Table of Contents

[CHAPTER 17: OBJECT TYPE 2](#_Toc416725116)

[Theory 2](#_Toc416725117)

[AIM 4](#_Toc416725118)

[Lab Exercise 17: OBJECT TYPE 5](#_Toc416725119)

[1. Create Object Type 6](#_Toc416725120)

[2. Constructors, Member and Static Methods 18](#_Toc416725121)

[3. MAP and Order Member Methods 32](#_Toc416725122)

[SUMMARY 42](#_Toc416725123)

[REFERENCES 43](#_Toc416725124)

[INDEX 44](#_Toc416725125)

# CHAPTER 17: OBJECT TYPE

## Theory

Oracle presents and alternative model to represent your data in a database: Oracle Object-Relational model. In this model, the Object Type is the core unit, instead of a Table in a regular entity-relational model. The idea comes mainly to cope with a widely accepted programming model: Object-Oriented Programming (OOP). OOP developers can easily map their model in the database level without rearranging it to be represented in tables.

Object Type is a user-defined type (UDT) which can be used in almost all places Oracle defined type can be used. Thus, it can be used as a column type in a table or as a variable type in a PL/SQL block. The main structure of Object Type is somehow similar to a package in that it contains two parts: (1) Object Specification, (2) Object Body. However, it contains attributes, is used where UDT is used and can be stored in tables not like package. Object Type is also somehow similar to a "Class" in Java and C programming languages in that it contains attributes and methods that determine the functionality of the object instance.

**Object Type** Body  
----------------

Function getId() ......

Function setName() .....

Procedure printInfo() ....

**Object Instance**

--------------------  
id=5

Name="Ray"

DOB=''

**Object Instance**

--------------------  
id=5

Name="Ernesto"

DOB=''

**Object Type PERSON**

Methods

Attributes

Function getId

Function setName

Procedure printInfo

Id INT

Name VARCHAR2

DOB DATE

There are three main types of methods (function or procedures) that can be used in the Object Type:

1. **Member Methods**: Methods determine the current instance behavior or perform and action/operation on the current instance. Thus, the instance of the object should be declared first to access these methods.
2. **Static Methods**: Methods don't depend on any instance of the Object Type. Thus, you don't need to declare an instance of the object to access these methods.
3. **Constructor Methods**: methods that are invoked either implicitly or explicitly when new instance of the object is created.

You may notice that in a regular table-relation model, there is a separation between data and methods. So, you need to store data in tables and the functionality in Packages or Stored Functions/Procedures. A regular model may lead to hundreds of nested methods that actually related to specific data. In Object-Relational model, however, the functionality is stuck to data.

It seems that there are three attitudes to use Object Types among developers:

1. Neglecting Object model: some developers prefer to use a regular table-relation model and not to use an Object-relation model at all.
2. Adopting Object model: some developers found it appropriate to adopt the object model in replacement to table model.
3. Mixing both models: Some developers prefer to get the benefits of them together. Actually, this trend seems to be the most adopted approach.

## AIM

The AIM of the following exercise is to demonstrate the use Object Type.

The steps involved will include:

* Create Object Type
* Constructors, MEMBER and STATIC Methods
* MAP and ORDER Methods

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.

Estimated Completion Time:

25 minutes

# Lab Exercise 17: OBJECT TYPE

|  |
| --- |
|  |

## Create Object Type

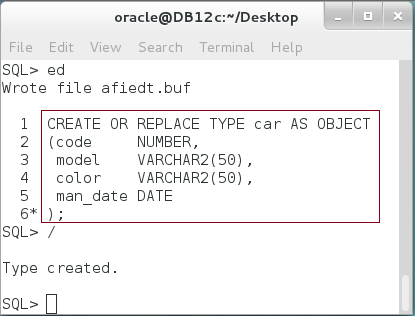
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

**Step 2:** Execute the following block:

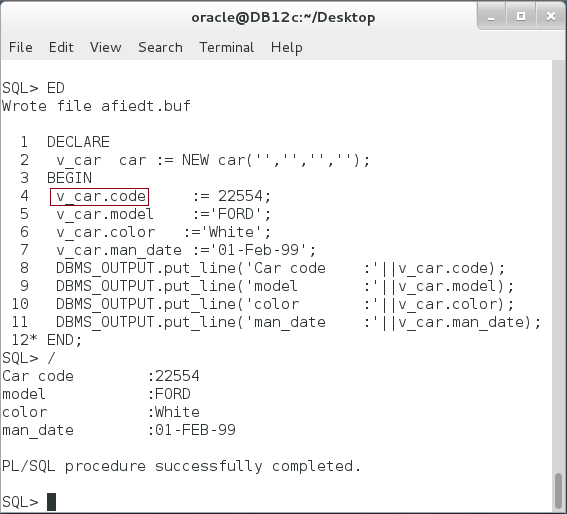
|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE** car **AS OBJECT** | Create object specification. |
| (code NUMBER, |
| model NUMBER, |
| color VARCHAR2(50), |
| man\_date DATE |
| ); |
| / |  |

****

**Please note:** The previous object does not have member functions or procedures. Thus, it does not need the object body. **Do you find the previous type similar a RECORD type? Can you create a standalone RECORD type?**

**Step 3:** You may make use of the previous type as shown below:

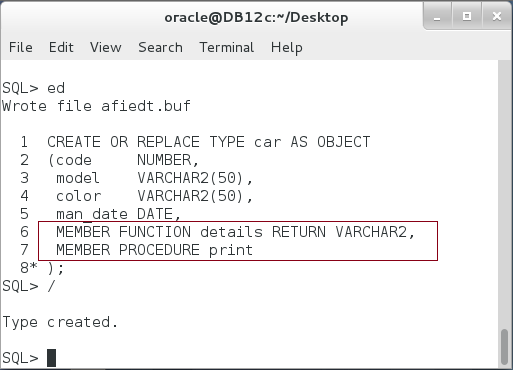
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Use a "." notation to use an Object instance |
| v\_car car := NEW car('','','',''); |
| BEGIN |
| **v\_car.code** := 22554; |
| **v\_car.model** :='FORD'; |
| **v\_car.color** :='White'; |
| **v\_car.man\_date** :='01-Feb-99'; |
| DBMS\_OUTPUT.put\_line('Car code :'||v\_car.code); |
| DBMS\_OUTPUT.put\_line('model :'||v\_car.model); |
| DBMS\_OUTPUT.put\_line('color :'||v\_car.color); |
| DBMS\_OUTPUT.put\_line('man\_date :'||v\_car.man\_date); |
| END; |
| / |  |



