Communications of the Association for Information Systems

Volume 9 Article 25

11-16-2002

AMCIS 2002 Workshops and Panels VI: Technical Note: Implementing Java-Based Stored Technical Procedures in the Oracle Database

Ruben E. Quinonez
California State University-Pomona, requinonez@csupomona.edu

Communications of the Association for Information Systems

SPECIAL ISSUE ON THE AMCIS 2002 WORKSHOPS VI: TECHNICAL NOTE: IMPLEMENTING JAVA-BASED STORED PROCEDURES IN THE ORACLE DATABASE

Ruben E. Quinonez
Computer Information Systems
California State University-Pomona
requinonez@csupomona.edu

ABSTRACT

This technical note provides an introduction to writing stored procedures using the Java language. The primary purpose of the workshop is to demonstrate the steps required to publish a Java method inside the Oracle 8i Lite database environment.

KEYWORDS: Oracle, Java, stored-procedures, database

I. INTRODUCTION

Java is becoming a programming language of choice [Stewart, 2002]. With its object-oriented and cross-platform features [Koffman & Wolz, 2002], Java brings a one-size-fits-all paradigm to application development. Application developers can benefit from learning a language that can be used across a wide range of operating systems. In addition to application development, Java is also used inside database development environments. Starting with the 8i release, Oracle supports Java for the development of database stored-procedures and triggers. This new development platform provides the cross-platform benefits not previously possible with Oracle's proprietary PL/SQL language. Businesses can now implement business rules into procedures that can be accessed from inside the database via Java-based stored-procedures or from stand-alone Java applications. Similarly, developers can leverage their Java programming skills when writing applications as well as database stored-procedures. This common development environment is expected to bring synergies particularly in applications that have a database component.

This article covers the development of Java static methods, how they are imported into the Oracle environment and how they can be used as stored-procedures. This article is designed for faculty with some knowledge of Java and Oracle database who are interested in integrating the two technologies in either a database course or a Java programming course.

II. STEPS FOR WRITING JAVA-BASED STORED-PROCEDURES

The steps required for publishing Java-based stored-procedures are:

1. Write a Java static method
2. Compile the method and test it outside the database
3. Import the method into the database
4. Publish the method as a database function
5. Execute the database function from the database SQL prompt

The example used in this article is based on a simple task: write a stored-procedure that adds two numbers. This simple task minimizes unnecessary Java and/or database complexities and it concentrates on the steps required to integrate the two technologies.

III. THE JAVA STEPS

Many options are available for writing Java classes. The developer environment used for this article is Sun's Java 2 SDK version 1.3.1_04[1]. Despite its command line interface, Sun's SDK is easy to install and it requires only about 35MB of hard disk space. Any Java runtime environment can be used as long as it supports the Java Native Interface (JNI).

Step 1: Write a Java static method

The Java class written for this workshop is called Mathlib.java. Its contents are as follows:

```
public class Mathlib {

  public static void main(String args[])  {
      System.out.println (Addnumbers(1,2));
  }

  public static int Addnumbers(int x, int y)  {
    return x + y;
  } // Add
}
```

One important detail in the Mathlib class is that method Addnumbers is static. Static means that the method belongs to the class, not to a class instance. Mathlib will not be instantiated inside the database environment.

One of the challenges in stored-procedure development is debugging. There are no provisions for debugging Java code inside the database environment. Therefore, the Java code must be free of bugs before loading it into the database. To simplify the process and concentrate on issues related to the Java code, a main method was added to the class only for the purpose of being able to run it (and debug it) as a stand-alone application.

Step 2: Compile the method and test it outside the database

Using the command line interface of Sun's SDK, compile the Java class:

```
C:\ javac Mathlib.java
```

[1] Available for download at http://java.sun.com/j2se/1.3/download.html

Once compiled[2], the class is executed:

```
C:\ java Mathlib
3
```

When the class is executed, its main method calls Addnumbers method and it passes two arguments: 1, 2. The Addnumbers method then returns to main the sum of the two arguments. Main uses System.out.println to display the answer: 3.

The next section discusses the steps required to publish the Java method inside the Oracle database.

IV. THE DATABASE STEPS

The database used for this article is Oracle 8i Lite release 4.0.1. Keep in mind that even minor release updates may change the procedures required to publish Java methods.

Now that we have a compiled the class, we are ready to import it into the Oracle database. The steps required for the database are executed using Oracle's SQL utility, SQL Plus.

Step 3: Import the method into the database

The syntax for loading Java classes in the Oracle environment is:

```
CREATE OR REPLACE JAVA CLASS USING BFILE ('java-directory', 'class-name');
```

where java-directory is the location of the java class and class-name is the name of the class. In our example, the directory is c:\data\javaoracle and the class name is Mathlib.class. Therefore, the command issued is:

```
SQL> CREATE OR REPLACE JAVA CLASS USING BFILE ('c:\data\javaoracle','Mathlib.class')
/

Operation 0 succeeded.
```

Note that the SQL utility returned the message "Operation 0 succeeded". This indicates that the Java class was successfully loaded into the database. Another important detail is the use of the slash ("/") instead of the traditional semicolon[3].

