Hinweise für die Übungsaufgaben zu Verteilte Systeme

- Jeder Teilnehmer bekommt ein Projektverzeichnis unter

  `/proj/i4vs/<loginname>`

  Dieses kann als Workspace für die Übungsaufgaben dienen.

- Die Lösungen sollen in Gruppen mit 3 Teilnehmern erstellt werden. Jede Gruppe bekommt ein eigenes Projekt-Repository, auf das unter folgender Adresse zugegriffen werden kann:

  `https://www4.cs.fau.de/i4vs/<gruppe>`

- Zur Erstellung des SVN-Repository muss von einem Übungspartner aus der Gruppe das Programm

  `/proj/i4vs/bin/vsgroups`

  aufgerufen werden. Die Zugangsdaten für das Repository werden daraufhin innerhalb von 48 Stunden per E-Mail mitgeteilt.

- Zum Abgabezeitpunkt sollte die finale Version der Lösung in das Repository eingecheckt sein. Die eigentliche **Abgabe der Aufgaben erfolgt durch Präsentation** der eigenen Lösung gegenüber einem Übungsleiter.

- Zusatzmaterial zu den Übungsaufgaben findet sich, falls verfügbar, im *Pub-Verzeichnis* unter

  `/proj/i4vs/pub/<aufgabe>`

- Die *Java Coding Guidelines* sollten befolgt werden:
  - Klassennamen beginnen mit einem Großbuchstaben
  - Methodennamen beginnen mit einem Kleinbuchstaben
  - Variablennamen beginnen mit einem Kleinbuchstaben
  - Konstante (`static final`) Variablen bestehen nur aus Großbuchstaben
  - Namen von Variablen beschreiben deren Zweck

- Wenn eine bestimmte Programmstruktur in der Aufgabenstellung verlangt wird, muss die Lösung diese einhalten. Die Lösung soll insbesondere
  - Namen von Klassen und Packages
  - Sichtbarkeit von Variablen und Methoden
  - Namen, Parametertypen und Rückgabewerte von Methoden

  wie in der Aufgabenstellung beschrieben verwenden.

- Der Code muss lesbar und nachvollziehbar sein. Falls komplizierte Teile vorkommen, sollten diese mit Kommentaren erläutert werden.

- Die Lösungen müssen eigenständig erstellt worden sein. Verstöße werden geahndet.

Bei Problemen mit der Aufgabenstellung könnt ihr euch jederzeit an uns wenden:

`vs@i4.informatik.uni-erlangen.de`

1 Übungsaufgabe #1: Java RMI

Im Rahmen der ersten drei Übungsaufgaben soll ein eigenes, plattformunabhängiges Fernaufrufsystem nach dem Vorbild von Java RMI entwickelt werden. Ziel der ersten Aufgabe ist es, den Umgang mit Java RMI kennenzulernen sowie ein Kommunikationssystem für das eigene Fernaufrufsystem bereitzustellen.

1.1 Java RMI (für alle)

Zunächst soll ein einfacher verteilter Dienst umgesetzt werden, dessen Implementierung auf Fernaufrufe mittels Java RMI zurückgreift. Bei der zu realisierenden Anwendung handelt es sich um ein von einem zentralen Server verwaltetes schwarzes Brett, an das verschiedene Nutzer per Fernaufruf Botschaften heften können; zusätzlich soll es Nutzern möglich sein, die letzten Botschaften abzufragen bzw. sich am Dienst zu registrieren, um über das Eintreffen neuer Botschaften informiert zu werden. Alle Botschaften umfassen eine Benutzerkennung (`uid`), einen Titel (`title`) sowie einen Text (`message`) und weisen folgendes einheitliches Format auf:

```
public class VSBoardMessage {
    private int uid;
    private String title;
    private String message;
}
```

1.1.1 Bereitstellung der Dienstimplementierung

Im ersten Schritt ist die Anwendung zu implementieren, die das schwarze Brett als Dienst zur Verfügung stellt, und hierfür folgende Schnittstelle `VSBoard` anbietet:

```
public interface VSBoard extends Remote {
    public void post(VSBoardMessage message) throws RemoteException;
    public VSBoardMessage[] get(int n) throws IllegalArgumentException, RemoteException;
    public void listen(VSBoardListener listener) throws RemoteException;
}

public interface VSBoardListener extends Remote {
    public void newMessage(VSBoardMessage message) throws RemoteException;
}
```

Ein Aufruf von `post()` fügt dem schwarzen Brett eine neue Botschaft hinzu. Mittels `get()` lassen sich die `n` neuesten Botschaften abfragen; sollte der Parameter `n` kleiner als 0 sein, wirft `get()` eine `IllegalArgumentException`. Mit Hilfe der Methode `listen()` kann sich ein Nutzer beim Dienst registrieren. Registrierte Nutzer werden jedes Mal durch einen Rückruf (`newMessage()`) informiert, sobald eine neue Botschaft ans schwarze Brett geheftet wird; Nutzer müssen hierzu die Schnittstelle `VSBoardListener` implementieren.

