JAVA and Oracle - The good, better and best

Tuesday, June 14th
11:30am - 12:30pm

Coleman Leviter, OCP
NYOUG/IOUG
Oracle Database Developer
cleviter@ieee.org

#C16LV

CV

- WMS Group – Seventeen years
- VAX Rewrite (.for) to UNIX (.c, .pc)
- VAX Forms to Oracle Forms
- TIFF file migration to Oracle
- XML Development
- WMS Development and Support
- IOUG Select Contributor
- IOUG Tips & Best Practices
- ODTUG Journal
- IOUG Collaborate
- Oracle Open World
- IOUG Collaborate Conference Chair
- IOUG Board of Directors
- NYOUG Vice President

Presentation Objectives

- Java Background/History
- Current Use
- Architecture, Applications
- Syntax
- Basic Types
- Why Java
- Examples
- Queries
- SQL*PLUS, SQL Developer (12c)

Java: Key History Points

Java language initiated by James Gosling and others beginning June, 1991 at Sun Microsystems

Sun Microsystems released Java 1.0 in 1995

Java syntax developed with a C/C++ syntax flavor for system and application programmers

In 2006, Sun released Java and Java Virtual Machine (JVM) as free and open source software under the terms of the General Public License, in effect allowing the public to use, run, study, share and modify the software.

Oracle acquires Sun in 2009/2010

Gosling resigns from Oracle in 2010

Java Language: Key Architecture Points

Computer programming language: concurrent (time sliced), class based (object blueprint) or Object Oriented Programming (OOP)

Write once, run anywhere: Java code, once compiled can run on all platforms that support Java without the need for recompilation.

Java applications -

- compiled to bytecode
- run on any Java virtual machine (JVM)
- translates bytecode into the platform's machine language
    a) Overhead of interpreting bytecode into machine instructions makes interpreted programs run more slowly than native executables
    b) just-in-time (JIT) compilers that compile bytecode into machine code during runtime increase performance; compilation is done during execution of a program

Java Language: Key Architecture Points (cont'd)

Language goals

a) It must be "simple, object-oriented, and familiar".
b) It must be "robust and secure".
c) It must be "architecture-neutral and portable".
d) It must execute with "high performance".
e) It must be "interpreted, threaded, and dynamic".

Automatic Memory Management

- Garbage collection: automatic.
- Java runtime for recovering memory when no longer in use

Java Syntax and Notation

Syntax mostly derived from C and C++. Unlike C++, Java is almost exclusively an object-oriented language

No global functions or variables. All code belongs to classes (blueprint).
All values are objects (location in memory)

Basic syntax

a) identifier – name of an element in the code; case sensitive
b) keywords (approximately 50) – boolean, for, continue, else, public, private, int, byte, case
c) literals – binary, octal, hexadecimal, float, double, boolean (true, false),
d) variables - identifiers associated with values.
   i.e. int count; count = 35; int count = 35;

Java Syntax and Notation (cont'd)

e) code blocks -

```
void doSomething() {
    int a;
    {
       int b;
       a = 1;
    }
    a = 2;
    b = 3; // Illegal because the variable is declared in an
           inner scope.
}
```

f) comments - /* */, //

```
/**
*
*
*/
```

Java Program Structure

Applications consist of collections of classes. Classes exist in packages

Every application has an entry point. The entry point is the “main” method.

More than one class can have a “main” method. The “main” class” is defined externally.

```
public static void main(String[] args) {    // method  is static*
}
```

Packages - part of a class name

- they are used to group and/or distinguish named entities from other ones.

  It's defined at the start of the file:

```
package myapplication.mylibrary;
    public class MyClass {
    }
```

*static means that the variable or method marked as such is available at the class level. In other words, you don't need to create an instance of the class to access it.

Java Program Structure (cont'd)

Classes with the “public” modifier

- placed in the files with the same name and java extension
- put into nested folders corresponding to the package name.
- The above class myapplication.mylibrary.MyClass will have the following path: "myapplication/mylibrary/MyClass.java".

Import declaration – a shortcut manner of referencing a named type

```
package myPackage;

import java.util.Random; // Single type declaration

public class ImportsTest {
    public static void main(String[] args) {
        /* The following line is equivalent to
         * java.util.Random random = new java.util.Random();
         * It would've been incorrect without the import declaration */
        Random random = new Random();
    }
}
```

Java Program Structure (cont'd)

Static Import Declaration

- allow access to static members defined in another class, interface annotation, or enum; without specifying the class name:

- import static java.lang.System.out; //'out' is a static field in java.lang.System

```
public class HelloWorld {
    public static void main(String[] args) {
        /* The following line is equivalent to:

            System.out.println("Hello World!");

            and would have been incorrect without the import declaration. */
        out.println("Hello World!");
    }
}
```

Java Program Structure (cont'd)

Operators

- similar to those in C++.
- no delete operator due to garbage collection mechanisms in Java,
- no operations on pointers since Java does not support them.
- Java has an unsigned right shift operator (>>>)
- while C's right shift operator's signedness is type-dependent.
- operators in Java cannot be overloaded, i.e. +, -, () method invocation, & logical and, = simple assignment, etc.

