Praktikum

Datenbanken und verteilte Systeme
Sommersemester 2015

Tutorial

1 Datenbankzugriff für die Server-Anwendung

Der Zugriff auf eine Datenbank kann über vielerlei Wege erfolgen, z.B. mittels existierender Programme wie dem *Squirrel SQL-Client*, aber auch mittels selbst erstellter Anwendungen. Beide Wege sollen im Folgenden beschritten werden.

1.1 Squirrel SQL-Client

Auf den VSIS-Poolrechnern ist der Squirrel SQL-Client inkl. des benötigten MySQL-Treibers bereits installiert. Zum Starten nutzen Sie das entsprechende Icon auf dem Desktop oder rufen Sie die Datei „squirrel_sql.bat" im Verzeichnis „D:\Programme\Squirrel-3.3.0" (o.ä.) auf. Möchten Sie von ihrem eigenen Notebook auf die Datenbank zugreifen, finden Sie im Anhang eine entsprechende Anleitung zur Einrichtung.

1.1.1

Für die Verbindung zur Datenbank benötigen Sie einen *Alias*. Dabei handelt es sich um eine Instanz einer Treiberkonfiguration, welche die Daten für eine Verbindung zu einer bestimmten Datenbank auf einem bestimmten Server enthält. Ein neues Alias lässt sich in der Alias-Sicht anlegen. Aus dem Menü *Windows* wählen Sie *View Aliases* und klicken im linken Menü auf das kleine Pluszeichen. In dem sich öffnenden Fenster *Add Alias* Geben Sie folgende Daten an:

Name:	Duvs
Driver:	MySQL Driver
URL:	jdbc:mysql://vsisls4/duvs**000** (Gruppenkennung)
	User Name: duvs**000**
	Password: Das Gruppenpasswort (Betreuer fragen)

Mit dem Button *Test* und anschließend *Connect* finden Sie heraus, ob die Verbindung erfolgreich war. Abschließend klicken Sie auf *OK*, um das Alias zu speichern.

1.1.2

Die Datenbank kann nun verwendet werden. Klicken Sie mit der rechten Maustaste auf das neue Alias und wählen Sie *Connect*.

Legen Sie nun mittels eines SQL-Statements (CREATE TABLE) eine neue Tabelle mit dem Namen „benutzer" an. Die Tabelle soll über folgende Attribute verfügen:

```
Kennung, Vorname, Nachname, Passwort und E-Mail-Adresse
```

Verwenden Sie dabei geeignete Datentypen für die Felder und vergeben Sie einen Primärschlüssel.

1.1.3

Fügen Sie mittels eines SQL-Statements (INSERT INTO) die Mitglieder Ihres Teams als Benutzer in die soeben erstellte Tabelle ein.

1.1.4

Lassen Sie sich mit einer SQL-Abfrage (SELECT) alle Tupel der Tabelle „benutzer" ausgeben. Haben Sie dies erfolgreich gemeistert, soll in einem nächsten Schritt der Zugriff auf die Datenbank von einem eigenen Java-Client aus erfolgen.

1.2 Java-Client mit JDBC unter Eclipse

1.2.1

Starten Sie Eclipse und erzeugen Sie unter *File->New->Java Project* ein neues Projekt. Wählen Sie einen geeigneten Projektnamen (etwa „duvs000").

1.2.2
Um auf die MySQL-Datenbank zugreifen zu können, benötigen Sie eine entsprechende Java-Bibliothek, die Ihnen den Zugriff erleichtert. Auf der Praktikumsseite finden Sie einen Link zur entsprechenden Bibliothek.

Die Archivdatei *mysql-connector-java-<version>-bin.jar* speichern Sie idealerweise in ihrem soeben neu angelegten Eclipse-Projekt. Über das Kontextmenü aktualisieren Sie die Projektansicht (*Refresh*), öffnen dann die Projekteigenschaften (*Properties*) und fügen unter dem Menüpunkt *Java Build Path → Libraries* das Archiv hinzu.

1.2.3
Erzeugen Sie unter *File->New->Class* eine neue Java-Klasse im Ordner `src`. Wählen Sie „JDBCTest" als Namen der neuen Klasse und lassen Sie sich eine Main-Methode generieren.

1.2.4
Zunächst muss eine Verbindung mit der Datenbank hergestellt werden. Dies geschieht durch das Laden des Datenbanktreibers und der Angabe der Serverdaten (s.u.). Importieren Sie dazu in der Klasse „JDBCTest" das Paket `java.sql.*` und fügen Sie den folgenden Code in die Main-Methode ein. Sorgen Sie zudem für eine angemessene Fehlerbehandlung.

```
Class.forName("com.mysql.jdbc.Driver");
String url= "jdbc:mysql:vsisls4.informatik.uni-hamburg.de/duvs000";
String username ="duvs000";
String password ="XXX";
Connection con = DriverManager.getConnection(url, username, password);
```

