Outline (Ch. 12, 3rd ed. – Ch. 21, 4th ed.)

- Overview of ODMG
  - Objects and Literals
  - Built-in Interfaces for Collection Objects
  - Atomic (User-defined) Objects
- Object Definition Language
- Object Query Language
- Overview of the O2 System

- **Overview of ODMG**

  The ODMG object model is the data model upon which the object definition langauge (ODL) and object query langauge (OQL) are based.

  Object model:

  - data type, type constructor
  - concepts for specifying object database schemas
  - statements for manipulating objects and retrieval

- **Overview of ODMG**
  - Objects and Literals

    basic building blocks of the object model

    Object: a data unit with obejct identifier and a state

    the state can change over time

    Literal: a value, not with object identifier

    a constatnt, possibly having a complex structure

    not change

- **Overview of ODMG**
  - Objects and Literals

An object is described by four characteristics:

object identifier (Object_Id): a unique system-wide identifier

name: used to refer to the object in a program; the system should be able to locate the object given the name.
Name is often used as the entry point.
Name is unique within a particular database.

lifetime:

persistent object - a permanently existing object in a database if it is not removed explicitely.
Transient object - an object in an executing program that disappears after the program terminates.

- **Overview of ODMG**
  - Objects and Literals

    Object - four characteristics:

    structure: specify how the object is constructed by using the type constructors.

    atomic object - a data unit with a specific structure

    (Note that an atomic object is neither an atom constructor nor an atomic literal.)

    collection object - an object representing a set of objects of some type

    A literal may have a simple or complex structure.

    - atomic literal: corresponds to the value of a basic data type: Long, Short, Unsigned, Float, Double, boolean values, single characters, strings, enumeration type.

- **Overview of ODMG**
  - Objects and Literals

    A literal may have a simple or complex structure.

    - structured literal: correspond roughly to values that are constructed using the tuple constructor:
      Date, Interval, Time, TimeStamp - built-in structures
      user-defined type structures

    Example:

```
struct Dept_Mgr {
        Employee        manager;
        date            startdate
}
```

- **Overview of ODMG**
  - Objects and Literals

    A literal may have a simple or complex structure.

    - collection literal: a value that is a collection of objects or values but the collection itself does not have an Object_Id.

      Set<t>, Bag<t>, List<t>, Array<t>

      Dictionary<k, v>

  Example:

  Set<string>, Bag<float>, ...

- **Overview of ODMG**
  - Built-in Interfaces for Collection Objects

    Interface - something like type, or class

    contains: visible attributes, relationships, operations;

    noninstantiable;

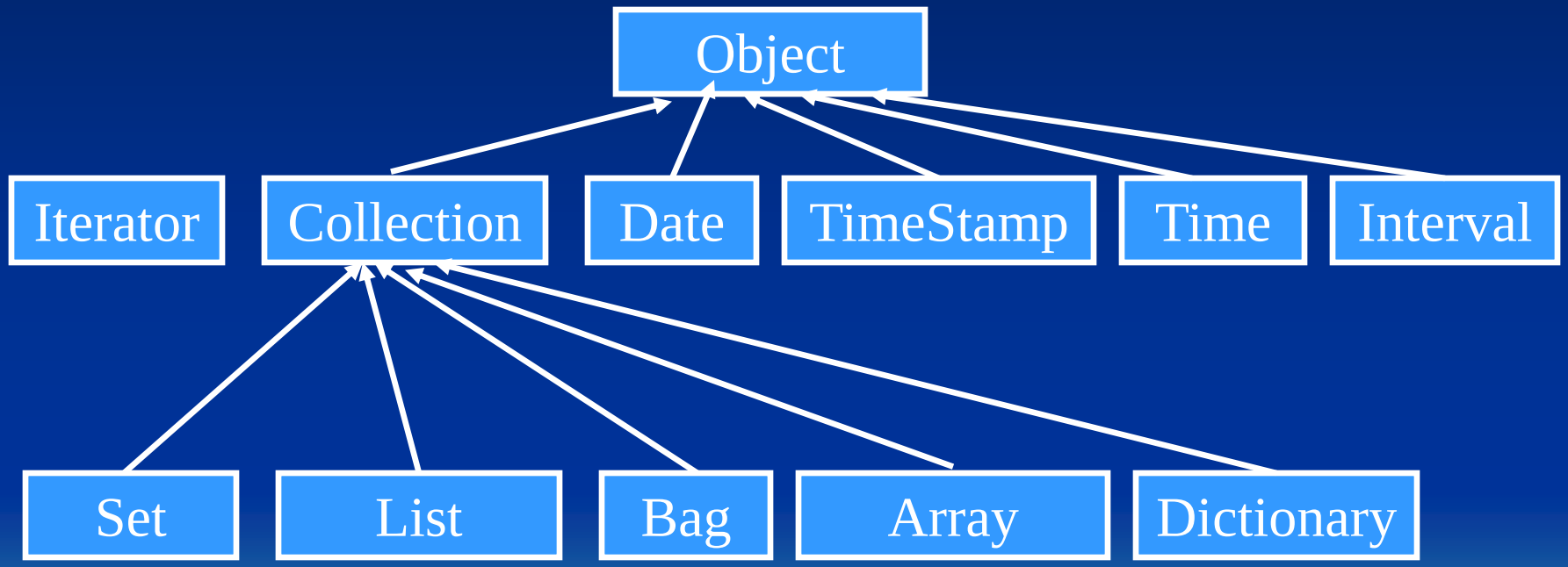
    Operations can be inherited by the user-defined objects.

