

4 A Sample Web Service Application

Objectives

After completing this chapter, you should be able to:

1. Write a Web Service that provides access to employee records stored in a relational database
2. Write a Web Service using JDK 6 or above
3. Publish a Web Service using basic Java Endpoint class
4. Test a Web Service with SOAPUI testing tool
5. Use wsimport to generate a Web Service stub for the client
6. Write a simple Web Service consumer to invoke a Web Service

4.1 A Sample application

Welcome to the world of Web Services! You may find this chapter technically challenging at first; however, as you work your way through the examples, you will find that the same patterns are used repeatedly throughout. If you think of writing Web Services as similar to writing any other Java class, that may help to ease any anxiety about the difficulty of this task.

In this application, we deploy a simple SOAP server using basic Java JDK delivery. In order to make this application work, you will need the following software packages that can be downloaded from the Internet (more instructions are included in Appendix A).

- Java JDK 6
- MySQL Community Server 5.6
- MySQL Employees sample database
- MySQL JDBC driver

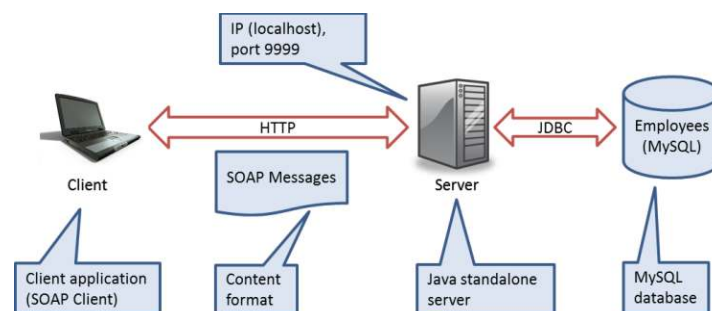


Figure 4-1. A n-tier application

A WS application is created using a Java framework to enable a WS consumer to manipulate employee records stored in a relational database. Accessing the database from a Web server is accomplished using JDBC technology. MySQL is the relational database used for this example. The transport protocol for the WS is HTTP.

To avoid adding complexity to an already complicated concept, security concerns are not considered in this example. Accessing the database from a remote machine (i.e., WS client) without proper authentication is not a good practice; however, in this application, accessing the database with fixed user ID and password is a matter of simplicity, not security. Furthermore, the use of the data source is much more efficient using direct JDBC calls, however, the sample code does not follow that standard convention.

The basic Java Endpoint class does not scale well in a business computing environment, but it is used here to allow the simplest Java environment capable of supporting a simple WS application. In later chapters, you can apply similar programming principles and techniques for WS programming to deploy WS applications on an Apache Tomcat or an Oracle WebLogic server. These two servers are covered in Chapters 5 and 6, respectively.

Remember, the central idea of this chapter in terms of WS programming is how to get data from the database through the use of WS technology, and SOAP in particular.

4.1.1 Use Case Diagram

Consider the following use case diagram for this sample application. From the perspectives of WS clients, it invokes four operations of an employee data service. Basic data exchange includes two major data types: employee number and employee record.

A resource can be created, read (or obtained), updated (or changed), and deleted. The concept of CRUD has been fundamental to computer programming since the beginning of the field computer science. We have a set of employee records stored in a database, and we want to manipulate them from a remote machine using SOAP via WS technology.

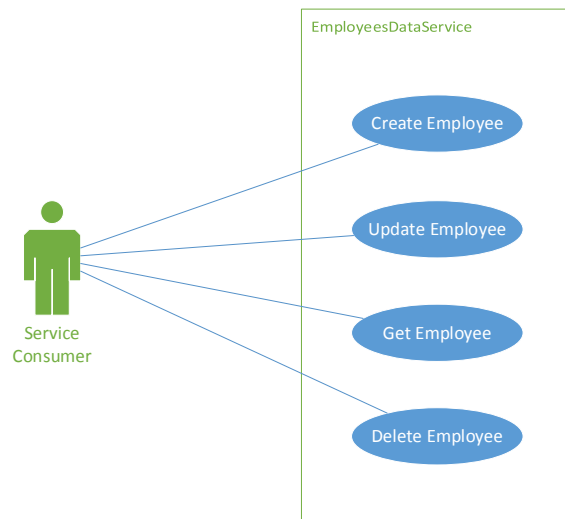


Figure 4-2. Use Cases

Use Case 1: Create Employee

Primary Actor: Service Consumer

Main Success Scenario:

1. An end-user enters required employee information.
2. The service consumer then verifies the information.
3. The service consumer then calls the *employees* data service to create a new employee.
4. The service consumer presents a new employee number to the end-user.

Use Case 2: Update Employee

Primary Actor: Service Consumer

Main Success Scenario:

1. An end-user updates the required employee information.
2. The service consumer then verifies the information.
3. The service consumer then calls the *employees* data service for the update.
4. The service consumer informs the end-user about the status of the update.

Use Case 3: Get (Read) Employee

Primary Actor: Service Consumer

Main Success Scenario:

1. An end-user enters an employee number.
2. The service consumer then validates the number.
3. The service consumer then calls the *employees* data service to retrieve the employee record.
4. The service consumer presents the employee record to the end-user.

Use Case 4: Delete Employee

Primary Actor: Service Consumer

Main Success Scenario:

1. An end-user enters an employee number.
2. The service consumer then validates the number.
3. The service consumer then calls the *employees* data service to remove the employee record.
4. The service consumer informs the end-user about the status of the deletion.

A SOAP exception is thrown in when an error condition occurs.

4.1.2 Sequence Diagram

In a typical WS call, many layers of software are involved; however, at a high level, the sequence of actions may be represented as in the following sequence diagram.

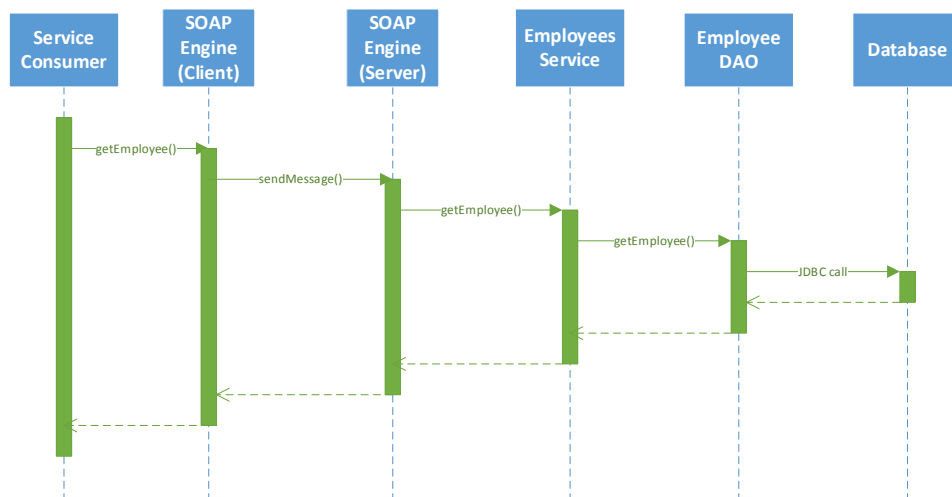


Figure 4-3. Sequence diagram of a *getEmployee* operation

Since all four operations are the basic request-response type of message exchange, a single sequence diagram of *getEmployee* operation is shown.

A consumer, when ready, invokes a SOAP engine on the client side to begin a SOAP call across the network. In this scenario, the consumer understands the SOAP message fully and constructs a SOAP message using SOAP with Attachment API for Java (SAAJ). Once a SOAP message is formed, the SOAP engine sends the message to the remote server via HTTP. After successfully receiving the message, the server processes the request by invoking the appropriate business or data services in the backend. In this case, the `getEmployee` method of the *employees* service is invoked. Before the data access layer is called, additional business logic processing can be done in this class to manipulate the data. `EmployeeDAO` is a component that interacts directly with the database using JDBC for data processing. The data source may not always be a relational database.

Once the processing is completed, the *employees* service, with the help of the WS package, forms a SOAP message and returns to the SOAP engine on the server side. As a part of the request-response message exchange pattern, the response is then returned to the SOAP engine on the client side. Once the client SOAP engine successfully receives the message, it returns to the Service Consumer for final processing.

The process of forming a SOAP message is often called ‘marshalling’. Conversely, the process of decoding a SOAP message into a native form for further processing is called ‘unmarshalling’.

This sequence diagram shows an example of a synchronous message exchange. In other words, activities in this diagram occur in sequence. In some cases, the processing may take a long time, and the server may return immediately before the processing completes. This is a form of asynchronous message exchange. When the server has completed processing the request, it may initiate a call to the client to return the response with the actual data or simply a notification. The client can also periodically poll the server for data. The second option suffers two problems. If the timing window between two polls is too large, the delay can be significant. If it is small, it wastes valuable processing power on both sides.

4.1.3 Deployment Diagram

The simple deployment of this WS application is depicted as follows:

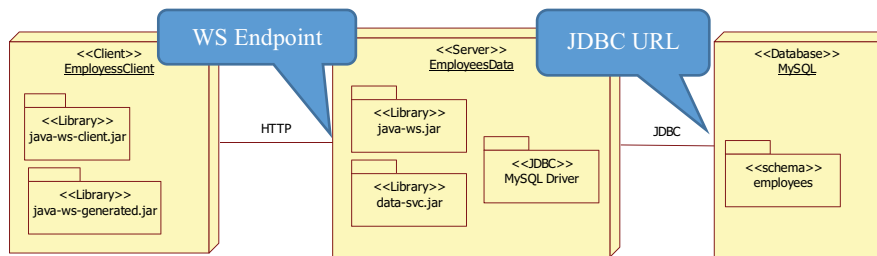


Figure 4-4. A Simple Deployment Diagram

In a real-world application, an IT organization may deploy a complex mesh of servers and databases to manage their WS activities. There can be many client applications. In this sample WS application, we create an environment that includes a client machine, a server machine, and a database machine. These machines can be virtual, which means that all three can be hosted by a single physical machine.

On the server side, we develop two sets of Java libraries – `java-ws.jar` and `data-svc.jar`. The first contains the WS code that interacts with the client over the network protocol HTTP. The second deals with the database access via JDBC calls with the help of MySQL driver code written by MySQL database developers. Together, they comprise a complete application.

On the client side, we develop a Java library that contains the WS client code, `java-ws-client.jar`. We use `wsimport` to generate the second library, `java-ws-generated.jar`. This second library contains all of the necessary code to interact with the server WS engine.

4.1.4 JDBC URL

To access a relational database from a Java application, a database connection must be established using a JDBC URL with the following format:

```
jdbc:<subprotocol>:<subname>
```

where

- `<subprotocol>` is the name of the driver that was registered with Oracle. In this application, 'mysql' is used.
- `<subname>` is the identification of the resource. It has the following format:

```
//host:port/subsubname
```

`subsubname` consists of the database schema name, user identification and password.

In this example, a full JDBC URL can be written as:

```
jdbc:mysql://localhost:3306/employees?user=empl_1&password=password
```

In order to access the database via JDBC connection, a database account was created and assigned to the *employees* database. It has all privileges to the *employees* database. The MySQL default access port is 3306. Before you run the SOAP server program, make sure to download a JDBC driver and include the driver in your Java's classpath.

4.1.4.1 DbConfig.java

In this sample code, the default values to connect to the MySQL database are (see DbConnection.java):

| | |
|-------------------------|-----------------------|
| Hostname | Saintmonica |
| Port number | 3306 |
| Account | empl_1 |
| Password | Password |
| Database name | Employees |
| JDBC driver name | com.mysql.jdbc.Driver |
| Subprotocol | Mysql |

Table 1. Database Configuration Parameters



www.sylvania.com

We do not reinvent
the wheel we reinvent
light.

Fascinating lighting offers an infinite spectrum of possibilities: Innovative technologies and new markets provide both opportunities and challenges. An environment in which your expertise is in high demand. Enjoy the supportive working atmosphere within our global group and benefit from international career paths. Implement sustainable ideas in close cooperation with other specialists and contribute to influencing our future. Come and join us in reinventing light every day.

Light is OSRAM

**OSRAM
SYLVANIA** 



Listing 41. DbConfig.java class

```
package com.bemach.data;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

/**
 * Make sure to download MySQL JDBC Driver from the website
 * Extract it, and include this file (name may be changed between
 * release): mysql-connector-java-5.1.24-bin.jar into your classpath
 *
 */
public class DbConfig {
    private String subprot = "mysql";
    private String host = "saintmonica";
    private String port = "3306";
    private String db = "employees";
    private String uid = "empl_1";
    private String psw = "password";
    private String driverName = "com.mysql.jdbc.Driver";

    public String getSubprot() {
        return subprot;
    }

    public void setSubprot(String subprot) {
        this.subprot = subprot;
    }

    public String getDriverName() {
        return driverName;
    }

    public void setDriverName(String driverName) {
        this.driverName = driverName;
    }

    public String getHost() {
        return host;
    }

    public void setHost(String host) {
        this.host = host;
    }
}
```



```
public String getPort() {
    return port;
}

public void setPort(String port) {
    this.port = port;
}

public String getDb() {
    return db;
}

public void setDb(String db) {
    this.db = db;
}

public String getUid() {
    return uid;
}

public void setUid(String uid) {
    this.uid = uid;
}

public String getPsw() {
    return psw;
}

public void setPsw(String psw) {
    this.psw = psw;
}
}
```

For the application, the DbConfigure class is a placeholder for all necessary configuration parameters for connecting to the MySQL database.

