2 Übungsaufgabe #2: Stubs & Skeletons

In Übungsaufgabe 1 wurde mit der Entwicklung eines Kommunikationssystems die Grundlage zum Nachrichtenaustausch zwischen Rechnern geschaffen. Im Rahmen dieser Übungsaufgabe sollen nun darauf aufbauend die auf Client- und Server-Seite zur Durchführung eines Fernaufrufs benötigten Komponenten realisiert werden. Bevor sich an einem entfernten Objekt eine Methode aufrufen lässt, wird zunächst eine Referenz auf das betreffende Objekt benötigt. Eine solche *Remote-Referenz* verfügt üblicherweise über folgende Attribute:

```
public class VSRemoteReference implements Serializable {
    private String host;
    private int port;
    private int objectID;
}
```

Mit Hilfe der in einer Remote-Referenz enthaltenen Informationen lassen sich demnach der Zielrechner (`host`) sowie das entfernte Objekt (`port` und `objectID`) eindeutig identifizieren.

2.1 Umleitung lokaler Methodenaufrufe (für alle)

Ziel bei der Entwicklung des Fernaufrufsystems ist es, den Aufruf von Methoden an entfernten Objekten für den Nutzer transparent zu gestalten: Im Optimalfall merkt der Nutzer nichts davon, dass sein lokaler Methodenaufruf in Wirklichkeit abgefangen und in einen Fernaufruf umgewandelt wird. Wie in der Tafelübung gezeigt, lässt sich die Umleitung von Methodenaufrufen unter Zuhilfenahme von *dynamischen Proxies* implementieren, die als Stellvertreter für das eigentliche Objekt fungieren. Im Kontext des Fernaufrufsystems wird dabei eine Invocation-Handler-Klasse `VSInvocationHandler` für die Umleitung eines lokalen Aufrufs zum Server benötigt.

```
public class VSInvocationHandler implements InvocationHandler, Serializable {
    public VSInvocationHandler(VSRemoteReference remoteReference);
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable;
}
```

Im Konstruktor wird dem `VSInvocationHandler` eine Remote-Referenz übergeben, die das durch den dynamischen Proxy repräsentierte entfernte Objekt eindeutig kennzeichnet. Die Methode `invoke()` führt die Umwandlung des lokalen Aufrufs in einen Fernaufruf durch; sie muss also für jeden Aufruf eine Verbindung zum Server öffnen, eine passende Anfrage generieren (Marshalling der Aufrufparameter) und senden, die Antwort empfangen und verwerten (Unmarshalling des Rückgabewerts), und schließlich die Verbindung beenden.

Aufgabe:

→ Implementierung der Klassen `VSRemoteReference` und `VSInvocationHandler`

Hinweise:

- Für Verbindungen soll die Klasse `VSObjectConnection` aus Teilaufgabe 1.2.2 zum Einsatz kommen.
- Zur besseren Kapselung sind für Anfragen und Antworten eigene Nachrichtenobjekte zu verwenden.

2.2 Verwaltung von Remote-Objekten (für alle)

Um die Methoden eines Objekts auf Server-Seite für einen Zugriff per Fernaufruf nutzbar zu machen, muss es zunächst als Remote-Objekt exportiert werden. Die Verwaltung von Remote-Objekten soll eine Instanz der Klasse `VSRemoteObjectManager` übernehmen, die mindestens folgende Methoden anbietet:

```
public class VSRemoteObjectManager {
    public static VSRemoteObjectManager getInstance();
    public Remote exportObject(Remote object);
    public Object invokeMethod(int objectID, String genericMethodName, Object[] args);
}
```

`VSRemoteObjectManager` ist dabei als Singleton zu implementieren, auf den mit Hilfe der statischen Methode `getInstance()` zugegriffen werden kann. Die Methode `exportObject()` exportiert das als Parameter `object` übergebene Objekt und macht es somit für künftige Fernaufrufe erreichbar. Als Rückgabewert liefert diese Methode einen dynamischen Proxy (→ *Stub*) zurück, der den Zugriff auf `object` per Fernaufruf ermöglicht. Eintreffende Anfragen für Methodenaufrufe an exportierten Objekten werden mittels `invokeMethod()` bearbeitet: `objectID` kennzeichnet dabei die in der Remote-Referenz enthaltene Objektidentifikationsnummer, `genericMethodName` weist auf die aufzurufende Methode hin und `args` enthält alle benötigten Aufrufparameter. Der Rückgabewert der aufgerufenen Methode wird von `invokeMethod()` als `Object` zurückgegeben.

