Exercises for Chapter 23: Object Types in Oracle

The Labs below provide you with exercises and suggested answers with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

Lab 23.1 Object Types

Answer the following questions.

Creating Object Types

In this exercise, you will be creating and manipulating object types.

Answer the following questions:

a) Create object type ENROLLMENT_OBJ_TYPE that has the following attributes:

```
ATTRIBUTE NAME   DATA TYPE      PRECISION
--------------   ---------      ---------
student_id       NUMBER                 8
first_name       VARCHAR2              25
last_name        VARCHAR2              25
course_no        NUMBER                 8
section_no       NUMBER                 3
enroll_date      DATE
final_grade      NUMBER                 3
```

Answer: The creation script should look similar to the following:

For Example *ch23_11a.sql*

```
CREATE OR REPLACE TYPE ENROLLMENT_OBJ_TYPE AS OBJECT
   (student_id   NUMBER(8)
   ,first_name   VARCHAR2(25)
   ,last_name    VARCHAR2(25)
   ,course_no    NUMBER(8)
   ,section_no   NUMBER(3)
   ,enroll_date  DATE
   ,final_grade  NUMBER(3));
```

b) Script below uses newly created object type. Execute it and explain the output produced.

For Example *ch23_12a.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;

BEGIN
   v_enrollment_obj.student_id := 102;
   v_enrollment_obj.first_name := 'Fred';
   v_enrollment_obj.last_name  := 'Crocitto';
   v_enrollment_obj.course_no  := 25;
END;
```

Answer: The output of the script should look similar to the following:

```
ORA-06530: Reference to uninitialized composite
ORA-06512: at line 5
```

This version of the script causes ORA-06530 error because it references individual attributes of the uninitialized object type instance. Before object attribute can be referenced the object must be initialized with the help of the constructor method.

c) Modify the script created in the previous exercise (ch23_12a.sql) so that it does not produce ORA-06530 error.

Answer: The script should be modified as follows. Affected statements are highlighted in bold.

For Example *ch23_12b.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;

BEGIN
   v_enrollment_obj :=
      enrollment_obj_type(102, 'Fred', 'Crocitto', 25, null, null, null);
END;
```

d) Modify the script created in the previous exercise (ch23_12b.sql) so that all object attributes are populated with corresponding values selected from the appropriate tables.

Answer: The modified script should look similar to one of the following scripts. All changes are shown in bold.

The first version of the script employs the `SELECT INTO` statement along with the constructor to initialize other attributes as well. Note that the `SELECT INTO` statement specifies where criteria for the `SECTION_NO` in addition to the criteria for the `STUDENT_ID` and `COURSE_NO`. This ensures that the `SELECT INTO` statement does not cause 'ORA-01422: exact fetch returns more than requested number of rows' error.

For Example *ch23_12c.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;

BEGIN
```

```
   SELECT
      enrollment_obj_type(st.student_id, st.first_name, st.last_name, c.course_no
                         ,se.section_no, e.enroll_date, e.final_grade)
     INTO v_enrollment_obj
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id = 102
      AND c.course_no   = 25
      AND se.section_no = 2;
END;
```

The second version of the script uses cursor FOR LOOP. This approach eliminates the need for additional criteria against the SECTION_NO.

For Example *ch23_12d.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;

BEGIN
   FOR REC IN (SELECT st.student_id, st.first_name, st.last_name, c.course_no
                     ,se.section_no, e.enroll_date, e.final_grade
                 FROM student st, course c, section se, enrollment e
                WHERE st.student_id = e.student_id
                  AND c.course_no   = se.course_no
                  AND se.section_id = e.section_id
                  AND st.student_id = 102
                  AND c.course_no   = 25)
   LOOP
      v_enrollment_obj :=
         enrollment_obj_type(rec.student_id, rec.first_name, rec.last_name
                            ,rec.course_no, rec.section_no, rec.enroll_date
                            ,rec.final_grade);
   END LOOP;
END;
```

e) Modify the script created in the previous exercise (use either versions of the script, ch23_2c.sql or ch23_2d.sql) so that enrollment object attributes are displayed.

Answer: The modified script should look similar to the following. Newly added statements are shown in bold.

For Example *ch23_12e.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;

BEGIN
   FOR REC IN (SELECT st.student_id, st.first_name, st.last_name, c.course_no
                     ,se.section_no, e.enroll_date, e.final_grade
                 FROM student st, course c, section se, enrollment e
```

```
            WHERE st.student_id = e.student_id
              AND c.course_no   = se.course_no
              AND se.section_id = e.section_id
              AND st.student_id = 102
              AND c.course_no   = 25)
   LOOP
      v_enrollment_obj :=
         enrollment_obj_type(rec.student_id, rec.first_name, rec.last_name
                            ,rec.course_no, rec.section_no, rec.enroll_date
                            ,rec.final_grade);

      DBMS_OUTPUT.PUT_LINE ('student_id:  '||v_enrollment_obj.student_id);
      DBMS_OUTPUT.PUT_LINE ('first_name:  '||v_enrollment_obj.first_name);
      DBMS_OUTPUT.PUT_LINE ('last_name:   '||v_enrollment_obj.last_name);
      DBMS_OUTPUT.PUT_LINE ('course_no:   '||v_enrollment_obj.course_no);
      DBMS_OUTPUT.PUT_LINE ('section_no:  '||v_enrollment_obj.section_no);
      DBMS_OUTPUT.PUT_LINE ('enroll_date: '||v_enrollment_obj.enroll_date);
      DBMS_OUTPUT.PUT_LINE ('final_grade: '||v_enrollment_obj.final_grade);
   END LOOP;
END;
```

This version of the script produces output as shown:

```
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  5
enroll_date: 01/30/2007 10:18
final_grade: 92
```

Using Object Types with Collections

In this exercise, you will continue exploring object types and how these may be used with collections. Answer the following questions:

a) Modify script ch23_12e.sql created in the previous exercise. In the new script, populate associative array of objects. Use multiple student IDs for this exercise, i.e., student IDs 102, 103, and 104.

Answer: The script should look similar to the script below.

For Example *ch23_13a.sql*

```
DECLARE
   TYPE enroll_tab_type IS TABLE OF enrollment_obj_type INDEX BY PLS_INTEGER;

   v_enrollment_tab enroll_tab_type;
   v_counter integer := 0;

BEGIN
   FOR REC IN (SELECT st.student_id, st.first_name, st.last_name, c.course_no
                     ,se.section_no, e.enroll_date, e.final_grade
                 FROM student st, course c, section se, enrollment e
                WHERE st.student_id = e.student_id
                  AND c.course_no   = se.course_no
                  AND se.section_id = e.section_id
                  AND st.student_id in (102, 103, 104))
   LOOP
      v_counter := v_counter + 1;
      v_enrollment_tab(v_counter) :=
         enrollment_obj_type(rec.student_id, rec.first_name, rec.last_name
                            ,rec.course_no, rec.section_no, rec.enroll_date
                            ,rec.final_grade);

      DBMS_OUTPUT.PUT_LINE ('student_id:  '||
         v_enrollment_tab(v_counter).student_id);
      DBMS_OUTPUT.PUT_LINE ('first_name:  '||
         v_enrollment_tab(v_counter).first_name);
      DBMS_OUTPUT.PUT_LINE ('last_name:   '||
         v_enrollment_tab(v_counter).last_name);
      DBMS_OUTPUT.PUT_LINE ('course_no:   '||
         v_enrollment_tab(v_counter).course_no);
      DBMS_OUTPUT.PUT_LINE ('section_no:  '||
         v_enrollment_tab(v_counter).section_no);
      DBMS_OUTPUT.PUT_LINE ('enroll_date: '||
         v_enrollment_tab(v_counter).enroll_date);
      DBMS_OUTPUT.PUT_LINE ('final_grade: '||
         v_enrollment_tab(v_counter).final_grade);
      DBMS_OUTPUT.PUT_LINE ('------------------');
   END LOOP;
END;
```

The script above defines associative array of objects that is populated with the help of the cursor `FOR` loop. Once a single row of the associative array has been initialized, it is displayed on the screen.

Take a closer look at how each row of the associative array is initialized:

```
v_enrollment_tab(v_counter) :=
   enrollment_obj_type(rec.student_id, rec.first_name, rec.last_name, rec.course_no
                      ,rec.section_no, rec.enroll_date, rec.final_grade);
```

A row is referenced by a subscript which in this case is variable, `v_counter`. Since each row represents an object instance, it is initialized by referencing the default constructor method associated with the corresponding object type.

When run, the script produces output as shown:

```
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  2
```

```
enroll_date: 01/30/2007 10:18
final_grade:
------------------
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  5
enroll_date: 01/30/2007 10:18
final_grade: 92
------------------
student_id:  103
first_name:  J.
last_name:   Landry
course_no:   20
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
------------------
student_id:  104
first_name:  Laetia
last_name:   Enison
course_no:   20
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
------------------
```

b) Modify the script created above (ch23_13a.sql) so that table of objects is populated via the `BULK SELECT INTO` statement.

Answer: The new version of the script should look similar to the following. Changes are highlighted in bold.

For Example *ch23_13b.sql*

```
DECLARE
   TYPE enroll_tab_type IS TABLE OF enrollment_obj_type INDEX BY PLS_INTEGER;

   v_enrollment_tab enroll_tab_type;

BEGIN
   SELECT
      enrollment_obj_type(st.student_id, st.first_name, st.last_name, c.course_no
                         ,se.section_no, e.enroll_date, e.final_grade)
     BULK COLLECT INTO v_enrollment_tab
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id in (102, 103, 104);

   FOR i IN 1..v_enrollment_tab.COUNT
   LOOP
      DBMS_OUTPUT.PUT_LINE ('student_id:  '||v_enrollment_tab(i).student_id);
      DBMS_OUTPUT.PUT_LINE ('first_name:  '||v_enrollment_tab(i).first_name);
      DBMS_OUTPUT.PUT_LINE ('last_name:   '||v_enrollment_tab(i).last_name);
      DBMS_OUTPUT.PUT_LINE ('course_no:   '||v_enrollment_tab(i).course_no);
      DBMS_OUTPUT.PUT_LINE ('section_no:  '||v_enrollment_tab(i).section_no);
      DBMS_OUTPUT.PUT_LINE ('enroll_date: '||v_enrollment_tab(i).enroll_date);
      DBMS_OUTPUT.PUT_LINE ('final_grade: '||v_enrollment_tab(i).final_grade);
      DBMS_OUTPUT.PUT_LINE ('------------------');
   END LOOP;
END;
```

In the version of the script, the cursor `FOR LOOP` has been replaced by the `BULK SELECT INTO` statement. As a result, the cursor `FOR LOOP` is replaced by the numeric `FOR LOOP` to display data on the screen. These changes eliminated the need for the variable `v_counter` that was used to reference individual rows of the associative array.

When run, this version of the script produces output that is identical to the previous version.

c) Modify the script created above (ch23_13b.sql) so that data stored in the table of objects is retrieved via the `SELECT INTO` statement before it is displayed.

Answer: As mentioned in Chapter 23, in order to select data from a table of objects, the underlying table type must be either a nested table or a varray that is created and stored in the database schema. This is accomplished by the following statement:

```
CREATE OR REPLACE TYPE enroll_tab_type AS TABLE OF enrollment_obj_type;
```

Once nested table type is created, the script is modified as follows. Changes are shown in bold letters.

For Example *ch23_13c.sql*

```
DECLARE
   v_enrollment_tab enroll_tab_type;

BEGIN
   SELECT
      enrollment_obj_type(st.student_id, st.first_name, st.last_name, c.course_no
                         ,se.section_no, e.enroll_date, e.final_grade)
     BULK COLLECT INTO v_enrollment_tab
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id in (102, 103, 104);

   FOR rec IN (SELECT *
                 FROM TABLE(CAST(v_enrollment_tab AS enroll_tab_type)))
   LOOP
      DBMS_OUTPUT.PUT_LINE ('student_id:  '||rec.student_id);
      DBMS_OUTPUT.PUT_LINE ('first_name:  '||rec.first_name);
      DBMS_OUTPUT.PUT_LINE ('last_name:   '||rec.last_name);
      DBMS_OUTPUT.PUT_LINE ('course_no:   '||rec.course_no);
      DBMS_OUTPUT.PUT_LINE ('section_no:  '||rec.section_no);
      DBMS_OUTPUT.PUT_LINE ('enroll_date: '||rec.enroll_date);
      DBMS_OUTPUT.PUT_LINE ('final_grade: '||rec.final_grade);
      DBMS_OUTPUT.PUT_LINE ('------------------');
   END LOOP;
END;
```

Note that in this version of the script, the numeric `FOR LOOP` is replaced by the cursor `FOR LOOP` against the nested table of objects. Note that the `DBMS_OUTPUT.PUT_LINE` statements are also changed so that they reference records returned by the cursor.

Lab 23.2 Object Type Methods

In this exercise, you will create various methods for the `enrollment_obj_type` created in the previous Lab.

Watch Out!

Before proceeding with this exercise you need to drop nested table type created in the previous Lab as follows:

```
DROP TYPE enroll_tab_type;
```

Recall that `enrollment_obj_type` was created as follows:

```
CREATE OR REPLACE TYPE ENROLLMENT_OBJ_TYPE AS OBJECT
   (student_id  NUMBER(8)
   ,first_name  VARCHAR2(25)
   ,last_name   VARCHAR2(25)
   ,course_no   NUMBER(8)
   ,section_no  NUMBER(3)
   ,enroll_date DATE
   ,final_grade NUMBER(3));
```

Create the following methods for the `enrollment_obj_type`:

a) Create user-defined constructor method that populates object type attributes by selecting data from the corresponding tables based on the incoming values for student ID, course and section numbers.

Answer: The script should look similar to the following:

For Example *ch23_14a.sql*

```
CREATE OR REPLACE TYPE enrollment_obj_type AS OBJECT
   (student_id  NUMBER(8),
    first_name  VARCHAR2(25),
    last_name   VARCHAR2(25),
    course_no   NUMBER(8),
    section_no  NUMBER(3),
    enroll_date DATE,
    final_grade NUMBER(3),

   CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                            ,in_student_id NUMBER
                                            ,in_course_no  NUMBER
                                            ,in_section_no NUMBER)
   RETURN SELF AS RESULT);
/

CREATE OR REPLACE TYPE BODY enrollment_obj_type AS

CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                         ,in_student_id NUMBER
                                         ,in_course_no  NUMBER
                                         ,in_section_no NUMBER)
RETURN SELF AS RESULT
IS
BEGIN
   SELECT st.student_id, st.first_name, st.last_name, c.course_no,
          se.section_no, e.enroll_date, e.final_grade
     INTO SELF.student_id, SELF.first_name, SELF.last_name,
          SELF.course_no, SELF.section_no, SELF.enroll_date,
          SELF.final_grade
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id = in_student_id
      AND c.course_no   = in_course_no
      AND se.section_no = in_section_no;

   RETURN;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN;
END;
END;
/
```

Take a closer look at the `SELECT INTO` statement of the constructor method above. This statement is very similar to the `SELECT INTO` statement used in the previous Lab:

```
SELECT
   enrollment_obj_type(st.student_id, st.first_name, st.last_name, c.course_no
                      ,se.section_no, e.enroll_date, e.final_grade)
  INTO v_enrollment_obj
  FROM student st, course c, section se, enrollment e
 WHERE st.student_id = e.student_id
   AND c.course_no   = se.course_no
   AND se.section_id = e.section_id
   AND st.student_id = 102
   AND c.course_no   = 25
   AND se.section_no = 2;
```

Note that the `SELECT INTO` statement in the constructor body does not reference system-defined default constructor. Instead, it uses built-in `SELF` parameter to reference individual attributes of the current object instance.

Newly added constructor method may be tested as follows:

For Example *ch23_15a.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;
BEGIN
   v_enrollment_obj := enrollment_obj_type(102, 25, 2);

   DBMS_OUTPUT.PUT_LINE ('student_id:  '||v_enrollment_obj.student_id);
   DBMS_OUTPUT.PUT_LINE ('first_name:  '||v_enrollment_obj.first_name);
   DBMS_OUTPUT.PUT_LINE ('last_name:   '||v_enrollment_obj.last_name);
   DBMS_OUTPUT.PUT_LINE ('course_no:   '||v_enrollment_obj.course_no);
   DBMS_OUTPUT.PUT_LINE ('section_no:  '||v_enrollment_obj.section_no);
   DBMS_OUTPUT.PUT_LINE ('enroll_date: '||v_enrollment_obj.enroll_date);
   DBMS_OUTPUT.PUT_LINE ('final_grade: '||v_enrollment_obj.final_grade);
END;
```

The test script produces output as shown:

```
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
```

b) Add member procedure method `GET_ENROLLMENT_INFO` that returns attribute values.

Answer: The member procedure method should look similar to the following. Newly added method is shown in bold.

For Example *ch23_14b.sql*

```
CREATE OR REPLACE TYPE enrollment_obj_type AS OBJECT
   (student_id  NUMBER(8),
    first_name  VARCHAR2(25),
    last_name   VARCHAR2(25),
    course_no   NUMBER(8),
    section_no  NUMBER(3),
    enroll_date DATE,
    final_grade NUMBER(3),

   CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                            ,in_student_id NUMBER
                                            ,in_course_no  NUMBER
                                            ,in_section_no NUMBER)
   RETURN SELF AS RESULT,

   MEMBER PROCEDURE get_enrollment_info (out_student_id  OUT NUMBER
                                        ,out_first_name  OUT VARCHAR2
                                        ,out_last_name   OUT VARCHAR2
                                        ,out_course_no   OUT NUMBER
                                        ,out_section_no  OUT NUMBER
                                        ,out_enroll_date OUT DATE
                                        ,out_final_grade OUT NUMBER));
/

CREATE OR REPLACE TYPE BODY enrollment_obj_type AS

CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                         ,in_student_id NUMBER
                                         ,in_course_no  NUMBER
                                         ,in_section_no NUMBER)
RETURN SELF AS RESULT
IS
BEGIN
   SELECT st.student_id, st.first_name, st.last_name, c.course_no,
          se.section_no, e.enroll_date, e.final_grade
     INTO SELF.student_id, SELF.first_name, SELF.last_name,
          SELF.course_no, SELF.section_no, SELF.enroll_date,
          SELF.final_grade
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id = in_student_id
      AND c.course_no   = in_course_no
      AND se.section_no = in_section_no;

   RETURN;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN;
END;

MEMBER PROCEDURE get_enrollment_info (out_student_id  OUT NUMBER
                                     ,out_first_name  OUT VARCHAR2
                                     ,out_last_name   OUT VARCHAR2
                                     ,out_course_no   OUT NUMBER
                                     ,out_section_no  OUT NUMBER
                                     ,out_enroll_date OUT DATE
                                     ,out_final_grade OUT NUMBER)
IS
BEGIN
   out_student_id  := student_id;
   out_first_name  := first_name;
   out_last_name   := last_name;
   out_course_no   := course_no;
   out_section_no  := section_no;
   out_enroll_date := enroll_date;
   out_final_grade := final_grade;
END;

END;
/
```

c) Add static method to the `enrollment_obj_type` object type that displays values of individual attributes.

Answer: The script should look similar to the following script. Changes are shown in bold.

For Example *ch23_14c.sql*

```
CREATE OR REPLACE TYPE enrollment_obj_type AS OBJECT
   (student_id  NUMBER(8),
    first_name  VARCHAR2(25),
    last_name   VARCHAR2(25),
    course_no   NUMBER(8),
    section_no  NUMBER(3),
    enroll_date DATE,
    final_grade NUMBER(3),

   CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                            ,in_student_id NUMBER
                                            ,in_course_no  NUMBER
                                            ,in_section_no NUMBER)
   RETURN SELF AS RESULT,

   MEMBER PROCEDURE get_enrollment_info (out_student_id  OUT NUMBER
                                        ,out_first_name  OUT VARCHAR2
                                        ,out_last_name   OUT VARCHAR2
                                        ,out_course_no   OUT NUMBER
                                        ,out_section_no  OUT NUMBER
                                        ,out_enroll_date OUT DATE
                                        ,out_final_grade OUT NUMBER),

   STATIC PROCEDURE display_enrollment_info (enrollment_obj enrollment_obj_type));
/

CREATE OR REPLACE TYPE BODY enrollment_obj_type AS

CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                         ,in_student_id NUMBER
                                         ,in_course_no  NUMBER
                                         ,in_section_no NUMBER)
RETURN SELF AS RESULT
IS
BEGIN
   SELECT st.student_id, st.first_name, st.last_name, c.course_no,
          se.section_no, e.enroll_date, e.final_grade
     INTO SELF.student_id, SELF.first_name, SELF.last_name,
          SELF.course_no, SELF.section_no, SELF.enroll_date,
          SELF.final_grade
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id = in_student_id
      AND c.course_no   = in_course_no
      AND se.section_no = in_section_no;
   RETURN;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN;
END;

MEMBER PROCEDURE get_enrollment_info (out_student_id  OUT NUMBER
                                     ,out_first_name  OUT VARCHAR2
                                     ,out_last_name   OUT VARCHAR2
                                     ,out_course_no   OUT NUMBER
                                     ,out_section_no  OUT NUMBER
                                     ,out_enroll_date OUT DATE
                                     ,out_final_grade OUT NUMBER)
IS
BEGIN
   out_student_id  := student_id;
   out_first_name  := first_name;
   out_last_name   := last_name;
   out_course_no   := course_no;
   out_section_no  := section_no;
   out_enroll_date := enroll_date;
   out_final_grade := final_grade;
END;

STATIC PROCEDURE display_enrollment_info (enrollment_obj enrollment_obj_type)
IS
BEGIN
   DBMS_OUTPUT.PUT_LINE ('student_id:  '||enrollment_obj.student_id);
   DBMS_OUTPUT.PUT_LINE ('first_name:  '||enrollment_obj.first_name);
   DBMS_OUTPUT.PUT_LINE ('last_name:   '||enrollment_obj.last_name);
   DBMS_OUTPUT.PUT_LINE ('course_no:   '||enrollment_obj.course_no);
   DBMS_OUTPUT.PUT_LINE ('section_no:  '||enrollment_obj.section_no);
   DBMS_OUTPUT.PUT_LINE ('enroll_date: '||enrollment_obj.enroll_date);
   DBMS_OUTPUT.PUT_LINE ('final_grade: '||enrollment_obj.final_grade);
END;

END;
/
```