Conditional statements:

```
if (i == 3) doSomething();
if (i == 2) {
    doSomething();
} else {
    doSomethingElse();
}
```

Java Program Structure (cont'd)

Switch Statements

- Switch statements in Java can use byte, short, char, and int
- (note: not long) primitive data types or their corresponding wrapper types
- Starting with J2SE 5.0, it is possible to use enum types.
- Starting with Java SE 7, it is possible to use Strings.
- Other reference types cannot be used in switch statements.

```
switch (ch) {
    case 'A':
        doSomething(); // Triggered if ch == 'A'
        break;
    case 'B':
    case 'C':
        doSomethingElse(); // Triggered if ch == 'B' or ch == 'C'
        break;
    default:
        doSomethingDifferent(); // Triggered in any other case
        break;
}
```

Java Program Structure (cont'd)

Iteration (for, while), Jump (labels, break, continue, return)

Exception handling

- Exception handling statements
- try-catch-finally statements
- Exceptions are managed within try ... catch blocks.

```
try {
    // Statements that may throw exceptions
    methodThrowingExceptions();
} catch (Exception ex) {
    // Exception caught and handled here
    reportException(ex);
} finally {
    // Statements always executed after the try/catch blocks
    freeResources();
}
```

- the statements within the try block are executed
- if any of them throws an exception, execution of the block is discontinued
- the exception is handled by the catch block
- multiple catch blocks, the first block with an exception variable whose type matches the type of the thrown exception is executed.

Java Program Structure (cont'd)

Constructors and initializers

- Is a special method called when an object is initialized.
- Its purpose is to initialize the members of the object.
- Main differences between constructors and ordinary methods
  - constructors are called only when an instance of the class is created
  - never return anything.
  - constructors are declared as common methods
  - they are named after the class and no return type is specified:

```
class Foo {
    String str;

    Foo() { // Constructor with no arguments
        // Initialization
    }

    Foo(String str) { // Constructor with one argument
        this.str = str;
    }
}
```

Java Program Structure (cont'd)

Methods

- Statements in Java must reside within methods
- Methods are similar to functions except they belong to classes.
- A method has a return value, a name and usually some parameters initialized when it is called with some arguments.
- Similar to C++, methods returning nothing have return type declared as void.
- Unlike in C++, methods in Java are not allowed to have default argument values and methods are usually overloaded instead.

```
class Foo {
    int bar(int a, int b) {
        return (a*2) + b;
    }
    /* Overloaded method with the same name but different set of arguments */
    int bar(int a) {
        return a*2;
    }
}
```

A method is called using . notation on an object, or in the case of a static method, also on the name of a class.

```
Foo foo = new Foo();
int result = foo.bar(7, 2); // Non-static method is called on foo

int finalResult = Math.abs(result); // Static method call
```

Java Features in Oracle

- You can write and load Java applications within the database
- It is a safe language with a lot of security features.
- Java has been developed to prevent anyone from tampering with the operating system where the Java code resides in.
- Some languages, such as C, can introduce security problems within the database.
- However, Java, because of its design, is a robust language that can be used within the database.

Although the Java language presents many advantages to developers, providing an implementation of a JVM that supports Java server applications in a scalable manner is a challenge.

Java and RDBMS: A Robust Combination
Multithreading
Memory Spaces Management
Footprint
Performance of an Oracle JVM
Dynamic Class Loading

Java Features in Oracle

Java and RDBMS: A Robust Combination

- Oracle Database provides Java applications with a dynamic data-processing engine
- Supports complex queries and different views of the same data.
- All client requests are assembled as data queries for immediate processing, and query results are generated dynamically.
- Java and Oracle Database helps you to create component-based, network-centric applications that can be easily updated as needs change.
- you can move applications and data stores off the desktop and onto intelligent networks and network-centric servers.
- you can access those applications and data stores from any client device.

Figure 1-1 shows a traditional two-tier, client/server configuration in which clients call Java stored procedures the same way they call PL/SQL stored procedures.
The figure also shows how Oracle Net Services Connection Manager can combine many network connections into a single database connection.
This enables Oracle Database to support a large number of concurrent users.

