3 Übungsaufgabe #3: Dynamische Proxyerzeugung

3.1 Allgemeines

Ziel dieser Aufgabe ist es, alle für einen einfachen Fernaufruf benötigten Komponenten bereitzustellen. Im Rahmen der Übung wird dabei ein Client-Server-Szenario angenommen, bei dem vorerst ausschließlich eine Seite (Client) den Dienst der anderen (Server) in Anspruch nimmt. Bevor sich an einem entfernt liegenden 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.

3.2 Kommunikationssystem (für alle)

Zunächst soll ein einfaches Kommunikationssystem für den Nachrichtenverkehr erstellt werden. Hierfür wird auf Client- und Serverseite jeweils eine Instanz der Klasse *VSCommunication* benötigt mit der es möglich ist Verbindungen zu entfernten Rechnern zu öffnen. Über diese *VSConnection* werden dann die eigentlichen Nutzdaten in Form von *VSMessage*-Objekten gesendet und empfangen. Die Klassen *VSCommunication* und *VSConnection* müssen mindestens folgende Methoden bereitstellen:

```
public class VSCommunication {
    public VSConnection openConnection(String host, int port);
    public void closeConnection(VSConnection connection);
}

public class VSConnection {
    public boolean sendMessage(VSMessage msg);
    public VSMessage receiveMessage();
}
```

openConnection() wird auf Client-Seite aufgerufen, baut eine Verbindung zum Server auf und gibt sie zurück. Der Verbindungsabbau funktioniert über einen Aufruf von *closeConnection()*.

Aufgabe:

→ Implementierung der Klassen *VSCommunication* und *VSConnection*

Hinweise:

- Intern soll *VSConnection* unter Einbeziehung der in Aufgabe 2 erzeugten Klassen *VSObjectOutputStream* und *VSObjectInputStream* über eine Socketverbindung kommunizieren.
- Da *VSCommunication* auch auf der Serverseite eingesetzt wird, muss zusätzlich ein Mechanismus zur Verbindungsannahme implementiert werden. Ferner soll dem Nutzer des Kommunikationssystems eine eingehende Verbindung in irgendeiner Form signalisiert werden (z.B. durch einen Listener).
- Der Aufbau von *VSMessage* ist beliebig.

3.3 Serverseite (für alle)

3.3.1 Export von Objekten

Sollen die Methoden eines Objekts für einen Zugriff per Fernaufruf nutzbar sein, muss es zunächst als Remote-Objekt verfügbar gemacht ("exportiert") werden. Diese Aufgabe übernimmt die Klasse *VSRemoteObjectManager*, die mindestens folgende Methoden aufweisen soll:

```
public class VSRemoteObjectManager {
    public VSRemoteObjectManager(InetSocketAddress serverAddress);
    public void exportObject(Object object);
    public VSRemoteReference getRemoteReference(Class interfaceClass);
    public Object invokeMethod(int objectID, String genericMethodName, Object[] args);
}
```

Im Konstruktor wird *VSRemoteObjectManager* die Adresse des lokalen Servers mitgegeben. *exportObject()* sorgt dafür, dass sich das als Parameter übergebene Objekt künftig für Fernaufrufe verwenden lässt. Eine Remote-Referenz auf ein zuvor exportiertes Objekt lässt sich per *getRemoteReference()* erhalten/erzeugen, wobei als Parameter der Klassenname des Objekts bzw. der von ihm angebotenen Schnittstelle übergeben wird. Der eigentliche Aufruf einer Methode erfolgt über *invokeMethod()*: *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*

Hinweise:

- Der *genericMethodName*-Parameter von *invokeMethod()* muss die Methodenbeschreibung enthalten, die beim Aufruf von *toGenericString()* am entsprechenden *java.lang.reflect.Method*-Objekt des Remote-Interface zurückgegeben wird.
- Die Klasse *VSRemoteObjectManager* soll so implementiert sein, dass sie mit beliebigen Remote-Objekten zurecht kommt.

