UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Aufgabenblatt 1: Allgemeines

(Zu bearbeiten bis 11.11.2019)

Client/Server-Programmierung Wintersemester 2019/20

Im Praktikum zur Vorlesung Client/Server-Programmierung wird eine extrem vereinfachte Börsenanwendung unter Zuhilfenahme verschiedener Middleware-Technologien realisiert. Die Anwendung erlaubt die Abfrage von Information zu Börsenkursen sowie die Verwaltung eines Aktiendepots. Die Börsenkurse sind in einer zentralen, schreibgeschützten MySQLDatenbank abgelegt.

Die serverseitige Anwendung besteht aus zwei vorgegebenen Java-Klassen: Die Klasse `StockExchange` stellt die folgenden Dienste für einen Kunden bereit:

1) **public String[] getAGs()**

   Gibt eine Liste aller AG-Namen in einem Array von Zeichenketten zurück. Im Fall eines Fehlers (z.B. bei der Verbindung mit der Datenbank) gibt die Methode der Einfachheit halber ein leeres Array zurück.

2) **public double getMarketValue(int day, String ag)**

   Gibt den Kurs der Aktie mit Namen `ag` am Tag `day` (1...31) zurück. Bei Eingabe eines ungültigen Tages und/oder AG-Namens wird `-1.0` zurückgegeben.

3) **public Depot openDepot()**

   Eröffnet ein Aktiendepot, das initial eine Bargeldeinlage von 10.000,00 EUR besitzt. Von diesem Betrag können anschließend Aktien gekauft werden. Bei Verkauf wird der Gewinn wieder der Bargeldeinlage gutgeschrieben. Das Ergebnis der Methode ist eine Referenz auf das Depot.

Die Klasse `Depot` stellt dem Anwender folgende Dienste bereit:

1) **public boolean buy(String ag, int amount, int day)**

   Kaufen einer Menge von Aktien einer AG zum aktuellen Kurs des gegebenen. Tags. Bei erfolgreicher Transaktion wird `true` zurückgegeben, ansonsten `false`.

2) **public boolean sell(String ag, int amount, int day)**

   Verkaufen einer Menge M von Aktien einer AG zum aktuellen Kurs des gegebenen Tags. Bei erfolgreicher Transaktion wird `true` zurückgegeben, ansonsten `false`.

3) **public int stockAmount(String ag)**

   Gibt die Anzahl der Aktien der gegebenen AG zurück, die man in seinem Depot besitzt.

4) **public double cashBalance()**

   Gibt das aktuelle Barguthaben des Depots aus.

5) **List<DepotEntry> getStatement(int day)**

   Gibt einen Depotauszug aus, der aus einer Liste mit folgendem Inhalt besteht:

   - zunächst sind alle Aktien im Depot mit Name, Stückzahl und Marktwert am gegebenen Tag aufgelistet,
   - dann folgt ein Eintrag für das Barguthaben,
   - schließlich ein Eintrag mit dem Gesamtwert des Depots am gegebenen Tag (inkl. Barguthaben).

   Die Klasse `DepotEntry` speichert den Namen der Aktie (bzw. „Cash“ und „Total“ für die letzten beiden Einträge), die Anzahl der Aktien (bzw. 0) und den Wert.

Die zu Grunde liegende Datenbank enthält die zwei Tabellen `ag_name` und `ag_data`, die folgendermaßen aufgebaut sind:

```
ag_name:
AG_ID:   int,       ID der Aktiengesellschaft (Primärschlüssel)
AG_NAME: String,    Name der Aktiengesellschaft

ag_data:
ID:      int,       Primärschlüssel
AG_ID:   int,       ID der Aktiengesellschaft
DAY:     int,       Tag als Zahlenwert (1...31); wir gehen davon
                    aus, dass an jedem Tag Handel stattgefunden hat
VALUE:   double,    Aktueller Kurs der Aktie an diesem Tag
                    (0.01 bis 9999.99 EUR)
```