Java Features in Oracle (cont'd)

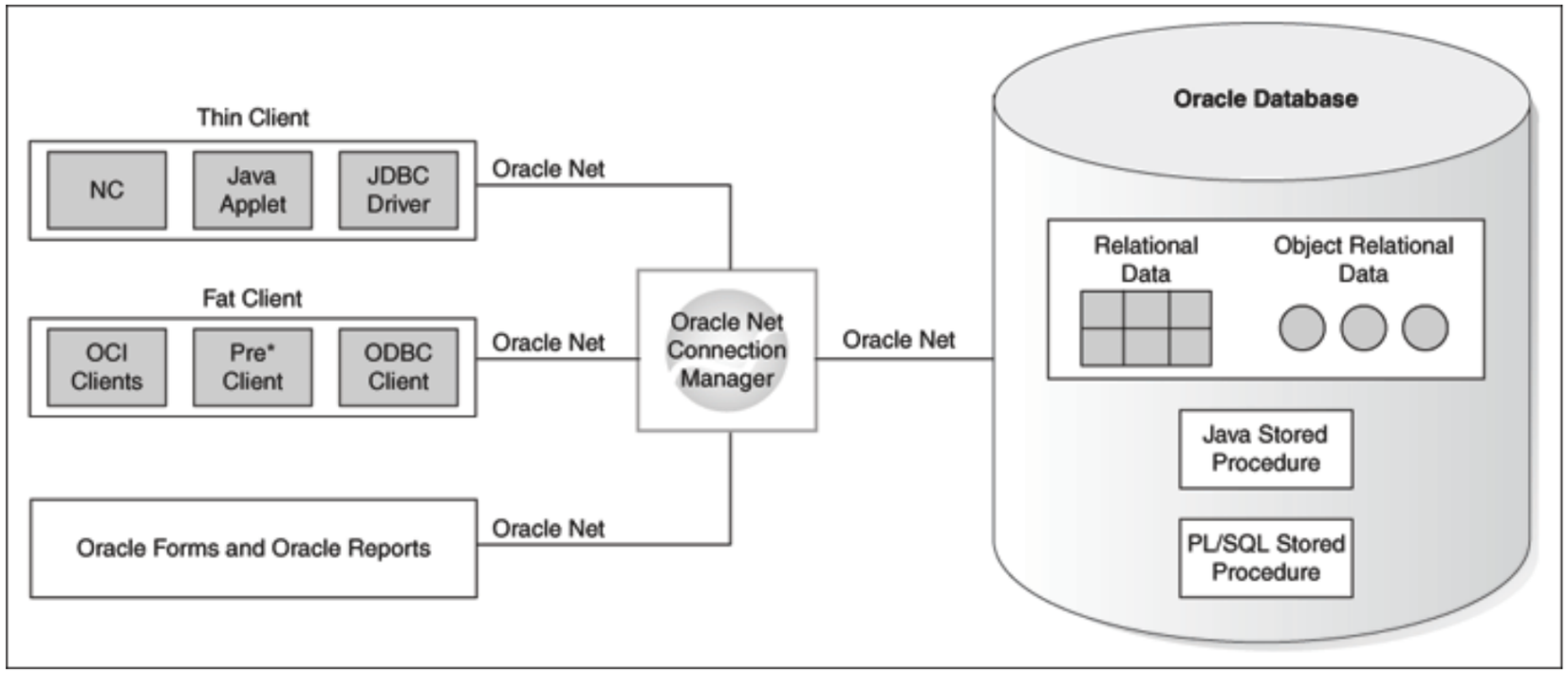


Figure 1-1 Two-Tier Client/Server Configuration Multithreading - Oracle JVM model

- Thousands of users connect to the server and run the same Java code
- Each user experiences it as if he or she is running his or her own Java code on his or her own JVM.
- The responsibility of an Oracle JVM is to make use of operating system processes and threads and the scalable approach of Oracle Database.
- The garbage collector of the Oracle JVM is more reliable and efficient because it never collects garbage from more than one user at any time.

Java New Features in Oracle 12cR1 (12.1)

Multiple JDK Support

- Oracle Database 12c Release 1 (12.1),
- Oracle JVM provides support for multiple Java Development Kit (JDK) versions, including the latest JDK version.
- The supported versions are a default version and the next higher version.
- Oracle Database 12c Release 1 (12.1) supports JDK 6 and JDK 7, where JDK 6 is the default JDK version.
- If you do not set the JDK version explicitly, the JDK version in effect is JDK 6.
- At any given point of time, only one JDK version is in effect for the Database.
- You can switch from one JDK version to another depending on your needs.
- Note: If you change the JDK version for a CDB (container database), then this change will affect all the corresponding PDBs (pluggable databases) uniformly.