4.1.5 Web Service Endpoint

A WS must be published via a unique service endpoint in order to be accessed by a WS client. A URL is a pointer to an available resource. This unique service endpoint can be stated using a URL with the following format:

<scheme>:<hier-part>?query

where

- <scheme> is http protocol
- <hier-part> is //host:port/path

In this example, the service endpoint is defined as:

<http://localhost:9999/doc/employees>

and

<http://localhost:9999/rpc/employees>

4.1.5.1 SvrConfig.java

For HTTP connectivity to be used for SOAP, the sample code must use the following default values:

| | |
|--------------------|-----------|
| Hostname | localhost |
| Port number | 9999 |
| Protocol | http |

Table 2. Server Configuration Parameters



360°
thinking.

Deloitte.

Discover the truth at www.deloitte.ca/careers

© Deloitte & Touche LLP and affiliated entities.



Listing 4-2. SvrConfig.java class

```
package com.bemach.ws.server;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import com.bemach.data.DbConfig;

/**
 * Make sure to download MySQL JDBC Driver from the website
 * Extract it, and include this file (name may be changed between
 * release): mysql-connector-java-5.1.24-bin.jar into your classpath
 *
 */
public class SvrConfig {
    private String listenHostname = "localhost";
    private String listenPort = "9999";
    private String listenInterface = "HelloWorld";
    private String listenProtocol = "http";
    private DbConfig dbCfg = new DbConfig();

    public DbConfig getDbCfg() {
        return dbCfg;
    }

    public void setDbCfg(DbConfig dbCfg) {
        this.dbCfg = dbCfg;
    }

    public String getListenHostname() {
        return listenHostname;
    }

    public void setListenHostname(String listenHostname) {
        this.listenHostname = listenHostname;
    }

    public String getListenPort() {
        return listenPort;
    }

    public void setListenPort(String listenPort) {
        this.listenPort = listenPort;
    }

    public String getListenInterface() {
        return listenInterface;
    }
}
```

```

public void setListenInterface(String listenInterface) {
    this.listenInterface = listenInterface;
}

public String getListenProtocol() {
    return listenProtocol;
}

public void setListenProtocol(String listenProtocol) {
    this.listenProtocol = listenProtocol;
}
}

```

The SvrConfig class consists of information that is used to form a service endpoint for both styles – document and RPC. Furthermore, the class contains the configuration parameters that the data access code uses in order to access the database.

4.1.6 About the employees¹ sample database from MySQL

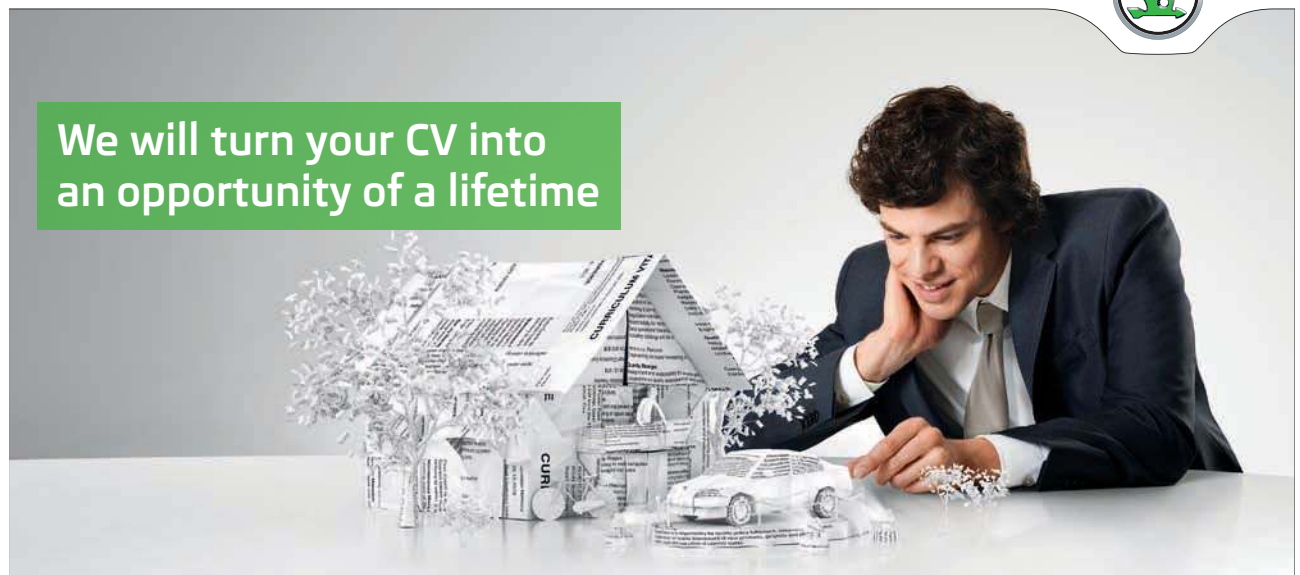
The employees database is a sample database from MySQL. The database schema was developed by professor Chua Hock Chuan at Nanyang Technological University in Singapore. The site can be visited at <http://www.ntu.edu.sg/home/ehchua/programming/sql/SampleDatabases.html>.

SIMPLY CLEVER

ŠKODA



We will turn your CV into
an opportunity of a lifetime



Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.

Send us your CV on
www.employerforlife.com



All fields of the employees table are required with the primary key being the employee number. The employees data record can be represented by Employee class in Java. This class is defined as follows:

4.1.6.1 Employee.java

We create an Employee data object that contains an employee record. Employee is a Java class that uses Java Architecture for XML Binding (JAXB) annotations to assist the marshalling process. JAXB allows Java developer to use Java API to read and write objects to and from an XML document. It eases the process of reading and writing XML documents in Java. In particular, the annotation provides a simpler mechanism for the SOAP engine to transform Java objects into XML and vice versa.

I joined MITAS because
I wanted **real responsibility**

The Graduate Programme
for Engineers and Geoscientists
www.discovermitas.com

Month 16
I was a construction
supervisor in
the North Sea
advising and
helping foremen
solve problems

Real work
International opportunities
Three work placements



 **MAERSK**



Listing 4-4. *Employee.java* Class

```
package com.bemach.data;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.io.Serializable;
import java.util.Calendar;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name="EmployeeService", namespace="http://bemach.com")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name="employee")

public class Employee implements Serializable{
    private static final long serialVersionUID = 1L;
    @XmlElement(required=true)
    private long emplNo;
    @XmlElement(required=true)
    private String firstName;
    @XmlElement(required=true)
    private String lastName;
    @XmlElement(required=true)
    private Calendar birthDate;
    @XmlElement(required=true)
    private String gender;
    @XmlElement(required=true)
    private Calendar hireDate;

    public long getEmplNo() {
        return emplNo;
    }
    public void setEmplNo(long emplNo) {
        this.emplNo = emplNo;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

```

public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}
public Calendar getBirthDate() {
    return birthDate;
}
public void setBirthDate(Calendar birthDate) {
    this.birthDate = birthDate;
}
public String getGender() {
    return gender;
}
public void setGender(String gender) {
    this.gender = gender;
}
public Calendar getHireDate() {
    return hireDate;
}
public void setHireDate(Calendar hireDate) {
    this.hireDate = hireDate;
}
}

```

All required fields are reflected in XML elements within the sequence. Optional elements often include `numOccurs="0"`. The Java data types are mapped neatly into XML intrinsic data types, as shown in the schema.

Listing 4-5. Data type of 'employee' within XSD

```

<xs:schema xmlns:tns="http://employees.rpc.ws.bemach.com/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
  targetNamespace="http://employees.rpc.ws.bemach.com/">
  <xs:element name="SOAPException" type="tns:SOAPException" />
  <xs:complexType name="employee">
    <xs:sequence>
      <xs:element name="emplNo" type="xs:long" />
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="lastName" type="xs:string" />
      <xs:element name="birthDate" type="xs:dateTime" />
      <xs:element name="gender" type="xs:string" />
      <xs:element name="hireDate" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SOAPException">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```


4.2 Develop a Web Service

A bottom-up approach for developing a Web Service involves the following activities:

- Write a data access object.
- Write a business logic object.
- Write a service object.
- Deploy a service to a server.
- Publish the server for use.

In this application, no business services are included, thus the activities are simplified as follows:



Listing 4-6. Activities for writing Web Services with Java

ie business school

#1 EUROPEAN BUSINESS SCHOOL
FINANCIAL TIMES 2013

#gobeyond

MASTER IN MANAGEMENT

Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.

- Choose your area of specialization.
- Customize your master through the different options offered.
- Global Immersion Weeks in locations such as London, Silicon Valley or Shanghai.

Because you change, we change with you.

www.ie.edu/master-management | mim.admissions@ie.edu | f t in YouTube

Download free eBooks at bookboon.com



Click on the ad to read more

We create two Java projects under Eclipse – data-svc and java-ws. The data-svc project holds Java code that interacts with the database via JDBC. This project creates a library called ‘data-svc.jar’. This library contains four Java classes:

- DbConfig.java
- DbConnection.java
- Employee.java
- EmployeeDao.java

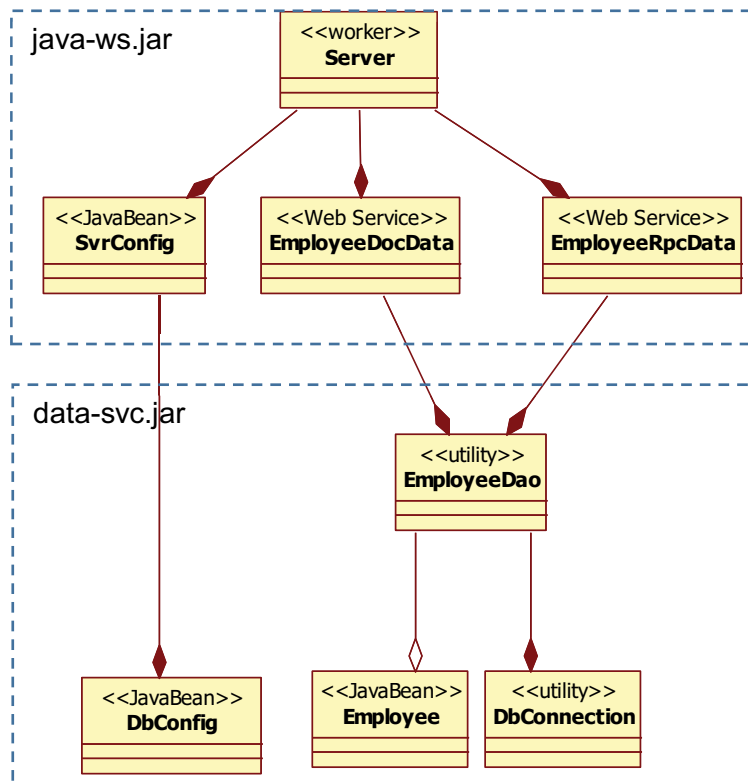
The java-ws project, which resulted in a java-ws.jar library, consists of the following Java classes:

- SvrConfig.java
- Server.java
- EmployeeDocData.java
- EmployeeRpcData.java

4.2.1 Class Diagram

A static view of the server application is depicted in the following class diagram.

Listing 4-7. A class diagram



In a class diagram, the hollow-diamond adornment indicates a part-whole relationship between the classes. This is called an ‘aggregation’. On the other hand, the solid-diamond adornment represents a composite relationship between the classes. A composition is stronger than an aggregation in that the former involves a complete management of the lifetime of the object. For example, at runtime, an EmployeeDao object is responsible for allocation and deallocation of the DbConnection object. The Employee object is allocated by the EmployeeDao but deallocated by the EmployeeDocData or EmployeeRpcData object.

The dotted-line boxes indicate the boundaries of the two libraries to be created for this application.

4.2.2 Write Data Access Class

The Data Access Object (DAO) design pattern is used to provide abstract and encapsulated access of data from the data sources. It manages the connection with the data source to store and retrieve data.

