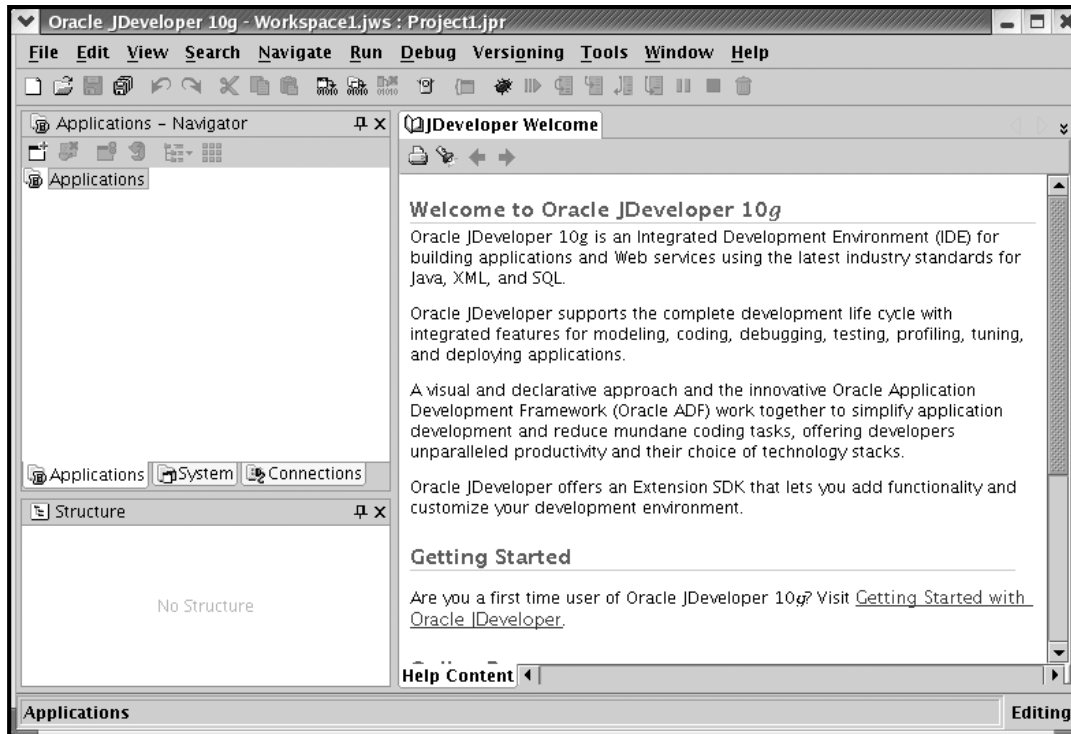




ORACLE®

Copyright © 2006, Oracle. All rights reserved.

# JDeveloper



ORACLE

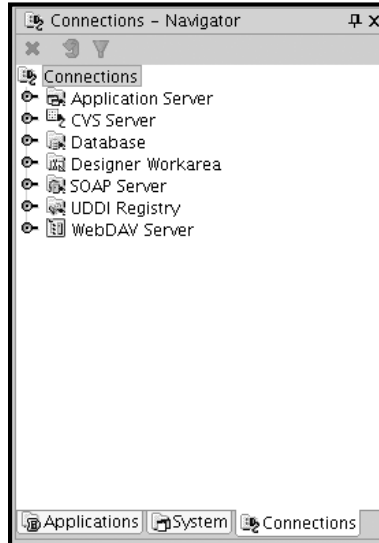
Copyright © 2006, Oracle. All rights reserved.

## JDeveloper

Oracle JDeveloper 10g is an integrated development environment (IDE) for developing and deploying Java applications and Web services. It supports every stage of the software development life cycle (SDLC) from modeling to deploying. It has the features to use the latest industry standards for Java, Extensible Markup Language (XML), and SQL while developing an application.

Oracle JDeveloper 10g initiates a new approach to J2EE development with the features that enables visual and declarative development. This innovative approach makes J2EE development simple and efficient.

