INTRODUCTION TO JASMINE ODQL

In Java
- Spring 2007 -

Rafael Cordones Marcos <rafa@dsv.su.se>

1 Introduction

Jasmine is an object database and, as you have seen in previous assignments, an object database contains classes and instances of these classes (also called objects). Many times we will need to integrate an application written in a programming language (Java, for instance) with the data contained in Jasmine. That is to say, we will need to use the objects contained in Jasmine from our Java code.

In the previous two laboratory assignments we have seen two ways of accessing data contained in a relational database from Java code. Namely, JDBC and SQLJ. By using JDBC or SQLJ we could fetch data which was kept in a relational database and operate with it in our Java code. The key difference with this assignment is that when accessing a relational database (with JDBC or SQLJ) the result of a query is always a set of rows composed of values of simple types like integers, strings, etc., but in object databases, query results can contain values of complex types like complex objects and/or collections of objects.

Recall that the JDBC and SQLJ drivers took care of mapping the types [3] used by the relational database (`VARCHAR(10)`, for instance) to the types used in Java (`String`).

But now, besides values of simple types, we need to map Jasmine classes to Java classes. That is to say, we need to access objects that reside in the Jasmine database from Java code. The best way [4] to do this is with the use of a proxy [5]. A proxy is basically an object that represents and behaves like another object. Your Java code will *use* a Java object (the proxy) and it will be this object that will be responsible for the communication with the object that resides in the Jasmine database.

Jasmine offers one Java Application Programmers Interface (API) called J-API and on top of this API, it provides two approaches to interface Java code with its resources (objects, classes, …) (shown in Figure 1):

1. making direct use of J-API from Java code
2. using a Java Class Generator (JPCG)

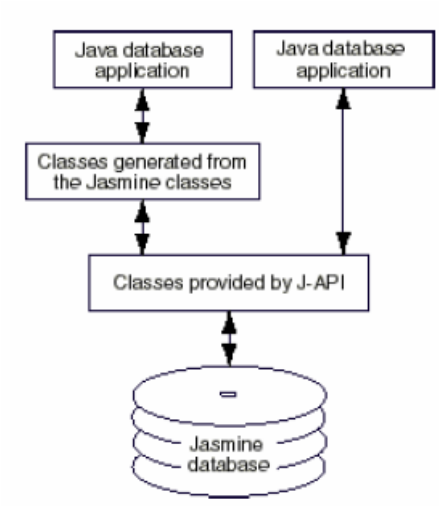


Figure 1. Jasmine's Java Proxies architecture (taken from [1]).

You can look at the previous figure that depicts these two interfacing possibilities and relate them to JDBC and SQLJ.

To learn how to use the Java Class Generator (JPCG) you can refer to Chapter 8 in *"Programmer's Guide for Java Proxies (JP) 2.0"* (see reference [1]). The class generator maps Jasmine classes, properties and data types to those of Java and allows the programmer to write Java code that uses data stored in a Jasmine database in a more transparent way. But in this introduction we will focus on option 1, i.e. we will use J-API to access the Jasmine database server directly.

2 An example

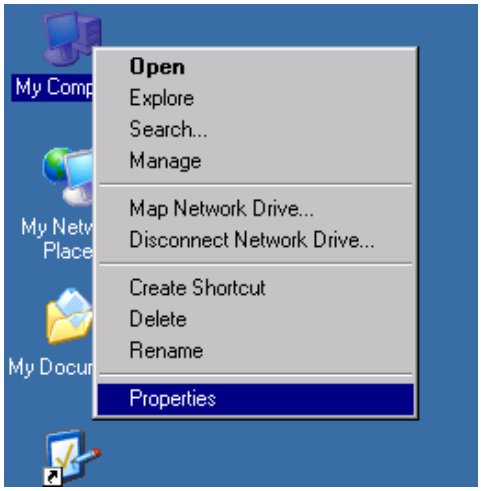
On the location \\Db-srv-1\StudentCourseMaterial\IS4 spring 2007\JavaODQLTest on the local network you'll find the file `JavaODQLTest.java`. It contains the complete source code of this example.

