Table of Contents

[CHAPTER 21: JAVA STORED PROCEDURE 2](#_Toc415997968)

[Theory 2](#_Toc415997969)

[AIM 3](#_Toc415997970)

[Lab Exercise 21: JAVA STORED PROCEDURE 4](#_Toc415997971)

[1. Simple Java Stored Procedure 5](#_Toc415997972)

[2. Access Database from Java 8](#_Toc415997973)

[3. Mapping Java Class to Object Type 14](#_Toc415997974)

[4. Java Exception 21](#_Toc415997975)

[SUMMARY 24](#_Toc415997976)

[REFERENCES 25](#_Toc415997977)

[INDEX 26](#_Toc415997978)

# CHAPTER 21: JAVA STORED PROCEDURE

## Theory

Oracle supports storing and executing Java code inside the database. For this purpose, Oracle introduces Oracle JVM. The Oracle JVM is a standard, Java-compatible environment that runs any pure Java application. The Oracle JVM provides a run-time environment for Java objects. It also supports all the core Java class libraries, including *java.lang*, *java.io*, *java.net*, *java.math*, and *java.util*. However, it cannot provide GUIs.

The Oracle JVM is embedded within Oracle Database. The database session is the environment in which the Oracle JVM runs and as such is the analog of the operating system process in which a standard client JVM runs. Oracle Java applications are not restricted to a single top-level main entrypoint, as in standard "*main*" method in a client java application.

To use a Java in Oracle database, you need to follow the steps below:

1. Create a well-written Java file.
2. Load and compile the file into the database using "" tool.
3. Create a PL/SQL subprogram as interface of the Java function.
4. Call Java Stored Procedure.

Java Stored Procedure can access database objects: tables, views, and other stored procedures. In the next lab, you will learn about how to create a connection, execute the query and fetch a result set.

Oracle supports a very good mapping between Java class and Object Type. Instead of map each method to a standalone function/procedure, it is better to think of gathering all related methods in one Object Type. Moreover, Oracle makes it easier to map and use methods by mapping Object attributes to Java class's attributes as well. This means the SELF in Object Type will map directly to "this" in a Java object instance. This also means that you do not need to define Java class's methods as "static". In return, Java class must define implement "SQLData" interface, which does the attributes mapping for you.

## AIM

The AIM of the following exercise is to use Java Stored Procedure.

The steps involved will include:

* Simple Java Stored Procedure
* Access Database From Java
* Mapping Java Class to Object Type
* Java Exception

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.
* Object Types
* Object Type Storage
* Object Type Manipulation
* Advanced Object Type Usage

Estimated Completion Time:

30 minutes

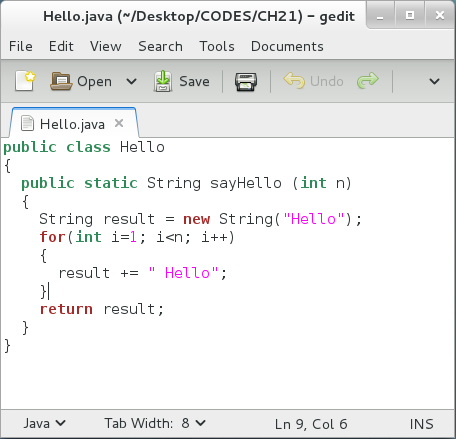
# Lab Exercise 21: JAVA STORED PROCEDURE

|  |
| --- |
|  |

## Simple Java Stored Procedure

**Step 1:** Open any text editor and create "Hello.java" file as shown below:

|  |  |
| --- | --- |
| Command | Description |
| public class Hello | "Hello" public class. |
| { |
| public **static** String sayHello (int n) | "**static**" method which returns a "String" and take "int". It method simply returns "Hello" word repeated "n" times. |
| { |
| String result = new String("Hello"); |
| for(int i=1; i<n; i++) |
| { |
| result += " Hello"; |
| } |
| return result; |
| } |
| } |  |

****

**Please note:** The file name should match the class name "Hello". The method must be static to map to a standalone method. In our case, it should be "FUNCTION" because it returns a value. If it does not return anything, "void", then it should be mapped to "PROCEDURE".

