Objektserialisierung

Serialisierung von Objekten

Umwandlung des Objektzustandes in einen Strom von Bytes, aus dem eine Kopie des Objekts zurückgelesen werden kann.

Serialisierung in Java

- einfacher Mechanismus zur Serialisierung von Objekten
- eigenes Datenformat

Anwendungen

- Abspeicherung von internen Programmzuständen
- Übertragung von Objekten zwischen verschiedenen JVMs, z.B. über ein Netzwerk

Aber: Zum Austausch von Daten mit anderen Programmen sind Standardformate (z.B. XML) oder speziell definierte Datenformate besser geeignet.

Serialisierung in Java

Java stellt einen Standardmechanismus zur Serialisierung von Objekten zu Verfügung.

Ablauf der Serialsierung eines Java-Objekts:

1. Metadaten, wie Klassenname und Versionsnummer, in den Byte-Strom schreiben
2. alle nichtstatischen Attribute (`private`, `protected`, `public`) serialisieren
3. die entstehenden Byte-Ströme in einem bestimmten Format zu einem zusammenfassen

Bei der Deserialisierung wird der Datenstrom eingelesen und ein Java Objekt mit dem gespeicherten Zustand erzeugt.

Serialisierung in Java

Kennzeichnung von serialisierbaren Objekten: Klasse implementiert das (leere) Interface `java.io.Serializable`.

Attribute einer serialisierbaren Klasse sollten Basistypen oder serialisierbare Objekte sein (sonst Laufzeitfehler bei Serialisierung).

Gründe für Kennzeichnungspflicht:

- Sicherheit, z.B. Zugriffsbeschränkung auf `private`-Attribute
- Serialisierbarkeit soll aktiv vom Programmierer erlaubt werden

Serialisierung von Objekten

Objekte schreiben

```
FileOutputStream f = new FileOutputStream("datei");
ObjectOutput s = new ObjectOutputStream(f);
s.writeObject(new Integer(3));
s.writeObject("Text");
s.flush();
```

Objekte lesen

```
FileInputStream in = new FileInputStream("datei");
ObjectInputStream s = new ObjectInputStream(in);
Integer int = (Integer)s.readObject();
String str = (String)s.readObject();
```

Serialisierung von Objekten

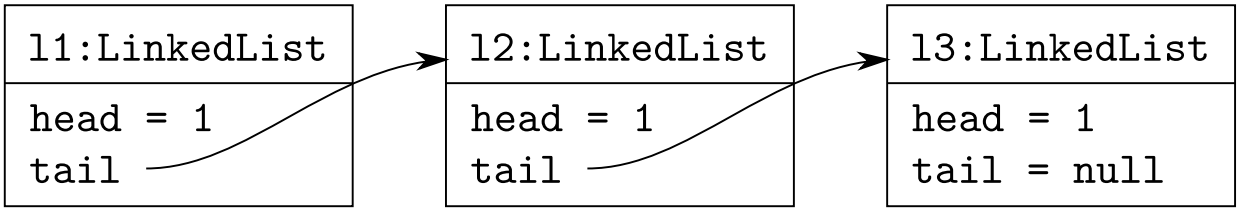
Binär kodierte Objekte in „`datei`“

```
aced0005737200116a6176612e6c616e  |....sr..java.lan|
672e496e74656765721 2e2a0a4f78187  |g.Integer.......|
380200014900057616c756578720010  |8...I..valuexr..|
6a6176612e6c616e672e4e756d626572  |java.lang.Number|
86ac951d0b94e08b0200007870000000  |..........xp....|
037400045465874                   |.t..Text|
```

Serialisierung – Beispiel

```
class LinkedList implements Serializable {
  private int head;
  private LinkedList tail;
  //...
}
```

Attribute (`head` und `tail`) sind serialisierbar.



Durch `s.writeObject(l1)` wird die gesamte Liste seralisiert, also auch `l2` und `l3`.