Recall that static methods are created for actions that do not need to access data associated with a particular object instance, and as such may not reference default parameter SELF. Then, in order to display attribute data associated with some object instance, the instance itself is passed in to the method.

The newly created method may be tested as follows:

For Example *ch23_15b.sql*

```
DECLARE
   v_enrollment_obj enrollment_obj_type;
BEGIN
   v_enrollment_obj := enrollment_obj_type(102, 25, 2);
   enrollment_obj_type.display_enrollment_info (v_enrollment_obj);
END;
```

Note the invocation call to the static method. *The call to the static method is qualified with object type name and not with object type instance name.*
The test script produces output as shown:

```
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
```

d) Add method to the object type `entollment_obj_type` so that its instances may be compared and/or sorted. The object instances should be compared based on the values of `course_no`, `section_no`, and `student_id` attributes.

Answer: Recall that in order to compare and sort object instances their corresponding type must have either map or order methods. For the purpose of this exercise, map method is added to the type definition as follows. Newly added method is shown in bold.

For Example *ch23_14d.sql*

```
CREATE OR REPLACE TYPE enrollment_obj_type AS OBJECT
   (student_id  NUMBER(8),
    first_name  VARCHAR2(25),
    last_name   VARCHAR2(25),
    course_no   NUMBER(8),
    section_no  NUMBER(3),
    enroll_date DATE,
    final_grade NUMBER(3),

   CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                            ,in_student_id NUMBER
                                            ,in_course_no  NUMBER
                                            ,in_section_no NUMBER)
   RETURN SELF AS RESULT,

   MEMBER PROCEDURE get_enrollment_info (out_student_id  OUT NUMBER
                                        ,out_first_name  OUT VARCHAR2
                                        ,out_last_name   OUT VARCHAR2
                                        ,out_course_no   OUT NUMBER
                                        ,out_section_no  OUT NUMBER
                                        ,out_enroll_date OUT DATE
                                        ,out_final_grade OUT NUMBER),

   STATIC PROCEDURE display_enrollment_info (enrollment_obj enrollment_obj_type),

   MAP MEMBER FUNCTION enrollment RETURN NUMBER);
/
```

```
CREATE OR REPLACE TYPE BODY enrollment_obj_type AS

CONSTRUCTOR FUNCTION enrollment_obj_type (SELF IN OUT NOCOPY enrollment_obj_type
                                         ,in_student_id NUMBER
                                         ,in_course_no  NUMBER
                                         ,in_section_no NUMBER)
RETURN SELF AS RESULT
IS
BEGIN
   SELECT st.student_id, st.first_name, st.last_name, c.course_no,
          se.section_no, e.enroll_date, e.final_grade
     INTO SELF.student_id, SELF.first_name, SELF.last_name,
          SELF.course_no, SELF.section_no, SELF.enroll_date,
          SELF.final_grade
     FROM student st, course c, section se, enrollment e
    WHERE st.student_id = e.student_id
      AND c.course_no   = se.course_no
      AND se.section_id = e.section_id
      AND st.student_id = in_student_id
      AND c.course_no   = in_course_no
      AND se.section_no = in_section_no;

   RETURN;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN;
END;

MEMBER PROCEDURE get_enrollment_info (out_student_id  OUT NUMBER
                                     ,out_first_name  OUT VARCHAR2
                                     ,out_last_name   OUT VARCHAR2
                                     ,out_course_no   OUT NUMBER
                                     ,out_section_no  OUT NUMBER
                                     ,out_enroll_date OUT DATE
                                     ,out_final_grade OUT NUMBER)
IS
BEGIN
   out_student_id  := student_id;
   out_first_name  := first_name;
   out_last_name   := last_name;
   out_course_no   := course_no;
   out_section_no  := section_no;
   out_enroll_date := enroll_date;
   out_final_grade := final_grade;
END;

STATIC PROCEDURE display_enrollment_info (enrollment_obj enrollment_obj_type)
IS
BEGIN
   DBMS_OUTPUT.PUT_LINE ('student_id:  '||enrollment_obj.student_id);
   DBMS_OUTPUT.PUT_LINE ('first_name:  '||enrollment_obj.first_name);
   DBMS_OUTPUT.PUT_LINE ('last_name:   '||enrollment_obj.last_name);
   DBMS_OUTPUT.PUT_LINE ('course_no:   '||enrollment_obj.course_no);
   DBMS_OUTPUT.PUT_LINE ('section_no:  '||enrollment_obj.section_no);
   DBMS_OUTPUT.PUT_LINE ('enroll_date: '||enrollment_obj.enroll_date);
   DBMS_OUTPUT.PUT_LINE ('final_grade: '||enrollment_obj.final_grade);
END;

MAP MEMBER FUNCTION enrollment RETURN NUMBER
IS
BEGIN
   RETURN (course_no + section_no + student_id);
END;

END;
/
```