**Step 2:** Load compile you Java file in your database, "HR" schema, using the following commands:

|  |  |
| --- | --- |
| Command | Description |
| loadjava -u hr@localhost:1521:DB12c -v -r -t '/home/oracle/Desktop/CODES/CH21/Hello.java' | Load java file in your database "DB12c", "HR" schema |
| oracle | "HR" password. |

****

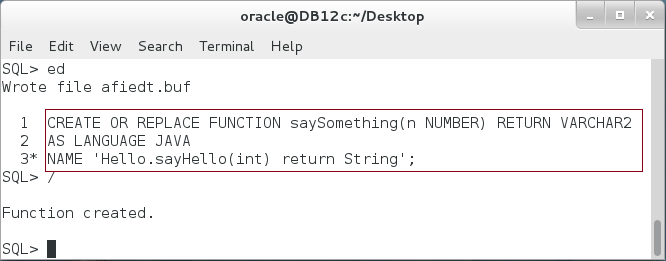
**Step 3:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

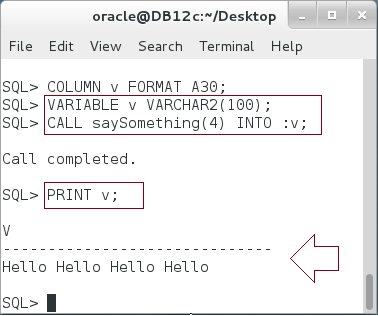
**Step 4:** Create SQL program interface for your Java class, so you can use it in other PL/SQL blocks. In this case, we chose to create a standalone FUNCTION as interface. Execute the following DDL statement:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE FUNCTION saySomething(n NUMBER) RETURN VARCHAR2 |  |
| **AS LANGUAGE JAVA** |  |
| **NAME** 'Hello.sayHello(int) return String'; |  |
| / |  |



**Step 5:** You can now using the FUNCTION as usual. For example, use the following block:

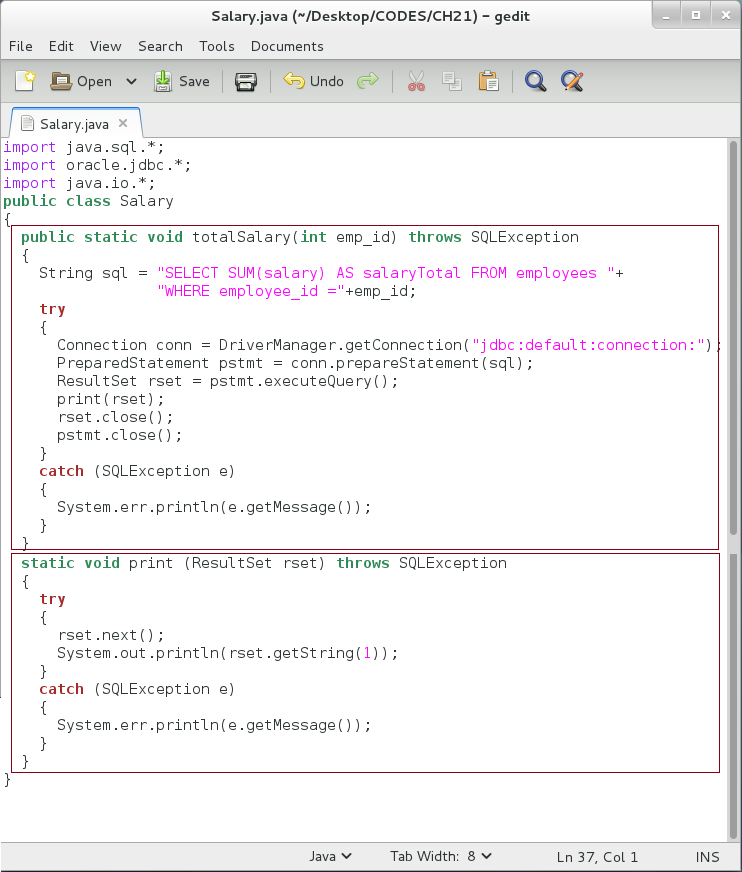
|  |  |
| --- | --- |
| Command | Description |
| COLUMN v FORMAT A30; | format a column [optional] |
| VARIABLE v VARCHAR2(100) | define SQL\*Plus variable |
| CALL saySomething(4) INTO :v; | Call the function |
| PRINT v; | Print the return value. |



