RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Informācijas tehnoloģijas institūts

Technology of Large Data

Castillo Torres Miguel Eduardo

Second practical work:

SQL + JAVA

Semester fall 2015

Contents

1. Small Java class program definition, compilation and output

Although I have been working without the Ortus' database, for this third task the work will use the database offered by Ortus (Computer Sciencie Faculty). Since the first task we have working with PL/SQL language, so now I will focus more in the Java language and how to integrate it with Oracle, that's why I decided to use Ortus' databse.
To show the same Java language basics from the beginning (As with PL/SQL), I will set up a small class that will create the program object type in my database, analogous objects and place them in the array from which data output will take place. I will write the program in a notepad ++ environment.

Text in blue and bold: SQL code
Text in red and bold: Java code

Objects in my database

```
create or replace type PROGRAM as object (
        number_id number(3,0),
        Name varchar2(30),
        Clasification varchar2(30)
);
```

File content (Definition):

Name_Surname.java

From the beginning, define a class of objects (like database object types)

```
class Program{
        private int number_id;
        private String Name;
        private String Clasification;
}
```

Define constructors similar as defining the type of database objects.

```
public Program(int i, String a, String b) {
        number_id=i;
        Name = a;
        Clasification = b;
}
```

To output the values, we need to create specific methods wich will return the objects.

```
public int getID() {return Ident_numurs;}
public String getNos() {return Nosaukums;}
public String getKlas() {return Klasifikacija;} }
public class Name_Surname{
public static void main(String[] args){
```

Now the program will expect an array

```
Program Programs[]= new Program[3];
Programs[0]= new Program(1,"Miguel","Student");
Programs[1]= new Program(2,"Abimale","Student");
Programs[2]= new Program(3,"Arturo","Chemist");
```

Now, to show the output

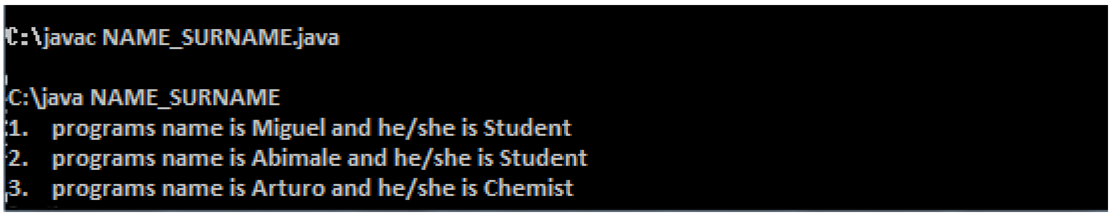
```
for (Program p : Programs)
        System.out.println(p.getID() + "programs Name is " + p.getNos() + "and
he/she is" + p.getKlas());
}
}
```

Then, on the Windows Command Line, I entered:

```
javac PIRM_UZD.java. Javac.exe
```

To compiled and create the java file. After that, Name_Surname the java command compiles the program and launches it. It is important to mention that the JDK or Java Development Kit starts the java files, but it does not install java applications. As we can remember for our lectures, JDK's task is to execute the compilation and support java programs.

The result:



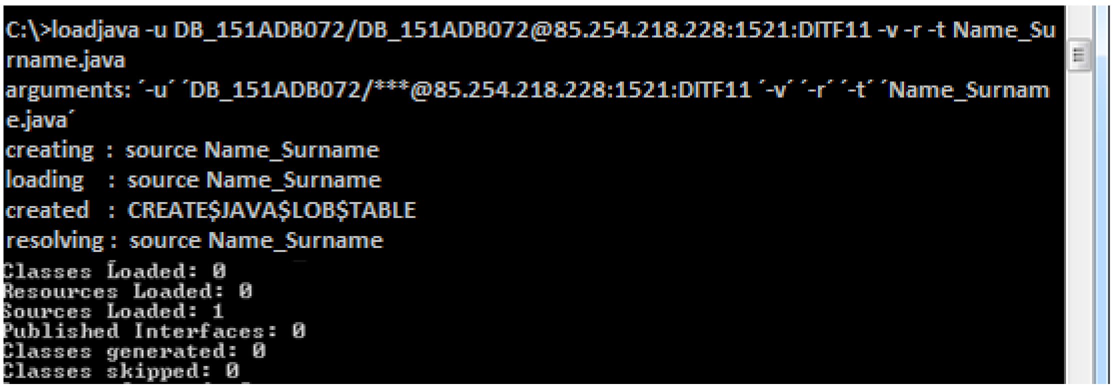
```
C:\javac NAME_SURNAME.java

C:\java NAME_SURNAME
1.   programs name is Miguel and he/she is Student
2.   programs name is Abimale and he/she is Student
3.   programs name is Arturo and he/she is Chemist
```

