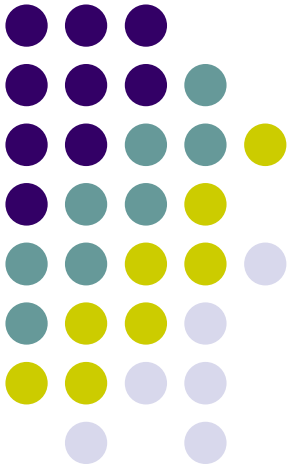
Database Systems

ORDB: Methods and Inheritance



Last Week

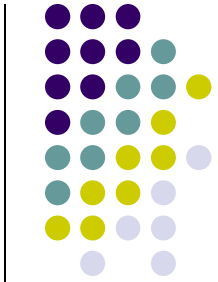
- Nested Collections
  - VARRAYs and Nested tables
    - Storing
    - Querying
    - Manipulating

- Any questions?

Encapsulation and UDTs: Methods

- Functions or procedures declared in an object type definition to implement behaviour of objects.
  - Declared in a CREATE TYPE statement and Defined in a CREATE TYPE BODY statement.
- Methods written in PL/SQL or Java are stored in the database.
  - preferable for data-intensive procedures and short procedures that are called frequently.
- Procedures in other languages, such as C, are stored externally.
  - preferable for computationally intensive procedures that are called less frequently.

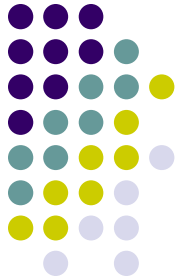
Member Method



- Define a member method in the object type for each operation an object of that type should perform.
- Example: Add a method priceInYen to MenuType.
- CREATE TYPE MenuType AS OBJECT (

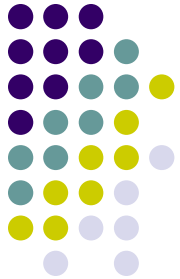
```
    bar REF BarType,
    beer REF BeerType,
    price FLOAT,
    MEMBER FUNCTION priceInYen(rate IN FLOAT)
      RETURN FLOAT
)
/
```

Example: Type Body



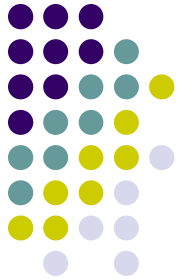
```
CREATE TYPE BODY MenuType AS
MEMBER FUNCTION
priceInYen(rate FLOAT)
RETURN FLOAT IS
   BEGIN
     RETURN rate * SELF.price;
   END;
END;
/
CREATE TABLE Sells OF MenuType;
```

Some Points to Remember



- SELF is a built-in parameter that denotes the object instance on which the method is currently being invoked.
- Member methods can reference the attributes and methods of SELF without using a qualifier.
  - The SELF bit in SELF.price is optional.
- Many methods will take no arguments.
  - In that case, do not use parentheses after the function name.
- The body can have any number of function definitions, separated by semicolons.
  - The body must include all the functions;

Adding a new method

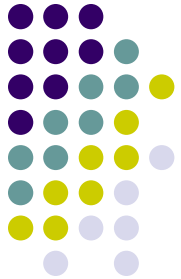


- Use ALTER TYPE to add a method:

```
ALTER TYPE MenuType
    ADD MEMBER
    FUNCTION
    priceInUSD(rate FLOAT)
    RETURN FLOAT
    CASCADE;
```

```
CREATE OR REPLACE TYPE BODY
    MenuType AS
    MEMBER FUNCTION
    priceInYen(rate FLOAT)
    RETURN FLOAT IS
        BEGIN
            RETURN rate * SELF.price;
        END priceInYen;
    MEMBER FUNCTION
    priceInUSD(rate FLOAT)
    RETURN FLOAT IS
        BEGIN
            RETURN rate * SELF.price;
        END priceInUSD;
END;
/
```

Example of Method Use

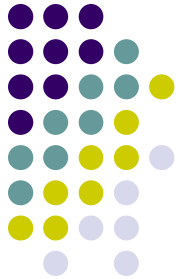


- Use an alias for the object followed by a dot, the name of the method, and argument(s) if any.
- EXAMPLE:

```
SELECT s.beer.name, s.priceInYen(106.0)
FROM Sells s
WHERE s.bar.name = 'Joe''s Bar';
```

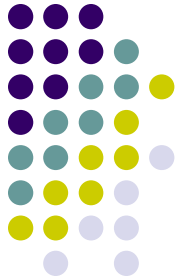
- Use parentheses, even if a method has no arguments.
  - E.g., select e.ename, e.age()
    from oremp e;
  - Assume age() is computed from attribute birthdate of the object type.