## Access Database from Java

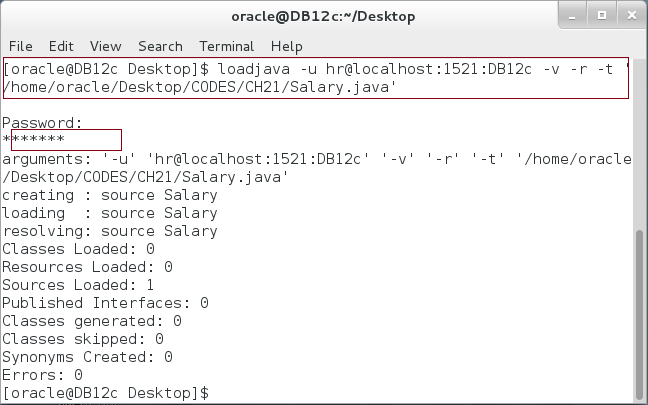
**Step 1:** Typically, you don't need a Java Class to be completely separated from the database user session. In most cases, Java Class should access the database Objects such as Tables, Views, and others. You should follow the steps appeared in the last section to create, load, and define SQL interface. In this section, however, we will draw more attention on how you should access the user session from a Java Class. Create "Salary.java" file as shown below:

|  |  |
| --- | --- |
| Command | Description |
| import java.sql.\*; |  |
| import oracle.jdbc.\*; |  |
| import java.io.\*; |  |
| **public class** Salary |  |
| { |  |
| **public static** void **totalSalary**(int emp\_id) throws SQLException | Declare "**totalSalary**" procedure. |
| { |
| String sql = "SELECT SUM(salary) AS salaryTotal FROM employees "+ | Define your SQL statement |
| "WHERE employee\_id ="+emp\_id; |
| try |  |
| { |  |
| Connection conn = DriverManager**.getConnection**("**jdbc:default:connection:**"); | use " **jdbc:default:connection:**" to access the current user session. |
| PreparedStatement pstmt = conn.**prepareStatement**(sql); | Prepare the SQL statement and execute it. |
| ResultSet rset = pstmt.**executeQuery**(); |
| print(rset); |
| rset.close(); |
| pstmt.close(); |
| } |  |
| catch (SQLException e) |  |
| { |  |
| System.err.println(e.getMessage()); |  |
| } |  |
| } |  |
| static void **print** (ResultSet rset) throws SQLException | Helper function to print the result set value. In our case, there is one value. Use "ResultSet.next()" function to navigate to the next row in the result set. |
| { |
| try |
| { |
| rset.**next()**; |
| System.out.println(rset**.getString(1)**); |
| } |
| catch (SQLException e) |
| { |
| System.err.println(e.getMessage()); |
| } |
| } |
| } |  |

****

**Step 2:** Open a new Terminal and load a java class in your database as shown below:

|  |  |
| --- | --- |
| Command | Description |
| loadjava -u hr@localhost:1521:DB12c -v -r -t '/home/oracle/Desktop/CODES/CH21/Salary.java' | load java file into your database. |
| oracle | "HR" password. |



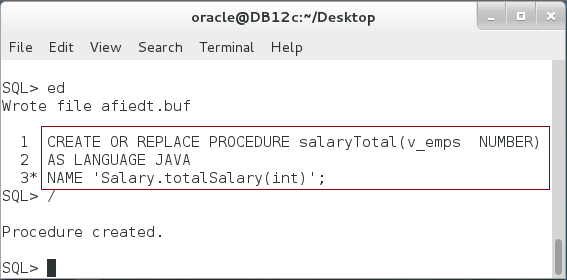
**Step 3:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

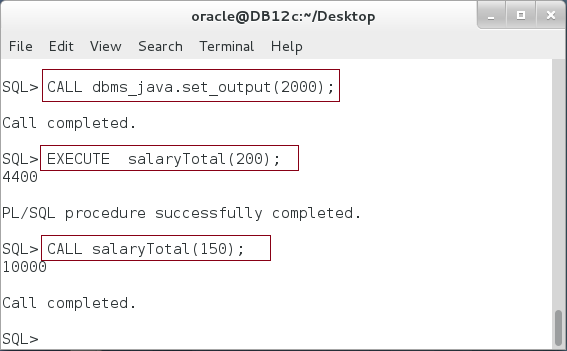
**Step 4:** In the previous case, you should use the NOT INSTANTIABLE object and AREA method. Re-Create the Object specification as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE PROCEDURE salaryTotal(v\_emps NUMBER) |  |
| **AS LANGUAGE JAVA** |  |
| **NAME 'Salary.totalSalary(int)';** |