2. LOADJAVA for downloading data from a Database and CREATE JAVA

In order to use LOADJAVA, you must install the Oracle client, where the bin folder is located (loadjava.bat program). For this purpose, I use the Oracle 11g R2 client. I download previously, and after that, I created Name_Surname.java source code.

To run the coomand, I wrote the following:
loadjava –u DB_121RDB505/DB_121RDB505@85.254.218.228:1521:DITF11 –v –r- t Name_Surname.java

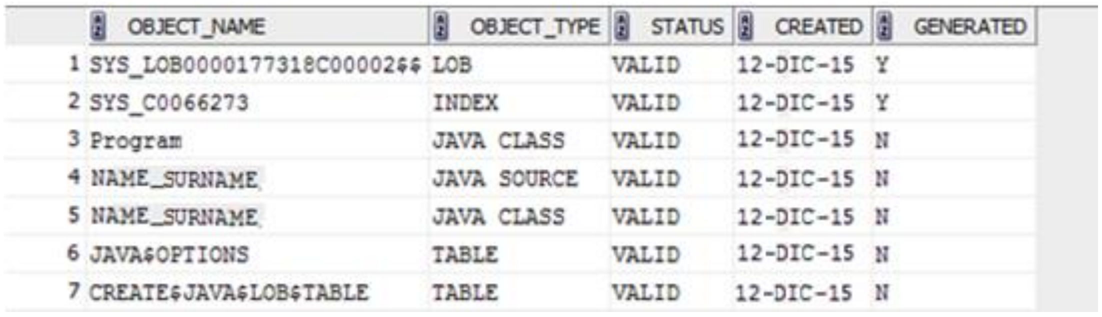
Which indicates to the loadjava command some important user parameters as DB_151ADB072 which is both user name and password (as issued to all RTU students in this course), 85254218228 is the server ip address and port is 1521 DITF11 is SID. NAME_SURNAME.java is the uploaded java file. The resulting extract:



```
C:\>loadjava -u DB_151ADB072/DB_151ADB072@85.254.218.228:1521:DITF11 -v -r -t Name_Su
rname.java
arguments: ´-u´ ´DB_151ADB072/***@85.254.218.228:1521:DITF11´-v´ ´-r´ ´-t´ ´Name_Surnam
e.java´
creating  : source Name_Surname
loading   : source Name_Surname
created   : CREATE$JAVA$LOB$TABLE
resolving : source Name_Surname
Classes Loaded: 0
Resources Loaded: 0
Sources Loaded: 1
Published Interfaces: 0
Classes generated: 0
Classes skipped: 0
```

Now, return to something common for us: PL/SQL. What we are reaching is the metadata obtained by the LOADJAVA command.

select OBJECT_NAME, OBJECT_TYPE, STATUS, CREATED, GENERATED
from USER_OBJECTS
where CREATED >= TO_DATE('12-12-2015', 'DD-MM-YYYY');



	OBJECT_NAME	OBJECT_TYPE	STATUS	CREATED	GENERATED
1	SYS_LOB0000177318C00002$$	LOB	VALID	12-DIC-15	Y
2	SYS_C0066273	INDEX	VALID	12-DIC-15	Y
3	Program	JAVA CLASS	VALID	12-DIC-15	N
4	NAME_SURNAME	JAVA SOURCE	VALID	12-DIC-15	N
5	NAME_SURNAME	JAVA CLASS	VALID	12-DIC-15	N
6	JAVA$OPTIONS	TABLE	VALID	12-DIC-15	N
7	CREATE$JAVA$LOB$TABLE	TABLE	VALID	12-DIC-15	N

As we see, a total of seven objects were created in the database after this first java program input. This is because the internal loadjava commands also compile java program and translate to SQL language. JAVA CLASS objects describe the different object classes, which were defined in the program. LOB table describes or are formed so-called "large objects". LOB and INDEX-type objects, as shown, generate an independent database from Java program object, indexing and recording.

