„You've got questions.
We've got dancing
paperclips."

- Microsoft

Sexy Unix Commands: date; unzip; touch; strip; finger; mount; gasp; yes; uptime;

Lehreinheit im Javakurs der Freitagsrunde
SoSe2008
Technische Universität Berlin

Mario Bodemann
Björn Lohrmann

Übersicht

- **Wiederholung**
- Java Packages
- Vererbung
- Kapselung Revisited
- toString()
- equals()

Wiederholung LV 5 - Codebeispiel

```
class Human {
   String name;

   Student(String name) {
      this.name = name;
   }

   void say(String message) {
      System.out.println(this.name+": "+message);
   }
}
```

Wiederholung: Objekte und Klassen

- Was war eine Klasse?
  - Im obigen Beispiel: **Human**
  - Allgemein: Der Bauplan eines Objektes
- Was war dann ein Objekt?
  - Als Beispiel:
    Human luigi = new Human("luigi");
  - Allgemein:
    ein konkrete Ausprägung einer Klasse

Wiederholung: Attribute und Verhalten

- Was waren Attribute?
  - Im Beispiel: **name**
  - Eigenschaften, die ein Objekt dieser Klasse haben kann
- Was war Verhalten?
  - Im Beispiel: **void say(String message)**
  - Methoden die ein Objekt der Klasse haben kann

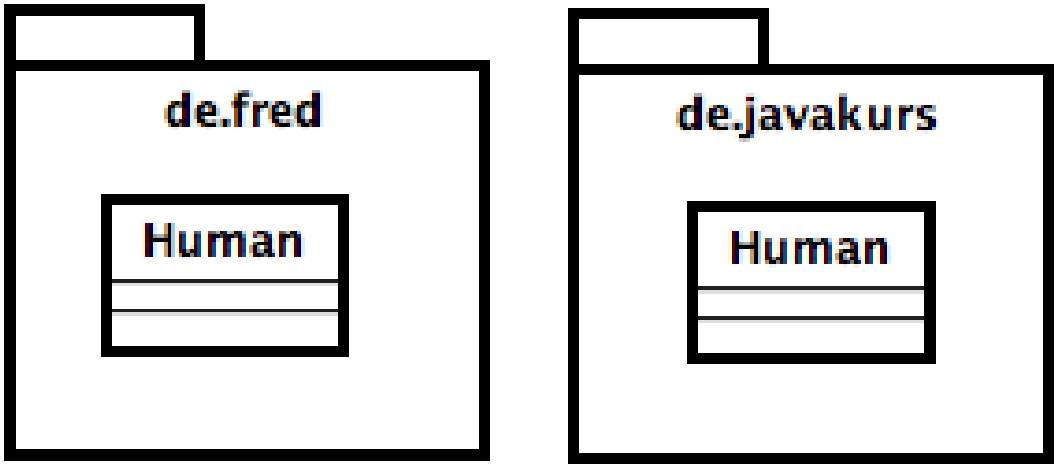
Übersicht

- Wiederholung
- **Java Packages**
- Vererbung
- Kapselung Revisited
- toString()
- equals()

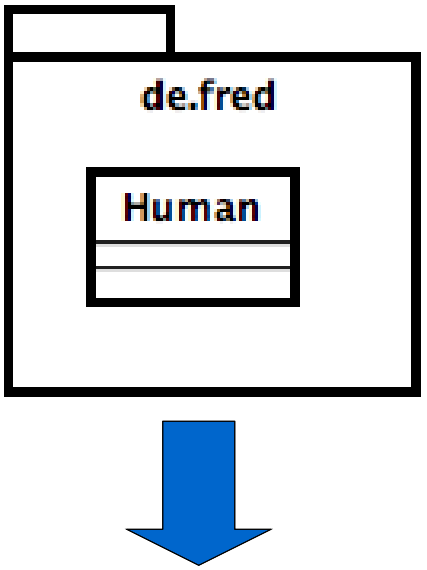
Situation: Fred gibt euch seine Klasse „Human“

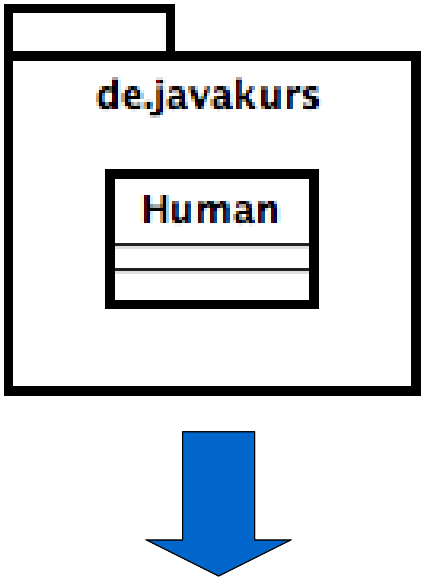
Problem: Zwei Dateien, gleicher Name

Lösung: Fred gibt euch seine Klasse(n) in einem Package!



Java Packages






```
package de.fred;

class Human {

}
```

```
package de.javakurs;

class Human {

}
```

Sagt Compiler: Human = de.javakurs.Human

Benutzung [illegible]klassen aus anderen packages:

```
import de.javakurs.Human;

class MyMain {

   public static void main(String[] args) {
      Human h = new Human();
   }
}
```

Problem:
In welchem package waren uns Beispiele bisher?