Example:

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

- **Overview of ODMG**
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.



- **Overview of ODMG**
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.

```
interface Collection : Object { …
        exeption            ElementNotFound(in 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();
        …
}
```

- **Overview of ODMG**
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.

```
interface Iterator  { …
        exeption        NoMoreElement();
        … ...
        boolean         at_end();
        void            reset();
        any             get_element() raises(NoMoreElement);
        void            next_position() raises(NoMoreElement);
        …
}
```

- **Overview of ODMG**
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.

Given a collection object o,

o.cardinality() - number of elements in the collection
o.is_empty() - true or false
o.insert_element(e) - insert e into o
o.remove_element(e) - remove e from o.
o.contains(e) - true or false.

The operation i = o.create_iterator() creates an iterator object i.

i.reset() - sets the iterator at the first element in a collection
i.next_position() - sets the iterator to the next element
i.get_element() - retrieve the current element.

- **Overview of ODMG**
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.

    Set is a subclass of Collection interface

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

Set<t> is a subclass of Set interface (e.g., Set<Student>).

Given a Set<t> object o (t is a data type),

```
p = o.create_union(s)          o.is_subset_of(s)
p = o.create_intersection(s)   o.is_proper_subset(s)
p = o.create_difference(s)     o.is_superset(s)
```

•Overview of ODMG

- Built-in Interfaces for Collection Objects

  Any collection object inherits the basic Collection Interface.

  Bag is a subclass of Collection interface

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

Bag<t> is a subclass of Bag interface.

Given a Bag<t> object o,

```
p = o.create_union(s)
p = o.create_intersection(s)
p = o.create_difference(s)
```

- Overview of ODMG
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.

    List is a subclass of Collection interface

```
interface List : Collection {
        exception    Invalid_Index(unsigned_long index);
        void         remove_element_at(in unsigned_long position)
                                        raised(InvalidIndex);
        any          retrieve_element_at(in unsigned_long position)
                                        raised(InvalidIndex);
        void         replace_element_at(in any element, in unsigned_long position)
                                        raised(InvalidIndex);
        void         insert_element_at(in any element, in unsigned_long position)
                                        raised(InvalidIndex);
        void         insert_element_first(in any element);
                     … …          }
```

List<t> is a subclass of List interface.

Given a List<t> object o,

```
o.insert_element_first(e)
o.insert_element_last(e)
o.insert_element_after(e, i)
o.insert_element_before(e, i)
o.remove_first_element(e)
o.remove_last_element(e)
o.remove_element_at(i)
e = o.retrieve_first_element()
e = o.retrieve_last_element()
e = o.retrieve_element_at(i)
p = o.concat(l)
o.append(l)
```

- Overview of ODMG
  - Built-in Interfaces for Collection Objects

    Any collection object inherits the basic Collection Interface.

    Array is a subclass of Collection interface

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

Array<t> is a subclass of Array interface.

Given an Array<t> object o,

```
o.replace_element_at(i, e)
e = o. remove_element_at(i)
e = o. retrieve_element_at(i)
o.resize(n)
```

•Overview of ODMG

- Built-in Interfaces for Collection Objects

  Any collection object inherits the basic Collection Interface.

