Table of Contents

[CHAPTER 19: OBJECT TYPE MANIPULATION 2](#_Toc416726495)

[Theory 2](#_Toc416726496)

[AIM 4](#_Toc416726497)

[Lab Exercise 19: OBJECT TYPE MANIPULATION 5](#_Toc416726498)

[1. Inheritance 6](#_Toc416726499)

[2. Overloading Methods 17](#_Toc416726500)

[3. Overriding Methods 22](#_Toc416726501)

[SUMMARY 31](#_Toc416726502)

[REFERENCES 32](#_Toc416726503)

[INDEX 33](#_Toc416726504)

# CHAPTER 19: OBJECT TYPE MANIPULATION

## Theory

Inheritance is a concept which is well known in Object Oriented Programming (OOP). According to this concept, the objects should have a hierarchal structure, in which one object represents a parent of other children objects. Consequently, a child object inherits both the attributes and methods of its father. Thus, a child concerns mainly on adding some attributes (more specialization) and methods (more functionality) or re-interpreting its father methods (overriding methods).

The value of inheritance concept appears mainly on maintenance and avoiding redundancy. When an error is raised in object functionality, it would be easy to detect and modify the child object type in relation than to detect and modify the object types without inheritance. Moreover, if you modify the parent's methods, the modification is automatically propagated to all children's methods.

By default, all Object Types are final: no children can be based on this Object type. To make Object Type inheritable, use NOT FINAL clause at the end of Object Type declaration. In addition, all Object Types are 'instantiable' by default: you may create an object instance of these objects. To make Object Type like an abstract class which can't be instantiated, use NOT INSTANTIABLE clause.

Overloading and Overriding methods are some useful concepts in hieratical structure. Overloading methods refers to the case when you create the same method twice or more in the Object Type but with different parameters' type or order. PL/SQL compiler selects which method to execute depending on the parameters' signature. Overriding methods, on the other hand, refers to the case when you redefine the methods inherited from the object father. PL/SQL compiler selects which methods to execute depending on the implicit parameter "SELF" of the method.

## AIM

The AIM of the following exercise is to explore the inheritance and overriding methods of Object Type

The steps involved will include:

* Inheritance
* Overloading Methods
* Overriding 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.
* Object Types
* Object Type Storage

Estimated Completion Time:

25 minutes

# Lab Exercise 19: OBJECT TYPE MANIPULATION

|  |
| --- |
|  |

## Inheritance

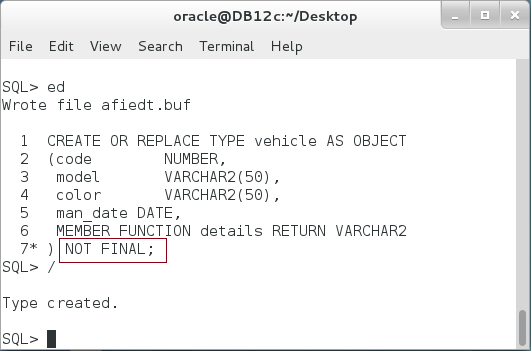
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

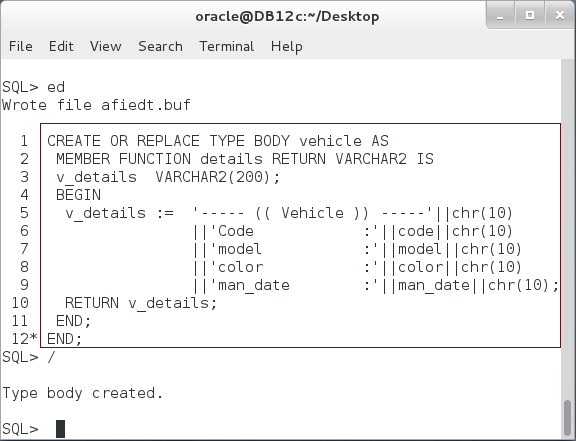
**Step 2:** Create a parent type, vehicle. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE** vehicle AS OBJECT | Create object specification with "**NOT FINAL**" clause. |
| (code NUMBER, |
| model VARCHAR2(50), |
| color VARCHAR2(50), |
| man\_date DATE, |
| MEMBER FUNCTION details RETURN VARCHAR2 |
| ) **NOT FINAL**; |  |
| / |  |