- **Lösung:**
- Bisher keine Angabe von „package xy“
- Daher im *default*-package

ACHTUNG
Klassen aus *default*-package sind nicht importierbar!

Übersicht

- Wiederholung
- Java Packages
- **Vererbung**
- Kapselung Revisited
- toString()
- equals()

Vererbung

```
class Human {
    String name;

    Human(String name) {
        this.name = name;
    }

    void sayName() {
        System.out.println(name);
    }
}
```

Problem: Wir wollen....

- Neue Klasse „Child“
- „Child“ hat Namen
- „Child“ hat Lieblingseis

Lösung: Vererbung

Vererbung (Bsp. Fortsetzung)

Child erbt von Human

Human = Superklasse

Child = Subklasse

```
class Child extends Human {
   String favIcecream;

   Child(String favIcecream) {
      this.favIcecream = favIcecream;
   }

   void orderIcecream() {
      System.out.println(this.favIcecream
         + " icecream, please!");
   }
}
```

Achtung: Child-Klasse kompiliert (noch) nicht!

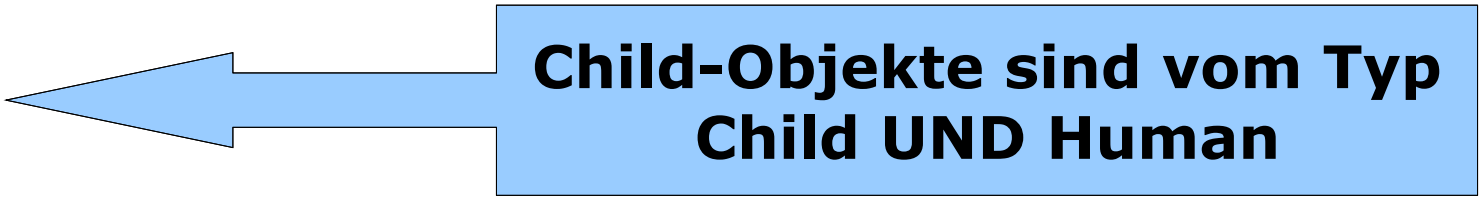
Was passiert bei Vererbung?

Was wird vererbt?

- **Attribute**

- **Methoden**

- **Typ**



Benutzung geerbter Attribute:

```
class Child extends Human {
   String favIcecream;

   Child(String favIcecream) {
      this.favIcecream = favIcecream;
   }

   void orderIcecream() {
      System.out.println("Hi, I am " + this.name);
      System.out.println(this.favIcecream
         + " icecream, please!");
   }
}
```

Geerbter Name

Vererbung von Attributen

Problem: Child-Klasse kompiliert nicht!

Grund: Name wird nicht initialisiert

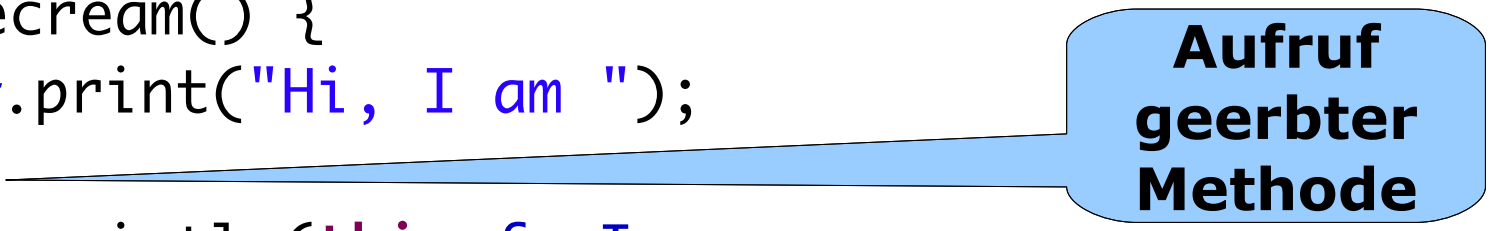
Lösung: Konstruktor korrigieren

```
Child(String favIcecream, String name) {
    super(name);
    this.favIcecream = favIcecream;
}
```

Aufruf Human-Konstruktor

Benutzung geerbter Methoden:

```
void orderIcecream() {
    System.out.print("Hi, I am ");
    sayName();
    System.out.println(this.favIcecream
        + " icecream, please!");
}
```



Vererbung von Methoden (Overriding)

