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. Dazu erhält jede Gruppe ein eigenes Projekt-Repository in GitLab (`https://gitlab.cs.fau.de/`).

- Zur Festlegung der Gruppenzugehörigkeit muss von einem Übungspartner aus der Gruppe das Programm

  `/proj/i4vs/bin/vsgroups`

  aufgerufen werden. Die Gruppennummer wird daraufhin innerhalb von 48 Stunden per E-Mail mitgeteilt.

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

- Sollte eine Präsentation der eigenen Implementierung am Abgabetermin nicht möglich sein, ist dies **im Voraus** entweder per E-Mail (siehe unten) oder persönlich einem Übungsleiter mitzuteilen.

- Hilfestellungen und Vorgaben zu den Übungsaufgaben finden sich 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.

- Es ist empfehlenswert, die jeweilige Aufgabenstellung **vor** der Bearbeitung einer Aufgabe **vollständig** zu lesen, um von Beginn an einen Überblick über das zu erstellende System zu haben.

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

`i4vs@lists.cs.fau.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 und anschließend ein Kommunikationssystem für das eigene Fernaufrufsystem bereitzustellen.

1.1 Auktionsdienst (für alle)

Zunächst soll ein verteilter Auktionsdienst mittels Java RMI umgesetzt werden. Der Dienst ermöglicht es seinen Nutzern, selbst neue Auktionen zu erstellen sowie Gebote für bereits laufende Auktionen abzugeben.

```
public class VSAuction {
    private final String name;
    private int price;
}
```

Jede Auktion (`VSAuction`) lässt sich mittels eines initial festzulegenden Namens `name` eindeutig identifizieren. Zentrale Aufgabe des Auktionsdiensts ist es, für jede Auktion das aktuell höchste Gebot `price` zu verwalten. Gewinner einer Auktion ist der Nutzer, der bis zum Ablauf der Auktion das höchste Gebot abgegeben hat.

```
public interface VSAuctionEventHandler {
    public void handleEvent(VSAuctionEventType event, VSAuction auction);
}
```

Neben der Auktionsverwaltung bietet der Dienst seinen Nutzern die Möglichkeit, sie über bestimmte Ereignisse zu informieren. Hierfür nutzt er die Methode `handleEvent()` der Schnittstelle `VSAuctionEventHandler`, mit deren Hilfe er einem Nutzer ein in Verbindung mit einer Auktion `auction` aufgetretenes Ereignis `event` mitteilen kann. Folgende Ereignisse (Enum `VSAuctionEventType`) werden hierbei betrachtet: `HIGHER_BID` informiert den Nutzer mit dem bisher höchsten Gebot, dass er überboten wurde, `AUCTION_END` benachrichtigt den Ersteller einer Auktion über deren Ende und `AUCTION_WON` teilt einem Nutzer mit, dass er den Zuschlag erhalten hat.

1.1.1 Bereitstellung der Dienstimplementierung

Im ersten Schritt ist die Anwendung zu implementieren, die den Auktionsdienst zur Verfügung stellt, und hierfür folgende Schnittstelle `VSAuctionService` anbietet:

```
public interface VSAuctionService {
    public void registerAuction(VSAuction auction, int duration,
                                VSAuctionEventHandler handler) throws VSAuctionException;
    public VSAuction[] getAuctions();
    public boolean placeBid(String userName, String auctionName, int price,
                            VSAuctionEventHandler handler) throws VSAuctionException;
}
```

Ein Aufruf von `registerAuction()` ermöglicht es einem Nutzer, eine Auktion `auction` zu registrieren, die nach `duration` Sekunden abläuft. Sollte eine Auktion mit demselben Namen bereits existieren, wird dies per `VSAuctionException` signalisiert. Die Methode `getAuctions()` liefert alle Auktionen zurück, die zum Zeitpunkt des Aufrufs laufen. Mittels `placeBid()` kann ein Nutzer `userName` ein neues Gebot `price` für eine Auktion `auctionName` abgegeben. Am Rückgabewert kann ein Nutzer erkennen, ob er das aktuell höchste Gebot abgegeben hat. Existiert keine Auktion mit dem angegebenen Namen, wirft die Methode eine `VSAuctionException`.

Aufgabe:

→ Implementierung einer Klasse `VSAuctionServiceImpl`, die `VSAuctionService` implementiert

Hinweis:

- Die unter `/proj/i4vs/pub/aufgabe1` bereitgestellten Klassen dürfen bei Bedarf beliebig erweitert werden.

1.1.2 Verteilung mittels Java RMI

Im nächsten Schritt soll der Auktionsdienst als Client-Server-Anwendung realisiert werden. Hierzu ist auf Server-Seite eine Klasse `VSAuctionRMIServer` zu implementieren, die einen Auktionsdienst instanziiert, ihn als Remote-Objekt exportiert und mittels Registry bekannt macht. Auf Client-Seite soll eine Klasse `VSAuctionRMIClient` (siehe `/proj/i4vs/pub/aufgabe1`) es dem Nutzer ermöglichen, über eine Shell mit dem Dienst zu interagieren.

Aufgabe:

→ Implementierung der Klassen `VSAuctionRMIServer` und `VSAuctionRMIClient`

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 empfangen. Ein Aufruf von `receiveChunk()` blockiert dabei so lange, bis das Datenpaket vollständig übermittelt 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();
}
```

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 verschiedene Objekte (z. B. `Integer`s, `String`s, Arrays, `VSAuction`s) über diese sendet. Bei den daraufhin vom Server zurückgesendeten Objekten ist zu überprüfen, ob ihr Zustand dem der Originalobjekte 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 vor allem 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 erläutert, erlaubt die Schnittstelle `Externalizable` eine klassenspezifische Implementierung der {S,Des}erialisierung von Objekten. 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 Di., 4.5.2021 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.