Aufgabe:

→ Implementierung der Klasse `VSRemoteObjectManager`

2.3 Client und Server (für alle)

Komplettiert wird die Server-Seite durch eine Komponente, die für die Verwaltung von Verbindungen und die Bearbeitung eintreffender Fernaufrufanfragen zuständig ist. Hierzu ist die Klasse `VSServer` aus Teilaufgabe 1.2.3 so anzupassen, dass sie folgende Aufgaben übernimmt: Empfang und Unmarshalling der Anfragen, Methodenaufruf per `VSRemoteObjectManager.invokeMethod()`, Marshalling und Versand der Antworten. `VSServer` übernimmt damit die Rolle eines generischen Skeletons.

Zum Testen des eigenen Fernaufrufsystems soll der Auktionsdienst aus Übungsaufgabe 1 zum Einsatz kommen. Hierzu sind, analog zu den Klassen `VSAuctionRMIClient` und `VSAuctionRMIServer` (siehe Teilaufgabe 1.1.2), zwei Klassen `VSAuctionClient` und `VSAuctionServer` zu implementieren, die das Exportieren und Registrieren der Remote-Objekte übernehmen sowie für die Interaktion mit dem Nutzer zuständig sind.

Aufgaben:

→ Anpassung der Klasse `VSServer`
→ Implementierung der Klassen `VSAuctionClient` und `VSAuctionServer`

Hinweise:

- Der aktuelle Stand der Implementierung erlaubt es, die Methoden des Auktionsdiensts zu nutzen, solange als `VSAuctionEventHandler`-Parameter jeweils `null` übergeben wird; die Benachrichtigung über Ereignisse wird erst in Teilaufgabe 2.4 unterstützt.
- Beim Testen der Implementierung ist zu überprüfen, dass das Fernaufrufsystem auf Server-Seite geworfene `VSAuctionException`s auch korrekt an den jeweiligen Aufrufer propagiert.
- Es soll weiterhin die von Java RMI bereitgestellte Registry zum Einsatz kommen.

2.4 Unterstützung von Rückrufen (für alle)

In der aktuellen Implementierung des Fernaufrufsystems werden für sämtliche Aufrufparameter einer Methode auf Server-Seite Kopien erzeugt (*Call-by-Value*). Um Rückrufe (*Callbacks*) zu ermöglichen, muss daher für die Weitergabe von Parametern zusätzlich *Call-by-Reference* unterstützt werden. Hierfür ist es erforderlich, dass ein Client-Stub bei der Zusammenstellung der Anfrage alle betroffenen Parameter durch passende Proxies ersetzt, die der Server-Seite einen Rückruf per Fernaufruf ermöglichen. *Call-by-Reference* soll dabei (wie in Java RMI) nur dann zum Einsatz kommen, wenn es sich bei einem Parameter um ein bereits exportiertes Remote-Objekt handelt; in allen anderen Fällen wird der Parameter (wie bisher) *by-Value* übertragen.

Aufgaben:

→ Erweiterung der Implementierung zur Unterstützung von Rückrufen
→ Testen der Implementierung

Hinweis:

- *Call-by-Reference* ist (mit umgekehrten Vorzeichen) auch für Rückgabewerte relevant.

2.5 Evaluierung des Fernaufrufsystems (optional für 5,0 ECTS)

Abschließend soll die Antwortzeit des eigenen Fernaufrufsystems ermittelt und mit der Antwortzeit eines Fernaufrufs in Java RMI verglichen werden. Hierzu ist ein Szenario zu implementieren, das die Antwortzeit eines `getAuctions()`-Aufrufs am Auktionsdienst in Abhängigkeit der Anzahl der zurückzugebenden Auktionen für beide Varianten evaluiert. Um sicherzustellen, dass die ermittelten Werte ein möglichst präzises Abbild der Realität darstellen, sind alle Messungen mehrfach durchzuführen und Durchschnittswerte zu bilden. Die auf diese Weise erhaltenen Ergebnisse jeder Variante sollen in einer kurzen Auswertung (z. B. in Form eines Diagramms) präsentiert und einander gegenübergestellt werden.

Aufgaben:

→ Vergleichende Evaluierung der beiden Fernaufrufsysteme anhand der `getAuctions()`-Methode
→ Wie verhält sich die Antwortzeit des eigenen Systems, wenn die Server-Verbindung für folgende Fernaufrufe wiederverwendet und nicht (wie in Teilaufgabe 2.1 gefordert) nach Erhalt der Antwort geschlossen wird?

Hinweise:

- Der bei TCP-Sockets in Java standardmäßig verwendete Nagle-Algorithmus sollte durch jeweils einmaligen Aufruf von `setTcpNoDelay(true)` an den Sockets der `VSConnection`s deaktiviert werden.
- Sollte sich herausstellen, dass das eigene System grandios unterlegen ist, ist dies mit Fassung zu tragen.

Abgabe: am Mi., 16.5.2018 in der Rechnerübung

Die für diese Übungsaufgabe erstellten Klassen sind in einem Subpackage `vsue.rpc` zusammenzufassen.