First, we create a Java project called ‘data-svc’ (see section 7.2.1). After we complete our coding of the Java classes, this project should appear as follows:

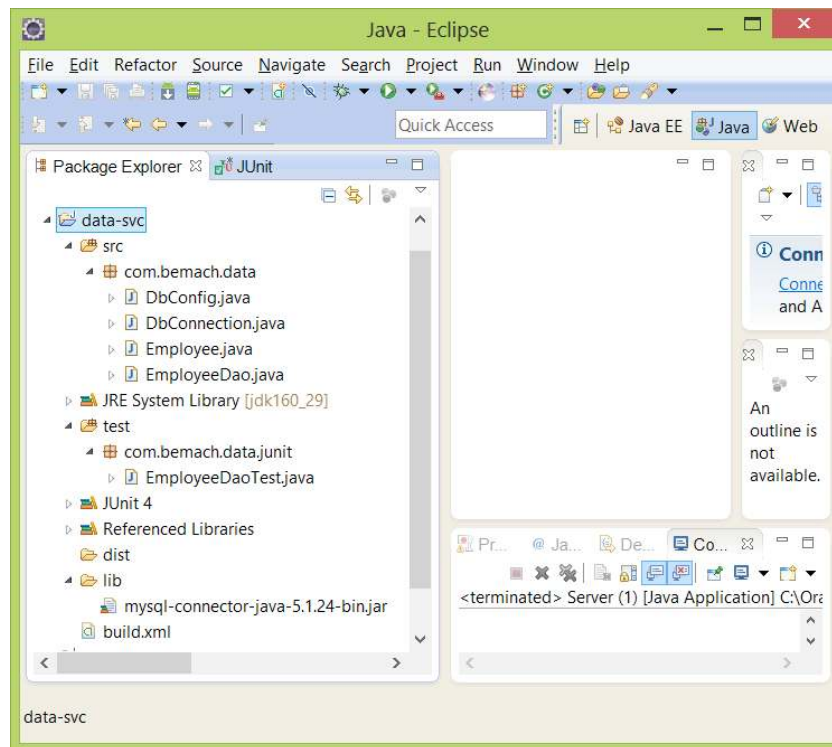


Figure 4-6. Java Project: data-svc

4.2.2.1 Import JDBC driver to the project

Following the instructions in section 7.2 to install MySQL and download an appropriate JDBC driver for MySQL database, the JDBC driver that is used for this application is mysql-connector-java-5.1.24-bin.jar.

Create a folder named 'lib' under the data-svc project by first selecting the project. Then, choose File → New → Folder. This folder will contain the JDBC driver library. Expand the project by clicking on the triangle to the left of the project name.

Now, import the JDBC driver that you have downloaded by clicking on the lib folder. Then, choose File → Import... The Select screen pops up as follows:

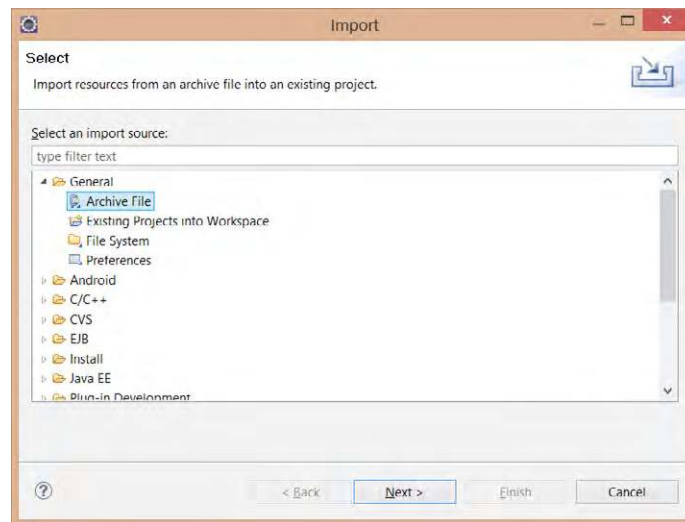


Figure 4-7. Select import type

"I studied English for 16 years but...
...I finally learned to speak it in just six lessons"
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download



Choose General → Archive File, and then click on the ‘Next’ button at the bottom of screen. An Archive file screen pops up as follows:

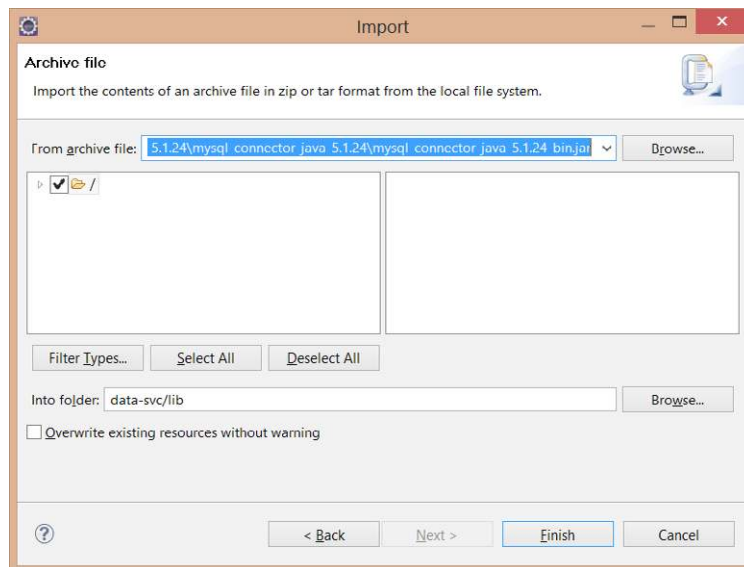


Figure 4-8. Import Archive file screen

If you know the location of the JDBC driver for MySQL database, enter the file name and location. Otherwise, use the ‘Browse’ button on the right and choose the file. Then, click the ‘Finish’ button on the bottom of the screen.

4.2.2.2 Reference to the library

Next, make sure the project has a reference to the MySQL JDBC driver library. First, choose the data-svc project. Then, select Project (menu) → Properties. The Properties for the data-svc screen will pop up as follows:

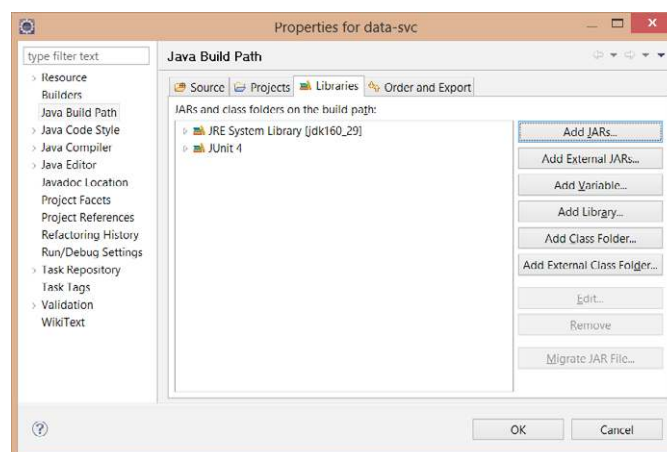


Figure 4-9. Java Build Path

On the left panel of the screen, choose Java Build Path. Select the Libraries tab from the top of the right panel. Click on the Add JARs... button. A JAR Selection screen will pop up as follows:

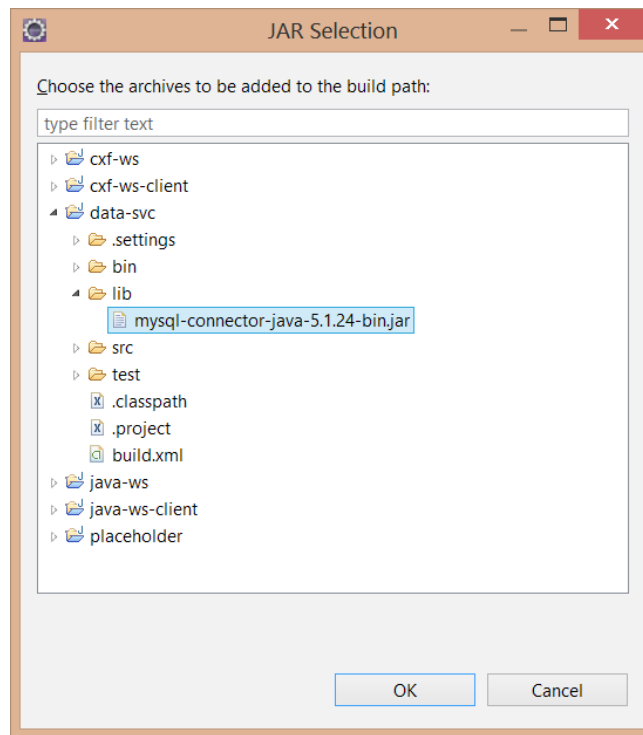


Figure 4-10. JAR Selection screen

Expand data-svc/lib folder. Select the JDBC driver. Then, click OK. Your Java Build Path screen should look like this:

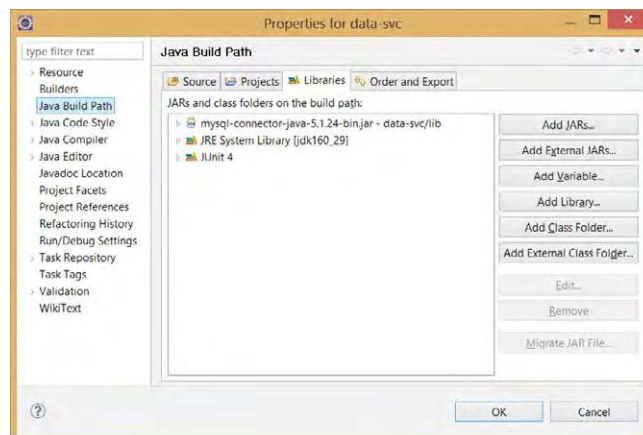


Figure 4-11. Java Build Path

Now, the coding can begin. In the following section, we create two classes – DbConnection.java and EmployeeDao.java.

4.2.2.3 DbConnection.java

In earlier sections, we created the DbConfig.java class to hold the configuration parameters for accessing the database. The next logical step is to create a class to manage all the JDBC connections for this application. Getting a database connection can also be accomplished using DataSource class; however, in this book, we use a basic method for obtaining a JDBC database connection.

Listing 4-8. DbConnection.java class

```
package com.bemach.data;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
```

Excellent Economics and Business programmes at:



university of
 groningen




“The perfect start
 of a successful,
 international career.”

CLICK HERE
 to discover why both socially
 and academically the University
 of Groningen is one of the best
 places for a student to be

www.rug.nl/feb/education



```

public final class DbConnection {
    private static final Logger LOG = Logger.getLogger(DbConnection.class.getName());
    private static final String ERROR_MSG = "ERROR: ";
    private Connection conn = null;

    public Connection getConn() {
        return conn;
    }

    private DbConnection(String driverName, String subprot, String host,
        String port, String db, String uid, String psw) {
        LOG.info("Getting DB connection ...");

        try {
            Class.forName(driverName);
            String url = String.format("jdbc:%s://%s:%s/%s?user=%s&password=%s",
                subprot, host, port, db, uid, psw);
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            LOG.log(Level.SEVERE, ERROR_MSG+e);
        } catch (ClassNotFoundException e) {
            LOG.log(Level.SEVERE, ERROR_MSG+e);
        }
    }

    public static DbConnection getInstance(String driverName, String subprot,
String host,
        String port, String db, String uid, String psw) {
        return new DbConnection(driverName, subprot, host, port, db, uid, psw);
    }

    public void close() {
        try {
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (SQLException e) {
            LOG.log(Level.SEVERE, ERROR_MSG+e);
        }
    }
}

```

The DriverManager class helps create a JDBC connection in three ways.

getConnection(String url)

getConnection(String url, Properties info)

getConnection(String url, String user, String password)

In this example, the first method is used and the URL can be seen as follows:

```
jdbc:mysql://localhost:3306/employees?user=empl_1&password=passwd
```

The `getConnection()` method requires all necessary database configuration parameters to connect to the database. When a JDBC connection is no longer needed, it must be explicitly closed by calling the `closeConn()` method. Accumulated open connections will strain the resources. In most JDBC drivers, closing a connection results in closing the Statement and Result sets that are associated with the connection.

4.2.2.4 *EmployeeDao.java*

This class provides basic access to the `employees2` table in the database. `PreparedStatement` is used to avoid potential SQL injection attack.

Our task is to develop a WS called `EmployeeDataService` that allows a client to create, read, update and delete a row from the `employees` table. For now, we are not concerned with security – we simply want to show how this can be done through a bottom-up approach to create a Web Service.

First, we create a class that allows access to this table. This class is called `EmployeeDao` and allows four basic operations on a row of the `employees` table. An `Employee` class represents each employee from the Java coding. `EmployeeDao` uses basic Java Database Connectivity (JDBC) to create a database connection, issues an SQL statement, and processes the return. It is a basic JDBC application.