****

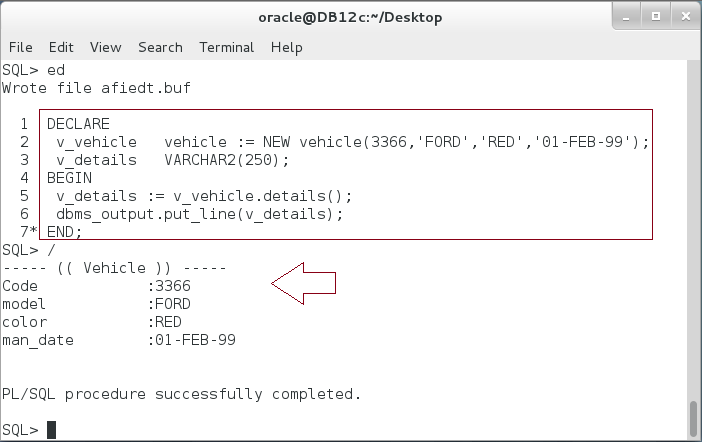
**Step 3:** Create a Type Body as shown below:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE BODY** vehicle AS | Create type body |
| MEMBER FUNCTION details RETURN VARCHAR2 IS |
| v\_details VARCHAR2(200); |
| BEGIN |
| v\_details := '----- (( Vehicle )) -----'||chr(10) |
| ||'Code :'||code||chr(10) |
| ||'model :'||model||chr(10) |
| ||'color :'||color||chr(10) |
| ||'man\_date :'||man\_date||chr(10); |
| RETURN v\_details; |
| END; |
| END; |
| / |  |



**Step 4:** Of course, you can use the type as usual: declare variables, set attitudes' values, and call methods as show below:

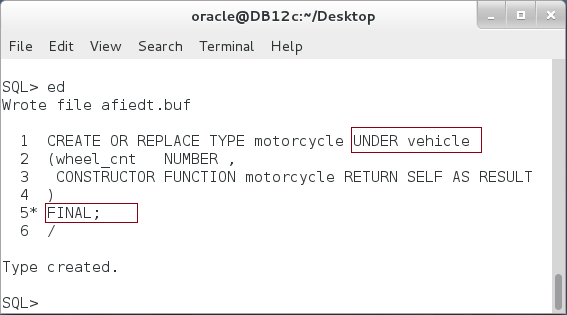
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_vehicle vehicle := NEW vehicle(3366,'FORD','RED','01-FEB-99'); |  |
| v\_details VARCHAR2(250); | Use the type as usual. |
| BEGIN |
| v\_details := v\_vehicle.details(); |
| dbms\_output.put\_line(v\_details); |
| END; |
| / |  |



**Please note:** So far, there is nothing new. In the next steps, you will create two children Object Types based on this Object Type.

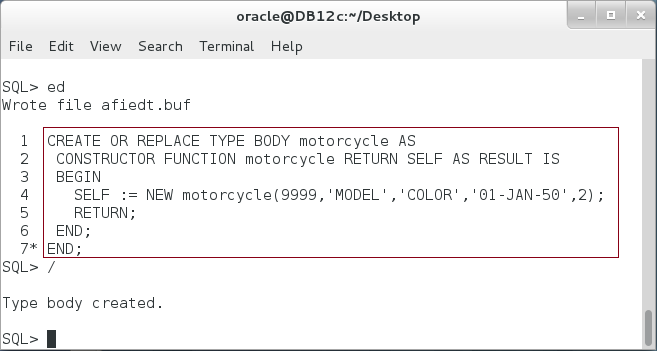
**Step 5:** Create the first Object child, "motorcycle". Execute the following DDL statement:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE motorcycle **UNDER vehicle** | create "motorcycle" based on parent type "vehicle" |
| (wheel\_cnt NUMBER , |
| CONSTRUCTOR FUNCTION motorcycle RETURN SELF AS RESULT |
| ) |
| **FINAL;** |
| / |



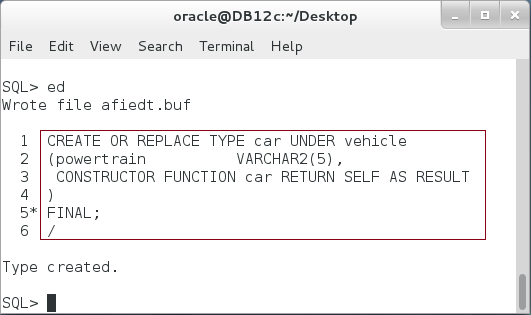
**Step 6:** Implement the child type body as shown below:

|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE BODY** motorcycle AS |  |
| CONSTRUCTOR FUNCTION motorcycle RETURN SELF AS RESULT IS |  |
| BEGIN |  |
| SELF := NEW motorcycle(9999,'MODEL','COLOR','01-JAN-50',2); |  |
| RETURN; |  |
| END; |  |
| END; |  |
| / |  |