Object Comparison



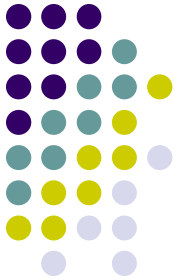
- The values of scalar data types such as CHAR or REAL have a predefined order.
- But, instances of an object type have no predefined order.
- To compare two items of a user-defined type, define an order relationship using a *map* or an *order* method.
  - At most one map method (or one order method) for an object type.

Map Methods



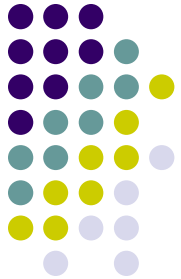
- Compare objects by mapping object instances to a scalar type.
  - DATE, NUMBER, VARCHAR, etc.
- Example: For an object type called RECTANGLE, the map method AREA can return its (HEIGHT * WIDTH) .
  - Then two rectangles can be compared by their areas.

Map Method



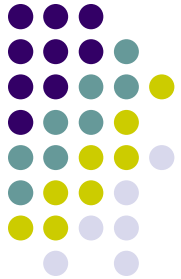
- A parameter-less member function that uses the MAP keyword.
- If an object type defines one, the method is called automatically to evaluate
  - comparisons such as obj_1 > obj_2 and
  - comparisons implied by the DISTINCT, GROUP BY, and ORDER BY clauses.

Example



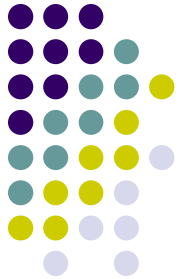
```
CREATE TYPE Rectangle_type AS OBJECT
( length NUMBER,
   width NUMBER,
  MAP MEMBER FUNCTION area RETURN NUMBER
);
CREATE TYPE BODY Rectangle_type AS MAP MEMBER
    FUNCTION area RETURN NUMBER IS
    BEGIN
      RETURN length * width;
   END area;
END;
```

Example



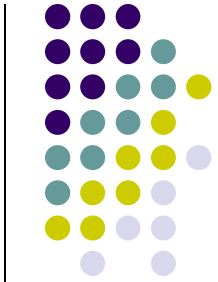
```
CREATE TABLE rectangles OF Rectangle_type;
INSERT INTO rectangles VALUES (1,2);
INSERT INTO rectangles VALUES (2,1);
INSERT INTO rectangles VALUES (2,2);
SELECT DISTINCT VALUE(r) FROM rectangles r;
   VALUE(R)(LEN, WID)
--------------------
  RECTANGLE_TYP(1, 2)
  RECTANGLE_TYP(2, 2)
```

Order Methods



- Order methods make direct object-to-object comparisons.
- A function with one declared parameter for another object of the same type.
- Definition of this method must return
  - < 0 if "self " is less than the argument object.
  - 0 if "self " is equal to the argument object.
  - > 0 if "self " is greater than the argument object.

Order Methods



- Called automatically whenever two objects need to be compared.

- Useful where comparison semantics may be too complex to use a map method.
  - E.g., to compare images, create an order method to compare by their brightness or number of pixels.

Example

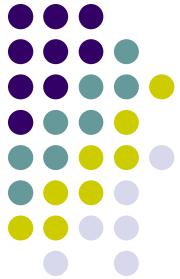
- An order method that compares customers by customer ID:

- CREATE TYPE Customer_typ AS OBJECT
  (id NUMBER,
    name VARCHAR2(20),
    addr VARCHAR2(30),
    ORDER MEMBER FUNCTION match (c Customer_typ) RETURN INTEGER);
  /

Example

```
CREATE TYPE BODY Customer_typ AS
ORDER MEMBER FUNCTION match (c Customer_typ)
RETURN INTEGER IS
BEGIN
        IF id < c.id THEN RETURN -1; -- any num <0
        ELSIF id > c.id THEN RETURN 1; -- any num >0
        ELSE RETURN 0;
        END IF;
END;
END;
 /
```

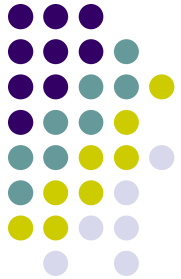
On Comparison Methods



- In defining an object type, you can specify either a map method or an order method for it, but not both.