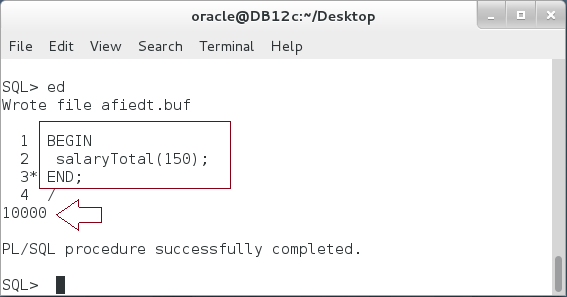
**Step 5:** Use a standalone procedure as usual. For example, as shown below:

|  |  |
| --- | --- |
| Line |  |
| CALL dbms\_java.set\_output(2000); | Use this to allow user output to SQL\*Plus buffer. |
| EXECUTE salaryTotal(200); | EXECUTE or CALL command to execute procedure |
| CALL salaryTotal(150); |



**Step 6:** Alternatively, you may use a PL/SQL block to access the procedure as shown below:

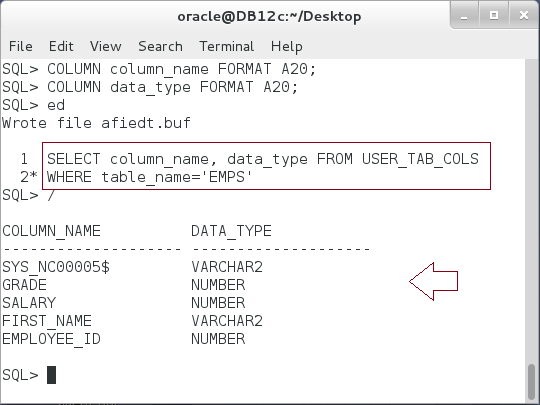
|  |  |
| --- | --- |
| Line |  |
| BEGIN |  |
| salaryTotal(150); |  |
| END; |
| / |



## Mapping Java Class to Object Type

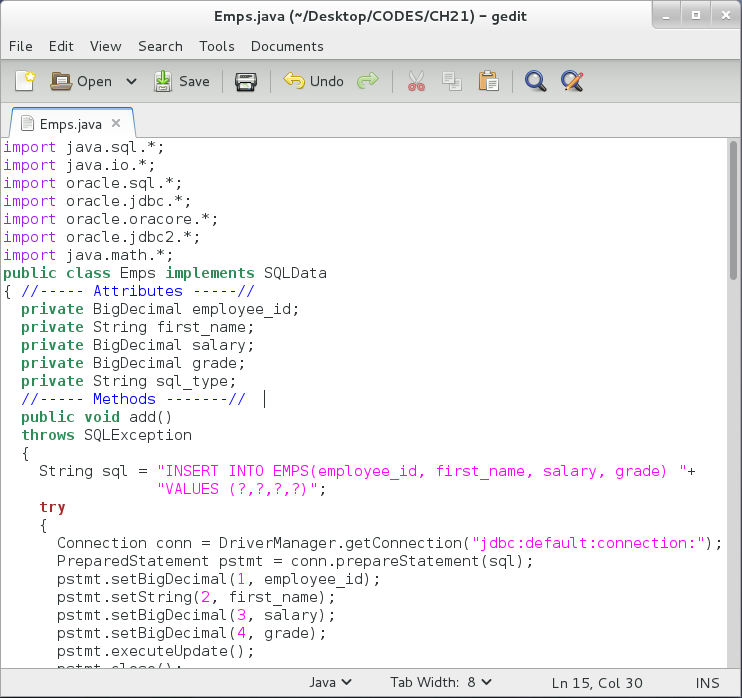
**Step 1:** We are going to use one of the tables created previously in the last chapters, "**EMPS**". Connect to "HR" schema and execute the following:

|  |  |
| --- | --- |
| Line | Description |
| COLUMN column\_name FORMAT A20; |  |
| COLUMN data\_type FORMAT A20; |  |
| SELECT column\_name, data\_type FROM USER\_TAB\_COLS | List "EMPS" columns. |
| WHERE table\_name='EMPS' |
| / |  |