****

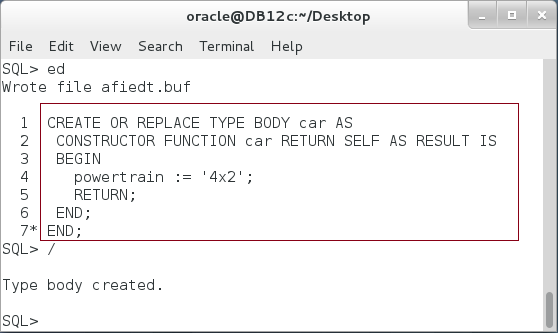
**Step 7:** Create the second child type as shown below:

|  |  |
| --- | --- |
| Command | Description |
| CREATE OR REPLACE TYPE car **UNDER vehicle** |  |
| (powertrain VARCHAR2(5), | **Create "car" data type based on the parent type, "vehicle"** |
| CONSTRUCTOR FUNCTION car RETURN SELF AS RESULT |
| ) |
| **FINAL;** |
| / |  |

****

**Step 8:** Implement "car" child object body as shown below:

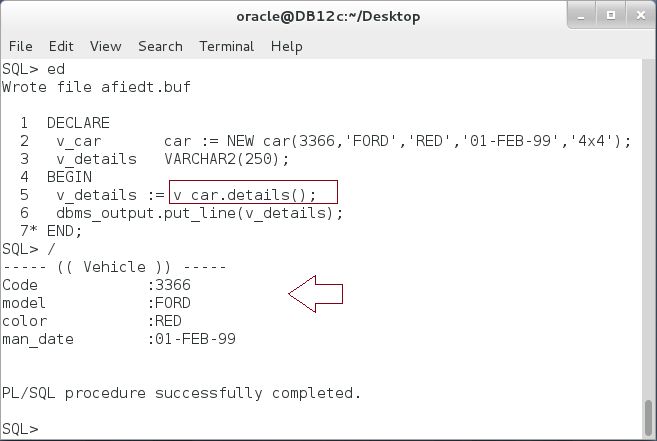
|  |  |
| --- | --- |
| Command | Description |
| **CREATE OR REPLACE TYPE BODY** car AS |  |
| CONSTRUCTOR FUNCTION car RETURN SELF AS RESULT IS | implement a constructor. |
| BEGIN |
| powertrain := '4x2'; |
| RETURN; |
| END; |
| END; |
| / |  |

****

**Please note:** Both children, "motorcycle" and "car", have no methods but their constructors

**Step 8:** Execute the following block:

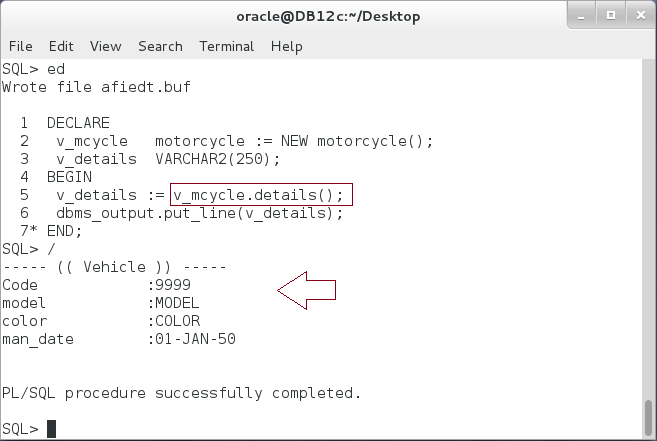
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | Create an Object Instance of "**car**" type. Execute "**details**" methods which was declared on its parent type, "**vehicle**". |
| v\_car car := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); |
| v\_details VARCHAR2(250); |
| BEGIN |
| v\_details := v\_car.**details();** |
| dbms\_output.put\_line(v\_details); |
| END; |
| / |

****

**Please note:** "**car**" object inherits the methods and attributes from its parents, "**vehicle**". In the previous example, "**car**" type does not declare any attribute except "**powertrain**" and does not declare any function beside its constructor.

