Object-Oriented Database Development

Objectives

- Definition of terms
- Create object-oriented database schemas in ODL
- Transform UML class diagrams to ODL schemas
- Identify type specifications for attributes, arguments, and operation return values
- Create objects and specify their attribute values
- Understand object-oriented database implementation steps
- Understand OQL syntax and semantics
- Understand object-oriented database applications

Object Definition Language (ODL)

- Corresponds to SQL's DDL (Data Definition Language)
- Specify the logical schema for an object-oriented database
- Based on the specifications of Object Database Management Group (ODMG)

Defining a Class

- **class** – keyword for defining classes
- **attribute** – keyword for attributes
- **operations** – return type, name, parameters in parentheses
- **relationship** – keyword for establishing relationship

```
class Student {
   attribute string name;
   attribute Date dateOfBirth;
   attribute Address address;
   attribute Phone phone;
// relationship between Student and CourseOffering
   relationship set <CourseOffering> takes inverse CourseOffering::taken_by;
// operations
   short age( );
   float gpa( );
   boolean register_for(string crse, short sec, string term);
};
```

See page 620

Defining an Attribute

- Value can be either:
  - Object identifier OR Literal
- Types of literals
  - Atomic – a constant that cannot be decomposed into components
  - Collection – multiple literals or object types
  - Structure – a fixed number of named elements, each of which could be a literal or object type
- Attribute ranges
  - Allowable values for an attribute
  - enum – for enumerating the allowable values

Kinds of Collections

- Set – unordered collection without duplicates
- Bag – unordered collection that may contain duplicates
- List – ordered collection, all the same type
- Array – dynamically sized ordered collection, locatable by position
- Dictionary – unordered sequence of key-value pairs without duplicates

Defining Structures

Structure = user-defined type with components

struct keyword

Example:

```
struct Address {
    String street_address
    String city;
    String state;
    String zip;
};
```