Java New Features in Oracle 12cR1 (12.1) (cont'd)

JNDI Support: Java Naming and Directory Interface

- Native Oracle JVM support for JNDI enables you to bind Oracle data source objects
- Contain specific database connection information, by a name in a directory structure.
- Uses of JNDI include:
  - connecting a Java application to an external directory service
  - (such as an address database or an LDAP server)
  - You can use this name to retrieve the particular connection information to establish a connection within an application.

- You can also change the database connection properties and the actual source database without changing the application by changing only the associated object to which a specific name is resolved.

- This feature also provides a general purpose directory service for storing objects and object references.

Java New Features in Oracle 12cR1 (12.1) (cont'd)

Support for Logging

- Oracle JVM extends the JDK Java Logging API in the area of logging properties lookup to enhance security of logging configuration management and to support logging configurations on a user basis.

- You must activate the LogManager in the session to initialize the logging properties in Oracle JVM.
- The logging properties are initialized once per session with the LogManager API that is extended with the database resident resource lookup.

Java New Features in Oracle 12cR1 (12.1) (cont'd)

Improved debugging features

- You can now use breakpoints and watchpoints while debugging Java stored procedures.
- Oracle Database provides the Java Debug Wire Protocol (JDWP) interface for debugging Java stored procedures. JDWP is supported by Java Development Kit (JDK) 1.4 and later versions.

Following are a few features that the JDWP interface supports:

- Listening for connections
- Changing the values of variables while debugging
- Evaluating arbitrary Java expressions, including method evaluations
- Setting or clearing breakpoints on a line or in a method
- Stepping through the code
- Setting or clearing field access or modification watchpoints

Jinitiator vs Java

SR 3-12277332811 : 10g Forms reduced height issue after compiling in 11g AIX 7.1.0.0

10g Forms reduced height issue after compiling in 11g AIX 7.1.0.0

In earlier Java versions, the space consumed by a font was calculated differently than in newer versions. The result is that if you move from a very old Java version to a newer one the space needed will change. Oracle Jinitiator is (at its core) the old Sun Java Plugin. So when you move from Jinitiator (or even an old Sun/Oracle Plugin) to a more modern version changes in visual appearance are expected. As I recall you mentioned a mix of Oracle Jinitiator and the old plugin (I seem to recall you mentioning version 6). Regardless of which, both are very old and the difference between Jinitiator to Java 6 to Java 8 likely will show some differences.

Jinitiator vs Java

WMS Test [usmliu50]

Key Level Help

FUNCTION	PC KEY	FUNCTION	PC KEY
Backspace	Backspace	Left	LeftArrow
Clear Form	Alt F10	List of Values	Home
Clear Item	Shift F7	Next Block	PageDown
Count Query Hits	PF3	Next Item	TAB
Delete	Delete	Next Record	Shift PageDown
Delete Record	Ctrl D	Next Set Records	Shift F9
Down	DownArrow	Previous Block	PageUp
Duplicate Row	Ctrl T	Previous Item	Shift TAB
Edit	Alt E	Previous Record	Shift PageUp
Enter Query	F11	Right	RightArrow
Execute Query	F12	Up	UpArrow
Exit	Esc		
Insert Record	Ctrl A		

Esc - To Return

FRM-40815: Variable GLOBAL.FACILITY_NAME does not exist.
Record: 1/1

Oracle Fusion Middleware Forms Services

Key Level Help

FUNCTION	PC KEY	FUNCTION	PC KEY
Backspace	Backspace	Left	LeftArrow
Clear Form	Alt F10	List of Values	Home
Clear Item	Shift F7	Next Block	PageDown
Count Query Hits	PF3	Next Item	TAB
Delete	Delete	Next Record	Shift PageDown
Delete Record	Ctrl D	Next Set Records	Shift F9
Down	DownArrow	Previous Block	PageUp
Duplicate Row	Ctrl T	Previous Item	Shift TAB
Edit	Alt E	Previous Record	Shift PageUp
Enter Query	F11	Right	RightArrow
Execute Query	F12	Up	UpArrow
Exit	Esc		
Insert Record	Ctrl A		