Listing 4-9. EmployeeDao.java class

```
package com.bemach.data;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.Calendar;
import java.util.logging.Logger;
```

```

/**
 * This class allows its application to perform the four (4) basic operations
 * of an Employee resource:
 * 1. Create
 * 2. Read
 * 3. Update
 * 4. Delete
 *
 * CRUD is classic in a sense that it is most like what an application does to
 * an authorized resource.
 *
 * Additional methods are:
 * getEmployeeByLastName
 * getEmplByFirstLastName
 *
 */
public class EmployeeDao {
    public static final Logger LOG = Logger.getLogger(EmployeeDao.class.getName());
    private DbConfig cfg = null;

    public EmployeeDao() {

    }

    /**
     * Constructor
     * @param cfg
     */
    public EmployeeDao (DbConfig cfg) {
        this.cfg = cfg;
        LOG.info("Constructing EmployeeDao ...");
    }

    /**
     * From a ResultSet returns an Employee record.
     *
     * @param rs
     * @return
     */
    protected Employee getEmpl(ResultSet rs) throws SQLException {
        Employee empl = new Employee();
        Calendar cal = Calendar.getInstance();
        empl.setEmplNo(rs.getInt("emp_no"));
        cal.setTimeInMillis(rs.getTimestamp("birth_date").getTime());
        empl.setBirthDate(cal);
        empl.setFirstName(rs.getString("first_name"));
        empl.setLastName(rs.getString("last_name"));
        empl.setGender(rs.getString("gender"));
        cal = Calendar.getInstance();
        cal.setTimeInMillis(rs.getTimestamp("hire_date").getTime());
        empl.setHireDate(cal);
        return empl;
    }
}

```

```

/**
 * Create a new employee.
 *
 * @param empl
 * @return
 */
public int createEmpl(Employee empl) throws SQLException {
    LOG.info("Create an employee");
    DbConnection dbConn = DbConnection.getInstance(cfg.getDriverName(),
        cfg.getSubprot(), cfg.getHost(),
        cfg.getPort(), cfg.getDb(),
        cfg.getUid(), cfg.getPsw());

    String sql = "SELECT MAX(EMP_NO) FROM EMPLOYEES";
    PreparedStatement stmt = null;
    ResultSet rs = null;
    try {
        stmt = dbConn.getConn().prepareStatement(sql);
        stmt.execute();
        rs = stmt.getResultSet();
        rs.next();
        int nextEmplNo = rs.getInt(1);
        stmt.close();
        rs.close();

        sql = "INSERT INTO EMPLOYEES (EMP_NO, BIRTH_DATE, FIRST_NAME, LAST_
NAME, GENDER, HIRE_DATE) " +
            "VALUES (?, ?, ?, ?, ?, ?)";
        stmt = dbConn.getConn().prepareStatement(sql);
        int idx = 1;
        stmt.setInt(idx++, ++nextEmplNo);
        Timestamp ts = new Timestamp(empl.getBirthDate().getTimeInMillis());
        stmt.setTimestamp(idx++, ts);
        stmt.setString(idx++, empl.getFirstName());
        stmt.setString(idx++, empl.getLastName());
        stmt.setString(idx++, empl.getGender());
        ts = new Timestamp(empl.getHireDate().getTimeInMillis());
        stmt.setTimestamp(idx++, ts);
        stmt.execute();
        return nextEmplNo;
    } finally {
        if (stmt != null) {
            stmt.close();
        }
        if (rs != null) {
            rs.close();
        }
        dbConn.close();
    }
}

```

```

/**
 * Update an employee record.
 *
 * @param empl
 * @return
 */
public boolean updateEmpl(Employee empl) throws SQLException {
    LOG.info("Update an employee");
    DbConnection dbConn = DbConnection.getInstance(cfg.getDriverName(),
        cfg.getSubprot(), cfg.getHost(),
        cfg.getPort(), cfg.getDb(),
        cfg.getUid(), cfg.getPsw());

    String sql = "UPDATE EMPLOYEES SET BIRTH_DATE=?, FIRST_NAME=?, LAST_NAME=?,
GENDER=?, HIRE_DATE=? " +
                "WHERE EMP_NO=?";
    PreparedStatement stmt = null;

    try {
        stmt = dbConn.getConnection().prepareStatement(sql);
        int idx = 1;
        Timestamp ts = new Timestamp(empl.getBirthDate().getTimeInMillis());
        stmt.setTimestamp(idx++, ts);
        stmt.setString(idx++, empl.getFirstName());
        stmt.setString(idx++, empl.getLastName());
        stmt.setString(idx++, empl.getGender());
        ts = new Timestamp(empl.getHireDate().getTimeInMillis());
        stmt.setTimestamp(idx++, ts);
        stmt.setInt(idx++, (int)empl.getEmplNo());

        stmt.execute();
        return true;
    } finally {
        if (stmt != null) {
            stmt.close();
        }
        dbConn.close();
    }
}

/**
 * Delete an employee by Employee Number
 * @param emplNo
 * @return
 */
public boolean deleteEmpl(int emplNo) throws SQLException {
    LOG.info("Delete an employee");
    DbConnection dbConn = DbConnection.getInstance(cfg.getDriverName(),
        cfg.getSubprot(), cfg.getHost(),
        cfg.getPort(), cfg.getDb(),
        cfg.getUid(), cfg.getPsw());

    String sql = "DELETE FROM EMPLOYEES WHERE EMP_NO=?";
    PreparedStatement stmt = null;

    try {
        stmt = dbConn.getConnection().prepareStatement(sql);
        stmt.setInt(1, emplNo);

        stmt.execute();
        return true;
    } finally {
        if (stmt != null) {
            stmt.close();
        }
        dbConn.close();
    }
}

```

```

/**
 * Get an employee of a given unique employee number ..
 *
 * @param emplNo
 * @return
 */
public Employee getEmpl (int emplNo) throws SQLException {
    LOG.info("Getting employee by Employee number: "+emplNo);
    DbConnection dbConn = DbConnection.getInstance(cfg.getDriverName(),
        cfg.getSubprot(), cfg.getHost(),
        cfg.getPort(), cfg.getDb(),
        cfg.getUid(), cfg.getPsw());

    String sql = "SELECT * FROM EMPLOYEES WHERE EMP_NO=?";
    PreparedStatement stmt = null;
    ResultSet rs = null;
    try {
        stmt = dbConn.getConn().prepareStatement(sql);
        stmt.setInt(1, emplNo);
        if (stmt.execute()) {
            rs = stmt.getResultSet();
            if (rs != null && rs.next()) {
                return getEmpl(rs);
            }
        }
    } finally {
        if (stmt != null) {
            stmt.close();
        }
        if (rs != null) {
            rs.close();
        }
        dbConn.close();
    }
    return null;
}
}

```

This class has four important methods that can create, read, update and delete an employee record from the employees table in the database. These methods will be reflected later through the four operations of the two Web Services called ‘EmployeeDocData’ and ‘EmployeeRpcData’.

4.2.2.4.1 createEmpl(Employee empl)

This method receives a new employee record called ‘empl’ and inserts it into the employees table using a JDBC PreparedStatement class. This method assumes that the callers of this method have already validated the content of the employee record. Another approach is to include a validation method in this class and call from each of the four operations. Notice that all operations dealing with the database are done through a PreparedStatement in order to limit SQL inject attacks from the outside.

First, we get the largest employee number in order to create a new employee record with a unique primary key. This way of getting an employee number may encounter a concurrency problem when another application or process inserts another record at the same time; however, we ignore this condition here for the sake of simplicity.

Once a new employee number has been received, the method inserts the record into the database and a new employee number is returned to the caller; however, the final clause will first make sure that the database connection has been closed. This is one of the many techniques to ensure that resources are properly deallocated after the method completes its task.

4.2.2.4.2 getEmpl(emplNo)

This method retrieves an employee record from the employees table of the database. A unique employee number is a required input. If the record is found, it is returned to the caller. Otherwise, an exception is thrown. Regardless, at this point, the database connection is closed.

4.2.2.4.3 updateEmpl(empl)

This method updates a record with all values from the input record except the employee number. A Boolean value of true or false is returned after the processing is completed. If the record exists and the update completes successfully, a Boolean value of truth is returned. Otherwise, the method returns false.

4.2.2.4.4 deleteEmpl(emplNo)

Similarly to other methods of this class, this method opens a database connection then issues an SQL statement to complete the task. After a successful completion, an employee record will be removed from the table and a Boolean value of truth is returned. Otherwise, the method returns a value of false.

Overall, this class is relatively simple. It performs the most frequent operations on a resource stored in the database. This class is kept simple because our focus is on the creation of Web Services not database operations. When more complex business rules and multi-datasource data access activities are involved, the fundamental concept of Web Services remains the same. The main focus of Web Service is the interface – it must be robust and capable of evolving over time.

In the next section, we briefly discuss how to test the data access object so that we can ensure some basic quality assurance of the development team.

4.2.2.5 JUnit Test for Data Access Object

We provided a basic JUnit test for the EmployeeDao.java class. This class is called 'EmployeeDataTest.java'.

Listing 4-10. *EmployeeDaoTest.java* Class

```
package com.bemach.data.junit;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import static org.junit.Assert.*;

import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.Calendar;
import java.util.logging.Logger;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import com.bemach.data.DbConfig;
import com.bemach.data.Employee;
import com.bemach.data.EmployeeDao;

public class EmployeeDaoTest {
    public static final Logger logger = Logger.getLogger(EmployeeDaoTest.class.getName());

    /**
     * @throws java.lang.Exception
     */
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    /**
     * @throws java.lang.Exception
     */
    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    private EmployeeDao dao;
```

```

/**
 * @throws java.lang.Exception
 */
@Before
public void setUp() throws Exception {
    logger.info("Testing employee dao class ...");
    DbConfig cfg = new DbConfig();
    cfg.setDriverName("com.mysql.jdbc.Driver");
    cfg.setHost("saintmonica");
    cfg.setPort("3306");
    cfg.setDb("employees");
    cfg.setUid("empl_1");
    cfg.setPsw("password");
    dao = new EmployeeDao(cfg);
}

/**
 * @throws java.lang.Exception
 */
@After
public void tearDown() throws Exception {
}

/**
 * Test method for {@link com.bemach.data.EmployeeDao#getEmpl(int)}.
 * @throws SQLException
 */
@Test
public void testGetEmplByEmplNo() throws SQLException {
    Employee empl = dao.getEmpl(10327);
    assertTrue("*** ERROR NULL ***", empl != null);
    logger.info("found "+empl.getFirstName()+"/"+empl.getLastName());
}

@Test
public void testCRUDEmpl() throws SQLException {
    logger.info(">>> get empl");
    Employee empl = dao.getEmpl(10001);
    empl.setFirstName("Test_First");
    empl.setLastName("Test_Last");
    Timestamp ts = Timestamp.valueOf("1970-01-01 0:0:0.0");
    Calendar cal = Calendar.getInstance();
    cal.setTimeInMillis(ts.getTime());
    empl.setBirthDate(cal);
    ts = Timestamp.valueOf("1970-01-01 0:0:0.0");
    cal.setTimeInMillis(ts.getTime());
    empl.setHireDate(cal);
    empl.setGender("F");

    logger.info(">>> create empl");
    int newEmplNo = dao.createEmpl(empl);
    logger.info(">>> get new empl");
    Employee newEmpl = dao.getEmpl(newEmplNo);
}

```



```

newEmpl.setGender("M");
logger.info(">>> update new empl");
dao.updateEmpl(newEmpl);
logger.info(">>> get new empl again");
newEmpl = dao.getEmpl(newEmplNo);
printOutput(newEmpl);
logger.info(">>> delete new empl");
dao.deleteEmpl(newEmplNo);
}

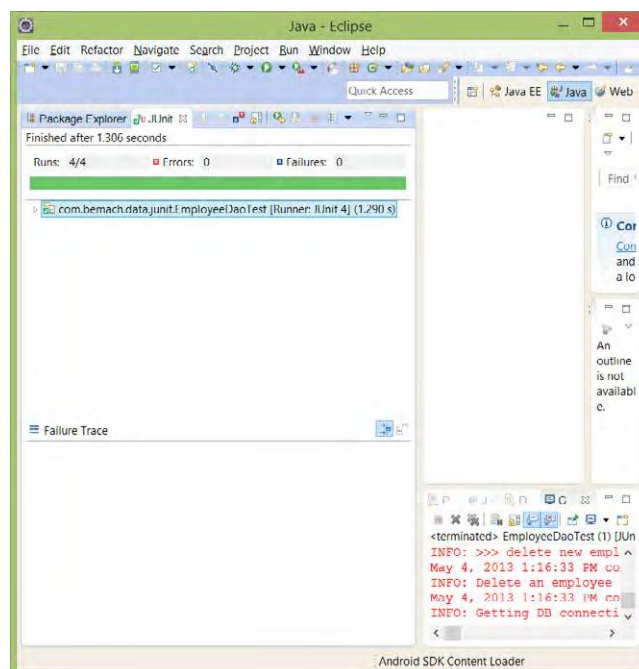
private void printOutput(Employee empl) {
    StringBuffer sb = new StringBuffer();
    sb.append(", emplno=").append(empl.getEmplNo());
    sb.append(", fname=").append(empl.getFirstName());
    sb.append(", lname=").append(empl.getLastName());
    sb.append(", hire=").append(empl.getHireDate());
    sb.append(", birth=").append(empl.getBirthDate());
    sb.append(", gender=").append(empl.getGender());
    logger.info(sb.toString());
}
}

```

After each operation, an employee record is formatted and displayed on the screen.

