Datenbanken 2
Aufgabe 2:
JDBC - Programmierung

Prof. Dr. Daniel Pfeifer, Dipl.-Inform. Med. Martin Wiesner

19. März 2018

1 Thematik

Anhand eines Kurzbeispiels sollen die Prinzipien des prozeduralen Datenbankzugriffs mittels JDBC erlernt werden. Hierbei spielen insbesondere Datenbank-Unabhängigkeit und unterschiedliche Konzepte des Zugriffs, wie z.B. die Unterschiede zwischen dynamischem und vorbereitetem SQL eine Rolle. An diesem Beispiel sollen vor allen Dingen auch prinzipielle Aspekte des Daten-Managements untersucht werden, wie die Primärschlüssel-Verwaltung oder die Problempunkte beim Einfügen, Ändern und Löschen von Daten (z.B. Referentielle Integrität).

Beim Datenbankzugriff mit JDBC müssen zunächst Daten aus der Datenbank in Java-Objekte übertragen werden.

Für die Realisierung dieser Aufgabe wird ein Framework von teilweise abstrakten Klassen und Interfaces vorgegeben, welches in Abschnitt 3 erläutert wird.

2 Initiale Datenbank-Konfiguration

Für das Erzeugen der Datenbank sollte eine Klasse angelegt werden, welche die in Aufgabe 1 entwickelten SQL-DDL-Befehle als DB-Schema in die von Ihnen lokal auf Ihrem Rechner angelegte Datenbank einfügt. Ihre Java-Implementierung dieser Aufgabe arbeitet dann gegen dieses DB-Schema.

Hierfür kann **beispielsweise** folgendermaßen vorgegangen werden:

```
1  public final class DBCreator {
2
3      protected static final String[] SQL_DDL_STATEMENTS = {
4          "SET WRITE_DELAY FALSE", // Specific to HsqlDB
5          "CREATE TABLE ... ",
6          //... ,
7          // your job! :-)
8          //... ,
9          "SHUTDOWN" // Specific to HsqlDB
10     };
11
12     public static void main(String[] args) throws Exception {
13         // Specific to HsqlDB, Use PostgreSQL
14         String url = "jdbc:hsqldb:file:myDBSchemaName";
15         new DBCreator().createDB(url, "sa", "");
16     }
17
18     public void createDB(String jdbcURL, String user, String password)
           throws ClassNotFoundException, SQLException {
19         // get your connection here and execute your DDL statements
20     }
21
22     protected Connection createConnection(String jdbcURL, String user,
           String password) throws ClassNotFoundException, SQLException {
23         // create a connection to your DB here
24     }
25 }
```

Das Codelisting der Klasse `DBCreator` ist nur eine Schablone, die **Sie** anpassen müssen. Es empfiehlt sich jedoch, dass Sie sich daran orientieren.