  Dictionary is a subclass of Collection interface

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

This allows the creation of a collection of association pairs <k, v>, where all k (key) values are unique. This allows for associative retrieval of a particular pair given its key value (similar to an index).

Dictionary<k, v> is a subclass of Dictionary interface.

Given a Dictionary<k, v> object o,

o.bind(k, v) - binds value v to the key k as an association <k, v> in the collection.
o.unbind(k) - removes the association with key k from o
v = o.lookup(k) - returns the value v associated with key k in o.
o.contains_key(k) - return true if k exists in o; otherwise, return false.

•Overview of ODMG

- **Atomic (User-defined) Objects**

  class - a specification of a data unit:

properties { attributes, relationships }

operations

An atomic object is an instance of some class.

- Overview of ODMG
  - Atomic (User-defined) Objects

    Example:

```
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);   }
```

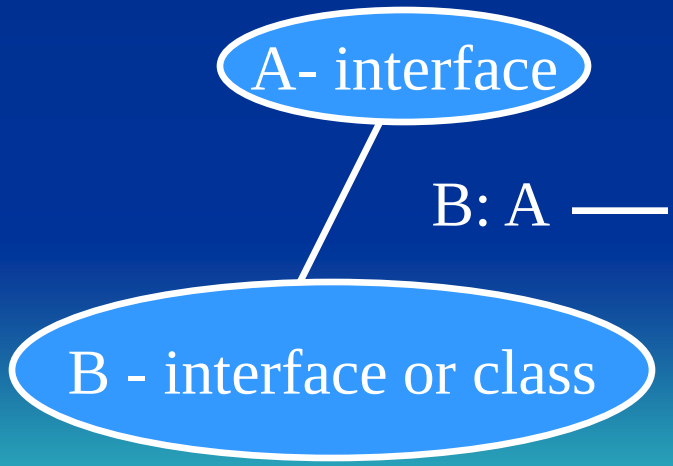
- Overview of ODMG
  - Atomic (User-defined) Objects

Example:

```
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>                      location;
    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); }
```

- Difference between Interfaces and classes

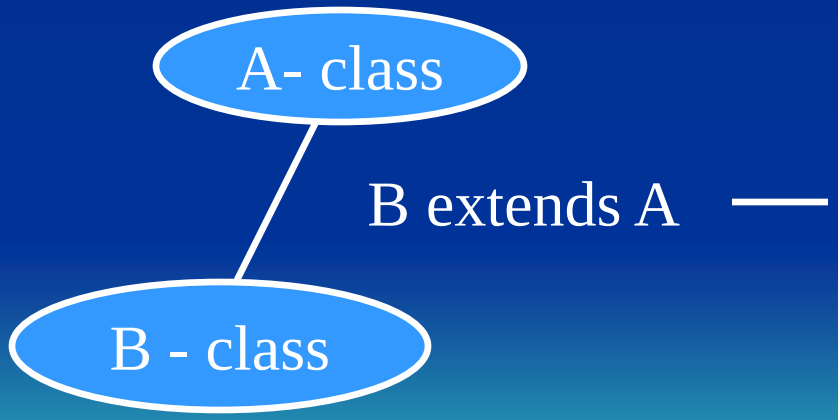
interface - specification of the abstract behaviour of an object type
may have state properties
behaviours can be inherited
state properties cannot be inherited
noninstantiable



B inherits the behaviours of A.

- Difference between Interfaces and classes

class - specification of the abstract behaviour and abstract state of an object type
Both the behaviours and states can be inherited
instantiable



B inherits the behaviours and states of A.

- Extents, Keys, and factory Objects

- Extents: the extent is given a name and will contain all persistent objects of the corresponding class.
  - The extent behaves as a set object that holds all persistent objects of the class.

```
class Department
(           extent    all_departments
            key       dname, dnumber)
{
            … …
}
```

  - If B is a subclass of A, then the extent of B (i.e., named all_B) must be the subset of the extent of A (named all_A):

$$all\_B \subseteq all\_A.$$

- Extents, Keys, and factory Objects

- Keys: A key consists of one or more properties (attributes or relationships) whose values are constrained to be unique for each object in the extent.
  Example: the Employee class has the ssn attribute as key; the Department class has two distinct keys: dname and dnumber.
- Composite key: A composite key is made of several properties (attributes), the properties that form the key are contained in parentheses.
  Example:
  ```
  class Vehicle
  (extent      all_vehicles
  key          (state, license_number)) { … … }
  ```

- Extents, Keys, and Factory Objects

- Factory Object - an interface that can be used to generate or create individual objects via its operations.

