ODB and the ODMG

(Object Data Management Group)

Object (Oriented) Databases (ODB)

The ODMG standard:

Object model

ODL

OQL

Language-bindings

ODMG: The Object Model

- Object:
  - Objects have identity (OID) and state, OID never changes, state may change over time
  - Objects may have names (entry points to the Database)
  - Objects have a lifetime (either persistent or transient

ODMG: The Object Model

- Object:
  - Objects have structure (*atomic* or *Collection*)
  - The structure specifies how to create the object using a type constructor
    - Atomic objects are any objects that are not collections
    - They may be composite (have internal structure)
    - They are not tuples in the relational sense of the word tuple

Literals

- Have state, but no OID, state is never changed (”constant objects”)
- State may be atomic or complex:
  - Atomic literals are simple, predefined types
  - Structured literals (roughly records or structs)
  - Collection literals (collections like in the Java or .NET API, but persistent)

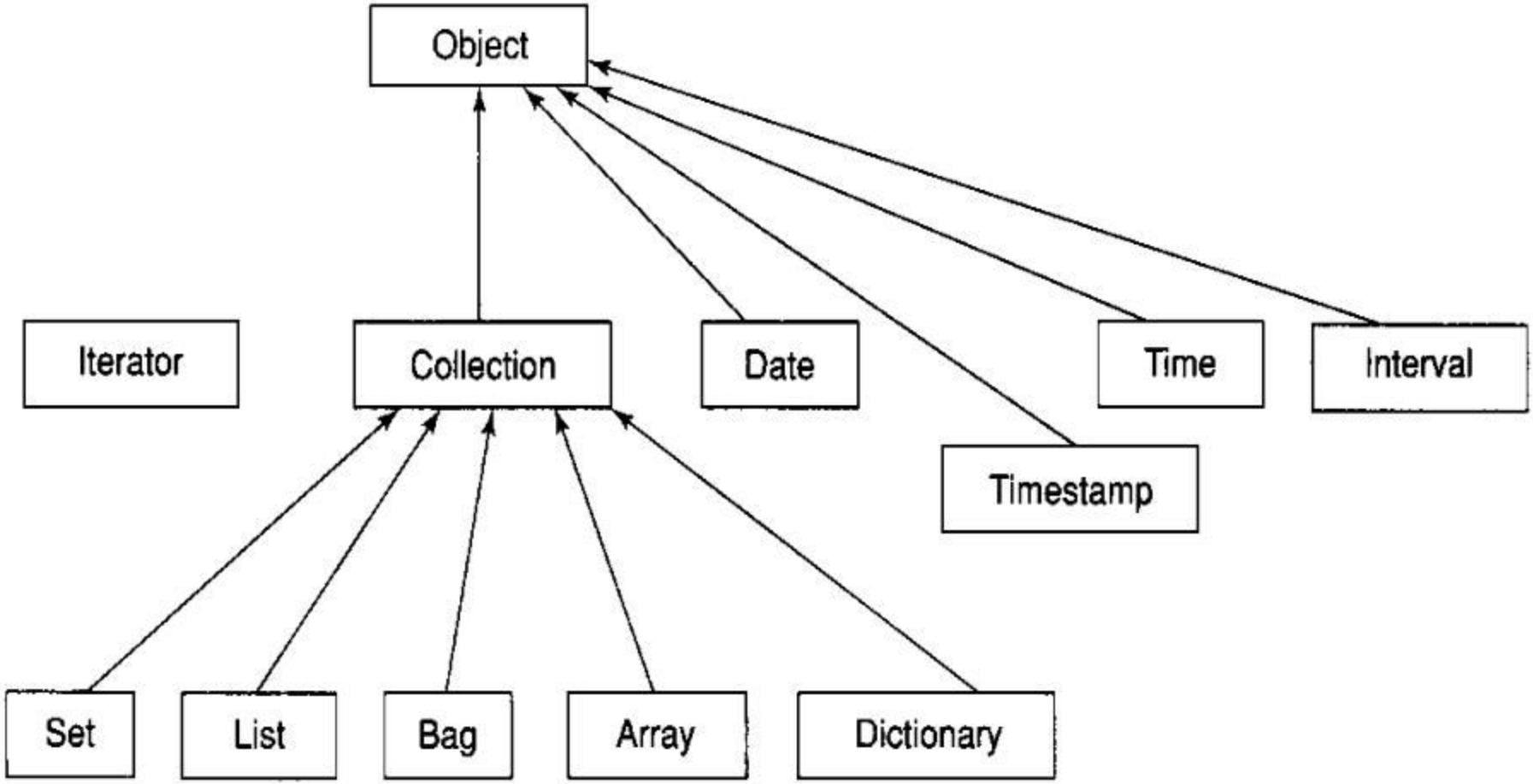


FIGURE 21.2 Inheritance hierarchy for the built-in interfaces of the object model.

```
interface Object {
   ...
   boolean      same_as(in Object other_object);
   Object       copy();
   void         delete();
};
```