# Connection Navigator



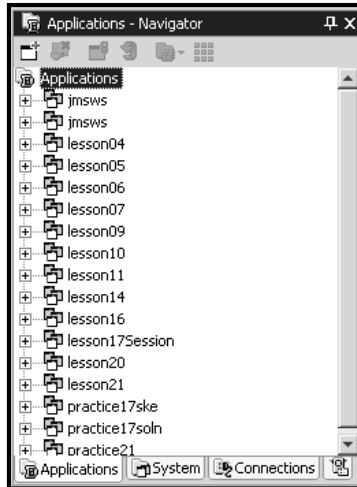
ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Connection Navigator

Using Oracle JDeveloper 10g, you can store the information necessary to connect to a database in an object called “connection.” A connection is stored as part of the IDE settings, and can be exported and imported for easy sharing among groups of users. A connection serves several purposes from browsing the database and building applications, all the way through to deployment.

# Application Navigator



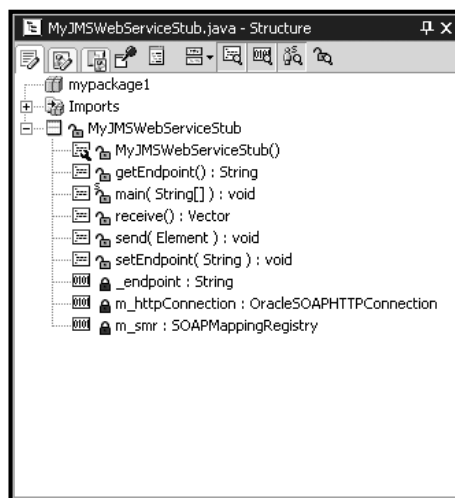
ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Application Navigator

The Application Navigator gives you a logical view of your application and the data it contains. The Application Navigator provides an infrastructure that the different extensions can plug into and use to organize their data and menus in a consistent, abstract manner. While the Application Navigator can contain individual files (such as Java source files), it is designed to consolidate complex data. Complex data types such as entity objects, UML (Unified Modeling Language) diagrams, Enterprise JavaBeans (EJB), or Web services appear in this navigator as single nodes. The raw files that make up these abstract nodes appear in the Structure window.

## Structure Window



ORACLE

Copyright © 2006, Oracle. All rights reserved.

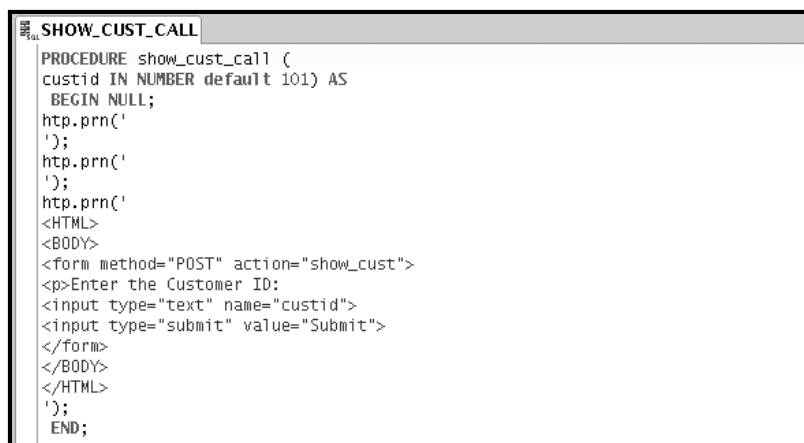
### Structure Window

The Structure window offers a structural view of the data in the document currently selected in the active window of those windows that participate in providing structure: the navigators, the editors and viewers, and the Property Inspector.

In the Structure window, you can view the document data in a variety of ways. The structures available for display are based upon document type. For a Java file, you can view code structure, user interface (UI) structure, or UI model data. For an XML file, you can view XML structure, design structure, or UI model data.

The Structure window is dynamic, always tracking the current selection of the active window (unless you freeze the window's contents on a particular view), as is pertinent to the currently active editor. When the current selection is a node in the navigator, the default editor is assumed. To change the view on the structure for the current selection, select a different structure tab.

## Editor Window



```

SQL SHOW_CUST_CALL
PROCEDURE show_cust_call (
  custid IN NUMBER default 101) AS
BEGIN NULL;
  http.prn('
  ');
  http.prn('
  ');
  http.prn('
  <HTML>
  <BODY>
  <form method="POST" action="show_cust">
  <p>Enter the Customer ID:
  <input type="text" name="custid">
  <input type="submit" value="Submit">
  </form>
  </BODY>
  </HTML>
  ');
END;

```

ORACLE

Copyright © 2006, Oracle. All rights reserved.