If all the tests are run successfully, the result should be displayed in green on the JUnit panel on the left-hand side of the Eclipse IDE:

Listing 4-11. JUnit test result



4.2.3 Package a Java Library

In order to run a build from a command line, `JAVA_HOME` and `ANT_HOME` variables need to be defined. Make sure to include `$JAVA_HOME/bin` or `%JAVA_HOME%\bin` in the `PATH` variable. `JAVA_HOME` should be pointed to the installed JDK. We've developed and tested with JDK 1.6. The Ant build script was of version 1.7.1.

This `build.xml` build script by default runs from the root of the project `data-svc` directory. The classes are stored in the `bin` directory, while the Java library `data-svc.jar` will be stored in the `dist` directory. A clean build command will remove both directories. Thus, two build commands should be used:

```
Ant dist (or simply ant)
```

```
Ant clean
```



LIGS University
based in Hawaii, USA

is currently enrolling in the
Interactive Online **BBA, MBA, MSc,**
DBA and PhD programs:

- ▶ enroll **by October 31st, 2014** and
- ▶ **save up to 11%** on the tuition!
- ▶ pay in 10 installments / 2 years
- ▶ Interactive **Online** education
- ▶ visit www.ligsuniversity.com to find out more!

Note: LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education. More info [here](#).



Listing 4-12. *build.xml* for *data-svc* Java Project

```
<project name="data-svc" default="dist" basedir=".">
  <description>
    Data Services
  </description>

  <!-- set global properties for this build -->
  <property environment="env" />
  <path id="classpath.base">
    <fileset dir="./lib" includes="**/*.jar" />
  </path>

  <path id="classpath.compile">
    <path refid="classpath.base" />
  </path>

  <target name="init">
    <mkdir dir="./bin" />
    <mkdir dir="./dist" />
  </target>

  <target name="compile" depends="init" description="compile the source ">
    <javac srcdir="./src" destdir="./bin" debug="true">
      <classpath refid="classpath.compile" />
    </javac>
  </target>

  <target name="dist" depends="compile" description="generate the distribution">
    <!-- Create the distribution directory -->
    <jar jarfile="./dist/data-svc.jar" basedir="./bin" />
  </target>

  <target name="clean" description="clean up">
    <!-- Delete the ${build} directory trees -->
    <delete dir="./dist" />
    <delete dir="./bin" />
  </target>
</project>
```

The output, a `data-svc.jar` file, is stored in the `dist` directory. The content of this JAR file should be as follows:

```
META-INF/
META-INF/MANIFEST.MF
com/
com/bemach/
com/bemach/data/
```

```

com/bemach/data/DbConfig.class
com/bemach/data/DbConnection.class
com/bemach/data/Employee.class
com/bemach/data/EmployeeDao.class

```

From the standpoint of business, EmployeeDao should be tested to ensure that it works at the level of the basic unit. All operations of the classes were thoroughly tested to ensure that the classes work.

4.2.4 Develop Java Classes for Web Services

To create a Java project under Eclipse IDE, please refer to Chapter 7. Import two libraries (i.e., data-svc.jar and MySQL JDBC driver) into the lib folder under java-ws project. Also, make sure to have these libraries in the Java Build Path. Refer to the previous section for instructions on how to make reference to the libraries for a Java project in Eclipse.

We develop a Web Service for employee with two different styles: document and RPC. First, we create a Java project called 'java-ws'. After we finish coding the required Java classes, the java-ws project should appear as follows:

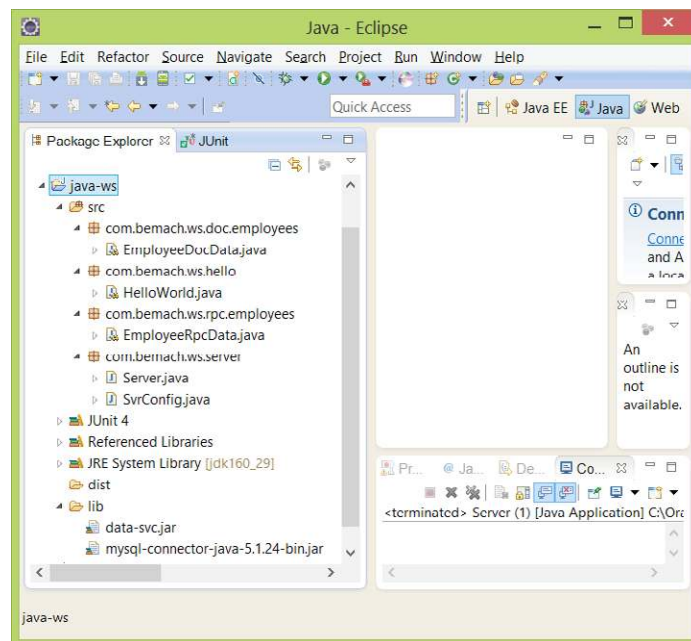


Figure 4-12. java-ws Java Project

The following sections describe the steps required to create Web Services in two main Java classes – EmployeeDocData.java and EmployeeRpcData.java

4.2.4.1 *EmployeeDocData.java*

Writing Web Service in Java can be done by incorporating Java annotation into Java classes. These classes provide WS using SOAP. Java WS annotations that are used include the following:

- `@WebService`: indicates this class to implement a Web Service
- `@SOAPBinding`: specifies Web Service to bind to a SOAP protocol
- `@WebMethod`: exposes an operation as a Web method.
- `@WebParam`: maps individual parameters to a WS message.

A document-style SOAP binding is used for this application.



.....Alcatel-Lucent 

www.alcatel-lucent.com/careers

What if you could build your future and create the future?

One generation's transformation is the next's status quo. In the near future, people may soon think it's strange that devices ever had to be "plugged in." To obtain that status, there needs to be "The Shift".



Listing 4-13. *EmployeeDocData.java* Class

```

package com.bemach.ws.doc.employees;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.util.logging.Logger;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.soap.SOAPException;

import com.bemach.data.DbConfig;
import com.bemach.data.Employee;
import com.bemach.data.EmployeeDao;

@WebService
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
public class EmployeeDocData {
    private static final Logger LOG =
    Logger.getLogger(EmployeeDocData.class.getName());
    private EmployeeDao dao = null;

    public EmployeeDocData (DbConfig cfg) {
        dao = new EmployeeDao(cfg);
    }

    @WebMethod
    public Employee getEmployee(@WebParam(name="emplNo") long emplNo) throws
    SOAPException, Exception {
        LOG.info("Doc.readEmployee");
        Employee employee = dao.getEmpl((int)emplNo);
        if (employee == null) {
            throw new SOAPException ("No such employee!");
        }
        return employee;
    }

    @WebMethod
    public long createEmployee(@WebParam (name="employee")Employee employee) throws
    Exception {
        LOG.info("Doc.createEmployee");
        return dao.createEmpl(employee);
    }

    @WebMethod
    public boolean updateEmployee(@WebParam (name="employee")Employee employee)
    throws Exception {
        LOG.info("Doc.updateEmployee.");
        return dao.updateEmpl(employee);
    }

    @WebMethod
    public boolean deleteEmployee(@WebParam(name="emplNo") long emplNo) throws
    Exception {
        LOG.info("Doc.deleteEmployee.");
        return dao.deleteEmpl((int)emplNo);
    }
}

```



This class has four major operations (i.e., CRUD) on an employee record. Note that for each operation, the service creates an `EmployeeDao` object to call the matching operation. The call is then returned as the return of the operation of the service. Note that the marshalling of the return object is accomplished with the assistance of the JAXB component in Java.

These four operations are simple. Each method calls the corresponding method provided by `EmployeeDao` instance.

4.2.4.1.1 @WebService Annotation

`@WebService` annotation indicates that this class (or an interface) implements a Web Service. This annotation has six (6) optional elements that can be used for a more detailed definition of a Web Service:

1. `endpointInterface`: the complete name of the service endpoint interface
2. `name`: the name of the `<portType>` element within the WSDL
3. `portName`: the name of the `<port>` element within the WSDL
4. `serviceName`: the name of the `<service>` element within the WSDL
5. `targetNamespace`: the `targetNamespace` attribute of the `<definition>` element of the WSDL
6. `wsdlLocation`: the content of the `location` attribute of the `<soap:address>` element

In this application, we did not include these optional elements. We will define the location of the WSDL when we create an `Endpoint` within the server code (`Server.java`).

4.2.4.1.2 @SOAPBinding Annotation

This annotation specifies how to map a Web Service onto the SOAP message protocol. These involve three optional elements:

1. `parameterStyle`: This can be either BARE or WRAPPED.
2. `style`: This can be either DOCUMENT or RPC
3. `use`: This can be either LITERAL or ENCODED.

In this sample application, we use DOCUMENT and RPC styles for two separate Web Services.

4.2.4.1.3 @WebMethod Annotation

This annotation customizes a method that is exposed as a WS operation. There are three (3) optional elements that can be used with this annotation:

1. `action`: name of an operation defined within the WSDL
2. `exclude`: excludes the method from being exposed as an operation of a Web Service
3. `operationName`: name of the operation.

4.2.4.1.4 @WebParam Annotation

Individual parameters of an operation can be named in the same way as the method. Use this annotation to change to different names within the WSDL. Optional parameters are:

1. `header`: if true, the parameter is extracted from the message header instead of from the message body.
2. `mode`: there are three basic modes – IN, OUT, and INOUT.
3. `name`: the parameter is mapped to name in XML element that represents the parameter. If DOCUMENT style is used, name is required.
4. `partName`: if RPC style is used, this is the name in the `wsdl:part` element.
5. `targetNamespace`: if DOCUMENT style is used, the parameter maps to a header.

4.2.4.2 *EmployeeRpcData.java*

This class implements the SOAP RPC style of Web Service. It is nearly identical to that of the document style with the exception of SOAPBinding annotation. This class is used to show the difference between the two styles in use today.

Listing 4-14. *EmployeeRpcData.java* Class

```

package com.bemach.ws.rpc.employees;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.util.logging.Logger;

import javax.ws.WebMethod;
import javax.ws.WebParam;
import javax.ws.WebService;
import javax.ws.soap.SOAPBinding;
import javax.xml.soap.SOAPException;

import com.bemach.data.DbConfig;
import com.bemach.data.Employee;
import com.bemach.data.EmployeeDao;


@WebService
@SOAPBinding(style=SOAPBinding.Style.RPC)
public class EmployeeRpcData {
    private static final Logger LOG = Logger.getLogger(EmployeeRpcData.class.
getName());
    private EmployeeDao dao = null;

    public EmployeeRpcData (DbConfig cfg) {
        dao = new EmployeeDao(cfg);
    }

    @WebMethod
    public Employee getEmployee(@WebParam(name="emplNo") long emplNo) throws
SOAPException, Exception {
        LOG.info("Rpc.readEmployee");
        Employee employee = getDao().getEmpl((int)emplNo);
        if (employee == null) {
            throw new SOAPException ("No such employee!");
        }
        return employee;
    }

    @WebMethod
    public long createEmployee(@WebParam (name="employee")Employee employee)
throws Exception {
        LOG.info("Rpc.createEmployee");
        return getDao().createEmpl(employee);
    }
}

```



```
@WebMethod
public boolean updateEmployee(@WebParam (name="employee")Employee employee)
throws Exception {
    LOG.info("Rpc.updateEmployee.");
    return getDao().updateEmpl(employee);
}

@WebMethod
public boolean deleteEmployee(@WebParam(name="emplNo")long emplNo) throws
Exception {
    LOG.info("Rpc.deleteEmployee.");
    return getDao().deleteEmpl((int)emplNo);
}

public EmployeeDao getDao() {
    return dao;
}

public void setDao(EmployeeDao dao) {
    this.dao = dao;
}
}
```

4.2.5 Hosting Web Services

Web Services need to be hosted by a server that provides some basic HTTP service endpoints. Note that this type of server is rather simplistic in its implementation for the purpose of WS demonstration. A more industrial-strength application server, such as WebLogic, JBOSS, or WebSphere, is more appropriate for medium-sized to large business settings.

4.2.5.1 *Server.java*

This class implements a HTTP server to host multiple Web Services (e.g., HelloWorld, EmployeeDocDataService and EmployeeRpcDataService). Each WS is uniquely identified with a service endpoint.