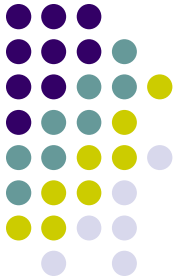
- If an object type has no comparison method, Oracle can compare two objects of that type only for equality or inequality.
  - Two objects of the same type count as equal only if the values of their corresponding attributes are equal.

On Comparison Methods



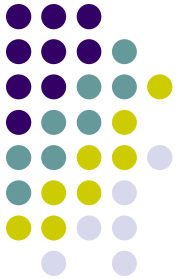
- When sorting or merging a large number of objects, use a map method.
  - One call maps all the objects into scalars, then sorts the scalars.
  - An order method is less efficient because it must be called repeatedly (it can compare only two objects at a time).

Methods on Nested Tables



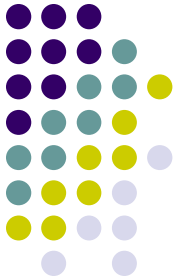
```
CREATE TYPE proj_t AS OBJECT (projno number,
    Projname varchar(15));
CREATE TYPE proj_list AS TABLE OF proj_t;
CREATE TYPE emp_t AS OBJECT
   ( eno number,
       projects proj_list,
     MEMBER FUNCTION projcnt RETURN INTEGER
   );
```

Methods on Nested Tables



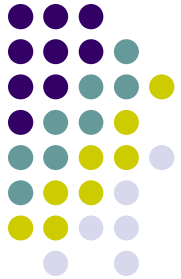
```
CREATE OR REPLACE TYPE BODY emp_t AS MEMBER
   FUNCTION projcnt RETURN INTEGER IS
      pcount INTEGER;
      BEGIN
        SELECT count(p.projno) INTO pcount
        FROM TABLE(self.projects) p;
        RETURN pcount;
      END;
   END;
   /
```

Methods on Nested Tables



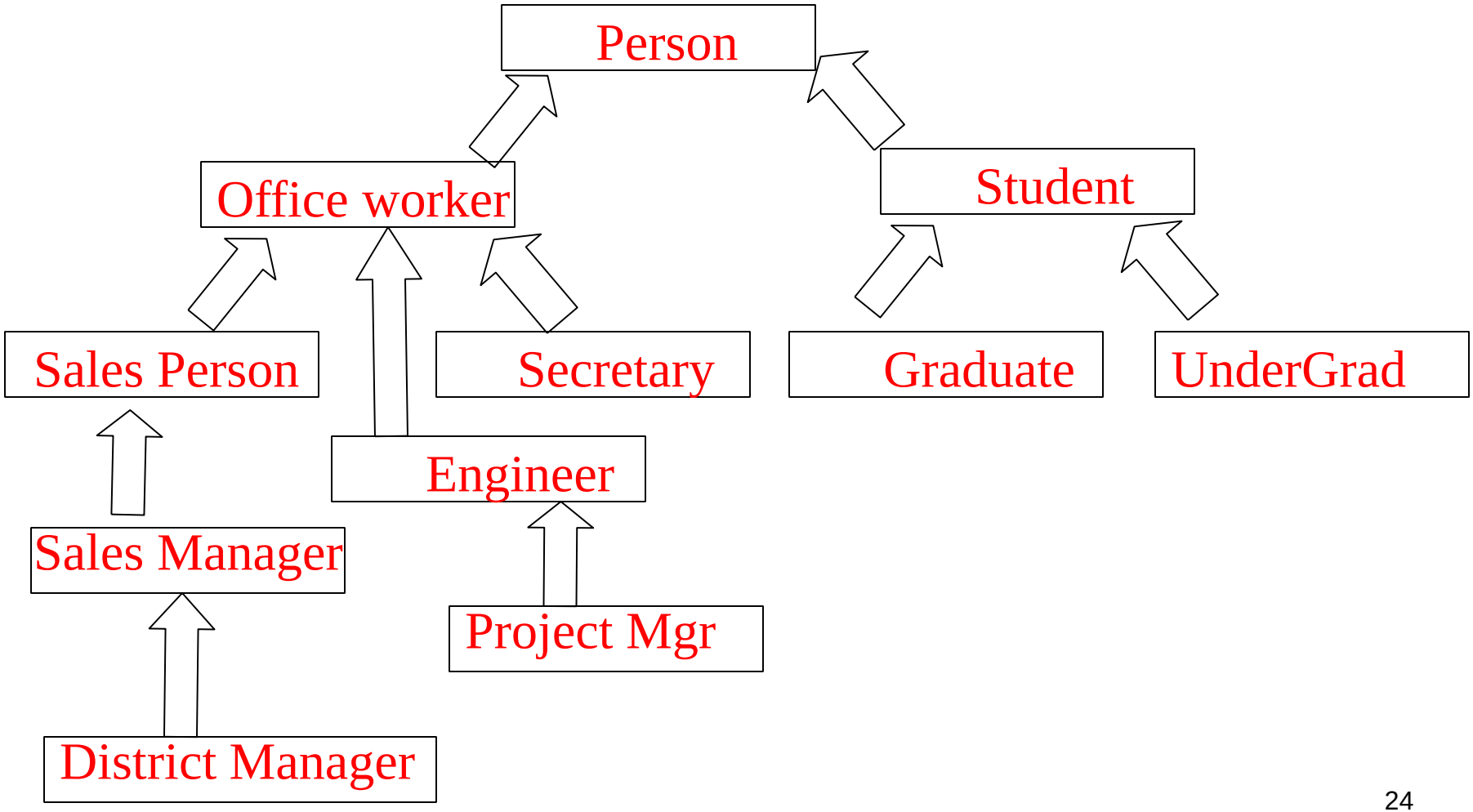
```
CREATE TABLE emptab OF emp_t
   (Eno PRIMARY KEY)
    NESTED TABLE projects STORE AS emp_proj_tab;

SELECT e.eno, e.projcnt() projcount
FROM emptab e;
```