The newly added function adds values stored in the `course_no`, `section_no`, and `student_id` attributes. The resulting value may now be used to compare different object instances as illustrated below:

For Example *ch23_15c.sql*

```
DECLARE
   v_enrollment_obj1 enrollment_obj_type;
   v_enrollment_obj2 enrollment_obj_type;
BEGIN
   v_enrollment_obj1 := enrollment_obj_type(102, 25, 2);
   v_enrollment_obj2 := enrollment_obj_type(104, 20, 2);

   enrollment_obj_type.display_enrollment_info (v_enrollment_obj1);
   DBMS_OUTPUT.PUT_LINE ('--------------------');
   enrollment_obj_type.display_enrollment_info (v_enrollment_obj2);

   IF v_enrollment_obj1 > v_enrollment_obj2
   THEN
      DBMS_OUTPUT.PUT_LINE ('Instance 1 is greater than instance 2');
   ELSE
      DBMS_OUTPUT.PUT_LINE ('Instance 1 is not greater than instance 2');
   END IF;
END;
```

When run, the test script produces the following output:

```
student_id:  102
first_name:  Fred
last_name:   Crocitto
course_no:   25
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
--------------------
student_id:  104
first_name:  Laetia
```

```
last_name:   Enison
course_no:   20
section_no:  2
enroll_date: 01/30/2007 10:18
final_grade:
Instance 1 is greater than instance 2
```

Try It Yourself

The projects in this section are meant to have you use all of the skills that you have acquired throughout this chapter. Here are some exercises that will help you test the depth of your understanding.

1) Create object type `student_obj_type` with attributes derived from the `STUDENT` table.

   Answer: The object type should look similar to the following:

   For Example *ch23_16a.sql*

```
CREATE OR REPLACE TYPE student_obj_type AS OBJECT
   (student_id        NUMBER(8),
    salutation        VARCHAR2(5),
    first_name        VARCHAR2(25),
    last_name         VARCHAR2(25),
    street_address    VARCHAR2(50),
    zip               VARCHAR2(5),
    phone             VARCHAR2(15),
    employer          VARCHAR2(50),
    registration_date DATE,
    created_by        VARCHAR2(30),
    created_date      DATE,
    modified_by       VARCHAR2(30),
    modified_date     DATE);
/
```

   Once this object type is created it can be used as follows:

   For Example *ch23_17a.sql*

```
DECLARE
   v_student_obj student_obj_type;
BEGIN
   -- Use default contructor method to initialize student object
   SELECT
      student_obj_type(student_id, salutation, first_name, last_name
                      ,street_address, zip, phone, employer, registration_date
                      ,null, null, null, null)
     INTO v_student_obj
     FROM student
    WHERE student_id = 103;

   DBMS_OUTPUT.PUT_LINE ('Student ID: '       ||v_student_obj.student_id);
   DBMS_OUTPUT.PUT_LINE ('Salutation: '       ||v_student_obj.salutation);
   DBMS_OUTPUT.PUT_LINE ('First Name: '       ||v_student_obj.first_name);
   DBMS_OUTPUT.PUT_LINE ('Last Name: '        ||v_student_obj.last_name);
   DBMS_OUTPUT.PUT_LINE ('Street Address: '   ||v_student_obj.street_address);
   DBMS_OUTPUT.PUT_LINE ('Zip: '              ||v_student_obj. zip);
   DBMS_OUTPUT.PUT_LINE ('Phone: '            ||v_student_obj.phone);
   DBMS_OUTPUT.PUT_LINE ('Employer: '         ||v_student_obj.employer);
   DBMS_OUTPUT.PUT_LINE ('Registration Date: '||v_student_obj.registration_date);
END;
```

When run, the test script produces the following output:

```
Student ID: 103
Salutation: Ms.
First Name: J.
Last Name: Landry
Street Address: 7435 Boulevard East #45
Zip: 07047
Phone: 201-555-5555
Employer: Albert Hildegard Co.
Registration Date: 01/22/2007 00:00
```

2) Add user-defined constructor function, member procedure, static procedure, and order function methods. You should determine on your own how these methods should be structured.