### Editor Window

You can view all your project files in one single editor window, you can open multiple views of the same file, or you can open multiple views of different files.

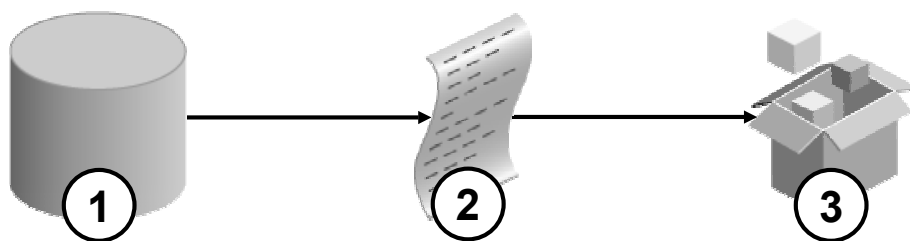
The tabs at the top of the editor window are the document tabs. Selecting a document tab gives that file focus, bringing it to the foreground of the window in the current editor.

The tabs at the bottom of the editor window for a given file are the editor tabs. Selecting an editor tab opens the file in that editor.

## Deploying Java Stored Procedures

**Before deploying Java stored procedures, perform the following steps:**

- 1. Create a database connection.**
- 2. Create a deployment profile.**
- 3. Deploy the objects.**



ORACLE

Copyright © 2006, Oracle. All rights reserved.

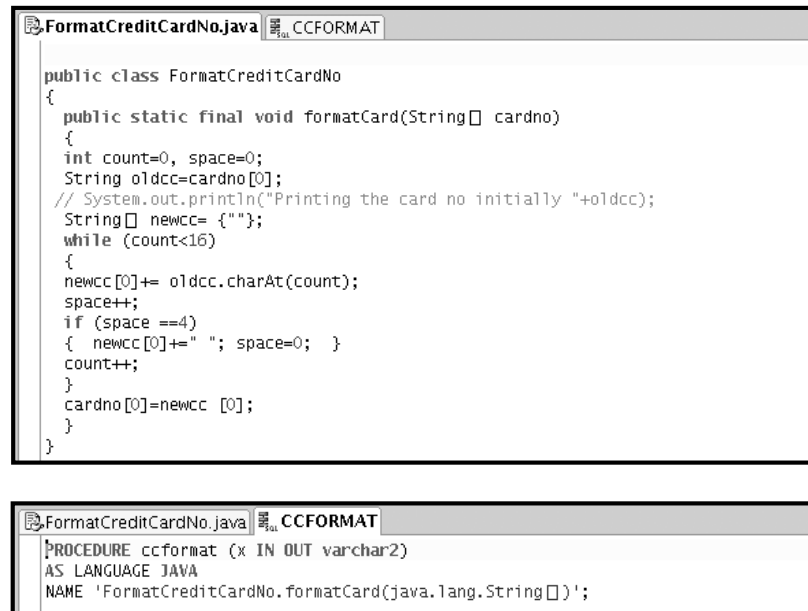
### Deploying Java Stored Procedures

Create a deployment profile for Java stored procedures, then deploy the classes and, optionally, any public static methods in JDeveloper using the settings in the profile.

Deploying to the database uses the information provided in the Deployment Profile Wizard and two Oracle Database utilities:

- `loadjava` loads the Java class containing the stored procedures to an Oracle database.
- `publish` generates the PL/SQL call specific wrappers for the loaded public static methods. Publishing enables the Java methods to be called as PL/SQL functions or procedures.

# Publishing Java to PL/SQL



ORACLE


Copyright © 2006, Oracle. All rights reserved.

## Publishing Java to PL/SQL

The slide shows the Java code and how to publish the Java code in a PL/SQL procedure.



# Creating Program Units

A screenshot of a code editor window. The title bar reads 'TEST\_JDEV'. The code inside is a PL/SQL function skeleton:

```
FUNCTION "TEST_JDEV" RETURN VARCHAR2
AS
BEGIN
    RETURN('');
END;
```

**Skeleton of the function**

ORACLE

Copyright © 2006, Oracle. All rights reserved.