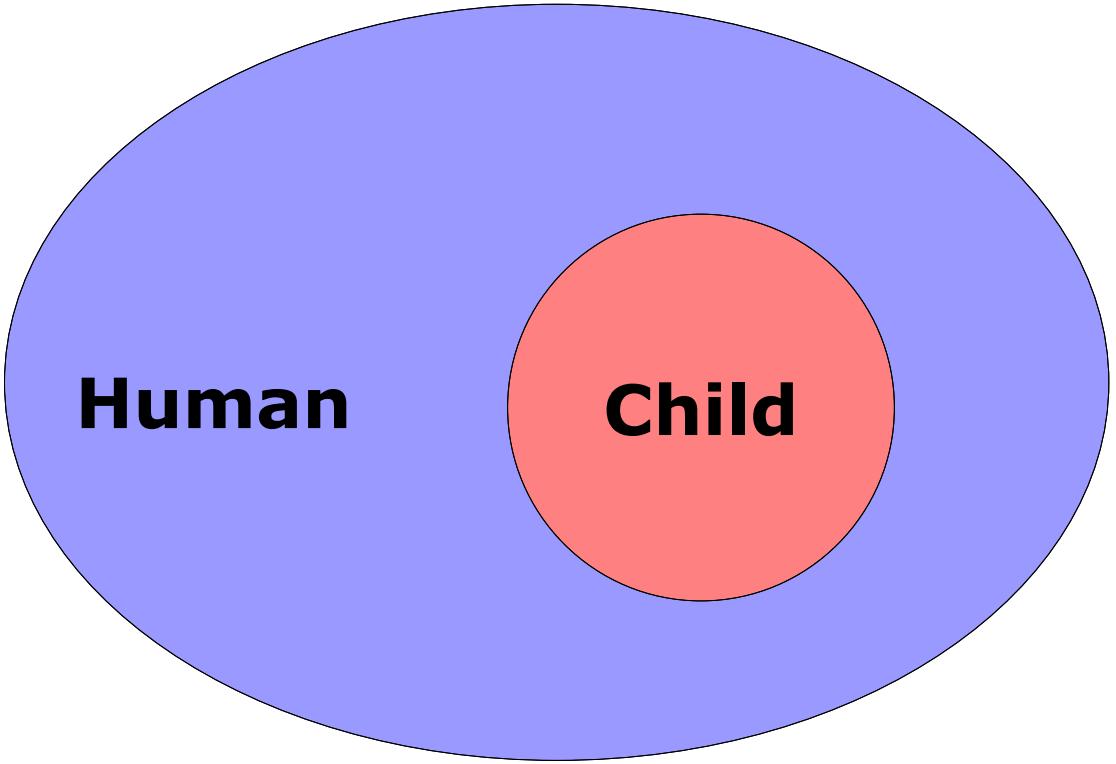
Problem: Child soll bei sayName() etwas anderes sagen, als Human.

Lösung: Overriding! Child definiert sayName() nochmal.

```
class Child extends Human {

   [...]

   void sayName() {
      System.out.println(„gaga“
         + this.name
         + „brubbel“);

   }
}
```

Vererbung des Typs

Child ist Subtyp von Human
=
Child ist Human, aber Human nicht immer Child



Vererbung des Typs (Bsp. 1)

```
Human fred = new Human("Fred");
Child freddie = new Child("Vanilla", "Freddie");

fred = freddie;
```

fred.sayName(); **RICHTIG**

fred.orderIcecream(); **FALSCH**

Vererbung des Typs (Bsp. 2)

```
Human fred = new Human("Fred");
Child freddie = new Child("Vanilla", "Freddie");

fred = freddie;
```

freddie = fred; **FALSCH**

freddie = (Child) fred; **RICHTIG**

```
freddie.orderIcecream();
```

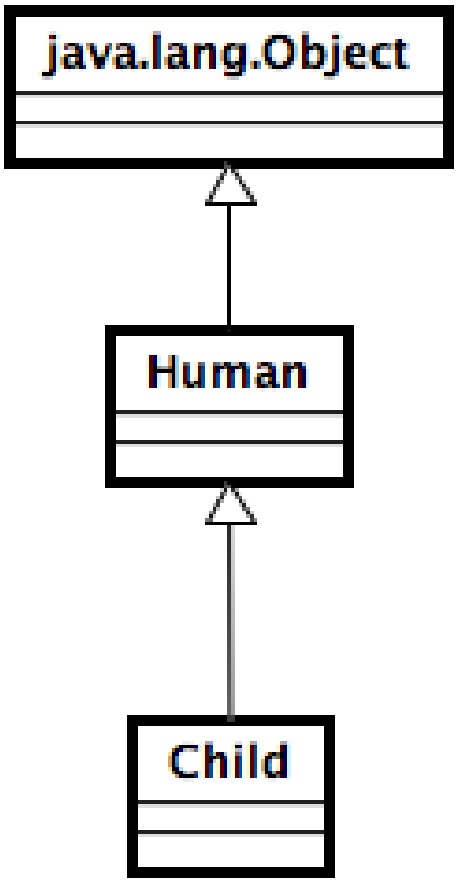
Vererbung des Typs (Bsp. 3)

```
Human fred = new Human("Fred");
Child freddie;

freddie = (Child) fred;

freddie.orderIcecream();
```