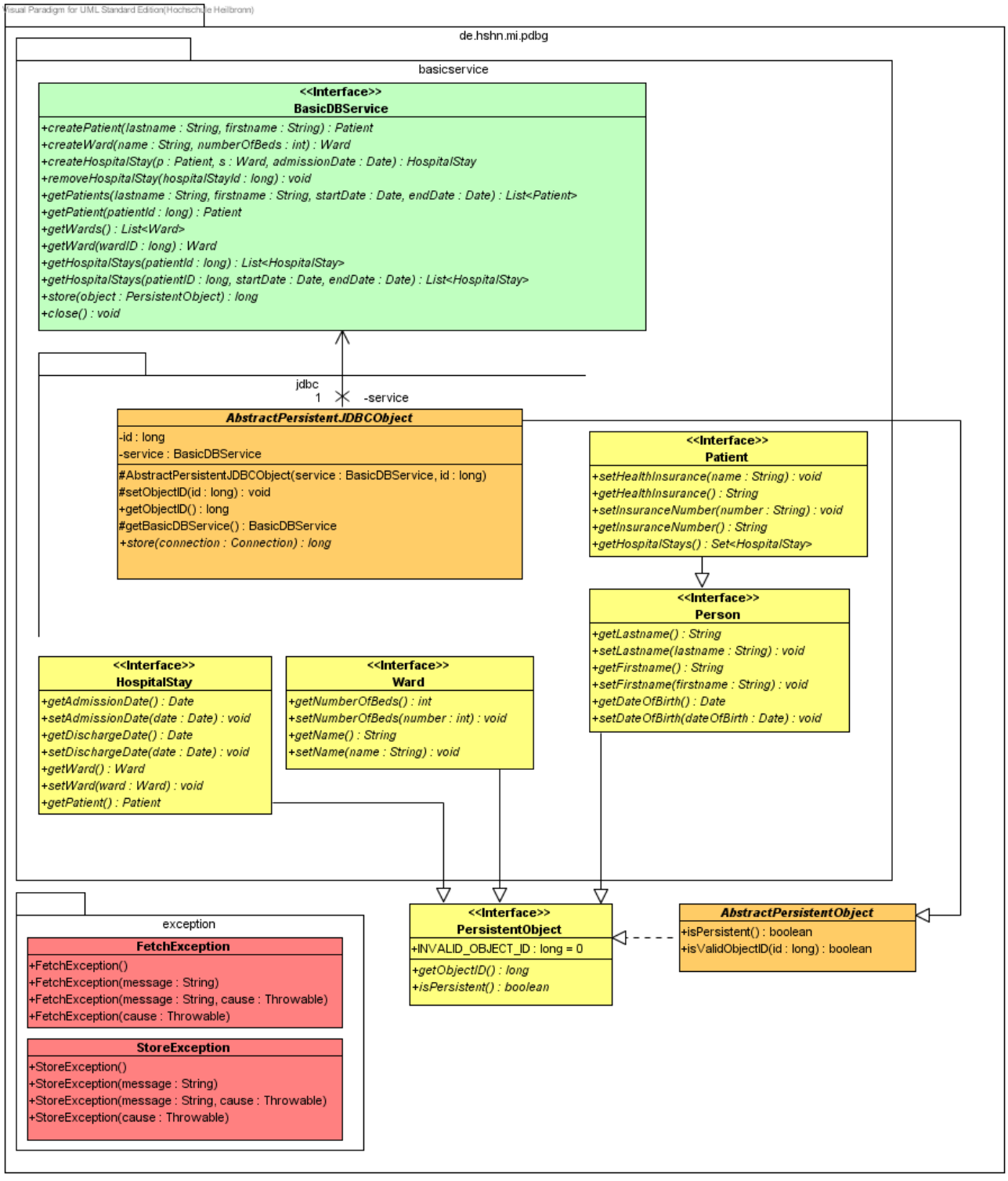

Abbildung 1: UML-Diagramm der Domänenklassen und Dienst-Schnittstelle

3 Anmerkungen zur Implementierung

Zur Illustration der Struktur des Frameworks, dient das UML-Diagramm aus Abbildung 1. Dort sind einige abgebildete Domänenklassen als Interfaces beschrieben und in gelb dargestellt. Die Persistenzfähigkeit der entsprechenden Interfaces wird dadurch ausgedrückt, dass sie vom Interface `PersistentObject` abgeleitet sind. Selbiges definiert die Methode `getObjectID()`, welche den Primärschlüssel eines persistenten Objektes zurückliefert.
Eine Teilaufgabe ist es, Implementierungen (also konkrete Klassen) für die vorgegebenen Interfaces zu realisieren. Diese können z.B. mit dem Suffix "Impl" benannt werden (z.B. `PatientImpl`) und müssen selbst angelegt werden[1].

Um Objekte zu persistieren, muss das Interface `BasicDBService` (grün) ebenfalls in einer eigenen Klasse implementiert werden. Durch die von Ihnen implementierte Klasse soll es möglich sein, neue Objekte in einem validen Objektzustand zu erzeugen und Abfragen an die Persistenzschicht zu stellen. Dazu muss die zu realisierende Implementierung eine JDBC-Connection verwalten, über welche Lese- und Änderungsanweisungen an die verwendete Datenbank abgesetzt werden können.
Alle in `BasicDBService` definierten Zugriffsmethoden müssen entsprechend mit Datenbankzugriffen ausprogrammiert werden. Zusätzlich sollte die Möglichkeit bestehen, Verbindungen zu erzeugen bzw. zu erhalten (mit `getConnection()`) bzw. wieder frei zu geben (`releaseConnection()`).