Example:

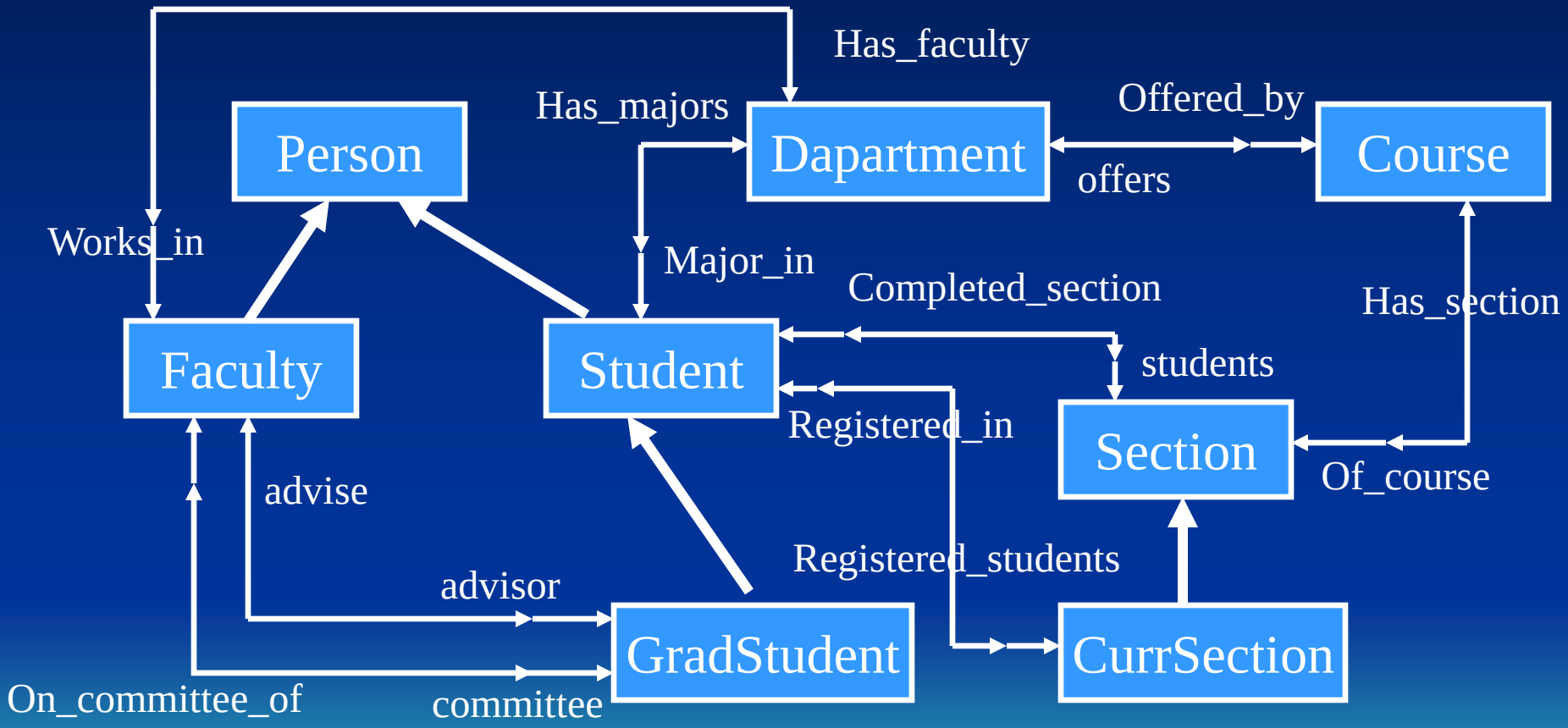
```
interface ObjectFactory {
    Object    new();
}
```

Built-in interface of factory objects

new() returns a new object with an Object_Id.

- **ODL Language**
  - ODL langauge is used to create object specifications: classes and interfaces
  - Using the specific langauge bindings to specify how ODL constructs can be mapped to constructs in specific programming language, such as C++, SMALLTALK, and JAVA.