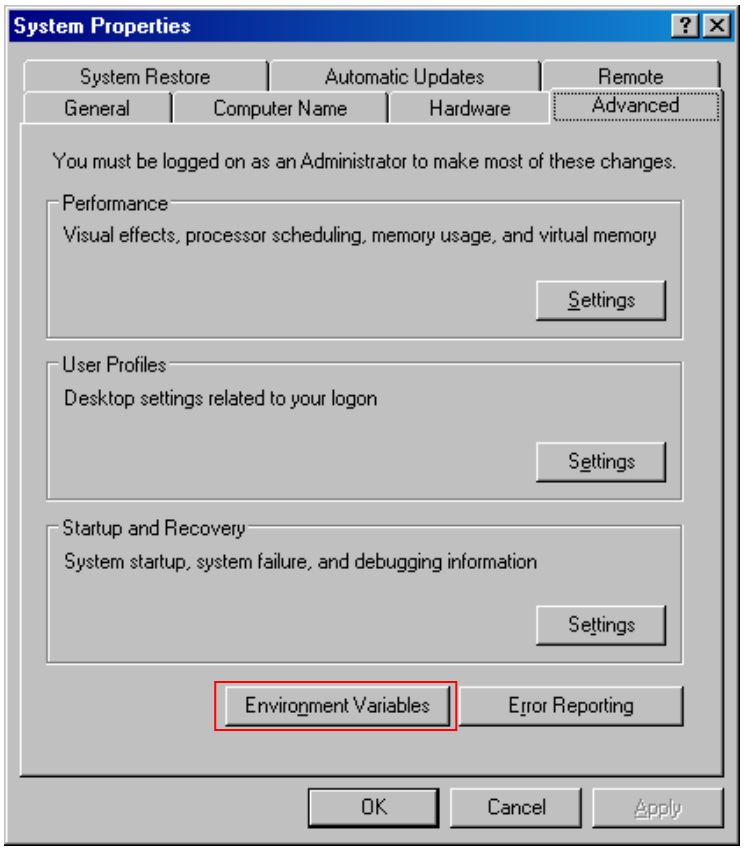
1. First of all, create a new folder (preferably on your d: drive) and name it something convenient like for example `JavaODQLTest`.
2. Copy the source file *JavaODQLTest.java* from the network to your new folder.

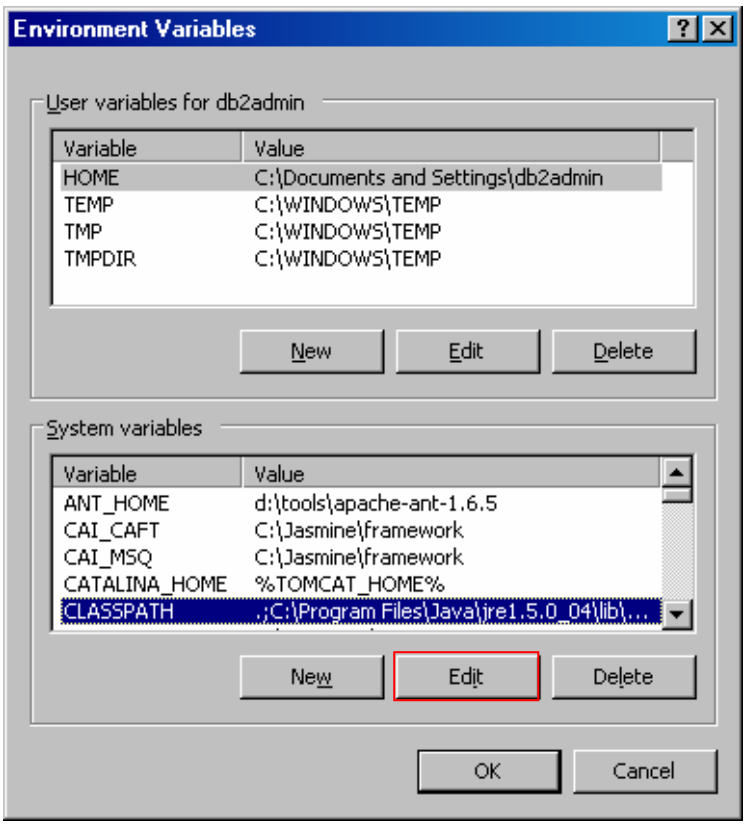
2.1 Preparation

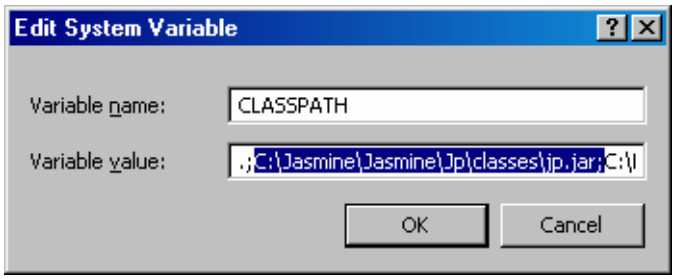
The *JavaODQLTest.java* file is not compiled yet. To be able to compile and execute a Java program that uses J-API you need to have the *J2SE Development Kit (JDK)* installed. This environment is free of charge and can be downloaded from http://java.sun.com/j2se/1.5.0/download.jsp but you probably have it already installed on your disk.