Aufgaben:

→ Implementierung der Klasse `VSBoardMessage`
→ Implementierung einer Klasse `VSBoardImpl`, die die Schnittstelle `VSBoard` implementiert

1.1.2 Verteilung mittels Java RMI

Im nächsten Schritt soll das schwarze Brett als Client-Server-Anwendung realisiert werden. Hierzu ist auf Server-Seite eine Klasse `VSBoardRMIServer` zu implementieren, die ein schwarzes Brett instanziiert, es als Remote-Objekt exportiert und anschließend mittels einer Registry bekannt macht.

Die Nutzung des schwarzen Bretts wird auf Client-Seite mittels einer Klasse `VSBoardRMIClient` abgewickelt. Die Interaktion mit dem Nutzer soll dabei über eine Eingabemaske ähnlich einer Shell erfolgen. Zum Erstellen einer neuen Botschaft unter der Benutzerkennung 47 könnte zum Beispiel folgender Aufruf dienen:

```
> post 47 "Das ist der Titel." "Das ist die Botschaft."
```

Aufgaben:

→ Implementierung der Klassen `VSBoardRMIServer` und `VSBoardRMIClient`
→ Testen der Implementierung

Hinweise:

- Der Export von Objekten soll explizit per `UnicastRemoteObject.exportObject()` erfolgen.
- Die eigene Implementierung ist mit mehreren Clients und auf mehrere Rechner verteilt zu testen.

1.2 Kommunikationssystem

Die Grundlage des eigenen Fernaufrufsystems stellt ein Mechanismus zum Austausch von Nachrichten zwischen Rechnern dar. Hierzu soll als unterste Schicht zunächst die Übermittlung von Datenpaketen (Byte-Arrays) über eine TCP-Verbindung realisiert werden. Eine darauf aufbauende zweite Schicht ermöglicht dann das Senden und Empfangen beliebiger Nachrichten in Form von Java-Objekten.

1.2.1 Übertragung von Datenpaketen (für alle)

Die Kommunikation mit einem anderen Rechner soll über eine Klasse `VSConnection` abgewickelt werden, die intern eine TCP-Verbindung zur Übertragung von Daten nutzt, und mindestens folgende Methoden bereitstellt:

```
public class VSConnection {
    public void sendChunk(byte[] chunk);
    public byte[] receiveChunk();
}
```

Mit Hilfe der Methode `sendChunk()` lassen sich Datenpakete beliebiger Größe über eine bereits bestehende Verbindung übertragen. Mittels `receiveChunk()` wird ein von der Gegenseite gesendetes Datenpaket in Empfang genommen. Ein Aufruf von `receiveChunk()` blockiert dabei solange, bis das übermittelte Datenpaket vollständig empfangen wurde. Zur Signalisierung von Fehlersituationen sollen beide Methoden geeignete Exceptions werfen.

Aufgabe:

→ Implementierung der Klasse `VSConnection`

Hinweise:

- Die Implementierung von `VSConnection` soll die Daten direkt auf den `OutputStream` des TCP-Sockets schreiben bzw. sie von seinem `InputStream` lesen. (Keine Verwendung von komplexeren Stream-Klassen!)
- Die Methode `OutputStream.write(int)` überträgt nur die niedrigsten 8 Bits des übergebenen Integer.

1.2.2 Übertragung von Objekten (für alle)

Unter Verwendung der Methoden `sendChunk()` und `receiveChunk()` der `VSConnection` aus Teilaufgabe 1.2.1 soll nun eine Möglichkeit zum Senden und Empfangen beliebiger Objekte in einer Klasse `VSObjectConnection` realisiert werden, die mindestens die beiden folgenden Methoden aufweist:

```
public class VSObjectConnection {
    public void sendObject(Serializable object);
    public Serializable receiveObject();
}
```

Einzige Voraussetzung für den Versand bzw. Empfang von Objekten mittels `sendObject()` bzw. `receiveObject()` ist, dass die Objekte die in Java für diesen Zweck vorgesehene Marker-Schnittstelle `Serializable` implementieren.

Aufgabe:

→ Implementierung der Klasse `VSObjectConnection`

Hinweis:

- Für die {S,Des}erialisierung von Objekten sind `Object{Out,In}putStream`s zu verwenden.

1.2.3 Client und Server (für alle)

Zur Überprüfung, ob die Implementierung von `VSObjectConnection` Objekte korrekt übermittelt, soll ein einfacher Echo-Dienst zum Einsatz kommen. Auf Server-Seite ist hierzu eine Klasse `VSServer` zu erstellen, die eingehende Verbindungen annimmt und sie jeweils in einem eigenen Worker-Thread bearbeitet. Die einzige Aufgabe eines Worker-Thread besteht darin, jedes von der Gegenseite eintreffende Objekt unverändert zurückzuschicken, solange der Client die Verbindung offen hält.