1.2.5
Testen Sie die Datenbankverbindung durch eine einfache Abfrage. Lassen Sie sich dazu aus der Datenbank alle Tupel aus der Tabelle „benutzer" auf der Konsole ausgeben. Sie können dafür zum Beispiel wie folgt vorgehen:

```
String sql = "SELECT * FROM benutzer";
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(sql);
while (rs.next()) {
      String vorname = rs.getString("Vorname");
      String nachname = rs.getString("Nachname");
      System.out.println(vorname + " " + nachname);
}
```

1.2.6
Nicht vergessen(!): Ist der Zugriff auf die Datenbank abgeschlossen, wird die Verbindung zur Datenbank beendet.

```
con.close();
```

1.2.7
Wählen Sie *Run -> Run As -> Java Application*, um die Anwendung zu testen.

2 Entwicklung einfacher Client-Server-Anwendungen

Das *Client-Server-Modell* ist das Standardkonzept für die Verteilung von Aufgaben innerhalb eines Netzwerks. Ein *Server* ist dabei ein Programm, das eine bestimmte Software-Funktionalität bereitstellt. Im Rahmen des Client-Server-Konzepts können andere Programme, die *Clients*, diese Funktionalität nutzen. Dabei können sich Clients und Server durchaus auf unterschiedlichen Rechnern befinden – sie müssen es aber nicht unbedingt! Das Programmieren und Testen von einfachen Client-Server-Anwendungen kann daher auch relativ komfortabel auf einem einzigen Rechner erfolgen und nach Fertigstellung auf die Zielrechner verteilt werden. Dieses Tutorial zeigt ein einfaches Beispiel für eine Client-Server-Beziehung zwischen zwei Java-Anwendungen mittels *Sockets*.

2.1 Entwicklung der Server-Anwendung

2.1.1

Starten Sie Eclipse und erzeugen Sie unter *File → New → Project → Java → Java Project* ein neues Projekt. Wählen Sie einen geeigneten Projektnamen, z.B. „ServerProjekt".

2.1.2

Erstellen Sie in dem Projekt eine neue Klasse mit Main-Methode für den Start der Server-Anwendung.

2.1.3

Erstellen Sie (z.B. in Ihrer Main-Methode) ein neues Exemplar der von Java bereitgestellten Klasse `ServerSocket`, um eingehende Kommunikationsverbindungen auf einem bestimmten Port ihres Rechners ihrer neuen Anwendung anzunehmen. Wählen Sie dazu einen beliebigen (freien) Port (z.B. 6666):

```
ServerSocket serverSocket = new ServerSocket(6666);
```

Nun muss der Server dazu gebracht werden, auf eingehende Verbindungen zu warten, was über den Aufruf der Methode `accept()` geschieht. Dabei wird die Verbindung zum Client, welcher den Server aufruft, als `Socket`-Objekt zurückgegeben. Dieser Socket stellt sozusagen eine Repräsentation der Verbindung zum (entfernten) Client dar und ist notwendig, um auf die vom Client übertragenen Daten zugreifen zu können und dem Client eine Antwort zu schicken:

```
Socket socket = serverSocket.accept();
```

Wenn eine eingehende Verbindung ankommt, läuft der Programmfluss weiter und die eingehenden Daten können verarbeitet werden. Da die Daten jedoch als *Streams* (d.h. als geordnete Folge von Bytes mit zumeist unbekannter Länge) transportiert werden, müssen diese zunächst durch geeignete *Reader* bzw. *Writer*-Klassen gelesen bzw. geschrieben werden. Im Anschluss können die eingehenden Daten verarbeitet und ggf. eine Antwort an den Client generiert werden. Diese wird dann wiederum als Stream zurückgesendet und nach Abschluss der gesamten Kommunikation werden alle *Reader* und *Writer* sowie der *Socket* geschlossen:

```
BufferedReader in = new BufferedReader(
                        new InputStreamReader(socket.getInputStream())
                    );

String request = in.readLine(); // Eine Zeile verarbeiten


// Beginn der anwendungsspezifischen Verarbeitung der Client-Anfrage
String response = "Hello " + request;
// Ende der Verarbeitung

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println(response);
out.close();
in.close();
socket.close();
```

Soll der Server irgendwann beendet werden, muss zudem der ServerSocket geschlossen werden:

```
serverSocket.close();
```

Da bei der Kommunikation im Allgemeinen vielfältige Fehler auftreten können (z.B. falls der angegeben Port belegt ist), muss darüber hinaus für eine angemessene Fehlerbehandlung gesorgt werden (hier nicht gezeigt).

2.2 Entwicklung der Client-Anwendung

2.2.1

Erzeugen Sie unter „File → New → Project → Java → Java Project" ein neues Projekt. Wählen Sie einen geeigneten Projektnamen, z.B. „ClientProjekt".

2.2.2

Erstellen Sie in dem Projekt eine neue Klasse mit Main-Methode für den Start der Client-Anwendung.