Inheritance

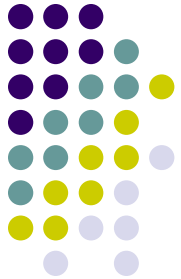


- A natural model for organising information.
  - e.g. captures the fact that sales managers are also salespeople.
- Methods and representation can be shared.
  - Reduces redundancy.
- New types and objects can be defined in existing hierarchies rather than from scratch.
  - Increases flexibility and extensibility.

Example: Person hierarchy

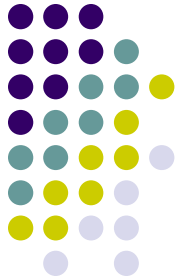


Inheritance in Oracle



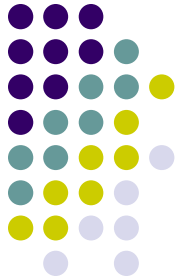
- It consists of a parent base type, or supertype, and one or more levels of child object types, or subtypes.
- Subtypes in a hierarchy are connected to their supertypes by inheritance.
  - subtypes automatically acquire the attributes and methods of their parent type.
  - any attribute or method updated in a supertype is automatically updated in subtypes also.

Specializing Subtypes



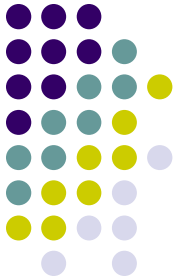
- Add new attributes the supertype does not have.
- A subtype cannot drop or change the type of an attribute it inherits from its parent.
- Add new methods that the parent does not have.
- Override the implementation of a parent method.

Specializing Subtypes



- Change the implementation of some methods a subtype inherits.
  - E.g., a shape object type might define a method calculate_area().
  - Two subtypes, rectangular and circular, might implement this method in a different way.

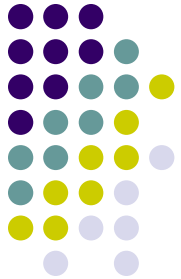
FINAL and NOT FINAL Types



- To permit subtypes, the object type must be defined as not final.
  - By including the keyword NOT FINAL in the type declaration.
  - By default, an object type is final.

- Example
  - CREATE TYPE Person_type AS OBJECT
    (pid NUMBER,
      name VARCHAR2(30),
      address VARCHAR2(100)) NOT FINAL;
  - Subtypes of Person_type can be defined.