**Please note:** so far, the object type is very similar to Record type.

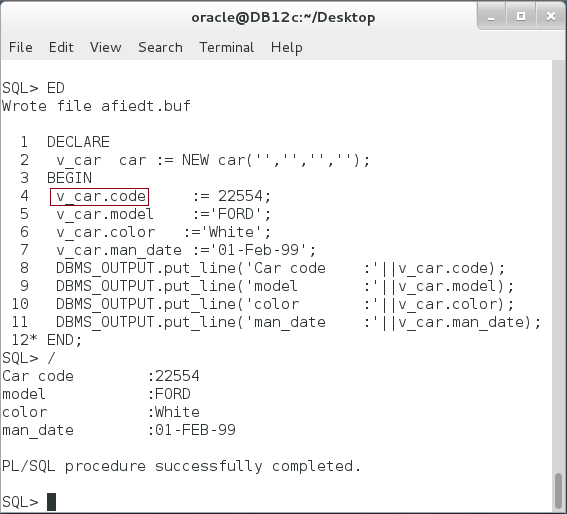
**Step 4:** We will modify the previous type to include function and procedure as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE car AS OBJECT |  |
| (code NUMBER, |  |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |  |
| man\_date DATE, | Add function and procedure. |
| **MEMBER FUNCTION** details RETURN VARCHAR2, |
| **MEMBER PROCEDURE** print |
| ); |  |
| / |  |



**Step 5:** Try to execute the previous block as shown below:

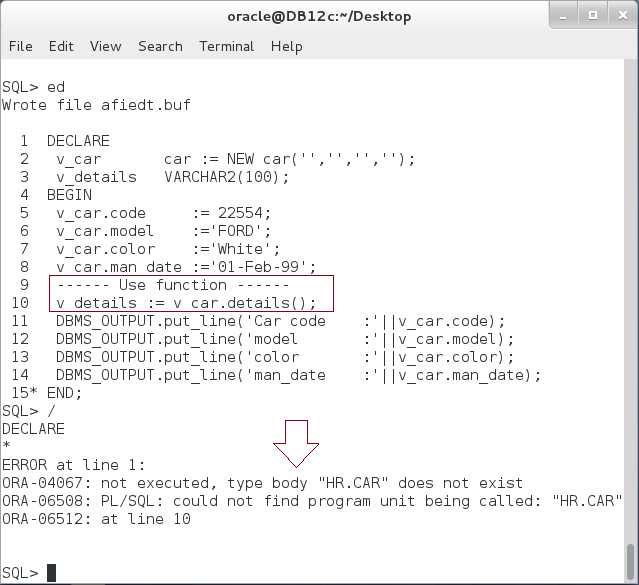
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_car car := NEW car('','','',''); |
| BEGIN |
| v\_car.code := 22554; |
| v\_car.model :='FORD'; |
| v\_car.color :='White'; |
| v\_car.man\_date :='01-Feb-99'; |
| DBMS\_OUTPUT.put\_line('Car code :'||v\_car.code); |
| DBMS\_OUTPUT.put\_line('model :'||v\_car.model); |
| DBMS\_OUTPUT.put\_line('color :'||v\_car.color); |
| DBMS\_OUTPUT.put\_line('man\_date :'||v\_car.man\_date); |
| END; |
| / |  |



**Please note:** It is somehow weird but useful to be able to use the object attributes even when its subprograms are not defined in the type body. Type specification is good in this context because it hides the body implementation failure and keeps the specification valid.

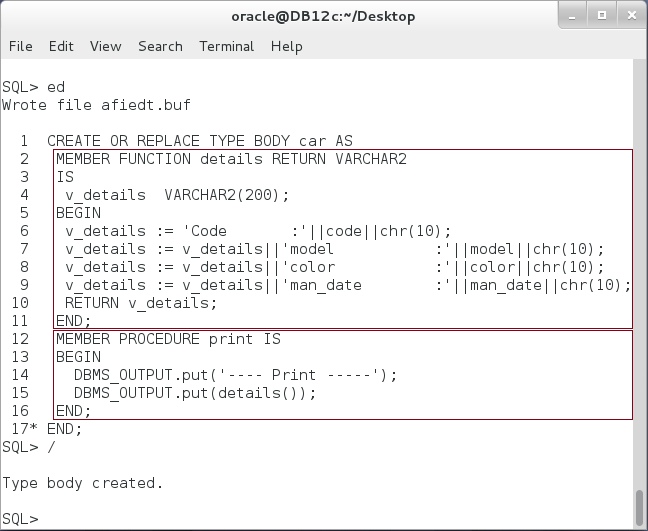
**Step 6:** Modify the previous block to try accessing the function "detail" as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_car car := NEW car('','','',''); |  |
| **v\_details VARCHAR2(100);** |  |
| BEGIN |  |
| v\_car.code := 22554; |  |
| v\_car.model :='FORD'; |  |
| v\_car.color :='White'; |  |
| v\_car.man\_date :='01-Feb-99'; |  |
| ------ Use function ------ | **Call an undefined function "details"** |
| **v\_details := v\_car.details();** |
| DBMS\_OUTPUT.put\_line('Car code :'||v\_car.code); |  |
| DBMS\_OUTPUT.put\_line('model :'||v\_car.model); |  |
| DBMS\_OUTPUT.put\_line('color :'||v\_car.color); |  |
| DBMS\_OUTPUT.put\_line('man\_date :'||v\_car.man\_date); |  |
| END; |  |
| / |  |