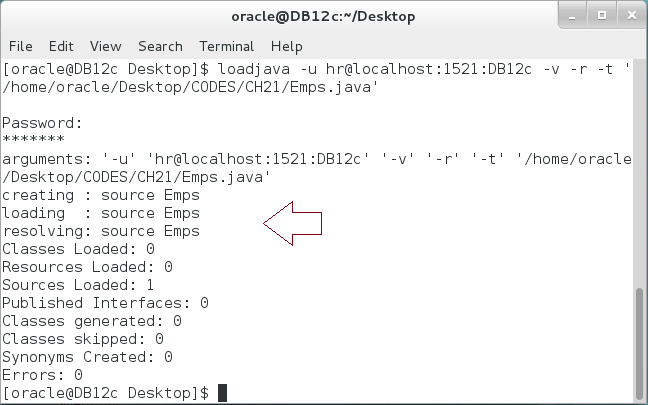
**Step 2:** Accordingly, you create a Java Class with attributes and common methods such as add, modify, and remove employee. The class implements **SQLData** interface. Therefore, the class should implement three methods: **getSQLTypeName, writeSQL, and readSQL.** These methods are responsible for synchronizing between Java class's attributes and database Object Type's attributes. Please also note that your methods, **add, modify, and remove,** are not static. They don't have to have parameters; they use the class attributes which are synchronized with Object Type. The class is long; ask your trainer to give you the class:

|  |  |
| --- | --- |
| Line | Description |
| import java.sql.\*; |  |
| import java.io.\*; |  |
| import oracle.sql.\*; |  |
| import oracle.jdbc.\*; |  |
| import oracle.oracore.\*; |  |
| import oracle.jdbc2.\*; |  |
| import java.math.\*; |  |
| **public class** Emps **implements SQLData** |  |
| { //----- Attributes -----// | Attributes |
| private BigDecimal employee\_id; |
| private String first\_name; |
| private BigDecimal salary; |
| private BigDecimal grade; |
| private String sql\_type; |  |
| //----- Methods -------// |  |
| **public void add()** throws SQLException | Add new employee using "add" procedure. Please note a parameterize query using "?". This is a best practice, according to Oracle. Since it enhance the query performance and avoid the SQL injection. |
| { |
| String sql = "INSERT INTO EMPS(employee\_id, first\_name, salary, grade) "+ |
| "VALUES (?,?,?,?)"; |
| try |
| { |
| Connection conn = DriverManager.getConnection("jdbc:default:connection:"); |
| PreparedStatement pstmt = conn.prepareStatement(sql); |
| pstmt.setBigDecimal(1, employee\_id); |
| pstmt.setString(2, first\_name); |
| pstmt.setBigDecimal(3, salary); |
| pstmt.setBigDecimal(4, grade); |
| pstmt.executeUpdate(); |
| pstmt.close(); |
| } |
| catch (SQLException e) |
| { |
| System.err.println(e.getMessage()); |
| } |
| } |  |
| **public void modify()** throws SQLException | Modify a record using "modify" method. Please note the order of the parameter. You should follow the same order appearing in the SQL statement. |
| { |
| String sql = "UPDATE EMPS "+ |
| "SET first\_name= ?, salary=?, grade=? "+ |
| "WHERE employee\_id=?"; |
| try |
| { |
| Connection conn = DriverManager.getConnection("jdbc:default:connection:"); |
| PreparedStatement pstmt = conn.prepareStatement(sql); |
| pstmt.setString(1, first\_name); |
| pstmt.setBigDecimal(2, salary); |
| pstmt.setBigDecimal(3, grade); |
| pstmt.setBigDecimal(4, employee\_id); |
| pstmt.executeUpdate(); |
| pstmt.close(); |
| } |
| catch (SQLException e) |
| { |
| System.err.println(e.getMessage()); |
| } |
| } |
| **public void remove()** throws SQLException | Delete a record from Emps table using "remove" procedure. |
| { |
| String sql = "DELETE FROM EMPS "+ |
| "WHERE employee\_id=?"; |
| try |
| { |
| Connection conn = DriverManager.getConnection("jdbc:default:connection:"); |
| PreparedStatement pstmt = conn.prepareStatement(sql); |
| pstmt.setBigDecimal(1, employee\_id); |
| pstmt.executeUpdate(); |
| pstmt.close(); |
| } |
| catch (SQLException e) |
| { |
| System.err.println(e.getMessage()); |
| } |
| } |
| **public String getSQLTypeName()** throws SQLException | This is to implement "**SQLData**" interface. |
| { |
| return sql\_type; |
| } |
| **public void readSQL**(SQLInput stream, String typeName) throws SQLException | This is to implement "**SQLData**" interface. Don't forget to write your attribute instead. |
| { |
| sql\_type = typeName; |
| employee\_id = stream.readBigDecimal(); |
| first\_name = stream.readString(); |
| salary = stream.readBigDecimal(); |
| grade = stream.readBigDecimal(); |
| } |
| **public void writeSQL**(SQLOutput stream) throws SQLException | This is to implement "**SQLData**" interface. Don't forget to write your attribute instead. |
| { |
| stream.writeBigDecimal(employee\_id); |
| stream.writeString(first\_name); |
| stream.writeBigDecimal(salary); |
| stream.writeBigDecimal(grade); |
| } |
| } |

