Hinweise für die Übungsaufgaben zu Verteilte Systeme

Arbeitsumgebung

- Alle Teilnehmenden bekommen ein eigenes Projektverzeichnis unter

  `/proj/i4vs/<loginname>`

  Dieses ist aus dem CIP-Pool zugreifbar und kann als Workspace für die Übungsaufgaben dienen.

- Hilfestellungen und Vorgaben zu den Übungsaufgaben finden sich im *Pub-Verzeichnis* unter

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

- Die Aufgaben sollen in 3er-Gruppen bearbeitet werden. Dazu erhält jede Gruppe ein eigenes Projekt-Repository im GitLab (`https://gitlab.cs.fau.de/`).

  Wichtig: Bei (zusätzlicher) Verwendung eines eigenen Repositories o.Ä. ist sicherzustellen, dass dieses **nicht von außerhalb der eigenen Gruppe** zugreifbar ist.

- Da die Aufgaben in VS teilweise aufeinander aufbauen, bietet es sich an, alle Aufgaben in einem gemeinsamen Projekt zu bearbeiten und keine separaten Verzeichnisse je Aufgabe anzulegen.

Bearbeitung der Aufgaben

- Die Aufgaben können an vielen Stellen in Arbeitspakete aufgeteilt werden, dennoch empfiehlt sich eine enge Zusammenarbeit in der Gruppe. Eine mögliche Arbeitsweise innerhalb der Gruppe ist

  1. Alle Gruppenmitglieder lesen sich **vor** der Bearbeitung das Aufgabenblatt **vollständig** durch
  2. Gemeinsames Treffen zur Absprache (u.A. Überblick Gesamtsystem, Zusammenspiel der einzelnen Komponenten, Aufteilung in Arbeitspakete, evtl. Pair-Programming)
  3. Implementierung der einzelnen Komponenten (in Gruppenarbeit oder einzeln)
  4. *Eventuell: Gemeinsames Treffen zur Integration und Testen der Gesamtfunktionalität*

- Der Code sollte lesbar und nachvollziehbar sein. Komplizierte Teile sollten mit Kommentaren erläutert werden.

- Wenn eine bestimmte Programmstruktur in der Aufgabenstellung verlangt wird, sollte die Lösung diese einhalten:

  – Namen von Klassen und Packages
  – Sichtbarkeit, Namen, Parametertypen und Rückgabewerte von Methoden (und Variablen)

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

- Aufgabenteile, die für die 5-ECTS-Variante optional sind, werden auf dem Aufgabenblatt dementsprechend gekennzeichnet (vgl. Teilaufgabe 1.2.4 bei Aufgabe 1). Für 7.5 ECTS müssen **alle** Aufgabenteile bearbeitet werden.

Abgaben

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

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

- Die Abgabe kann entweder während der regulären Rechnerübung erfolgen oder an einem individuell vereinbarten Termin stattfinden. Die Vereinbarung für individuelle Termine erfolgt über einen Online-Terminplaner.

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

`i4vs@lists.cs.fau.de`

Für organisatorische Fragen dient die folgende Mailingliste:

`i4vs-owner@lists.cs.fau.de`

Übungsaufgabe #1: Java RMI

Im Rahmen der ersten drei Übungsaufgaben soll ein eigenes Fernaufrufsystem nach dem Vorbild von Java RMI entwickelt werden. Ziel der ersten Aufgabe ist es dabei zunächst, den Umgang mit Java RMI anhand einer Beispielanwendung kennenzulernen und anschließend ein Kommunikationssystem für das eigene Fernaufrufsystem zu implementieren.

1.1 Auktionsdienst (für alle)

Als Beispielanwendung dient ein verteilter Auktionsdienst, der mittels Java RMI als Client-Server-System umgesetzt werden soll. Der Dienst ermöglicht es Nutzenden, selbst neue Auktionen zu erstellen, Informationen über laufende Auktionen zu erfragen sowie eigene Gebote für aktuelle 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 Client, 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 Nutzenden die Möglichkeit, sie über bestimmte Ereignisse per Rückruf zu informieren. Hierfür verwendet er die Methode `handleEvent()` der auf Client-Seite zu implementierenden Schnittstelle `VSAuctionEventHandler` (vgl. `deposit()` in `VSAccount` im Bank-Server-Beispiel [Folien 1.2 : 6–17]). Ein einzelner `handleEvent()`-Aufruf repräsentiert hierbei genau ein Ereignis `event` für eine Auktion `auction`. Folgende Ereignisse (Enum `VSAuctionEventType`) werden allgemein betrachtet: `HIGHER_BID` informiert den Client mit dem bisher höchsten Gebot, dass er überboten wurde, `AUCTION_END` benachrichtigt den Ersteller einer Auktion über deren Ende und `AUCTION_WON` teilt einem Client mit, dass er den Zuschlag erhalten hat.

1.1.1 Bereitstellung der Dienstimplementierung

Im ersten Schritt ist die Auktionsdienstlogik in Form eines lokales Objekts zu implementieren, also ohne Berücksichtigung der späteren Verteilung. Konkret verfügt die Beispielanwendung über folgende Schnittstelle `VSAuctionService`:

```
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 Client, eine Auktion `auction` zu starten, die nach `duration` Sekunden abläuft; der Parameter `handler` dient der Übergabe einer (Remote-)Referenz über die der Auktionsdienst den Client später bei Ereignissen zurückrufen kann. 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 Client `userName` ein neues Gebot `price` für eine Auktion `auctionName` einreichen. Am Rückgabewert lässt sich anschließend erkennen, ob der Client 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 verteilte 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 [Folie 1.2 : 1]. Auf Client-Seite soll eine Klasse `VSAuctionRMIClient` (siehe `/proj/i4vs/pub/aufgabe1`) es Nutzenden 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 über ein Netzwerk 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 serialisierbaren 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 bestehende Verbindung übertragen. Per `receiveChunk()` wird ein von der Gegenseite gesendetes Datenpaket empfangen. Die Methode `receiveChunk()` blockiert dabei so lange, bis das Datenpaket vollständig übermittelt wurde. Ein Aufruf von `sendChunk()` lässt sich somit also genau einem Aufruf von `receiveChunk()` zuordnen. Zur Signalisierung von Fehlersituationen sollen beide Methoden geeignete Exceptions werfen. Die genaue Umsetzung des Verbindungsaufbaus ist freigestellt.

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 realisiert werden. Die hierzu gedachte Klasse `VSObjectConnection` weist mindestens die beiden folgenden Methoden auf:

```
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 [Folie 1.3 : 6].

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 über einen Server-Socket annimmt und sie jeweils in einem eigenen Worker-Thread bearbeitet. Die einzige Aufgabe eines solchen 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 zurückgeschickten Objekten ist zu überprüfen, ob ihre Zustände jeweils exakte Kopien der Originalobjekte repräsentieren.

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. Dieser Sachverhalt soll nun im Rahmen von Teilaufgabe 1.2.4 anhand von Objekten einer Beispielklasse `VSTestMessage` genauer untersucht 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 (→ Belegung der Attribute mit unterschiedlichen 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 im Begleitmaterial erläutert [Folie 1.3 : 7], 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:

- Bei der Umsetzung von Optimierungen reicht es im Rahmen dieser Teilaufgabe aus, sich auf die Methoden `readExternal()` und `writeExternal()` zu beschränken. Ansätze, die über diese beiden Methoden hinausgehen, sollten zwar gedanklich skizziert werden, eine konkrete Implementierung ist allerdings nicht erforderlich.

Abgabe: am Mi., 03.05.2023 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.