Answer: Newly modified student object should be similar to the following:

For Example *ch23_16b.sql*

```
CREATE OR REPLACE TYPE student_obj_type AS OBJECT
  (student_id        NUMBER(8),
   salutation        VARCHAR2(5),
   first_name        VARCHAR2(25),
   last_name         VARCHAR2(25),
   street_address    VARCHAR2(50),
   zip               VARCHAR2(5),
   phone             VARCHAR2(15),
   employer          VARCHAR2(50),
   registration_date DATE,
   created_by        VARCHAR2(30),
   created_date      DATE,
   modified_by       VARCHAR2(30),
   modified_date     DATE,

   CONSTRUCTOR FUNCTION student_obj_type
      (SELF IN OUT NOCOPY STUDENT_OBJ_TYPE
      ,in_student_id  IN NUMBER,   in_salutation IN VARCHAR2
      ,in_first_name  IN VARCHAR2, in_last_name  IN VARCHAR2
      ,in_street_addr IN VARCHAR2, in_zip        IN VARCHAR2
      ,in_phone       IN VARCHAR2, in_employer   IN VARCHAR2
      ,in_reg_date    IN DATE,     in_cr_by      IN VARCHAR2
      ,in_cr_date     IN DATE,     in_mod_by     IN VARCHAR2
      ,in_mod_date    IN DATE)
   RETURN SELF AS RESULT,
   CONSTRUCTOR FUNCTION student_obj_type (SELF IN OUT NOCOPY STUDENT_OBJ_TYPE
                                         ,in_student_id IN NUMBER)
   RETURN SELF AS RESULT,

   MEMBER PROCEDURE get_student_info
      (student_id  OUT NUMBER,   salutation OUT VARCHAR2
      ,first_name  OUT VARCHAR2, last_name  OUT VARCHAR2
      ,street_addr OUT VARCHAR2, zip        OUT VARCHAR2
      ,phone       OUT VARCHAR2, employer   OUT VARCHAR2
      ,reg_date    OUT DATE,     cr_by      OUT VARCHAR2
      ,cr_date     OUT DATE,     mod_by     OUT VARCHAR2
      ,mod_date    OUT DATE),

   STATIC PROCEDURE display_student_info (student_obj IN STUDENT_OBJ_TYPE),

   ORDER MEMBER FUNCTION student (student_obj STUDENT_OBJ_TYPE)
   RETURN INTEGER);
/

CREATE OR REPLACE TYPE BODY student_obj_type AS

CONSTRUCTOR FUNCTION student_obj_type
   (SELF IN OUT NOCOPY STUDENT_OBJ_TYPE
   ,in_student_id  IN NUMBER,   in_salutation IN VARCHAR2
   ,in_first_name  IN VARCHAR2, in_last_name  IN VARCHAR2
   ,in_street_addr IN VARCHAR2, in_zip        IN VARCHAR2
   ,in_phone       IN VARCHAR2, in_employer   IN VARCHAR2
   ,in_reg_date    IN DATE,     in_cr_by      IN VARCHAR2
   ,in_cr_date     IN DATE,     in_mod_by     IN VARCHAR2
   ,in_mod_date    IN DATE)
RETURN SELF AS RESULT
IS
BEGIN
   -- Validate incoming value of zip
   SELECT zip
     INTO SELF.zip
     FROM zipcode
    WHERE zip = in_zip;

   -- Check incoming value of student ID
   -- If it is not populated, get it from the sequence
   IF in_student_id IS NULL
   THEN
      student_id := STUDENT_ID_SEQ.NEXTVAL;
   ELSE
      student_id := in_student_id;
   END IF;

   salutation        := in_salutation;
   first_name        := in_first_name;
   last_name         := in_last_name;
   street_address    := in_street_addr;
   phone             := in_phone;
   employer          := in_employer;
   registration_date := in_reg_date;

   IF in_cr_by IS NULL THEN created_by := USER;
   ELSE                     created_by := in_cr_by;
   END IF;

   IF in_cr_date IS NULL THEN created_date := SYSDATE;
   ELSE                       created_date := in_cr_date;
   END IF;

   IF in_mod_by IS NULL THEN modified_by := USER;
   ELSE                      modified_by := in_mod_by;
   END IF;

   IF in_mod_date IS NULL THEN modified_date := SYSDATE;
   ELSE                        modified_date := in_mod_date;
   END IF;

   RETURN;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN;
END;

CONSTRUCTOR FUNCTION student_obj_type (SELF IN OUT NOCOPY STUDENT_OBJ_TYPE
                                      ,in_student_id IN NUMBER)
RETURN SELF AS RESULT
IS
BEGIN
   SELECT student_id, salutation, first_name, last_name, street_address, zip
         ,phone, employer, registration_date, created_by, created_date
         ,modified_by, modified_date
     INTO SELF.student_id, SELF.salutation, SELF.first_name,
          SELF.last_name, SELF.street_address, SELF.zip,
          SELF.phone, SELF.employer, SELF.registration_date,
          SELF.created_by, SELF.created_date,
          SELF.modified_by, SELF.modified_date
     FROM student
    WHERE student_id = in_student_id;

   RETURN;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      RETURN;
END;

MEMBER PROCEDURE get_student_info
   (student_id  OUT NUMBER,   salutation OUT VARCHAR2
   ,first_name  OUT VARCHAR2, last_name  OUT VARCHAR2
   ,street_addr OUT VARCHAR2, zip        OUT VARCHAR2
   ,phone       OUT VARCHAR2, employer   OUT VARCHAR2
   ,reg_date    OUT DATE,     cr_by       OUT VARCHAR2
   ,cr_date     OUT DATE,     mod_by      OUT VARCHAR2
   ,mod_date    OUT DATE)
IS
BEGIN
   student_id  := SELF.student_id;
   salutation  := SELF.salutation;
   first_name  := SELF.first_name;
   last_name   := SELF.last_name;
   street_addr := SELF.street_address;
   zip         := SELF.zip;
   phone       := SELF.phone;
   employer    := SELF.employer;
   reg_date    := SELF.registration_date;
   cr_by       := SELF.created_by;
   cr_date     := SELF.created_date;
   mod_by      := SELF.modified_by;
   mod_date    := SELF.modified_date;
END;

STATIC PROCEDURE display_student_info (student_obj IN STUDENT_OBJ_TYPE)
IS
BEGIN
   DBMS_OUTPUT.PUT_LINE ('Student ID: '       ||student_obj.student_id);
   DBMS_OUTPUT.PUT_LINE ('Salutation: '       ||student_obj.salutation);
   DBMS_OUTPUT.PUT_LINE ('First Name: '       ||student_obj.first_name);
   DBMS_OUTPUT.PUT_LINE ('Last Name: '        ||student_obj.last_name);
   DBMS_OUTPUT.PUT_LINE ('Street Address: '   ||student_obj.street_address);
   DBMS_OUTPUT.PUT_LINE ('Zip: '              ||student_obj.zip);
   DBMS_OUTPUT.PUT_LINE ('Phone: '            ||student_obj.phone);
   DBMS_OUTPUT.PUT_LINE ('Employer: '         ||student_obj.employer);
   DBMS_OUTPUT.PUT_LINE ('Registration Date: '||student_obj.registration_date);
END;

ORDER MEMBER FUNCTION student (student_obj STUDENT_OBJ_TYPE)
RETURN INTEGER
IS
BEGIN
   IF    student_id < student_obj.student_id THEN RETURN -1;
   ELSIF student_id = student_obj.student_id THEN RETURN  0;
   ELSIF student_id > student_obj.student_id THEN RETURN  1;
   END IF;
END;

END;
/
```