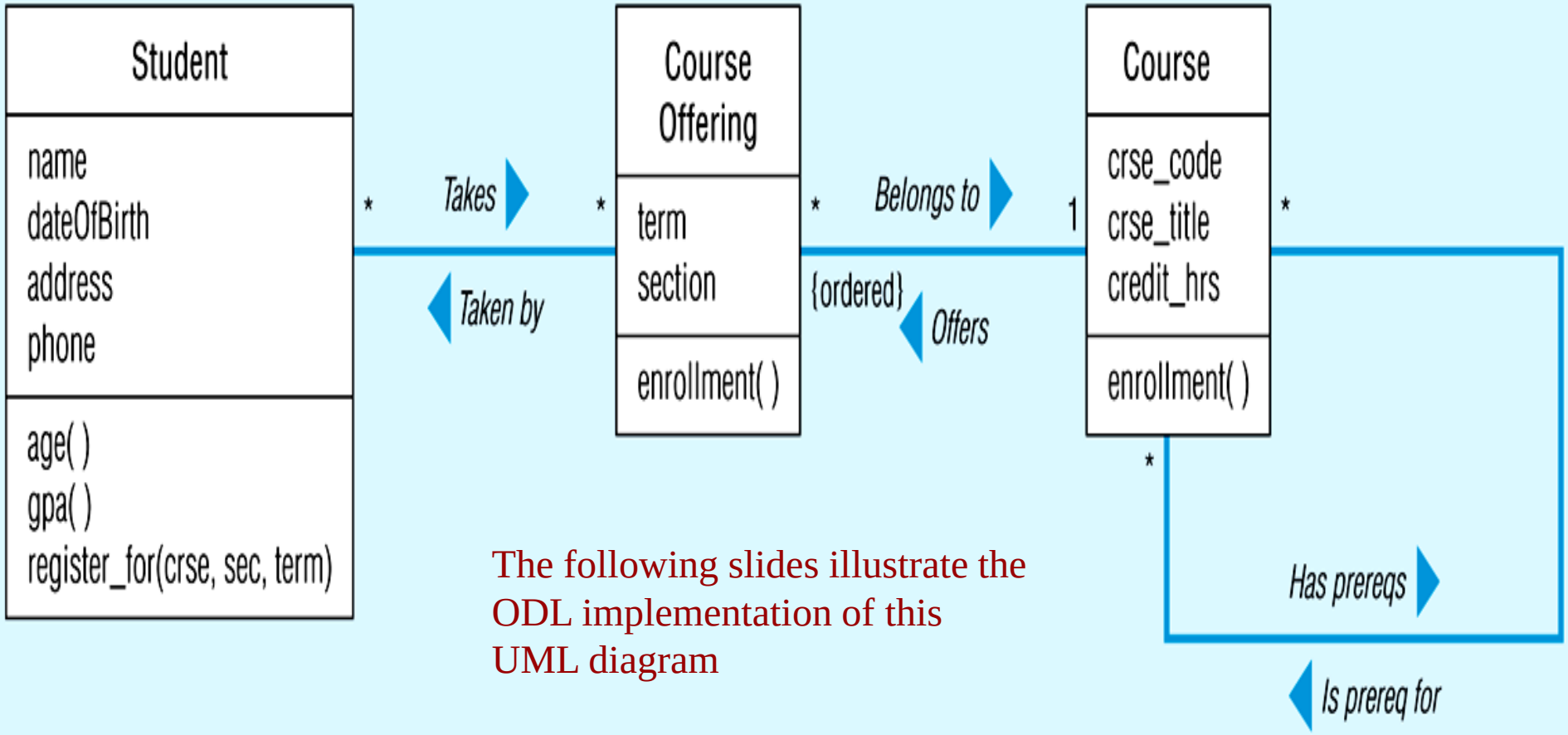
Defining Operations

- Return type
- Name
- Parentheses following the name
- Arguments within the parentheses

Defining Relationships

- Only unary and binary relationships allowed
- Relationships are bi-directional
  - implemented through use of **inverse** keyword
- ODL relationships are specified:
  - **relationship** indicates that class is on many-side
  - **relationship set** indicates that class is on one-side and other class (many) instances unordered
  - **relationship list** indicates that class is on one-side and other class (many) instances ordered

Figure 15-1: UML class diagram for a university database



The following slides illustrate the ODL implementation of this UML diagram

Figure 15-2: ODL Schema for university database

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

Figure 15-2: ODL Schema for university database (cont.)

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

class keyword begins the class definition.Class components enclosed between **{** and **}**

Figure 15-2: ODL Schema for university database (cont.)

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

attribute has a data type and a name

specify allowable values using **enum**

Figure 15-2: ODL Schema for university database (cont.)