•ODL Language



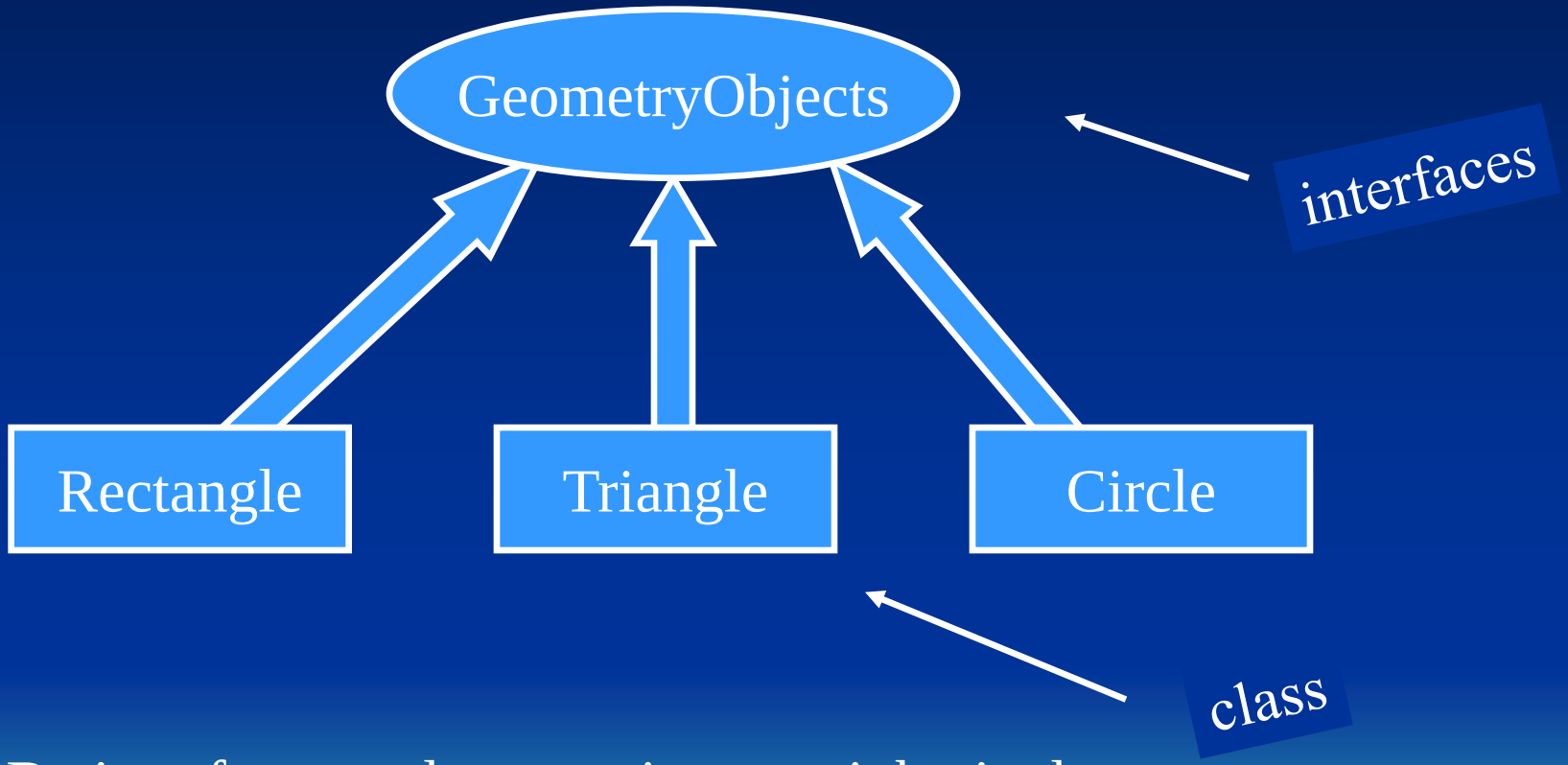
•ODL Language

```
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();
      }
```

•ODL Language

```
class Faculty extends Person
   (   extent  faculy      )
   {   attribute      string                  rank;
       attribute      float                   saraly;
       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);
       }
```

•**ODL Language**



By interfaces, only operations are inherited.