****

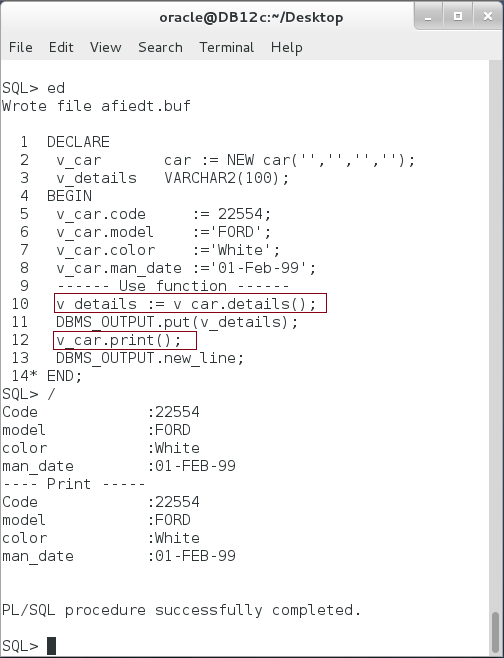
**Step 7:** Create the object body as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE BODY car AS |  |
| **MEMBER FUNCTION details RETURN VARCHAR2** | **"details" function** |
| IS |
| v\_details VARCHAR2(200); |
| BEGIN |  |
| v\_details := 'Code :'||code||chr(10); |  |
| v\_details := v\_details||'model :'||model||chr(10); |  |
| v\_details := v\_details||'color :'||color||chr(10); |  |
| v\_details := v\_details||'man\_date:'||man\_date||chr(10); |  |
| RETURN v\_details; |
| END; |  |
| **MEMBER PROCEDURE print IS** | **"print" procedure** |
| BEGIN |
| DBMS\_OUTPUT.put\_line('------ Print -------'); |
| DBMS\_OUTPUT.put(details()); |
| END; |
| END; |  |

****

**Step 8:** Use the type functions and procedures as shown below:

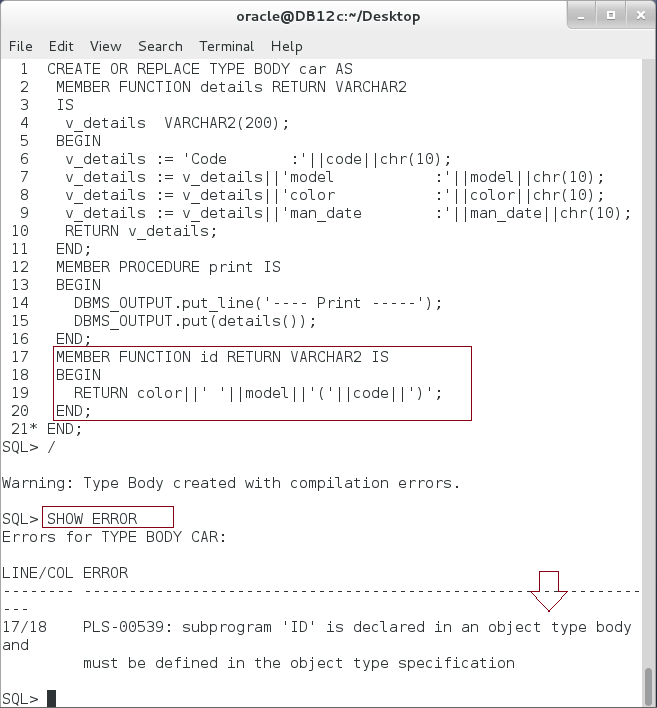
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_car car := NEW car('','','',''); |  |
| v\_details VARCHAR2(100); |  |
| BEGIN |  |
| v\_car.code := 22554; |  |
| v\_car.model :='FORD'; |  |
| v\_car.color :='White'; |  |
| v\_car.man\_date :='01-Feb-99'; |  |
| ------ Use function ------ | **Call a function "details"** |
| v\_details **:= v\_car.details();** |
| DBMS\_OUTPUT.put(v\_details); |  |
| **v\_car.print();** | **Call a procedure "print"** |
| DBMS\_OUTPUT.new\_line; |
| END; |  |
| / |  |

****

**Please note:** The Object Type may contains functions or procedures, somehow similar to the package.

**Step 8:** Try to define a function in Object Type body which is not declared in the specification:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE BODY** car AS |  |
| MEMBER FUNCTION details RETURN VARCHAR2 |  |
| IS |  |
| v\_details VARCHAR2(200); |  |
| BEGIN |  |
| v\_details := 'Code :'||code||chr(10); |  |
| v\_details := v\_details||'model :'||model||chr(10); |  |
| v\_details := v\_details||'color :'||color||chr(10); |  |
| v\_details := v\_details||'man\_date:'||man\_date||chr(10); |  |
| RETURN v\_details; |  |
| END; |  |
| MEMBER PROCEDURE print IS |  |
| BEGIN |
| DBMS\_OUTPUT.put\_line('---- Print -----'); |  |
| DBMS\_OUTPUT.put(details()); |  |
| END; |  |
| **MEMBER FUNCTION id RETURN VARCHAR2 IS** | Add "id" function. |
| **BEGIN** |
| **RETURN color||' '||model||'('||code||')';** |
| **END;** |
| END; |  |
| / |  |

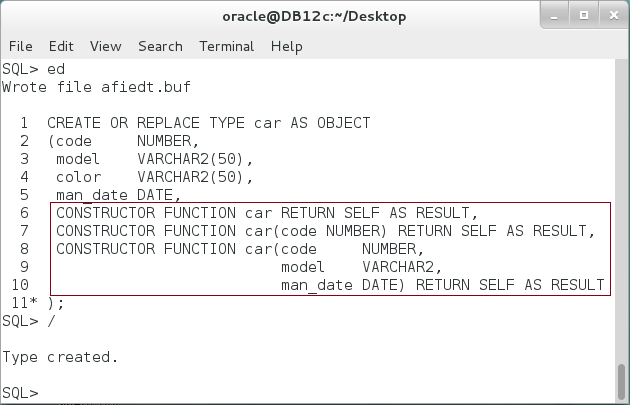
****

**Please note:** The Object Type should NOT declare functions or procedures in the body that does not declared previously in Object Type specification.