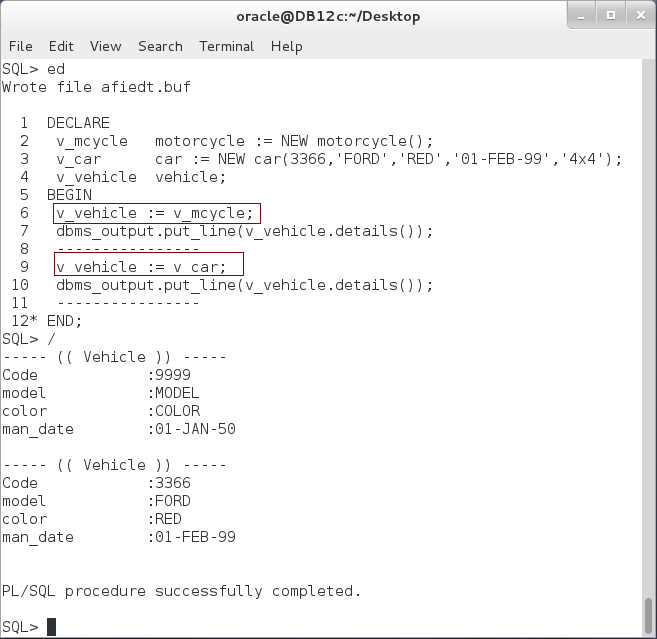
**Step 9:** "**motorcycle**" is as "**car**" type. It inherits attributes and methods from "**vehicle**" object. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| v\_mcycle motorcycle := NEW motorcycle(); |  |
| v\_details VARCHAR2(250); |  |
| BEGIN |  |
| v\_details := **v\_mcycle.details();** |  |
| dbms\_output.put\_line(v\_details); |  |
| END; |  |
| / |  |

****

**Step 10:** You may also use the polymorphism concepts with inherited objects, so you may use the parent object to handle or to refer to its children. Execute the following block:

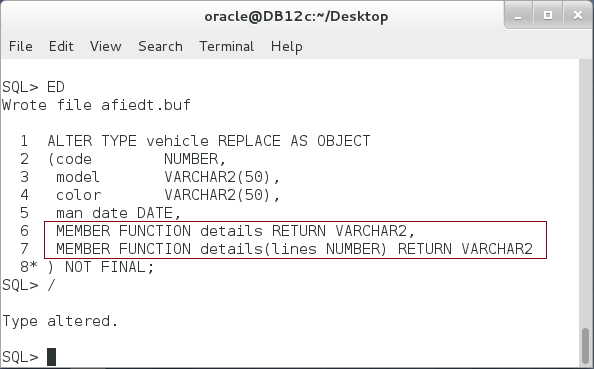
|  |  |
| --- | --- |
| Command | Description |
| DECLARE | **motorcycle** object. |
| v\_mcycle motorcycle := NEW motorcycle(); |
| v\_car car := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); | **car** object |
| v\_vehicle vehicle; | **vehicle** object |
| BEGIN |  |
| v\_vehicle := v\_mcycle; | **vehicle** object handle **motorcycle** object |
| dbms\_output.put\_line(v\_vehicle.details()); |
| ---------------- |
| v\_vehicle := v\_car; | **vehicle** object handle **car** object |
| dbms\_output.put\_line(v\_vehicle.details()); |
| ---------------- |
| END; |
| / |  |

****

## Overloading Methods

**Step 1:** Add new method, "details", to the parent object type "vehicle". Execute the following DDL statement:

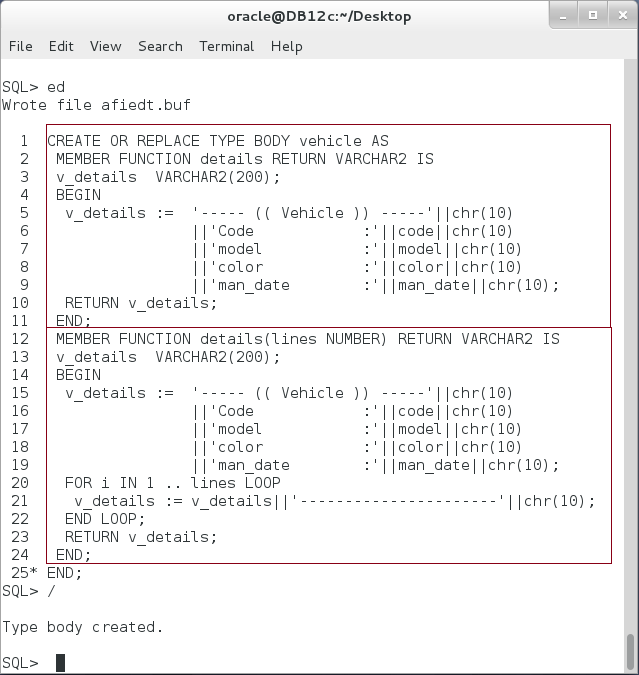
|  |  |
| --- | --- |
| Command | Description |
| ALTER TYPE vehicle REPLACE AS OBJECT |  |
| (code NUMBER, |  |
| model VARCHAR2(50), |  |
| color VARCHAR2(50), |  |
| man\_date DATE, |  |
| MEMBER FUNCTION details RETURN VARCHAR2, |  |
| **MEMBER FUNCTION details(lines NUMBER) RETURN** VARCHAR2 |  |
| ) NOT FINAL; |  |
| / |  |