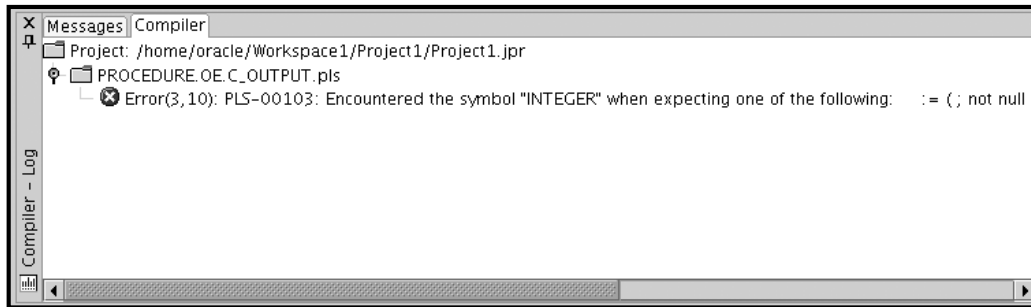
## Creating Program Units

To create a PL/SQL program unit:

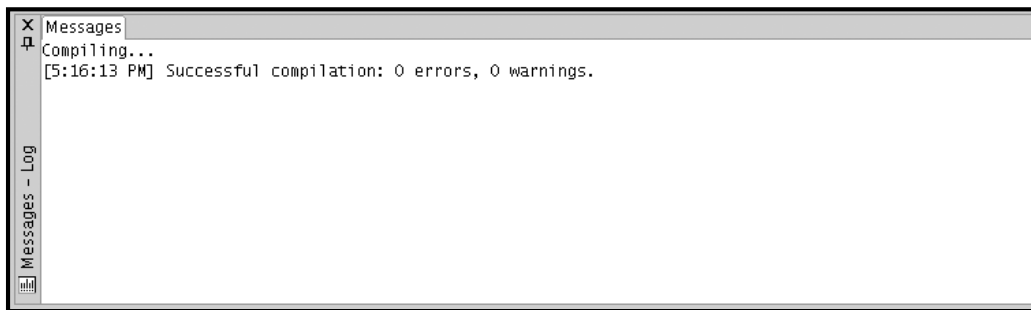
1. Select View > Connection Navigator.
2. Expand Database and select a database connection.
3. In the connection, expand a schema.
4. Right-click a folder corresponding to the object type (Procedures, Packages, and Functions).
5. Choose New PL/SQL object\_type. The Create PL/SQL dialog box appears for the function, package, or procedure.
6. Enter a valid name for the function, package, or procedure, and click OK.

A skeleton definition will be created and opened in the Code Editor. You can then edit the subprogram to suit your need.

# Compiling



## Compilation with errors



## Compilation without errors

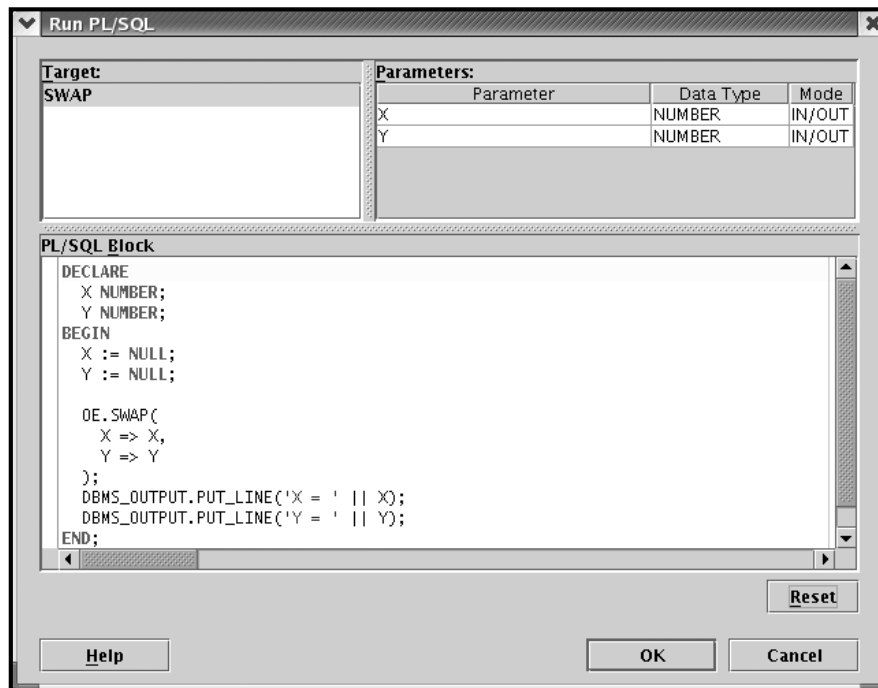
ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Compiling

