Spring and JDBC

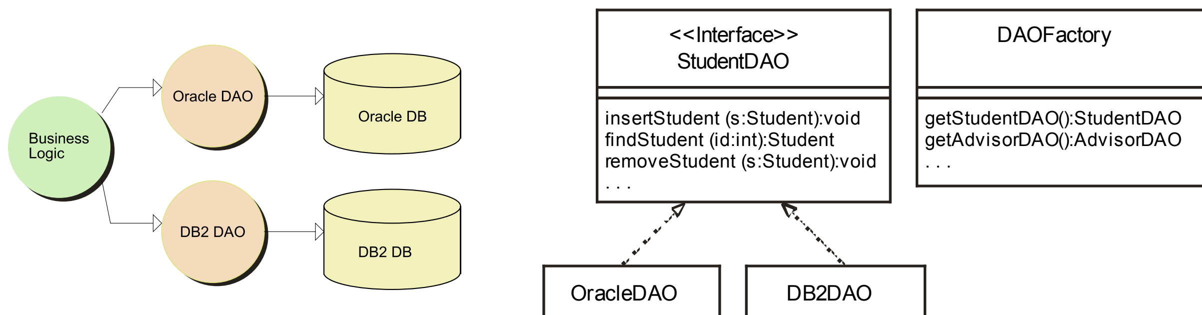
- The DAO Design Pattern
- JdbcTemplate and SimpleJdbcTemplate
- JdbcDaoSupport

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The DAO Design Pattern

- Spring supports and encourages the JEE DAO design pattern and simplifies coding, even if you you use basic JDBC



In a perfect world, or if JDBC was perfect, changing databases would not require any new code. However, in the real world, if you decide to change databases, at least some of your JDBC code probably will be affected. The goal of the DAO pattern is to minimize such changes if we do make a change.

By defining an interface for the DAO, you can minimize the impact of changes on the business logic. Here we show defining a StudentDAO interface that specifies the required behavior for any Student data-access object. The Oracle-specific and DB2-specific classes implement the interface with database-specific behavior.

The DAO Factory interface is an optional part of the pattern that is not really needed in Spring, since Spring programs generally use dependency injection to "hide" the creation of objects, including the DAO objects themselves.

You can read more about the DAO design pattern at:

`http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html`

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What's Wrong with JDBC?

- Even if you use DAOs, JDBC involves a lot of low-level, tedious and repetitive coding
- Managing JDBC connections is prone to errors
- OR/M solutions like Hibernate help with some of these issues

```
1     Connection con = null;
2     try {
3       con = dataSource.getConnection();
4       . . .
5     }
6     catch (SQLException exc) {
7     . . . }
8     finally {
9       try {
10        if (con != null) con.close();
11      catch (Exception e) {}
12    }
```


The snippet of code shown here should look familiar to anyone who's worked much with JDBC. Note the "try-catch" and "finally" blocks and how such programs need to manually manage connections.

While this code is not all that conceptually difficult, it is tedious and verbose.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introducing the Spring JDBC Module

- Spring provides the following support for database access:
  - Classes that retrieve an underlying JDBC **data source**
  - **Template** classes that simplify database coding, including templates for basic JDBC, Hibernate and JPA
  - **Support** classes that you can subclass when writing your DAOs
  - An improved exception hierarchy that's based on unchecked (runtime) exceptions
- In this chapter, we will examine the Spring support for basic JDBC - later chapters will cover Spring and OR/Ms such as Hibernate

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JdbcTemplate Class

- This is the workhorse class for programs that use Spring's basic JDBC support
- It provides methods to let you query, update, insert and delete rows in a table