## Constructors, Member and Static Methods

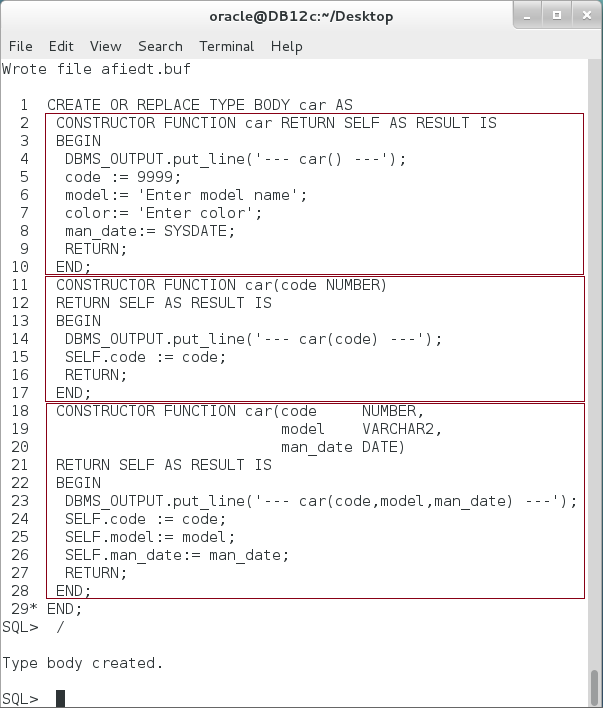
**Step 1:** A constructor is a special function that is invoked when creating new instance of the Object Type. Beside the default constructor, which is implicitly created with all attributes as input, you can define your own constructor with or without IN parameters. The name of the constructor should be the same as the Object Type. Execute the following:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE** car AS OBJECT |  |
| (code NUMBER, |  |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |  |
| man\_date DATE, |  |
| **CONSTRUCTOR FUNCTION** car **RETURN SELF AS RESULT**, | Three different constructors |
| **CONSTRUCTOR FUNCTION** car(code NUMBER) **RETURN SELF AS RESULT,** |
| **CONSTRUCTOR FUNCTION** car(code NUMBER, |
| model VARCHAR2, |
| man\_date DATE) **RETURN SELF AS RESULT** |
| ); |  |
| / |  |



**Step 2:** Implement the constructors in the Objet Type Body as shown below:

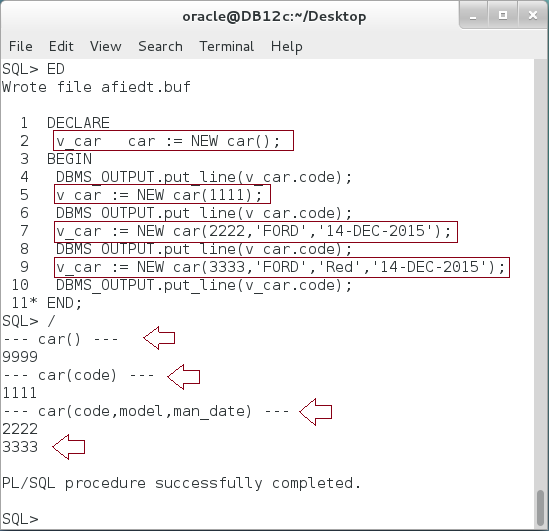
|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY car AS | A constructor that does not have any parameters. |
| **CONSTRUCTOR FUNCTION** car **RETURN SELF AS RESULT IS** |
| **BEGIN** |
| DBMS\_OUTPUT.put\_line('--- car() ---'); |
| code := 9999; |
| model:= 'Enter model name'; |
| color:= 'Enter color'; |
| man\_date:= SYSDATE; |  |
| RETURN; |  |
| **END**; |  |
| **CONSTRUCTOR FUNCTION car(code NUMBER)** | A constructor that has one parameter: code |
| **RETURN SELF AS RESULT IS** |
| **BEGIN** |
| DBMS\_OUTPUT.put\_line('--- car(code) ---'); |
| SELF.code := code; |
| RETURN; |
| **END**; |
| **CONSTRUCTOR FUNCTION car(code NUMBER,** | A constructor that has three parameters:  code, model, man\_date |
| **model VARCHAR2,** |
| **man\_date DATE)** |
| **RETURN SELF AS RESULT IS** |
| **BEGIN** |
| DBMS\_OUTPUT.put\_line('--- car(code,model,man\_date) ---'); |
| SELF.code := code; |
| SELF.model:= model; |
| SELF.man\_date:= man\_date; |
| RETURN; |
| **END;** |
| END; |  |
| / |  |

****

**Please note:** A **SELF** is a built-in parameter that denotes the object instance currently invoking the method. Using **SELF** parameter helps you distinguishing between the function's parameters and the object's attributes, for example, the IN parameter "code" is distinguished from "SELF.code". A constructor must return **SELF**.

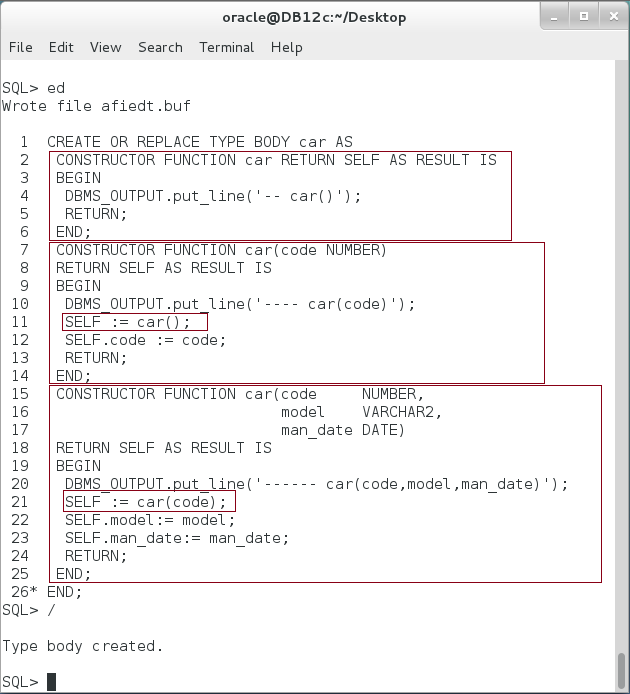
**Step 3:** To know more about a selected package, first you may use a description command:

|  |  |
| --- | --- |
| Line |  |
| DECLARE | Create new instance using "**car()**" constructor. |
| v\_car car := **NEW car();** |
| BEGIN |  |
| DBMS\_OUTPUT.put\_line(v\_car.code); |  |
| v\_car := **NEW car(1111);** | Create new instance using "**car(code)**" constructor. |
| DBMS\_OUTPUT.put\_line(v\_car.code); |
| v\_car := **NEW car(2222,'FORD','14-DEC-2015');** | Create new instance using "**car(code,model, man\_date)**" constructor. |
| DBMS\_OUTPUT.put\_line(v\_car.code); |
| v\_car := **NEW car(3333,'FORD','Red','14-DEC-2015');** | Create new instance using **implicit** **constructor**. |
| DBMS\_OUTPUT.put\_line(v\_car.code); |
| END; |



