

Speicherung von Objekten I

Zentrale Konzepte:

- ◉ Serialisierung/Deserialisierung
- ◉ Datenströme
- ◉ Versionsnummer

Möglichkeiten der Speicherung eines Objektzustands

- ein Objekt lässt sich **serialisieren** und in eine Datei schreiben.

```
-!srSpielfigur#%0+,ÖRÄIstÄrkeLtyptL
java/lang/String;[waffentLjava/lang/
String;xp2tElbur[Ljava.lang.String;ÖVçé
{GxptBogentSchwerttStaubsq~ÈtTrolluq~
tbloÄYe HÄndetgroÄYe Axtsq~xtZaubere
ruq~tZaubersprÄ]chetUnsichtbarkeit
```

- der Objektzustand kann in Textform in eine Datei geschrieben werden.

```
50,Elb,Bogen,Schwert,Staub
200,Troll,bloße Hände,große Axt
120,Zauberer,Zaubersprüche,Unsichtbarkeit
```

- Der Objektzustand kann in einer Datenbank gespeichert werden.

Stellen Sie sich vor, Sie müssen drei Spielfiguren speichern ...

Spielfigur
int stärke String typ Waffe[] waffen
getWaffe() benutzeWaffe() erhöheStärke() // Weiteres

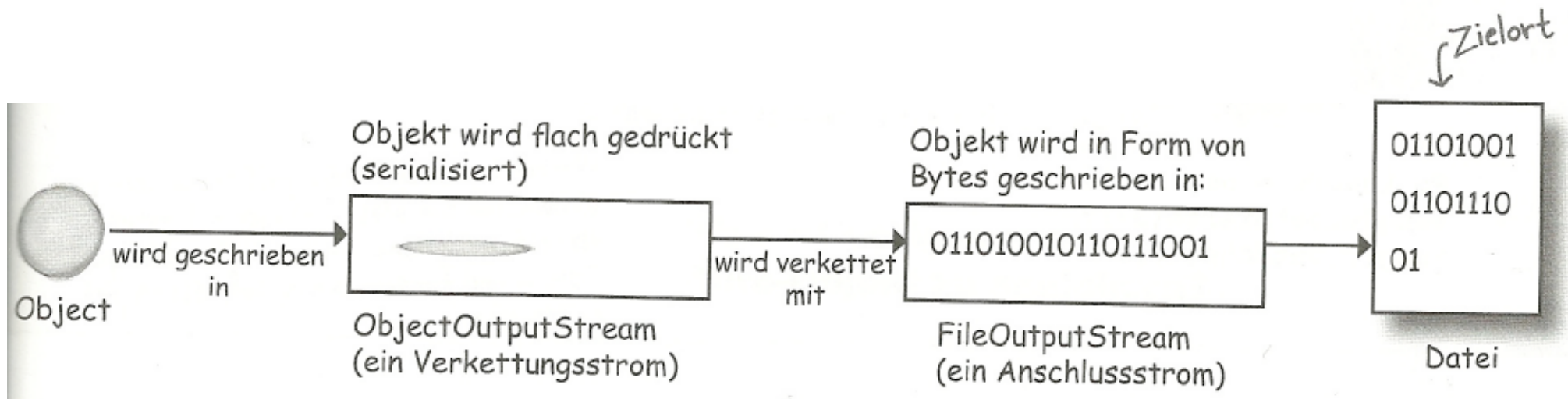
Stärke: 50
Typ: Elb
Waffen: Bogen,
Schwert, Staub
Objekt

Stärke: 200
Typ: Troll
Waffen: bloße
Hände, große Axt
Objekt

Stärke: 120
Typ: Zauberer
Waffen: Zaubersprüche,
Unsichtbarkeit
Objekt

Konzept der Verkettung von Strömen

- Ein **Anschlusstrom (connection stream)** repräsentiert eine Verbindung zu einer Quelle oder einem Zielort. Er weiss, wie man die Quelle bzw. den Zielort öffnet, Informationen aus der Quelle liest bzw. in den Zielort schreibt und wie man Quelle bzw. Zielort wieder schließt.
- Ein Anschlusstrom muss mit einem sog. **Verkettungsstrom (chain stream)** verbunden werden, um etwas aus dem Anschlusstrom lesen oder in den Anschlusstrom schreiben zu können.
- Ein Verkettungsstrom kann selber keine Verbindung zu einer Quelle oder einem Ziel herstellen, ist aber in der Lage z.B. ein Objekt aus einem Anschlusstrom einzulesen bzw. in einen Anschlusstrom zu schreiben.