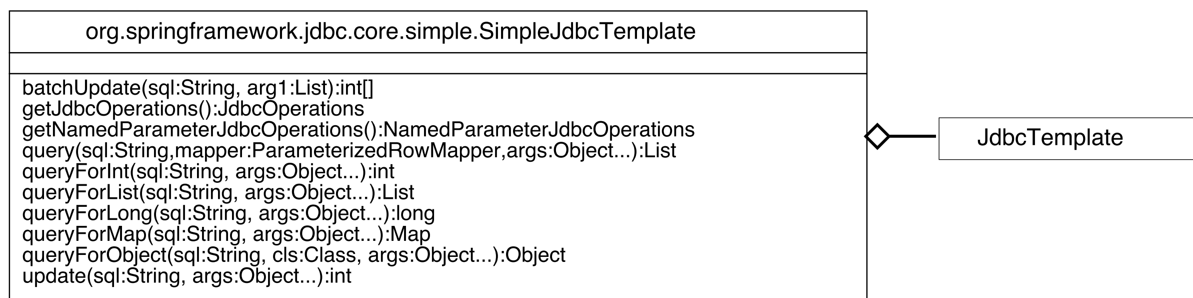
org.springframework.jdbc.core.JdbcTemplate
batchUpdate(sql:String[]):int[] call(csc:CallableStatementCreator, parms:List):Map createConnectionProxy(con:Connection):Connection execute(sql:String):void extractOutputParameters(cs:CallableStatement, parms:List):Map getFetchSize():int getMaxRows():int getQueryTimeout():int getSingleColumnRowMapper(cls:Class):RowMapper handleWarnings(warning:SQLWarning):void isIgnoreWarnings():boolean isResultsMapCaseInsensitive():boolean isSkipResultsProcessing():boolean isSkipUndeclaredResults():boolean queryForInt(sql:String, args:Object[]):int queryForList(sql:String):List queryForLong(sql:String):long queryForMap(sql:String):Map queryForObject(sql:String, arg1:Class):Object queryForRowSet(arg0:String):SqlRowSet setFetchSize(size:int):void setIgnoreWarnings(b:boolean):void setMaxRows(rows:int):void setQueryTimeout(timeOut:int):void setResultsMapCaseInsensitive(b:boolean):void setSkipResultsProcessing(b:boolean):void setSkipUndeclaredResults(b:boolean):void update(sql:String):int

Note that there are many more methods in this class that we didn't show in the figure so that the figure fits on the page.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The SimpleJdbcTemplate Class

- This class "wraps" a JdbcTemplate and uses Java 5 **generics** and **varargs** to make coding easier
- If needed, you can retrieve a reference to the wrapped JdbcTemplate by calling *getJdbcOperations*
- Note that in Spring 3, SimpleJdbcTemplate is deprecated since all of its functions have been folded back into JdbcTemplate



The SimpleJdbcTemplate class defines fewer methods than a JdbcTemplate, but if you need the ones not defined, you can retrieve the composed JdbcTemplate:

```
JdbcOperations template = myJdbcTemplate.getJdbcOperations()
```

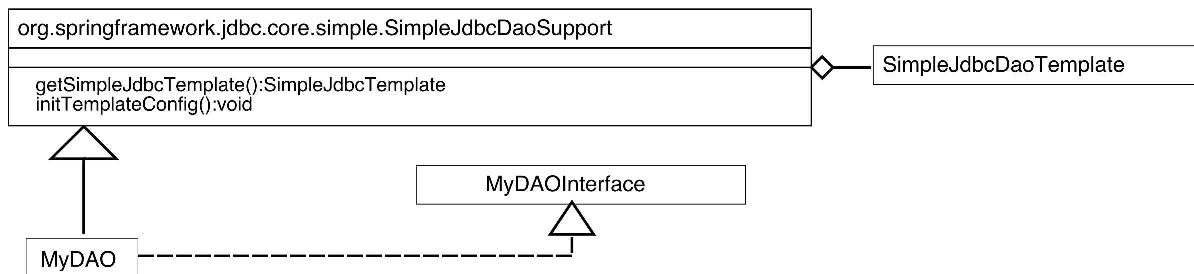
Note that many of the methods are defined using the Java 5 Object... "varargs" syntax - you can provide multiple parameters, for example, values to be inserted into several database columns. In particular, note the "query" method: it accepts a ParameterizedRowMapper object, which is a class you can write that converts a result-row into an object.

