Aufgabe 8

Studierende haben als Einführung zum Thema JDBC folgende Aufgabe erhalten.

Erstellen Sie eine Datenbanksoftware zur Verwaltung von Mitarbeitern, die mit der folgenden Klasse spezifiziert werden.

```
public class Mitarbeiter {
  private int minr;
  private String name;
  private Mitarbeiter chef;
  private static int micount =1000;// fuer eindeutige minr
```

Jeder Mitarbeiter hat eine eindeutige Mitarbeiternummer, einen Namen und optional die Angabe des Chefs (Vorgesetzten). Bilden Sie die Klasse so auf Tabellen ab, dass keine NULL-Werte benötigt werden. Weiterhin ist sicherzustellen, dass jeder Mitarbeiter nur Vorgesetzter von maximal drei anderen Mitarbeitern ist. Stellen Sie dann eine Zugriffsklasse zur Verfügung die folgende Funktionalität anbietet.

- ein Mitarbeiter-Objekt wird in der Datenbank eingetragen
- es besteht die Möglichkeit einen Mitarbeiter über seine Nummer zu löschen
- es besteht die Möglichkeit einem Mitarbeiter einen (neuen) Chef über die Mitarbeiternummern zuzuordnen, dabei darf niemand direkt oder über eine Kette anderer Mitarbeiter Vorgesetzter von sich selbst werden
- ein Mitarbeiter-Objekt kann über die Mitarbeiternummer gefunden werden, nutzen Sie Optional um keinen null-Wert als Ergebnis zu liefern, falls es keinen zur Nummer passenden Mitarbeiter gibt
- es gibt die Möglichkeit eine Liste aller in der Datenbank eingetragen Mitarbeiter zu erhalten
- ergänzen Sie eine Möglichkeit die Verbindung zur Datenbank aufzubauen und die Tabellen einzuspielen, sowie die Datenbank wieder zu löschen

Schreiben Sie für alle Funktionalitäten Tests mit denen Sie das gewünschte Verhalten und die Reaktion in Problemsituationen überprüfen.

Von der Veranstaltungsseite ist eine vermeintliche Lösung erhältlich. Der enthaltene Nutzungsdialog dient der Veranschaulichung, spielt hier aber keine Rolle. Die zur Verfügung stehende Lösung hat bereits einige sinnvolle Tests, die alle durchlaufen, aber den Code nicht vollständig abdecken.

Nachdem Sie sich in die Lösung eingearbeitet haben besteht Ihre Aufgabe darin, die Testüberdeckung der Klasse DAO mit möglichst sinnvollen Tests auf 100% zu erhöhen. An mehreren Stellen ist dabei die Nutzung von Byteman sinnvoll.

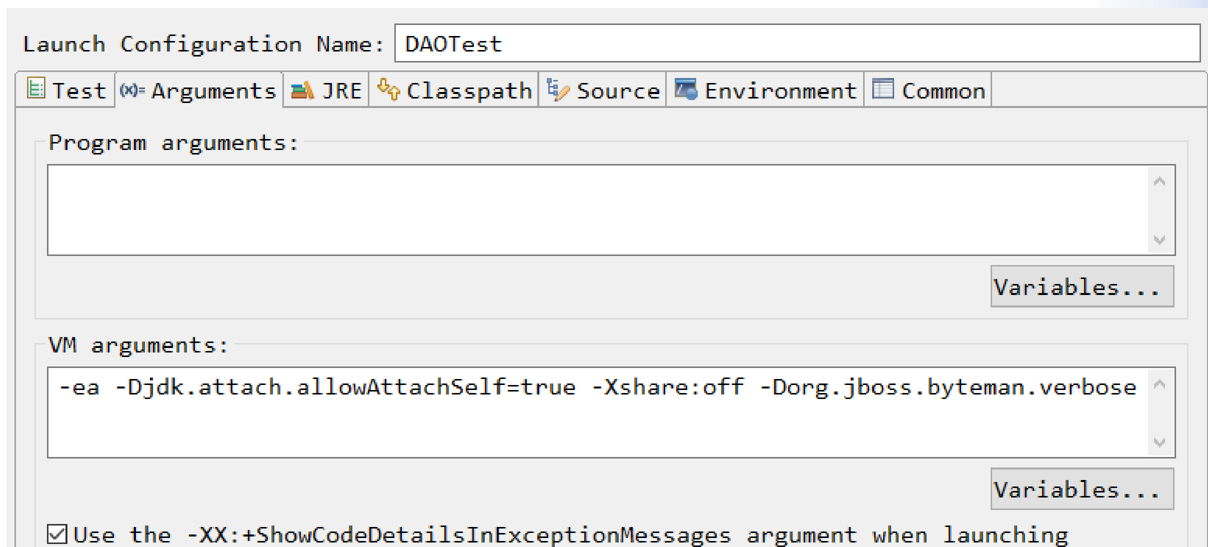
DAOTest (31.01.2021 17:30:13)