•ODL Language

```
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          rotation(in float angle_of_rotation);
  };
```

•ODL Language

```
class Rectangle : GeometryObject
(  extent      rectangle  )
{  attribute     struct Point {short x, short y}
                                              reference_point;

   attribute     short                        length;
   attribute     short                        height;
   attribute     float                        orientation_angle;
};
```

•OQL Language

The object query language (OQL) is the query language proposed for the ODMG object model.

- Simple OQL queries, database entry points, and iterator variables

  OQL syntax: select ... from ... where

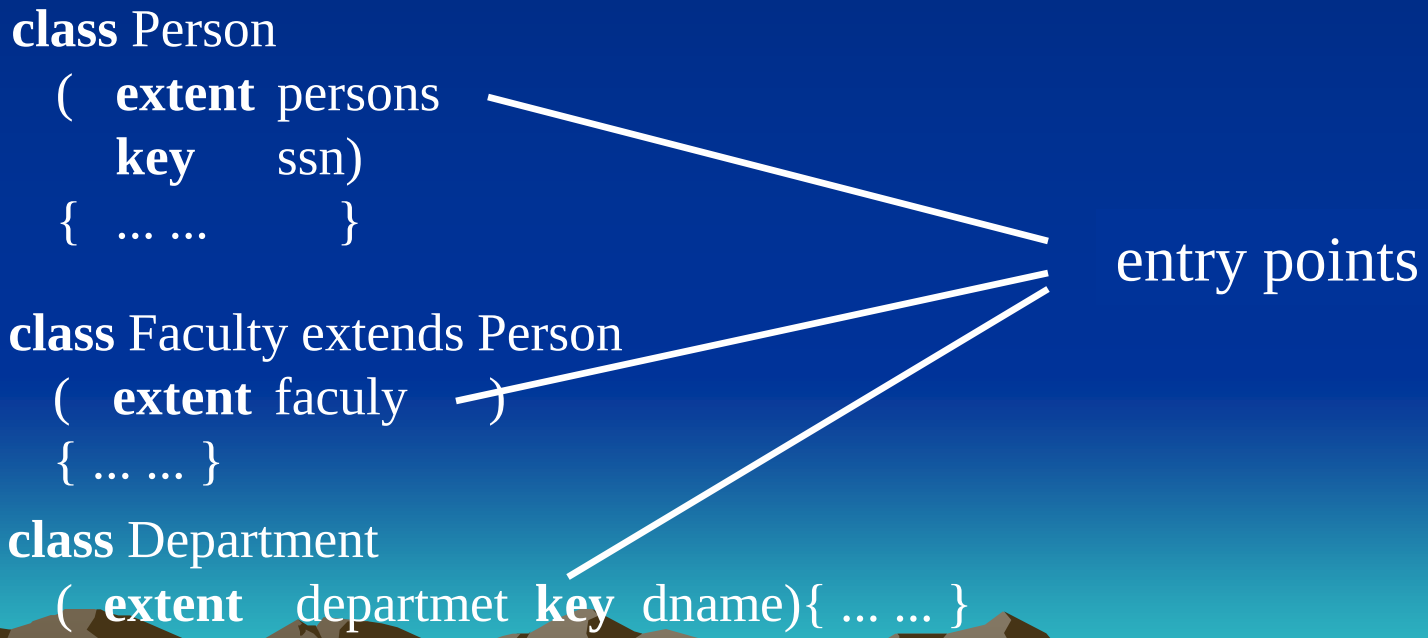
  Example:

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

•OQL Language

Entry point to the database: needed for each query which can be any named *persistent object*:

the name of the extent of a class



- **OQL Language**

Iterator variable:

An entry point refers to a persistent collection of objects.

An iterator variable is defined over a collection of objects.

Example:

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

d is the iterator vaiable

d **in** department

department d

department **as** d

- **OQL Language**
  - Query results and path expressions
    - Any persistent name is a query, whose result is a reference to that persistent object.
      Q1: faculty
      Q1a: csdepartment
      (Here we assume that ‘csdepartment’ is a persistent name to a single department object.)
    - path expression - specify a path to related attributes and objects.
      Q2: csdepartment.chair;
      Q2a: csdepartment.chair.rank;
      Q2b: csdepartment.has_faculty;

- **OQL Language**
  - Query results and path expressions
    - query: return the ranks of compter science faculty:

```
Q3:  select   f.rank
     from     f in csdepartment.has_faculty;

Q3a: select   distinct f.rank
     from     f in csdepartment.has_faculty
```