****

**Please note:** The new added function "details" is exactly the same as the previously declared "details" function except it has a parameter "lines". This is a method overloading.

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

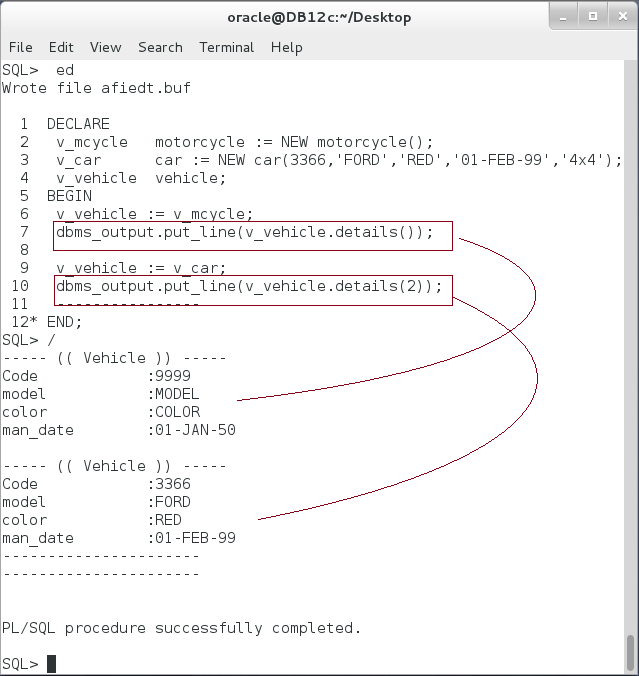
|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY vehicle AS |  |
| MEMBER FUNCTION **details** RETURN VARCHAR2 IS | First method "details" with no parameters. |
| v\_details VARCHAR2(200); |
| BEGIN |
| v\_details := '----- (( Vehicle )) -----'||chr(10) |
| ||'Code :'||code||chr(10) |
| ||'model :'||model||chr(10) |
| ||'color :'||color||chr(10) |
| ||'man\_date :'||man\_date||chr(10); |
| RETURN v\_details; |
| END; |
| MEMBER FUNCTION **details(lines NUMBER)** RETURN VARCHAR2 IS | The second method "details" with one input parameter |
| v\_details VARCHAR2(200); |
| BEGIN |
| v\_details := '----- (( Vehicle )) -----'||chr(10) |
| ||'Code :'||code||chr(10) |
| ||'model :'||model||chr(10) |
| ||'color :'||color||chr(10) |
| ||'man\_date :'||man\_date||chr(10); |
| **FOR i IN 1 .. lines LOOP** |
| **v\_details := v\_details||'----------------------'||chr(10);** |
| **END LOOP;** |
| RETURN v\_details; |
| END; |
| END; |  |
| / |  |



**Please note:** the second "details" method with one input parameter "lines", print separator lines after the details.

**Step 3:** Execute the following block:

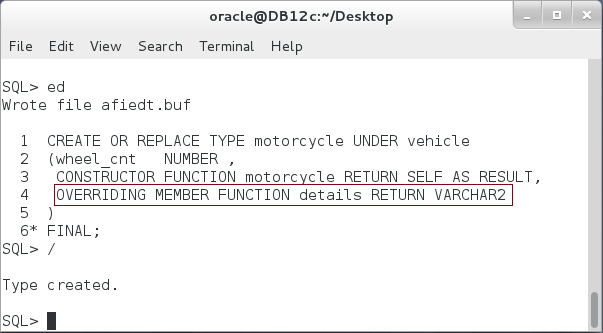
|  |  |
| --- | --- |
| Line |  |
| DECLARE |  |
| v\_mcycle motorcycle := NEW motorcycle(); |  |
| v\_car car := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); |  |
| v\_vehicle vehicle; |  |
| BEGIN |  |
| v\_vehicle := v\_mcycle; | Call the first "details" method |
| dbms\_output.put\_line(**v\_vehicle.details()**); |
| ---------------- |
| v\_vehicle := v\_car; | Call the second "details" method |
| dbms\_output.put\_line(**v\_vehicle.details(2)**); |
| ---------------- |
| END; |  |
| / |  |