After editing the skeleton definition, you need to compile the program unit. Right-click the PL/SQL object that you need to compile in the Connection Navigator and then select Compile. Alternatively, you can also press [CTRL] + [SHIFT] + [F9] to compile.

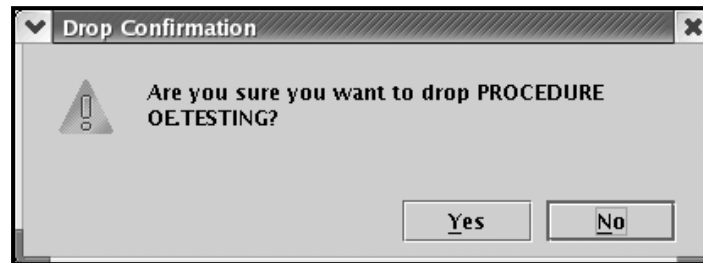
# Running a Program Unit



## Running a Program Unit

To execute the program unit, right-click the object and click Run. The Run PL/SQL dialog box appears. You may need to change the NULL values with reasonable values that are passed into the program unit. After you change the values, click OK. The output will be displayed in the Message-Log window.

# Dropping a Program Unit



ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Dropping a Program Unit

To drop a program unit, right-click the object and select Drop. The Drop Confirmation dialog box appears; click Yes. The object will be dropped from the database.

## Debugging PL/SQL Programs

- **JDeveloper support two types of debugging:**
  - **Local**
  - **Remote**
- **You need the following privileges to perform PL/SQL debugging:**
  - **DEBUG ANY PROCEDURE**
  - **DEBUG CONNECT SESSION**

ORACLE

Copyright © 2006, Oracle. All rights reserved.

### Debugging PL/SQL Programs

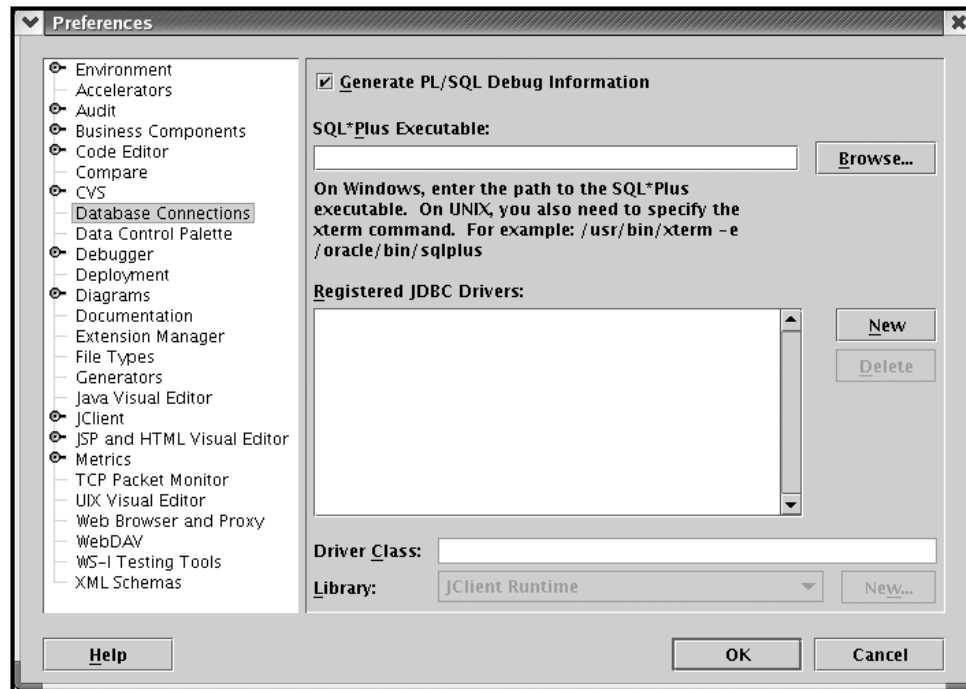
JDeveloper offers both local and remote debugging. A local debugging session is started by setting breakpoints in source files, and then starting the debugger. Remote debugging requires two JDeveloper processes: a debugger and a debuggee, which may reside on a different platform.