- HelloWorld Web Service: <http://localhost:9999/java-ws/hello?WSDL>
- Employee Document Web Service: <http://localhost:9999/doc/employees?wsdl>
- Employee RPC Web Service: <http://localhost:9999/rpc/employees?wsdl>

Listing 4-15. Server.java Class

```
package com.bemach.ws.server;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.util.logging.Logger;

import javax.xml.ws.Endpoint;
import javax.xml.ws.EndpointReference;

import com.bemach.data.DbConfig;
import com.bemach.ws.doc.employees.EmployeeDocData;
import com.bemach.ws.hello.HelloWorld;
import com.bemach.ws.rpc.employees.EmployeeRpcData;

/**
 *
 */
public final class Server {
    private static final Logger LOG = Logger.getLogger(Server.class.getName());
    private static final String MYSQL_DRIVER="com.mysql.jdbc.Driver";
    private static final String DB_HOST = "saintmonica";
    private static final String DB_PORT = "3306";
    private static final String DB_SID = "employees";
    private static final String DB_USER = "empl_1";
    private static final String DB_PSW = "password";
    private Server() {
    }

    protected static DbConfig getDbConfig() {
        DbConfig dbCfg = new DbConfig();
        dbCfg.setDriverName(MYSQL_DRIVER);
        dbCfg.setHost(DB_HOST);
        dbCfg.setPort(DB_PORT);
        dbCfg.setDb(DB_SID);
        dbCfg.setUid(DB_USER);
        dbCfg.setPsw(DB_PSW);
        return dbCfg;
    }
}
```

```

private static final String HOST_NAME = "localhost";
private static final String PORT_NO = "9999";
private static final String HELLO_SVC_NAME = "java-ws/hello";
private static final String RPC_EMPL_SVC_NAME = "rpc/employees";
private static final String DOC_EMPL_SVC_NAME = "doc/employees";
private static final String PROTOCOL = "http";

protected static SvrConfig getSvrConfig() {
    SvrConfig svrCfg = new SvrConfig();
    svrCfg.setListenHostname(HOST_NAME);
    svrCfg.setListenPort(PORT_NO);
    svrCfg.setListenInterface(HELLO_SVC_NAME);
    svrCfg.setListenProtocol(PROTOCOL);
    return svrCfg;
}

protected static Endpoint publish(SvrConfig cfg, Object svc) {
    String url = String.format("%s://%s:%s/%s",
        cfg.getListenProtocol(),
        cfg.getListenHostname(),
        cfg.getListenPort(),
        cfg.getListenInterface());
    Endpoint ep = Endpoint.publish(url, svc);
    EndpointReference epr = ep.getEndpointReference();
    LOG.info("\nEndpoint Ref:\n"+epr.toString());
    return ep;
}

protected static void startHelloWorld() {
    SvrConfig cfg = getSvrConfig();
    cfg.setListenHostname(HOST_NAME);
    cfg.setListenInterface(HELLO_SVC_NAME);
    cfg.setListenPort(PORT_NO);
    cfg.setListenProtocol(PROTOCOL);

    HelloWorld hello = new HelloWorld();
    publish(cfg, hello);
    LOG.info("HelloWorld service started successfully ...");
}

protected static void startRpcEmployees() {
    SvrConfig svrCfg = getSvrConfig();
    svrCfg.setListenHostname(HOST_NAME);
    svrCfg.setListenInterface(RPC_EMPL_SVC_NAME);
    svrCfg.setListenPort(PORT_NO);
    svrCfg.setListenProtocol(PROTOCOL);
    DbConfig dbCfg = getDbConfig();
    svrCfg.setDbCfg(dbCfg);

    EmployeeRpcData rpcEmpl = new EmployeeRpcData(dbCfg);
    publish(svrCfg, rpcEmpl);
    LOG.info("Employees (RPC) service started successfully ...");
}

```

```
protected static void startDocEmployees() {
    SvrConfig svrCfg = getSvrConfig();
    svrCfg.setListenHostname(HOST_NAME);
    svrCfg.setListenInterface(DOC_EMPL_SVC_NAME);
    svrCfg.setListenPort(PORT_NO);
    svrCfg.setListenProtocol(PROTOCOL);
    DbConfig dbCfg = getDbConfig();
    svrCfg.setDbCfg(dbCfg);

    EmployeeDocData docEmpl = new EmployeeDocData(dbCfg);
    publish(svrCfg, docEmpl);

    LOG.info("Employees (Document) service started successfully ...");
}

/**
 * Start WS Server with multiple service endpoints...
 *
 * @param args
 */
public static void main(String[] args) {
    startHelloWorld();
    startRpcEmployees();
    startDocEmployees();
}
}
```

4.2.5.2 Package the Web Services

This Ant build script builds the java-ws.jar library and stores it in the dist directory. This build requires two Java libraries: data-svc.jar and mysql-connector-java-5.1.24-bin.jar.

Listing 4-16. *build.xml* for java-ws Java Project

```
<project name="java-ws" default="dist" basedir=".">
  <description>
    Web Service usign Java.
  </description>

  <!-- set global properties for this build -->
  <property environment="env" />
  <path id="classpath.base">
    <fileset dir="./lib" includes="**/*.jar" />
  </path>

  <path id="classpath.compile">
    <path refid="classpath.base" />
  </path>

  <target name="init">
    <mkdir dir="./bin" />
    <mkdir dir="./dist" />
  </target>

  <target name="compile" depends="init" description="compile the source ">
    <javac srcdir="./src" destdir="./bin" debug="true">
      <classpath refid="classpath.compile" />
    </javac>
  </target>

  <target name="dist" depends="compile" description="generate the distribution">
    <!-- Create the distribution directory -->
    <jar jarfile="./dist/java-ws.jar" basedir="./bin" />
  </target>

  <target name="clean" description="clean up">
    <!-- Delete the ${build} directory trees -->
    <delete dir="./dist" />
    <delete dir="./bin" />
  </target>
</project>
```

The output of this build is a JAR file stored in the dist directory. The contents of this library consist of the following elements:

```
META-INF/
META-INF/MANIFEST.MF
com/
com/bemach/
com/bemach/ws/
```

```

com/bemach/ws/doc/
com/bemach/ws/doc/employees/
com/bemach/ws/hello/
com/bemach/ws/rpc/
com/bemach/ws/rpc/employees/
com/bemach/ws/server/
com/bemach/ws/doc/employees/EmployeeDocData.class
com/bemach/ws/hello/HelloWorld.class
com/bemach/ws/rpc/employees/EmployeeRpcData.class
com/bemach/ws/server/Server.class
com/bemach/ws/server/SvrConfig.class

```

4.3 Deploy Web Services

The server instance runs indefinitely. Use control-C to terminate the process. An alternative way to get the configuration parameters is to load them from a Java properties file. Note that, for Windows, the CLASSPATH separator is semi-colon (;) as opposed to colon (:) on UNIX.

Join the best at the Maastricht University School of Business and Economics!

Top master's programmes

- 33rd place Financial Times worldwide ranking: MSc International Business
- 1st place: MSc International Business
- 1st place: MSc Financial Economics
- 2nd place: MSc Management of Learning
- 2nd place: MSc Economics
- 2nd place: MSc Econometrics and Operations Research
- 2nd place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

Visit us and find out why we are the best!
Master's Open Day: 22 February 2014

Maastricht University is the best specialist university in the Netherlands (Elsevier)

www.mastersopenday.nl



```
java -cp ./dist/java-ws.jar;../data-svc/dist/data-svc.jar;../lib/mysql-connector-java-5.1.24-bin.jar com.bemach.ws.server.Server
```

`mysql-connector-java-5.1.24-bin.jar` is a JDBC driver for MySQL database.

Next, we use SOAP to test the Web Services.

4.4 Check WSDL and XSD

We produce three services with three distinct service endpoints. After the server is running, we verify that these service endpoints are active and ready for service invocations. From a browser, we go to the URLs. The service endpoint for the HelloWorld example, <http://localhost:9999/java-ws/hello?WSDL>, was examined in earlier chapters. We visit the two `employees` service endpoints:

4.4.5.1 Document style

WSDL and XSD of the `employees` Web Service are shown in the following listings. A client application developer uses these WSDL documents to generate a Web Service stub for use inside their application.

Listing 4-17. WSDL of a DOCUMENT Style

```

<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://employees.doc.ws.bemach.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://employees.doc.ws.bemach.com/"
  name="EmployeeDocDataService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://employees.doc.ws.bemach.com/"
        schemaLocation="http://localhost:9999/doc/employees?xsd=1" />
    </xsd:schema>
    <xsd:schema>
      <xsd:import namespace="http://bemach.com"
        schemaLocation="http://localhost:9999/doc/employees?xsd=2" />
    </xsd:schema>
  </types>
  <message name="getEmployee">
    <part name="parameters" element="tns:getEmployee" />
  </message>
  <message name="getEmployeeResponse">
    <part name="parameters" element="tns:getEmployeeResponse" />
  </message>
  <message name="SOAPException">
    <part name="fault" element="tns:SOAPException" />
  </message>
  <message name="createEmployee">
    <part name="parameters" element="tns:createEmployee" />
  </message>
  <message name="createEmployeeResponse">
    <part name="parameters" element="tns:createEmployeeResponse" />
  </message>
  <message name="updateEmployee">
    <part name="parameters" element="tns:updateEmployee" />
  </message>
  <message name="updateEmployeeResponse">
    <part name="parameters" element="tns:updateEmployeeResponse" />
  </message>
  <message name="deleteEmployee">
    <part name="parameters" element="tns:deleteEmployee" />
  </message>
  <message name="deleteEmployeeResponse">
    <part name="parameters" element="tns:deleteEmployeeResponse" />
  </message>
  <portType name="EmployeeDocData">
    <operation name="getEmployee">
      <input message="tns:getEmployee" />
      <output message="tns:getEmployeeResponse" />
      <fault message="tns:SOAPException" name="SOAPException" />
    </operation>
    <operation name="createEmployee">
      <input message="tns:createEmployee" />
      <output message="tns:createEmployeeResponse" />
    </operation>

```

```

<operation name="updateEmployee">
  <input message="tns:updateEmployee" />
  <output message="tns:updateEmployeeResponse" />
</operation>
<operation name="deleteEmployee">
  <input message="tns:deleteEmployee" />
  <output message="tns:deleteEmployeeResponse" />
</operation>
</portType>
<binding name="EmployeeDocDataPortBinding" type="tns:EmployeeDocData">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="getEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
    <fault name="SOAPException">
      <soap:fault name="SOAPException" use="literal" />
    </fault>
  </operation>
  <operation name="createEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
  <operation name="updateEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
  <operation name="deleteEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>

```



```
<service name="EmployeeDocDataService">
  <port name="EmployeeDocDataPort" binding="tns:EmployeeDocDataPortBinding">
    <soap:address location="http://localhost:9999/doc/employees" />
  </port>
</service>
</definitions>
```

Consider the operation `getEmployee` (highlighted). This operation has one input and one output element. These elements are defined in the message area above. These messages are `getEmployee` and `getEmployeeResponse`, which are of `tns:getEmployee` and `tns:getEmployeeResponse` types, respectively. The types are defined in the schema located at <http://localhost:9999/doc/employees?xsd=1>. See highlighted area.

URL for associated schema:



> Apply now

REDEFINE YOUR FUTURE
**AXA GLOBAL GRADUATE
PROGRAM 2015**

redefining / standards 

agence cdg © Photomistop



Listing 4-18. Schema (XSD) of a Web Service

```

<xs:schema xmlns:tns="http://employees.doc.ws.bemach.com/"
  xmlns:ns1="http://bemach.com" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  version="1.0" targetNamespace="http://employees.doc.ws.bemach.com/">
  <xs:import namespace="http://bemach.com"
    schemaLocation="http://localhost:9999/doc/employees?xsd=2" />
  <xs:element name="SOAPException" type="tns:SOAPException" />
  <xs:element name="createEmployee" type="tns:createEmployee" />
  <xs:element name="createEmployeeResponse" type="tns:createEmployeeResponse" />
  <xs:element name="deleteEmployee" type="tns:deleteEmployee" />
  <xs:element name="deleteEmployeeResponse" type="tns:deleteEmployeeResponse" />
  <xs:element name="getEmployee" type="tns:getEmployee" />
  <xs:element name="getEmployeeResponse" type="tns:getEmployeeResponse" />
  <xs:element name="updateEmployee" type="tns:updateEmployee" />
  <xs:element name="updateEmployeeResponse" type="tns:updateEmployeeResponse" />
  <xs:complexType name="deleteEmployee">
    <xs:sequence>
      <xs:element name="emplNo" type="xs:long" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="deleteEmployeeResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="createEmployee">
    <xs:sequence>
      <xs:element name="employee" type="tns:employee" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="employee">
    <xs:sequence>
      <xs:element name="emplNo" type="xs:long" />
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="lastName" type="xs:string" />
      <xs:element name="birthDate" type="xs:dateTime" />
      <xs:element name="gender" type="xs:string" />
      <xs:element name="hireDate" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="createEmployeeResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:long" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getEmployee">
    <xs:sequence>
      <xs:element name="emplNo" type="xs:long" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="getEmployeeResponse">
    <xs:sequence>
      <xs:element name="return" type="tns:employee" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