Esc - To Return

FRM-40815: Variable GLOBAL.FACILITY_NAME does not exist.
Record: 1/1

Java Fun Time

Example 1 – Hello.java

```
C:\Users\cal\java>type Hello.java
public class Hello
{
   public static String world()
   {
      return "Hello world";
   }
}
```

```
javac  -cp .;ojdbc6.jar    Hello.java   --compile, -cp .;ojdbc6.jar  optional for this program
```

```
loadjava -user scott/T1ger@orcl Hello.class  – load into Oracle
```

```
--sqlplus session – activate SQL Developer or TOAD
Create or replace function helloworld  RETURN VARCHAR2 AS    --create function
LANGUAGE JAVA NAME 'Hello.world () return java.lang.String';
/
DECLARE   -- run anonymous block
   lclString VARCHAR2(20);
begin
   select helloworld() INTO lclString from dual;
   dbms_output.put_line( lclString);
end;
```

```
select   * from dba_java_classes where name like '%Hello%' and owner = 'SCOTT'
```

```
DROP JAVA CLASS "Hello";   -- drop class
```

```
DROP function helloworld;
```

Example 2 – connJDBC.java

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;

public class connJDBC {
	public static void main(String[] argv) {
		System.out.println("-------- Oracle JDBC Connection Testing ------");
		try {
			Class.forName("oracle.jdbc.driver.OracleDriver");
		} catch (ClassNotFoundException e) {

			System.out.println("Where is your Oracle JDBC Driver?");
			e.printStackTrace();
			return;
		}
		System.out.println("Oracle JDBC Driver Registered!");
		Connection connection = null;
		try {
			connection = DriverManager.getConnection(
					"jdbc:oracle:thin:@localhost:1521:orcl", "scott","T1ger");
		} catch (SQLException e) {

			System.out.println("Connection Failed! Check output console");
			e.printStackTrace();
			return;
		}
		if (connection != null) {
			System.out.println("You made it, take control your database now!");
		} else {
			System.out.println("Failed to make connection!");
		}
	}
}

C:\Users\cal\java>java -cp .;ojdbc6.jar connJDBC
```

Example 3 – theDate.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;
/**
 * Simple Java Program to connect Oracle database by using Oracle JDBC thin driver
 * Make sure you have Oracle JDBC thin driver in your classpath before running this program
 * @author
 */
public class theDate{
    public static void main(String args[]) throws SQLException {
        //URL of Oracle database server
        String url = "jdbc:oracle:thin:@cal15t:1521:orcl";

        //properties for creating connection to Oracle database
        Properties props = new Properties();
        props.setProperty("user", "scott");
        props.setProperty("password", "T1ger");

        //creating connection to Oracle database using JDBC
        Connection conn = DriverManager.getConnection(url,props);

        String sql ="select sysdate as current_day from dual";

        //creating PreparedStatement object to execute query
        PreparedStatement preStatement = conn.prepareStatement(sql);

        ResultSet result = preStatement.executeQuery();

        while(result.next()){
            System.out.println("Current Date from Oracle : " +        result.getString("current_day"));
        }
        System.out.println("done");
    }
}
```

```
C:\Users\cal\java>java -cp .;ojdbc6.jar theDate
```

Example 4 – connBanner.java

```
import java.sql.*;
//import java.util.Properties;

public class connBanner {
  public static void main (String[] args) throws Exception
  {
   Class.forName ("oracle.jdbc.OracleDriver");

   Connection conn = DriverManager.getConnection
     ("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "T1ger");  // localhost is valid
                    // @//machineName:port/SID,   userid,  password
   try {
     Statement stmt = conn.createStatement();
     try {
       ResultSet rset = stmt.executeQuery("select BANNER from SYS.V_$VERSION");
       try {
         while (rset.next())
           System.out.println (rset.getString(1));   // Print col 1
       }
       finally {
          try { rset.close(); } catch (Exception ignore) {}
       }
     }
     finally {
       try { stmt.close(); } catch (Exception ignore) {}
     }
   }
   finally {
     try { conn.close(); } catch (Exception ignore) {}
   }
  }
}

C:\Users\cal\java>javac  connBanner.java
C:\Users\cal\java>java -cp .;ojdbc6.jar connBanner
```

Example 5 – queryEmp.java

```
import java.sql.*;

public class queryEmp {
	public static void main(String args[]) {
		try {
			Class.forName("oracle.jdbc.driver.OracleDriver");

			Connection con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:orcl", "scott", "T1ger");

			Statement stmt = con.createStatement();

			ResultSet rs = stmt.executeQuery("select * from emp");
			 System.out.println ("EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO");
			while (rs.next())
				System.out.println(rs.getInt(1) + "  " + rs.getString(2) + "  "
					+ rs.getString(3) + "  " + rs.getString(4) + "  "
					+ rs.getString(5) + "  " + rs.getString(6) + "  "
					+ rs.getString(7) + "  " + rs.getString(8) + "  ");
			con.close();

		} catch (Exception e) {
			System.out.println(e);
		}

	}
}
```

```
C:\Users\cal\java>javac  queryEmp.java
C:\Users\cal\java>java -cp .;ojdbc6.jar queryEmp
```