To debug a PL/SQL program, it must be compiled in `INTERPRETED` mode. You cannot debug a PL/SQL program that is compiled in `NATIVE` mode. This mode is set in the database's `init.ora` file.

PL/SQL programs must be compiled with the `DEBUG` option enabled. This option can be enabled using various ways. Using `SQL*Plus`, execute `ALTER SESSION SET PLSQL_DEBUG = true` to enable the `DEBUG` option. Then you can create or recompile the PL/SQL program you want to debug. Another way of enabling the `DEBUG` option is by using the following command in `SQL*Plus`:

```
ALTER <procedure, function, package> <name> COMPILE DEBUG;
```

# Debugging PL/SQL Programs



ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Debugging PL/SQL Programs (continued)

Before you start with debugging, make sure that the Generate PL/SQL Debug Information check box is selected. You can access the dialog box by using Tools > Preferences > Database Connections.

Instead of manually testing PL/SQL functions and procedures as you may be accustomed to doing from within SQL\*Plus or by running a dummy procedure in the database, JDeveloper enables you to test these objects in an automatic way. With this release of JDeveloper, you can run and debug PL/SQL program units. For example, you can specify parameters being passed or return values from a function giving you more control over what is run and providing you output details about what was tested.

**Note:** The procedures or functions in the Oracle database can be either stand-alone or within a package.

## Debugging PL/SQL Programs (continued)

To run or debug functions, procedures, or packages, perform the following steps:

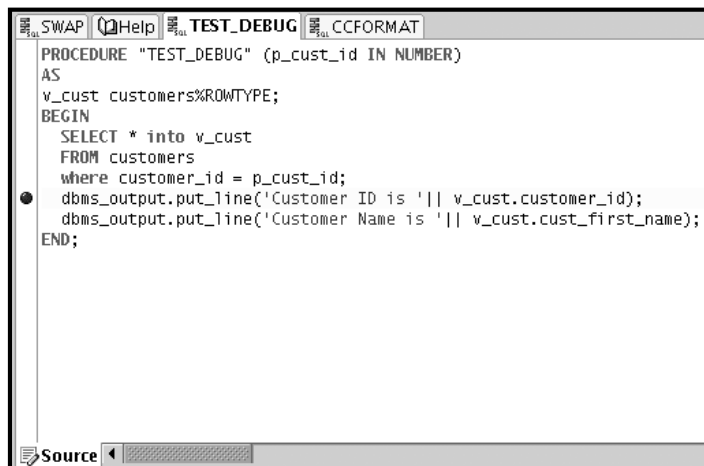
1. Create a database connection by using the Database Wizard.
2. In the Navigator, expand the Database node to display the specific database username and schema name.
3. Expand the Schema node.
4. Expand the appropriate node depending on what you are debugging: Procedure, Function, or Package body.
5. (Optional for debugging only) Select the function, procedure, or package that you want to debug and double-click to open it in the Code Editor.
6. (Optional for debugging only) Set a breakpoint in your PL/SQL code by clicking to the left of the margin.

**Note:** The breakpoint must be set on an executable line of code. If the debugger does not stop, the breakpoint may have not been set on an executable line of code (ensure that the breakpoint was verified). Also, verify that the debugging PL/SQL prerequisites were met. In particular, make sure that the PL/SQL program is compiled in INTERPRETED mode.

