Verteilte Systeme - Übung

Tobias Distler, Michael Gernoth

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

www4.informatik.uni-erlangen.de

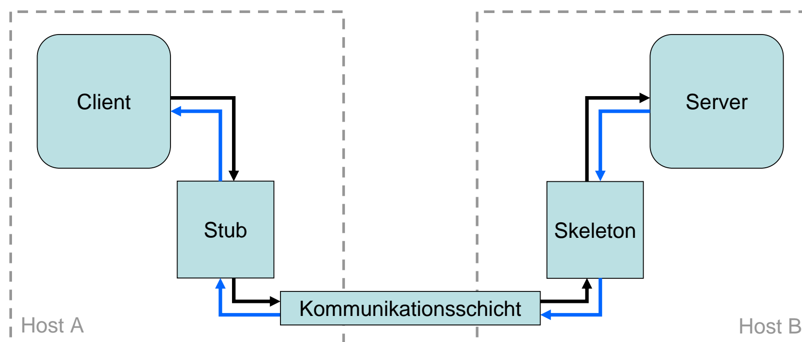
Sommersemester 2010

Überblick

Marshalling & Unmarshalling
- Serialisierung & Deserialisierung
- Serialisierung & Deserialisierung in Java
- Java Reflection API
- Übungsaufgabe 2

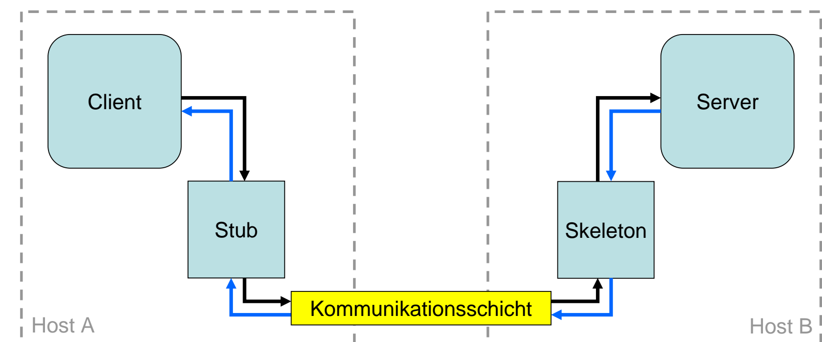
VS-Übung

- Entwicklung eines eigenen Fernaufrufsystems
- Orientierung an Java-RMI



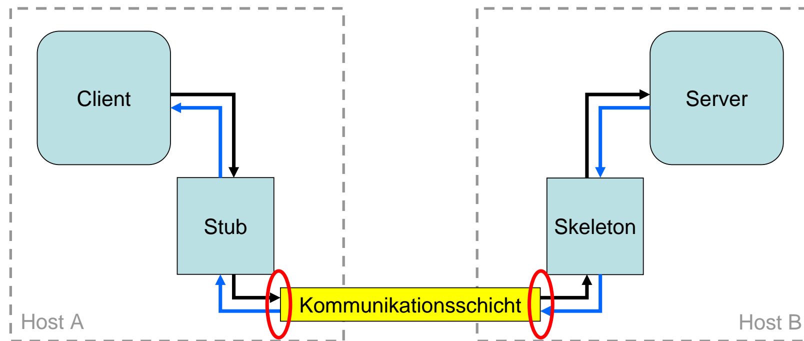
Übungsaufgabe 2

- Entwicklung eines eigenen Fernaufrufsystems
- Zunächst: Aufbau der Kommunikationsschicht



Marshalling & Unmarshalling

- Marshalling: Verpacken von Informationen in einer Nachricht
- Unmarshalling: Auspacken von Informationen aus einer Nachricht



Marshalling & Unmarshalling

Problemstellungen

- Unterschiedliche Datentypen
- Heterogenität bei der lokalen Repräsentation von Datentypen
- Unterschiedliche Parameterübergabearten

Unterschiedliche Datentypen

- Primitive Datentypen
  - z.B. `char`, `boolean`, `int`,...
- Benutzerdefinierte Datentypen
  - z.B. `structs`
- Felder
  - z.B. `int[47]`
- Referenzen
  - z.B. `char *`
- Objekte
  - z.B. Strings, Dateien,...