Jasmine's Java API (J-API) is packaged into a JAR file called `jp.jar` which you can find in `C:\Jasmine\Jasmine\Jp\classes`. When compiling a Java program that makes use of this API we will need to tell the Java compiler and the Java virtual machine where to find it. We do this by modifying the `CLASSPATH` variable as follows (Note that this is already done on the prepared disks):




System Properties
System Restore
Automatic Updates
Remote
General
Computer Name
Hardware
Advanced
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings
User Profiles
Desktop settings related to your logon
Settings
Startup and Recovery
System startup, system failure, and debugging information
Settings
Environment Variables
Error Reporting
OK
Cancel
Apply


Environment Variables
User variables for db2admin
Variable
Value
HOME
C:\Documents and Settings\db2admin
TEMP
C:\WINDOWS\TEMP
TMP
C:\WINDOWS\TEMP
TMPDIR
C:\WINDOWS\TEMP
New
Edit
Delete
System variables
Variable
Value
ANT_HOME
d:\tools\apache-ant-1.6.5
CAI_CAFT
C:\Jasmine\framework
CAI_MSQ
C:\Jasmine\framework
CATALINA_HOME
%TOMCAT_HOME%
CLASSPATH
;C:\Program Files\Java\jre1.5.0_04\lib\...
New
Edit
Delete
OK
Cancel



Add `C:\Jasmine\Jasmine\Jp\classes\jp.jar;` (note the semicolon at the end).

All Command Prompt windows that are already open will not detect this change. So make sure to open a new Command Prompt window.

2.2 Compilation and execution

To compile the example open a Command Prompt window, go to the directory where you saved the example file and type the compilation command:

```
D:\JavaODQLTest>javac JavaODQLTest.java
```

To execute the example type the java execution command:

```
D:\JavaODQLTest>java JavaODQLTest
Looking for customer Donald Duck
Found customer Donald Duck
Customer has these credit cards:
333 338 222 997 634
4444 2345 3456 4567
1234 2345 3456 4567
434 878 999 7634
```

In order to run the program successfully, the Jasmine database must be running on the local machine.

2.3 Explanation of the source code

We will briefly go through the main classes and methods used in the example. Open *JavaODQLTest.java* to view the application source code.

```
import jp.jasmine.japi.*;
```

This statement includes the classes from the J-API packages.

```
db = new Database();
db.startSession();
db.startTransaction();
```

With the first statement we open a new connection to the local Jasmine database server[1]. Note that we do not need to give any address when creating the connection when it is a local connection, i.e. the Jasmine server is running on the same computer where we are executing the Java application. The other two statements set up the database for sending queries.

```
ODQLStatement odql = db.getODQLStatement();
```

[1] *"A J-API client using* remote *access goes through an RMI server as an intermediary to communicate with a Jasmine database server."[1, Chapter 3]*

We get an ODQL instance so we can send ODQL queries to the database.

```
odql.defaultCF("CAStore");
```

We set the class family to which we will send the queries.

```
odql.execute("cs = select Customer from Customer where " +
             "  Customer.name == cname;");
```

We send an ODQL statement to Jasmine for execution. Please note that we need to previously define the ODQL variables cs and cname for this query to work (see the source code).

```
DBCollection dbcCustomers = (DBCollection) odql.getVar("cs");
```

Once we have executed the query we want to get the result into Java, we use the class DBCollection and the method "getVar" from the ODQLStatement class. In order to be able to loop through the elements of the collection, the DBCollection class provides a method that returns a Java Enumeration:

```
Enumeration eCustomers = dbcCustomers.elements();
```

See http://java.sun.com/j2se/1.5.0/docs/api/java/util/Enumeration.html for more information of the methods that the Enumeration interface provides.

```
DBObject c = (DBObject) eCustomers.nextElement();
```

For each object in DBCollection we create a Java proxy object and use the "getProperty" method of the DBObject class to access the values of its properties:

```
String theName = (String) c.getProperty("name");
```

In case the property we want to retrieve is a complex value (set, bag, …) we do it in a similar way but we use the Enumeration interface to access the elements:

```
DBCollection dbcCreditCards =
       (DBCollection) c.getProperty("creditcards");
Enumeration eCreditCards = dbcCreditCards.elements();
```

3 How to proceed

We recommend that you start with the example that we have given you and that already works and **add small modifications incrementally**, i.e. do not start from scratch!

So the algorithm we propose that you follow in your work is:

1. Make a small change
2. Compile and test. Does it work?
    a. If yes, go to step 1 and add another small change that takes you to your final objective.
    b. If no, try to fix the error and go to Step 2.

As you can see from the previous algorithm, if you do not know where you are going it will be difficult to get there!

4 References

1. Programmer's Guide for Java Proxies (JP) 2.0
   See the file `C:\Jasmine\Doc\ODB\jp.pdf`
2. Sample programs:
   See the file `C:/Jasmine/Jasmine/Jp/samples/samples.html`
3. Datatype: http://en.wikipedia.org/wiki/Datatype
4. Patterns:
   http://wiki.java.net/bin/viewauth/Javapedia/Patterns
5. The Proxy Design Pattern:
   http://wiki.java.net/bin/view/Javapedia/ProxyPattern
6. "Introduction to CA Jasmine ODB" found in the Computer Environment Tutorials compendium.