Now we're going to delete the program Name_Surname with its existing database objects and insert/create the new java program Name_Surname2. The first exercise was not intended to return any output, and therefore we had to slightly modify the java source code.

From the beginning, define object class (as object types of a database)

```
class Program2 {
private int Id_Number1;
private String Name1;
private String Class;
public Program2(int i, String a, String b) {
        Id_Number1 = i;
        Name1 = a;
        Class = b;
}
```

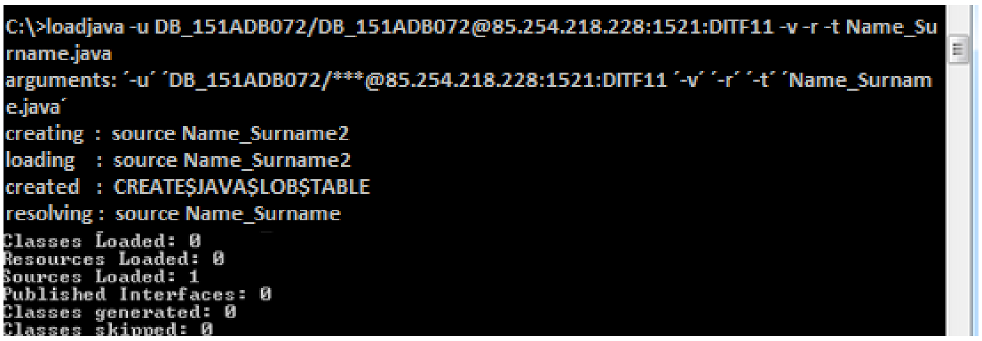
Output method described in the given object class

```
public int getID1() {return Id_Number1;}
public String getNos1() {return Name1;}
public String getClass1() {return Class;} }

public class Name_Surname2{
public static String Output1() {
String Output1 = " ";

Program2 Programs1[]= new Program2[3];
Programs1[0]= new Program2(1,"Miguel","Databases");
Programs1[1]= new Program2(2,"Mariya","Graphics");
Program1[2]= new Program2(3,"Sam","Maths");
```

Time to make some output

```
for (Program2 p : Programs1) Output1=(Output1 + p.getID1()+". " + p.getNos1()
+ " " + p.getClass1());
return Output1;}
public static void main(String[] args){
String Salida1 = Output1();
System.out.println(Salida1);}}
```

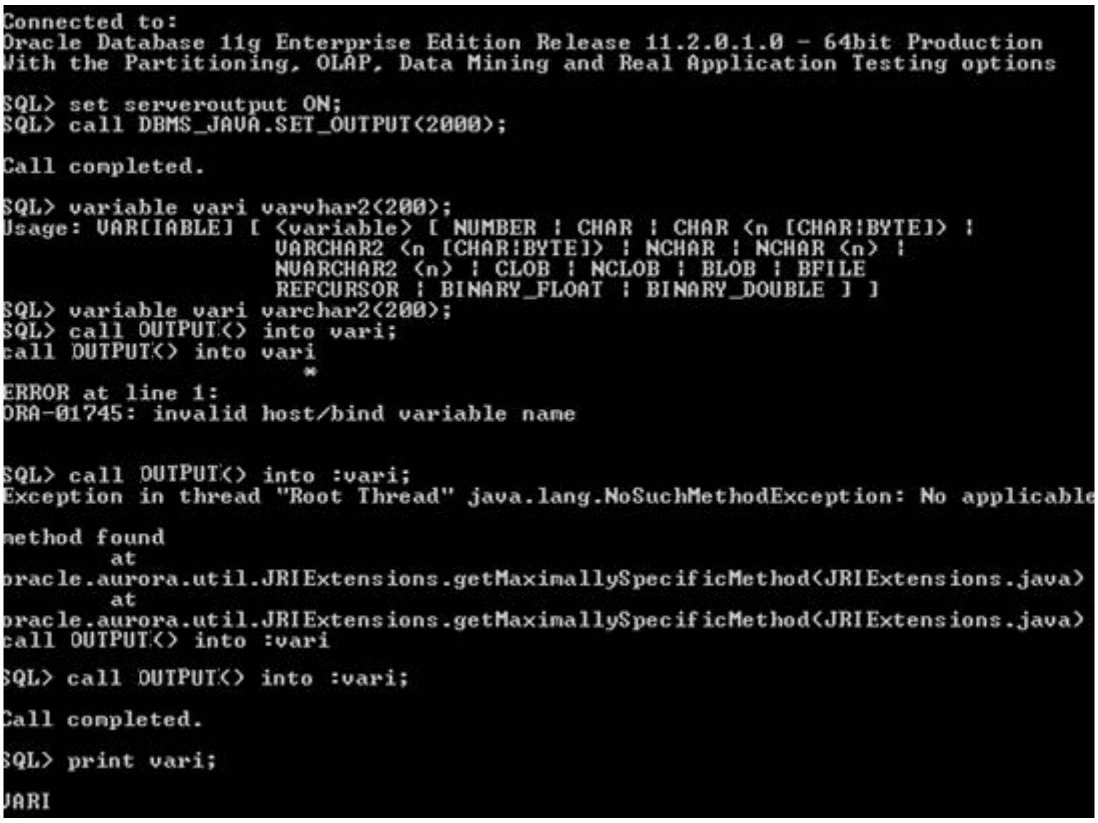
Again, I use the loadjava command to retrieve the java program. Then the function defined them, using the inserted in the program of the function Output of the same character in a series of three removes all created object data.



```
C:\>loadjava -u DB_151ADB072/DB_151ADB072@85.254.218.228:1521:DITF11 -v -r -t Name_Su
rname.java
arguments: ´-u´ ´DB_151ADB072/***@85.254.218.228:1521:DITF11 ´-v´ ´-r´ ´-t´ ´Name_Surnam
e.java´
creating : source Name_Surname2
loading  : source Name_Surname2
created  : CREATE$JAVA$LOB$TABLE
resolving : source Name_Surname
Classes Loaded: 0
Resources Loaded: 0
Sources Loaded: 1
Published Interfaces: 0
Classes generated: 0
Classes skipped: 0
```