## Overriding Methods

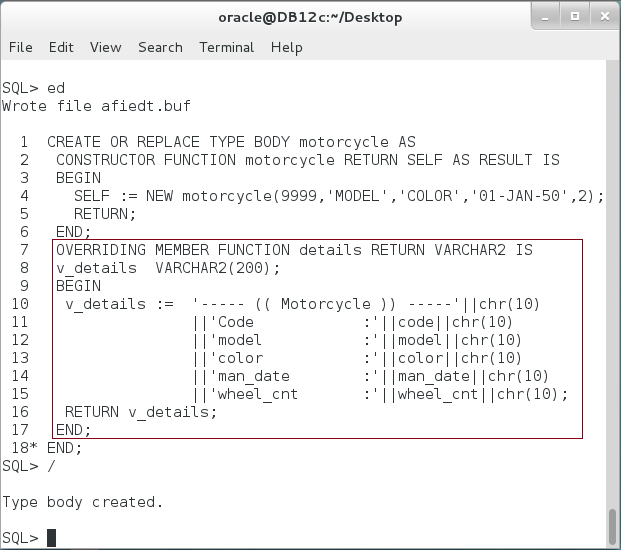
**Step 1:** Add "**details**" method in "**motorcycle**" object declaration, so it overrides its parent method as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE motorcycle UNDER vehicle |  |
| (wheel\_cnt NUMBER , |  |
| CONSTRUCTOR FUNCTION motorcycle RETURN SELF AS RESULT, |  |
| **OVERRIDING MEMBER FUNCTION details RETURN VARCHAR2** | Overrides "**details**" function |
| ) |
| FINAL; |
| / |



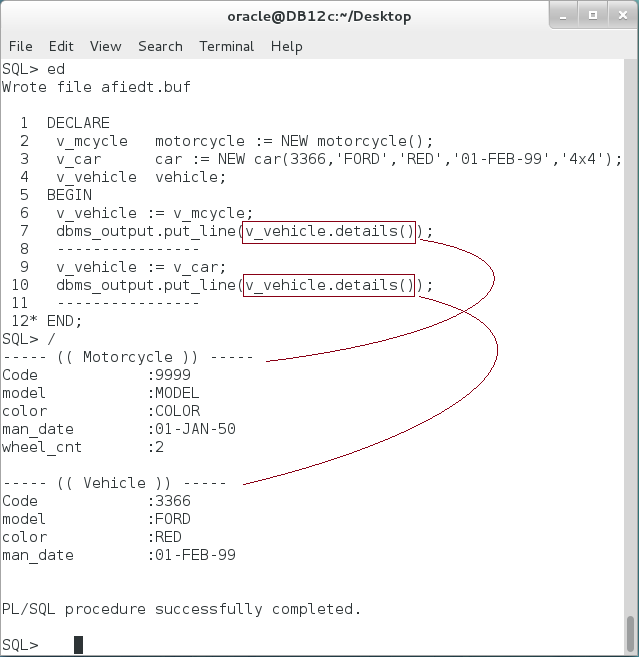
**Step 2:** Implement "details" function in the "motorcycle" Object Type as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY motorcycle AS |  |
| CONSTRUCTOR FUNCTION motorcycle RETURN SELF AS RESULT IS |  |
| BEGIN |  |
| SELF := NEW motorcycle(9999,'MODEL','COLOR','01-JAN-50',2); |  |
| RETURN; |  |
| END; |  |
| **OVERRIDING MEMBER FUNCTION details RETURN VARCHAR2 IS** | implement "details" function. |
| v\_details VARCHAR2(200); |
| BEGIN |
| v\_details := '----- **(( Motorcycle ))** -----'||chr(10) | More specific details of "motorcycle" object. |
| ||'Code :'||code||chr(10) |
| ||'model :'||model||chr(10) |
| ||'color :'||color||chr(10) |
| ||'man\_date :'||man\_date||chr(10) |
| ||'wheel\_cnt :'||wheel\_cnt||chr(10); |
| RETURN v\_details; |  |
| END; |  |
| END; |  |
| / |  |