Kompiliert der Code? **JA**

Funktioniert der Code zur Laufzeit? **NEIN**

Jede Java-Klasse ist Subklasse von java.lang.Object

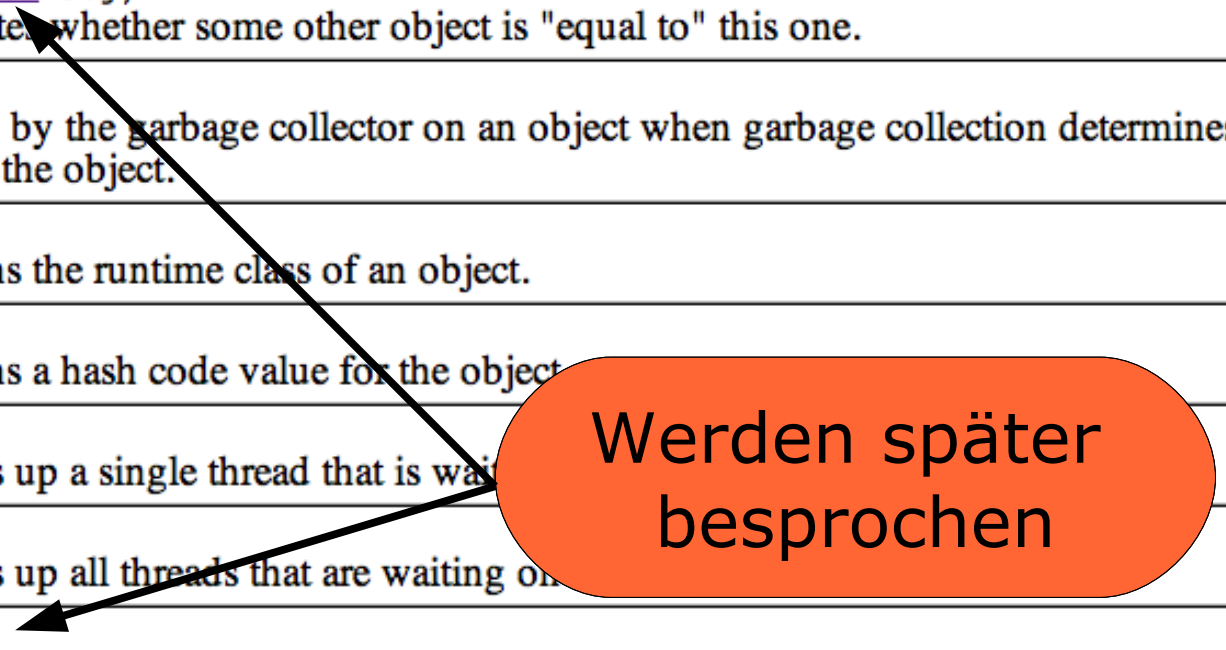


Constructor Summary

Object()

Method Summary

protected Object	**clone**() Creates and returns a copy of this object.
boolean	**equals**(Object obj) Indicates whether some other object is "equal to" this one.
protected void	**finalize**() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class	**getClass**() Returns the runtime class of an object.
int	**hashCode**() Returns a hash code value for the object.
void	**notify**() Wakes up a single thread that is wai
void	**notifyAll**() Wakes up all threads that are waiting on
String	**toString**() Returns a string representation of the object.
void	**wait**() Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	**wait**(long timeout) Causes current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	**wait**(long timeout, int nanos) Causes current thread to wait until another thread invokes the notify() method or the notifyAll()



Übersicht

- Wiederholung
- Java Packages
- Vererbung
- **Kapselung Revisited**
- toString()
- equals()

Problem: Ein Human soll immer (gültigen) Namen haben!

Lösung: Zugriff auf „name“ Attribut einschränken.

```
public class Human {
    private String name;

    [...]

    String getName() {
        return name;
    }

    void setName(String newName) {
        if (newName != null && newName.length() > 0) {
            this.name = newName;
        }
    }
}
```

Nur Human-Klasse hat noch Zugriff

Kapselung von Attributen:
- **Kein direkter Zugriff von außen**
- **Zugriff nur über Methoden (`getName()`, `setName()`)**

Kapselungsstufen in Java:

- **Public:**
  - → **`public void say() {...}`**
- **Private:**
  - → **`private void say() {...}`**
- **Ohne Modfizierer:**
  - → **`void say() {...}`**

Anwendbar auf:

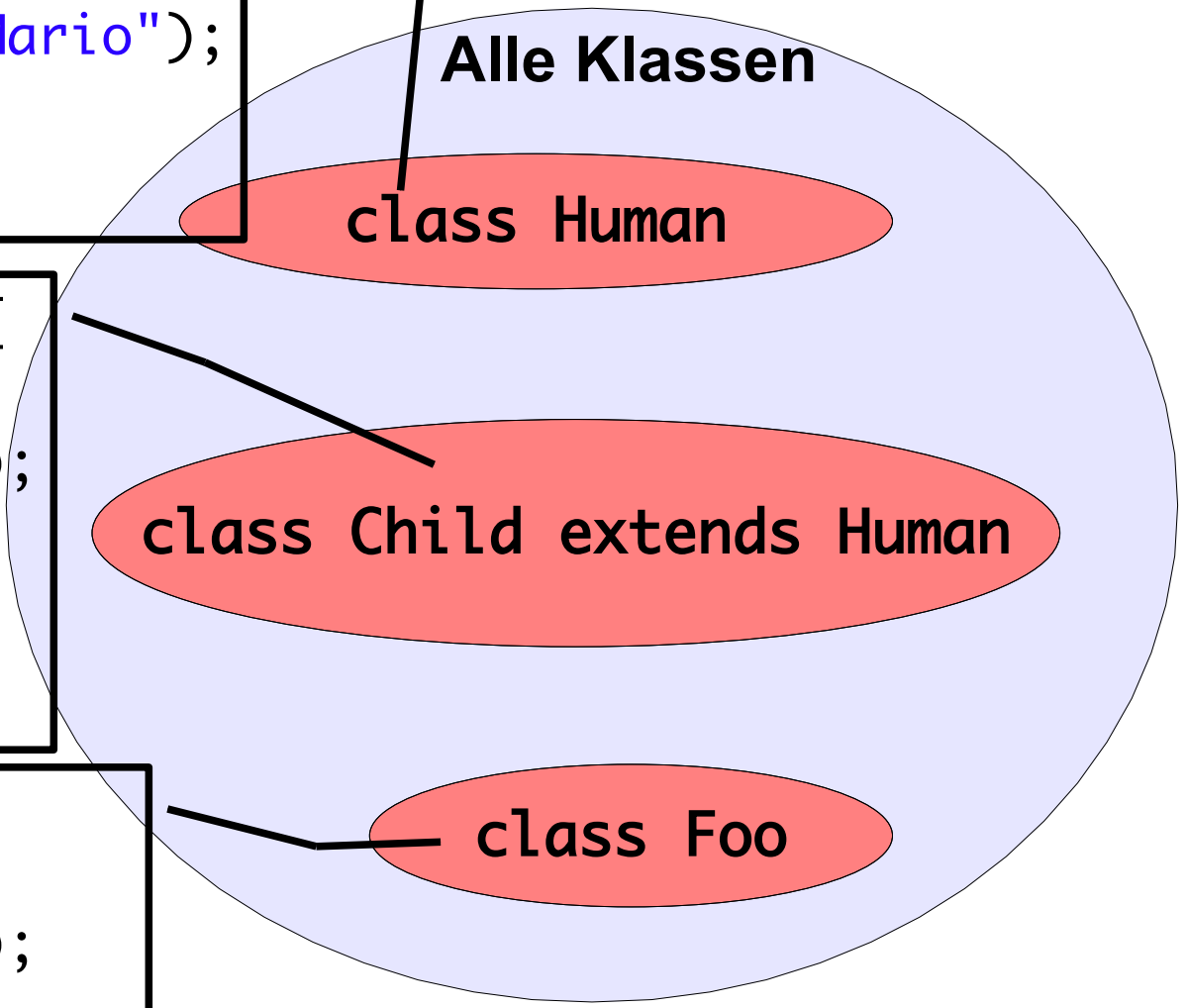
Klassen
Attribute
Methoden

Kapselung (Bsp: Public)

```java
public class Human {
    public void sayName() {
        System.out.println("Mario");
    }
}
```

```java
class Child extends Human {
    void foo() {
        Human h = new Human();
        h.sayName();
    }
}
```

```java
class Foo {
    void bar() {
        Human h = new Human();
        h.sayName();
    }
}
```

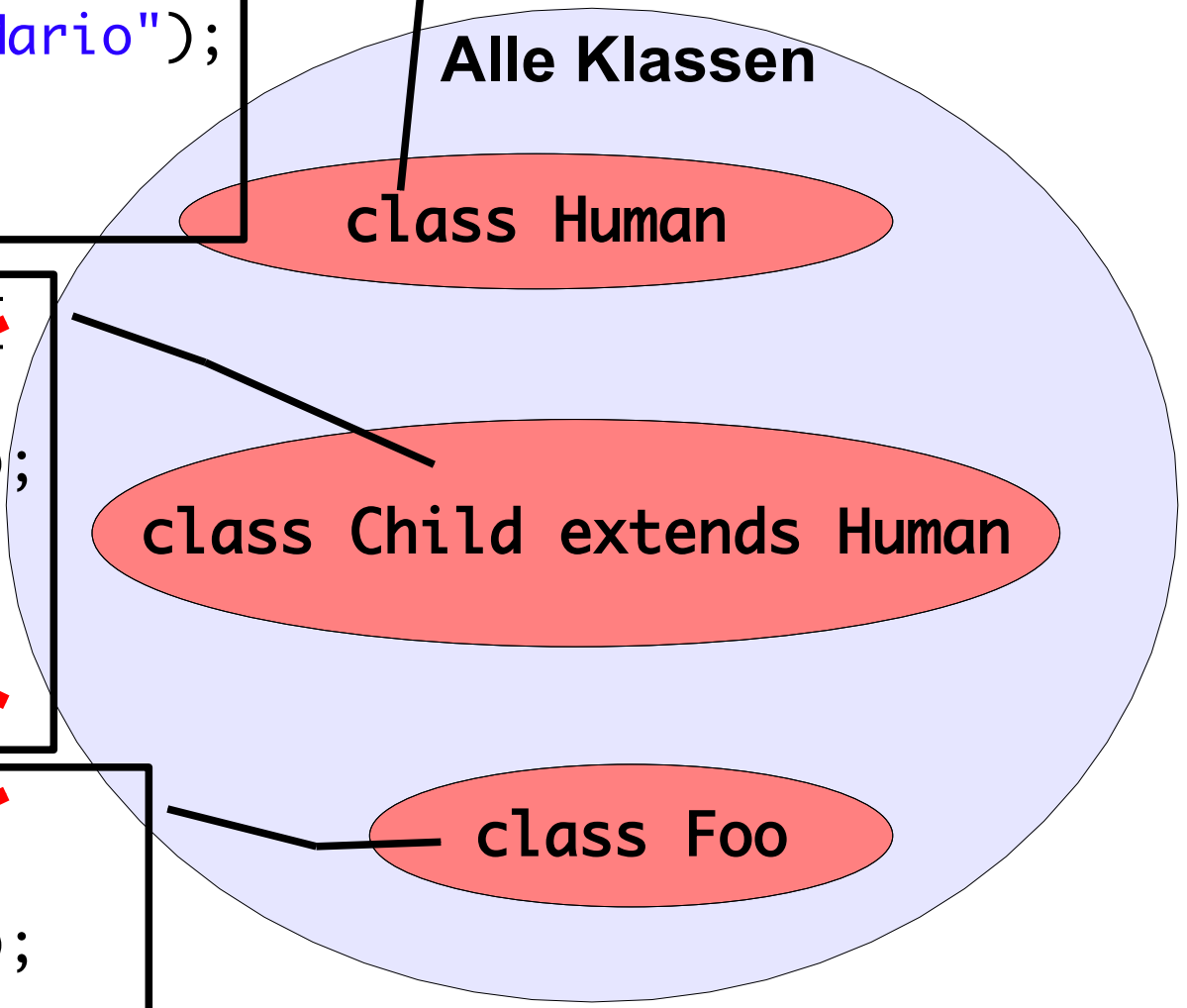


Kapselung (Bsp: Public)