FIGURE 21.1A Overview of the interface definitions for part of the ODMG object model. The basic `Object` interface, inherited by all objects.

```
interface Date : Object {
        enum            Weekday
        {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
        enum            Month
        {January, February, March, April, May, June, July, August, September, October, November, December};
        unsigned short  year();
        unsigned short  month();
        unsigned short  day();
        ...
        boolean         is_equal(in Date other_Date);
        boolean         is_greater(in Date other_Date);
        ...
};

interface Time : Object {
        ...
        unsigned short  hour();
        unsigned short  minute();
        unsigned short  second();
        unsigned short  millisecond();
        ...
        boolean         is_equal(in Time other_Time);
        boolean         is_greater(in Time other_Time);
        ...
        Time            add_interval(in Interval some_Interval);
        Time            subtract_interval(in Interval some_Interval);
        Interval        subtract_time(in Time other_Time);
};
```

```
interface Timestamp : Object {
    ...
    unsigned short    year();
    unsigned short    month();
    unsigned short    day();
    unsigned short    hour();
    unsigned short    minute();
    unsigned short    second();
    unsigned short    millisecond();
    ...
    Timestamp         plus(in Interval some_Interval);
    Timestamp         minus(in Interval some_Interval);
    boolean           is_equal(in Timestamp other_Timestamp);
    boolean           is_greater(in Timestamp other_Timestamp);
    ...
};

interface Interval : Object {
    unsigned short    day();
    unsigned short    hour();
    unsigned short    minute();
    unsigned short    second();
    unsigned short    millisecond();
    ...
    Interval          plus(in Interval some_Interval);
    Interval          minus(in Interval some_Interval);
    Interval          product(in long some_value);
    Interval          quotient(in long some_value);
    boolean           is_equal(in Interval other_Interval);
    boolean           is_greater(in Interval other_Interval);
    ...
};
```

```
interface Collection : Object {
    ...
    exception           ElementNotFound{any element; };
    unsigned long       cardinality();
    boolean             is_empty();
    ...
    boolean             contains_element(in any element);
    void                insert_element(in any element);
    void                remove_element(in any element)
                                raises(ElementNotFound);
    Iterator            create_iterator(in boolean stable);
    ...
};

interface Iterator {
    exception           NoMoreElements();
    ...
    boolean             is_stable();
    boolean             at_end();
    void                reset();
    any                 get_element() raises(NoMoreElements);
    void                next_position() raises(NoMoreElements);
    ...
};

interface Set : Collection {
    Set                 create_union(in Set other_set);
    ...
    boolean             is_subset_of(in Set other_set);
    ...
};
```

```
interface Bag : Collection {
    unsigned long    occurrences_of(in any element);
    Bag              create_union(in Bag other_bag);
    ...
};

interface List : Collection {
    exception        Invalid_Index{unsigned_long index; };
    any              remove_element_at(in unsigned long position)
                         raises(InvalidIndex);
    any              retrieve_element_at(in unsigned long position)
                         raises(InvalidIndex);
    void             replace_element_at(in any element, in unsigned long position)
                         raises(InvalidIndex);
    void             insert_element_after(in any element, in unsigned long position)
                         raises(InvalidIndex);
    ...
    void             insert_element_first(in any element);
    ...
    any              remove_first_element() raises(InvalidIndex);
    ...
    any              retrieve_first_element() raises(InvalidIndex);
    ...
    List             concat(in List other_list);
    void             append(in List other_list);
};
```

```
interface Array : Collection {
   exception        Invalid_Index{unsigned_long index; };
   any              remove_element_at(in unsigned long index)
                        raises(InvalidIndex);
   any              retrieve_element_at(in unsigned long index)
                        raises(InvalidIndex);
   void             replace_element_at(in unsigned long index, in any element)
                        raises(InvalidIndex);
   void             resize(in unsigned long new_size);
};

struct Association {any key; any value; };

interface Dictionary : Collection {
   exception        KeyNotFound{any key; };
   void             bind(in any key, in any value);
   void             unbind(in any key) raises(KeyNotFound);
   any              lookup(in any key) raises(KeyNotFound);
   boolean          contains_key(in any key);
};
```

Atomic (domain) Objects

- Called atomic, but are often composite (structured)
- Corresponds to entities in a conceptual model
- Objects have:
  - Properties (state):
    - Attributes
    - Relationships
  - Operations (behaviour)

```
class Employee
(   extent  all_employees
    key     ssn  )
{
    attribute       string                      name;
    attribute       string                      ssn;
    attribute       date                        birthdate;
    attribute       enum Gender{M, F}           sex;
    attribute       short                       age;
    relationship    Department                  works_for
                        inverse Department::has_emps;
    void            reassign_emp(in string new_dname)
                        raises(dname_not_valid);
};


class Department
(   extent  all_departments
    key     dname, dnumber  )
{
    attribute       string              dname;
    attribute       short               dnumber;
    attribute       struct Dept_Mgr {Employee manager, date startdate}
                                        mgr;
    attribute       set<string>         locations;
    attribute       struct Projs {string projname, time weekly_hours}
                                        projs;
    relationship    set<Employee>       has_emps inverse Employee::works_for;
    void            add_emp(in string new_ename) raises(ename_not_valid);
    void            change_manager(in string new_mgr_name; in date startdate);
};
```

OBS!
Inverse =>
automatisk integritet

FIGURE 21.3 The attributes, relationships, and operations in a class definition.

Inheritance

- Between interfaces:
  - only (abstract) behaviour (specification)
  - Any properties defined in an interface are not inherited
  - “keyword”: ”**:**”
- Between classes:
  - both state and behaviour
  - Keyword: “**extends**
- Like Java or C#

The Extension of a Class (**extent**)

- All objects belonging to a class is automatically stored persistently in the extension of the class
- The extension of a class is the set of all objects of that class
- The extension is declared using the keyword **extent** (not exten**ds**, which specifies inheritance)
- The extension of some class *T* is an object of type *Set<T>*