The student object type created above has two overloaded constructor functions, member procedure, static procedure, and order function methods.

Both constructor functions have the same name as the object type. The first constructor function evaluates incoming values of student ID, ZIP code, created and modified users and dates. Specifically, it checks if incoming student ID is null then it populates it from the `STUDENT_ID_SEQ` if it is. It also validates that the incoming value of ZIP exists in the

ZIPCODE table. Finally, it checks if incoming values of created and modified user and date are null. If any of these incoming values are null, the constructor function populates corresponding attributes with the default values based on system functions USER and SYSDATE. The second constructor function initialize object instance based on the incoming value of student ID via the SELECT INTO statement.

The member procedure GET_STUDENT_INFO populates out parameters with corresponding values of object attributes. The static procedure DISPLAY_STUDENT_INFO displays values of the incoming student object. Recall that static methods do not have access to the data associated with a particular object type instance, and as a result, they may not reference default parameter SELF. The order member function compares two instances of the student object type based on values of the student_id attribute.

The newly created object type may be tested as follows:

For Example *ch23_17b.sql*

```
DECLARE
   v_student_obj1 student_obj_type;
   v_student_obj2 student_obj_type;

   v_result integer;
BEGIN
   -- Populate student objects via user-defined constructor methods
   v_student_obj1 := student_obj_type (in_student_id  => NULL
                                      ,in_salutation  => 'Mr.'
                                      ,in_first_name  => 'John'
                                      ,in_last_name   => 'Smith'
                                      ,in_street_addr => '123 Main Street'
                                      ,in_zip         => '00914'
                                      ,in_phone       => '555-555-5555'
                                      ,in_employer    => 'ABC Company'
                                      ,in_reg_date    => TRUNC(sysdate)
                                      ,in_cr_by       => NULL
                                      ,in_cr_date     => NULL
                                      ,in_mod_by      => NULL
                                      ,in_mod_date    => NULL);
   v_student_obj2 := student_obj_type(103);

   -- Display student information for both objects
   student_obj_type.display_student_info (v_student_obj1);
   DBMS_OUTPUT.PUT_LINE ('================================');
   student_obj_type.display_student_info (v_student_obj2);
   DBMS_OUTPUT.PUT_LINE ('================================');

   -- Compare student objects
   v_result := v_student_obj1.student(v_student_obj2);
   DBMS_OUTPUT.PUT_LINE ('The result of comparison is '||v_result);

   IF v_result = 1
   THEN
      DBMS_OUTPUT.PUT_LINE ('v_student_obj1 is greater than v_student_obj2');

   ELSIF v_result = 0
   THEN
```

```
      DBMS_OUTPUT.PUT_LINE ('v_student_obj1 is equal to v_student_obj2');

   ELSIF v_result = -1
   THEN
      DBMS_OUTPUT.PUT_LINE ('v_student_obj1 is less than v_student_obj2');
   END IF;

END;
```

The test script produces output as follows:

```
Student ID: 414
Salutation: Mr.
First Name: John
Last Name: Smith
Street Address: 123 Main Street
Zip: 00914
Phone: 555-555-5555
Employer: ABC Company
Registration Date: 11/20/2014 00:00
================================
Student ID: 103
Salutation: Ms.
First Name: J.
Last Name: Landry
Street Address: 7435 Boulevard East #45
Zip: 07047
Phone: 201-555-5555
Employer: Albert Hildegard Co.
Registration Date: 01/22/2007 00:00
================================
The result of comparison is 1
v_student_obj1 is greater than v_student_obj2
```