JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

Prepared Statements

Informationssysteme
4CHITT 2015/16

Moritz Mühlehner, Lukas Sträßler

Version 1.0

Note:

Betreuer: Borko Michael

Begonnen am 18. Februar 2016

Beendet am 25. Februar 2016

Inhaltsverzeichnis

1 Einführung

PreparedStatements sind in JDBC eine Möglichkeit SQL-Befehle vorzubereiten um SQL-Injections zu vermeiden. Die Typüberprüfung kann somit schon bei der Hochsprache abgehandelt werden und kann so das DBMS entlasten und Fehler in der Businesslogic behandelbar machen.

1.1 Ziele

Es ist erwünscht Konfigurationen nicht direkt im Sourcecode zu speichern, daher sollen Property-Files [3] zur Anwendung kommen bzw. CLI-Argumente (Library verwenden) [1,4] verwendet werden. Dabei können natürlich Default-Werte im Code abgelegt werden.
Das Hauptaugenmerk in diesem Beispiel liegt auf der Verwendung von PreparedStatements [2]. Dabei sollen alle CRUD-Aktionen durchgeführt werden.

1.2 Aufgabenstellung

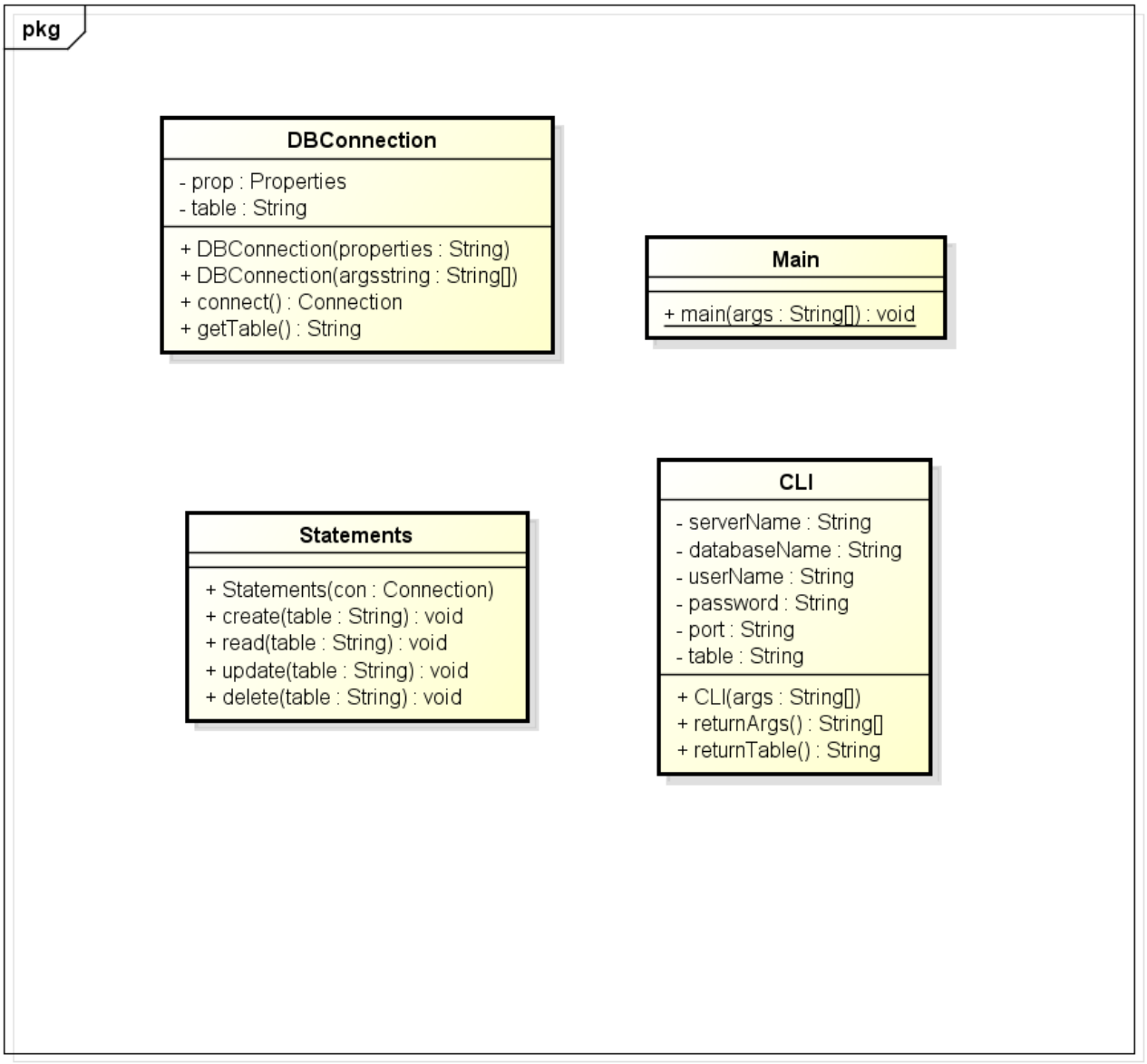
Verwenden Sie Ihren Code aus der Aufgabenstellung "Simple JDBC Connection" um Zugriff auf die Postgresql Datenbank "Schokofabrik" zur Verfügung zu stellen. Dabei sollen die Befehle (CRUD) auf die Datenbank mittels PreparedStatements ausgeführt werden. Verwenden Sie mindestens 10000 Datensätze bei Ihren SQL-Befehlen. Diese können natürlich sinnfrei mittels geeigneten Methoden in Java erstellt werden.
Die Properties sollen dabei folgende Keys beinhalten: host, port, database, user, password
Vergessen Sie nicht auf die Meta-Regeln (Dokumentation, Jar-File, etc.)! Die Testfälle sind dabei zu ignorieren. Diese Aufgabe ist als Gruppenarbeit (2 Personen) zu lösen.