Inheritance and Class Extension

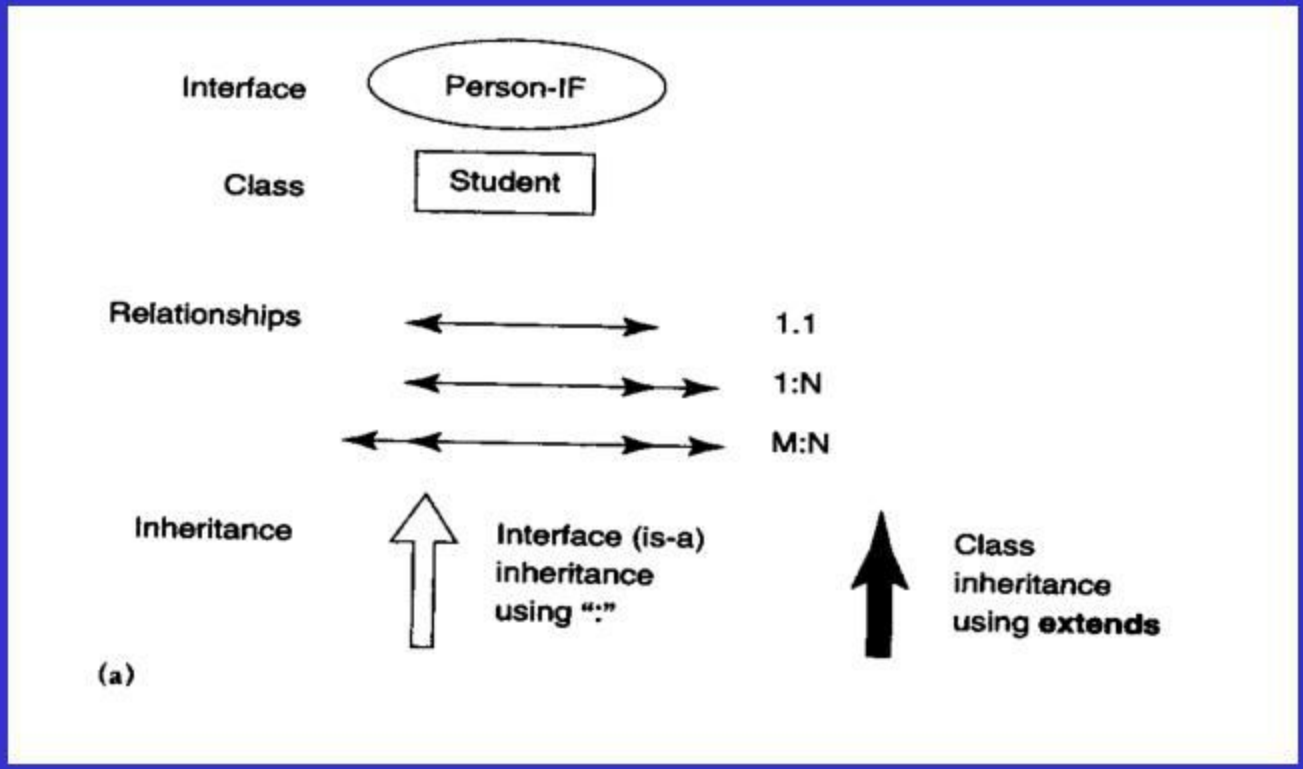
- **class** B **extends** A

Means (among other things) that

- The extension of B is a subset of the extension of A

ODL: *Object Definition Language*

- Graphical notation:



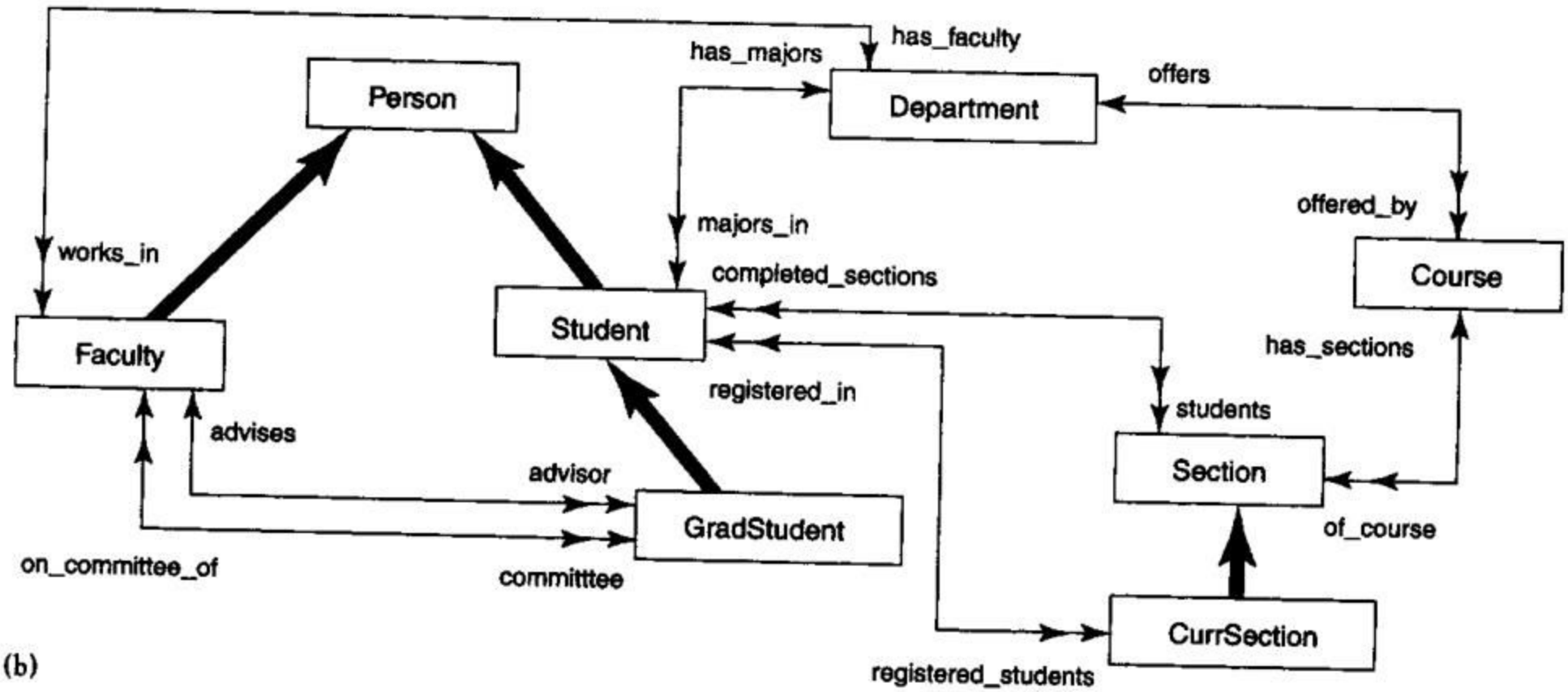


FIGURE 21.5 An example of a database schema. (a) Graphical notation for representing ODL schemas. (b) A graphical object database schema for part of the UNIVERSITY database.

```
class Person
(   extent persons
    key     ssn  )
{
    attribute     struct Pname {string fname, string mname, string lname }
                                          name;
    attribute     string                  ssn;
    attribute     date                    birthdate;
    attribute     enum Gender{M, F}       sex;
    attribute     struct Address
                  {short no, string street, short aptno, string city, string state, short zip }
                                          address;
    short    age();
};

class Faculty extends Person
(   extent faculty  )
{
    attribute     string          rank;
    attribute     float           salary;
    attribute     string          office;
    attribute     string          phone;
    relationship Department       works_in inverse Department::has_faculty;
    relationship set<GradStudent> advises inverse GradStudent::advisor;
    relationship set<GradStudent> on_committee_of
                                  inverse GradStudent::committee;
    void     give_raise(in float raise);
    void     promote(in string new_rank);
};
```

Note!
extends
extent

```
class Grade
(   extent grades   )
{
    attribute      enum GradeValues{A,B,C,D,F,I,P}
                                        grade;
    relationship Section section inverse Section::students;
    relationship Student student inverse Student::completed_sections;
};

class Student extends Person
(   extent students   )
{
    attribute      string               class;
    attribute      Department           minors_in;
    relationship Department majors_in inverse Department::has_majors;
    relationship set<Grade> completed_sections inverse Grade::student;
    relationship set<CurrSection> registered_in
                                        inverse CurrSection::registered_students;
    void    change_major(in string dname) raises(dname_not_valid);
    float   gpa();
    void    register(in short secno) raises(section_not_valid);
    void    assign_grade(in short secno; in GradeValue grade)
                                        raises(section_not_valid,grade_not_valid);
```

```
class Degree
{
    attribute     string              college;
    attribute     string              degree;
    attribute     string              year;
};

class GradStudent extends Student
(   extent grad_students   )
{
    attribute     set<Degree>     degrees;
    relationship Faculty advisor inverse Faculty::advises;
    relationship set<Faculty> committee inverse Faculty::on_committee_of;
    void    assign_advisor(in string lname; in string fname)
                                        raises(faculty_not_valid);
    void    assign_committee_member(in string lname; in string fname)
                                        raises(faculty_not_valid);
};

class Department
(   extent departments key dname   )
{
    attribute     string              dname;
    attribute     string              dphone;
    attribute     string              doffice;
    attribute     string              college;
    attribute     Faculty             chair;
    relationship set<Faculty> has_faculty inverse Faculty::works_in;
    relationship set<Student> has_majors inverse Student::majors_in;
    relationship set<Course> offers inverse Course::offered_by;
};
```