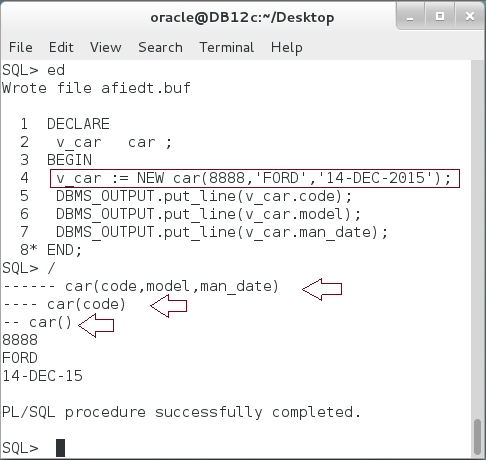
**Step 4:** To avoid repeating the conditions on attribute values on each constructor, you may call constructor inside the constructor as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY car AS |  |
| **CONSTRUCTOR FUNCTION car RETURN** SELF AS RESULT IS |  |
| BEGIN |  |
| DBMS\_OUTPUT.put\_line('-- car()'); |  |
| RETURN; |  |
| END; |  |
| **CONSTRUCTOR FUNCTION car(code NUMBER)** | "**car(code)**" constructor calls "**car()**" constructor. |
| **RETURN SELF AS RESULT IS** |
| BEGIN |
| DBMS\_OUTPUT.put\_line('---- car(code)'); |
| **SELF := car();** |
| SELF.code := code; |
| RETURN; |
| END; |
| **CONSTRUCTOR FUNCTION car(code NUMBER,** | "**car(code,model, man\_date)**" constructor calls "**car(code)**" constructor. |
| **model VARCHAR2,** |
| **man\_date DATE)** |
| **RETURN SELF AS RESULT IS** |
| BEGIN |
| DBMS\_OUTPUT.put\_line('----- car(code,model,man\_date)'); |
| **SELF := car(code);** |
| SELF.model:= model; |
| SELF.man\_date:= man\_date; |
| RETURN; |
| END; |  |
| END; |  |
| / |  |



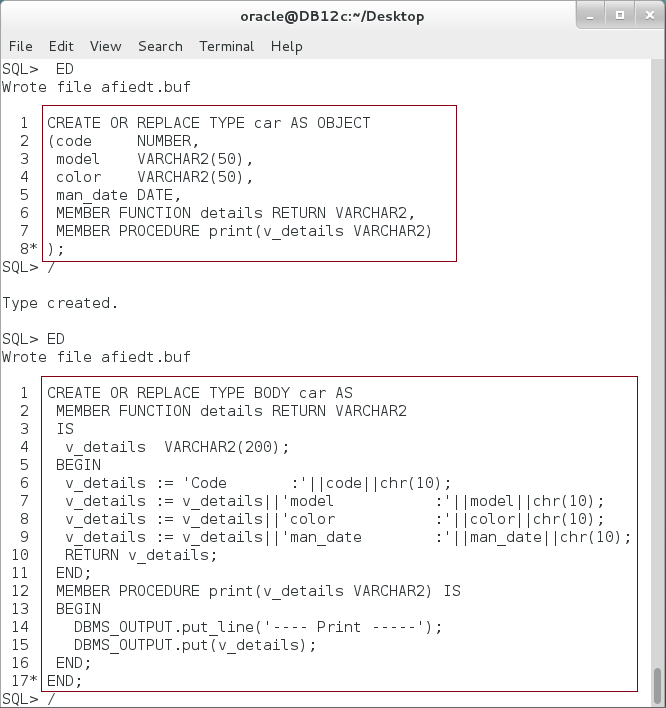
**Step 4:** Test the previous implementation using the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_car car ; |  |
| BEGIN | Calling "car(code,model, man\_date)" will automatically call "car(code)" and "car()" constructors. |
| **v\_car := NEW car(8888,'FORD','14-DEC-2015');** |
| DBMS\_OUTPUT.put\_line(v\_car.code); |
| END; |



**Step 5:** A MEMBER function/procedure is a subprogram that depends on the Object Instance. Thus, you must create an instance of the object before using member function/procedure. Re-create the previous "car" object as shown below:

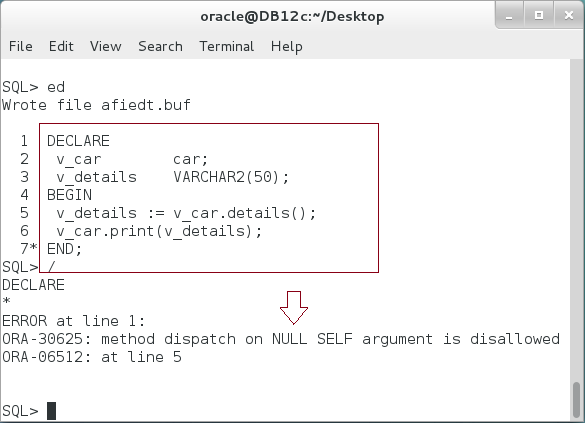
|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE TYPE car AS OBJECT** |  |
| (code NUMBER, |  |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |  |
| man\_date DATE, |  |
| **MEMBER FUNCTION details RETURN VARCHAR2,** |  |
| **MEMBER PROCEDURE print(v\_details VARCHAR2)** |  |
| ); |  |
| / |  |
|  |  |
| **CREATE OR REPLACE TYPE BODY car AS** |  |
| **MEMBER FUNCTION details RETURN VARCHAR2** |  |
| IS |  |
| v\_details VARCHAR2(200); |  |
| BEGIN |  |
| v\_details := 'Code :'||code||chr(10); |  |
| v\_details := v\_details||'model:'||model||chr(10); |  |
| v\_details := v\_details||'color :'||color||chr(10); |  |
| v\_details := v\_details||'man\_date:'||man\_date||chr(10); |  |
| RETURN v\_details; |  |
| END; |  |
| **MEMBER PROCEDURE print(v\_details VARCHAR2) IS** |  |
| BEGIN |  |
| DBMS\_OUTPUT.put\_line('---- Print -----'); |  |
| DBMS\_OUTPUT.put(v\_details); |  |
| END; |  |
| END; |  |
| / |  |



