```
156
157 object DbTool {
158
159   val a1:Article = Article(1, "apple", 0.5)
160   val a2:Article = Article(2, "banana", 0.8)
161
162   val articles:Set[Article] = Set(a1,a2)
163
164   val c1:Customer = Customer(1, "Maier", "Franz")
165   val c2:Customer = Customer(2, "Huber", "Sepp")
166
167   val customers:Set[Customer] = Set(c1,c2)
168
169   val o1:Order = Order(100, 1, "2015-01-01 08:00:00")
170   val o2:Order = Order(101, 1, "2015-01-02 09:00:00")
171
172   val orders:Set[Order] = Set(o1,o2)
173
174   val op1:OrderPosition = OrderPosition(100, 1, 1, "apple", 5, 0.5)
175   val op2:OrderPosition = OrderPosition(100, 2, 2, "banana", 1, 0.8)
176
177   val orderpositions:Set[OrderPosition] = Set(op1,op2)
178
```

Hier finden Inserts in die Datenbank statt.

```
189
190 object Article extends Db.DbEntity[Article] {
191   def toDb(c: Connection)(a: Article) : Int = {
192     val pstmt = c.prepareStatement(insertSql)
193     pstmt.setInt(1, a.artnr)
194     pstmt.setString(2, a.description)
195     pstmt.setDouble(3, a.price)
196     pstmt.executeUpdate()
197   }
198   def fromDb(rs: ResultSet): List[Article] = {
199     val lb : ListBuffer[Article] = new ListBuffer[Article]()
200     while (rs.next()) lb.append(Article(rs.getInt("artr"), rs.getString("description"), rs.getDouble("price")))
201     lb.toList
202   }
```

Hier werden die Datentypen der jeweiligen Felder festgelegt. Zum Beispiel soll die Werte der Preise vom Datentyp Double sein.

Im nächsten Ausschnitt sieht man, wie unsere Funktionen für Drop, create und insert definiert werden.

```
203
204     def dropTableSql: String = "drop table if exists article"
205     def createTableSql: String = "create table article (artnr integer, description string, price double)"
206     def insertSql: String = "insert into article (artnr, desc, price) VALUES (?, ?, ?)"
207     def queryAll(con: Connection): ResultSet = query(con)("select * from article")
208   }
209
210   case class Article(artnr : Int, description :  String, price : Double) extends Db.DbEntity[Article] {
211     def toDb(c: Connection)(a: Article) : Int = 0
212     def fromDb(rs: ResultSet): List[Article] = List()
213     def dropTableSql: String = ""
214     def createTableSql: String = ""
215     def insertSql: String = ""
216   }
217
218
219
220   object Customer extends Db.DbEntity[Customer] {
221     def toDb(c: Connection)(cu: Customer) : Int = {
222       val pstmt = c.prepareStatement(insertSql)
223       pstmt.setInt(1, cu.cnr)
224       pstmt.setString(2, cu.firstname)
225       pstmt.setString(3, cu.lastname)
226       pstmt.executeUpdate()
227     }
228
229     def fromDb(rs: ResultSet): List[Customer] = {
230       val lb : ListBuffer[Customer] = new ListBuffer[Customer]()
231       while (rs.next()) lb.append(Customer(rs.getInt("cnr"), rs.getString("firstname"), rs.getString("lastname")))
232       lb.toList
233     }
234
235     def dropTableSql: String = "drop table if exists customer"
236     def createTableSql: String = "create table customer (cnr integer, firstname string, lastname string)"
237     def insertSql: String = "insert into customer (cnr, firstname, lastname) VALUES (?, ?, ?, ?)"
238     def queryAll(con: Connection): ResultSet = query(con)("select * from customer")
239   }
240
```

```
284
285   object Order extends Db.DbEntity[Order] {
286     def toDb(c: Connection)(o: Order) : Int = {
287       val pstmt = c.prepareStatement(insertSql)
288       pstmt.setInt(1, o.ordnr)
289       pstmt.setInt(2, o.kdnr)
290       pstmt.setString(3, o.date)
291       pstmt.executeUpdate()
292     }
293     def fromDb(rs: ResultSet): List[Order] = {
294       val lb : ListBuffer[Order] = new ListBuffer[Order]()
295       while (rs.next()) lb.append(Order(rs.getInt("ordnr"), rs.getInt("kdnr"), rs.getString("date")))
296       lb.toList
297     }
298
299     def dropTableSql: String = "drop table if exists Order"
300     def createTableSql: String = "create table Order (ordnr  Integer,kdnr  Integer, date  timestring)"
301     def insertSql: String = "insert into Order (ordnr, kdnr, date) VALUES (?, ?, ?)"
302     def queryAll(con: Connection): ResultSet = query(con)("select * from order")
303
304   }
305
306   case class Order(ordnr : Int,kdnr : Int, date : String) extends Db.DbEntity[Order] {
307     def toDb(c: Connection)(o: Order) : Int = 0
308     def fromDb(rs: ResultSet): List[Order] = List()
309     def dropTableSql: String = ""
310     def createTableSql: String = ""
311     def insertSql: String = ""
312   }
```

Unser zweites Objekt bezieht sich auf die Bestellungen. Alle Datentypen und Funktionen werden hier definiert, zudem gibt es auch eine eigene Klasse "Order". Die in diesem Abschnitt auch definiert wird. Es handelt sich hierbei um eine case class.