Inheritance between Interfaces

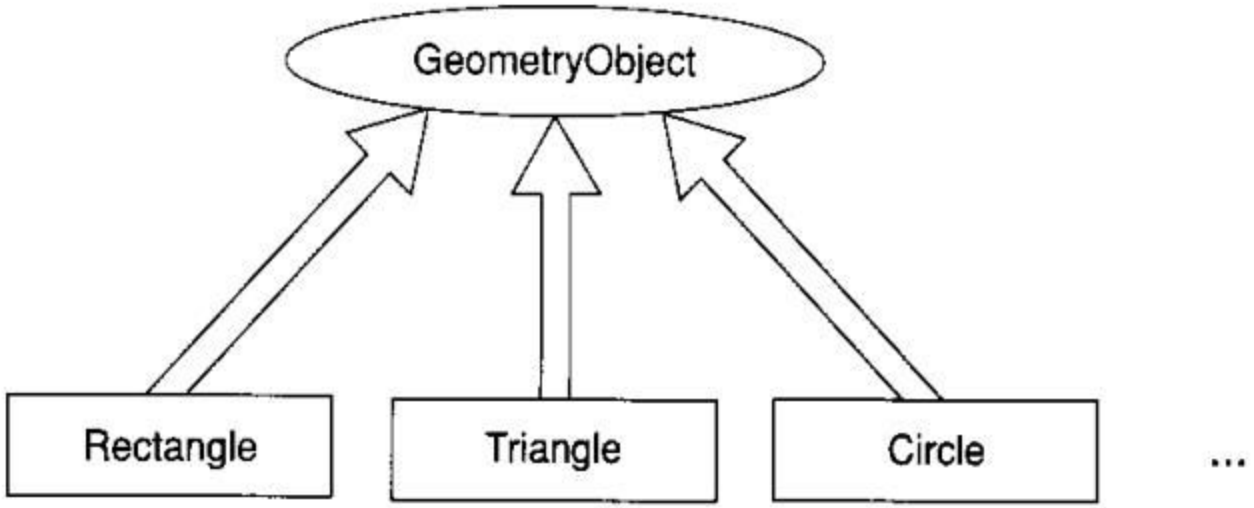


FIGURE 21.7A An illustration of interface inheritance via ":". Graphical schema representation.

```
interface GeometryObject
{
    attribute       enum Shape{Rectangle,Triangle,Circle,...} shape;
    attribute       struct Point {short x, short y}       reference_point;
    float     perimeter();
    float     area();
    void      translate(in short x_translation; in short y_translation);
    void      rotate(in float angle_of_rotation);
};

class Rectangle : GeometryObject
(   extent  rectangles      )
{
    attribute       struct Point {short x, short y}       reference_point;
    attribute       short                 length;
    attribute       short                 height;
    attribute       float                 orientation_angle;
};

class Triangle : GeometryObject
(   extent  triangles   )
{
    attribute       struct Point {short x, short y}       reference_point;
    attribute       short                 side_1;
    attribute       short                 side_2;
    attribute       float                 side1_side2_angle;
    attribute       float                 side1_orientation_angle;
};

class Circle : GeometryObject
(   extent  circles     )
{
    attribute       struct Point {short x, short y}       reference_point;
    attribute       short                 radius;
};

...
```

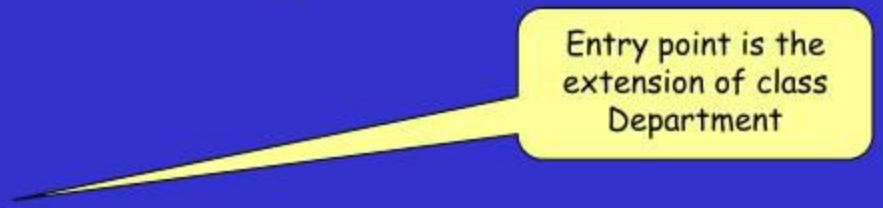
Only operations are inherited

Must be repeated

OQL: *Object Query Language*

- Based on SQL
  - Non-procedural
  - Supports object navigation using references
  - Embedded or stand-alone
- For instance departments in an Engineering College:

```
SELECT   d.dname
FROM     d IN departments
WHERE    d.college = 'Engineering'
```



- Returns a *Bag* of *dname*, SELECT DISTINCT returns a *Set*

Path-Expressions

- departments;
  - //returns a reference to the extension of Department
- csdepartment;
  - //a reference to the CS-dept-object
- csdepartment.chair;
  - //a *path* to the *chair* object of the CS-dept-object (type *Faculty*)
- csdepartment.chair.rank;
  - //the *rank* attribute of the *chair* -object of the CS-dept-object
- csdepartment.has_faculty;
    - //returns Set<Faculty>
  - etc. etc.

Java-binding - FastObjects

```
Database db =  new Database();
Transaction txn = new Transaction();
db.open("addressDB",Database.OPEN_READ_WRITE);
txn.begin();
OQLQuery query = new OQLQuery(
      "select x from Person x where x.name = \"Doug Barry\"");
Collection result = (Collection) query.execute();
Iterator iter = result.iterator();
while ( iter.hasNext() ){
      Person person = (Person) iter.next();
      person.address.street = "13504 4th Avenue South";
 }
txn.commit();
db.close();
. . .
```

```
try{
  MyClass myObject = new MyClass();
  db.bind(myObject, "myName"); //root
 }
 catch (ObjectNameNotUniqueException exc){
  txn.abort();
  System.out.println( exc);
 }
 try{
  MyClass myObject = (MyClass)db.lookup("myName");
  System.out.println(myObject);
 }
 catch(ObjectNameNotFoundException exc){
  System.out.println(exc);
  txn.abort();
 }
```

Using Extension

```
//...
db.makePersistent( new MyClass() );
//...
// print all stored instances of MyClass
Extent myClasses = new Extent(MyClass.class);
while ( myClasses.hasNext() ) {
    System.out.println( myClasses.next() );
}
//...
```

Conclusion:

OOPL

+ embedded ”SQL”