→ Kein allgemeines Vorgehen möglich

Heterogenität

"Byte Sex"-Problem

- Big Endian
  - Most-significant byte first
  - z.B. SPARC, Motorola
  - Network Byte Order
- Little Endian
  - Least-significant byte first
  - z.B. Intel x86

	0	1	2	3
big endian	14	a2	b5	c8
little endian	c8	b5	a2	14

0x14a2b5c8

Heterogenität

Repräsentation von Fließkommazahlen

- Allgemein
  - Vorzeichen (s)
  - Mantisse (m)
  - Exponent (e)
  - → Zahlenwert: $(-1)^s * m * 2^e$
- Variationsmöglichkeiten
  - Anzahl der Bits für m und e
  - Speicherreihenfolge von m, e und s
  - Byte-Order

Heterogenität

Lösungsvarianten

- Kanonische Repräsentation
  - Nutzung einer allgemeingültigen Form als Zwischenrepräsentation
  - z.B. IEEE-Standard
  - → Evtl. unnötige Konvertierungen (z.B. wenn Sender und Empfänger identische Repräsentation nutzen)
- "Sender makes it right"
  - Sender kennt Datenrepräsentation des Empfängers
  - Sender konvertiert Daten
  - → Multicast an heterogene Gruppe nicht möglich
- "Receiver makes it right"
  - Kennzeichnung des Datenformats
  - Empfänger konvertiert Daten
  - → Bereitstellung sämtlicher Konvertierungsroutinen notwendig (Unproblematisch für Byte-Order-Konvertierung.)

Unterschiedliche Parameterübergabearten

- Eingabeparameter
  - Teil der Anfragenachricht
  - z.B. `value` in

```
void setValue(int value);
```

- Ausgabeparameter
  - Teil der Antwortnachricht
  - z.B. `String`-Returnwert in

```
String toString();
```

- Ein-/Ausgabeparameter
  - Teil der Anfrage- und Antwortnachricht
  - z.B. `Stack` in

```
Object pop(Stack stack);
```


Unterschiedliche Parameterübergabearten

Lösung: Einsatz geeigneter Semantiken

- Eingabeparameter
  - Call-by-value
- Ausgabeparameter
  - Call-by-result
- Ein-/Ausgabeparameter
  - Call-by-value-result

→ Zeiger dereferenzieren

Marshalling & Unmarshalling

Relevanz der Problematiken in Java

- Unterschiedliche Datentypen
  - Primitive Datentypen
  - Objektreferenzen
  - → Problem relevant (siehe Übungsaufgabe 2)
- Heterogenität bei der lokalen Repräsentation von Datentypen
  - Plattformunabhängigkeit
  - → Problem für Java-Nutzer nicht relevant
- Unterschiedliche Parameterübergabearten
  - Eingabeparameter
  - Ausgabeparameter
  - Ein-/Ausgabeparameter
  - → Problem relevant (siehe Übungsaufgabe 3)

Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Serialisierung & Deserialisierung von Objekten

ObjectStream-Klassen

- `ObjectOutputStream`

```
ObjectOutputStream(OutputStream out);
void writeObject(Object obj);           // Objekt senden
[...]
```

- `ObjectInputStream`

```
ObjectInputStream(InputStream in);
Object readObject();                    // Objekt empfangen
[...]
```


Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Interfaces

- Interface `java.io.Serializable`
  - Muss von jedem Objekt implementiert werden, das über einen `Object{Out,In}putStream` ausgetauscht wird
  - Marker-Interface → keine zu implementierenden Methoden
  - → {S,Des}erialisierung wird vom Object{Out,In}putStream übernommen
- Interface `java.io.Externalizable`
  - Schnittstellenbeschreibung

```
public interface Externalizable extends Serializable {
    void writeExternal(ObjectOutput out);
    void readExternal(ObjectInput in);
}
```

  - → {S,Des}erialisierung wird vom Objekt selbst übernommen

transient-Schlüsselwort

- Einige Objekte sollen nicht serialisiert werden
  - Sicherheitsaspekte
  - Effizienzüberlegungen
- Einige Objekte können nicht serialisiert & deserialisiert werden, da sich ihr Zustand nicht so ohne weiteres wiederherstellen lässt
  - `FileInputStream`
  - `Socket`, `ServerSocket`
  - `Thread`