**Please note:** "car" Object Type has two member subprograms:"details" and "print".

**Step 5:** Try to access the member subprogram without declaring new instance as shown below:

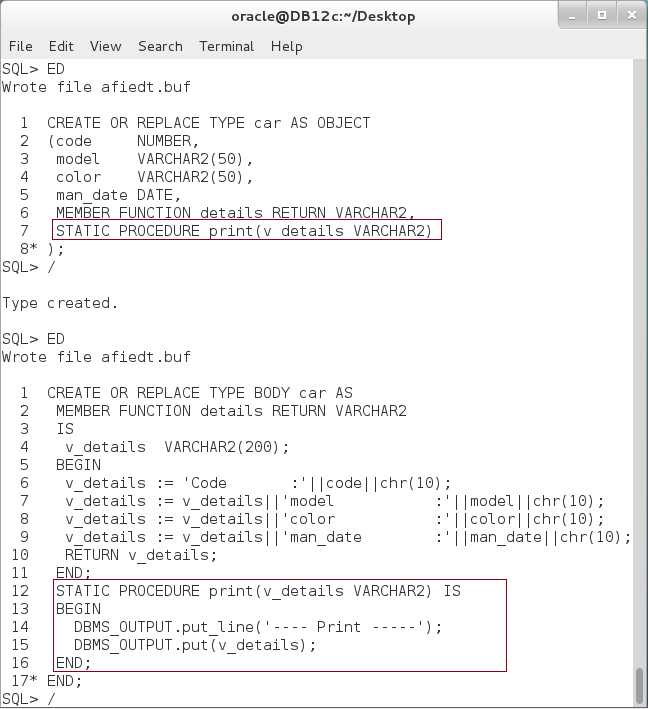
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_car car; |  |
| v\_details VARCHAR2(50); |  |
| BEGIN |  |
| v\_details **:= v\_car.details();** | Call member subprograms without creating new instance. |
| **v\_car.print(v\_details);** |
| END; |
| / |



**You may notice:** A "print" member procedure does not actually depend on the current instance. It just print the input parameter on the SQL\*Plus console! However, since it was defined as member procedure, it can't be called unless you create an Object instance.

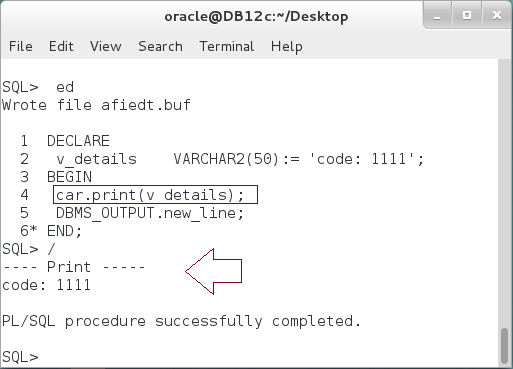
**Step 6:** Re-create "car" object to be STATIC procedure as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE car AS OBJECT |  |
| (code NUMBER, |  |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |  |
| man\_date DATE, |  |
| MEMBER FUNCTION details RETURN VARCHAR2, |  |
| **STATIC PROCEDURE print(v\_details VARCHAR2)** |  |
| ); |  |
| / |  |
|  |  |
| CREATE OR REPLACE TYPE BODY car AS |  |
| MEMBER FUNCTION details RETURN VARCHAR2 |  |
| IS |  |
| v\_details VARCHAR2(200); |  |
| BEGIN |  |
| v\_details := 'Code :'||code||chr(10); |  |
| v\_details := v\_details||'model:'||model||chr(10); |  |
| v\_details := v\_details||'color :'||color||chr(10); |  |
| v\_details := v\_details||'man\_date:'||man\_date||chr(10); |  |
| RETURN v\_details; |  |
| END; |  |
| **STATIC PROCEDURE print(v\_details VARCHAR2) IS** |  |
| BEGIN |  |
| DBMS\_OUTPUT.put\_line('---- Print -----'); |  |
| DBMS\_OUTPUT.put(v\_details); |  |
| END; |  |
| END; |  |
| / |  |



**Step 7:** Try to access the STATIC subprogram without declaring new instance as shown below:

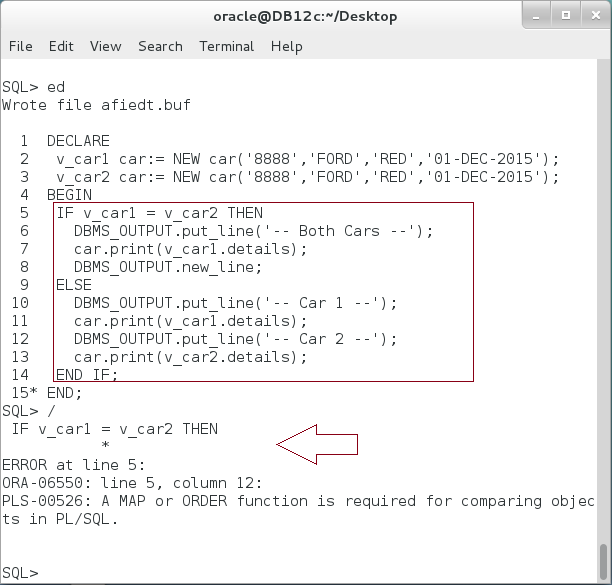
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_details VARCHAR2(50):= 'code: 1111'; |  |
| BEGIN |  |
| **car.print(v\_details);** | Use STATIC procedure. |
| DBMS\_OUTPUT.new\_line; |
| END; |
| / |