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

extent = the set of all instances of the class

Figure 15-2: ODL Schema for university database (cont.)

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

Operation definition: return type, name, and argument list. Arguments include data types and names

Figure 15-2: ODL Schema for university database (cont.)

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

relationship sets indicate 1:N relationship to an *unordered* collection of instances of the other class

inverse establishes the bidirectionality of the relationship

Figure 15-2: ODL Schema for university database (cont.)

```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

relationship list indicates 1:N relationship to an *ordered* collection of instances of the other class

Figure 15-2: ODL Schema for university database (cont.)

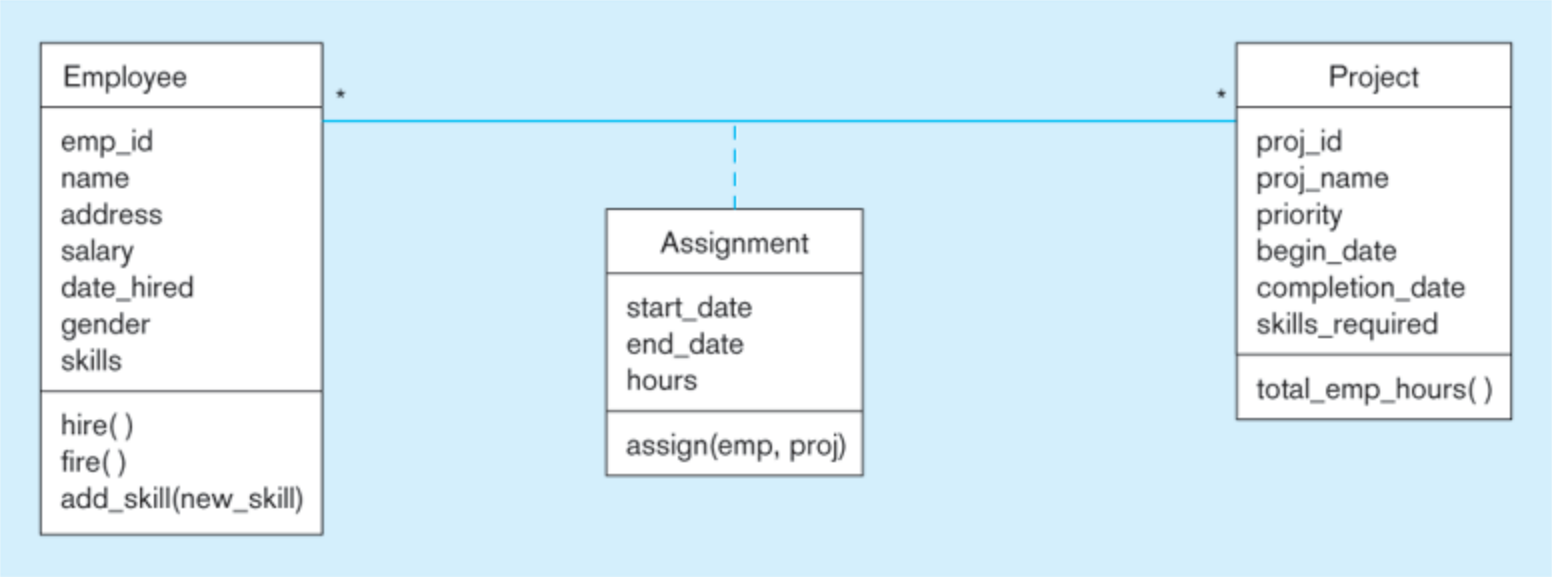
```
class Student {
(     extent students)
      attribute string name;
      attribute Date dateOfBirth;
      attribute Address address;
      attribute Phone phone;
      relationship set (CourseOffering) takes inverse CourseOffering::taken_by;
      short age( );
      float gpa( );
      boolean register_for(string crse, short sec, string term);
};