Der Zugriff auf die Datenbank erfolgt über eine vorgegebene Java-Klasse:

```
public class ConnectDB
{
    // Konstruktor, lädt den Treiber
    public ConnectDB();

    // öffnet eine Verbindung zur Datenbank
    public boolean open();

    // schliesst eine bestehende Verbindung zur Datenbank
    public boolean close();

    // führt den SQL-String auf die geöffnete DB aus und liefert die Rückgabe in
    // einem ResultSet; im Fehlerfall wird eine SQLException ausgeworfen
    public java.sql.ResultSet execute(String SQL) throws java.sql.SQLException;
}
```

Weitere Informationen, insbesondere zu den verwendeten SQL-Anweisungen, finden Sie im Anhang.

In der 1. Praktikumsaufgabe sollen Sie zunächst einen Client erstellen, der die gegebenen Klassen nutzt, und Client und Server anschließend mit Hilfe von RMI auf zwei Rechner verteilen. In den folgenden Übungen wird dieselbe Anwendung dann unter Verwendung anderer Middleware-Technologien realisiert und getestet:

2. Aufgabe: CORBA
3. Aufgabe: EJB und JSP
4. Aufgabe: Web-Services mit Axis2

Zu jeder Aufgabe wird ein gesondertes Aufgabenblatt mit Hinweisen und Vorgehensweisen der jeweiligen Technologie bereitgestellt.

Hinweise zur Abgabe

Wenn Sie Aufgaben zu Hause lösen, sollten Sie sie zum Abgabetermin entweder auf USB-Stick mitnehmen oder vorab per SFTP auf den Dateiserver laden (siehe die nachfolgenden Hinweise dazu). Beachten Sie aber bitte, dass Sie in der Lage sein sollten, Ihr Programm **im Labor H-A 4111 im Betrieb zu zeigen** (entweder auf den verfügbaren Rechnern oder Ihrem eigenen Laptop).

Hinweise zu Linux

Eine Übersicht über die wichtigsten Linux-Kommandos können Sie unserem Info-Blatt entnehmen, das Sie in den Materialien zum Aufgabenblatt finden oder unter
http://www.bs.informatik.uni-siegen.de/www/lehre/material/csp/Infoblatt.pdf
herunterladen können.

Hinweise zu SSH

Sie können von zu Hause aus Dateien per SFTP auf den Dateiserver des Labors H-A 4111 hochladen. Verbinden Sie Ihren Client dazu mit `bsgate1.bs.informatik.uni-siegen.de` unter Verwendung Ihres Benutzernamens (*cspXXX*). Dabei ist die Authentifizierung allerdings **nicht** mit Passwort, sondern lediglich über das *Public Key* Verfahren möglich.

Falls Sie zu Hause eine Linux-Umgebung haben, müssen Sie einmalig folgende Schritte ausführen, um sich per SFTP verbinden zu können:

- Prüfen Sie zunächst, ob auf Ihrem Rechner zu Hause bereits eine Datei `~/.ssh/id_rsa.pub` existiert, die Ihren öffentlichen SSH-Schlüssel enthält. Falls ja, können (und sollten) Sie den nächsten Schritt überspringen.
- Erzeugen Sie sich ein SSH-Schlüsselpaar mit `ssh-keygen -b 4096 -t rsa`. Bei der ersten Frage („`Enter file in which to save the key`“) drücken Sie einfach `<Return>`. Anschließend werden Sie aufgefordert, eine Passphrase einzugeben. Wählen Sie hier ein sicheres Passwort.
- Kopieren Sie sich dann die Datei `~/.ssh/id_rsa.pub` auf einen USB-Stick.
- Loggen Sie sich dann im Labor H-A 4111 auf einen Rechner ein.
- Erzeugen Sie (falls noch nicht vorhanden) dort das Verzeichnis `~/.ssh` mit `mkdir ~/.ssh`.
- Fügen Sie anschließend die Schlüsseldatei (für Linux: `id_rsa.pub`) auf Ihrem USB-Stick an die Datei `~/.ssh/authorized_keys` an: `cat id_rsa.pub >> ~/.ssh/authorized_keys`
- Sie können sich nun von zu Hause aus per SFTP mit dem Dateiserver verbinden.