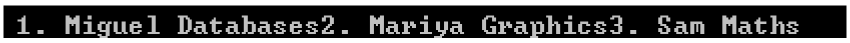
create or replace function Output return varchar2 as
language java name 'Name_Surname2.Output() return java.lang.String';

Next, I will use sql plus program to get the data.
First we connect to the server



```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> set serveroutput ON;
SQL> call DBMS_JAVA.SET_OUTPUT(2000);

Call completed.

SQL> variable vari varvhar2(200);
Usage: VAR[IABLE] [ <variable> [ NUMBER | CHAR | CHAR (n [CHAR|BYTE]) |
                    VARCHAR2 (n [CHAR|BYTE]) | NCHAR | NCHAR (n) |
                    NVARCHAR2 (n) | CLOB | NCLOB | BLOB | BFILE
                    REFCURSOR | BINARY_FLOAT | BINARY_DOUBLE ] ]
SQL> variable vari varchar2(200);
SQL> call OUTPUT() into vari;
call OUTPUT() into vari
                      *
ERROR at line 1:
ORA-01745: invalid host/bind variable name


SQL> call OUTPUT() into :vari;
Exception in thread "Root Thread" java.lang.NoSuchMethodException: No applicable

method found
        at
oracle.aurora.util.JRIExtensions.getMaximallySpecificMethod(JRIExtensions.java)
        at
oracle.aurora.util.JRIExtensions.getMaximallySpecificMethod(JRIExtensions.java)
call OUTPUT() into :vari

SQL> call OUTPUT() into :vari;

Call completed.

SQL> print vari;

VARI
```

And then we receive the data.



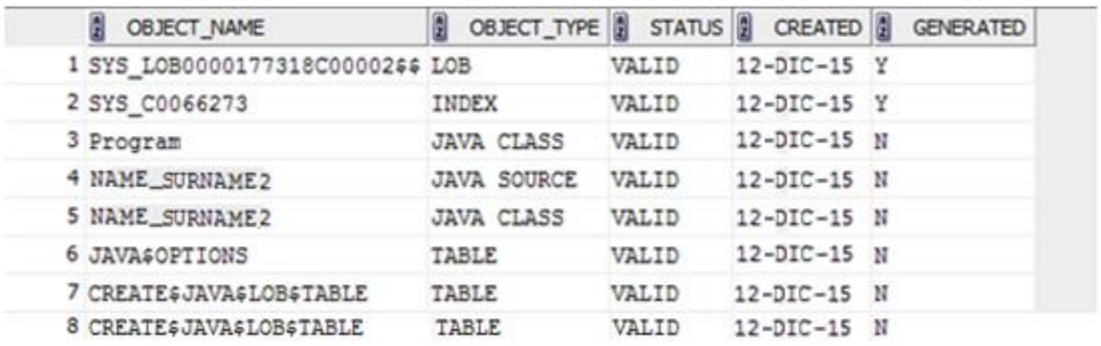
```
 1. Miguel Databases2. Mariya Graphics3. Sam Maths
```

