Fortgeschrittene Programmiertechniken

Prof. Dr. J. Pauli, Dipl.-Inform. J. Hoefinghoff, J. Kapitza, M. Cherubim

Um große Datenmengen effektiv zu speichern und zu verwalten ist der Einsatz von Datenbanken unvermeidbar. Verwenden Sie für das Übungsblatt bitte die bereitgestellte Postgres Datenbank mit der bereits existierenden Tabelle products. Die Zugangsdaten zur Datenbank finden Sie im Anhang unter den weiteren Informationen. Die nötigen Bibliotheken zum benutzen der Datenbank haben Sie bereits durch die pom.xml heruntergeladen. Für diese Aufgabe ist eine Internetverbindung notwendig!

Aufgabe 1

Update

Da Ids nun automatisch vergeben werden, sollte der IdGenerator aus Blatt 2 nicht mehr verwendet werden! Verwendung führt auf der Seite der Datenbank zu Fehlern. Stellen Sie sicher, dass Sie die ID nicht selbstständig setzen.

Für die nächste Abgabe ändern Sie die Artifact ID in der pom.xml (Abb. 1) direkt unter der Gruppen ID von `u1u2` auf `u3u4`. Und benennen (optional) Ihr Projekt mit der in Eclipse zu Verfügung gestellten Rename (Bild Abb. 2) Funktion um. Testen sie mit **clean assembly:assembly** ob anschließend das zip mit dem Namen `GRUPPE-u3u4-abgabe.zip` erstellt wurde. Die Bilder sollen Ihnen eine Hilfestellung sein.