Step 4: Publish the method as a database function

The next step is to publish the method as a database function[4]. The syntax for publishing the method is:

```
CREATE OR REPLACE FUNCTION function-name (argument-list)
RETURN INT AS LANGUAGE JAVA NAME 'method-name' (argument-type-list)
return return-type'
```

Where function-name is the name of the Java method once published in the database (which may be different from the Java method name), argument-list is the argument type and name used in the Java method. Method-name is the Java method prefixed by its class name. Argument-

[2] The compilation step produces a file with the extension ".class" that will be used when importing it to the database.

[3] The Oracle 8i Lite requires a slash instead of a semicolon for Step 4 of this workshop. Step 3 can be executed with a semicolon or a slash.

[4] A Java class is called a function once loaded into the database environment.

type-list is the argument type list (excluding the argument names) required by the method. Lastly, the return-type is the data type returned by the Java method. In our example, the command is:

```
SQL> CREATE OR REPLACE FUNCTION AddNumbers (x int, y int)
  2  RETURN INT AS LANGUAGE JAVA NAME 'Mathlib.AddNumbers (int, int)
  3  return int'
  4  /
Operation 0 succeeded
```

In this example, the integer data type ("int") is common to Java and Oracle. Other data types may not be the same. For example, Java uses "String" for string data while Oracle uses CHAR, VARCHAR, or VARCHAR2 for string data. For more details, refer to Oracle 8i Lite documentation located at *directory-name*/DOC/LITE/jdg/html/jdgovrw.htm#1011219 (where directory-name is the location of your Oracle 8i installation).

Step 5: Execute the database function

Now that we have loaded and published the Java method as a database function, we are ready to test it. One convenient way is to use Oracle's "dual" table[5]. Therefore, using the "dual" table, the command is:

```
SQL> SELECT AddNumbers (5, 3) FROM DUAL;

ADDNUMBERS
----------
         8
```

The final task is to execute the function using data from a table. In this article, we use a sample table called Mathlib. The SQL commands used to create and populate the Mathlib table are:

```
SQL> create table Mathlib (
  2    x        number(2),
  3    y        number(2)
  4  );

Table created.

SQL> insert into Mathlib values (1,1);

1 row created.

SQL> insert into Mathlib values (2,2);

1 row created.
```

Executing the function Addnumbers against the Mathlib table results in:

```
SQL> SELECT x,y,AddNumbers (x,y) FROM Mathlib;

         X          Y ADDNUMBERS
---------- ---------- ----------
         1          1          2
         2          2          4
```

[5] The "dual" table is a dummy table that can be used to test database functions.

Note that function Addnumbers can accept as arguments a column name of an existing table, a constant, or the result of a calculation. The important criterion is that the two arguments are integers.

V. INCORPORATING JAVA STORED-PROCEDURES TO THE CLASSROOM

Oracle Java-based stored-procedures are by their very nature the integration of Java and Oracle. As such, Java-based stored-procedures can be incorporated into courses that have as prerequisites at least one of the two technologies (Java or Oracle). At my university, students complete an introductory course in Java before taking a database course. Therefore, I found it convenient to introduce them to Java stored-procedures in their database class. Other potential areas are intermediate or advanced level Java courses assuming that the students took a database class previously.

VI. CONCLUSION

Java-based stored-procedures offer a one-size-fits-all paradigm to application development as well as database procedure development. In addition, this technology provides students with an integrative view of systems development. As it applies to students learning business application development, Java and databases should not be considered independent technologies but rather interrelated components of a business application. This workshop is designed with the goal of assisting faculty interested in integrating the two technologies in either a database or a Java programming course.

Editor's Note: This technical note is based on the author's workshop at AMCIS 2002 in Dallas, Texas. It was received on September 13, 2002 and was published on November 16, 2002 together with the other articles from the Workshops and Panels. The special issue was under the editorship of Les Ball.

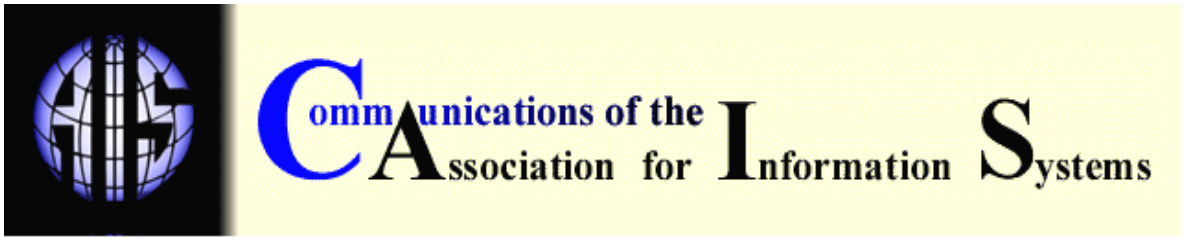
REFERENCES

Koffman, E., & Wolz, U. (2002). *Problem Solving with Java*. Boston MA: Addison Wesley.

Stewart, B. (2002). *Java as a Teaching Language*. O'Reilly. Retrieved 2/25/2002, 2002, from the World Wide Web: http://java.oreilly.com/news/teachjava_0101.html

ABOUT THE AUTHOR

Ruben E. Quinonez is assistant professor of Computer Information Systems at Cal Poly Pomona. He completed his Ph.D. at the Claremont Graduate University. He has over ten years of industry experience in the areas of application and database development.



ISSN: 1529-3181