What I wrote in SQL Plus indicated a user name / password @ IP database / SID

Set serveroutput ON; allows to retrieve data from the database.
Call DBMS_JAVA.SET_OUTPUT (2000); gives the opportunity to work with Java applications.
Variable can VARCHAR2 (200); refers to changing the user's computer memory, which creates a connection. This variable with the command call Output () into: vari;

Metadata

```
select  OBJECT_NAME, OBJECT_TYPE, STATUS, CREATED, GENERATED
from  USER_OBJECTS
where CREATED >= TO_DATE('12-12-2015', 'DD-MM-YYYY');
```

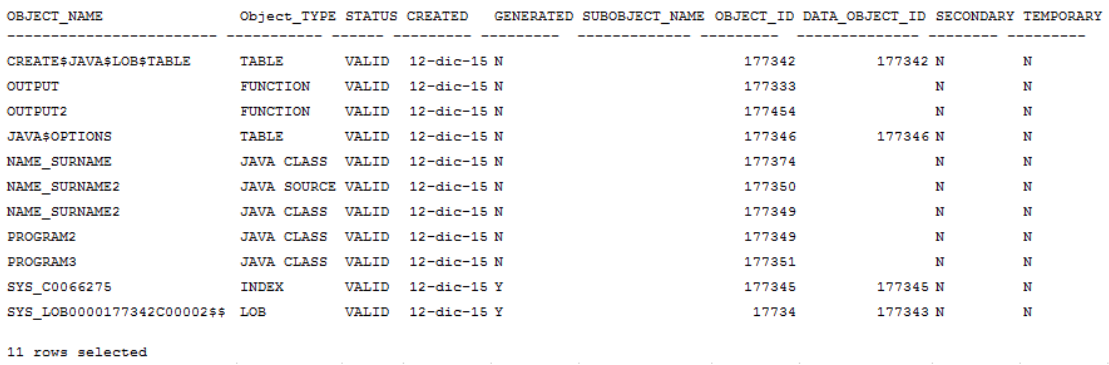


	OBJECT_NAME	OBJECT_TYPE	STATUS	CREATED	GENERATED
1	SYS_LOB0000177318C00002$$	LOB	VALID	12-DIC-15	Y
2	SYS_C0066273	INDEX	VALID	12-DIC-15	Y
3	Program	JAVA CLASS	VALID	12-DIC-15	N
4	NAME_SURNAME2	JAVA SOURCE	VALID	12-DIC-15	N
5	NAME_SURNAME2	JAVA CLASS	VALID	12-DIC-15	N
6	JAVA$OPTIONS	TABLE	VALID	12-DIC-15	N
7	CREATE$JAVA$LOB$TABLE	TABLE	VALID	12-DIC-15	N
8	CREATE$JAVA$LOB$TABLE	TABLE	VALID	12-DIC-15	N

Then, use the command CREATE JAVA to define a java program code inside SQL * Plus and immediately load it in the database.