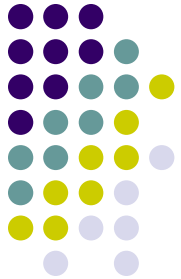
Altering object type



- You can change a final type to a not final type and vice versa with an ALTER TYPE statement.
  - If a NOT FINAL type has no current subtypes.

- For example,
  - ALTER TYPE Person_type FINAL;

Creating Subtypes

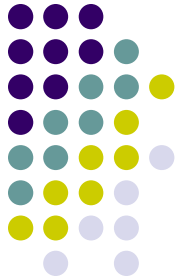


- Use a CREATE TYPE statement with an UNDER parameter to specify the parent type:

```
CREATE TYPE Student_type UNDER Person_type
   ( deptid NUMBER,
      major VARCHAR2(30)) NOT FINAL;
/
```

  - Student_type inherits all the attributes and methods declared in or inherited by Person_type.
- New attributes in a subtype must have different names from the attributes or methods in all its supertypes in the type hierarchy.

Multiple child types

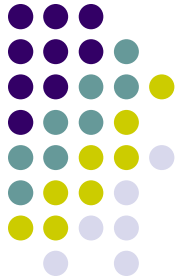


- A type can have multiple child subtypes, and these can also have subtypes.
- Example:

```
CREATE TYPE Employee_type UNDER Person_type
   ( empid NUMBER,
       mgr VARCHAR2(30)
    );
/
```

  - In addition to student_typ under person_type given earlier

Subtype under another subtype



- The new subtype inherits all the attributes and methods of its parent type, both declared and inherited.
- Example:

```
CREATE TYPE PartTimeStudent_type UNDER
    Student_type
 (numhours NUMBER);
```

Table of supertype

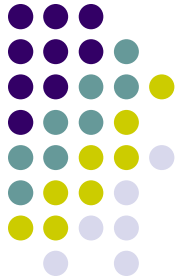
- Creating a supertype table

```
Create table person_tab of person_type
    (pid primary key);
```

- Inserting a subtype object/row

```
Insert into person_tab values
    ( student_type(4, 'Edward Learn',
          '65 Marina Blvd, Ocean Surf, WA, 6725',
          40, 'CS')
    );
```

Selecting all instances



- Using VALUE() function to select all instances of a super type:
  - Select all persons such as employees, students, etc. in the table:
- SELECT VALUE(p) FROM person_tab p;

```
VALUE(P)(PID, NAME, ADDRESS)
-------------------------------------------
Person_type(21937, 'Fred', '4 Ambrose Street')
Student_type(27362, `Peter', … , 21, 'Oragami')
PartTimeStudent_type(2134, 'Jack',…, 13, 'Physics',
  5)
Person_type(21362, 'Mary', …)
Student_type(18437, `Susan', … , 13, 'Maths')
PartTimeStudent_type(4318, 'Jill',…, 21, 'Pottery',
  2)
Person_type(39374, 'George', …)
```

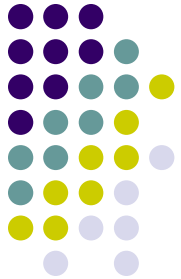
Selecting instances

- From student type and its subtypes

```
SELECT VALUE(s)
   FROM person_tab s
   WHERE VALUE(s) IS OF (Student_type);
```

```
VALUE(P)(PID, NAME, ADDRESS)
-----------------------------------------
Student_typ(27362, `Peter', … , 21, 'Oragami')
PartTimeStudent_type(2134,'Jack',…,13,'Physics',
   5)
Student_typ(18437, `Susan', … , 13, 'Maths')
PartTimeStudent_type(4318,'Jill',…,21,'Pottery',
   2)
```

Selecting instances



- From student type but not from subtypes

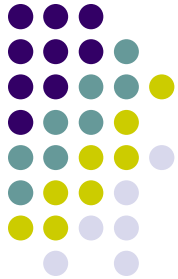
  SELECT VALUE(s)

  FROM person_tab s

  WHERE VALUE(s) IS OF (ONLY student_type);

```
VALUE(P)(PID, NAME, ADDRESS)
-----------------------------------------------
Student_typ(27362, `Peter', … , 21, 'Oragami')
Student_typ(18437, `Susan', … , 13, 'Maths')
```