Ein serialisiertes Objekt in eine Datei schreiben

- ① Einen *FileOutputStream* Objekt erzeugen.

```
FileOutputStream fileStream = new FileOutputStream("Spiel.ser");
```

FileOutputStream weiss, wie man sich mit einer Datei verbindet und eine erzeugt.

- ② Einen *ObjectOutputStream* Objekt erzeugen.

```
ObjectOutputStream os = new ObjectOutputStream(fileStream);
```

ObjectOutputStream weiss, wie man Objekte schreibt.

- ③ Das zu serialisierende Objekt schreiben.

```
os.writeObject(figur1);
```

```
os.writeObject(figur2);
```

```
os.writeObject(figur3);
```

Die von figur1, figur2, figur3 referenzierten Objekte werden in die Datei „Spiel.ser“ geschrieben.

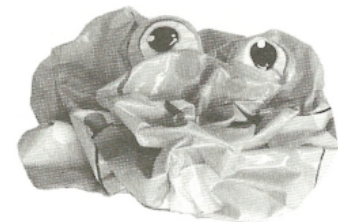
- ④ Den *ObjectOutputStream* schließen.

```
os.close();
```

*Durch Schließen des obersten Stroms werden auch die darunter liegenden Ströme geschlossen. D.h. der *FileOutputStream* und die Datei werden geschlossen.*

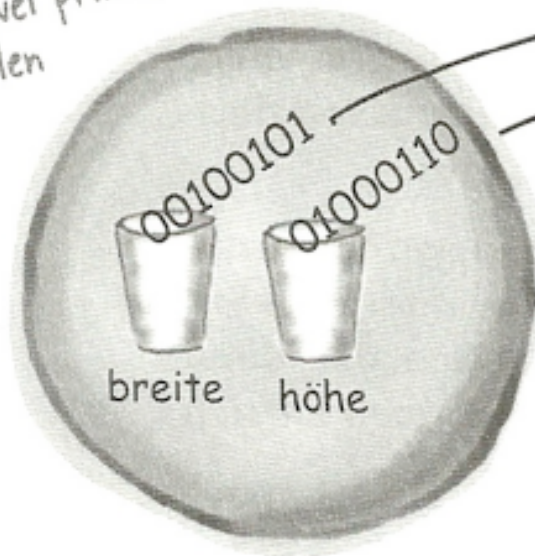
Was passiert mit dem Objekt bei der Serialisierung?

- Ein Objekt lebt auf dem *Garbage Collectible Heap*. Sein Zustand entspricht der Belegung seiner Attribute mit konkreten Werten.
- Wenn ein Objekt serialisiert wird, so werden lediglich seine Attributwerte und der Typ des Objekts gespeichert.
- Aus den gespeicherten Informationen lässt sich ein identisches Objekt rekonstruieren, wenn es wieder eingelesen wird.

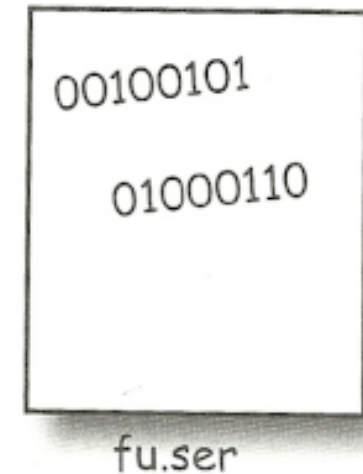


Was passiert mit dem Objekt bei der Serialisierung?