## MAP and Order Member Methods

**Step 1:** Execute the following block:

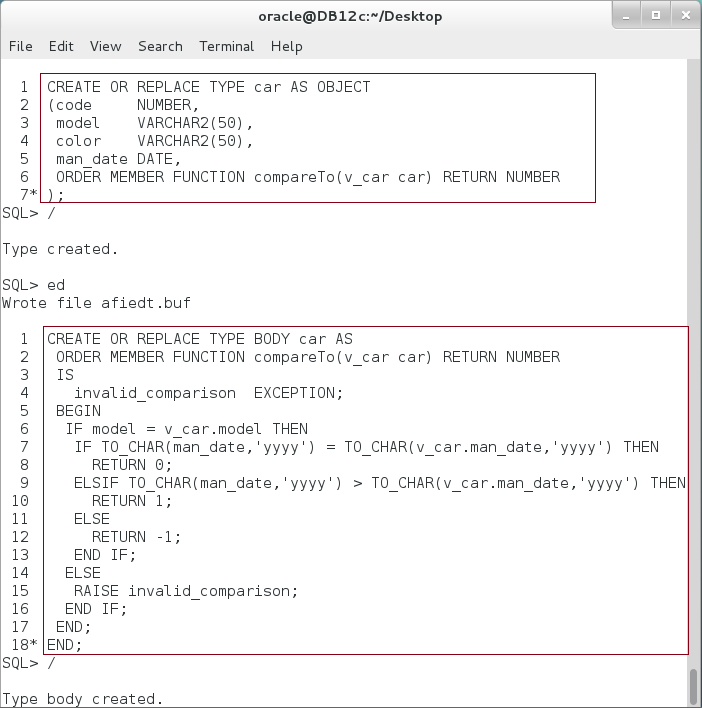
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Define two identical object instances. |
| v\_car1 car:= NEW car('8888','FORD','RED','01-DEC-2015'); |
| v\_car2 car:= NEW car('8888','FORD','RED','01-DEC-2015'); |
| BEGIN |
| IF **v\_car1 = v\_car2** THEN | Compare two objects using "=" operation. |
| DBMS\_OUTPUT.put\_line('-- Both Cars --'); |
| car.print(v\_car1.details); |  |
| DBMS\_OUTPUT.new\_line; |  |
| ELSE |  |
| DBMS\_OUTPUT.put\_line('-- Car 1 --'); |  |
| car.print(v\_car1.details); |  |
| DBMS\_OUTPUT.put\_line('-- Car 2 --'); |  |
| car.print(v\_car2.details); |  |
| END IF; |  |
| END; |  |
| / |  |

****

**Please note:** You must define MAP or ORDER function to compare two objects.

**Step 2:** Re-Create "car" Object Type and define an ORDER function as shown below:

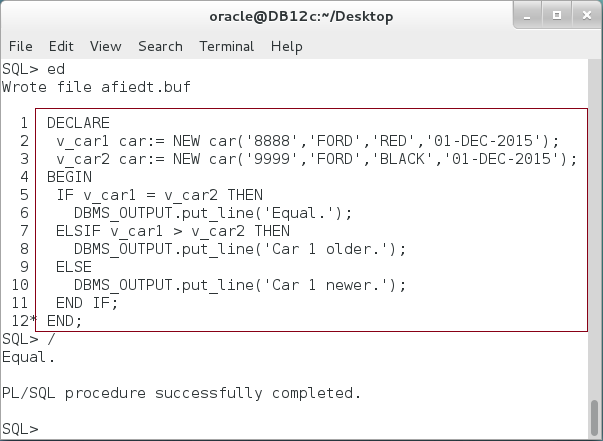
|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE car AS OBJECT |  |
| (code NUMBER, |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |
| man\_date DATE, |  |
| **ORDER MEMBER FUNCTION compareTo(v\_car car) RETURN NUMBER** |
| ); |  |
| / |  |
|  |  |
| **CREATE OR REPLACE TYPE BODY car AS** |  |
| **ORDER MEMBER FUNCTION compareTo(v\_car car) RETURN NUMBER** |  |
| IS |  |
| invalid\_comparison EXCEPTION; |  |
| BEGIN |  |
| IF model = v\_car.model THEN |  |
| IF TO\_CHAR(man\_date,'yyyy') = TO\_CHAR(v\_car.man\_date,'yyyy') THEN |  |
| RETURN 0; |  |
| ELSIF TO\_CHAR(man\_date,'yyyy') > TO\_CHAR(v\_car.man\_date,'yyyy') THEN |  |
| RETURN 1; |  |
| ELSE |  |
| RETURN -1; |  |
| END IF; |  |
| ELSE |  |
| RAISE invalid\_comparison; |  |
| END IF; |  |
| END; |  |
| END; |  |
| / |  |

****

**Please note:** MAP function should take one input from the same Object Type to compare it with SELF object.

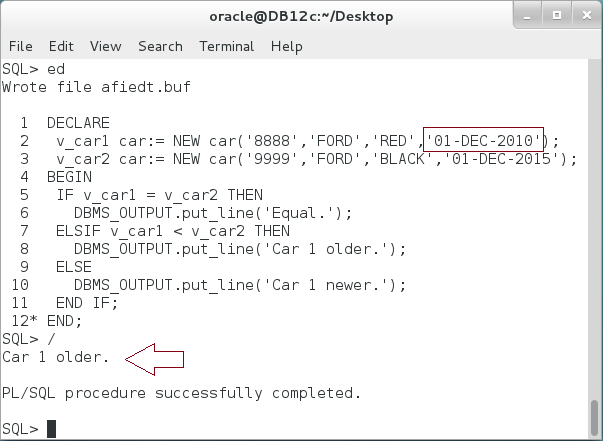
**Step 3:** Try to compare "v\_car1" and "v\_car2" Object Type using MAP function as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Object instances are different in colors and code numbers. However, it is logical to state that they are equal. |
| v\_car1 car:= NEW car('8888','FORD','RED','01-DEC-2015'); |
| v\_car2 car:= NEW car('9999','FORD','BLACK','01-DEC-2015'); |
| BEGIN |
| IF v\_car1 = v\_car2 THEN |
| DBMS\_OUTPUT.put\_line('Equal.'); |
| ELSIF v\_car1 < v\_car2 THEN |
| DBMS\_OUTPUT.put\_line('Car 1 older.'); |  |
| ELSE |  |
| DBMS\_OUTPUT.put\_line('Car 1 newer.'); |  |
| END IF; |  |
| END; |  |
| / |  |

