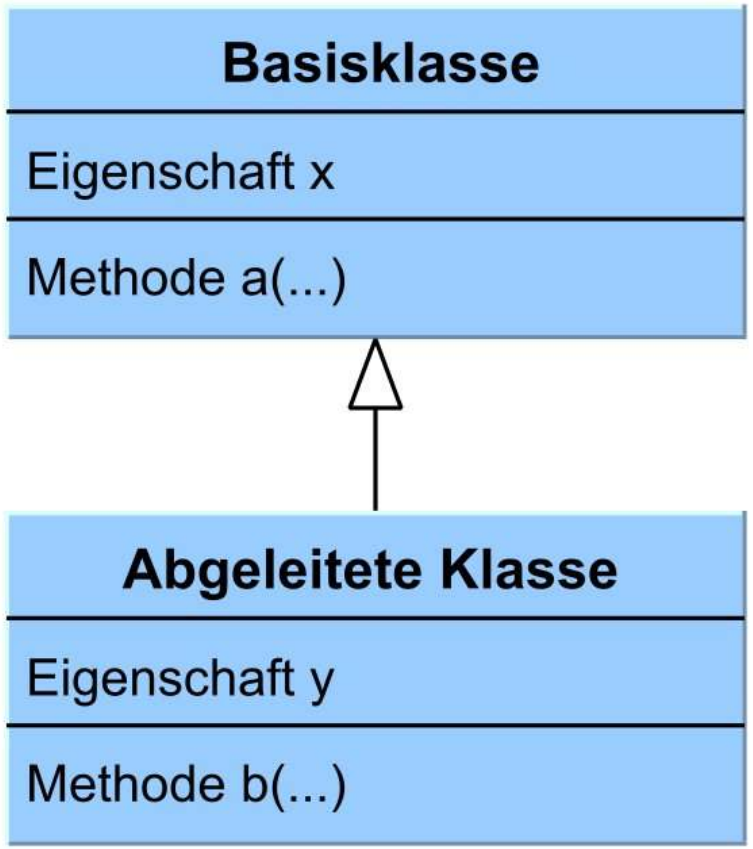
Aufgabe Klassendiagramm

- Zeichne ein passendes Klassendiagramm bestehend aus den folgenden Klassen:
  - Auto
  - Kawasaki
  - Motorrad
  - Ferrari 360
  - Lamborghini Aventador
  - Fahrzeug
  - Transportmittel
  - Flugzeug
  - Boeing 777
  - Airbus A380

Vererbung: Kompatibilität

- Abgeleitetes Objekt kann überall dort verwendet werden, wo ein Basisobjekt gefordert ist, aber nicht umgekehrt
- Im folgenden darf **K** entweder A sein (da gleicher Typ) oder eine davon abgeleitete Klasse (B oder C):

```
A a = new K();
```



Aufgabe Kompatibilität

- Kompiliert der Code rechts?

```
class A{…}
class B extends A{…}
class C extends B{…}
```

```
B b = new B();
A a = b;
C c = new B();
B b1 = new C();
B b2 = c;
```

Lösung Kompatibilität

Kompilierfehler: kann nicht von B nach C konvertieren!

```
class A{…}
class B extends A{…}
class C extends B{…}
```

C «ist-ein» B, jedoch ist B kein C!

```
B b = new B();
A a = b;
C c = new B();
B b1 = new C();
B b2 = c;
```

Vererbung: Sichtbarkeit

- In der abgeleiteten Klasse können Variablen und Methoden der Basisklasse direkt verwendet werden
- Voraussetzung: Sichtbarkeit gewährleistet!

Modifizierer bei Vererbung

	Klasse	Paket	Sub-Klasse	Global
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	x
(keiner)	✓	✓	x	x
`private`	✓	x	x	x

Prüfung 08.2015 Aufgabe 3b

- Whiteboard

Polymorphie

- Weiteres fundamentales Konzept der objektorientierten Programmierung (**Prüfungsrelevant**)
- Polymorphie in Java:
  Deklariert man eine (nicht statische) Methode mit gleichem Namen und gleicher Signatur in abgeleiteten Klassen und in der gemeinsamen Basisklasse, so wird zur Laufzeit automatisch über die auszuführende Variante entschieden.
  - Man sagt, Methode wird in der abgeleiteten Klasse *überschrieben*

Beispiel Polymorphie

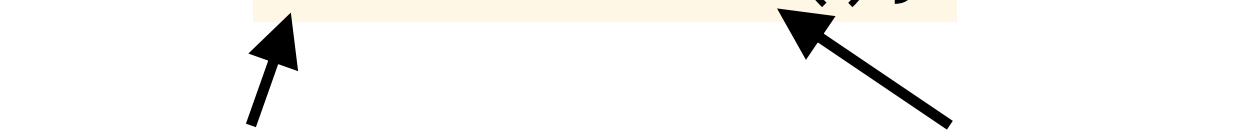
```
class A{
     public void echo(){…}
}

class B extends A{
     public void echo(){…}
}

class C extends B{
     public void echo(){…}
}
```

- echo-Methode wird von den Klassen B und C überschrieben
- Die echo-Methoden geben jeweils den Namen der Klasse aus

Polymorphie in Java

- Bei überschriebenen (nicht statischen) Methoden wird der dynamische Typ gewählt («dynamic dispatching»)

```
B b = new C();
```



Aufgabe Polymorphie in Java

- Was wird auf der Konsole ausgegeben?

```
A u = new B();
B v = new C();
A w = new C();
u.echo();
v.echo();
w.echo();
```

Lösung Polymorphie in Java

```
A u = new B();
B v = new C();
A w = new C();
u.echo();
v.echo();
w.echo();
```

Konsole:

```
B
C
C
```

Prüfung 02.2016 Aufgabe 4

- Whiteboard

Abstrakte Klassen

- Abstrakte Klassen sind spezielle Klassen, welche nicht instanziert werden können
- Abstrakte Klassen können Methoden implementieren
- Abstrakte Klassen können Konstruktoren definieren
- Innerhalb von abstrakten Klassen können abstrakte Methoden deklariert werden
- Sobald eine abstrakte Methode deklariert wird, muss die ganze Klasse als abstrakt markiert werden

Abstrakte Methoden

- Abstrakte Methoden sind Methoden, welche in einer abstrakten Basisklasse definiert, jedoch erst in der vererbenden Klasse implementiert werden

```
abstract class Beispiel{
    abstract public double run();
}
```