Objekt mit zwei primitiven
Instanzvariablen



Die Werte werden he-
rausgesaugt und in den
Strom gepumpt.



```
Fu kungFu = new Fu();  
kungFu.setBreite(37);  
kungFu.setHoehe(70);
```

```
FileOutputStream fs = new FileOutputStream("Fu.ser");  
ObjectOutputStream os = new ObjectOutputStream(fs);  
os.writeObject(kungFu);
```

Interface Serializable

- Die zu serialisierende Klasse muss das Interface **Serializable** implementieren.
- *Serializable* ist ein Marker-Interface, d.h. es besitzt keine Methodendeklaration. Es dient nur dem Zweck, eine Klasse als serialisierbar zu markieren.
- Implementiert eine Oberklasse *O* das Interface *Serializable*, so sind auch alle Subklassen der Klasse *O* serialisierbar.

```
public class Spielfigur implements Serializable{
    private int staerke;
    private String typ;
    private String[] waffen;
    public static final int MAX_WAFFEN_ANZ = 10;

    public Spielfigur(int staerke, String typ, String[] waffen) {
        this.staerke = staerke;
        this.typ = typ;
        setzeWaffen(waffen);
    }

    ...
}
```


Ein serialisiertes Objekt aus einer Datei lesen

- ① Einen *FileInputStream* erzeugen.

```
FileInputStream fileStream = new FileInputStream("Spiel.ser");
```

FileInputStream weiss, wie man sich mit einer vorhandenen Datei verbindet.

- ② Einen *ObjectInputStream* erzeugen.

```
ObjectInputStream os = new ObjectInputStream(fileStream);
```

ObjectInputStream weiss, wie man Objekte liest.

- ③ Das Objekt lesen und in den richtigen Typ casten.

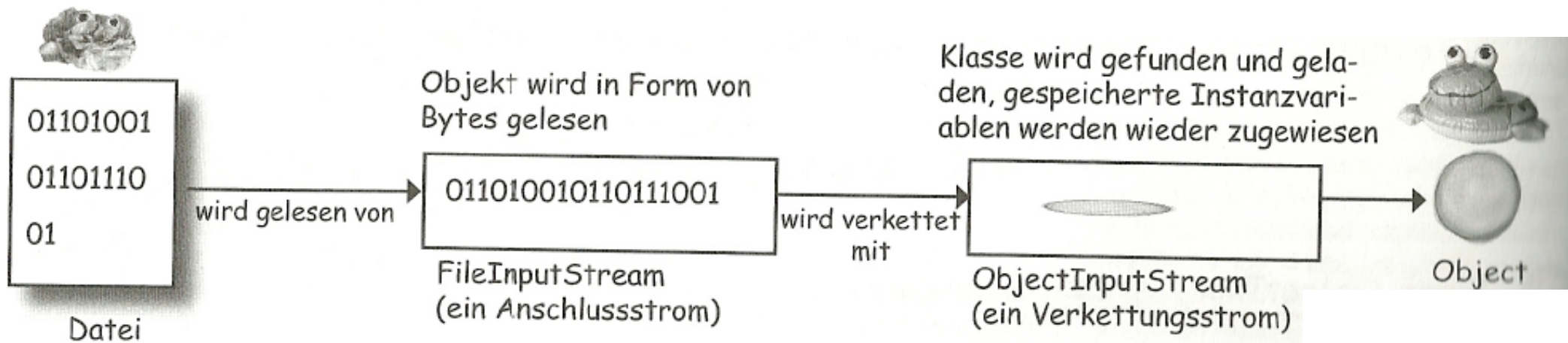
```
Spielfigur elb = (Spielfigur)os.readObject();  
Spielfigur troll = (Spielfigur)os.readObject();  
Spielfigur zauberer = (Spielfigur)os.readObject();
```

readObject() liest das jeweils nächste Objekt aus dem Strom. Dabei werden die Objekte in der gleichen Reihenfolge eingelesen, wie sie geschrieben wurden.

- ④ Den *ObjectInputStream* schließen.