→ Schlüsselwort `transient`
  - Mit `transient` gekennzeichnete Attribute werden bei der {S,Des}erialisierung ignoriert

```
class TransientExample {
    transient Thread t = new Thread();
}
```


Konstanten

- `static final`
  - Klassenspezifische Konstanten
  - Müssen bei ihrer Deklaration initialisiert werden
  - Klassenweit eindeutig (→ beim Empfänger bekannt)
  - → Nicht übertragen!
- `final` (nicht `static`)
  - Objektspezifische Konstanten
  - Können auch erst im Konstruktor initialisiert werden:

```
public class FinalExample {
    public final int MAX_VALUE;

    public FinalExample(int maxValue) {
        MAX_VALUE = maxValue;
    }
}
```

  → Übertragen!

Einschub

F: Welche Informationen über ein Objekt müssen übertragen werden, um es beim Empfänger vollständig deserialisieren zu können?

A: Alle nichtrekonstruierbaren Teile des Objektzustands.

F: Welche Teile sind das?

A: Alle Werte von Attributen, die nicht „`transient`" oder „`static final`" sind.

F: Was ist mit Attributen, die selbst Objekte sind?

A: Diese Objekte müssen ebenfalls serialisiert werden.

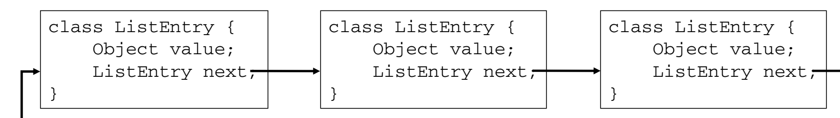
→ Alle vom Ausgangsobjekt (direkt oder indirekt) referenzierten Objekte sind in die Serialisierung einzubeziehen.

Einschub

→ Alle vom Ausgangsobjekt (direkt oder indirekt) referenzierten Objekte sind in die Serialisierung einzubeziehen.

Mögliches Problem?

- Zyklische Objektgraphen
- z.B. verkettete Listen



→ Es wird ein Abbruchkriterium benötigt.

Strom-Gedächtnis

→ Es wird ein Abbruchkriterium benötigt.

- Lösung: Gedächtnis-Mechanismus
  - Jedes zum ersten Mal gesendete bzw. empfangene Objekt wird in einer Tabelle eingetragen
  - Statt ein Objekt erneut zu versenden wird ein Verweis auf seinen Tabelleneintrag übermittelt → keine mehrmalige Serialisierung
  - Sender und Empfänger erzeugen ihre Tabellen in identischer Weise
  - Erhält der Empfänger einen Verweis auf einen Tabelleneintrag, so gibt er eine Referenz auf das bereits bestehende Objekt zurück → keine Neuinstanzierung
- Probleme
  - Tabelle wächst mit jedem neuen Objekt (keine Löschung!)
  - Änderungen an Objekten werden nicht berücksichtigt

Beispielklasse

```
public class ExampleObject {
    private String stringAttr;
    public int intAttr;

    public ExampleObject(String s, int x) {
        stringAttr = s;
        intAttr = x;
    }

    public String getStringAttr() {
        return stringAttr;
    }
}
```


Strom-Gedächtnis

Sender

```
ObjectOutputStream out = [...];
ExampleObject eo = new ExampleObject("", 47);
out.writeObject(eo);
eo.intAttr = 48;
out.writeObject(eo);
```

Empfänger

```
ObjectInputStream in = [...];
ExampleObject eo1 = (ExampleObject) in.readObject();
System.out.println(eo1.intAttr);
ExampleObject eo2 = (ExampleObject) in.readObject();
System.out.println(eo2.intAttr);
```

Ausgabe

```
47
47
```

→ Änderung an `eo` nicht beim Empfänger sichtbar!

Strom-Gedächtnis

Lösung

Bei Änderungen an bereits gesendeten Objekten:
Reset des Gedächtnis am `ObjectOutputStream`

```
void reset();
```

→ Tabelle wird auf beiden Seiten gelöscht
→ Sender und Empfänger fangen bei "Null" an

Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Java Reflection API — Überblick

- Bietet die Möglichkeit das Laufzeitverhalten von Applikationen zu analysieren und es gegebenenfalls sogar zu beeinflussen
- Paket: `java.lang.reflect`
- Ausführliches Tutorial

http://java.sun.com/docs/books/tutorial/reflect/index.html

This is a relatively advanced feature and should be used only by developers who have a strong grasp of the fundamentals of the language.