Fortgeschrittene Programmiertechniken

Prof. Dr. J. Pauli, Dipl.-Inform. J. Hoefinghoff, J. Kapitza, M. Cherubim

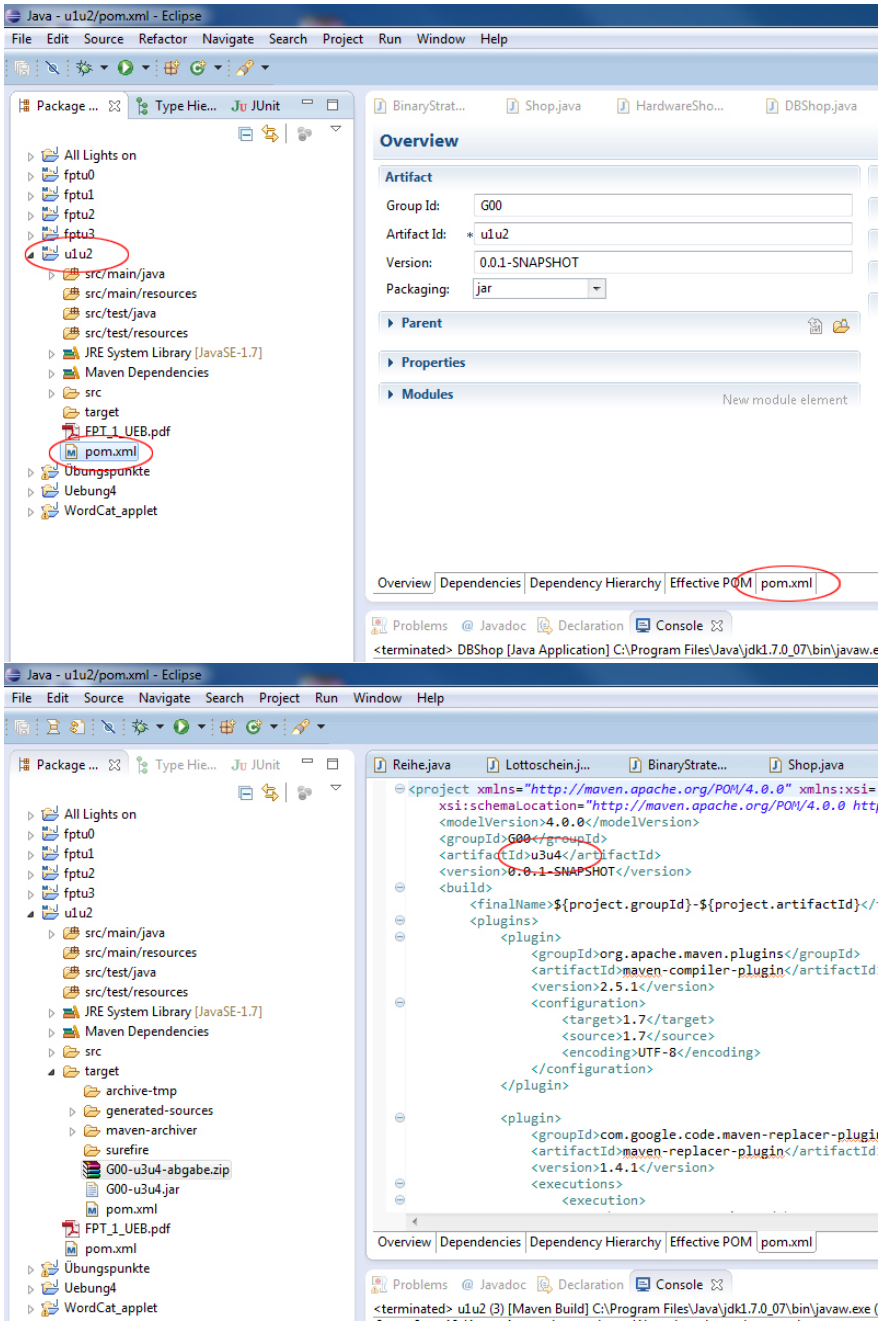


Abbildung 1: Maven pom.xml ändern

Fortgeschrittene Programmiertechniken

Prof. Dr. J. Pauli, Dipl.-Inform. J. Hoefinghoff, J. Kapitza, M. Cherubim

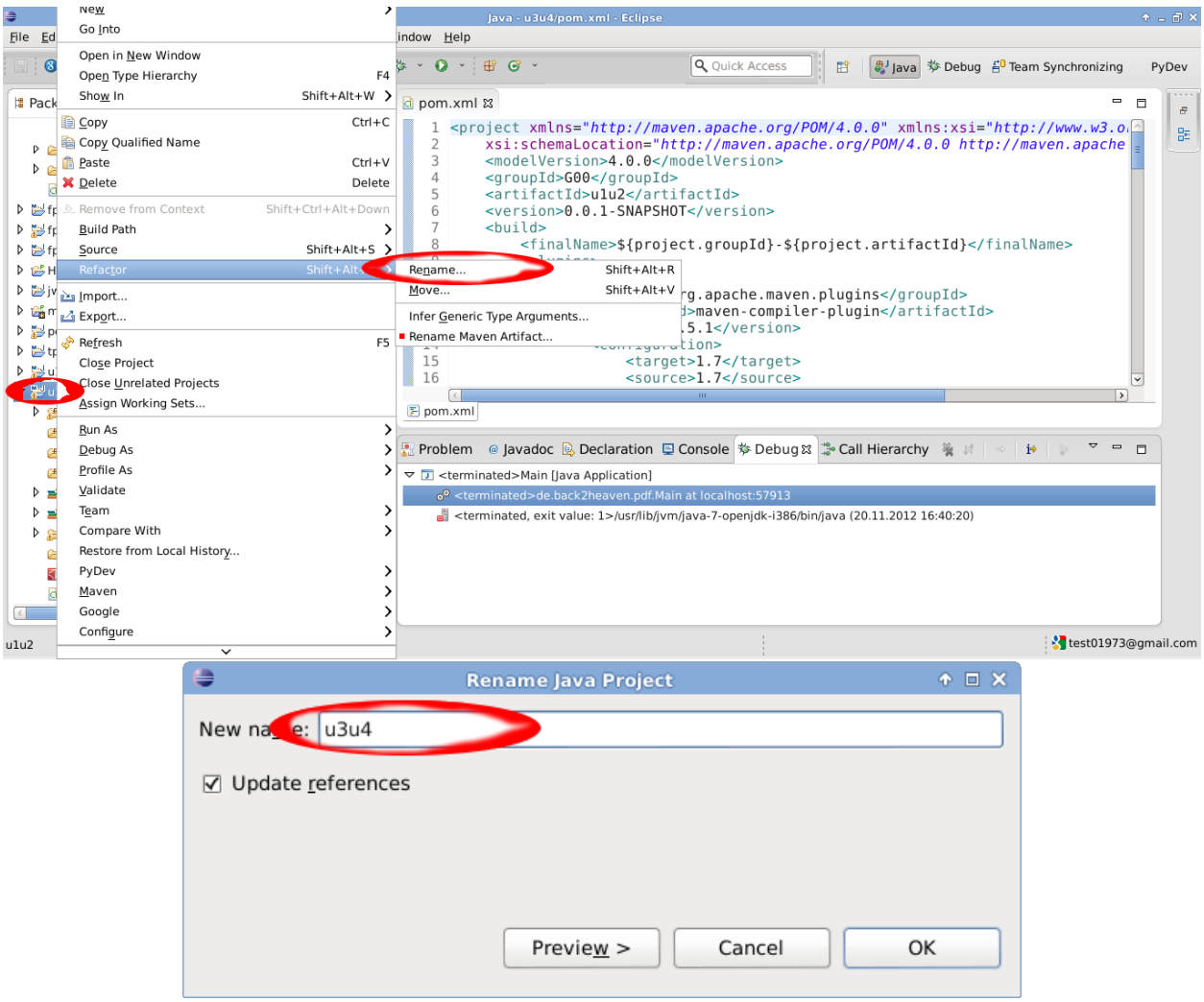


Abbildung 2: Eclipse rename

Aufgabe 2

JDBC

Um einen Einblick zu bekommen, wie mit JDBC und einer Datenbank gearbeitet wird, sollen Sie zunächst eine Verbindung zu einer bestehenden Datenbank herstellen. Schreiben Sie eine Klasse *JDBCConnector*, welche die einzelnen Schritte zur Datenbankabfrage mittels JDBC durchführt.

a) Lassen Sie sich mit Hilfe der Klasse *java.sql.DatabaseMetaData* Statusinformationen zur Verbindung auf der Konsole ausgeben.

- Lesen Sie die Url der Datenbank aus.
- Lesen Sie den Benutzernamen aus.
- Lassen Sie sich alle vorhandenen Tabellen der Datenbank anzeigen.

b) Erweitern Sie die Klasse *JDBCConnector* um drei Methoden:

- insert(String name, double price, int quantity): LONG (id)