Um im Moment der Persistierung eindeutige Primärschlüssel für jedes Domänenobjekt zu erzeugen, bietet es sich an, von der abstrakten Klasse `AbstractPersistentJDBCObject` (orange) zu subklassieren; die neue Klasse ist dann für die Generierung von eindeutigen Primärschlüsseln zuständig. Für die Generierung von Primärschlüsseln kann eine sog. "Sequence" verwendet werden. Diese "Sequence" wird als Zahlengenerator verwendet. Generell lässt sich sagen, dass eine Zahl, welche durch eine "Sequence" generiert wurde, nicht noch einmal von der gleichen "Sequence" generiert wird. Dieses Vorgehen ist aber nicht zwangsläufig notwendig und kann auch auf andere Weise durchgeführt werden, solange die vergebenen Schlüssel nachweislich eindeutig sind.

Für die Implementierungen der Domänenobjekte kann man dann je nach Vorgehen von der oben erwähnten Subklasse oder direkt von `AbstractPersistentJDBCObject` erben. In selbiger ist die abstrakte Methode `store(..)` definiert, welche in den zu erstellenden Klassen (`...Impl`) jeweils spezifisch implementiert werden muss.

Aus dem Diagramm ist zudem ersichtlich, dass jede Spezialisierung (Subklasse) von `AbstractPersistentJDBCObject` einen Bezug zu einer Instanz des `BasicDBService` hat. Üblicherweise arbeitet das System mit nur einer Instanz des `BasicDBService`. Dieser Bezug bildet die Grundlage für Abfragen von abhängigen Objekten bzw. für das Persistieren von "JDBC-Objekten" oder sonstiger Kommunkation mit der Datenbank.

[1] Sinnvolle Package-Namen verwenden!

4 Validierung gegen JUnit-Testfälle

Für eine erfolgreiche Abnahme der Aufgabe müssen die Implementierung und die entwickelten Datenbankzugriffe mit bereitgestellten JUnit-Testfällen getestet werden. Dazu muss in der Klasse `BasicDBServiceFactory` die Methode `createBasicDBService` implementiert werden.

```
1 public class BasicDBServiceFactory {
2
3     public static BasicDBService createBasicDBService() {
4         //... The code needed to instantiate an implementation of a
5         //... BasicDBService
6     }
7 }
```

WICHTIG: Der Klassenname und das Package der Klasse dürfen nicht verändert werden!

Nach entsprechender Implementierung können die JUnit-Tests über die Klasse `BasicDBServiceTestSuite` gestartet werden (Eclipse: "Run as..." - "JUnit Test"). Die Klasse befindet sich in der `PDBG_A2-test.jar` Datei (vgl. Abbildung 2).