Java Reflection API — Überblick

- Ermöglicht zur Laufzeit
  - Analyse von
    - Attributen
    - Konstruktoren
    - Methoden
  - Erzeugung neuer Objekte
  - Modifikation bestehender Objekte
  - Dynamische Methodeaufrufe
  - ...
- Zentrale Klasse: `java.lang.Class`
  - Pro Objekttyp existiert ein unveränderliches `Class`-Objekt
  - Beispiel

```
String x = "x";
String y = "y";
boolean b = (x.getClass() == y.getClass()); // -> true
```


Erzeugung von `Class`-Objekten

- Allgemein
  - Funktioniert auch für primitive Datentypen

```
Class c = <Klassenname>.class;
```

- Nur bei Objektreferenzen
  - Dynamisch

```
Class c = <Objekt>.getClass();
```

  - Statisch

```
Class c = Class.forName(<Klassenname>);
```

Erzeugung von `Class`-Objekten

Navigation durch Klassenhierarchien

- Oberklasse

```
Class c = <Class-Objekt>.getSuperclass();
```

- Memberklassen/-schnittstellen/-enumerations

```
Class[] c = <Class-Objekt>.getClasses();
```

- Implementierte Schnittstellen

```
Class[] c = <Class-Objekt>.getInterfaces();
```

- Geschachtelte Klassen

```
Class c = <Class-Objekt>.getEnclosingClass();
```


Zentrale Methoden `java.lang.Class<T>`

- Information

```
boolean isPrimitive();
boolean isArray();
boolean isEnum();
Package getPackage();
```

- Cast

```
T cast(Object object);
```

- Typvergleich

```
boolean isInstance(Object object);
boolean isAssignableFrom(Class<?> cls);
```


Analyse einer Klasse

- Attribute

```
Field getField(String name);
Field getDeclaredField(String name);
Field[] getFields();
Field[] getDeclaredFields();
```

- Konstruktoren

```
Constructor getConstructor(Class<?>... paramTypes);
Constructor getDeclaredConstructor(Class<?>... pTs);
Constructor[] getConstructors();
Constructor[] getDeclaredConstructors();
```

- Methoden

```
Method getMethod(String name, Class<?>... pTs);
Method getDeclaredMethod(String n, Class<?>... pTs);
Method[] getMethods();
Method[] getDeclaredMethods();
```


Analyse einer Klasse

Allgemeine Semantik

- Zugriff auf `public`-Elemente einer Klasse

```
[...] get{Field,Constructor,Method}([...])
[...] get{Field,Constructor,Method}s()
```

- Zugriff auf alle Elemente einer Klasse

```
[...] getDeclared{Field,Constructor,Method}([...])
[...] getDeclared{Field,Constructor,Method}s()
```

Analyse von Attributen `java.lang.reflect.Field`

Wert auslesen

- Allgemein

  `Object get(Object object);`

- Primitive Datentypen

  `<Datentyp> get<Datentyp>(Object object);`

  z.B. `getBoolean()`, `getCharacter()`, `getInt()`,...

Marshalling & Unmarshalling | Serialisierung & Deserialisierung / Serialisierung & Deserialisierung in Java / Java Reflection API / Übungsaufgabe 2

Analyse von Methoden `java.lang.reflect.Method`

- Methodenname bestimmen

  `String getName();`

- Parameter bestimmen

  `Class<?>[] getParameterTypes();`

- Rückgabewert bestimmen

  `Class<?> getReturnType();`

- Exceptions bestimmen

  `Class<?>[] getExceptionTypes();`

Marshalling & Unmarshalling | Serialisierung & Deserialisierung / Serialisierung & Deserialisierung in Java / Java Reflection API / Übungsaufgabe 2

Analyse von Konstruktoren `java.lang.reflect.Constructor<T>`

- Konstruktorname bestimmen

  `String getName();`

- Parameter bestimmen

  `Class<?>[] getParameterTypes();`

- Exceptions bestimmen

  `Class<?>[] getExceptionTypes();`

Marshalling & Unmarshalling | Serialisierung & Deserialisierung / Serialisierung & Deserialisierung in Java / Java Reflection API / Übungsaufgabe 2

Modifiers `java.lang.reflect.Modifier`

- Anwendbar für
  - Klassen
  - Attribute
  - Methoden
  - Konstruktoren
- Auslesen

  `int m = <Class/Field/Method/Constructor-Objekt>.getModifiers();`

- Analyse mittels statischer Methoden der Klasse `Modifier`

  `if(Modifier.isPublic(m)) {...}`

  - Analog: `isFinal()`, `isStatic()`, `isTransient()`,...