1.3 Quellen

[1] Apache Commons CLI; Online:
http://commons.apache.org/proper/commons-cli/
[2] Java Tutorial JDBC "Prepared Statements"; Online:
https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html
[3] Java Tutorial Properties; Online:
https://docs.oracle.com/javase/tutorial/essential/environment/properties.html
[4] Overview of Java CLI Libraries; Online:
http://stackoverflow.com/questions/1200054/java-library-for-parsing-command-line-parameters

2 Ergebnisse

2.1 UML-Klassendiagramm (Designüberlegung)



2.2 Property-Files

Property Files werden verwendet um Programme leicht für externe Benutzer konfigurierbar zu machen. Die Properties werden in dem Format key=value gespeichert und sind daher für den Benutzer leicht veränderbar.
Um Properties in Java einzulesen und zu verwenden gibt es eine eigene Properties Klasse.
Dazu muss `java.util.Properties` importiert werden.

```
import java.util.Properties;
```

Die Properties Klasse beinhaltet mehrere wichtige Methoden wie z.B. `setProperty(„key“,“value“)`, `getProperty(„key“)`, `remove(„key“)`, `store(FileOutputStream, String comment)` und `load(InputStream)`.

Um Properties verwenden zu können muss ein Property Objekt erstellt werden.

```
private Properties prop;
prop = new Properties();
```

Der Name des Files wird beim Erstellen eines DBConnection Objekts an einen InputStream übergeben.

```
public DBConnection(String properties)
new DBConnection("dbconnection.properties");
private InputStream input;
input = DBConnection.class.getResourceAsStream(properties);
```

Mit diesem InputStream werden die Properties geladen. (`prop.load(input);`)

Mit `getProperty(„key“)` bekommt man den Value für den Key welcher als String Parameter angegeben wird. `prop.getProperty("server")`
Der Value ist ebenfalls ein String.

2.3 CLI

CLI-Arguments sind Werte, die vor dem Ausführen des Programmes angegeben werden. Es soll also möglich sein Servername, Database Name, Username, Password, Port und Table anzugeben.

Dafür erstellen wir ein `Options`-Objekt. Diesem fügen wir nun verschiedene Options hinzu. Der 1. Parameter steht dabei für einen single-character, welcher der Name von der Option ist. Der 2. Parameter, ein boolischer, beschreibt ob ein Argument nach dieser Option benötigt wird, oder nicht. Der 3. und letzte Parameter ist die eigene dokumentierte Beschreibung der Option.

```
Options options = new Options();//Erstellen des Options Objektes
options.addOption(„N“, true, „Server Name“);//Hinzufügen deiner
Option
```