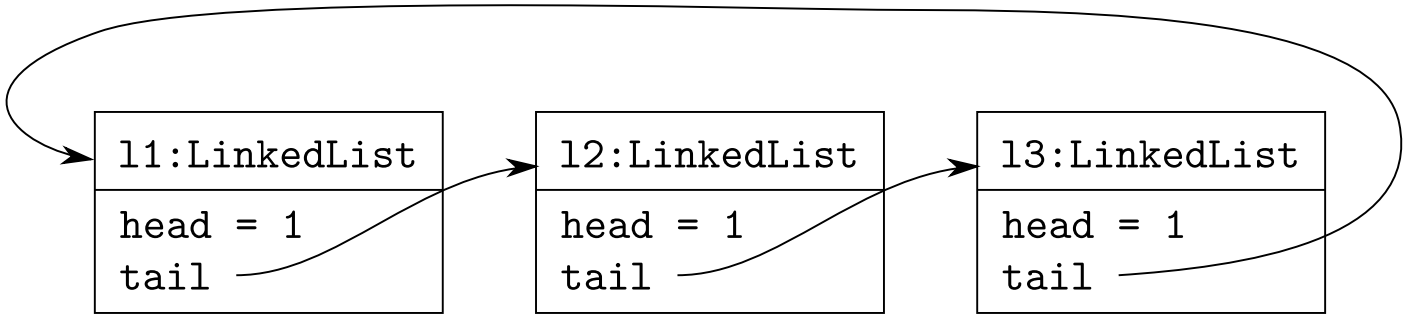
Serialisierung – Beispiel

Serialisierung speichert den Objektgraphen.

Kommt ein Objekt bei der Serialisierung eines anderen Objekts mehrmals vor, so wird nur eine Kopie gespeichert.

Durch Serialisierung und Deserialisierung erhält man eine Kopie der Datenstruktur mit dem gleichen Objektgraphen. Identitätsgleichheit (==) von Objekten im Objektgraph bleibt erhalten.

Beispiel Rückwärtszeiger in einer verketteten Liste werden richtig wiederhergestellt.



Transient

Attribute, die nicht serialisiert werden sollen, können als `transient` markiert werden.

Beipiele

- rekonstruierbare Daten, z.B. Caches oder andere aus Effizienzgründen gespeicherte Daten
- nichtserialisierbare Felder, z.B. Threads
- sensitive Daten

```
public class Account {
  private String username;
  private transient String password;
  //...
}
```

N.B.: sensitive Daten wie Passwörter sollten ohnehin möglichst nicht im Hauptspeicher gehalten werden

Anpassen der Serialisierungsprozedur

Serialisierungsmethoden können angepasst werden.

- Schreiben von zusätzlichen Daten, z.B. Datum
- wiederherstellen von transienten und nichtserialisierbaren Feldern, z.B. Wiederherstellung von Caches, Neustart von Threads

Dazu müssen in der serialisierbaren Klasse zwei Methoden mit folgender Signatur geschrieben werden:

```
private void writeObject(ObjectOutputStream oos)
  throws IOException

private void readObject(ObjectInputStream ois)
    throws ClassNotFoundException, IOException
```

Anpassen der Serialisierungsprozedur – Beispiel

```
private void writeObject(ObjectOutputStream oos)
  throws IOException {
  // zuerst die Standardserialisierung aufrufen:
  oos.defaultWriteObject();
  // zusätzliche Daten schreiben
  oos.writeObject(new java.util.Date());
}

private void readObject(ObjectInputStream ois)
    throws ClassNotFoundException, IOException {
  // zuerst die Standarddeserialisierung aufrufen:
  ois.defaultReadObject();
  // zusätzliche Daten lesen:
  date = (Date)ois.readObject();
  // mit transient markierte Felder wiederherstellen
  ...
}
```

Versionsnummern

Serialisierte Objekte haben eine Versionsnummer. Objekte mit falscher Versionsnummer können nicht deserialisiert werden.

Versionsnummer kann als statisches Attribut definiert werden:

```
public static long serialVerUID = 1L
```

Ist keine Nummer angegeben, so benutzt Java einen Hashwert, der sich u.A. aus den *Namen* der Klassenattribute errechnet

- $\rightsquigarrow$ Änderungen an der Klasse (z.B. Hinzufügen einer Methode) machen serialisierte Daten inkompatibel.
- verhindert vesehentliches Einlesen inkompatibler Daten

Durch Angabe einer `serialVerUID` kann man Kompatibilität steuern, evtl. mit eigener `readObject`-Funktion.

Serialisierbarkeit und Vererbung

Alle Unterklassen einer serialisierbaren Klasse sind serialisierbar.

Kann Serialisierung verhindern mit:

```
private void readObject(ObjectInputStream stream)
    throws IOException, ClassNotFoundException {
        throw new NotSerializableException();
}
private void writeObject(ObjectOutputStream stream)
    throws IOException {
        throw new NotSerializableException();
}
```

Serialisierbarkeit und Vererbung

Ist eine Oberklasse einer serialisierbaren Klasse *nicht* serialisierbar, so werden ihre privaten Felder *nicht* serialisert.

Beim Deserialisieren wird der Konstruktor ohne Argumente der ersten nichtserialisierbaren Oberklasse aufgerufen (ein Konstruktor ohne Argumente muss existieren!).

`readObject` und `writeObject` müssen geeignet implementiert sein, damit der Zustand der Oberklasse beim Deserialisieren wiederhergestellt wird.