Objekterzeugung/-modifikation/-nutzung

Die Java Reflection API bietet Möglichkeiten

- Objekte zu erzeugen
- Attribute zu modifizieren
- Methoden aufzurufen

Objekterzeugung

- Neue Objektinstanz (`java.lang.reflect.Constructor<T>`)

```
T newInstance(Object... initargs);
```

- Beispiel
  ohne Reflections

```
public static void main(String[] args) {
    ExampleObject eo = new ExampleObject("Hello", 47);
    [...]
}
```

mit Reflections

```
public static void main(String[] args) throws Exception {
    Class[] pTs = new Class[] { String.class, int.class };
    Constructor<ExampleObject> c =
                    ExampleObject.class.getConstructor(pTs);
    ExampleObject eo = c.newInstance("Hello", 47);
    [...]
}
```


Objekterzeugung

Spezialfall: Arrays

- Eigene Reflection-Klasse (`java.lang.reflect.Array`)
- Statische Methoden zur Array-Erzeugung
  - Eindimensional

```
Object newInstance(Class<?> componentType, int length)
```

  - Mehrdimensional

```
Object newInstance(Class<?> componentType, int[] dimensions)
```

- Beispiel

```
String[] stringArray;
stringArray = (String[]) Array.newInstance(String.class,
                                            10);
```


Attributmodifikation

- Attribut auf neuen Wert setzen (`java.lang.reflect.Field`)
  - Allgemein

```
void set(Object object, Object value);
```

  - Primitive Datentypen

```
void set<Datentyp>(Object object, <Datentyp> value);
```

  z.B. `setBoolean()`, `setCharacter()`, `setInt()`,...

- Beispiel

```
Field if = eo.getClass().getField("intAttr");
if.setInt(eo, 48);
```

Attributmodifikation

- Problem

```
Field sf = eo.getClass().getDeclaredField("stringAttr");
sf.set(eo, "Hello, again");
```

→ IllegalAccessException

- Grund: `stringAttr` ist `private`
- Lösung

```
void setAccessible(boolean flag);
```

- Beispiel

```
Field sf = eo.getClass().getDeclaredField("stringAttr");
sf.setAccessible(true);
sf.set(eo, "Hello, again");
sf.setAccessible(false);
```


Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Methodenaufruf

- Aufruf (`java.lang.reflect.Method`)

```
Object invoke(Object obj, Object... args);
```

- Beispiel

```
public String getStringAttr() {}
```

```
Method m = eo.getClass().getMethod("getStringAttr",
                                   new Class[0]);
String s = (String) m.invoke(eo, (Object[]) null);
```

- Näheres in der Tafelübung zu Aufgabe 3...

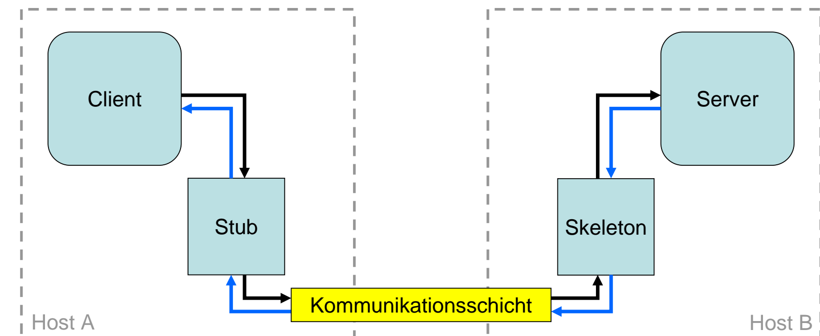
Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Marshalling & Unmarshalling
Serialisierung & Deserialisierung
Serialisierung & Deserialisierung in Java
Java Reflection API
Übungsaufgabe 2