The reason Spring 3 deprecates SimpleJdbcTemplate is that Spring 3 requires Java 5 and the whole framework was refactored with that in mind. That makes SimpleJdbcTemplate redundant.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JdbcDaoSupport Classes

- Spring provides optional JdbcDaoSupport and SimpleJdbcDaoSupport (deprecated in Spring 3)
- These classes are intended to be used as a superclass for your DAOs
- They compose a JdbcTemplate (or SimpleJdbcTemplate) and free you from having to create or inject a template



2 - 7

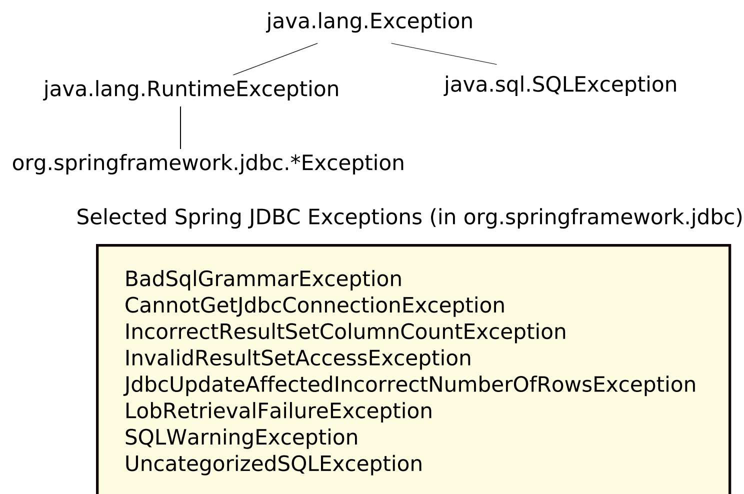
Most applications that access the database use the DAO design pattern. Spring makes it easy to get started writing DAOs by providing these superclasses.

Note that using these "support" classes is entirely optional – you could instead write your DAOs to have no Spring superclass, obtain a data source via injection, then create the template yourself.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring JDBC Exceptions

- JDBC provides only the single SQLException and it's a **checked** exception, which can require messy nested **try-catch** blocks
- Spring provides many more precisely named **unchecked** exceptions -- exception handling is simpler and more intuitive



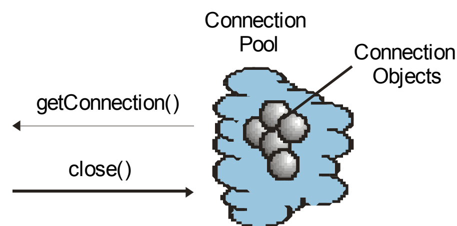
The Spring exceptions are unchecked, so you don't need try-catch blocks. This is a standard Spring philosophy. The idea is that you cannot recover from most JDBC exceptions, so why clutter the code with all of the try-catch blocks? You will need a try-catch block SOMEWHERE in the program to avoid uncaught exceptions, however.

To support this, Spring provides the SQLErrorCodeSQLExceptionTranslator class which consults a Spring-provided table of vendor error codes and their corresponding Spring exceptions.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JDBC Connection Review

- Non-Spring JDBC applications can obtain a connection in one of two ways:
    - Using the DriverManager interface
    - Using a DataSource
- The DriverManager approach is appropriate for standalone Java programs or for testing of enterprise applications
- The DataSource approach requires a JEE runtime that provides a JNDI service, but allows for **pooled** connections



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Working with Data Sources in Spring

- Spring lets you treat both JDBC approaches in a similar fashion
- Spring provides:
    - A DriverManagerDataSource class that uses the JDBC DriverManager approach
    - A JndiObjectFactoryBean class or a <jee:jndi-lookup> configuration element that locate JNDI-based DataSources
- Since you configure these in the Spring configuration file, you can easily switch between the two approaches

---

In Spring 1, we used the JndiFactoryBean, but in Spring 2 and later, it's easier to use the "jndi-lookup" element in the Spring configuration file to retrieve DataSource objects from JNDI.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DriverManager Source

- The Spring DriverManagerDataSource wraps the JDBC DriverManager and is appropriate for standalone programs
- You can inject the data source bean into your DAOs using XML or annotations

```
<bean id="studentDataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName"
    value="org.apache.derby.jdbc.EmbeddedDriver"/>
  <property name="url"
    value="jdbc:derby:c:\springclass\db\StudentDB"/>
</bean>
```

The DriverManagerDataSource encapsulates a non-pooled JDBC DataSource. Note that we must provide connection properties.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a JNDI-Based DataSource

- You can use <jee:jndi-lookup> to create a data source via a JNDI lookup
- If your "client" defines a **resource-reference**, then you should specify **resource-ref="true"** - Spring will prepend *java:comp/env* to the provided JNDI name

```
<jee:jndi-lookup id="studentDataSource"
  jndi-name="studentDS" resource-ref="true">
```

With this technique, the programmer does not need to know connection details, only the JNDI name provided by an administrator.