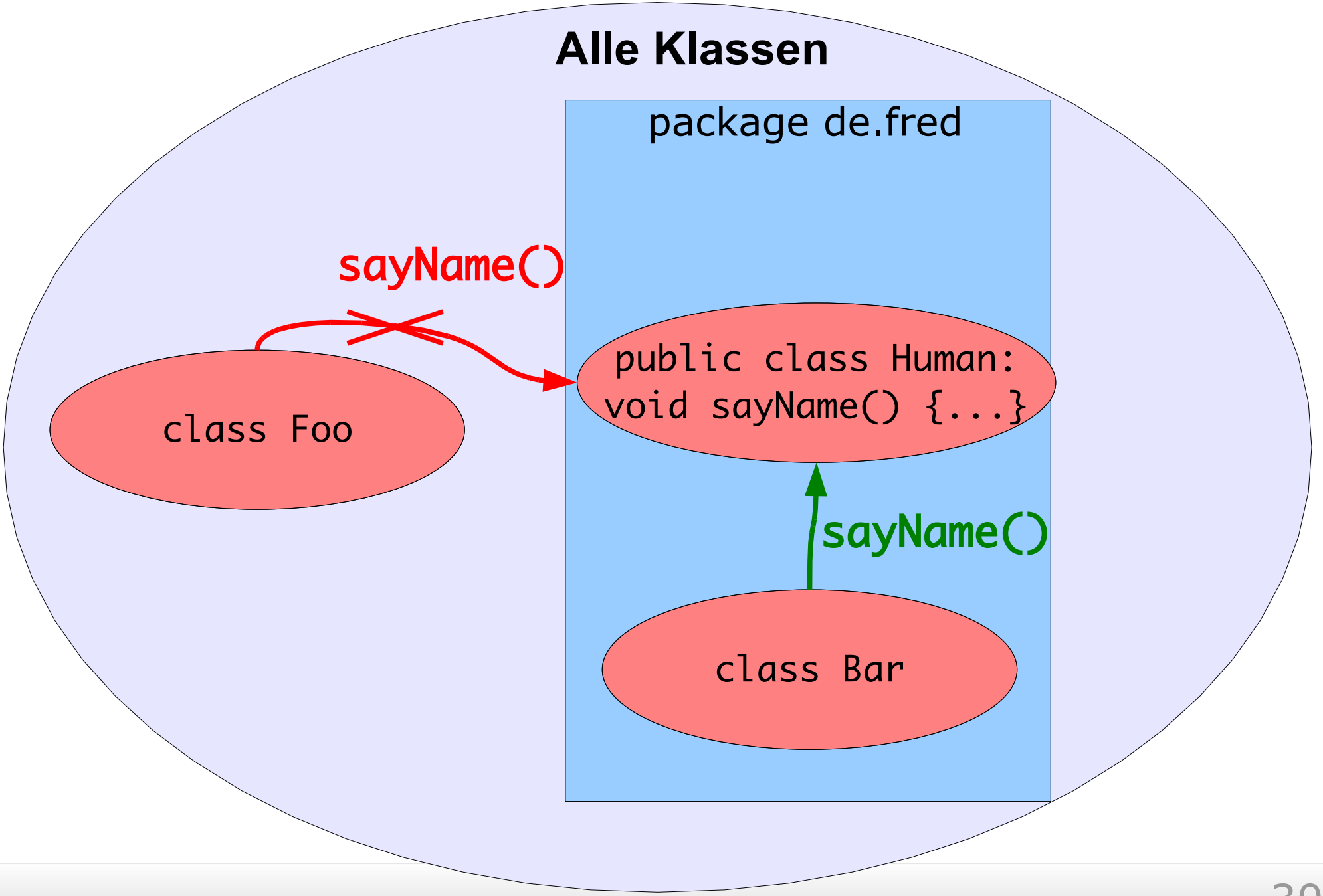
```
public class Human {
    private void sayName() {
        System.out.println("Mario");
    }
}
```

```
class Child extends Human {
    void foo() {
        Human h = new Human();
        h.sayName();
    }
}
```

```
class Foo {
    void bar() {
        Human h = new Human();
        h.sayName();
    }
}
```



Kapselung (Bsp: ohne Modifizierer)



Zusammenfassung

Chuck Norris says…

- Java Packages unterteilen den Klassen-Namensraum
- Vererbung:
  - Erben von Attributen, Methoden, Typen
  - Overriding
  - Alles erbt von java.lang.Object!
- public, private etc. schränken die Sichtbarkeit ein

Übersicht

- Wiederholung
- Java Packages
- Vererbung
- Kapselung Revisited
- **toString()**
- equals()

Ein Beispiel für System.out.println(Object obj)

```
public static void main(String[] args) {
   // create some Humans
   Human luigi = new Human("Luigi");
   Human luigi2= new Human("Luigi");

   // print the Humans
   System.out.println(luigi);
   System.out.println(luigi2);
}
```

Die Ausgabe von System.out.println(Object obj)

```
[mario bin]# java Main
Student@1888759
Student@6e1408
```

System.out.println(Object obj) - Wie funktioniert es?