Example 6 – insertEmp.java (1 of 3)

```
//package com.mkyong.jdbc;

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class insertEmp {

	private static final String DB_DRIVER = "oracle.jdbc.driver.OracleDriver";
	private static final String DB_CONNECTION = "jdbc:oracle:thin:@localhost:1521:orcl";
	private static final String DB_USER = "scott";
	private static final String DB_PASSWORD = "T1ger";
	private static final DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
	public static void main(String[] argv) {
		try {
			insertRecordIntoDbUserTable();
		} catch (SQLException e) {
			System.out.println(e.getMessage());
		}
	}
	private static void insertRecordIntoDbUserTable() throws SQLException {
		Connection dbConnection = null;
		Statement statement = null;

		String insertTableSQL = "INSERT INTO EMP " +
			" (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO)" +
			"  VALUES  ('6533', 'GARDE',  'CIO', " +
			"'3930', " +
			"to_date('05/24/1987','mm/dd/yyyy'), " +
			"'8600', '860','40' ) "  ;
```

Example 6 – insertEmp.java (2 of 3)

```
try {
        dbConnection = getDBConnection();
        statement = dbConnection.createStatement();
        System.out.println(insertTableSQL);
        // execute insert SQL stetement
        statement.executeUpdate(insertTableSQL);
        System.out.println("Record is inserted into EMP table!");
        // execute delete SQL stetement
} catch (SQLException e) {

        System.out.println(e.getMessage());

} finally {

                if (statement != null) {
                        statement.close();
                }
                if (dbConnection != null) {
                        dbConnection.close();
                }
        }
}
```

Example 6 – insertEmp.java (3 of 3)

```
private static Connection getDBConnection() {
        Connection dbConnection = null;
        try {
                Class.forName(DB_DRIVER);

        } catch (ClassNotFoundException e) {

                System.out.println(e.getMessage());
        }
        try {

                dbConnection = DriverManager.getConnection(
                      DB_CONNECTION, DB_USER,DB_PASSWORD);
                return dbConnection;

                } catch (SQLException e) {

                        System.out.println(e.getMessage());
                }
                return dbConnection;
        }
        private static String getCurrentTimeStamp() {
                java.util.Date today = new java.util.Date();
                return dateFormat.format(today.getTime());
        }
}
```

```
C:\Users\cal\java>javac  insertEmp.java
C:\Users\cal\java>java -cp .;ojdbc6.jar insertEmp
```

Example 7 – deleteEmp.java (1 of 3)

```
//package com.mkyong.jdbc;

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class deleteEmp {

	private static final String DB_DRIVER = "oracle.jdbc.driver.OracleDriver";
	private static final String DB_CONNECTION = "jdbc:oracle:thin:@localhost:1521:orcl";
	private static final String DB_USER = "scott";
	private static final String DB_PASSWORD = "T1ger";
	private static final DateFormat dateFormat = new SimpleDateFormat(
			"yyyy/MM/dd HH:mm:ss");

	public static void main(String[] argv) {

		try {

			insertRecordIntoDbUserTable();

		} catch (SQLException e) {

			System.out.println(e.getMessage());

		}

	}
```

Example 7 – deleteEmp.java (2 of 3)

```
private static void insertRecordIntoDbUserTable() throws SQLException {

        Connection dbConnection = null;
        Statement statement = null;

        String insertTableSQL = "INSERT INTO EMP " +
                " (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO)" +
                "  VALUES  ('6533', 'GARDE',  'CIO', " +
                "'3930', " +
                "to_date('05/24/1987','mm/dd/yyyy'), " +
                "'8600', '860','40' ) "  ;

        String deleteTableSQL = "DELETE FROM EMP WHERE EMPNO = 6533 ";


                try {
                        dbConnection = getDBConnection();
                        statement = dbConnection.createStatement();

//                      System.out.println(insertTableSQL);

                        // execute insert SQL stetement
//                      statement.executeUpdate(insertTableSQL);

                        // execute delete SQL stetement
                        statement.executeUpdate(deleteTableSQL);

                        System.out.println("Record is removed from EMP table!");
```