Spring 1.x provided the JndiObjectFactoryBean, but newer applications should use jee:jndi-lookup.

If your client is a servlet, you can define a resource reference in the servlet's Web application deployment descriptor. This is an extra level of indirection that's considered a best practice in JEE since it insulates clients from changes in the actual JNDI name.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a DAO

- DAO classes normally implement a DAO interface, optionally extend JdbcDaoSupport and use injection to obtain the data source

```
1     @Repository
2     public class StudentDAOJdbcImpl implements StudentDAO
3     {
4         private DataSource studentDataSource;
5         private JdbcTemplate template;
6
7         @Autowired
8         public void setDataSource(DataSource ds)
9         {
10            studentDataSource = ds;
11            template = new JdbcTemplate(ds);
12        }
13    . . .
14    }
```


The @Repository annotation is a special case of the @Component annotation designed for use in the data tier. You have to configure the <context:component-scan> element in the XML to include the DAO's package.

Note how we use autowiring to obtain the data source reference on a "set" method, then create the JDBC template that we can use for queries, etc.

Not shown is the interface, StudentDAO, which defines all of the methods that the DAO provides.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JdbcTemplate Query Methods

- The JdbcTemplate provides numerous ways to query for various types

```
<T> List<T> query(String sql, RowMapper<T> rm, Map<String,?> args)
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
<T> List<T> query(String sql, RowMapper<T> rm, SqlParameterSource args)
int queryForInt(String sql, Map<String,?> args)
int queryForInt(String sql, Object... args)
int queryForInt(String sql, SqlParameterSource args)
List<Map<String,Object>> queryForList(String sql, Map<String,?> args)
List<Map<String,Object>> queryForList(String sql, Object... args)
List<Map<String,Object>> queryForList(String sql, SqlParameterSource args)
long queryForLong(String sql, Map<String,?> args)
long queryForLong(String sql, Object... args)
long queryForLong(String sql, SqlParameterSource args)
Map<String,Object> queryForMap(String sql, Map<String,?> args)
Map<String,Object> queryForMap(String sql, Object... args)
Map<String,Object> queryForMap(String sql, SqlParameterSource args)
<T> T queryForObject(String sql, Class<T> requiredType, Map<String,?> args)
<T> T queryForObject(String sql, Class<T> requiredType, Object... args)
<T> T queryForObject(String sql, Class<T> requiredType, SqlParameterSource args)
<T> T queryForObject(String sql, RowMapper<T> rm, Map<String,?> args)
<T> T queryForObject(String sql, RowMapper<T> rm, Object... args)
<T> T queryForObject(String sql, RowMapper<T> rm, SqlParameterSource args)
```

SimpleJdbcTemplate has quite a few query methods, and its wrapped JdbcTemplate defines even more (recall that you can retrieve the wrapped JdbcTemplate via getJdbcOperations()).

Many of these query methods use Java 5 generics and varargs so that using them is relatively easy.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Querying Row Count

- A common DAO operation is to return the count of rows in a table

```
1     @Repository
2     public class StudentDAOJdbcImpl implements StudentDAO
3     {
4         // initialized in setDataSource()
5         private JdbcTemplate template;
6
7     . . .
8
9       public int getStudentCount()
10      {
11        return template.queryForInt(
12           "SELECT COUNT(*) FROM student");
13      }
14    }
```

Note how we provide the actual SQL for the query, but we don't have to connect to the database, create any statements or handle exceptions (if we don't want to). Spring JDBC handles all of that for us.