7. Make sure that either the Code Editor or the procedure in the Navigator is currently selected.
8. Click the Debug toolbar button; or, if you want to run without debugging, click the Run toolbar button.
9. The Run PL/SQL dialog box is displayed.
  - Select a target that is the name of the procedure or function that you want to debug. Note that the content in the Parameters and PL/SQL Block boxes change dynamically when the target changes.
  - Note:** You will have a choice of target only if you choose to run or debug a package that contains more than one program unit.
  - The Parameters box lists the target's arguments (if applicable).
  - The PL/SQL Block box displays code that was custom-generated by JDeveloper for the selected target. Depending on what the function or procedure does, you may need to replace the NULL values with reasonable values so that these are passed into the procedure, function, or package. In some cases, you may need to write additional code to initialize values to be passed as arguments. In this case, you can edit the PL/SQL block text as necessary.
10. Click OK to execute or debug the target.
11. Analyze the output information displayed in the Log window.

In the case of functions, the return value will be displayed. DBMS\_OUTPUT messages will also be displayed.

## Setting Breakpoints



ORACLE

Copyright © 2006, Oracle. All rights reserved.

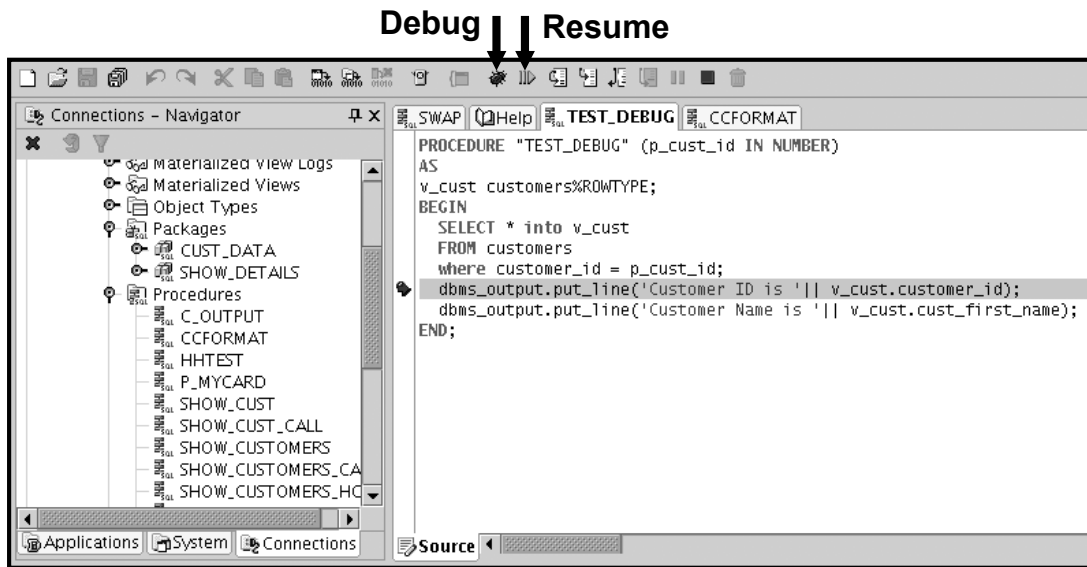
### Setting Breakpoints

Breakpoints help you to examine the values of the variables in your program. It is a trigger in a program that, when reached, pauses program execution allowing you to examine the values of some or all of the program variables. By setting breakpoints in potential problem areas of your source code, you can run your program until its execution reaches a location you want to debug. When your program execution encounters a breakpoint, the program pauses, and the debugger displays the line containing the breakpoint in the Code Editor. You can then use the debugger to view the state of your program. Breakpoints are flexible in that they can be set before you begin a program run or at any time while you are debugging.

To set a breakpoint in the Code Editor, click the left margin next to a line of executable code. Breakpoints set on comment lines, blank lines, declaration, and any other nonexecutable lines of code are not verified by the debugger and are treated as invalid.



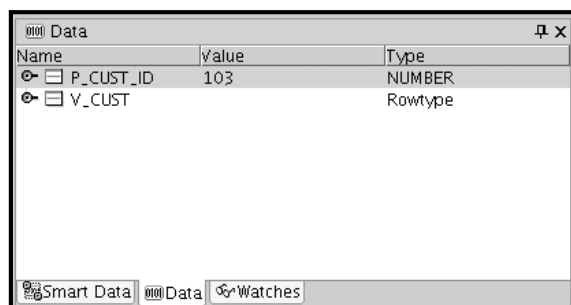
# Stepping Through Code



## Stepping Through Code

After setting the breakpoint, start the debugger by clicking the Debug icon. The debugger will pause the program execution at the point where the breakpoint is set. At this point, you can check the values of the variables. You can continue with the program execution by clicking the Resume icon. The debugger will then move on to the next breakpoint. After executing all the breakpoints, the debugger will stop the execution of the program and display the results in the Debugging – Log area.