Example 7 – deleteEmp.java (3 of 3)

```
        } catch (SQLException e) {
                System.out.println(e.getMessage());
        } finally {
                if (statement != null) {
                        statement.close();
                }
                if (dbConnection != null) {
                        dbConnection.close();
                }
        }
}
private static Connection getDBConnection() {
        Connection dbConnection = null;
        try {
                Class.forName(DB_DRIVER);

        } catch (ClassNotFoundException e) {

                System.out.println(e.getMessage());
        }
        try {
                dbConnection = DriverManager.getConnection(
                   DB_CONNECTION, DB_USER,DB_PASSWORD);
                return dbConnection;

        } catch (SQLException e) {
                System.out.println(e.getMessage());
        }
        return dbConnection;
    }
    private static String getCurrentTimeStamp() {
        java.util.Date today = new java.util.Date();
        return dateFormat.format(today.getTime());
    }
}
```

```
C:\Users\cal\java>javac  deleteEmp.java
C:\Users\cal\java>java -cp .;ojdbc6.jar deleteEmp
```

Example 8 – insertPetey.java (1 of 2)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

/*
 * Here we will learn to Insert Record in DB using JDBC Statements in Oracle and MySQL DB
 */
public class insertPetey {

	public static void main(String[] args) {

		String tableCreateQuery = "create table PETEY(name varchar2(10), breed varchar2(20))";
		String inserRecordQuery = "insert into PETEY values('petey','lab golden mix')";
		Statement statement = null;
		Connection connection = null;

		Scanner scanner = new Scanner(System.in);
		System.out
				.println("Please provide below details to connect Oracle Database");
		System.out.println("Enter Database");
		String dbName = scanner.next();
		System.out.println("Enter UserName");
		String userName = scanner.next();
		System.out.println("Enter Password");
		String password = scanner.next();

		try {

			Class.forName("oracle.jdbc.driver.OracleDriver");

		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
```

Example 8 – insertPetey.java (2 of 2)

```
try {
        connection = DriverManager.getConnection(
                        "jdbc:oracle:thin:@localhost:1521:" + dbName, userName,
                        password);
} catch (SQLException e) {
        e.printStackTrace();
}
if (connection != null) {
        System.out.println("nSuccessfullly connected to Oracle DB");
        try {
                statement = connection.createStatement();
                System.out.println("Insert Record Query :" + inserRecordQuery);
                statement.execute(tableCreateQuery);
                System.out.println("nTable Created Successfully");

                statement.execute(inserRecordQuery);
                System.out.println("nRecord Inserted Successfully");
        } catch (SQLException e) {
                e.printStackTrace();
        }
    } else {
        System.out.println("nFailed to connect to Oracle DB");
    }
  }
}
G:\personal_files\ioug_collaborate_16\java>javac insertPetey.java
G:\personal_files\ioug_collaborate_16\java>java -cp .;ojdbc6.jar  insertPetey
Please provide below details to connect Oracle Database
Enter Database
orcl
Enter UserName
scott
Enter Password
T1ger
nSuccessfullly connected to Oracle DB
Insert Record Query :insert into PETEY values('petey','lab golden mix')
nTable Created Successfully
nRecord Inserted Successfully
```

Example 9 – ohHello.java

```
import java.sql.*;

public class ohHello{
    public static void main(String[] args) {

        System.out.println("oh Hello :)");

    }
}
```

```
G:\personal_files\ioug_collaborate_16\java>javac ohHello.java

G:\personal_files\ioug_collaborate_16\java>java  -cp .;ojdbc6.jar  ohHello
oh Hello :)

G:\personal_files\ioug_collaborate_16\java>
```

Example 10 – adjEmp.java

```
import java.sql.*;
import oracle.jdbc.driver.*;

public class adjEmp {
  public static void raiseSalary (int empNo, float percent)
  throws SQLException {
    Connection conn = new OracleDriver().defaultConnection();
    String sql = "UPDATE emp SET sal = sal * ? WHERE empno = ?";
    try {
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setFloat(1, (1 + percent / 100));
      pstmt.setInt(2, empNo);
      pstmt.executeUpdate();
      pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }
}
```

```
– cmd line
G:\personal_files\ioug_collaborate_16\java>javac  -cp .;ojdbc6.jar  adjEmp.java

G:\personal_files\ioug_collaborate_16\java>loadjava -user scott/T1ger@orcl adjEmp.class

– SQL Developer
CREATE OR REPLACE PROCEDURE raise_salary (empno NUMBER, pct NUMBER)
AS LANGUAGE JAVA
NAME 'adjEmp.raiseSalary(int, float)';


DECLARE
   emp_id  NUMBER;
   percent NUMBER;
BEGIN
   -- get values for emp_id and percent
   raise_salary(7369, 5);
END;
```

Example 11– jdb: Java debug (1 of 3)