****

**Step 3:** Load the Java class in your database as shown:

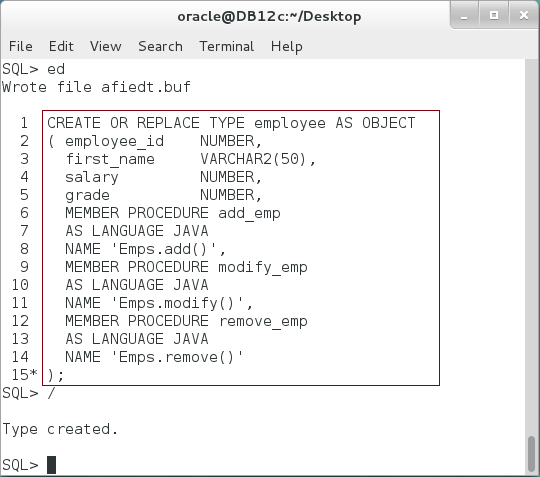
|  |  |
| --- | --- |
| Line | Description |
| loadjava -u hr@localhost:1521:DB12c -v -r -t '/home/oracle/Desktop/CODES/CH21/Emps.java' | Load Java into your database |
| oracle | "HR" password |

****

**Please note:** You should follow Java-to-Oracle data type mapping. For example, Oracle data type **NUMBER** should be mapped to "**java.math.BigDecimal**". Otherwise, you will receive data-type conversion error. Please review the mapping instructions here: http://docs.oracle.com/cd/B19306\_01/java.102/b14188/datamap.htm.

**Step 4:** Connect to "**HR**" and create "**employee**" Object Types as shown below:

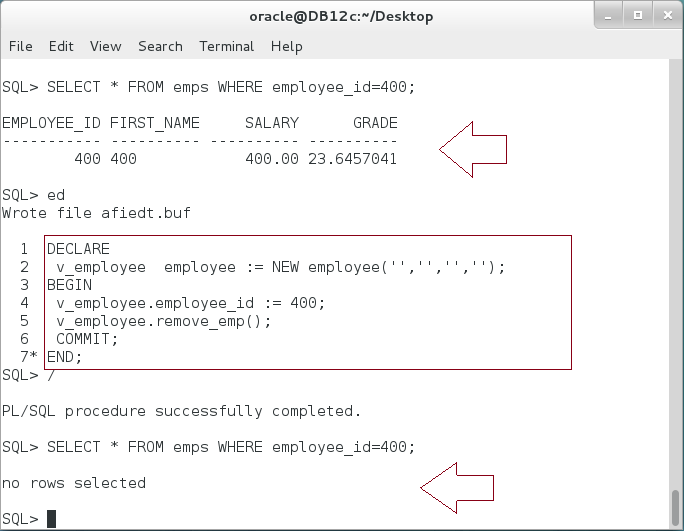
|  |
| --- |
| Line |
| CREATE OR REPLACE TYPE employee AS OBJECT |
| ( employee\_id NUMBER, |
| first\_name VARCHAR2(50), |
| salary NUMBER, |
| grade NUMBER, |
| MEMBER **PROCEDURE** add\_emp |
| **AS LANGUAGE JAVA** |
| **NAME** 'Emps.add()', |
| MEMBER **PROCEDURE** modify\_emp |
| **AS LANGUAGE JAVA** |
| **NAME** 'Emps.modify()', |
| MEMBER **PROCEDURE** remove\_emp |
| **AS LANGUAGE JAVA** |
| **NAME** 'Emps.remove()' |
| ); |
| / |