- System.out.println(Object obj)
  - **Human** erbt von **Object** (Subtyping)
  - Ruft **obj.toString()** auf
  - Die Methode **Object.toString()** gibt den Klassennamen des Objektes und eine eindeutige Zahl zurück (daher *Student@1888759*)
- Können wir **toString()** verändern?
  - Ja, das Zauberwort heißt *Overriding*
  - Man überschreibt die toString()-Methode mit seiner eigenen

Die neue Klasse Version 1.0

```
public class Human {

   [ ... ]

   public String toString() {
      return "Mensch \"" + this.name + "\"";
   }
}
```

Die Ausgabe von toString() nach der Änderung

```
[mario bin]# java Main
Mensch "Luigi"
Mensch "Luigi"
```

Übersicht

- Wiederholung
- Java Packages
- Vererbung
- Kapselung Revisited
- toString()
- **equals()**

equals(Object obj) – Ein Codefragment

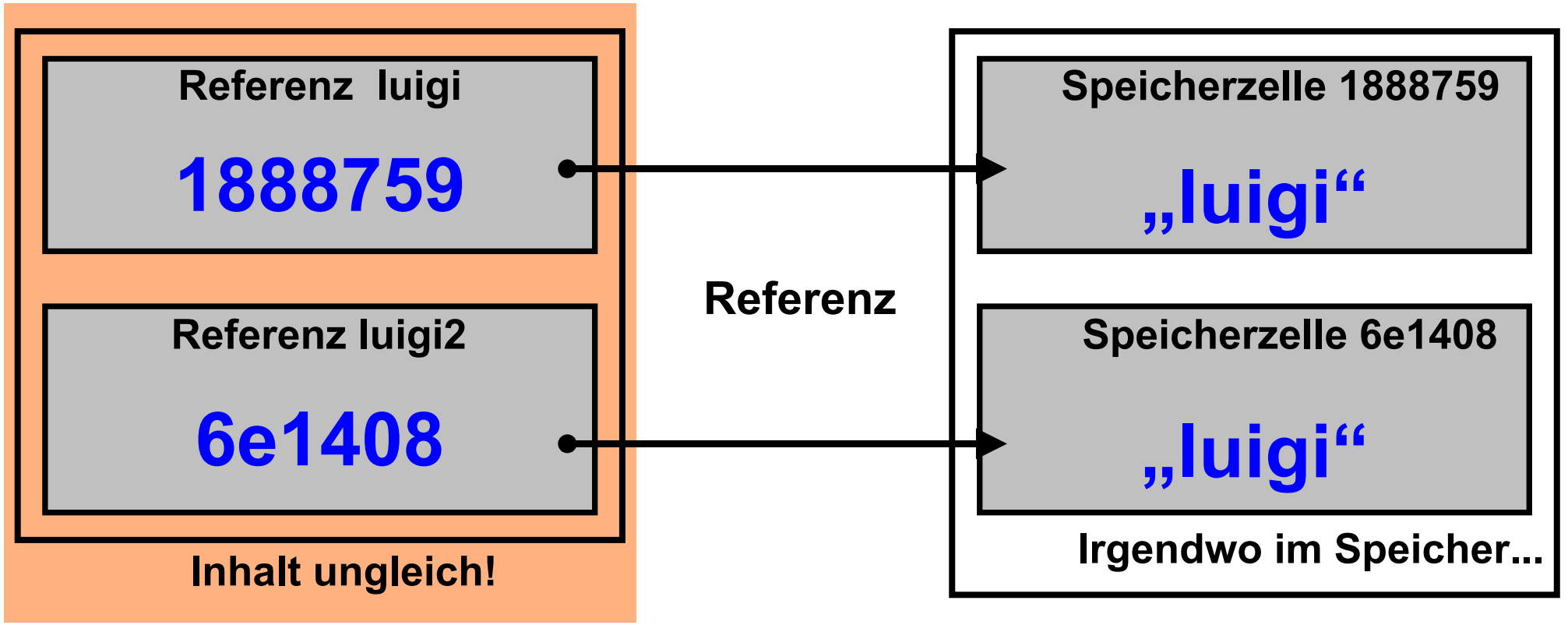
```
public static void main(String[] args) {
   // create some Humans
   Human luigi = new Human("Luigi");
   Human luigi2= new Human("Luigi");

   // compare the Humans
   System.out.println( "Gleich? " +
                luigi.equals(luigi2) );
}
```

```
[mario bin]# java Main
Gleich? false
```

Noch nicht ganz korrekt

- Aus der API
  - **equals()** vergleicht zwei Objekte
  - Aber **luigi** ist gleich zu **luigi2**!
- Das Problem
  - In **Object.equals(Object obj)** steht
    ```
    return (this == obj);
    ```
  - Und **==** vergleicht nur die Referenzen nicht, die Eigenschaften der Objekte
  - Daher sollten wir **Object.equals()** überschreiben, damit es nicht mehr **==** verwendet.

Grafik


Referenz luigi
1888759
Referenz luigi2
6e1408
Inhalt ungleich!
Referenz
Speicherzelle 1888759
„luigi“
Speicherzelle 6e1408
„luigi“
Irgendwo im Speicher...