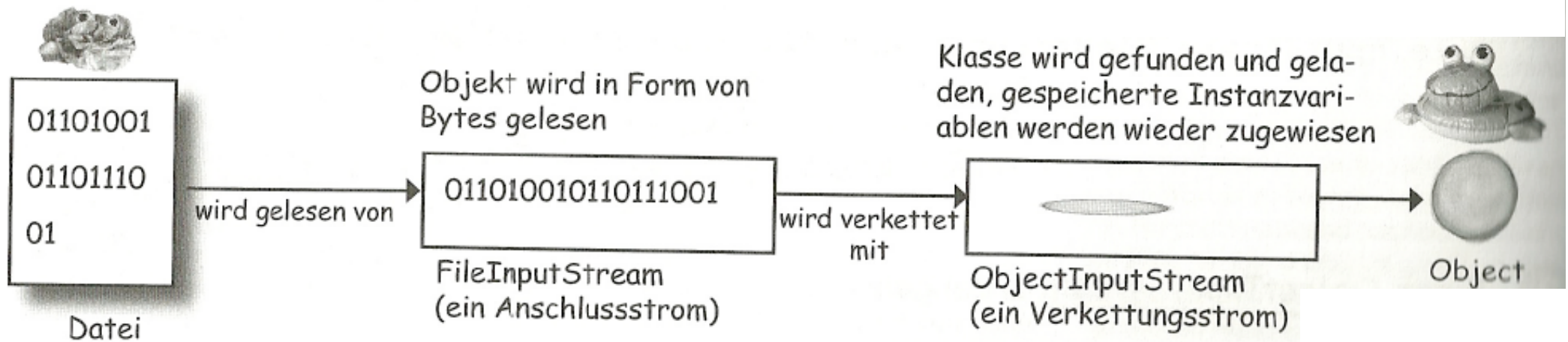
```
os.close();
```


Was passiert mit dem Objekt bei der Deserialisierung?



- ① Das Objekt wird vom Anschlussstrom aus der Datei gelesen und in den Verkettungsstrom geschoben.
- ② Die JVM bestimmt die Klasse des Objekts anhand der Informationen, die mit dem Objekt gespeichert wurden.
- ③ Die JVM versucht die Klasse des Objekts zu finden und zu laden. Wenn die Klasse nicht gefunden wird, löst die JVM eine Exception aus und die Deserialisierung schlägt fehl.

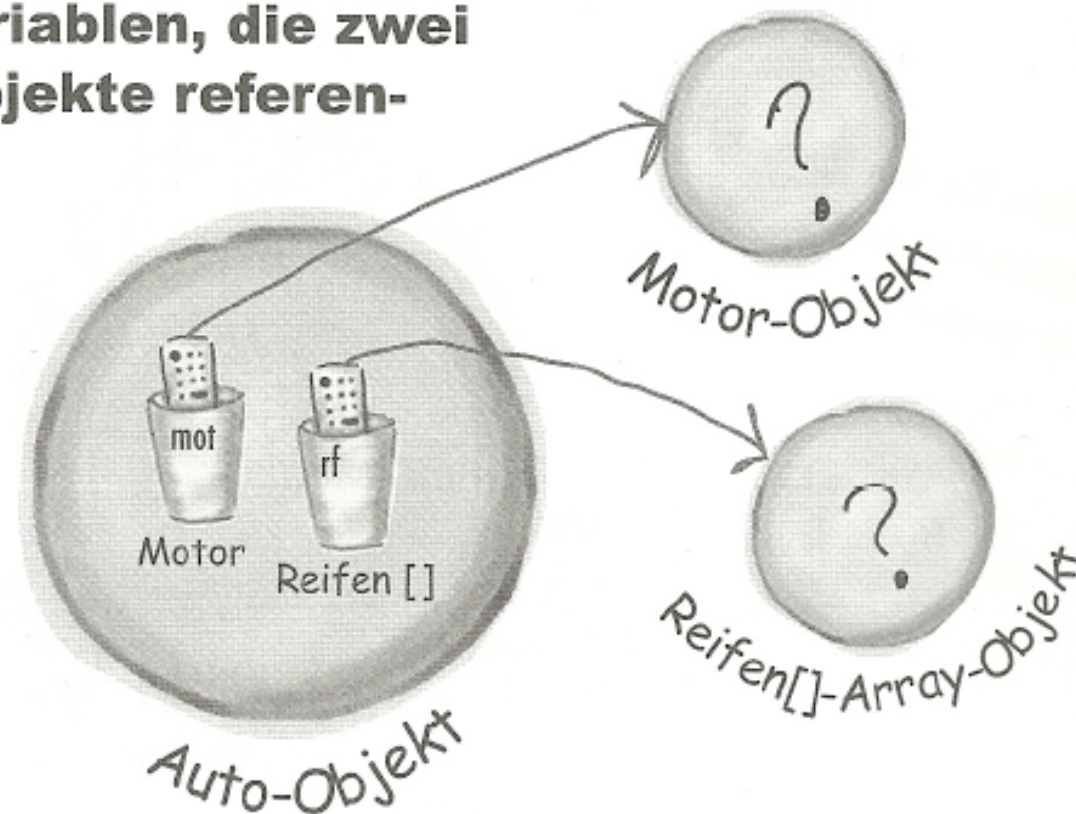
Was passiert mit dem Objekt bei der Deserialisierung?