3.3.2 Server-Skeleton

Komplettiert wird die Serverseite durch die Klasse *VSServer* deren Aufgabenbereich Folgendes abdeckt: Verbindungsannahme, Empfang und Unmarshalling der Anfragen, Methodenaufruf per *invokeMethod()* von *VSRemoteObjectManager*, Marshalling und Versand der Antworten. *VSServer* übernimmt damit die Rolle eines generischen Server-Skeletons.

```
public class VSServer {
    public VSServer();
    public void init(int port);
    public void exportObject(Object object);
}
```

Mit Hilfe der *init()*-Methode lässt sich ein Server initialisieren und festlegen, auf welchem Port gelauscht werden soll. *exportObject()* besitzt die selbe Semantik wie in *VSRemoteObjectManager*.

Aufgabe:

→ Implementierung der Klasse *VSServer*

Hinweise:

- Die *exportObject()*-Methode von *VSServer* muss nichts anderes tun als *exportObject()* am *VSRemoteObjectManager* aufzurufen.
- Auch *VSServer* muss mit beliebigen Remote-Objekten zurecht kommen.

3.4 Clientseite (für alle)

Im Rahmen dieser Übungsaufgabe wird auf den Einsatz eines Namensdienstes verzichtet. Daher muss dessen grundlegende Funktionalität (d.h. das Auffinden von Remote-Objekten) nachgebildet werden. Dies erfolgt in der Klasse *VSClient* mittels der Methode *lookup()*:

```
public class VSClient {
    public VSClient();
    public void init();
    public Object lookup(String host, int port, Class interfaceClass);
}
```

lookup() erhält den Hostnamen des Servers sowie die Nummer des Ports auf dem dieser neue Verbindungen annimmt. Zusätzlich wird mit dem Parameter *interfaceClass* festgelegt welches Interface von dem Remote-Objekt benötigt wird. Anhand dieser Information lässt sich beim Server eine Remote-Referenz auf ein passendes Objekt anfordern. Sobald diese beim Client eintrifft, kann mit ihrer Hilfe ein passender Proxy für das Remote-Objekt erzeugt und als Rückgabewert von *lookup()* weitergereicht werden.
Der Proxy muss so implementiert sein, dass er: den lokalen Funktionsaufruf abfängt, eine Verbindung zum Server öffnet, eine passende Anfrage generiert (Marshalling der Aufrufparameter) und sendet, die Antwort empfängt und verwertet (Unmarshalling des Rückgabewerts) und am Ende die Verbindung schließt.

Aufgabe:

→ Implementierung der Klasse *VSClient*

Hinweise:

- Die Generierung von Proxies soll mittels *java.lang.reflect.Proxy.newProxyInstance()* realisiert werden. Die dafür benötigte Implementierung der Schnittstelle *java.lang.reflect.InvocationHandler* ist (analog zur Serverseite) so zu implementieren, dass sie mit beliebigen Remote-Objekten funktioniert.

3.5 Testen der Implementierung

Die Implementierung muss den Test *VSProxyTest.java* erfolgreich bestehen: Dieser besteht im Wesentlichen aus zwei Einzeltestfällen:

- *VSRemoteObjectManager*-Test: Dieser Testfall überprüft, ob auf ein exportiertes Objekt (serverseitig lokal) mittels *invokeMethod()* erfolgreich zugegriffen werden kann.
- Fernaufruf-Test: Dieser Testfall überprüft, ob Fernaufrufe zwischen einem Client und einem Server funktionieren.

Alle für die Tests notwendigen Dateien liegen unter */proj/i4vs/pub/a3* und müssen in das bereits bestehende Subpackage *vsue.tests* integriert werden.

3.6 Abgabe: bis 3.6.2008, 12:00 Uhr

Die für diese Teilaufgabe erstellten Dateien sind in einem Subpackage *vsue.proxy* zusammenzufassen.