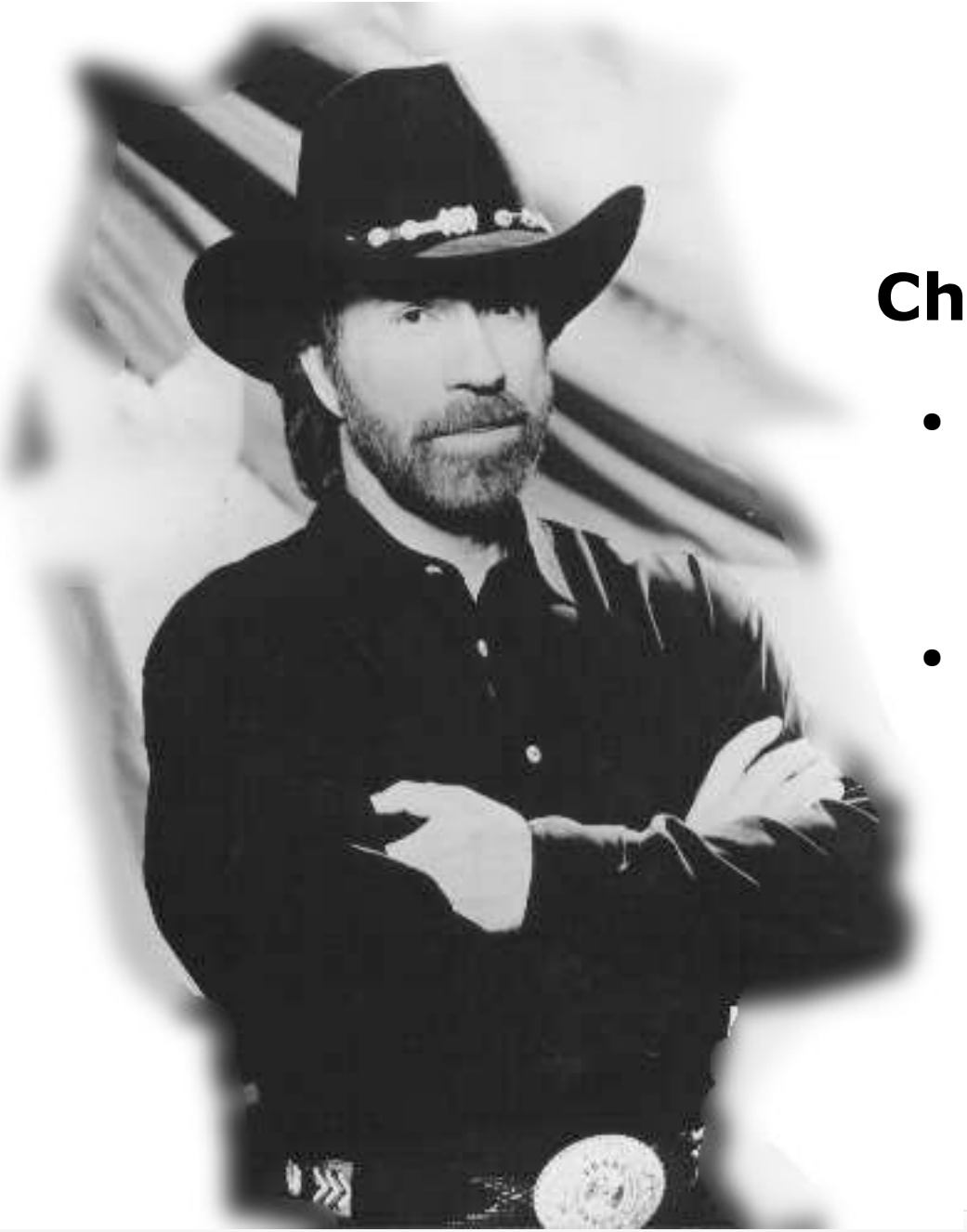
Der Code zu Human.equals(Object obj)

```
public class Human{

   [...]

   public boolean equals(Object obj) {
      if( obj == null ) {
         return false;
      }

      if( obj instanceof Human ) {
         Human other = (Human)obj;

         return other.name.equals(this.name);
      } else {
         return false;
      }
   }
}
```

Die Ausgabe von equals nach der Veränderung

```
[mario bin]# java Main
Gleich? true
```

Zusammenfassung



Chuck Norris says...

- toString() stellt Objekte als Strings dar
- equals() vergleicht Objekte auf Inhaltsgleichheit

Danke für die Aufmerksamkeit

Stichworte zur weiteren Recherche:

- static
- Exceptions
- Javadoc
- Interfaces, Abstrakte Klassen
- Dynamisches Binden
- Polymorphie
- Jar-Files
- Java Generics
- Collections, Iteratoren

Integrierte Entwicklungsumgebungen (IDEs):

- Eclipse
- Sun NetBeans
- CodeGear Jbuilder
- IntelliJ IDEA

Literaturlinks:

- Java ist auch eine Insel:

http://www.galileocomputing.de/openbook/javainsel7

- Javabuch:

http://www.javabuch.de

- Java API (Java 1.5.0):

http://java.sun.com/j2se/1.5.0/docs/api


Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
eclipse
Kurs
Coming up November 2008....
Für jeden, der...
...seine Java-Hausaufgaben in kürzerer Zeit schaffen will
...einfacher und effizienter Java-Programme erstellen möchte
...die Entwicklungsumgebung eclipse gerne kennenlernen würde

Übungen nur in folgenden Räumen heute:

- MA 241
- FR 2516
- FR 2517
- FR 5535
- FR 5538
- FR 5539

Folgende Räume sind geschlossen:

- FR 5083
- FR 5087
- FR 6514