NOTE: In Spring 3.2.2 or later, queryForInt() is deprecated. The proper equivalent code is:

```
int count = template.queryForObject(
        "SELECT COUNT(*) FROM Student", Integer.class);
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Rows to Objects

- Spring defines **row mappers**, which are interfaces you can implement so a template can convert a result-set row to an object
- Here we show using a ParameterizedRowMapper which requires Java 5 and works with a SimpleJdbcTemplate

```
1    private class StudentRowMapper implements
2      ParameterizedRowMapper<Student>  {
3      public Student mapRow(ResultSet rs, int rowNum)
4       throws SQLException  {
5          Student s = new Student();
6          s.setGpa(rs.getDouble("gpa"));
7          s.setName(rs.getString("name"));
8          s.setStudentID(rs.getInt("studentID"));
9          return s;
10      }
11   }
```

Several of the query methods shown on the last page accept a row-mapper object - the template calls the row-mapper after a query to convert the result(s) to object(s).

Most programmers implement these row-mapper classes as Java "inner classes" within the DAO.

The ParameterizedRowMapper uses Java 5 generics to type, or parameterize the object that it returns.

The ParameterizedRowMapper interface is in the package:

`org.springframework.jdbc.core.simple`

NOTE: In Spring 3.2.2 or later, ParameterizedRowMapper is deprecated in favor of RowMapper, which has the same syntax and is functionally equivalent.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Query All

```
1     public class StudentDAOJdbcImpl
2       implements StudentDAO
3     {
4         // initialized in setDataSource()
5         private JdbcTemplate template;
6     . . .
7         private class StudentRowMapper
8            implements ParameterizedRowMapper<Student>
9         { . . . // shown earlier }
10
11        public Collection<Student> findAllStudents()
12        {
13            List<Student> students = template
14               .query("SELECT * FROM student", mapper);
15            return students;
16        }
17    }
```

Here we are using the query method:

```
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
```

Note that we are passing zero arguments for the last Object... parameter.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Find Student By ID

```
1     public class StudentDAOJdbcImpl
2       implements StudentDAO
3     {
4
5     . . .
6
7         public Student findStudentByID(int studentID)
8         {
9             return template.queryForObject(
10              "SELECT * FROM student WHERE studentID=?",
11               mapper, studentID);
12        }
13    }
```

Here we are using the query method:

```
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
```

Note that we are passing a single integer argument for the last Object... parameter. The template uses that integer to supply the value for the '?' in the SQL. If the SQL had more than one '?', we could simply pass as many arguments to the "query()" as was needed.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

SimpleJdbcTemplate Insert, Update and Delete

```
int update(String sql, Map<String,?> args)
int update(String sql, Object... args)
int update(String sql, SqlParameterSource args)

int[] batchUpdate(String sql, List<Object[]> batchArgs)
int[] batchUpdate(String sql, List<Object[]> batchArgs, int[] argTypes)
int[] batchUpdate(String sql, Map<String,?>[] batchValues)
int[] batchUpdate(String sql, SqlParameterSource[] batchArgs)
```

Note that the non-batch versions of these return an integer, which is the count of rows affected. The batch versions return an array of integers, which are the count of rows affected for each of the batch operations.

According to the Spring docs:

```
Most JDBC drivers provide improved performance if you batch multiple
calls to the same prepared statement. By grouping updates into
batches you limit the number of round trips to the database.
```

There is an example of batching in the Spring docs at:

```
http://static.springsource.org/spring/docs/2.5.6/reference/jdbc.html
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Insert

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6
7    . . .
8
9        public int insertStudent(Student s)
10       {
11         return template.update(
12           "INSERT INTO student
13           (studentId,name,gpa) VALUES(?,?,?)",
14           s.getStudentID(), s.getName(), s.getGpa());
15       }
16   }
```


Here are using the update method of the form:

```
int update(String sql, Object... args)
```

We supply three values for the Object... varargs parameter that correspond to the three '?' in the SQL. Note that we are expecting the caller to provide the primary key (studentID).

The update() method returns the count of rows affected. In this case, we'd expect the count to be one, since we inserted a single row.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete DAO Example

Student.java

StudentDAO.java

StudentDAOJdbcImpl.java

DisplayStudents.java

FindStudent.java

InsertStudent.java

spring.xml

web.xml

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1     package university;
2
3     public class Student
4     {
5         private int studentID;
6         private String name;
7         private double gpa;
8
9         public double getGpa()
10        {
11            return gpa;
12        }
13
14        public void setGpa(double gpa)
15        {
16            this.gpa = gpa;
17        }
18
19        public String getName()
20        {
21            return name;
22        }
23
24        public void setName(String name)
25        {
26            this.name = name;
27        }
28
29        public int getStudentID()
30        {
31            return studentID;
32        }
33
34        public void setStudentID(int studentID)
35        {
36            this.studentID = studentID;
37        }
38    }
```

```java
1    package dao;
2
3    import java.util.Collection;
4
5    import university.Student;
6
7    public interface StudentDAO
8    {
9        public int getStudentCount();
10       public Collection<Student> findAllStudents();
11       public Student findStudentByID(int studentID);
12       public int insertStudent(Student s);
13       public int insertStudentReturnId(Student s);
14   }
```

```
1    package dao;
2
3    import java.sql.ResultSet;
4    import java.sql.SQLException;
5    import java.util.Collection;
6    import java.util.HashMap;
7    import java.util.List;
8    import java.util.Map;
9
10   import javax.sql.DataSource;
11
12   import org.springframework.beans.factory.annotation.Autowired;
13   import org.springframework.jdbc.core.JdbcTemplate;
14   import org.springframework.jdbc.core.simple.ParameterizedRowMapper;
15   import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
16   import org.springframework.stereotype.Repository;
17
18   import university.Student;
19
20   @Repository("studentDAO")
21   public class StudentDAOJdbcImpl implements StudentDAO
22   {
23       private DataSource studentDataSource;
24       private JdbcTemplate template;
25       private StudentRowMapper mapper;
26
27       @Autowired
28       public void setDataSource(DataSource ds)
29       {
30           studentDataSource = ds;
31           template = new JdbcTemplate(ds);
32           mapper = new StudentRowMapper();
33       }
34
35       private class StudentRowMapper implements ParameterizedRowMapper<Student>
36       {
37           public Student mapRow(ResultSet rs, int rowNum) throws SQLException
38           {
39               Student s = new Student();
40               s.setGpa(rs.getDouble("gpa"));
41               s.setName(rs.getString("name"));
42               s.setStudentID(rs.getInt("studentID"));
43               return s;
44           }
45       }
```

```
46
47      public int getStudentCount()
48      {
49          return template.queryForInt("SELECT COUNT(*) FROM student");
50      }
51
52      public Collection<Student> findAllStudents()
53      {
54          List<Student> students = template
55                  .query("SELECT * FROM student", mapper);
56          return students;
57      }
58
59      public Student findStudentByID(int studentID)
60      {
61          Student s = template.queryForObject(
62                  "SELECT * FROM student WHERE studentID=?", mapper, studentID);
63          return s;
64      }
65
66      public int insertStudent(Student s)
67      {
68          return template.update("INSERT INTO student " +
69                 "(studentID, name,gpa) VALUES(?,?,?)",
70                  s.getStudentID(), s.getName(), s.getGpa());
71      }
72
73      // This insertStudent version uses the SimpleJdbcInsert
74      // class to retrieve an autogenerated key
75      public int insertStudentReturnId(Student s)
76      {
77          SimpleJdbcInsert inserter = new SimpleJdbcInsert(studentDataSource);
78          inserter.setTableName("student");
79          inserter.usingGeneratedKeyColumns("StudentID");
80          Map<String, Object> parms = new HashMap<String, Object>();
81          parms.put("name", s.getName());
82          parms.put("gpa", s.getGpa());
83
84          Number id = inserter.executeAndReturnKey(parms);
85          return id.intValue();
86      }
87
88  }
```

```
1    package servlets;
2
3    import java.io.IOException;
4    import java.io.PrintWriter;
5    import java.util.Collection;
6
7    import javax.servlet.ServletContext;
8    import javax.servlet.ServletException;