****

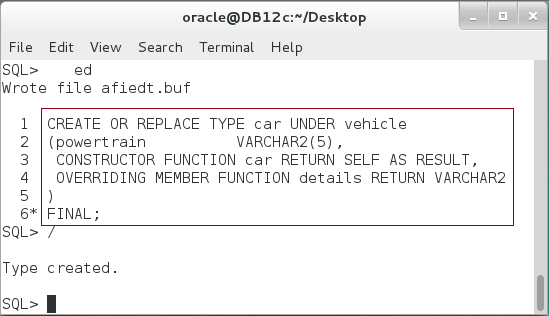
**Step 3:** Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_mcycle motorcycle := NEW motorcycle(); |  |
| v\_car car := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); |  |
| v\_vehicle vehicle; |  |
| BEGIN |  |
| v\_vehicle := v\_mcycle; | Motorcycle object refers to its "details" method |
| dbms\_output.put\_line(**v\_vehicle.details()**); |
| ---------------- |
| v\_vehicle := v\_car; | "car" object is still referring to its parent "details" method. |
| dbms\_output.put\_line(**v\_vehicle.details()**); |
| ---------------- |
| END; |
| / |  |

****

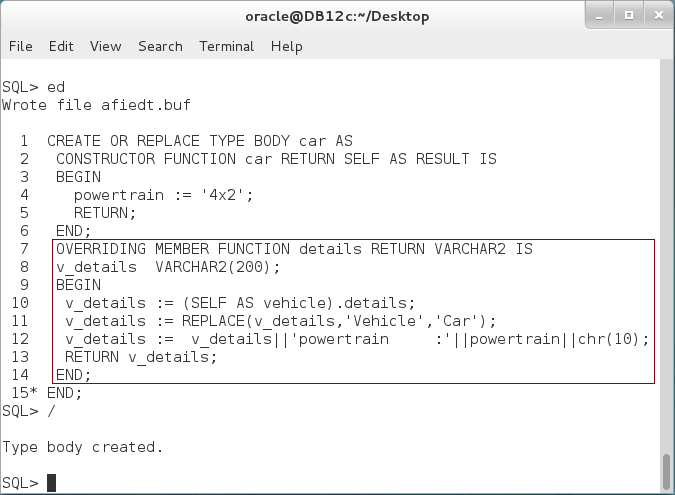
**Step 4:** Add overriding "details" method to "car" as shown below:

|  |  |
| --- | --- |
| Line | Description |
| **CREATE OR REPLACE TYPE car UNDER vehicle** |  |
| (powertrain VARCHAR2(5), |  |
| CONSTRUCTOR FUNCTION car RETURN SELF AS RESULT, |  |
| **OVERRIDING MEMBER FUNCTION details RETURN VARCHAR2** |  |
| ) |  |
| FINAL; |  |
| / |  |



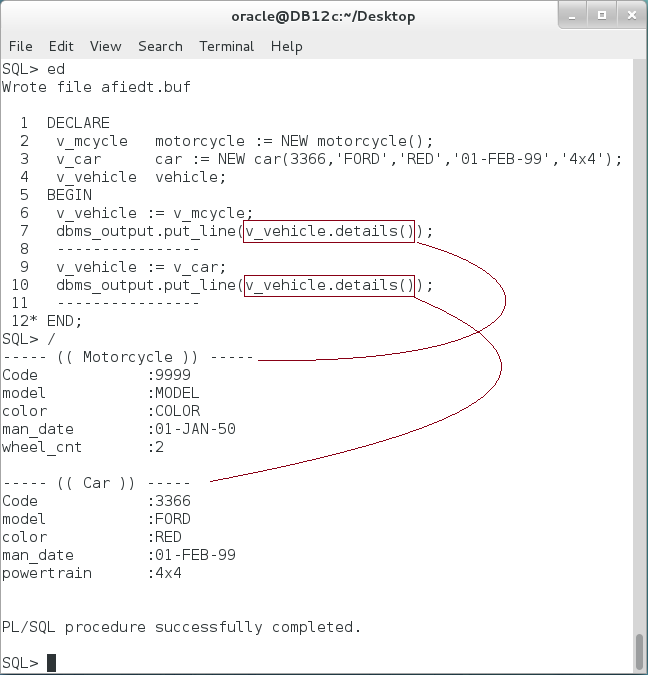
**Step 5:** Implement the "details" function of "car" object type as shown below:

|  |  |
| --- | --- |
| Line | Description |
| CREATE OR REPLACE TYPE BODY car AS |  |
| CONSTRUCTOR FUNCTION car RETURN SELF AS RESULT IS |  |
| BEGIN |  |
| powertrain := '4x2'; |  |
| RETURN; |  |
| END; |  |
| **OVERRIDING MEMBER FUNCTION details RETURN VARCHAR2 IS** | Implement "details" method in "car" object to override its parent method. You are still able to access the parent method using **"(SELF as parent)**" expression. |
| v\_details VARCHAR2(200); |
| BEGIN |
| v\_details := **(SELF AS vehicle)**.details; |
| v\_details := REPLACE(v\_details,'Vehicle','Car'); |
| v\_details := v\_details||'powertrain :'||powertrain||chr(10); |
| RETURN v\_details; |
| END; |
| END; |
| / |