## Examining and Modifying Variables



**Data window**

ORACLE

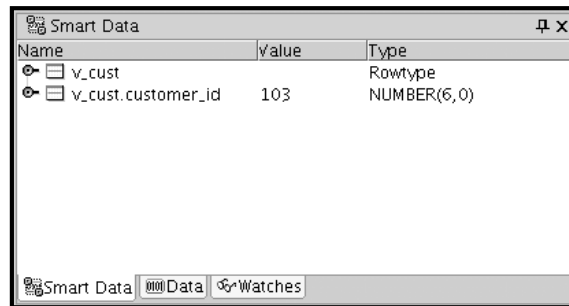
Copyright © 2006, Oracle. All rights reserved.

### Examining and Modifying Variables

When the debugger is ON, you can examine and modify the value of the variables using the Data, Smart Data, and Watches windows. You can modify program data values during a debugging session as a way to test hypothetical bug fixes during a program run. If you find that a modification fixes a program error, you can exit the debugging session, fix your program code accordingly, and recompile the program to make the fix permanent.

You use the Data window to display information about variables in your program. The Data window displays the arguments, local variables, and static fields for the current context, which is controlled by the selection in the Stack window. If you move to a new context, the Data window is updated to show the data for the new context. If the current program was compiled without debug information, you will not be able to see the local variables.

# Examining and Modifying Variables



**Smart Data window**

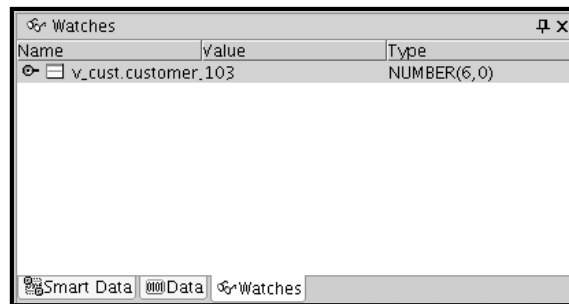
ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Examining and Modifying Variables (continued)

Unlike the Data window that displays all the variables in your program, the Smart Data window displays only the data that is relevant to the source code that you are stepping through.

# Examining and Modifying Variables



**Watches window**

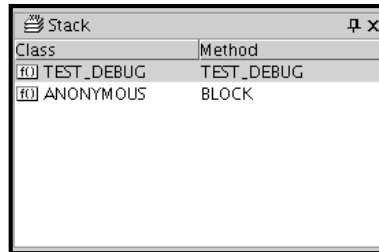
ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Examining and Modifying Variables (continued)

A watch enables you to monitor the changing values of variables or expressions as your program runs. After you enter a watch expression, the Watch window displays the current value of the expression. As your program runs, the value of the watch changes as your program updates the values of the variables in the watch expression.

# Examining and Modifying Variables



**Stack window**

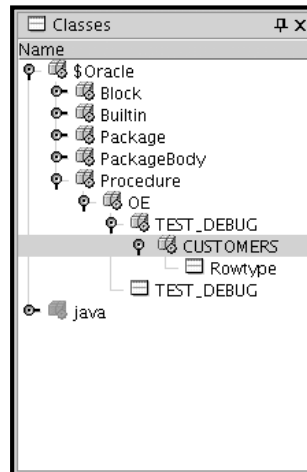
ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Examining and Modifying Variables (continued)

You can activate the Stack window by using View > Debugger > Stack. It displays the call stack for the current thread. When you select a line in the Stack window, the Data window, Watch window, and all other windows are updated to show data for the selected class.

# Examining and Modifying Variables



**Classes window**

ORACLE

Copyright © 2006, Oracle. All rights reserved.

## Examining and Modifying Variables (continued)

The Classes window displays all the classes that are currently being loaded to execute the program. If used with Oracle Java Virtual Machine (OJVM), it also shows the number of instances of a class and the memory used by those instances.