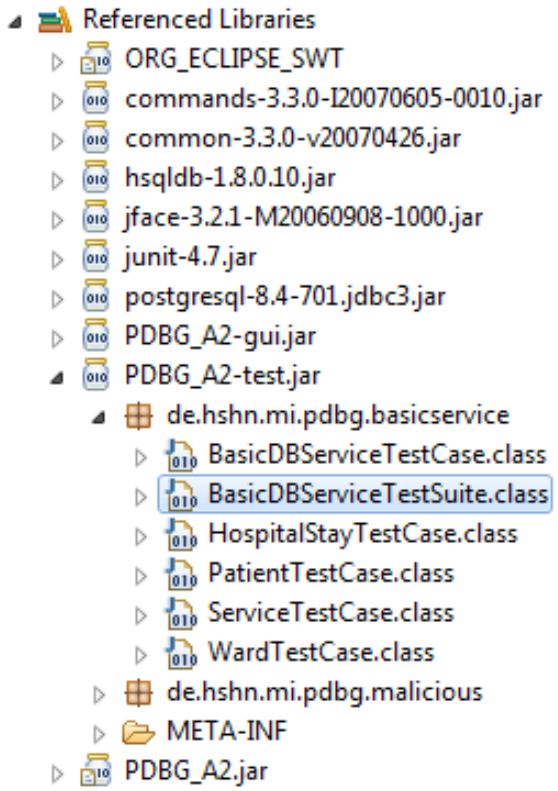

Abbildung 2: Übersicht der Testklassen

WICHTIG: Es muss zuvor noch ein VM-Parameter in der Run-Configuration gesetzt werden, welcher das Prüfen von Assertions in der VM aktiviert (vgl. Abbildung 3). Ohne diese Einstellung ist das Fehlschlagen einiger Tests vorprogrammiert.