Auf Client-Seite ist eine Klasse `VSClient` zu realisieren, die eine Verbindung zum Server herstellt und anschließend verschiedene Objekte (z. B. `Integer`s, `String`s, Arrays, `VSBoardMessage`s) über diese sendet. Bei den daraufhin vom Server zurück gesendeten Objekten ist zu überprüfen, ob ihr Zustand dem der Original-Objekte entspricht.

Aufgaben:

→ Implementierung der Klassen `VSClient` und `VSServer`
→ Testen der `VSObjectConnection` mit unterschiedlichen Objekten

1.2.4 Analyse der serialisierten Daten (optional für 5,0 ECTS)

Die Implementierung der `VSObjectConnection` aus Teilaufgabe 1.2.2 greift zur {S,Des}erialisierung von als `Serializable` gekennzeichneten Objekten auf die Standardmechanismen des `Object{Out,In}putStream` zurück. Bei der Entwicklung dieser Mechanismen standen Prinzipien wie Flexibilität und Fehlertoleranz im Vordergrund, Ziele wie Effizienz und insbesondere die Minimierung der Datenmenge wurden dagegen hinten angestellt. Dies soll im Rahmen dieser Teilaufgabe anhand von Objekten einer Beispielklasse `VSTestMessage` verdeutlicht werden.

```
public class VSTestMessage implements Serializable {
    private int integer;
    private String string;
    private Object[] objects;
}
```

Zur Analyse sind dabei die vom `ObjectOutputStream` in der Methode `VSObjectConnection.sendObject()` serialisierten Daten näher zu betrachten. Hierzu ist die Methode so zu erweitern, dass die einzelnen Bytes des erzeugten Datenpakets als Hexadezimalwerte auf der Kommandozeile ausgegeben werden. Zusätzlich ist das Datenpaket als `String` am Bildschirm darzustellen, um die darin enthaltenen Zeichenketten sichtbar zu machen.

Aufgaben:

→ Übertragung verschiedener `VSTestMessage`-Objekte (→ unterschiedliche Belegung der Attribute mit Werten) zwischen `VSClient` und `VSServer` und Analyse der daraus resultierenden serialisierten Daten

→ Wie viele Bytes umfasst eine „leere“ `VSTestMessage` (`integer` ist `0`, `string` und `objects` sind `null`)?

→ Mit welchem Byte-Wert signalisiert der `ObjectOutputStream`, dass ein Attribut `null` ist?

→ Welchen Einfluss hat der Wert von `integer` auf die Größe der serialisierten Daten?

→ Welchen Einfluss hat ein einzelner Buchstabe in `string` auf die Größe der serialisierten Daten?

→ Welchen Einfluss hat die Array-Größe von `objects` auf die Größe der serialisierten Daten?

Hinweis:

- Die Methode `Integer.toHexString()` liefert die Hexadezimaldarstellung eines Werts als Zeichenkette.

1.2.5 Optimierte Serialisierung und Deserialisierung (optional für 5,0 ECTS)

Wie in der Tafelübung vorgestellt, erlaubt es die Schnittstelle `Externalizable` dem Programmierer, die {S,Des}erialisierung von Objekten klassenspezifisch zu implementieren. Dies kann zum Beispiel dazu genutzt werden, den Umfang der serialisierten Daten zu reduzieren. Ziel dieser Teilaufgabe ist es, dies durch eine manuelle Implementierung der Methoden `readExternal()` und `writeExternal()` für `VSTestMessage`-Objekte zu zeigen. Für die Attribute der Klasse `VSTestMessage` sollen dabei folgende Annahmen getroffen werden:

- Der Wertebereich von `integer` liegt zwischen `Integer.MIN_VALUE` und `Integer.MAX_VALUE`.
- Die Zeichenkette `string` umfasst höchstens `Integer.MAX_VALUE` Zeichen.
- In `objects` können beliebige Objekte enthalten sein; die maximale Array-Größe ist `Short.MAX_VALUE`.

Aufgaben:

→ Implementierung der Methoden `readExternal()` und `writeExternal()` für die Klasse `VSTestMessage`

→ Wie viele Bytes umfasst eine „leere“ `VSTestMessage` (vgl. Teilaufgabe 1.2.4)?

→ Minimierung der serialisierten Datenmenge (soweit möglich)

→ Mit welchen Maßnahmen ließe sich die Datenmenge noch weiter reduzieren?

Hinweis:

- Die Optimierungen sollen sich auf die Methoden `readExternal()` und `writeExternal()` beschränken.

Abgabe: am Mi. 7.5.2014 in der Rechnerübung

Die für diese Übungsaufgabe erstellten Klassen sind jeweils in einem Subpackage `vsue.rmi` (Teilaufgabe 1.1) bzw. `vsue.communication` (Teilaufgabe 1.2) zusammenzufassen.