Selecting a Subtype Attribute

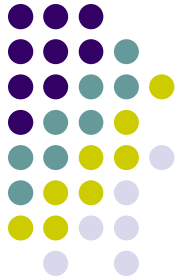


- TREAT() function to make the system treat each person as a part-time student to access the subtype attribute numhours:

```
SELECT Name, TREAT(VALUE(p) AS
PartTimeStudent_type).numhours hours
FROM person_tab p
WHERE VALUE(p) IS OF (ONLY PartTimeStudent_type);
```

```
NAME       hours
-------------------------------------------
Jack       5
Jill       2
```

NOT INSTANTIABLE Types



- Use this option with types intended solely as supertypes of specialized subtypes.
  - CREATE TYPE Address_typ AS OBJECT(...) NOT INSTANTIABLE NOT FINAL;*
  - CREATE TYPE AusAddress_typ UNDER Address_typ(...);
  - CREATE TYPE IntlAddress_typ UNDER Address_typ(...);

* You cannot create instances of the Address_typ only (similar to "*abstract classes*" in OO)

NOT INSTANTIABLE Methods

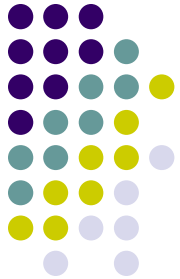
- Use this option to declare a method in a type without implementing it there.
  - A type that contains a non-instantiable method must itself be declared not instantiable.
  - CREATE TYPE T AS OBJECT (

    x NUMBER,

    NOT INSTANTIABLE MEMBER FUNCTION func1()*
    RETURN NUMBER) NOT INSTANTIABLE NOT FINAL;

* The type body for T does not contain a definition for func1

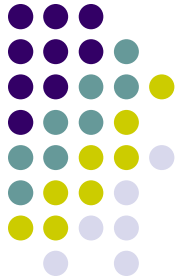
NOT INSTANTIABLE Methods



- Define a method as non-instantiable if every subtype is to override the method in a different way.

- If a subtype does not implement every inherited non-instantiable method, the subtype must be declared not instantiable.
  - A non-instantiable subtype can be defined under an instantiable supertype.

FINAL and NOT FINAL Methods

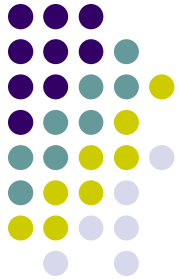


- If a method is declared to be final, subtypes cannot override it by providing their own implementation.
  - Unlike types, methods are not final by default.
  - They must be explicitly declared to be final.
- An overriding method is specified in a CREATE TYPE BODY statement.

Example

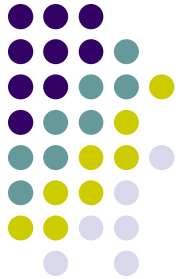
```
CREATE TYPE MyType AS OBJECT
    (  ...,
       MEMBER PROCEDURE Print,
       FINAL MEMBER FUNCTION foo(x NUMBER) …, ...
    )   NOT FINAL;
/
CREATE TYPE MySubType UNDER MyType
    ( ...,
      OVERRIDING MEMBER PROCEDURE Print,
       ...);
/
```

Overloading Methods



- A subtype can add new methods that have the same names as methods it inherits.
  - Methods that have the same name but different signatures in a type are called overloads.
  - The compiler uses the methods' signatures to tell them apart.

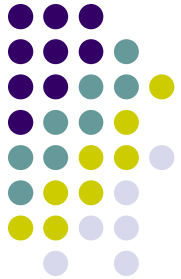
Example: Overloading Methods



```
CREATE TYPE MyType AS OBJECT
  ( ...,
    MEMBER FUNCTION fun(x NUMBER)…,
    ...) NOT FINAL;
/
CREATE TYPE MySubType UNDER MyType
  ( ...,
     MEMBER FUNCTION fun(x DATE) …,
    ...);
/
```

- Same function name, different signature

Summary



- Nested Collections
  - Varrays and nested tables
    - Storing
    - Querying
    - Manipulating
- Inheritance in Oracle
  - FINAL/NOT FINAL
  - Subtypes (UNDER)
  - Getting at particular subtypes
  - INSTANTIABLE/NOT INSTANTIABLE (Types and methods)
  - Overriding & Overloading