Falls Sie Windows verwenden, hängt die genaue Vorgehensweise von Ihrer SFTP Client-Software ab. Wie oben, müssen Sie sich aber zunächst zu Hause ein Schlüsselpaar erzeugen. Dann exportieren Sie den *öffentlichen* Schlüssel im OpenSSH-Format. Diesen Schlüssel fügen Sie dann im Labor an die Datei `~/.ssh/authorized_keys` an. Falls Ihr SFTP-Client den Schlüssel nicht im OpenSSH-Format exportieren kann, können Sie im Labor auch das Kommando `ssh-keygen -i` nutzen, um das Format umzuwandeln.

Hinweise zu SQL

SQL ist eine deklarative Abfragesprache für relationale Datenbanken. SQL (noch häufig irrtümlich erklärt als Abkürzung von „Structured Query Language“) ist aus SEQUEL (Structured English Query Language) hervorgegangen, das von IBM entworfen wurde. Sie hat eine relativ einfache Syntax, die an die englische Umgangssprache angelehnt ist, und stellt eine Reihe von Befehlen zur Definition von Datenstrukturen nach der relationalen Algebra, zur Manipulation von Datenbeständen (Anfügen, Bearbeiten und Löschen von Datensätzen) und zur Abfrage von Daten zur Verfügung. 1986 wurde der erste SQL-Standard vom ANSI verabschiedet, welcher dann 1987 von der ISO ratifiziert wurde [http://de.wikipedia.org/wiki/SQL]. Der Befehl `SELECT` ist der mächtigste Befehl in SQL. Die Grundsyntax in MySQL lautet:

```
SELECT [DISTINCT | ALL] select_expression,... FROM tables ...
[WHERE where_definition]
[GROUP BY feld_name,...]
[ORDER BY feld_name [ASC | DESC] ,...]
[LIMIT [offset,] rows]
```

Wir beschränken uns lediglich auf einen Teil der Syntax, der für die Erfüllung der Aufgabe erforderlich ist. Alle folgenden Beispiele beziehen sich bereits auf die zur Verfügung gestellte MySQL-Datenbank mit den Tabellen `ag_name` und `ag_data`. Die SQL-Requests werden als Zeichenkette an das Datenbanksystem gesendet. In Java dient typischerweise JDBC als Schnittstelle zur Datenbank. Im Praktikum können Sie am Anfang auch die Klasse `ConnectDB` (speziell die Methode `execute(String sql)`) zur Kommunikation mit der Datenbank verwenden.

Die kürzestmöglichen `SELECT`-Anweisungen für unsere Datenbank lauten:

```
SELECT * FROM ag_name
SELECT * FROM ag_data
```

Diese geben alle Daten von allen Spalten der jeweiligen Tabelle zurück. Sind nur wenige Spalten von Interesse, so kann man statt des *Wildcard*-Operators (`*`) die jeweilige Spalte angeben:

```
SELECT DAY, VALUE FROM ag_data
```

Hier werden alle Inhalte der Spalten `DAY` und `VALUE` von `ag_data` zurück gegeben. Soll eine Filterung über die Einträge statt finden, so geschieht das mit der `WHERE` Klausel gefolgt von einem Boole'schen Wert:

```
SELECT VALUE FROM ag_data WHERE DAY = '3'
```

Diese Anfrage gibt alle Inhalte der Spalte `VALUE` von `ag_data` zurück, die vom Tag 3 stammen. Beachten Sie die Verwendung der Hochkommata! Ist der Tag variabel, so kann er über eine Java String-Konkatenation eingefügt werden:

```
int x = 3;
db.execute("SELECT VALUE FROM ag_data WHERE DAY = '" + x + "'");
```

Der Boole'sche Ausdruck in der `WHERE` Klausel ist natürlich mit Hilfe von Boole'schen Operatoren erweiterbar:

```
SELECT VALUE FROM ag_data WHERE DAY = '4' OR DAY = '5'
```