9    import javax.servlet.annotation.WebServlet;
10   import javax.servlet.http.HttpServletRequest;
11   import javax.servlet.http.HttpServletResponse;
12
13   import org.springframework.web.context.WebApplicationContext;
14   import org.springframework.web.context.support.WebApplicationContextUtils;
15
16   import university.Student;
17   import dao.StudentDAO;
18
19
20   @SuppressWarnings("serial")
21   @WebServlet("/displayStudents")
22   public class DisplayStudents extends javax.servlet.http.HttpServlet
23   {
24       @Override
25       protected void doGet(HttpServletRequest request, HttpServletResponse response)
26       {
27           response.setContentType("text/html");
28           PrintWriter out = response.getWriter();
29
30           ServletContext servletContext = getServletContext();
31           WebApplicationContext ctx =
32               WebApplicationContextUtils.getRequiredWebApplicationContext(
33                   servletContext);
34
35           try
36           {
37               StudentDAO dao = (StudentDAO)ctx.getBean("studentDAO");
38               out.println("There are " + dao.getStudentCount() + " students");
39
40               Collection<Student> students = dao.findAllStudents();
41               out.println("<table border='1'>");
42               for(Student s : students)
43               {
44                   out.println("<tr>");
45                   out.println("<td>" + s.getStudentID() + "</td>");
```

```
46                  out.println("<td>" + s.getName() + "</td>");
47                  out.println("<td>" + s.getGpa() + "</td>");
48                  out.println("</tr>");
49              }
50              out.println("</table>");
51          }
52          catch (Exception e)
53          {
54              out.println("Something went wrong: " + e);
55          }
56
57  //        Connection con = null;
58  //        try
59  //        {
60  //            Context ctx = new InitialContext();
61  //            DataSource ds =
62  //                (DataSource)ctx.lookup("java:comp/env/studentDS");
63  //            con = ds.getConnection();
64  //            Statement st = con.createStatement();
65  //            ResultSet rs = st.executeQuery("SELECT COUNT(*) FROM student");
66  //            rs.next();
67  //            out.println("There are " + rs.getInt(1) + " students");
68  //        }
69  //        catch (NamingException e)
70  //        {
71  //            e.printStackTrace();
72  //        }
73  //        catch (SQLException e)
74  //        {
75  //            e.printStackTrace();
76  //        }
77  //        finally
78  //        {
79  //            try
80  //            {
81  //               if (con != null) con.close();
82  //            } catch (Exception e1) {}
83  //        }
84
85
86      }
87  }
```

```
1     package servlets;
2
3     import java.io.IOException;
4     import java.io.PrintWriter;
5
6     import javax.servlet.ServletContext;
7     import javax.servlet.ServletException;
8     import javax.servlet.annotation.WebServlet;
9     import javax.servlet.http.HttpServletRequest;
10    import javax.servlet.http.HttpServletResponse;
11
12    import org.springframework.dao.EmptyResultDataAccessException;
13    import org.springframework.web.context.WebApplicationContext;
14    import org.springframework.web.context.support.WebApplicationContextUtils;
15
16    import university.Student;
17    import dao.StudentDAO;
18
19    @SuppressWarnings("serial")
20    @WebServlet("/findStudent")
21    public class FindStudent extends javax.servlet.http.HttpServlet
22    {
23        protected void doGet(HttpServletRequest request,
24                HttpServletResponse response) throws ServletException, IOException
25        {
26            response.setContentType("text/html");
27            PrintWriter out = response.getWriter();
28
29            ServletContext servletContext = getServletContext();
30            WebApplicationContext ctx = WebApplicationContextUtils
31                    .getRequiredWebApplicationContext(servletContext);
32
33            try
34            {
35                int studentID = Integer.parseInt(request.getParameter("studentID"));
36                StudentDAO dao = (StudentDAO) ctx.getBean("studentDAO");
37                Student s = dao.findStudentByID(studentID);
38                if (s != null)
39                {
40                    out.println("<p>Student ID: " + s.getStudentID() + "</p>");
41                    out.println("<p>Name: " + s.getName() + "</p>");
42                    out.println("<p>GPA: " + s.getGpa() + "</p>");
43                }
44            }
45            catch (NumberFormatException e)
```

```
46              {
47                  out.println("Invalid number entered. Press 'Back' and try again.");
48              }
49              catch (EmptyResultDataAccessException e)
50              {
51                  out.println("Student not found");
52              }
53              catch (Exception e)
54              {