- ④ Ein neues Objekt der gefundenen Klasse wird erzeugt. Dabei wird der Konstruktor nicht aufgerufen, da das Objekt nicht mit den Anfangswerten initialisiert werden soll.
- ⑤ Kommt irgendwo weiter oben im Vererbungsbaum eine nicht-serialisierbare Klasse vor, so wird deren Konstruktor ausgeführt und alle Konstruktoren, die über der nicht-serialisierbaren Klasse im Vererbungsbaum stehen.
- ⑥ Die Attribute des Objekt erhalten die Werte aus dem serialisierten Zustand.

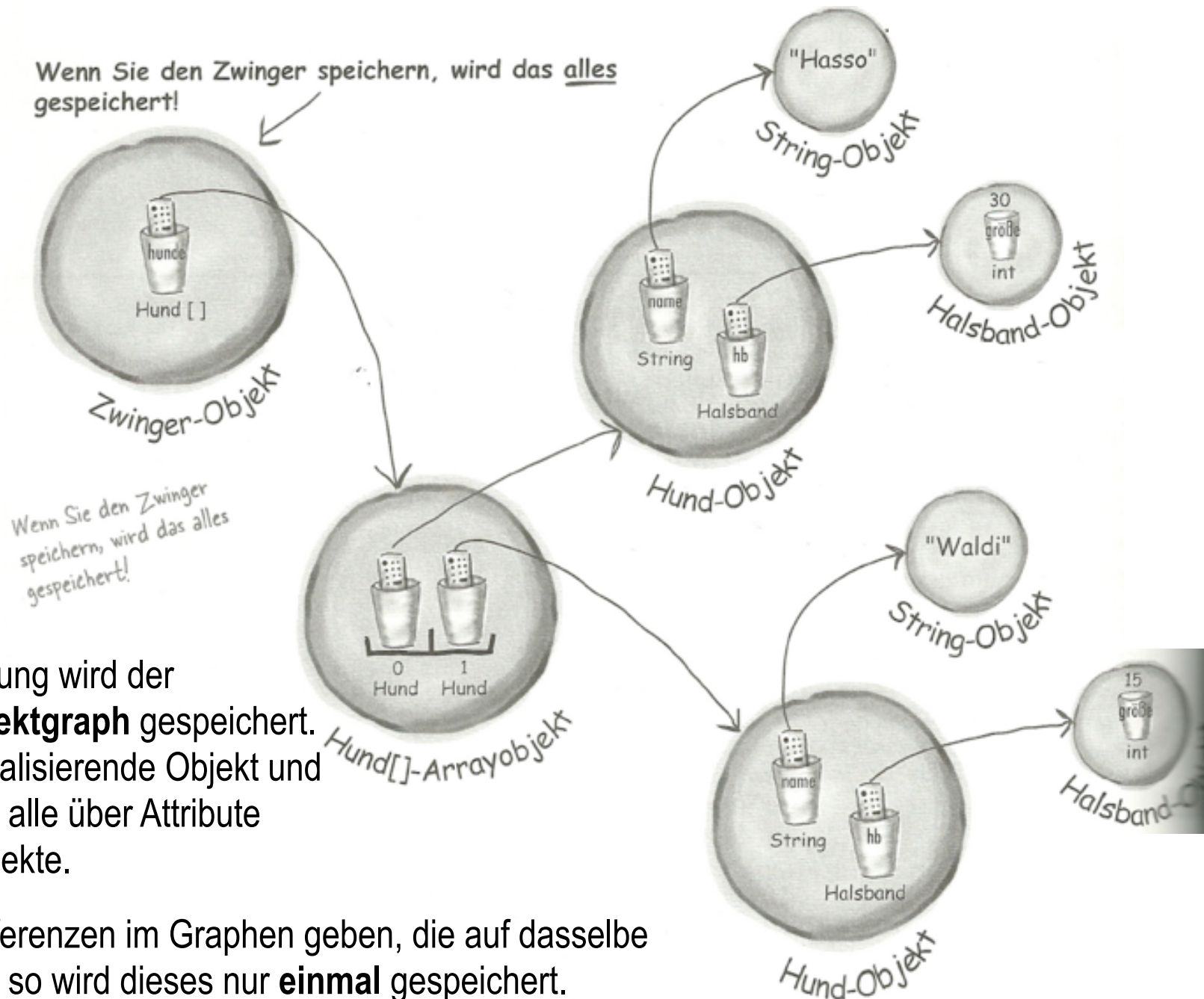
Attribute mit Objektreferenzen

Das Auto-Objekt hat zwei Instanzvariablen, die zwei andere Objekte referenzieren.