**Please note:** You don't need to create the Object Type Body. You've just linked between Object Type's methods and the Java Class using Object Type specification.

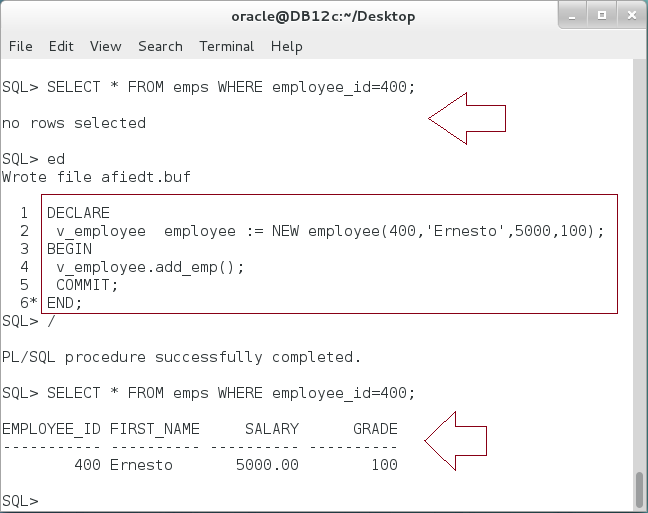
**Step 5:** Use the Object Type to delete some record as shown:

|  |  |
| --- | --- |
| Line | Description |
| SELECT \* FROM emps WHERE employee\_id=400; | Query employee id =400 |
| / |  |
| DECLARE |  |
| v\_employee employee := NEW employee('','','',''); | Delete employee id 400 using "remove\_emp" procedure. |
| BEGIN |
| v\_employee.employee\_id := 400; |
| v\_employee.remove\_emp(); |
| COMMIT; |
| END; |
| / |  |
| SELECT \* FROM emps WHERE employee\_id=400; | Re-query the employee again |
| / |



**Step 6:** Use the Object Type to add some record as shown:

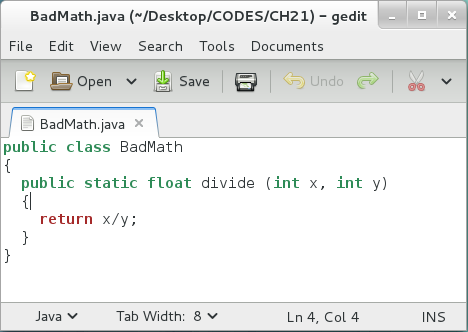
|  |  |
| --- | --- |
| Line | Description |
| SELECT \* FROM emps WHERE employee\_id=400 | Query employee id =400 |
| / |  |
| DECLARE |  |
| v\_employee employee := NEW employee(400,'Ernesto',5000,100); | Add employee id 400 using "add\_emp" procedure. |
| BEGIN |
| v\_employee**.add\_emp**(); |
| COMMIT; |
| END; |
| / |
| SELECT \* FROM emps WHERE employee\_id=400 |  |
| / |  |