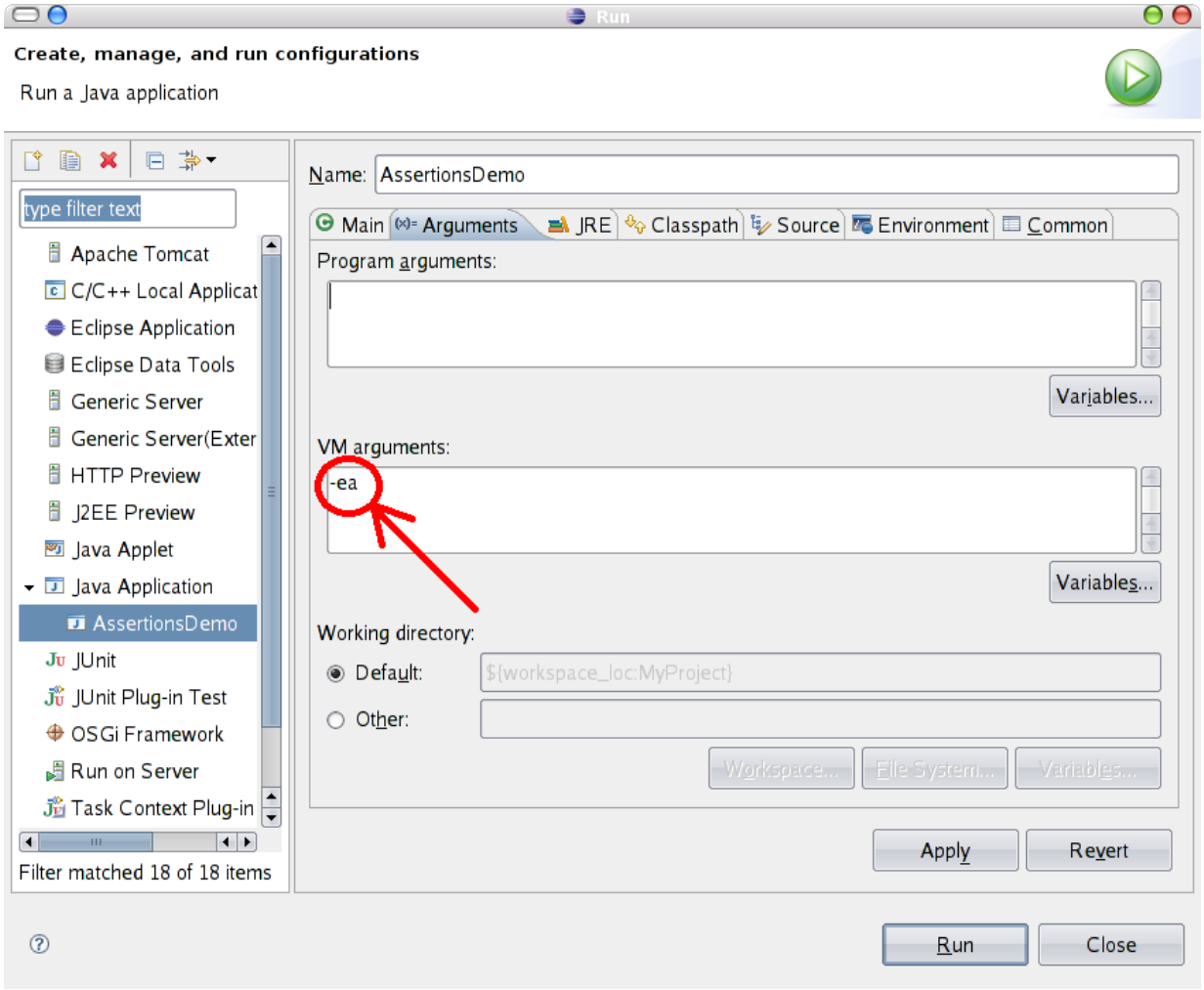

Abbildung 3: Exemplarische Run-Configuration zur Verdeutlichung der zu setzenden VM-Argumente. Sie benötigen diesen Parameter bei der Ausführung der JUnit-TestSuite

.

4.1 Testdokumentation

Alle Klassen, die das Testprogramm betreffen, sind im Package `hshn.mi.pdbg.basicservice` definiert. Das Prüfverhalten der einzelnen Tests ist entsprechend in der mitgelieferten **Javadoc** verbal beschrieben. Die Dokumentation beschreibt sowohl die Kriterien für eine erfolgreiche Durchführung der Tests, als auch die durch den Testablauf jeweiligen involvierten Methoden der Dienst-Schnittstelle und der Domänenklassen.

5 Hinweise

Bei der Implementierung der Klassen ist Folgendes zu berücksichtigen:

- In den Domänenklassen gibt es nur eine `store()`-Methode, durch welche das Einfügen und die Aktualisierung von Objekten realisiert wird. Die Unterscheidung, ob `INSERT` oder `UPDATE` durchgeführt wird, kann man über geeignete Wahl der Primärschlüssel realisieren. Dazu sollte eine neu erzeugtes, noch nicht gespeichertes Domänenobjekt als Primärschlüssel den Wert `PersistentObject.INVALID_OBJECT_ID` erhalten. Hierbei handelt es sich um eine Konstante, deren Zahlenwert mit einem durch eine "Sequence" generierten Primärschlüssel nie übereinstimmen darf.

- Die Schnittstelle schreibt vor, dass die generierten Schlüssel den Datentyp `long` besitzen müssen. Vorgegeben ist lediglich, dass beim Speichern eines Objektes über die öffentliche `store()`-Methode eine `long`-Darstellung des aktuellen Schlüssels zurückgegeben wird.

- Bei der Ausführung von SQL-Anweisungen sollen abhängig vom Kontext entweder `PreparedStatement`- oder `Statement`-Objekte benutzt werden.

- Die Implementierungen der abstrakten Klassen können nach Bedarf um eigene Methoden ergänzt werden. Die vordefinierte Schnittstelle legt lediglich fest, welche Methoden öffentlich sichtbar sind und somit von der Testapplikation aufgerufen werden.

- Die Datenbankverbindung wird über eine eigene Implementierung der Schnittstelle `BasicDBService` verwaltet. Bei der Initialisierung wird von dieser Klasse ein neues Exemplar erzeugt. Dazu sollten etwa notwendige JDBC-Verbindungsparameter übergeben werden.

- Die Verwaltung der Datenbankverbindung im `BasicDBService` sollte möglichst effizient gestaltet werden. D.h. es empfiehlt sich, nicht bei jedem Aufruf von `getConnection()` eine neue Verbindung zur Datenbank aufzubauen. Stattdessen sollte besser eine intern gespeicherte Verbindung wieder verwendet werden.

- Bei der Implementierung ist darauf zu achten, dass die im **Javadoc** spezifizierten Zusicherungen (engl. Assertions) geprüft werden und die möglicherweise zu werfenden Exceptions (`StoreException`, `FetchException`) im jeweiligen Fehlerfall ausgelöst werden, da das Verhalten im Fehlerfall durch die JUnit-Tests ebenfalls geprüft wird.