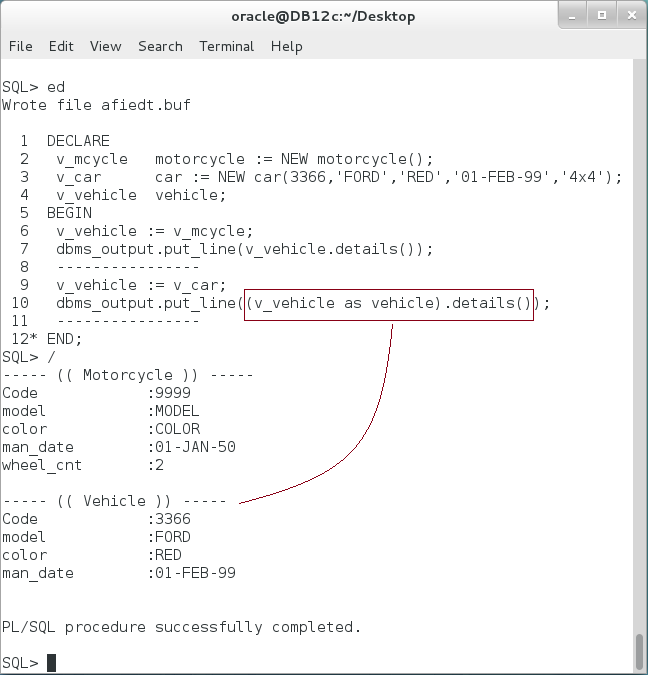
**Step 6:** Both, "motorcycle" and "car" details methods are now referring to their methods, which override their parent "details" method. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_mcycle motorcycle := NEW motorcycle(); |  |
| v\_car car := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); |  |
| v\_vehicle vehicle; |  |
| BEGIN |  |
| v\_vehicle := v\_mcycle; |  |
| dbms\_output.put\_line(**v\_vehicle.details()**); | Refers to "motorcycle" method. |
| ---------------- |
| v\_vehicle := v\_car; |
| dbms\_output.put\_line(**v\_vehicle.details()**); | Refers to "**car**" method. |
| ---------------- |
| END; |
| / |  |



**Step 7:** You are still able to access the parent method using **"(SELF AS parent)**" expression. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| v\_mcycle motorcycle := NEW motorcycle(); |  |
| v\_car car := NEW car(3366,'FORD','RED','01-FEB-99','4x4'); |  |
| v\_vehicle vehicle; |  |
| BEGIN |  |
| v\_vehicle := v\_mcycle; |  |
| dbms\_output.put\_line(v\_vehicle.details()); |  |
| ---------------- |  |
| v\_vehicle := v\_car; |  |
| dbms\_output.put\_line(**(v\_vehicle as vehicle)**.details()); | Refers to the parent "vehicle" method. |
| ---------------- |
| END; |
| / |



# SUMMARY

Object Type can represent the Object-Oriented Programming concepts of Inheritance, Polymorphism, Overloading and Overriding. Oracle supports single parent inheritance; the child object can only inherit its attributes and methods from one parent. According to Polymorphism concept, the parent object can handle and refer to its children objects. However, you are still able to refer and use the parent methods using "(SELF AS *parent*)" expression, CAST, and TREAT functions. Overloading methods enables you to define the same method in the same Object Type twice and more but with different parameter signature. Overriding methods enables you to define the same method with the same parameter signature in the children object to override its parent method.

After completing this lab exercise, you should be able to use inheritance, polymorphism, overloading, and overriding using Object Type.

# REFERENCES

* https://docs.oracle.com/database/121/ADOBJ/adobjint.htm#ADOBJ001
* http://www.java2s.com/Code/Oracle/Object-Oriented-Database/ALTERTYPEtoaddmoremethodsandattributes.htm
* https://docs.oracle.com/database/121/ADOBJ/adobjadv.htm#ADOBJ7410

# INDEX

inheritance 2, 4, 27

OOP 2

Overloading 2, 4, 15, 27

Overriding 2, 4, 19, 27

polymorphism 14, 27