****

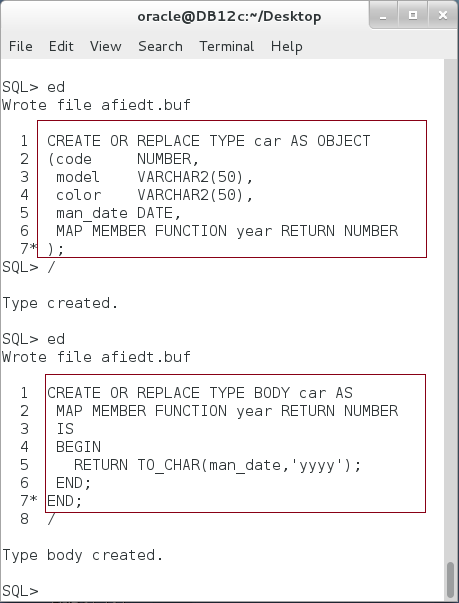
**Step 4:** Try to change the manufacturing date as shown:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_car1 car:= NEW car('8888','FORD','RED',**'01-DEC-2010'**); |
| v\_car2 car:= NEW car('9999','FORD','BLACK','01-DEC-2015'); |
| BEGIN |
| IF v\_car1 = v\_car2 THEN |
| DBMS\_OUTPUT.put\_line('Equal.'); |
| ELSIF v\_car1 < v\_car2 THEN |
| DBMS\_OUTPUT.put\_line('Car 1 older.'); |  |
| ELSE |  |
| DBMS\_OUTPUT.put\_line('Car 1 newer.'); |  |
| END IF; |  |
| END; |  |
| / |  |

****

**Step 5:** You are not allowed to a MAP function with ORDER function. You should select one of them at most. Add a MAP method in replacement to ORDER function and re-create "car" Object Type as shown below:

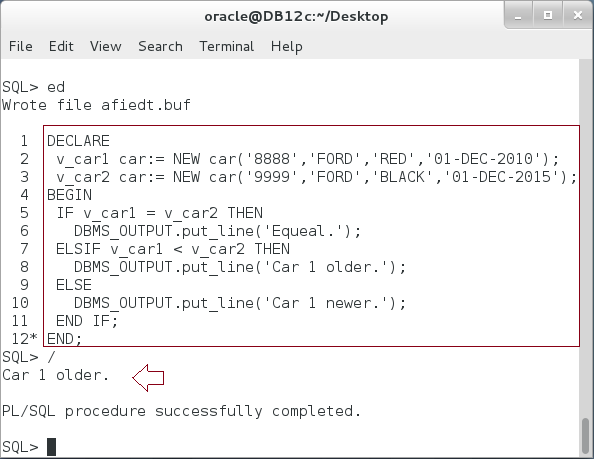
|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE car AS OBJECT |  |
| (code NUMBER, |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |
| man\_date DATE, |  |
| **MAP MEMBER FUNCTION year RETURN NUMBER** |
| ); |  |
| / |  |
|  |  |
| CREATE OR REPLACE TYPE BODY car AS |  |
| **MAP MEMBER FUNCTION year RETURN NUMBER** |  |
| IS |  |
| BEGIN |  |
| **RETURN TO\_CHAR(man\_date,'yyyy');** |  |
| END; |  |
| END; |  |
| / |  |

****

**Please note:**  MAP function should not have any parameter to determine the order of Object Instance in comparison to other Object Instances. Thus, it should depend only on its attribute value. MAP function is more efficient than ORDER function to compare a list of Object Instances. However, it does not support that details which can be done using in ORDER function as you can see in the previous function.

**Step 6:** Test the MAP function as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_car1 car:= NEW car('8888','FORD','RED',**'01-DEC-2010**'); |
| v\_car2 car:= NEW car('9999','FORD','BLACK',**'01-DEC-2015'**); |
| BEGIN |
| IF v\_car1 = v\_car2 THEN |
| DBMS\_OUTPUT.put\_line('Equal.'); |
| ELSIF v\_car1 **<** v\_car2 THEN |
| DBMS\_OUTPUT.put\_line('Car 1 older.'); |  |
| ELSE |  |
| DBMS\_OUTPUT.put\_line('Car 1 newer.'); |  |
| END IF; |  |
| END; |  |
| / |  |

****

# SUMMARY

Object Type is a user-defined type that can be used to represent data in addition to functionality in the same unit. Object Type can be seen as CLASS in Object-Oriented Programming; it provides PL/SQL developer with the ability to represent complex data structure. PL/SQL developer may map a business rules from class diagram directly to Object-Type diagram; there is no need to re-map or to rethink about how to represent the model in a Table-Relational model. Some developers may ignore Object Type as it may be seen as an alternative to Table-Relational model. Other group of developers may stand for Object-Relational model as replacement to Table-Relational model. Finally, other group tries to mix between both trends. The idea is to get the most of both trends. Object Type subprograms can be CONSTRUCTOR, MEMBER or STATIC. MEMBER subprograms are regular, MAP or ORDER.

After completing this lab exercise, you should be able to create Object Type.

# REFERENCES

* https://docs.oracle.com/database/121/ADOBJ/adobjint.htm#ADOBJ001
* https://docs.oracle.com/database/121/ADOBJ/adobjbas.htm#ADOBJ7095

# INDEX

Object Type 2, 3, 4, 6, 14, 16, 25, 30, 32, 34, 37

Object-Relational model 2, 3

OOP 2