  Achten Sie beim erstellen des Statements auf das Vorhandensein des Parameters `Statement.RETURN_GENERATED_KEYS`.
  INSERT INTO products(name,price,quantity) VALUES (?,?,?)
- insert(Product product): void
  Nach dem erfolgreichen Einfügen in die Datenbank soll das Produkt die von der Datenbank generierte Id übernehmen
- read(long productId): Product
  SELECT id,name,price,quantity FROM products WHERE id=?

Hinweise: Für das Absetzen von SQL-Befehlen nutzen Sie die Klasse *java.sql.PreparedStatement*.

Aufgabe 3

OpenJPA

Nachdem Sie Ihren Datenbestand mit JDBC in einer Datenbank persistent gemacht haben, sollten Sie nun OpenJPA für die gleiche Aufgabe verwenden. Dazu müssen Sie Ihre Produktklasse dahingehend anpassen, dass Ihre Klasse über beschreibende Annotationen verfügt, damit das Framework damit arbeiten kann. Beispiele sind Ihnen aus der Vorlesung bekannt oder auf der Projektseite zu finden. Ziel dieser Aufgabe ist es Produkte in eine Datenbank einzufügen und alle Produkte der Datenbank auszulesen. Sie sollten gemerkt haben, dass eine Lösung mit JDBC umständlich ist. Benutzen Sie für diese Aufgabe die gleiche Datenbank wie für Aufgabe 1.