Element	Coverage	Covered I...	Missed Instructions	Total Ins...
sqmAufgabeByteManLoe	77,7 %	1.757	505	2.262
src	77,7 %	1.757	505	2.262
main	0,0 %	0	395	395
db	94,9 %	1.662	89	1.751
DAOTest.java	91,6 %	968	89	1.057
DAO.java	100,0 %	694	0	694
entity	81,9 %	95	21	116

Sie werden dabei (hoffentlich) feststellen, dass die Lösung einen konzeptuellen Fehler enthält, der zu drei ähnlichen Fehlern in den einzelnen Methoden und damit in Ihren Tests führt. Sie sollten das Problem recht einfach lösen können, womit Sie sich als Zusatzaufgabe beschäftigen sollen. Der für verteilte Systeme hier genutzte kritische Weg zur Generierung eindeutiger Ids kann vereinfachend ignoriert werden.

Hinweise:
Im vorgegebenen Projekt ist Byteman bereits eingebunden. Da das Überprüfen von Regeln recht aufwändig ist, wird vorgeschlagen das Protokoll der ausgeführten Regeln mitlaufen zu lassen. Die virtuelle Maschine wird deshalb mit folgenden Parametern aufgerufen.

```
-Djdk.attach.allowAttachSelf=true -Xshare:off -Dorg.jboss.byteman.verbose
```



Bei der Erstellung der Regeln sei u. a. an folgende Möglichkeiten erinnert, für die es leider keine wirklich systematische Dokumentation gibt. Hier werden einige Attribute der @BMRule-Annotation genannt, die in ähnlicher Form auch in der Textform zur Verfügung stehen.

`isInterface = true` // die targetClass ist ein Interface, dessen Implementierungen bearbeitet werden sollen (könnte hier wichtig sein)

`condition = "callerEquals(\"DAO.connect\",true)"` // ermöglicht die Regelausführung auf eine konkrete aufrufende Methode einzuschränken, Conditions können mit AND, OR und NOT verknüpft werden

`binding = "par = $1.contains(\"'X'\")"` // bei Parametern können deren Methoden beim Binding aufgerufen werden, hier hat $1 den Typ String, par erhält damit einen Booleschen Wert

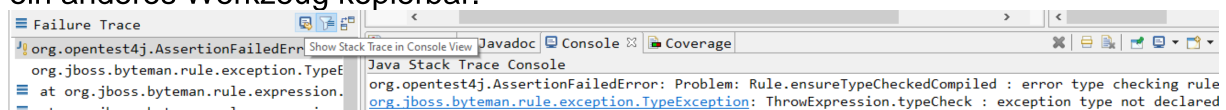
Nebenbei muss nicht für alle fehlenden Überdeckungen mit Byteman gearbeitet werden. In einem realen Projekt wäre noch ein Werkzeug zur Visualisierung der Datenbank in Benutzung, was hier nicht zur Verfügung steht. Das hier zu lösende Problem bei Byteman ist es, möglichst genau die Stelle zu spezifizieren, an der eine

Aktion, hier das Auslösen einer Exception, auftreten soll. Dabei wären Zeilennummern sicherlich eine schlechte Lösung.

Neben den oben eingestellten Kommandos, kann man einfach in eine „action“ eine Ausgabe einbauen, um sicherer zu sein, dass die Aktion ausgeführt wird.

```
action = "System.out.println(\"testname\"); throw new SQLException()"
```

Um sich eine JUnit-Fehlermeldung genauer anzusehen, kann das linke der rechten kleinen Icons der Failure Trace genutzt werden. Mit einem Rechtsklick auf dem ersten Element der Failure Trace ist mit „Copy Trace“ die detaillierte Fehlermeldung auch in ein anderes Werkzeug kopierbar.



Grundsätzlich soll diese Aufgabe zum Experimentieren mit Byteman motivieren und zeigen, dass vollständige Abdeckungen (die kein Allheilmittel sind) generell erreichbar sind. Im nächsten Schritt kann man mit Überlegungen zur Systematik beginnen. Ist es im konkreten Fall z. B. sinnvoll die zu testende Klasse zu verändern oder sollte diese, wenn möglich, nicht durch die Regeln modifiziert werden.