Übungsaufgabe 2: Überblick

- Ziel: Senden einer Nachricht (Java-Objekt) über eine Socketverbindung
- Methode: Verwendung eines Objekt-Stroms



Übungsaufgabe 2: Aufgabenstellung

Implementierung eines Objekt-Stroms

- Serialisierung

  `ObjectOutputStream` ⟶ `VSObjectOutputStream`

```
VSObjectOutputStream(OutputStream out);
OutputStream getOutputStream();
void writeObject(Object obj);
void reset();
void close();
```

- Deserialisierung

  `ObjectInputStream` ⟶ `VSObjectInputStream`

```
VSObjectInputStream(InputStream in);
InputStream getInputStream();
Object readObject();
void close();
```


Übungsaufgabe 2: Aufgabenstellung

Implementierung eines Objekt-Stroms

- {S,Des}erialisierung im Objekt-Strom

  `Serializable` verwenden

- {S,Des}erialisierung im Objekt

  `Externalizable` ⟶ `VSExternalizable`

```
public interface VSExternalizable extends Serializable {
    public void writeExternal(VSObjectOutputStream out)
    public void readExternal(VSObjectInputStream in)
}
```

- Nachbildung des Gedächtnis-Mechanismus

Lösungsskizze

`VSObjectOutputStream.writeObject()`

```
public void writeObject(Object obj) throws [...] {
    if('obj' wurde bereits gesendet) {
        sende 'obj'-Handle;
    } else {
        Class objClass = obj.getClass();
        if('objClass' ist Array) {
            Sonderbehandlung -> Arrays;
        } else {
            sende 'obj' als "normales" Objekt;
        }
    }
}
```


Lösungsskizze

Objekt senden

```
if('objClass' implementiert VSExternalizable) {
    rufe writeExternal() an 'obj' auf;
} else if('objClass' implementiert Serializable) {
    for(alle Attribute) {
        if(Attribut ist als "static final" gekennzeichnet ||
           Attribut ist als "transient" gekennzeichnet) {
            ignoriere Attribut;
        } else if(Attribut ist primitiver Datentyp) {
            Sonderbehandlung -> primitive Datentypen;
        } else {
            writeObject(Attribut); <-- Rekursion!
        }
    }
} else {
    werfe NotSerializableException;
}
```

Testen der Implementierung

- Test im Pub-Verzeichnis (`/proj/i4vs/pub/a2/`)
- Senden & Empfangen von `VSMessageExample`-Objekten
- 3 Einzeltests
  - Exceptions
  - Einzelnes `VSMessageExample`-Objekt
  - Mehrere (verkettete) `VSMessageExample`-Objekte
- Wichtig: `VSObjectOutputStream` und `VSObjectOutputStream` sollen auf beliebige Objekte ausgelegt sein!

Randnotiz: Klassen ohne Default-Konstruktoren

- Deserialisierung eines Objekts am Empfänger
  - Ziel: Bereitstellung von Speicherplatz für die Daten eines bereits bestehenden Objekts
  - Problem: Ausführung eines Konstruktors kann Seiteneffekte haben
- Wie macht's Java?
  - Verwendung der Factory `sun.reflect.ReflectionFactory`
  - Bereitstellung eines Spezial-Konstruktors
- Änderung am `VSObjectInputStream`
  - Code unter `/proj/i4vs/pub/a2/non-default-constructor.txt`
  - Erzeugung des Objekts per Default-Konstruktor...

```
Object object = objectClass.newInstance();
```

...ersetzen durch...


```
import sun.reflect.ReflectionFactory;
[...]

// Suche erste Oberklasse, die nicht 'Serializable' ist
Class c = objectClass;
while(Serializable.class.isAssignableFrom(c)) {
    c = c.getSuperclass();
    if(c == null) throw new NotSerializableException();
}

// Hole Default-Konstruktor dieser Klasse
Constructor instConstr = c.getConstructor(new Class[0]);

// Hole Spezialkonstruktor
ReflectionFactory reflFactory =
    (ReflectionFactory) AccessController.doPrivileged(
        new ReflectionFactory.GetReflectionFactoryAction());
Constructor constructor =
    reflFactory.newConstructorForSerialization(objectClass,
                                               instConstr);

// Instanziere Objekt
Object object = constructor.newInstance((Object[]) null);
```