Hinweis: Um OpenJPA verwenden zu können, sollten Sie in Ihrer Klasse set-Methoden hinzufügen.

a) Passen Sie Ihre Product-Klasse an, so dass diese dem verwendeten Schema der SQL Tabellen genügen.

b) Erstellen Sie eine Verbindung mit der Datenbank unter Verwendung einer Konfigurationsdatei.

c) Speichern und lesen Sie Ihre Objekte mit Hilfe von OpenJPA.

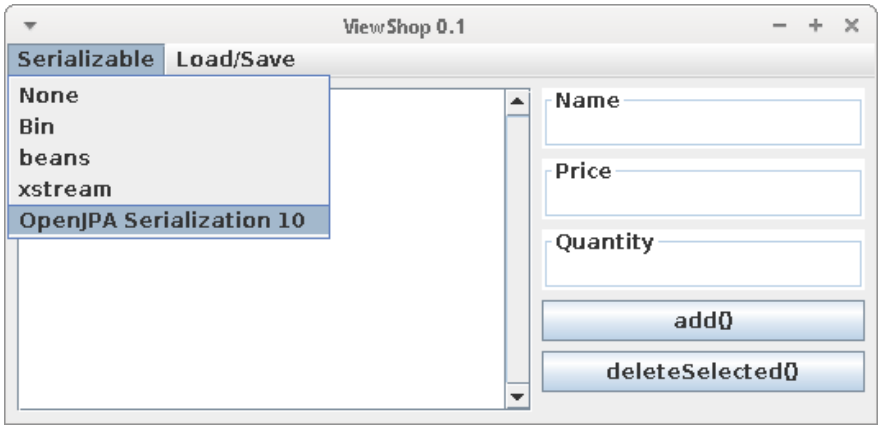
d) Welche Nachteile hat eine Konfigurationsdatei?

e) Erstellen Sie eine Verbindung mit der Datenbank, ohne eine Konfigurationsdatei zur Hilfe zu nehmen.

Aufgabe 4

Integration

Ermöglichen Sie dem Nutzer eine der Strategien (JDBC, OpenJPA) über ein Menü oder ähnlichem auszuwählen (Vgl. Blatt 2) und verwenden zu können. Die Anwendung soll die Datenspeicherung und das Datenlesen dann mit einer der beiden Datenbankstrategien durchführen. Sie müssen das Strategie-Muster (Interfaces aus dem Übungsblatt 2) weiterverwenden. Denken Sie daran, dass viele Daten in der Datenbank stehen können und begrenzen Sie ihre Anfrage. (Hilfe auf nächster Seite beachten)



Weitere Informationen:

```
URL : jdbc:postgresql://java.is.uni-due.de/ws1011
Treiber: org.postgresql.Driver
Option für OpenJPA:
  openjpa.RuntimeUnenhancedClasses = supported
  openjpa.jdbc.SynchronizeMappings = false
  Username: ws1011
  Passwort: ftpw10
Für eine Begrenzung der Ergebnisse: Statement.setMaxRows(int maxRows);
```

```
//Beispiel ID
@Id
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="products_SEQ")
@SequenceGenerator(name="products_SEQ",  sequenceName="products_id_seq", allocationSize=1)
private long id;

// Hilfsmethode um Klassen anzugeben und ohne Konfigurationsdatei zu arbeiten.
List<Class<?>> types = new ArrayList<Class<?>>();
types.add(Product.class);
if (!types.isEmpty()) {
        StringBuffer buf = new StringBuffer();
        for (Class<?> c : types) {
                if (buf.length() > 0) { buf.append(";"); }
                buf.append(c.getName());
        } // <class>Product</class>
        map.put("openjpa.MetaDataFactory", "jpa(Types=" + buf.toString() + ")");
}

// Beispielanfrage counter=0;
Query q = em.createQuery("SELECT p FROM Product p");
q.setFirstResult(counter);
q.setMaxResults(10);

//Factory erstellen:
Persistence.createEntityManagerFactory("openjpa");
```

Tabelle 1: products Table (bitte nicht versuchen diese anzulegen!)

Spaltenname	Typ
id	Integer
name	String
price	Double
quantity	Integer