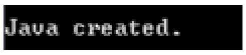
CREATE JAVA:

```
CREATE or replace JAVA source named "Name_Surname" AS
public class Name_Surname{
public static String Output1() {
String Output1 = " ";
Program3 Programs1[]= new Program3[2];
Programs1[0]= new Program3(4,"Carlos","Programming Languages");
Programs1[1]= new Program3(5,"Dinara","Software Evolution Technologies");
for (Program3 p : Programs1) Output1=(Output1 + p.getID1()+". " + p.getNos1()
+ " " + p.getClass1());
return Output1;}
public static void main(String[] args){
String Salida1 = Output1();
System.out.println(Salida1);}}

class Program3 {
private int ID_Number1;
private String Name1;
```



```
OBJECT_NAME                Object_TYPE STATUS CREATED   GENERATED SUBOBJECT_NAME OBJECT_ID DATA_OBJECT_ID SECONDARY TEMPORARY
------------------------ ----------- ------ --------- ---------  ------------- ---------  -------------- -------- ---------
CREATE$JAVA$LOB$TABLE      TABLE       VALID  12-dic-15 N                          177342         177342 N         N
OUTPUT                     FUNCTION    VALID  12-dic-15 N                          177333                N         N
OUTPUT2                    FUNCTION    VALID  12-dic-15 N                          177454                N         N
JAVA$OPTIONS               TABLE       VALID  12-dic-15 N                          177346         177346 N         N
NAME_SURNAME               JAVA CLASS  VALID  12-dic-15 N                          177374                N         N
NAME_SURNAME2              JAVA SOURCE VALID  12-dic-15 N                          177350                N         N
NAME_SURNAME2              JAVA CLASS  VALID  12-dic-15 N                          177349                N         N
PROGRAM2                   JAVA CLASS  VALID  12-dic-15 N                          177349                N         N
PROGRAM3                   JAVA CLASS  VALID  12-dic-15 N                          177351                N         N
SYS_C0066275               INDEX       VALID  12-dic-15 Y                          177345         177345 N         N
SYS_LOB0000177342C00002$$  LOB         VALID  12-dic-15 Y                           17734         177343 N         N

11 rows selected
```

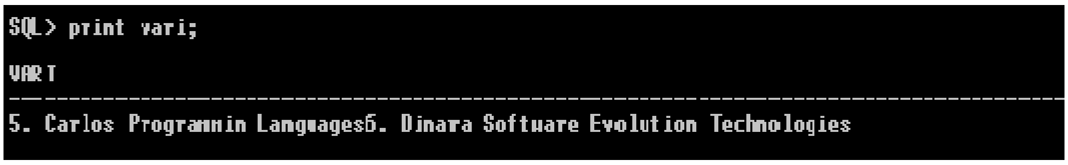
private String Class;
public Programma3(int i, String a, String b) {
    Id_Number1 = i;
    Name1 = a;
    Class = b;
}
public int getID1() {return Id_Number1;}
public String getNos1() {return Name1;}
public String getKlas1() {return Class;} }
/



```
Java created.
```

Creation of the program to be outputed, like the previous program.
create or replace function Izvade2 return varchar2 as
language java name ‘Name_Surnname.Output() return java.lang.String';

This is what vari Ouputed:



```
SQL> print vari;

VARI
--------------------------------------------------------------------------------
5. Carlos Programmin Languages6. Dinara Software Evolution Technologies
```

Select OBJECT_NAME, OBJECT_TYPE, STATUS, CREATED, GENERATED, SUBOBJECT_NAME, OBJECT_ID, DATA_OBJECT_ID, SECONDARY, TEMPORARY
from USER_OBJECTS
where CREATED >= TO_DATE('12-12-2015', 'DD-MM-YYYY');

A large part of these objects descriptive metadata attributes explained in previous works. OBJECT_NAME is the name of the object, OBJECT_TYPE is the type of, STATUS indicates whether an object is compiled or not, which is especially important

using internal database or SQL Plus command CREATE JAVA, which led directly loaded object, but the compilation is not performed.
If the inserted java class program is a mistake, it will know when the program will be used in a function, and if there will be an error, it will be transformed into a program, you want to retrieve it again and so on. Hence, I think it wiser to use the JAVAC command to compile java program is ready and load it with the Oracle loadjava program. Then at least it is safe, that the program works (of course, can still be a logic error).
GENERATED indicating whether the database itself is automatically generated by these objects.
SUBOBJECT_NAME points to the sub-bodies of the name (if any).
OBJECT_ID is this object a unique identifier attributed to the object database.
DATA_OBJECT_ID is this object identifier of the data entered. As can be seen, the same java objects are not defined for the data they occur in these classes is executed programs.
SECONDARY describes whether a file is created with ODCIINDEXCREATE method. In my case there is none.
TEMPORARY indicates whether the object will not delete the session (connection) at the end. Also of my work I am not.
Another metadata attribute, which did not allow me to be removed along with the rest (do not know why) is the OWNER, which describes which these objects belong. Of course, these objects belong to my user.