55                  out.println("Something went wrong: " + e);
56              }
57          }
58      }
```

```
1     package servlets;
2
3     import java.io.IOException;
4     import java.io.PrintWriter;
5
6     import javax.servlet.ServletContext;
7     import javax.servlet.ServletException;
8     import javax.servlet.annotation.WebServlet;
9     import javax.servlet.http.HttpServletRequest;
10    import javax.servlet.http.HttpServletResponse;
11
12    import org.springframework.web.context.WebApplicationContext;
13    import org.springframework.web.context.support.WebApplicationContextUtils;
14
15    import university.Student;
16    import dao.StudentDAO;
17
18    @SuppressWarnings("serial")
19    @WebServlet("/insertStudent")
20    public class InsertStudent extends javax.servlet.http.HttpServlet
21    {
22        protected void doPost(HttpServletRequest request,
23                HttpServletResponse response) throws ServletException, IOException
24        {
25            response.setContentType("text/html");
26            PrintWriter out = response.getWriter();
27
28            ServletContext servletContext = getServletContext();
29            WebApplicationContext ctx = WebApplicationContextUtils
30                    .getRequiredWebApplicationContext(servletContext);
31
32            String name = request.getParameter("name");
33            String s = request.getParameter("gpa");
34
35            double gpa = Double.parseDouble(s);
36
37            try
38            {
39                StudentDAO dao = (StudentDAO) ctx.getBean("studentDAO");
40
41                Student student = new Student();
42                student.setGpa(gpa);
43                student.setName(name);
44                int rows = dao.insertStudent(student);
45                out.println("Insert successful, " + rows + " inserted.");
```

```
46          }
47          catch (Exception e)
48          {
49              out.println("Something went wrong: " + e);
50          }
51      }
52  }
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <beans xmlns="http://www.springframework.org/schema/beans"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4        xmlns:aop="http://www.springframework.org/schema/aop"
5        xmlns:context="http://www.springframework.org/schema/context"
6        xmlns:jee="http://www.springframework.org/schema/jee"
7        xmlns:jms="http://www.springframework.org/schema/jms"
8        xmlns:lang="http://www.springframework.org/schema/lang"
9        xmlns:tx="http://www.springframework.org/schema/tx"
10       xmlns:util="http://www.springframework.org/schema/util"
11       xsi:schemaLocation="http://www.springframework.org/schema/aop
12           http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
13           http://www.springframework.org/schema/beans
14           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
15           http://www.springframework.org/schema/context
16           http://www.springframework.org/schema/context/spring-context-3.1.xsd
17           http://www.springframework.org/schema/jee
18           http://www.springframework.org/schema/jee/spring-jee-3.1.xsd
19           http://www.springframework.org/schema/lang
20           http://www.springframework.org/schema/lang/spring-lang-3.1.xsd
21           http://www.springframework.org/schema/tx
22           http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
23           http://www.springframework.org/schema/util
24           http://www.springframework.org/schema/util/spring-util-3.1.xsd">
25
26      <context:component-scan base-package="dao"/>
27
28
29       <jee:jndi-lookup id="studentDataSource" jndi-name="java:jboss/datasources/Exam
30
31   <!--    <bean id="studentDAO" class="dao.StudentDAOJdbcImpl"> -->
32
33   <!--        <property name="dataSource" ref="studentDataSource" /> -->
34
35   <!--    </bean>  -->
36
37
38
39   </beans>
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.
3      <display-name>SpringJdbcTemplateWeb</display-name>
4        <context-param>
5            <param-name>contextConfigLocation</param-name>
6            <param-value>/WEB-INF/spring.xml</param-value>
7        </context-param>
8        <listener>
9            <display-name>ContextLoaderListener</display-name>
10           <listener-class>
11               org.springframework.web.context.ContextLoaderListener
12           </listener-class>
13       </listener>
14     <welcome-file-list>
15       <welcome-file>index.html</welcome-file>
16       <welcome-file>index.htm</welcome-file>
17       <welcome-file>index.jsp</welcome-file>
18       <welcome-file>default.html</welcome-file>
19       <welcome-file>default.htm</welcome-file>
20       <welcome-file>default.jsp</welcome-file>
21     </welcome-file-list>
22   </web-app>
```

Chapter Summary

In this chapter, you learned:

- The basics of Spring's JDBC support
- How to use SimpleJdbcTemplate and SimpleJdbcDaoSupport

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum