Getting Started with JDBC

Christopher Burns – CIS 327 – Java II

Introduction to JDBC

JDBC (Java DataBase Connectivity) is Java's interface to database systems. Using JDBC you can connect to a wide variety of databases ranging from commercial relational database servers to flat data files. The JDBC interface allows you to connect to a database and send SQL (Structured Query Language) commands as well as calling stored procedures.

As is common with Java, there are many online tutorials that can help you in learning JDBC. A good place to start is Sun's "JDBC Basics" at http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html

Selecting a Database

You will first need to decide on what database you will use. Since JDBC drivers are available for most databases you have the freedom to choose most any major database platform. If you have an MSDN Universal license or are taking one of the classes on Oracle you may want to consider MS SQL Server or Oracle. There are free and trial versions available of some of these such as Oracle Personal (http://otn.oracle.com/software/products/8i_personal).

As an alternative to the commercial database servers you may also want to consider an open source database server. These include MySQL (http://www.mysql.com/) and Hypersonic (http://sourceforge.net/projects/hsqldb/).

If you prefer to avoid having to setup a database server you might also consider Microsoft Access or CSV (Command Separated Values) text files. The JDBC interface to these is the same as with the DBMS (DataBase Management Systems) so if you later choose the change your database type it is a simple matter.

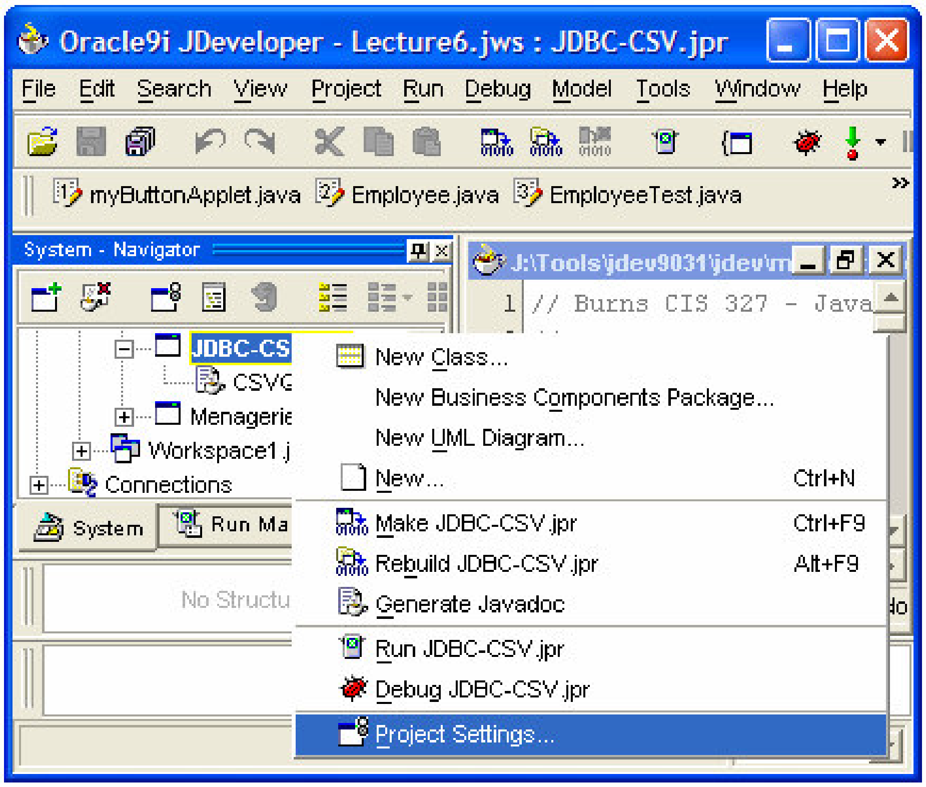
Getting JDBC Database Drivers

Before connecting to your database through Java, you will first need to locate a JDBC driver for your database. Sun has a resource for finding these drivers at http://industry.java.sun.com/products/jdbc/drivers

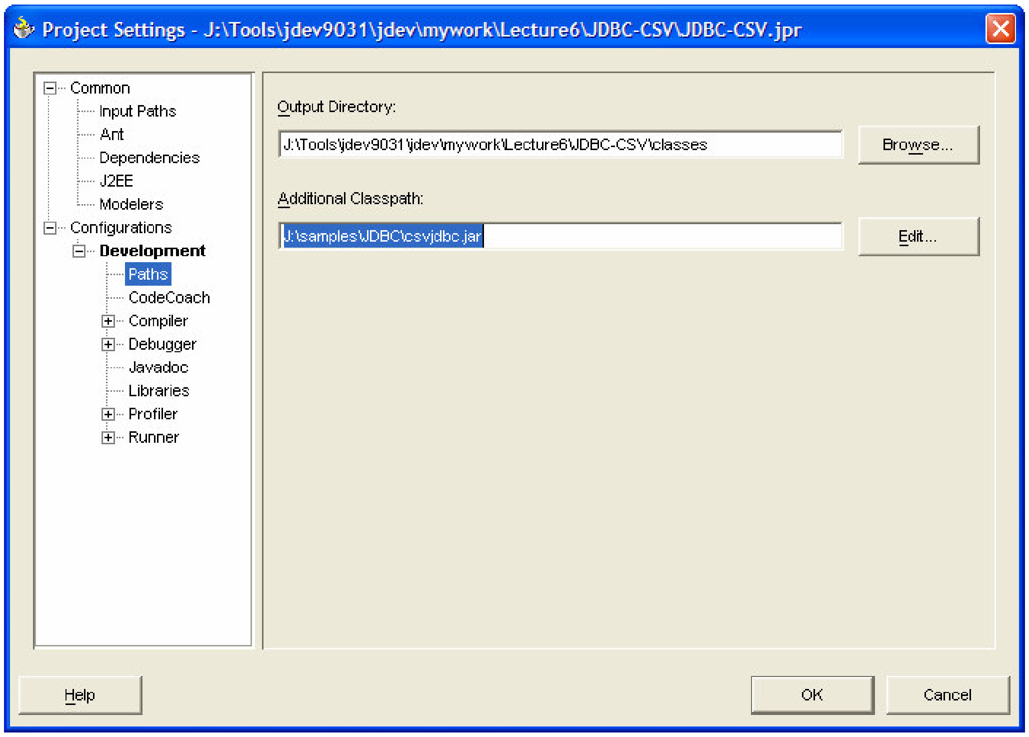
MS Access	(use JDBC ODBC bridge; see example)
SQL Server	http://jtds.sourceforge.net
Oracle	http://technet.oracle.com/software/tech/java/sqlj_jdbc/content.html
CSV/Text	http://sourceforge.net/projects/csvjdbc/
MySQL	http://sourceforge.net/projects/mmmysql/

Once you've downloaded a JDBC driver you'll need to ensure that it is either in your class path or decompressed into your project directory. JDBC drivers are typically packaged in a JAR file. Decompressing a JAR file can be done from the command line using the command: `jar xvf filename.jar`

It is easier to include the JAR file in your classpath. With JDeveloper this can be done through the project settings dialog.



Open the project settings by right-clicking on the project



Project Settings - J:\Tools\jdev9031\jdev\mywork\Lecture6\JDBC-CSV\JDBC-CSV.jpr

Output Directory: J:\Tools\jdev9031\jdev\mywork\Lecture6\JDBC-CSV\classes

Additional Classpath: J:\samples\JDBC\csvjdbc.jar

Browse to Configurations/Development/Paths. Add your JDBC JAR file to the Additional Classpath.

If you are using the JDK at the command line, you can include your JAR file by adding the `-classpath` option to your java.exe command.

Connecting to a Database

Now that you have your JDBC driver, you are ready to connect to your database. There are three steps that are taken to establish this connection. First you will need to include the class path for the JDBC classes. This is done by adding an import statement before your class implementation.

```
import java.sql.*;
```

Next, dynamically load your JDBC driver using the Java statement “Classes.forName”. The reason for this runtime loading of the JDBC driver class is that it allows the driver to be changed at runtime, making your code more flexible.

```
Class.forName( “org.relique.jdbc.csv.CsvDriver” );
```

The name of the JDBC driver class is in the form of the full package name followed by the class name. This is usually included with the documentation of your JDBC driver, but can also be ascertained by examining the package within the JAR file.

Third, create an instance of the Connection class for your database. This is returned from the static getConnection method of the DriverManager class. The getConnection method takes the URL of the database and has overloads that also can take a user id and password.

```
String databaseURL = “192.168.5.12”
Connection connection = DriverManager.getConnection( databaseURL );
```

The URL of your database maybe a filename or a network address. This depends on the type of database you are connecting to. Check the documentation of your JDBC driver to determine the expected format.

Both Class.forName and DriverManager.getConnection throw errors so you will need to have exception handling around these calls.

Look at the JDK documentation for more information on Class and DriverManager:

http://java.sun.com/j2se/1.4.1/docs/api/java/lang/Class.html
http://java.sun.com/j2se/1.4.1/docs/api/java/sql/DriverManager.html

Accessing the Database

At this point you should now be able to connect to your database. You can access your database by sending command statements. This is done through the Statement class. A Statement object is created through an active connection object.

```
Statement statement = connection.createStatement();
```

The statement object is a SQL channel to your database. Through it you can send SQL commands and receive result sets.

A sub class of Statement is the PreparedStatement class. This class allows you to create a SQL command string with parameters that can be called multiple times. If the DBMS supports prepared statements, your SQL statements will be compiled and optimized so they will run faster and more efficiently.

```
PreparedStatement preparedQuery =
        connection.prepareStatement(
         "SELECT * FROM myTable WHERE surname = ?” );
```

The use of PreparedStatements versus Statements is based on the capabilities of your DBMS and coding convenience. When you are learning JDBC, the Statement class is easier to use and debug.

Basic SQL Commands

Since the Statement object expects SQL commands, here is a quick summary of some basic commands. This is not intended to be a SQL reference, but if you are not familiar with SQL it should help get you started.

Create a table

A table can be created in your database by specifying the name of the table, column names and column types. The column types, or data types, may vary for different databases.

syntax:

```
CREATE TABLE tableName   ( column1Name column1Type,
                           column2Name column2Type,
                           …
                         )
```

example:

```
CREATE TABLE myTable     ( Surname varchar(30),
                           Phone varchar(10),
                           EmployeeID int )
```

Add a record to the table

Records can be added to the table you created, or an existing table, by using the SQL command INSERT.

syntax:

```
INSERT INTO tableName        ( column1Name, column2Name, … )
                     VALUES  ( column1Value, column2Value, … )
```

example:

```
INSERT INTO myTable          ( Surname, EmployeeID )
                     VALUES  ( “Jones”, 12 )
```

If you are inserting values into all of the columns it is not necessary to specify the column names but be sure the VALUES are in the order of the columns.

example:

```
INSERT INTO myTable VALUES ( “Smith”, “5405551212”, 10 )
```

Change a record in the table

Records in a table can be modified using the SQL command UPDATE.

syntax:

```
UPDATE tableName SET        ( column1Name = column1NewValue,
                              column2Name = column2NewValue,
                              ...
                            )
                     WHERE  ( column1Name = column1Value,
                              …
                            )
```

example:

```
UPDATE myTable SET          ( Phone = “7035551212” )
                     WHERE  ( Surname = “Jones” )
```

The WHERE statement is a query that allows you to select records to update the meet your specified criteria.

Delete a record from the table

Records in a table can be deleted by using the SQL command DELETE. Be careful the criteria you use for you WHERE clause as multiple records can be deleted.

syntax:

DELETE FROM *tableName* WHERE (*columnName* = *columnValue*, …)

example:

DELETE FROM myTable WHERE (Surname = “Smith”)

Delete a table from the database

Tables in your database can be deleted using the DROP command.

syntax:

DROP TABLE *tableName*

example:

DROP TABLE myTable

Retrieve records from a table

Records can be retrieved from a table using the SQL command SELECT. Records and columns from multiple tables my also be retrieved by using the JOIN operator. The joining of tables is beyond the scope of this introductory text.

syntax:

```
SELECT column1Name, column2Name …  FROM tableName
                        WHERE   ( column1Name = column1Value,
                                ( column2Name = column2Value,
                                   …
                                )
```

example:

```
SELECT Surname FROM myTable WHERE ( EmployeeID = 10 )
```

If you wish to retrieve all rows from the table you may use one of the SQL wildcard characters "*".

example:

```
SELECT * FROM myTable WHERE ( EmployeeID = 10 )
```

If you wish to retrieve all rows you may eliminate the query criteria.

example:

```
SELECT * FROM myTable
```

Sample JDBC Application

Using JDBC-ODBC Bridge for Microsoft Access Database

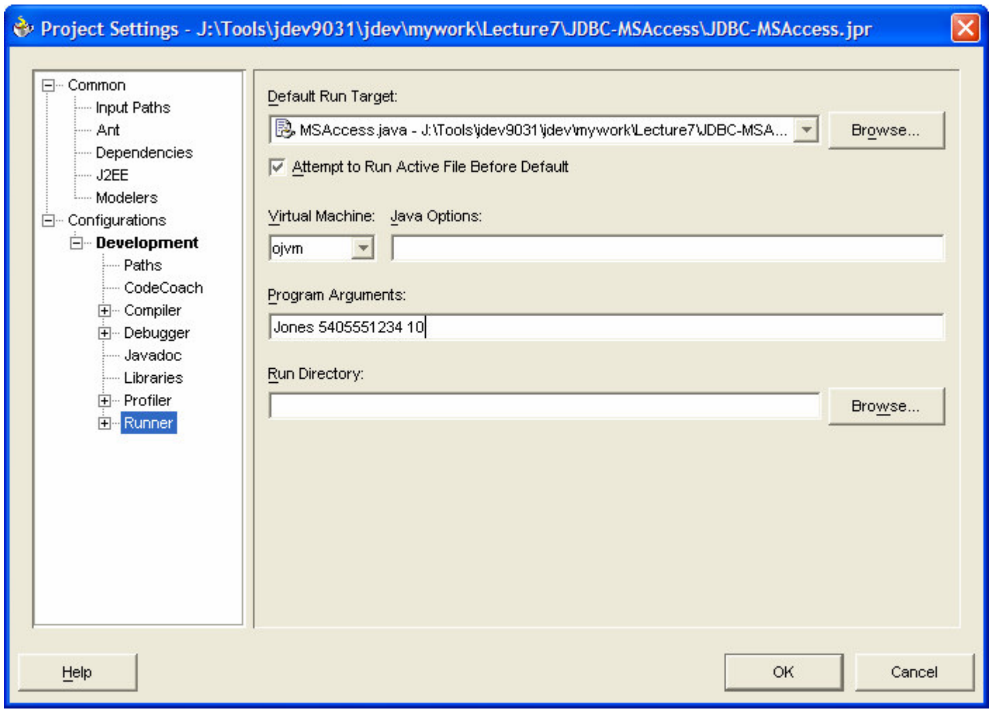
The JDBC-ODBC bridge provides a mechanism for using Microsoft's ODBC (Open DataBase Connectivity) programming interface from Java. The bridge is considered "experimental" by Sun, and is not included in the Java runtime of all browsers. If you create an applet using the bridge, users of your applet may need to download the full JRE to use the bridge.

Refer to Sun's description of the bridge at:
http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/bridge.doc.html

This example is using the bridge to open an Access database. There are JDBC drivers available for Access commercially and if you intend to create and distribute an applet that reads from an Access database you may want to consider these.

The example does not require downloading any special JDBC drivers and can be entered directly into JDeveloper and executed. It does require that a Microsoft Access database named DB1.MDB be created at C:\.

If you are running this program in JDeveloper, you can add parameters to pass to the application by accessing the project settings and adding them to the Configurations/Runner/Development, Program Arguments, text box.



Default Run Target: MSAccess.java - J:\Tools\jdev9031\jdev\mywork\Lecture7\JDBC-MSA...

☑ Attempt to Run Active File Before Default

Virtual Machine: ojvm

Java Options:

Program Arguments: Jones 5405551234 10

Run Directory:

```
// Burns CIS 327 - Java II
//
// MS Access database connection through JDBC ODBC bridge
//
import java.sql.*;

public class MSAccess
{
    static final String driverJDBC  = "sun.jdbc.odbc.JdbcOdbcDriver";

    // You can modify these to match your access database
    static String dataBaseFile = "db1.mdb";
    static String dataBasePath = "c:\\";
    static boolean dataBaseExclusive = false;

    public static void main(String args[])
    {
        try
        {
            // dynamically load Java class, in this case the JDBC driver
            //  in this case, since the class is a constant string, it
            //  is equivalent to "import com.inet.csv.CsvDriver"
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            String dataBaseURL =
                "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ="
                + dataBaseFile + ";DefaultDir="
                + dataBasePath;

            if ( dataBaseExclusive )
                dataBaseURL += ";Exclusive=1";

            // create a database connection object
            Connection connection = DriverManager.getConnection(dataBaseURL, null, null);

            // create a statement object for sending SQL statements to the database
            Statement st = connection.createStatement();

            // create a table in the database
            String createTableStatement = "CREATE TABLE myTable " +
                "(surname VARCHAR(30), phone VARCHAR(10), employeeID INT)";
            System.out.println( createTableStatement );
            st.executeUpdate( createTableStatement );

            // add a record to the database if two arguements were provided
            if ( args.length == 3 )
            {
                String insertStatement = "INSERT INTO myTable " +
                        "VALUES('" + args[0] + "','" + args[1] + "','" + args[2] + "')";
                System.out.println( insertStatement );
                st.executeUpdate( insertStatement );
            }

            // execute a select query
            String queryStatement = "SELECT * FROM myTable";
            System.out.println( queryStatement );
            ResultSet rs = st.executeQuery( queryStatement );

            // iterate through the record set and print the results
            while( rs.next() )
            {
                for( int j=1; j<=rs.getMetaData().getColumnCount(); j++ )
                {
                        System.out.print( rs.getString(j)+"\t" );
                }
                System.out.println();
            }

            // delete the record we just added
            String deleteStatement = "DELETE FROM myTable WHERE " +
                "surname = '" + args[0] + "'";
            System.out.println( deleteStatement );
            st.executeUpdate( deleteStatement );

            // delete the table we created
            String deleteTableStatement = "DROP TABLE myTable";
            System.out.println( deleteTableStatement );
            st.executeUpdate( deleteTableStatement );

            //close the objects
            st.close();
            connection.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

JDBC is still evolving

At the time of this writing, the most recent JDBC release is 3.0 and comes bundled with J2SE 1.4. An updated interface to JDBC called the RowSet Implementation is scheduled to be released in the near future as part of the J2SE 1.5. This will make it easier to pass tabular data (row sets) between components, thus allowing greater flexibility in data handling objects.