Sollen die Daten sortiert ausgegeben werden, so wird der `ORDER BY` Operator verwendet. Mit `ASC` oder `DESC` kann man angeben, ob auf- oder absteigend sortiert werden soll:

```
SELECT DAY, VALUE FROM ag_data ORDER BY DAY ASC
```

Das oben angegebene Beispiel holt die Spalten `DAY` und `VALUE` aus der Tabelle `ag_data` und sortiert die Einträge aufsteigend nach dem Wert des Tages. Es kann auch eine mehrfache Sortierung erfolgen mit

```
SELECT DAY, VALUE FROM ag_data ORDER BY DAY ASC, VALUE ASC
```

wobei zuerst nach `DAY` und innerhalb des Tages jeweils aufsteigend nach `VALUE` sortiert wird.

Oftmals sollen mit einer Anfrage zusammengehörige Informationen aus verschiedenen Datenbanktabellen geholt werden. Z.B. könnten Sie auf den Namen einer AG und den jeweiligen Kurs zugreifen wollen, die jedoch in zwei verschiedenen Tabellen gespeichert sind. Mit der Anfrage

```
SELECT AG_NAME, DAY, VALUE FROM ag_data, ag_name
WHERE ag_data.AG_ID = ag_name.AG_ID
```

erhalten Sie die Namen der AGs zusammen mit den Aktienkursen an den jeweiligen Tagen. Dabei werden die beiden Tabellen über den Primärschlüssel `AG_ID` verknüpft. Dies bezeichnet man als „*natural join*“. In MySQL kann man diese Anfrage auch in einer speziellen Form ausdrücken, die ggf. effizienter ausgewertet werden kann:

```
SELECT AG_NAME, DAY, VALUE FROM ag_data NATURAL JOIN ag_name
```

Ein interessantes Merkmal von SQL ist, dass man in einer Anfrage bereits auch einfache Verarbeitungsschritte angeben kann, um Ergebnisse zusammenzufassen. Z.B. kann die Summe aller Aktienwerte eines Tages mit der folgenden Anfrage ermittelt werden:

```
SELECT SUM(VALUE) FROM ag_data WHERE DAY = '1'
```

Unabhängig davon, welches `SELECT` Statement Sie absetzen, erhalten Sie in Java (unter Verwendung von JDBC oder der Klasse `ConnectDB`) in jedem Fall ein Objekt der Klasse `java.sql.ResultSet` mit dem Ergebnis der Abfrage zurück. Das Ergebnis wird im `ResultSet` zeilenweise gespeichert und muss dementsprechend ausgewertet werden. Dabei müssen Sie für jede Zeile die einzelnen Spaltenwerte entsprechend ihrem Datentyp mit der passenden `get`-Methode auslesen:

```
double aktienIndex = 0.0;
double maxKursWert = 0.0;
String maxKursName;
ResultSet res;
try {
    res = db.execute("SELECT AG_NAME, VALUE FROM ag_data " +
                     "NATURAL JOIN ag_name WHERE DAY = '3'");
    while (res.next()) {
        aktienIndex += res.getDouble(2);
        if (maxKursWert < res.getDouble(2)) {
            maxKursWert = res.getDouble(2);
            maxKursName = res.getString(1);
        }
   }
}
catch(Exception e) {
    System.err.println("Fehler beim Datenbankzugriff!");
}
```

Das obige Beispiel summiert die Kurse aller Aktien am Tag 3 in die Variable `aktienIndex` und speichert den Namen der AG mit dem höchsten Kurs (`maxKursName`) sowie den Wert des Kurses (`maxKursWert`). Das Argument bei `getDouble()` und `getString()` gibt an, welche Spalte verwendet werden soll. Die Zahl bezieht sich dabei auf die Position im `SELECT`-Statement, im Beispiel also 1 für `AG_NAME` und 2 für `VALUE`. Alternativ kann auch der Spaltenname verwendet werden (z.B. `res.getDouble("VALUE")`). Weitere Methoden von `ResultSet` sind in der Java-Dokumentation unter `http://docs.oracle.com/javase/8/docs/api` beschrieben.