- **OQL Language**
  - Query results and path expressions
    - query result with a complex structure, using **struct** keyword

Q4: csdepartment.chair.advises;

Q4a: **select struct** (name: **struct**(last_name: s.name.lname,
first_name: s.name.fname),
degrees: (**select struct** (deg: d.degree,
yr: d.year,
college: d.college)
from d **in** s.degree)
from s **in** csdepartment.chair.advises:

- **OQL Language**
  - Query results and path expressions
    - query result with a complex structure, using **struct** keyword

Q5: **select struct** (last_name: s.name.lname, first_name: s.name.fname, gpa: s.gpa),
from s **in** csdepartment.has_majors
where s.class = ‘senior’
order by gpa **desc**, last_name **asc**, first_name **asc**;

- **OQL Language**
  - Query results and path expressions
    - query result with a complex structure, using **struct** keyword

Q5a: **select struct** (last_name: s.name.lname, first_name: s.name.fname, gpa: s.gpa),
from s **in** students
where s.major_in.dname = ‘Computer Science’ **and** s.class = ‘senior’
order by gpa **desc**, last_name **asc**, first_name **asc**;

•Overview of the O2 System

- Data Definition in O2
  In O2, the schema definition uses the C++ (or JAVA) language binding for ODL as defined in ODMG.
  - In C++, a particular library is used to provide classes and operations that implement the ODL constructs.
  - The class library added to C++ for the ODMG standard uses the prefix **d_** for class declaration that deal with database concepts.

```
d_Object
d_Collection<String>
d_Ref<Student>     /*a class to refer to 'Student' objects*/
d_set<d_Ref<Student>>
```

•Overview of the O2 System

- Data Definition in O2

```
struct Ename {
        d_String        fname;
        d_String        mname;
        d_String        lname;
    }
```

```
struct Address {
        d_Ushort        no;
        d_String        street;
        d_Ushort        aptno;
        d_String        city;
        d_String        state;
        d_Ushort        zip;
}
```

- **Overview of the O2 System**
  - Data Definition in O2

```
class Person : public d_Object {
        public:
        //Attributes
        Ename                                   ename;
        d_String                                ssn;
        d_Date                                  birthdate;
        enum Gender{M, F}                       sex;
        Address                                 address;
        //Operations
        Person(const char* pname);
        d_Ushort  age();
        //Extent
        static  d_Set<d_Ref<Person>>    persons;
        static  const char* const               extent_name; }
```

- **Overview of the O2 System**
  - Data Definition in O2

```
class Faculty : public Person {
        public:
        //Attributes
        d_String                                rank;
        d_Float                                 salary;
        d_String                                office;
        d_String                                phone;
        //Relationship (syntax is ODMG 1.1 compliant)
        d_Ref<Department>   works_in inverse Department::has_faculty;
        d_Set<d_Ref<GradStudent>> advise inverse GradStudent ::advisor;
        d_Set<d_Ref<GradStudent>> on_committee_of inverse
                                                GradStudent ::committee;
```

- **Overview of the O2 System**
  - Data Difinition in O2

```
//Operations
Faculty(const char* fname, d_float salary);
void give_raise(in d_float raise);
void promote(in d_String new_rank);
//Extent
static      d_Set<d_Ref<Faculty>>    faculty;
static      const char* const        extent_name;
}
```

- **Overview of the O2 System**
  - Data Manipulation in O2

    Application for O2 can be developed using the C++ (or JAVA) O2 binding, which provides an ODMG-compliant native language binding to the O2 database.

```
//Faculty Class
const char* const Faculty::extent_name = "faculty";
//Faculty constructor here
Faculty::Faculty(const char* fname, d_Float fsalary):
        Person(fname)
{       salary = fsalary;
        //Put this new faculty into the extension
        faculty -> insert_element(this);       }
```

- **Overview of the O2 System**
  - Data Manipulation in O2

```
void Faculty::give_raised(d_Float raise)
{
        salary += raise;
}

void Faculty::promote(d_String new_rank)
{
        rank = new_rank;
}
```

- **Overview of the O2 System**
  - Data Manipulation in O2

```
//Faculty Class
const char* const Faculty::extent_name = “faculty”;
//Faculty constructor here
Faculty::Faculty(const char* fname, d_Float fsalary):
        Person(fname)
{
        salary = fsalary;
        //Put this new faculty into the extension
        faculty -> insert_element(this);
}
```