

---

## 2 Übungsaufgabe #2: Marshalling

### 2.1 Allgemeines

In dieser Aufgabe soll die Grundfunktionalität der Java-Klassen *ObjectOutputStream* und *ObjectInputStream* nachgebildet werden ( $\rightarrow$  *VObjectOutputStream* bzw. *VObjectInputStream*), mit deren Hilfe es möglich ist, Objekte über einen gewöhnlichen *ByteStream* auszutauschen. Die beiden zu implementierenden Klassen müssen dabei folgende Konstruktoren und Methoden anbieten:

```
public class VObjectOutputStream {
    public VObjectOutputStream(OutputStream out);
    public OutputStream getOutputStream(); // Getter-Methode für 'out'
    public void writeObject(Object object) throws IOException, NotSerializableException;
    public void reset() throws IOException;
    public void close() throws IOException;
}

public class VObjectInputStream {
    public VObjectInputStream(InputStream in);
    public InputStream getInputStream(); // Getter-Methode für 'in'
    public Object readObject() throws IOException, ClassNotFoundException;
    public void close() throws IOException;
}
```

### 2.2 Klassenspezifische Serialisierung (optional für Nebenfächler)

Bevor ein Objekt übermittelt werden kann, muss es zuerst serialisiert werden. Zunächst soll dieser Schritt klassenspezifisch erfolgen, wofür in Java das Interface *Externalizable* zur Verfügung steht. Um diesen Mechanismus nachzubilden, soll in dieser Aufgabe eine (leicht abgewandelte) Schnittstelle *VSEExternalizable* mit identischer Semantik zum Einsatz kommen:

```
public interface VSEExternalizable extends Serializable {
    public void writeExternal(VObjectOutputStream out) throws IOException;
    public void readExternal(VObjectInputStream in) throws IOException;
}
```

*writeExternal()* übernimmt dabei die Serialisierung und *readExternal()* die Deserialisierung der internen Daten des Objekts an dem sie aufgerufen werden.

Aufgaben:

- $\rightarrow$  Implementierung der Klassen *VObjectOutputStream* und *VObjectInputStream*, so dass diese beim Aufruf von  $\{write, read\}Object()$  für  $\{S, Des\}$ erialisierung die *VSEExternalizable*-Methoden des übergebenen Objekts verwenden - sofern *VSEExternalizable* überhaupt implementiert wird.
- $\rightarrow$  Test der Implementierung, indem für ein Beispielobjekt (z.B. *VSMMessageExample* aus 2.5) die Methoden *writeExternal()* und *readExternal()* implementiert werden.

### 2.3 Allgemeine Serialisierung (für alle)

Üblicherweise liegen  $\{S, Des\}$ erialisierung eines Objekts in Java im Verantwortungsbereich des Objektstroms, d.h. dieser entscheidet alleinig darüber, wie ein Objekt zu  $\{, de\}$ serialisieren ist - sofern es *Serializable* implementiert. Diese Automatisierung soll nun auch für *VObject\{Out, In\}putStream* realisiert werden. Es wird empfohlen sich dabei an der in der Tafelübung vorgeschlagenen Lösungsskizze zu orientieren und die Funktionalität schrittweise zu erweitern. Eine mögliche Reihenfolge bei der Vorgehensweise ist folgende:

1. Primitive Datentypen
2. Arrays
3. Objekt-Referenzen

Aufgaben:

- $\rightarrow$  Erweiterung bzw. Implementierung von *VObjectOutputStream* und *VObjectInputStream*, so dass jede Art von serialisierbaren Objekten übertragen werden können.

---

Hinweise:

- Beim Aufruf von *writeObject()* muss zunächst überprüft werden, ob das zu übertragende Objekt nach dem Java-Standard auch tatsächlich serialisierbar ist. Ist dies nicht der Fall, soll eine *NotSerializableException* geworfen werden.
- Arrays sind spezielle Objekte und müssen gesondert betrachtet werden (→ siehe Tafelübung).
- Objekt-Referenzen können auch *null* sein.
- Mit „*transient*“ oder „*static final*“ gekennzeichnete Attribute sind von der Übertragung auszuschließen.
- Handelt es sich bei einem Attribut um eine Objektreferenz, so muss das dazugehörige Objekt ebenfalls übermittelt werden.

## 2.4 Stream-Gedächtnis

Um Zyklen in Objektgraphen bewältigen zu können bzw. zu verhindern, dass identische Daten mehrmals gesendet werden müssen, führen sowohl der Aus- als auch der Eingabestrom eine Aufzeichnung über die ausgetauschten Objekte. Bei jedem weiteren Aufruf von *writeObject()* mit dem selben Objekt muss somit nur ein Handle (z.B. der Index eines Tabelleneintrags) übertragen werden. Erreicht ein Handle die InputStream-Seite, wird von *readObject()* eine Referenz auf das bereits bestehende, korrespondierende Objekt zurückgegeben. (Es wird kein neues Objekt erzeugt!) Diese Vorgehensweise soll nun auch in *VXObject{Out,In}putStream* integriert werden.

Aufgaben:

- Implementierung des Gedächtnis-Mechanismus. (für alle)
- Implementierung der Methode *reset()* des *VXObjectOutputStreams*, mit der sich das Gedächtnis zwischenzeitlich löschen bzw. zurücksetzen lässt. (optional für Nebenfächler)

## 2.5 Testen der Implementierung (für alle)

Abschließend sollen die implementierten Klassen getestet werden. Hierfür sind die Dateien *VSExternalizable.java*, *VSMMessageExample.java* und *VSMarshallingTest.java* aus dem Pub-Verzeichnis (*/proj/i4vs/pub/a2/*) zu kopieren und im Repository unter *vsue/tests* (siehe 2.6) abzulegen. Der Test kann im CIP-Pool wie folgt auf der Kommandozeile kompiliert und gestartet werden:

```
cd <Projektverzeichnis>
javac -cp /usr/share/java/junit.jar:vsue/marshalling: vsue/tests/*.java
junit vsue.tests.VSMarshallingTest
```

*VSMarshallingTest* ist ein JUnit-Test (mehr dazu in späteren Übungen), der hintereinander drei Einzeltests ausführt. In den Tests wird versucht mehrere Objekte vom Typ *VSMMessageExample* über einen *VXObjectOutputStream* in eine Datei zu schreiben und sie anschließend wieder per *VXObjectInputStream* auszulesen. Für eine erfolgreiche Bearbeitung der 2. Übungsaufgabe ist es erforderlich, dass alle Einzeltests bestanden werden. Dies wird durch die Zeile “OK (3 tests)” am Ende der Ausgabe signalisiert.

## 2.6 Abgabe: am 27.5.2009/17.6.2009 in der Übung

Die Teile des im Rahmen der kompletten VS-Übung implementierten Fernaufrufsystems werden in einem einzigen Package (*vsue*) zusammengefasst. Für jede Teilaufgabe soll dabei ein gesondertes Subpackage angelegt werden, in dem alle neu hinzugekommenen Dateien abzulegen sind. Für Aufgabe 2 ist dies *vsue.marshalling*. Entsprechend muss die Ordnerstruktur im Projektverzeichnis nach Bearbeitung dieser Aufgabe (ohne Ordner aus Aufgabe 1) wie folgt aussehen:

```
<Projektverzeichnis>
|- vsue
   |- marshalling (Dateien für Aufgabe 2)
   |- tests      (Test-Dateien)
```