2.2.3

Erstellen Sie (z.B. in Ihrer Main-Methode) eine neue Instanz der von Java bereitgestellten Klasse `Socket`, um eine Kommunikationsverbindung zu ihrer Server-Anwendung zu erstellen. Hierzu muss die Adresse des Servers und die Port-Nummer angegeben werden, unter welcher Ihre Server-Anwendung erreichbar ist. Falls Client und Server beide auf einem Rechner laufen, kann anstelle der konkreten Adresse auch `"localhost"` angegeben werden:

```
Socket socket = new Socket("localhost",6666);
```

2.2.4

Implementieren Sie nun über den Socket eine einfache Anfrage an den Server und geben Sie dessen Antwort auf der Konsole aus. Sie können dazu die im Server-Teil des Tutorials dargestellte Stream-Verarbeitung in umgekehrter Reihenfolge verwenden: zuerst den *Request* über den *OutputStream* senden und anschließend die *Response* über den *InputStream* empfangen.

2.2.5
Testen Sie Ihre Anwendung. Wählen Sie hierzu unter Eclipse im Kontextmenü Ihrer Server-Klasse *Run as → Java Application*, um zunächst den Server zu starten. Starten Sie dann auf die gleiche Weise den Client, um einen Test-Aufruf durchzuführen.

Vergessen Sie nicht, Ihren Server zu beenden, bevor Sie die Server-Anwendung erneut starten, da ansonsten der Port noch belegt ist!

2.3 Verarbeitung der Anfragen von mehreren Clients

Wie Ihnen vielleicht aufgefallen ist, kann ihr Server bisher nur eine einzige Anfrage von einem einzelnen Client verarbeiten. Um auch mehrere Anfragen gleichzeitig entgegen nehmen zu können, müssen Sie diese in jeweils eigenen leichtgewichtigen Prozessen verarbeiten. Hierfür stellt Java das Konzept der *Threads* bereit.

2.3.1
Fügen Sie Ihrem Server-Projekt eine neue Klasse namens „RequestHandler" hinzu, welche von der von Java bereitgestellten Klasse `Thread` erbt. Von dieser Klasse soll nun für jeden eingehenden Aufruf eines Clients eine neue Instanz erstellt werden, welche sich um die Bearbeitung genau dieser Anfrage kümmert. Durch die Verwendung eines so entstehenden neuen Threads können Anfragen mehrerer Clients dann *nebenläufig* bearbeitet werden.

Ein Thread muss stets die Methode `public void run()` implementieren, welche den nebenläufig auszuführenden Code enthält und beim Start des Threads automatisch aufgerufen wird. In diesem Fall können Sie hier als Verarbeitungslogik die im Server-Teil des Tutorials dargestellte Stream-Verarbeitung durchführen. Damit alle hierfür benötigten Informationen zur Verfügung stehen, übergeben wir das vom Server beim Eintreffen einer neuen Anfrage erzeugte *Socket* an den Konstruktor des *RequestHandlers*:

```
public class RequestHandler extends Thread {

	private Socket socket = null;

	public RequestHandler(Socket socket) {
		super("RequestHandler");	// Name des Threads
		this.socket = socket;
	}

	public void run() {
		// Nebenläufig auszuführende Verarbeitungslogik
	}
}
```

2.3.2

Nun muss beim Eintreffen einer neuen Verbindung nur noch eine neue Instanz des `RequestHandlers` erzeugt und der Thread gestartet werden. Dies geschieht über die Methode `start()`, welche wir zum Beispiel aus einer (Endlos-)Schleife aufrufen können, welche permanent auf neue eingehende Verbindungen lauscht. Die Implementierung für die Server-Klasse unserer Server-Anwendung reduziert sich damit (ohne Beachtung der Fehlerbehandlungsroutinen) auf den folgenden Code:

```
ServerSocket serverSocket = new ServerSocket(6666);

boolean listening = true;
while (listening) {
       new RequestHandler(serverSocket.accept()).start();
}

serverSocket.close();
```

2.3.3

Testen Sie Ihre neue Server-Anwendung mit einem oder mehreren Clients. Sie können auch versuchen, die Anwendung auf mehrere Rechner zu verteilen.

2.4 Integration des Datenbank-Zugriffs

Erweitern Sie Ihre Client-Server-Anwendung nun so, dass der Server nach der Übermittlung einer korrekten Kennung mit Passwort eine Liste der Namen (Vorname und Nachname) der in der Datenbank eingetragenen Benutzer zurücksendet (vgl. Kapitel 4). Verwenden Sie bei der Datenbank-Abfrage für das Prüfen der Kennung ein `PreparedStatement`, um Ihre Anwendung vor böswilligen Angriffen zu schützen. Die Übertragung zwischen Client und Server können Sie als einfache Zeichenkette realisieren, wobei Kennung und Passwort jeweils einzeln in einer Zeile stehen und auch bei der zurückzugebenen Liste der Namen jeweils ein Name pro Zeile übertragen wird.

Machen Sie sich dabei auch Gedanken, wie auf Fehlerfälle (z.B. falsches Passwort, Verbindung zur Datenbank gestört) geeignet reagiert werden kann und berücksichtigen Sie dies bei Ihrer Implementierung.