```

```

<xs:complexType name="SOAPException">
  <xs:sequence>
    <xs:element name="message" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="updateEmployee">
  <xs:sequence>
    <xs:element name="employee" type="tns:employee" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="updateEmployeeResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:boolean" />
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

4.4.5.2 RPC Style

The difference between the WSDLs of RPC and Document styles can be difficult to detect; however, XSDs are visibly different. All data types for the document style are defined using XML schema, while all the simple data types (e.g., integer, long, string) are defined within the WSDL.



The image shows the BI Norwegian Business School logo, which is a central blue square with 'BI' in white, surrounded by a colorful, multi-directional burst of lines. Various business disciplines are listed around the logo: Business, Strategic Marketing Management, International Business, Leadership & Organisational Psychology, Shipping Management, and Financial Economics.

Empowering People. Improving Business.

BI Norwegian Business School is one of Europe's largest business schools welcoming more than 20,000 students. Our programmes provide a stimulating and multi-cultural learning environment with an international outlook ultimately providing students with professional skills to meet the increasing needs of businesses.

BI offers four different two-year, full-time Master of Science (MSc) programmes that are taught entirely in English and have been designed to provide professional skills to meet the increasing need of businesses. The MSc programmes provide a stimulating and multi-cultural learning environment to give you the best platform to launch into your career.

- MSc in Business
- MSc in Financial Economics
- MSc in Strategic Marketing Management
- MSc in Leadership and Organisational Psychology

BI NORWEGIAN BUSINESS SCHOOL

EFMD **EQUIS ACCREDITED**

www.bi.edu/master



Listing 4-19. WSDL of a RPC Style

```

<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://employees.rpc.ws.bemach.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://employees.rpc.ws.bemach.com/"
  name="EmployeeRpcDataService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://employees.rpc.ws.bemach.com/"
        schemaLocation="http://localhost:9999/rpc/employees?xsd=1" />
    </xsd:schema>
    <xsd:schema>
      <xsd:import namespace="http://bemach.com"
        schemaLocation="http://localhost:9999/rpc/employees?xsd=2" />
    </xsd:schema>
  </types>
  <message name="getEmployee">
    <part name="emplNo" type="xsd:long" />
  </message>
  <message name="getEmployeeResponse">
    <part name="return" type="tns:employee" />
  </message>
  <message name="SOAPException">
    <part name="fault" element="tns:SOAPException" />
  </message>
  <message name="createEmployee">
    <part name="employee" type="tns:employee" />
  </message>
  <message name="createEmployeeResponse">
    <part name="return" type="xsd:long" />
  </message>
  <message name="updateEmployee">
    <part name="employee" type="tns:employee" />
  </message>
  <message name="updateEmployeeResponse">
    <part name="return" type="xsd:boolean" />
  </message>
  <message name="deleteEmployee">
    <part name="emplNo" type="xsd:long" />
  </message>
  <message name="deleteEmployeeResponse">
    <part name="return" type="xsd:boolean" />
  </message>
  <portType name="EmployeeRpcData">
    <operation name="getEmployee">
      <input message="tns:getEmployee" />
      <output message="tns:getEmployeeResponse" />
      <fault message="tns:SOAPException" name="SOAPException" />
    </operation>
    <operation name="createEmployee">
      <input message="tns:createEmployee" />
      <output message="tns:createEmployeeResponse" />
    </operation>
  </portType>
</definitions>

```

```

<operation name="updateEmployee">
  <input message="tns:updateEmployee" />
  <output message="tns:updateEmployeeResponse" />
</operation>
<operation name="deleteEmployee">
  <input message="tns:deleteEmployee" />
  <output message="tns:deleteEmployeeResponse" />
</operation>
</portType>
<binding name="EmployeeRpcDataPortBinding" type="tns:EmployeeRpcData">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="rpc" />
  <operation name="getEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </input>
    <output>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </output>
    <fault name="SOAPException">
      <soap:fault name="SOAPException" use="literal" />
    </fault>
  </operation>
  <operation name="createEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </input>
    <output>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </output>
  </operation>
  <operation name="updateEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </input>
    <output>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </output>
  </operation>
  <operation name="deleteEmployee">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </input>
    <output>
      <soap:body use="literal" namespace="http://employees.rpc.ws.bemach.com/" />
    </output>
  </operation>
</binding>

```



```

<service name="EmployeeRpcDataService">
  <port name="EmployeeRpcDataPort" binding="tns:EmployeeRpcDataPortBinding">
    <soap:address location="http://localhost:9999/rpc/employees" />
  </port>
</service>
</definitions>

```

Unlike the document style, the XSD documents for the RPC style are kept simple. Most of the basic data types are defined inside the WSDL instead of in the XSD.

Listing 4-20. XSD of a Web Service (RPC)

```

<xs:schema xmlns:tns="http://employees.rpc.ws.bemach.com/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
  targetNamespace="http://employees.rpc.ws.bemach.com/">
  <xs:element name="SOAPException" type="tns:SOAPException" />
  <xs:complexType name="employee">
    <xs:sequence>
      <xs:element name="emplNo" type="xs:long" />
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="lastName" type="xs:string" />
      <xs:element name="birthDate" type="xs:dateTime" />
      <xs:element name="gender" type="xs:string" />
      <xs:element name="hireDate" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SOAPException">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

Listing 4-21. An Additional XSD of a Web Service

```

<xs:schema xmlns:ns1="http://employees.rpc.ws.bemach.com/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
  targetNamespace="http://bemach.com">
  <xs:import namespace="http://employees.rpc.ws.bemach.com/"
    schemaLocation="http://localhost:9999/rpc/employees?xsd=1" />
  <xs:element name="EmployeeService" type="ns1:employee" />
</xs:schema>

```

4.5 Test Web Services with SOAPUI

First, create a SOAPUI project for each of the WS endpoints. Then, run the operation of each Web Service.

4.5.1 SOAPUI projects

Web Service can be tested using SOAPUI, which is an open source cross-platform functional testing tool that can be used to test Web Services. Like Eclipse, SOAPUI is organized into projects. Each project usually manages one service endpoint. Each service endpoint contains one or more operations that can be called from a client machine. In order to test a Web Service, all you really need is a service endpoint URL provided by your service provider.

The following figure shows how to create a SOAPUI test project for the employees data service with document style at this service endpoint: <http://localhost:9999/doc/employees?WSDL>.

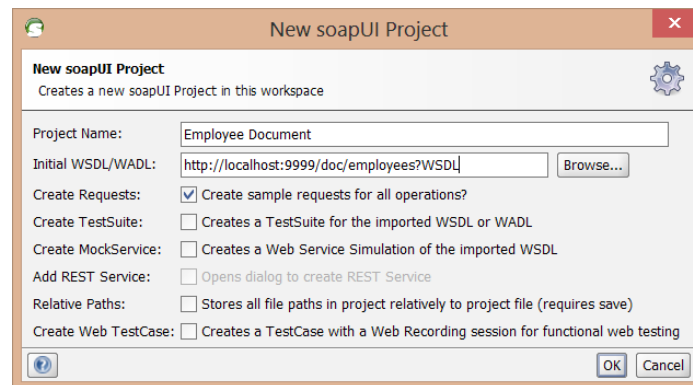


Figure 4-13. Create a SOAPUI Project

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now



Go to www.helpmyassignment.co.uk for more info

 **Helpmyassignment**

Once the project has been created, a set of operations will appear, as shown in the following figures. Note that you can now start testing these WS operations. In our example, four operations are visible: createEmployee, deleteEmployee, getEmployee, and updateEmployee.

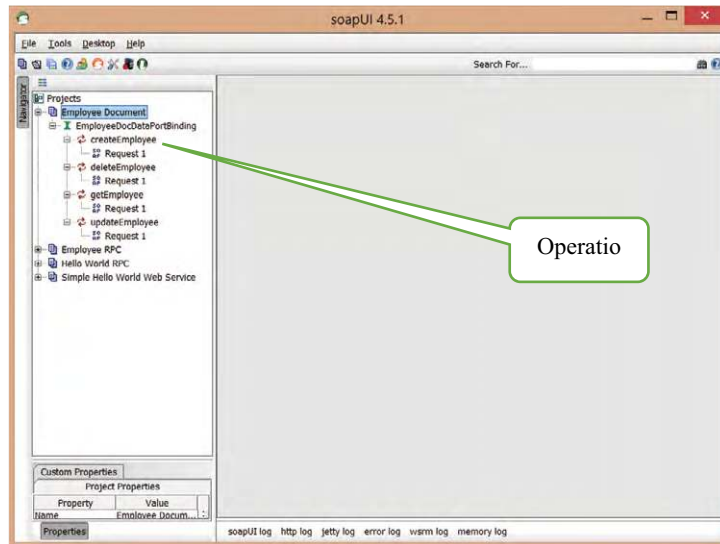


Figure 4-14. List of Oerations of a Web Service

Double-clicking on Request1 of the createEmployee operation will cause a multi-pane window to be displayed. You can fill in the parameters and run the test by clicking on the green triangle to the left panel.

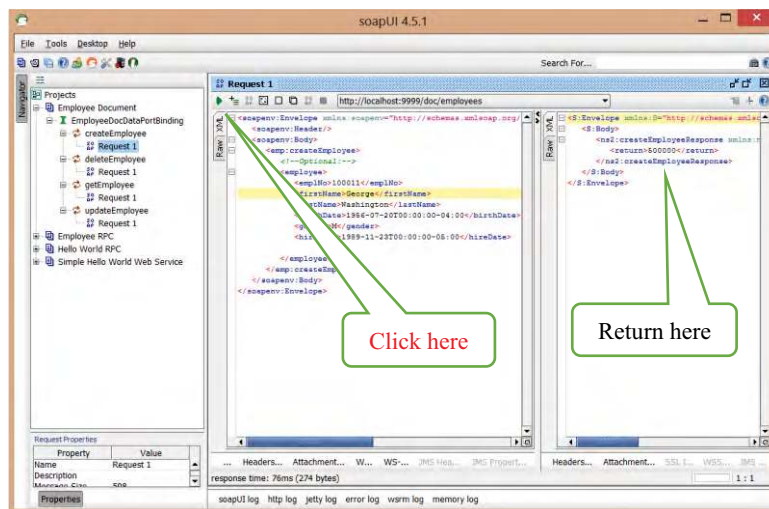


Figure 4-15. Execute a SOAP Operations

Other operations of the service can be tested in the same way.

Web Service with an RPC style can be tested in a similar way. The only difference is that when you create a SOAPUI project, you provide a different service endpoint (URL). A SOAPUI project can be created for EmployeeData service with RPC style as follows:

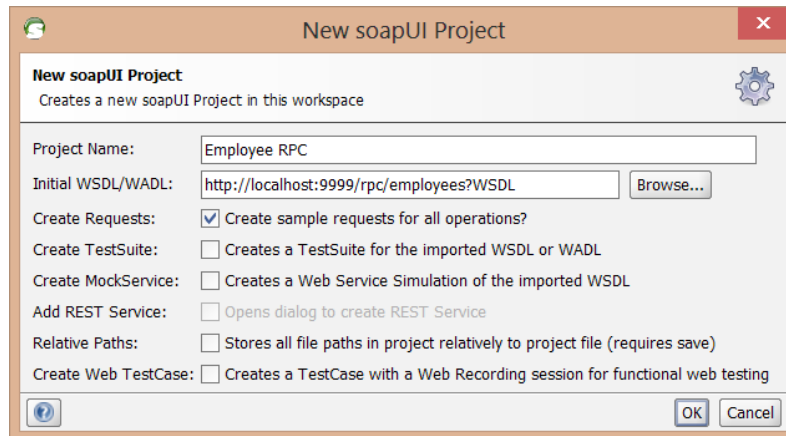


Figure 4-16. Create a new SOAPUI Project

4.6 Develop a Web Service Consumer

Developing a WS consumer (or client) involves three major activities: creating a client stub, creating a client code that uses the client stub to call service operations, and running the client. These activities are described as follows.



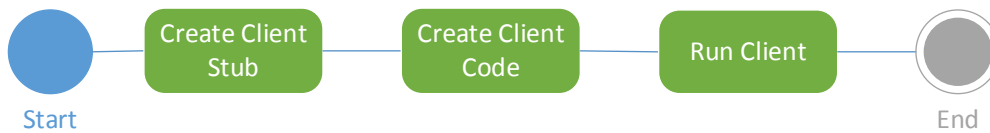


Figure 4-17. Activities for Creating a Web Service Client

4.6.1 Creating Client Stub with wsimport

Writing a SOAP client with SAAJ can be complex and time-consuming. See section 2.4 for details. Instead, we will use the wsimport tool to generate WS artifacts (stubs).

First, create a Java Project under Eclipse IDE and call it ‘java-ws-client’.

