```java
package model.dao.impl;

import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import db.DB;
import db.DbException;
import model.dao.SellerDao;
import model.entities.Department;
import model.entities.Seller;

public class SellerDaoJDBC implements SellerDao
{
    // Dependência do DAO com a conexão
    private Connection conn;

    // Para forçar uma injeção de dependência
    public SellerDaoJDBC(Connection conn)
    {
        this.conn = conn;
    }

    // MÉTODO DE INSERÇÃO DE DADOS

    @Override
    public void insert(Seller obj)
    {
        PreparedStatement st = null;

        try
        {
            st = conn.prepareStatement
                (
                    "INSERT INTO seller " +
                    "(Name, Email, BirthDate, BaseSalary, DepartmentId) " +
                    "VALUES (?, ?, ?, ?, ?)", // ? -> chama-se placeholder
                    Statement.RETURN_GENERATED_KEYS
                );

            st.setString(1, obj.getName());
            st.setString(2, obj.getEmail());
            st.setDate(3, new java.sql.Date(obj.getBirthDate().getTime()));
```

```
51              st.setDouble(4, obj.getBaseSalary());
52              st.setInt(5, obj.getDepartment().getId());
53
54              int rowsAffected = st.executeUpdate();
55
56              if (rowsAffected > 0)
57              {
58                  ResultSet rs = st.getGeneratedKeys();
59                  if (rs.next())
60                  {
61                      int id = rs.getInt(1); // 1ª coluna de getGeneratedKeys
62                      obj.setId(id);
63                  }
64                  DB.closeResultSet(rs);
65              }
66              else
67              {
68                  throw new DbException("Erro inexperado! Nenhuma linha
  afetada!");
69              }
70          }
71          catch (SQLException e)
72          {
73              throw new DbException(e.getMessage());
74          }
75          finally
76          {
77              DB.closeStatement(st);
78          }
79      }
80
81      // MÉTODO DE ATUALIZAÇÃO DE DADOS
82
83      @Override
84      public void update(Seller obj)
85      {
86          PreparedStatement st = null;
87
88          try
89          {
90              st = conn.prepareStatement
91                  (
92                      "UPDATE seller " +
93                      "SET Name = ?, Email = ?, BirthDate = ?, BaseSalary =
  ?, DepartmentId = ? " +
94                      "WHERE Id = ?" // referente ao Id do vendedor
95                  );
96
97              st.setString(1, obj.getName());
98              st.setString(2, obj.getEmail());
```

```
 99            st.setDate(3, new java.sql.Date(obj.getBirthDate().getTime()));
100            st.setDouble(4, obj.getBaseSalary());
101            st.setInt(5, obj.getDepartment().getId());
102            st.setInt(6, obj.getId());
103
104            st.executeUpdate();
105        }
106        catch (SQLException e)
107        {
108            throw new DbException(e.getMessage());
109        }
110        finally
111        {
112            DB.closeStatement(st);
113        }
114    }
115
116    // MÉTODO DE EXCLUSÃO DE DADOS
117
118    @Override
119    public void deleteById(Integer id)
120    {
121        PreparedStatement st = null;
122
123        try
124        {
125            st = conn.prepareStatement
126                (
127                    "DELETE FROM seller WHERE Id = ?"
128                );
129
130            st.setInt(1, id);
131            st.executeUpdate();
132        }
133        catch (SQLException e)
134        {
135            throw new DbException(e.getMessage());
136        }
137        finally
138        {
139            DB.closeStatement(st);
140        }
141    }
142
143    // MÉTODO PARA BUSCA DE VENDEDOR PELO ID
144
145    @Override
146    public Seller findById(Integer id)
147    {
148        PreparedStatement st = null;
```

```
149         ResultSet rs = null;
150 
151         try
152         {
153             st = conn.prepareStatement
154                 (
155                     "SELECT seller.*,department.Name as DepName " +
156                     "FROM seller INNER JOIN department " +
157                     "ON seller.DepartmentId = department.Id " +
158                     "WHERE seller.Id = ?"
159                 );
160 
161             st.setInt(1, id);
162             rs = st.executeQuery();
163 
164             /* O ResultSet retorna a consulta SQL no formato de tabela, mas como estamos
165              * programando com orientação a objetos, a classe DAO fica responsável por
166              * transformar os dados do BD relacional em objetos associados */
167 
168             if(rs.next())
169             {
170                 Department dep = instantiateDepartment(rs);
171                 Seller obj = instantiateSeller(rs, dep);
172                 return obj;
173             }
174             return null;
175         }
176         catch (SQLException e)
177         {
178             throw new DbException(e.getMessage());
179         }
180         finally
181         {
182             DB.closeStatement(st);
183             DB.closeResultSet(rs);
184         }
185     }
186 
187     // METODO AUXILIAR PARA INSTANCIAR O DEPARTMENT DE findById
188 
189     private Department instantiateDepartment(ResultSet rs) throws SQLException
190     {
191         Department dep = new Department();
192         dep.setId(rs.getInt("DepartmentId"));
193         dep.setName(rs.getString("DepName"));
194         return dep;
```

```
195     }
196
197     // METODO AUXILIAR PARA INSTANCIAR O SELLER DE findById
198
199     private Seller instantiateSeller(ResultSet rs, Department dep) throws
    SQLException
200     {
201         Seller obj = new Seller();
202         obj.setId(rs.getInt("Id"));
203         obj.setName(rs.getString("Name"));
204         obj.setEmail(rs.getString("Email"));
205         obj.setBirthDate(rs.getDate("BirthDate"));
206         obj.setBaseSalary(rs.getDouble("BaseSalary"));
207         obj.setDepartment(dep); // Objeto Department já montado
208         return obj;
209     }
210
211     // MÉTODO PARA LISTAR TODOS OS VENDEDORES
212
213     @Override
214     public List<Seller> findAll()
215     {
216         PreparedStatement st = null;
217         ResultSet rs = null;
218
219         try
220         {
221             st = conn.prepareStatement
222                 (
223                     "SELECT seller.*,department.Name as DepName " +
224                     "FROM seller INNER JOIN department " +
225                     "ON seller.DepartmentId = department.Id " +
226                     "ORDER BY Name"
227                 );
228
229             rs = st.executeQuery();
230
231             /* O ResultSet retorna a consulta SQL no formato de tabela, mas
    como estamos
232              * programando com orientação a objetos, a classe DAO fica
    responsável por
233              * transformar os dados do BD relacional em objetos associados
    */
234
235             List<Seller> list = new ArrayList<>();
236             Map<Integer, Department> map = new HashMap<>();
237
238             while(rs.next())
239             {
240                 /* Se o departamento já existir, o map.get vai capturá-lo;
```

```
241                 * o if será falso e o dep será reaproveitado. Se o departamento
242                 * não existir, map.get retorna nulo para a variável dep, o if será
243                 * verdadeiro, vai instanciar e salvar o departamento no dep */
244
245                 Department dep = map.get(rs.getInt("DepartmentId"));
246
247                 if (dep == null)
248                 {
249                     dep = instantiateDepartment(rs);
250                     map.put(rs.getInt("DepartmentId"), dep);
251                 }
252
253                 Seller obj = instantiateSeller(rs, dep);
254                 list.add(obj);
255             }
256             return list;
257         }
258         catch (SQLException e)
259         {
260             throw new DbException(e.getMessage());
261         }
262         finally
263         {
264             DB.closeStatement(st);
265             DB.closeResultSet(rs);
266         }
267     }
268
269     // MÉTODO PARA PROCURAR VENDEDORES POR DEPARTAMENTO
270
271     @Override
272     public List<Seller> findByDepartment(Department department)
273     {
274         PreparedStatement st = null;
275         ResultSet rs = null;
276
277         try
278         {
279             st = conn.prepareStatement
280                 (
281                     "SELECT seller.*,department.Name as DepName " +
282                     "FROM seller INNER JOIN department " +
283                     "ON seller.DepartmentId = department.Id " +
284                     "WHERE DepartmentId = ? ORDER BY Name"
285                 );
286
287             st.setInt(1, department.getId());
```

```
288             rs = st.executeQuery();
289 
290             /* O ResultSet retorna a consulta SQL no formato de tabela, mas como estamos
291              * programando com orientação a objetos, a classe DAO fica responsável por
292              * transformar os dados do BD relacional em objetos associados */
293 
294             List<Seller> list = new ArrayList<>();
295             Map<Integer, Department> map = new HashMap<>();
296 
297             while(rs.next())
298             {
299                 /* Se o departamento já existir, o map.get vai capturá-lo;
300                  * o if será falso e o dep será reaproveitado. Se o departamento
301                  * não existir, map.get retorna nulo para a variável dep, o if será
302                  * verdadeiro, vai instanciar e salvar o departamento no dep */
303 
304                 Department dep = map.get(rs.getInt("DepartmentId"));
305 
306                 if (dep == null)
307                 {
308                     dep = instantiateDepartment(rs);
309                     map.put(rs.getInt("DepartmentId"), dep);
310                 }
311 
312                 Seller obj = instantiateSeller(rs, dep);
313                 list.add(obj);
314             }
315             return list;
316         }
317         catch (SQLException e)
318         {
319             throw new DbException(e.getMessage());
320         }
321         finally
322         {
323             DB.closeStatement(st);
324             DB.closeResultSet(rs);
325         }
326     }
327 }
328 
```