Was ist nötig, um ein Auto-Objekt zu speichern?

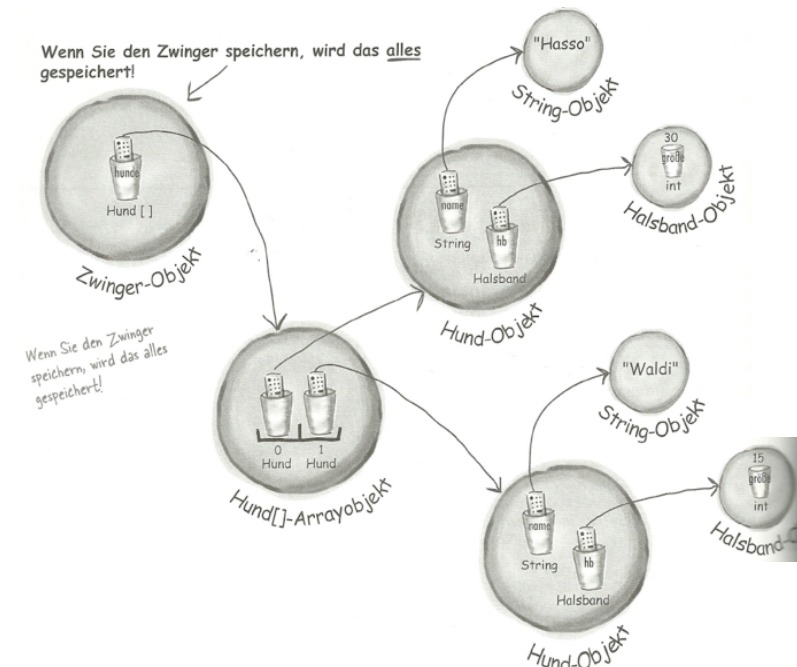
Objektgraph



- Bei der Serialisierung wird der **vollständige Objektgraph** gespeichert. Zuerst das zu serialisierende Objekt und davon ausgehend alle über Attribute referenzierten Objekte.
- Sollte es zwei Referenzen im Graphen geben, die auf dasselbe Objekt verweisen, so wird dieses nur **einmal** gespeichert.

Transiente Attribute

- Serialisierung heisst: **Alles oder nichts**.
D.h. gibt es ein Attribut, das eine Objektreferenz auf ein Objekt besitzt, dessen Klasse nicht serialisierbar ist, so schlägt die gesamte Serialisierung fehl.
- Gibt es Attribute, die nicht serialisiert werden können oder sollen, so kann man diese als **transient** deklarieren.



```
class Chat implements Serializable {
    transient String aktuelleID;
    transient String portNr;
    String benutzerName;
    ...
}
```

- Bei der Deserialisierung bekommen transiente Attribute den jeweiligen Standardwert des Typs zugewiesen (null bei Objektreferenzen, 0 bei elementaren Zahltypen, false bei boolean usw.).

Transiente Attribute: Umgang beim Einlesen des Objekts

- Wurde bei der Deserialisierung ein als transient deklariertes Objekt mit null belegt, so gibt es zwei Möglichkeiten:
 - 1.) Wenn die Werte des Attributs keine Rolle spielen, dann sollte die Klasse eine Methode anbieten, die das Attribut nach dem Deserialisieren mit geeigneten Startwerten belegt.
 - 2.) Wenn die Eigenschaftswerte des nicht gespeicherten Objekts eine Rolle spielen, dann müssen diese Werte separat gespeichert werden (z.B. auch durch Serialisierung). Die Klasse sollte eine Methode anbieten, die nach der Deserialisierung aus diesen Werten ein neues Objekt erzeugen und dem Attribut zuweisen kann.