3. Java Class meta-data inspection (Data input)

This chapter will describe specific Java class methods of application procedures, as this, in my opinion, is one of the most useful applications of the JDBC interface, as it allows to simplify the definition of a query that queries the database by many to be repeated, it is very useful.
Previous practical work set up very much like table command to demonstrate the different methods of operation. In all these tables were placed in the program finished and unfinished program object collections.
Creation of a feature that makes the operation much easier, as well as, allows you to add these items outside of the database (for example, use SQL Plus)

Creation of the java source code:

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

       public class cols_rows {
       public static void main (String[] args)
       throws SQLException  {
              Connection conn =
DriverManager.getConnection("jdbc:default:connection:");
       String sql = "INSERT INTO TABLE(select A." + args[1] + " FROM " +
args[0] + " A where " + args[2] + ") values (" + args[3] + ")";
        try    {
        Statement vaic = conn.createStatement();
        query.executeUpdate(sql);
        System.out.println(sql);
        query.close();}
        catch (SQLException e) { System.err.println(e.getMessage()); } }}
```

create a package and define the procedures that will be held in the JAVA class execution. You need to enter the table name, collection name, provided after the table which selects the correct collection and the object itself, which will be placed in the collection.

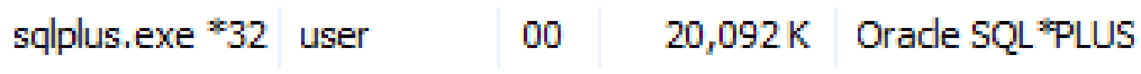
CREATE OR REPLACE PACKAGE Cols AS
PROCEDURE rows (Table_name VARCHAR2, Column_name VARCHAR2, Name VARCHAR2, Object VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY Cols AS
PROCEDURE rows (Table_name VARCHAR2, Column_name VARCHAR2, Name VARCHAR2, Object VARCHAR2) AS LANGUAGE JAVA
NAME ‘cols_rows.main(java.lang.String[])';
END;

call cols_rows('Test_Table','Column_name’,Name=''Miguel''','PROGRAM(''12'',''12'',''15''’)

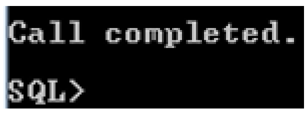
select * from Test_Table;

Since java class and the procedure is similar in structure as well as the procedure fulfilled in SQL Developer environment, I do not know what the problem is, but SQL Plus process resources consumed and does not refer to any activity.



```
sqlplus.exe *32   user   00   20,092 K   Oracle SQL*PLUS
```

Problem solution: Close SQL Developer.



```
Call completed.
SQL>
```

4. Java class methods of execution SQL command (data output)

As in the previous exercise, the table will work with the existing collections. This time, the procedure shall build a Java class method. I Will retrieve not only a collection but also two. I use the PreparedStatement object query definition. First, the source texts:

```
import java.sql.*;

import oracle.jdbc.*;
import oracle.jdbc.pool.OracleDataSource;

        public class cols_rows {
        public static void main (String[] args)
        throws SQLException  {
                Integer n =0;
                Connection conn =
DriverManager.getConnection("jdbc:default:connection:");
        String sql = "Select value(B).Name FROM TABLE(select A."+ args[1] +"
FROM " + args[0] + " A where " + args[2]+"=?) B";
         try    {
         PreparedStatement pstmt = conn.prepareStatement(sql);
         pstmt.setString(1,args[3]);
         ResultSet Record = pstmt.executeQuery();
         while(Record.next()){
                        String obj = Record.getString("value(b).Name");
                        n = n+1;
                        System.out.println(n + ". "+args[1]+" Name "+args[3]+"
has " + obj+".");
                }
         if (args.length>4){
                        String sql2 = "Select value(B).Name FROM TABLE(select
A."+ args[4] +" FROM " + args[0] + " A where " + args[2]+"=?) B";
                 PreparedStatement pstmt2 = conn.prepareStatement(sql2);
                 pstmt2.setString(1,args[3]);
                 ResultSet Record2 = pstmt2.executeQuery();
                 n=0;
                 while(Record2.next()){
                                String obj =
Record2.getString("value(b).Nosaukums");
                                n = n+1;
                                System.out.println(n + ". "+args[4]+" Name
"+args[3]+" has " + obj+".");
                        }
                 pstmt2.close();
         }
         System.out.println(sql);
         pstmt.close();}
         catch (SQLException e) { System.err.println(e.getMessage()); } }}
```

As shown in the code, there is a difference between the actions of the program if the function defines four or five attributes. Four defined attributes indicated in the table in the name of the collection name, the table row identifying the attribute name and the attribute value. The fifth attribute is another collection. If it is specified, it is executed in this collection output (collection must be at the same table).
I use PreparedStatement which gives the possibility of identifying attribute value with a question mark and define what value is entered into the query execution time. The problem is that it was necessary to use args [] changing the query text at every time, as the question mark can not contain fields and values of the table, just ordinary variables.
The following procedures describe the use of this program for the class, where I chose to simply complement the existing package Cols:

```
create or replace PACKAGE Cols AS
PROCEDURE rows (Table_name VARCHAR2, Cols_name VARCHAR2, Name
VARCHAR2, Object VARCHAR2);
PROCEDURE Record (Table_name VARCHAR2, Cols_name VARCHAR2,
Name VARCHAR2, Name_Collection VARCHAR2);
END;

create or replace PACKAGE BODY Cols
AS PROCEDURE rows (Table_name VARCHAR2, Cols_name VARCHAR2,
ANme VARCHAR2, Object VARCHAR2) AS LANGUAGE JAVA NAME
'cols_rows.main(java.lang.String[])';
PROCEDURE Record (Table_name VARCHAR2, Cols_name VARCHAR2,
Name VARCHAR2, Name_Collection VARCHAR2) AS LANGUAGE JAVA
NAME 'kol_izv.main(java.lang.String[])';
END;
```

6. Defining methods with the Java class and its application

Create a new object types VOLUME, which will indicate the height, length and width values as well MEMBER VOL_APR method that will calculate the volume.

```
create or replace type VOLUME as object
(HEIGHT number,
LENGTH number,
BASE number,
Member function VOL_APR return number);
```

define the volume calculation function.

```
create or replace type body VOLUME as
MEMBER function VOL_APR return number is
begin
return (java_tilp_apr(SELF.HEIGTH, SELF.LENGTH,SELF.BASE));
end VOL_APR;
end;
```

```
create or replace FUNCTION java_vol_apr(HEIGHT IN NUMBER, LENGTH
IN NUMBER, BASE IN NUMBER)
  RETURN NUMBER
  AS LANGUAGE JAVA
  NAME     'vol_met.value(java.lang.Integer,java.lang.Integer,java.lang.Integer)
return
java.lang.int';
```

Here the java class is calling a procedure similar to previous descriptions of the codet, but, unlike them, must also have its value type that will return by the method is performed.

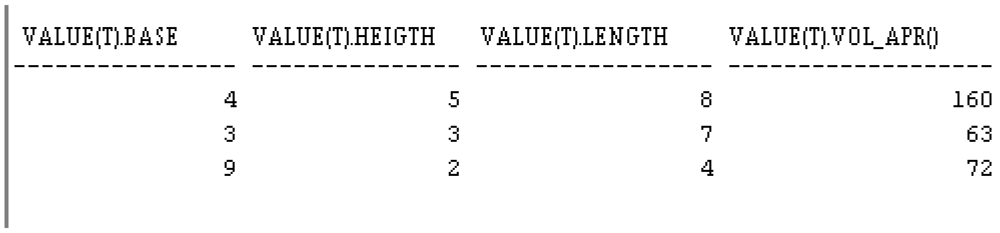
```
import java.sql.*;
import oracle.jdbc.*;
import java.util.*;
import java.io.*;
	public class vol_met{
	public static int value(Integer arg1,Integer arg2,Integer arg3)
throws SQLException{
		return (arg1*arg2*arg3);}}
```

This successful small example shows how member methods can be linked to the java class methods. Next we will make data entry and retrieval:

```
INSERT INTO VOLUME VALUES('8','5','4')
```

INSERT INTO VOLUME VALUES('7','3','3')
INSERT INTO VOLUME VALUES('4','2','9')

select value(T).Base, value(T).Length, value(T).Heigth,value(T).VOL_APR() FROM VOLUME T;



```
VALUE(T).BASE     VALUE(T).HEIGTH   VALUE(T).LENGTH     VALUE(T).VOL_APR()
----------------- ---------------- ------------------ --------------------
                4                5                  8                  160
                3                3                  7                   63
                9                2                  4                   72
```

8*4*5=160, 9*7=63 un 9*8 =72,

This simple example show us that is possible to create a lot of complex equations for complex objects and perform various operations with JAVA object attributes and methods, just as it can be done with conventional methods MEMBER. It has already become a coding logic problems, but I think exercise is to demonstrate the use of JAVA methods basic principle.