+ persistent Collection

==

ODB

More on: http://www.versant.com/

Object Query Language OQL

CIS 764

11/15/2007

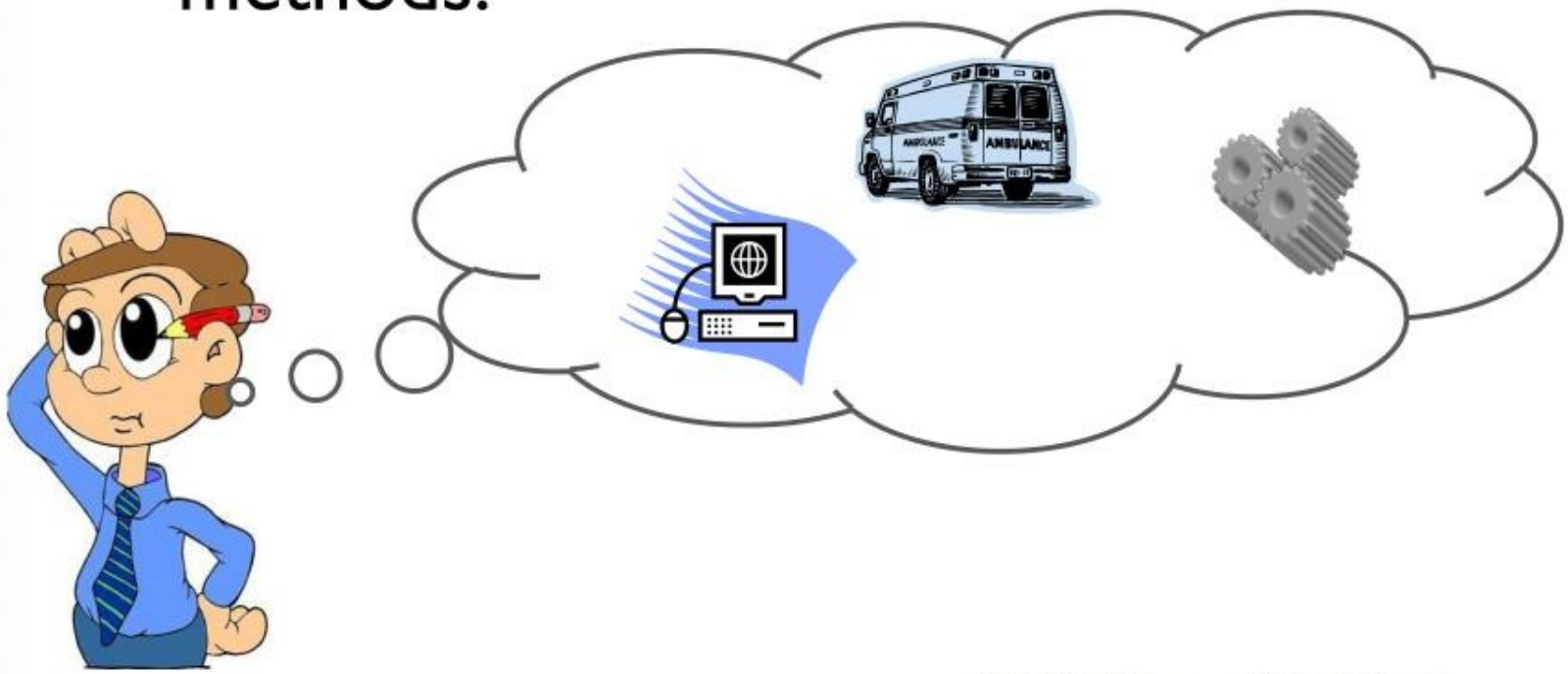
Waleed Aljandal

Content

- Introduction about OQL.
- The different between SQL & OQL output.
- Why we use OQL.
- introduction about EJB QL
- EJB QL syntax.
- Some examples.

What is OQL?

- OQL is a powerful and easy-to-use SQL-like query language with special features dealing with complex objects, values and methods.



Query result for OQL & SQL

Example:

Product no	Name	Color
P1	Ford Mustang	Black
P2	Toyota Celica	Green
P3	Mercedes SLK	Black

The following is a sample query

"what are the names of the black product?"

Select distinct p.name

From products p

Where p.color = "black"

⇒Valid in both SQL and OQL, but the results are different.

Result of the query (SQL)

Original table

Product no	Name	Color
P1	Ford Mustang	Black
P2	Toyota Celica	Green
P3	Mercedes SLK	Black

Result

Name
Ford Mustang
Mercedes SLK

=> Returns a table with rows.

Result of the query (OQL)

Original table

Product no	Name	Color
P1	Ford Mustang	Black
P2	Toyota Celica	Green
P3	Mercedes SLK	Black