Klassenattribute

- Die Werte von Klassenattributen werden bei der Serialisierung nicht gespeichert, weil diese für alle Objekte gleich sind und in der Klasse gespeichert werden.

Fallstrick Versionsnummer

- Was passiert eigentlich, wenn sich die Klasse eines serialisierten Objekts ändert und man versucht ein vorher serialisiertes Objekt zu laden?

```
java.io.InvalidClassException: Spielfigur; local class  
incompatible: stream classdesc serialVersionUID =  
411574669253933539, local class serialVersionUID =  
4307505358672231038
```

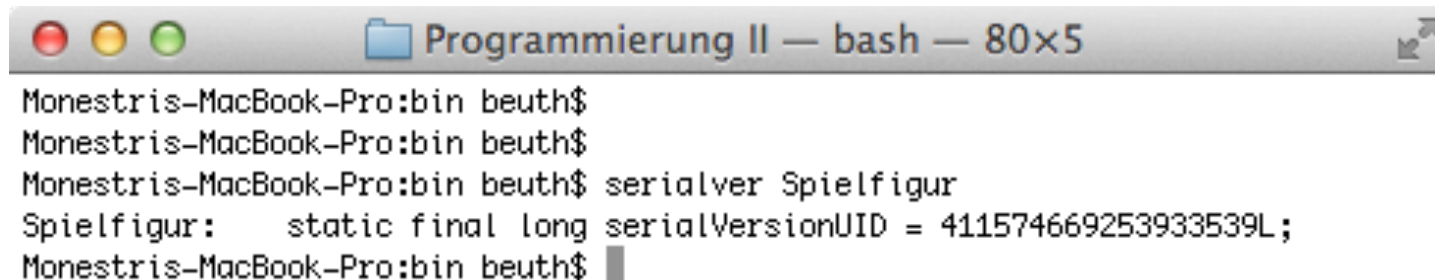
serialVersionUID

- Wenn ein Objekt serialisiert wird, dann wird das Objekt mit einer Versionsnummer für die Klasse des Objekts versehen. Diese Nummer wird als *serialVersionUID* (**u**nique **i**dentifier) bezeichnet.
- Die *serialVersionUID* wird aus der Klassenstruktur berechnet. D.h. ändert man die Klasse, so ändert sich die *serialVersionUID*.
- Passt die *serialVersionUID* des eingelesenen Objekts nicht zu der *serialVersionUID* der Klassenbeschreibung, so schlägt die Deserialisierung fehl.

Setzen der Versionsnummer

Falls man davon ausgeht, dass eine serialisierbare Klasse sich weiter entwickeln könnte, man die vorher geladenen Objekte aber trotzdem noch einlesen möchte, so kann man der Klasse selber eine *serialVersionUID* geben.

- ① Verwenden Sie das `serialver`-Kommandozeilen-Werkzeug, um die VersionsID für Ihre Klasse zu bekommen.

A screenshot of a macOS terminal window titled "Programmierung II — bash — 80x5". The terminal shows the following commands and output:

```
Monestris-MacBook-Pro:bin beuth$  
Monestris-MacBook-Pro:bin beuth$  
Monestris-MacBook-Pro:bin beuth$ serialver Spielfigur  
Spielfigur:  static final long serialVersionUID = 411574669253933539L;  
Monestris-MacBook-Pro:bin beuth$
```

- ② Setzen Sie die *serialVersionUID* als Klassenkonstante in Ihrer Klasse.

```
public class Spielfigur implements Serializable{  
    static final long serialVersionUID = 411574669253933539L;  
    private int staerke;  
    private String typ;  
    private String[] waffen;  
    ...  
}
```

Setzen der Versionsnummer: Sie haben die Verantwortung!

- ◉ Wenn man die Versionsnummer als Konstante der Klasse gesetzt hat, dann lässt sich ein Objekt auch dann deserialisieren, wenn die Klasse inzwischen geändert wurde.
- ◉ Hierbei werden dann alle nicht bekannten Attribute des Objekts mit Standardwerten belegt.
- ◉ Sie tragen also die Verantwortung dafür, dass ein älteres Objekt, das mit einer anderen Version der Klasse erstellt wurde, auch dann noch korrekt funktioniert, wenn das Objekt mit einer Klasse in einer neueren Version wieder eingeladen wird.