## Java Exception

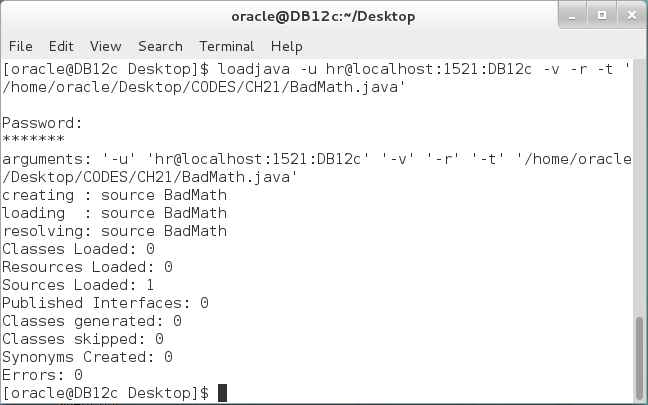
**Step 1:** Create a simple Java Class to divide two numbers as shown below:

|  |  |
| --- | --- |
| Line | Description |
| public class **BadMath** |  |
| { |  |
| public static float **divide** (int x, int y) |  |
| { |  |
| return x/y; |  |
| } |  |
| } |  |



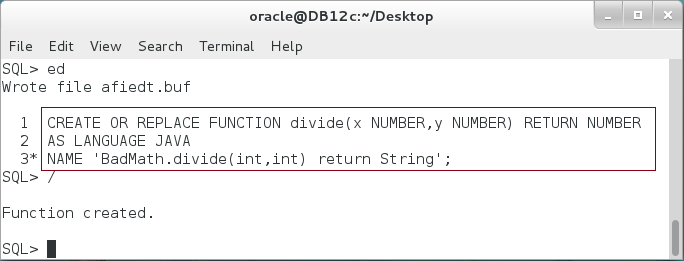
**Step 2:** Load the java file as shown:

|  |  |
| --- | --- |
| Line | Description |
| loadjava -u hr@localhost:1521:DB12c -v -r -t '/home/oracle/Desktop/CODES/CH21/BadMath.java' |  |
| oracle |  |



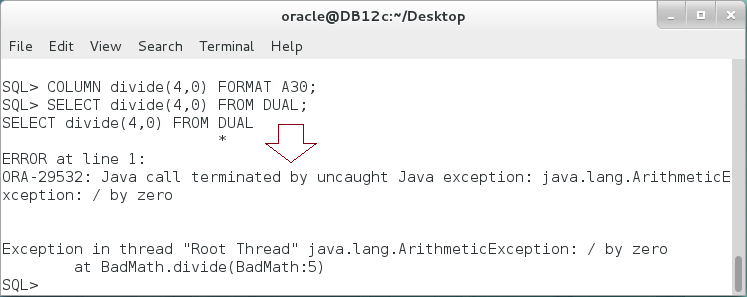
**Step 3:** Create a standalone function as SQL interface of your java class as shown:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE FUNCTION divide(x NUMBER,y NUMBER) RETURN NUMBER |  |
| AS LANGUAGE JAVA |  |
| NAME 'BadMath.divide(int,int) return String' |  |
| / |  |

****

**Step 4:** Execute the function as shown below:

|  |  |
| --- | --- |
| Line | Description |
| COLUMN divide(4,0) FORMAT A30; |  |
| SELECT divide(4,0) FROM DUAL; |  |

****

**Please note:** All Java Exceptions propagate to Oracle stored procedure on "ORA-29532". You should be either handle the error specifically in the Java class or generally in Oracle stored procedure.

# SUMMARY

Object supports programming using external programming languages, in specific Java. For better performance and security, Oracle stores, compiles, and runs Java classes inside database. Java sources are managed in Oracle JVM, which are responsible for running and linking between Java session and Database session. In order to use Java class, you should first create a class you want. Second, load it to Oracle Database schema using "loadjava" utility. Third, create an SQL interface for your Java source. Finally, use your SQL interface in your PL/SQL codes as usual. Oracle supports using standalone subprograms, packages, and Object Types as interfaces for Java Classes. In this chapter, we explored standalone and Object Type as interface. Using Object Type, in specific, brings new mapping concept, Object Type to Java Class mapping. Accordingly, each Java class corresponds to one Object Type which includes attributes synchronization. Therefore, you may use Java attributes as if they were Object Type's attributes.

After completing this lab exercise, you should be able to use Java Stored Procedure with different SQL interface.

# REFERENCES

* https://docs.oracle.com/database/121/JJDEV/chfive.htm#JJDEV13234
* https://docs.oracle.com/database/121/JJDEV/cheight.htm#JJDEV13295
* http://docs.oracle.com/cd/B19306\_01/java.102/b14188/datamap.htm

# INDEX

Java Stored Procedure 2, 4, 6, 25

Oracle JVM 2, 25