1. At the command prompt, go to the Java Project for Eclipse called ‘java-ws-client’.
2. Create a folder called ‘generated’.
3. To generate WS stubs, run the following commands:

```
wsimport -d . http://localhost:9999/doc/employees?WSDL
wsimport -d . http://localhost:9999/rpc/employees?WSDL
wsimport -d . http://localhost:9999/java-ws/hello?WSDL
```

4. To create a Java library, run the following command:

```
jar -cvf ../lib/java-ws-generated.jar *
```

5. To verify the content of the created jar, run this command:

```
jar -tf ../lib/java-ws-generated.jar *
```

The content of the library should appear as follows:

```
META-INF/
META-INF/MANIFEST.MF
com/
com/bemach/
com/bemach/ObjectFactory.class
com/bemach/ws/
com/bemach/ws/doc/
com/bemach/ws/doc/employees/
com/bemach/ws/doc/employees/CreateEmployee.class
```

```
com/bemach/ws/doc/employees/CreateEmployeeResponse.class
com/bemach/ws/doc/employees/DeleteEmployee.class
com/bemach/ws/doc/employees/DeleteEmployeeResponse.class
com/bemach/ws/doc/employees/Employee.class
com/bemach/ws/doc/employees/EmployeeDocData.class
com/bemach/ws/doc/employees/EmployeeDocDataService.class
com/bemach/ws/doc/employees/GetEmployee.class
com/bemach/ws/doc/employees/GetEmployeeResponse.class
com/bemach/ws/doc/employees/ObjectFactory.class
com/bemach/ws/doc/employees/package-info.class
com/bemach/ws/doc/employees/SOAPException.class
com/bemach/ws/doc/employees/SOAPException_Exception.class
com/bemach/ws/doc/employees/UpdateEmployee.class
com/bemach/ws/doc/employees/UpdateEmployeeResponse.class
com/bemach/ws/hello/
com/bemach/ws/hello/HelloWorld.class
com/bemach/ws/hello/HelloWorldService.class
com/bemach/ws/hello/ObjectFactory.class
com/bemach/ws/hello/package-info.class
com/bemach/ws/hello/Say.class
com/bemach/ws/hello/SayResponse.class
com/bemach/ws/rpc/
com/bemach/ws/rpc/employees/
com/bemach/ws/rpc/employees/Employee.class
com/bemach/ws/rpc/employees/EmployeeRpcData.class
com/bemach/ws/rpc/employees/EmployeeRpcDataService.class
com/bemach/ws/rpc/employees/ObjectFactory.class
com/bemach/ws/rpc/employees/package-info.class
com/bemach/ws/rpc/employees/SOAPException.class
com/bemach/ws/rpc/employees/SOAPException_Exception.class
```

These commands generate java binary code that can become part of a client program that calls Web Services. Next, we create a Java library that contains the generated code: `java-ws-generated.jar`. This library should be included as part of a library set for the Eclipse IDE. It is also a part of the Java CLASSPATH during execution.

4.6.2 Create Client Code

We present two types of client code – document style and RPC style. Both are commonly used in WS programming today; however, document style is preferred.

After the coding has been completed, the java-ws-client project should look like this:

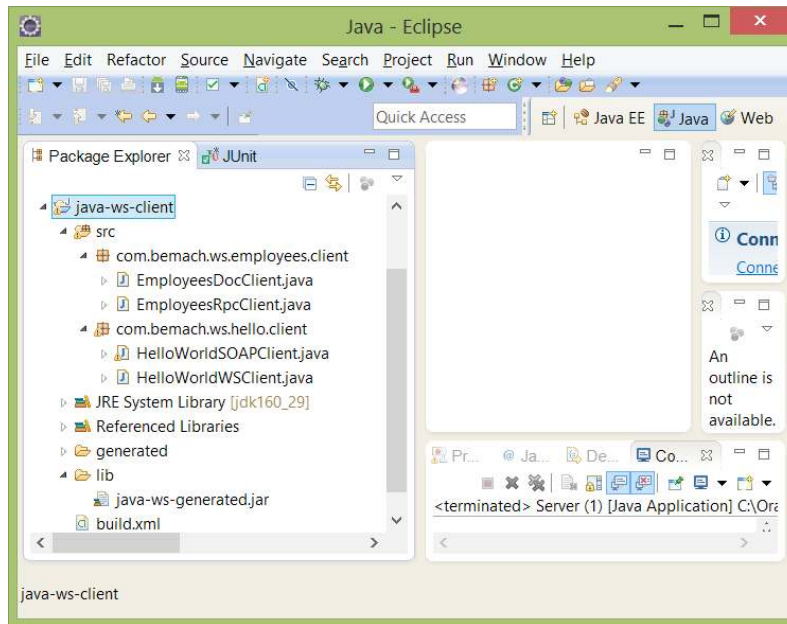


Figure 4-18. Screenshot of java-ws-client Java Project

“I studied English for 16 years but...
...I finally learned to speak it in just six lessons”
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download

4.6.2.1 *EmployeesDocClient.java*

This client code uses the generated client stub for making WS calls to the remote server. One important class is the QName, where we create a qualified name that contains the targetNamespace and the name attributes of the definitions element of the WSDL. The next important class is the URL where we create a service endpoint as the location attribute of the soap:address element of the WSDL. From these two classes, we can then create a service which we map onto the set of operations that the service provides. Mapping the port onto the EmployeeDocData is specified as a type attribute of the binding element within the WSDL. Once we get the port, we can call the operations as we did with the local method invocation in Java.

Listing 4-22. *EmployeesDocClient.java* Class

```
package com.bemach.ws.employees.client;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Logger;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import com.bemach.ws.doc.employees.Employee;
import com.bemach.ws.doc.employees.EmployeeDocData;
import com.bemach.ws.doc.employees.SOAPException_Exception;
import com.bemach.ws.rpc.employees.EmployeeRpcDataService;

/**
 * This code relies on ws client generated code using wsimport program:
 * wsimport -d . http://localhost:9999/doc/employees?WSDL
 * wsimport -d . http://localhost:9999/rpc/employees?WSDL
 * wsimport -d . http://localhost:9999/ch-1/HelloWorld?WSDL
 *
 * jar cvf ../ws-ch-1-generated.jar *
 *
 */
```

```

public class EmployeesDocClient {
    private static final Logger LOG = Logger.getLogger(EmployeesDocClient.
class.getName());
    private EmployeeDocData emplDs = null;

    public EmployeesDocClient(String urlStr, String targetNs, String name)
        throws MalformedURLException {
        LOG.info("Constructor ...");
        QName qName = new QName(targetNs, name);
        URL url = new URL(urlStr);
        Service service = EmployeeRpcDataService.create(url, qName);
        emplDs = service.getPort(EmployeeDocData.class);
    }

    public Employee get(long id) throws SOAPException_Exception {
        return emplDs.getEmployee(id);
    }

    public long create(Employee empl) {
        return emplDs.createEmployee(empl);
    }

    public boolean delete(long id) {
        return emplDs.deleteEmployee(id);
    }

    public boolean update(Employee empl) {
        return emplDs.updateEmployee(empl);
    }

    /**
     * @param args
     * @throws MalformedURLException
     * @throws SOAPException_Exception
     */
    public static void main(String[] args)
        throws MalformedURLException, SOAPException_Exception {
        LOG.info("Calling Employee (Document) data service ... ");
        String targetNameSpace = "http://employees.doc.ws.bemach.com/";
        String name = "EmployeeDocDataService";
        String urlStr = String.format("http://localhost:%s/doc/
employees", args[0]);

        EmployeesDocClient cli = new EmployeesDocClient(urlStr, targetName-
eSpace, name);

        long oldEmplNo = Integer.valueOf(args[1]);
        Employee empl = cli.get(oldEmplNo);
        LOG.info("last="+empl.getLastName());
        LOG.info("hire="+empl.getHireDate());
        LOG.info("last="+empl.getLastName());
        LOG.info("first="+empl.getFirstName());
    }
}

```



```

empl.setFirstName("Silvester");
empl.setLastName("Johnny");
long newEmplNo = cli.create(empl);
LOG.info("emplNo="+newEmplNo);

Employee newEmpl = cli.get(newEmplNo);

newEmpl.setLastName("New-name");
newEmpl.setEmplNo(newEmplNo);
boolean status = cli.update(newEmpl);
LOG.info("update:"+status);
LOG.info("last="+newEmpl.getLastName());
LOG.info("first="+newEmpl.getFirstName());

status = cli.delete(newEmplNo);
LOG.info("deleteEmployee:"+status);
LOG.info("Exit!");
}
}

```

4.6.2.2 *EmployeesRpcClient.java*

This class is nearly identical to that of the document-style client code. Thus, any difference between the two styles of client code is nearly impossible to notice at this level. The significant difference is in the coding within the SOAP engine on the client side.

Listing 4-23. *EmployeesRpcClient.java* Class

```

package com.bemach.ws.employees.client;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Logger;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import com.bemach.ws.rpc.employees.Employee;
import com.bemach.ws.rpc.employees.EmployeeRpcData;
import com.bemach.ws.rpc.employees.EmployeeRpcDataService;
import com.bemach.ws.rpc.employees.SOAPException_Exception;

```

```

/**
 * This code relies on ws client generated code using wsimport program:
 * wsimport -d . http://localhost:9999/rpc/Employees?WSDL
 * jar cvf ../ws-ch-1-generated.jar *
 *
 */
public class EmployeesRpcClient {
    private static final Logger LOG = Logger.getLogger(EmployeesRpcClient.
class.getName());

    private EmployeeRpcData emplDs = null;

    public EmployeesRpcClient(String urlStr, String targetNs, String name)
throws MalformedURLException {
        LOG.info("Constructor ...");
        QName qName = new QName(targetNs, name);
        URL url = new URL(urlStr);
        Service service = EmployeeRpcDataService.create(url, qName);
        emplDs = service.getPort(EmployeeRpcData.class);
    }

    public Employee get(long id) throws SOAPException_Exception {
        return emplDs.getEmployee(id);
    }

    public long create(Employee empl) {
        return emplDs.createEmployee(empl);
    }

    public boolean delete(long id) {
        return emplDs.deleteEmployee(id);
    }

    public boolean update(Employee empl) {
        return emplDs.updateEmployee(empl);
    }

    /**
     * @param args
     * @throws MalformedURLException
     * @throws SOAPException_Exception
     */
    public static void main(String[] args)
        throws MalformedURLException, SOAPException_Exception {
        LOG.info("Calling Employee (RPC) data service ... ");

        String targetNameSpace = "http://employees.rpc.ws.bemach.com/";
        String name = "EmployeeRpcDataService";
        String urlStr = String.format("http://localhost:%s/rpc/employees", args[0]);

        EmployeesRpcClient cli = new EmployeesRpcClient(urlStr, targetNameSpace, name);

```

```

long oldEmplNo = Integer.valueOf(args[1]);
Employee empl = cli.get(oldEmplNo);

LOG.info("last="+empl.getLastName());
LOG.info("hire="+empl.getHireDate());
LOG.info("last="+empl.getLastName());
LOG.info("first="+empl.getFirstName());

empl.setFirstName("Silvester");
empl.setLastName("Johnny");
long newEmplNo = cli.create(empl);
LOG.info("emplNo="+newEmplNo);

Employee newEmpl = cli.get(newEmplNo);

newEmpl.setLastName("New-name");
newEmpl.setEmplNo(newEmplNo);
boolean status = cli.update(newEmpl);
LOG.info("update:"+status);
LOG.info("last="+newEmpl.getLastName());
LOG.info("first="+newEmpl.getFirstName());

status = cli.delete(newEmplNo);
LOG.info("deleteEmployee:"+status);
LOG.info("Exit!");
}
}

```



What do you want to do?

No matter what you want out of your future career, an employer with a broad range of operations in a load of countries will always be the ticket. Working within the Volvo Group means more than 100,000 friends and colleagues in more than 185 countries all over the world. We offer graduates great career opportunities – check out the Career section at our web site www.volvogroup.com. We look forward to getting to know you!

VOLVO
 AB Volvo (publ)
www.volvogroup.com

VOLVO TRUCKS | REHAULT TRUCKS | MACK TRUCKS | VOLVO BUSES | VOLVO CONSTRUCTION EQUIPMENT | VOLVO PENTA | VOLVO AERO | VOLVO IT
 VOLVO FINANCIAL SERVICES | VOLVO 3P | VOLVO POWERTRAIN | VOLVO PARTS | VOLVO TECHNOLOGY | VOLVO LOGISTICS | BUSINESS AREA ASIA



4.6.3 Run the Client Application

With each of the client codes, we implemented one of the main methods for unit-testing purposes. Each in this class can run as a standalone Java application. To run these applications, the following command is used:

```
java -cp java-ws-client.jar;./lib/java-ws-generated.jar com.bemach.  
ws.employees.client.EmployeesDocClient
```

or

```
java -cp java-ws-client.jar;./lib/ws-ch-1-generated.jar com.bemach.  
ws.employees.client.EmployeesRpcClient
```