```
// : c15:SimpleDebugging.java
// {ThrowsException}
// From 'Thinking in Java, 3rd ed.' (c) Bruce Eckel 2002
// www.BruceEckel.com. See copyright notice in CopyRight.txt.
public class SimpleDebugging {
  private static void foo1() {
    System.out.println("In foo1");
    foo2();
  }

  private static void foo2() {
    System.out.println("In foo2");
    foo3();
  }

  private static void foo3() {
    System.out.println("In foo3");
    int j = 1;
    j--;
    int i = 5 / j;
  }

  public static void main(String[] args) {
    foo1();
  }
} ///:~
```

Example 11– jdb: Java debug (2 of 3)

```
G:\personal_files\ioug_collaborate_16\java>javac -g  -cp .;ojdbc6.jar SimpleDebugging.java

G:\personal_files\ioug_collaborate_16\java>java  -cp .;ojdbc6.jar SimpleDebugging
In foo1
In foo2
In foo3
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at SimpleDebugging.foo3(SimpleDebugging.java:20)
        at SimpleDebugging.foo2(SimpleDebugging.java:13)
        at SimpleDebugging.foo1(SimpleDebugging.java:8)
        at SimpleDebugging.main(SimpleDebugging.java:24)

G:\personal_files\ioug_collaborate_16\java>jdb -classpath .;ojdbc6.jar SimpleDebugging
Initializing jdb ...
> catch Exception
Deferring all Exception.
It will be set after the class is loaded.
> run
run SimpleDebugging
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: In foo1
In foo2
In foo3
```

Example 11– jdb: Java debug (3 of 3)

```
Exception occurred: java.lang.ArithmeticException (uncaught)"thread=main", SimpleDebugging.foo3(), line=20
bci=15
20        int i = 5 / j;

main[1] list
16      private static void foo3() {
17        System.out.println("In foo3");
18        int j = 1;
19        j--;
20 =>      int i = 5 / j;
21      }
22
23      public static void main(String[] args) {
24        foo1();
25      }
main[1]

main[1] locals
Method arguments:
Local variables:
j = 0
main[1]
```

Sources

https://en.wikipedia.org/wiki/Java

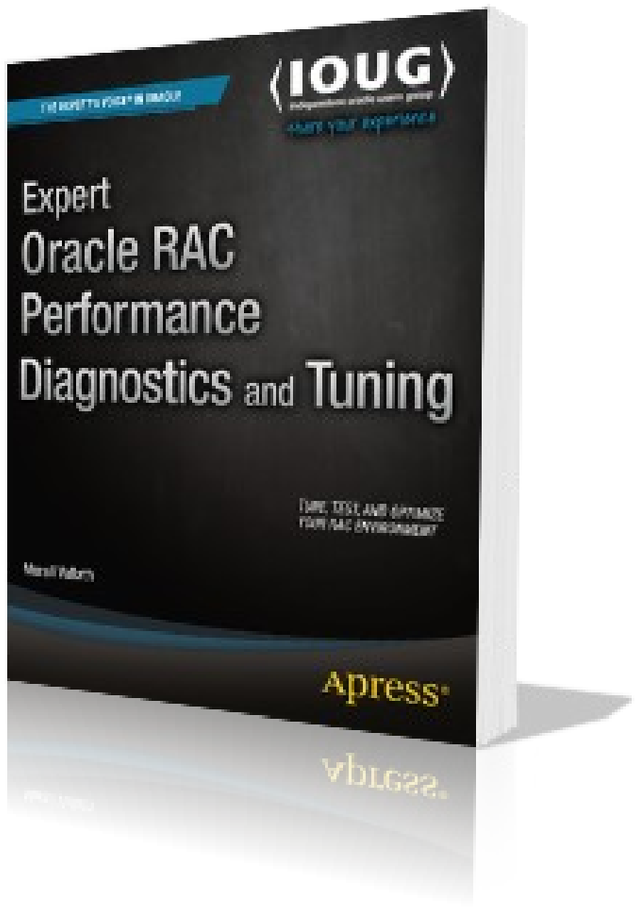
http://www.java2s.com/Tutorial/Java/

https://docs.oracle.com/javase/tutorial/

Thinking in Java, 3rd ed.' (c) Bruce Eckel 2002

JOIN or RENEW

Did you know that IOUG Members get up to 60% off of IOUG Press eBooks?



Connect with IOUG

Twitter: @IOUG or follow hash tag: #IOUG

Facebook: IOUG's official Facebook Fan Page: www.ioug.org/facebook

LinkedIn: Connect and network with other Oracle professionals and experts in the IOUG Community LinkedIn group. www.ioug.org/linkedin

Questions

Coleman Leviter, OCP
NYOUG/IOUG
Oracle Database Developer
cleviter@ieee.org