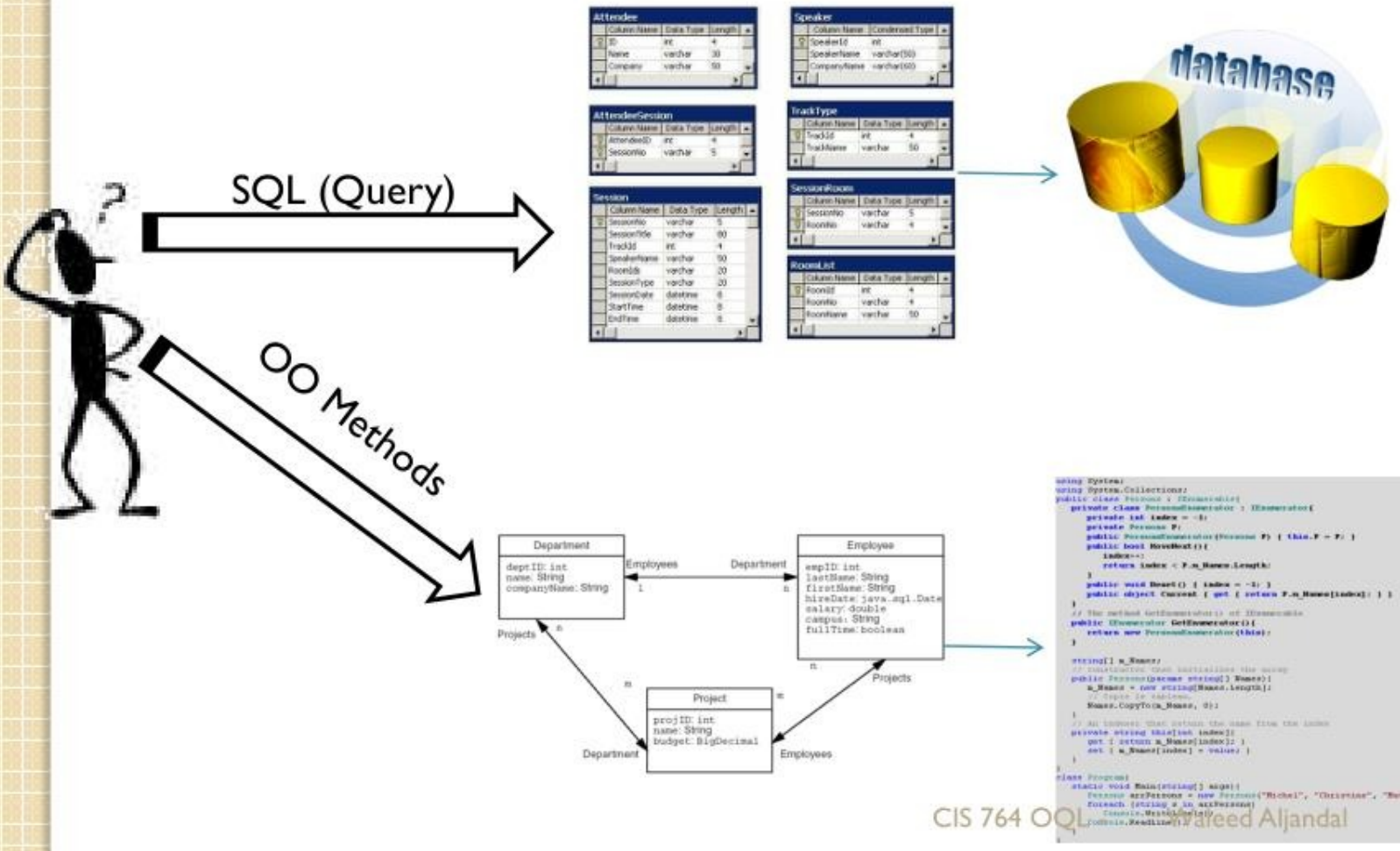
Result

String	String
Ford Mustang	Mercedes SLK

⇒Returns a collection of objects.

Why we use OQL?

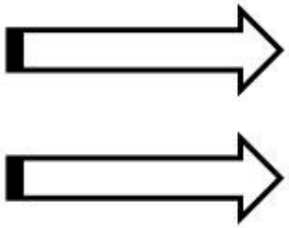
- The Object Oriented programmer

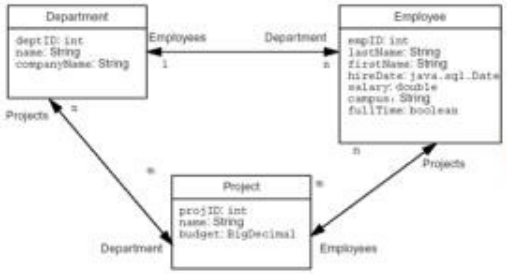


Why we use OQL?

- The Object Oriented programmer

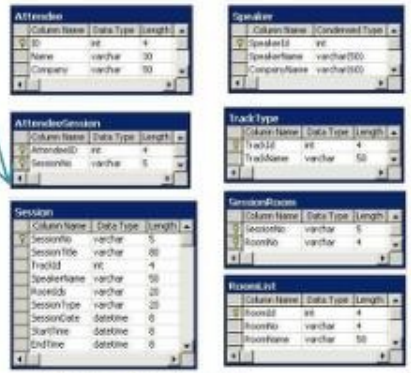
OQL (Query)





OO Methods

Mapping to relation DB



You don't need to know the database structure (tables, fields)

EJB QL

- First appeared in EJB 2.0
- Enhancement in EJB 3.0
  - Single and multiple value result types
  - Aggregate functions, with sorting and grouping Clauses
  - More natural join syntax, support both inner and outer joins
  - Conditional expressions involving subqueries

EJB QL

- *Syntax*

SELECT expr
FROM schema [**AS** var, **IN**(path) **AS** var1, ...]
[**WHERE** expr]
[**ORDER BY** expr]

[**OFFSET** integer] → Returns items starting from the offset value
[**LIMIT** integer] → Limits the number of items returned

EJB QL

- *Syntax*
  - *Some Expressions*
    - **IS [NOT] NULL --- IS [NOT] EMPTY**
    - **IN** (collection_valued_path_expression)
    - **Function_name(expr, ...)**

Function	Description
CONCAT(string, string)	Contatenates two strings
SUBSTRING(string, start, len)	Selects a substring
LOCATE(string, start [, start])	Finds a substring
LENGTH(string)	Returns the string length
ABS(number)	Returns the absolute value
SQRT(double)	Returns the square root of a number