Nun brauchen wir einen `CommandLineParser`. Dieser wandelt die Angaben in der CLI in einen String Array zurück und gibt ein `CommandLine`-Objekt zurück. Dieses brauchen wir, um später die Methoden `hasOption(String opt)` und `getOptionValue(String opt)` anwenden zu können. Mit `hasOption(String opt)` überprüfen wir, ob die, in den Parametern angegebene Option, ein Argument hat. Ist dieser Fall eingetreten, holen wir uns dieses Argument mit der Methode `getOptionValue(String opt)`. Diesen Wert speichern wir in ein Attribut, welches wir via einer getter-Methode dann verwenden können.

```
CommandLineParser parser = new DefaultParser();//Erzeugen eines
CommandLineParser Objektes
CommandLine line = parser.parse(Options opt, String[]
args);//Erzeugen eines CommandLine-Objektes. Der 1. Parameter sind
die Options die umgewandelt werden sollen. 2. Parameter, Liste mit
den Command Line Arguments.
```

```
if (line.hasOption(„N“))
{
serverName = line.getOptionValue(„N“);
}
```

2.4 Prepared Statements

Prepared Statements sind wie normale SQL-Anweisungen, nur das sie bereits vor dem Ausführen „existieren". Was hingegen Prepared Statements deutlich von normalen Anweisungen unterscheiden ist, dass statt konkreten Werten, Platzhalter in der Anweisung stehen. Mithilfe von Prepared Statements, kann man SQL-Injections verhindern. Das DBMS überprüft nämlich vor Ausführung der Anweisung, die Gültigkeit der Parameter.

Um ein Prepared Statements vorzubereiten, speichern wir zuerst den Befehl, welchem wir am Server ausführen wollen, in einen String:

```
String createPostgre = "INSERT INTO "+table+" VALUES(?);";
```

Table wird der Methode als Parameter übergeben. Interessanter ist das ? in dem Befehl. Es ist ein Platzhalter, der später eingesetzt wird.
Nun erstellen wir ein Prepared Statement:

```
PreparedStatement create;
Create = c.preparedStatement(createPostgre); //c ist die
Connection zur Datenbank, die im Konstruktor übergeben wird. Als
Parameter übergen wir den String, wo unser Befehl drinnen steht,
den wir ausführen wollen.
```

Nun müssen wir statt dem ?, noch einen Wert einsetzten. Dies machen wir mit folgendem Befehl:

```
create.setInt(1,x); //Der erste Parameter, steht für die Nummer
des ?, dass ersetzt werden soll. Hier 1, da es nur eines gibt.
Würde es mehrere geben, könnten wir hier zum Beispiel 2
einsetzten. x steht für einen beliebigen Integer Wert, der
eingesetzt werden kann. In unserem Fall der x-Wert einer for-
Schleife.
```

Nun führen wir das Prepared Statement auf dem Server aus:

```
create.execute();
```

Danach geben wir das Statement-Objekt noch frei:

```
create.close();
```

3 Aufwand

3.1 Geschätzter Aufwand

Der geschätzte Aufwand ist wie folgt aufgeschlüsselt:

Moritz Mühlehner:

Durchgeführte Arbeit	**Datum**	**tatsächlicher Aufwand**
Übungsdurchführung	19. – 24.02.2016	5 h
Protokoll	24.02.2016	2 h
Gesamt	**19. - 24.02.2016**	**7 h**

Lukas Sträßler:

Durchgeführte Arbeit	**Datum**	**tatsächlicher Aufwand**
Übungsdurchführung	19. – 24.02.2016	5 h
Protokoll	24.02.2016	2 h
Gesamt	**19. - 24.02.2016**	**7 h**

Gesamt:

Durchführender	**tatsächlicher Aufwand**
Lukas Sträßler	7 h
Moritz Mühlehner	7 h
Gesamt	**14 h**

3.2 Tatsächlicher Aufwand

Der tatsächliche Aufwand ist wie folgt aufgeschlüsselt:

Moritz Mühlehner:

Durchgeführte Arbeit	**Datum**	**tatsächlicher Aufwand**
Übungsdurchführung	19. – 25.02.2016	8 h
Protokoll	25.02.2016	1 h
Gesamt	**19 - 25.02.2016**	**9 h**

Lukas Sträßler:

Durchgeführte Arbeit	**Datum**	**tatsächlicher Aufwand**
Übungsdurchführung	19. – 25.02.2016	6 h
Protokoll	25.02.2016	1 h
Gesamt	**19 - 25.02.2016**	**7 h**

Gesamt:

Durchführender	**tatsächlicher Aufwand**
Lukas Sträßler	7 h
Moritz Mühlehner	9 h
Gesamt	**16 h**