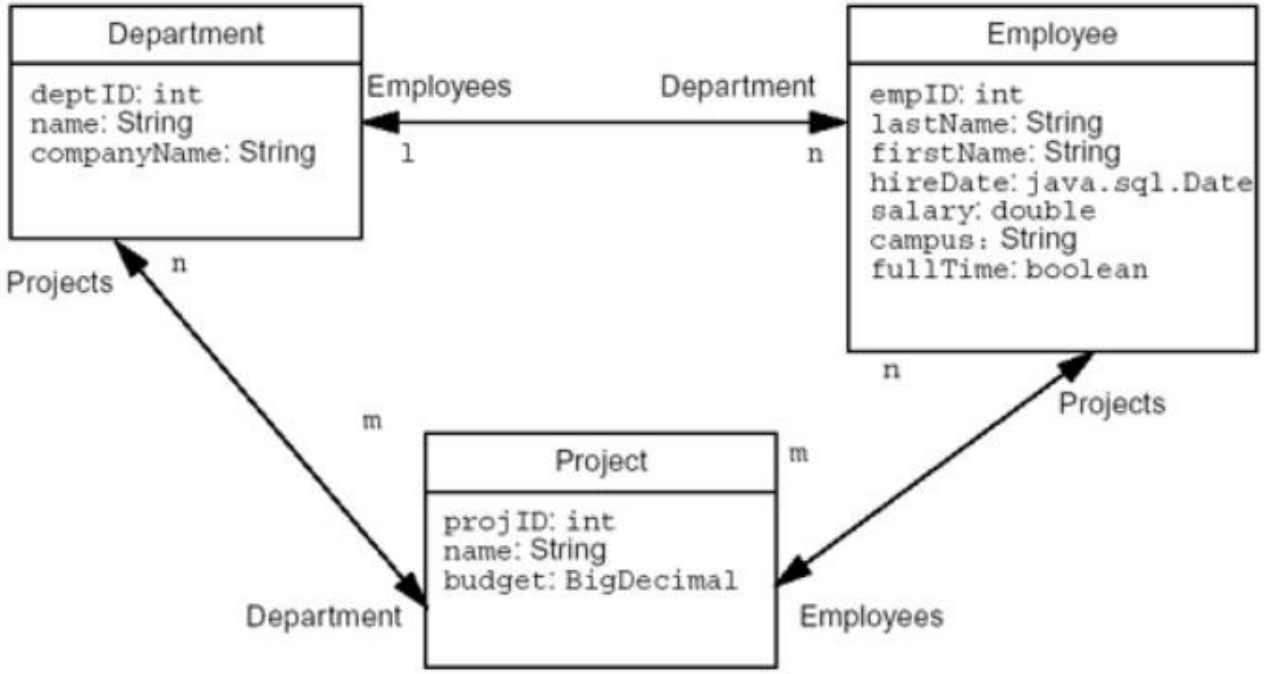
EJB QL vs. SQL

- *Syntax:*

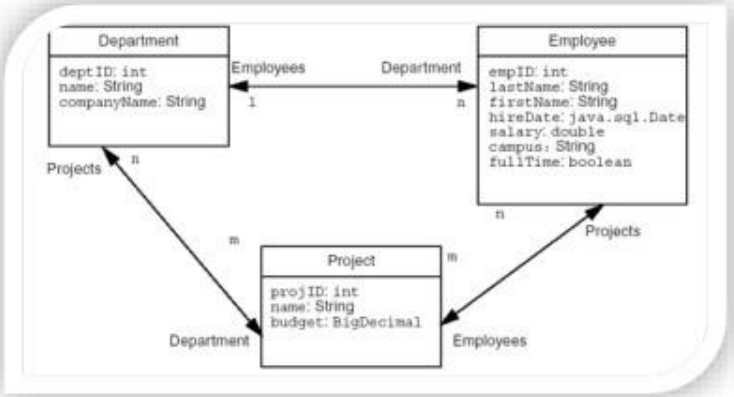
EJB QL	SQL
`SELECT OBJECT (e) FROM Employee e`	`SELECT * FROM EMPLOYEE`
`SELECT OBJECT(si) FROM StockItem AS si WHERE si.price BETWEEN 10 and 100`	`SELECT * FROM STOCKITEM WERE STOCKITEM.PRICE >= 10 AND STOCKTIEM.PRICE <=100`
`SELECT DISTINCT w.lastname FROM Winners w WHERE w.lastname LIKE 'Mac%'`	`SELECT DISTINCT LASTNAME FROM WINNERS WHERE WINNERS.LASTNAME LIKE 'Mac%'`

Learn by example:

Case Study



Learn by example:



Queries :

- Find all employee in "Customer Support" department.

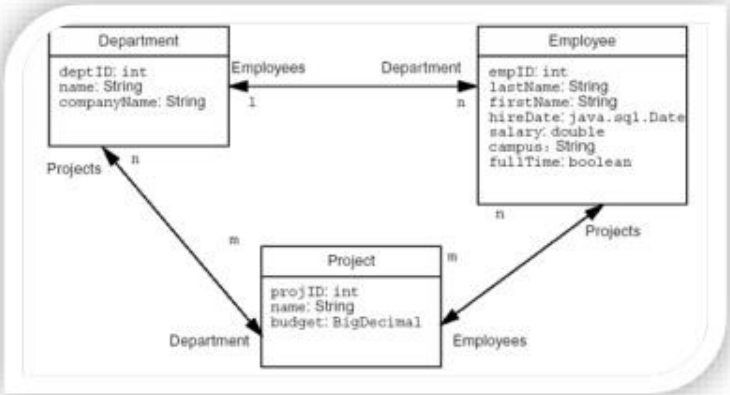
```
SELECT d.employees FROM Department d WHERE d.name = 'Customer Support'
```

— *For dynamic query we use parameter reference*

```
SELECT d.employees FROM Department d WHERE d.name = ?1
```

?1 → Related to the first method parameter.

Learn by example:



Queries :

- Find the names of the employees who are not assigned to a campus.

```
SELECT OBJECT(e) FROM Employee e WHERE e.campus IS NULL
```

- Find the names of the departments--in a given campus--whose employees work on projects with budgets that exceed a given amount:

```
SELECT OBJECT(d) FROM Department d, IN(d.employees) e, IN(e.projects) p
   WHERE e.campus = ?1 AND p.budget > ?2
```

References

- **Jeffrey D. Ullman / OQL lecture note /** http://infolab.stanford.edu/~ullman/fcdb/spr99/lec15.pdf (1999)
- **IBM / Information center** http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/rque_ejbqlrestrict.html (2007)
- **Hibernate.org / Introducing EJB3 Persistence – chapter 7/** http://www.hibernate.org/hib_docs/entitymanager/reference/en/html/queryhql.html (2006)
- ***Dale Green* / Enterprise JavaBeansQuery Language /** http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html (2002)
- ***Shing Wai Chan and Marina Sum*/ Enterprise JavaBeans Query Language**/ http://developers.sun.com/appserver/reference/techart/ejbql.html (2004)

Questions