class CourseOffering {
(     extent courseofferings)
      attribute string term;
      attribute enum section {1, 2, 3, 4, 5, 6, 7, 8};
      relationship set (Student) taken_by inverse Student::takes;
      relationship Course belongs_to inverse Course::offers;
      short enrollment( );
};

class Course {
(     extent courses)
      attribute string crse_code;
      attribute string crse_title;
      attribute short credit_hrs;
      relationship set (Course) has_prereqs inverse Course::is_prereq_for;
      relationship set (Course) is_prereq_for inverse Course::has_prereqs;
      relationship list (CourseOffering) offers inverse CourseOffering::belongs_to;
      short enrollment( );
};
```

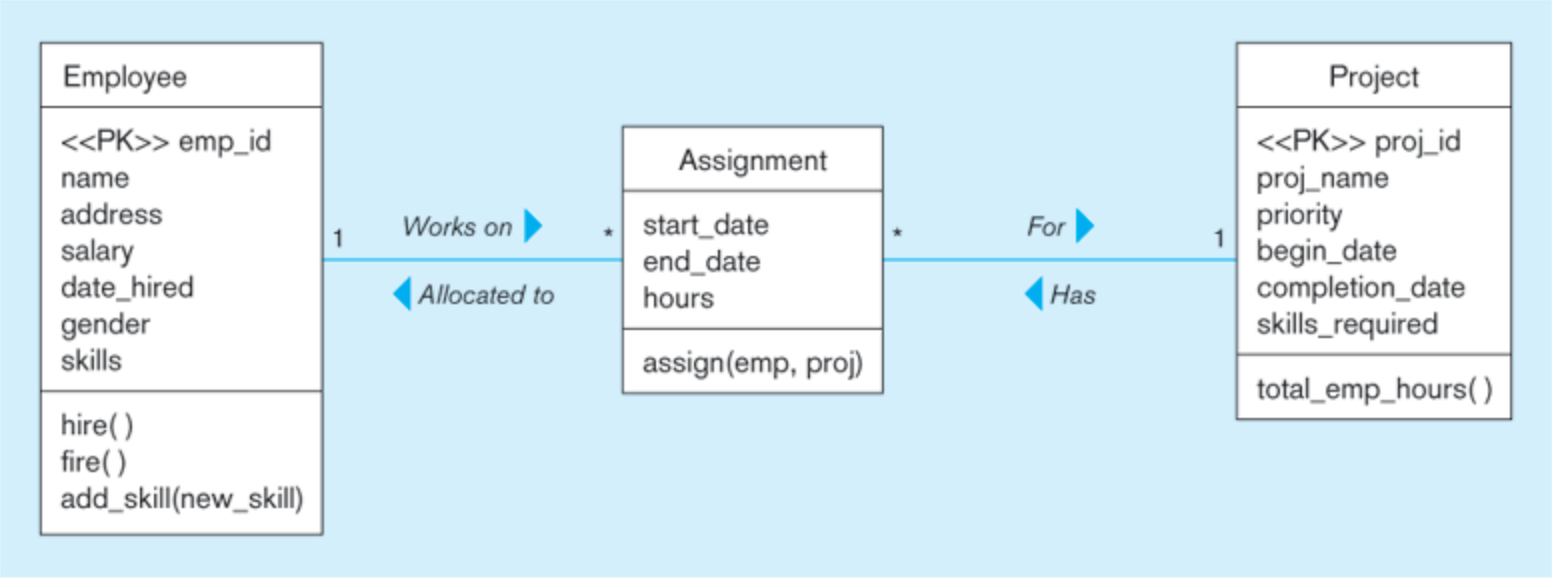
relationship indicates N:1 relationship to an instance of the other class

Figure 15-3a UML class diagram for an employee project database - Many-to-many relationship with an association class



In order to capture special features of assignment, this should be converted into two 1:N relationships

Figure 15-3b UML class diagram for an employee project database - Many-to-many relationship broken into two one-to-many relationships

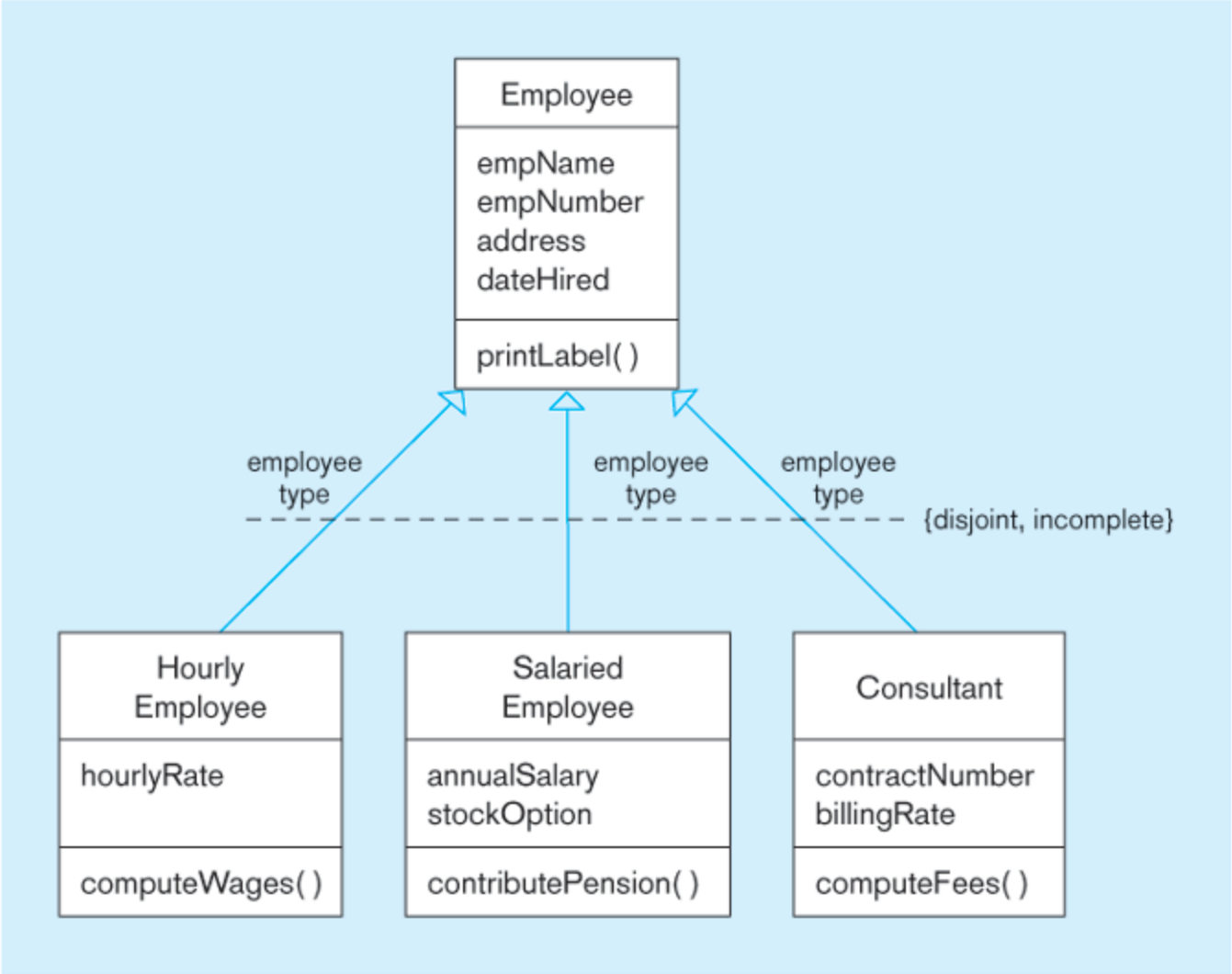



```
class Employee {
(extent employees
    key emp_id)
     ……
    attribute set <string> skills_required;
};
```

Note:
key indicates identifier
(candidate key)

Note: **attribute set** indicates a multivalued attribute

Figure 15-4 UML class diagram showing employee generalization

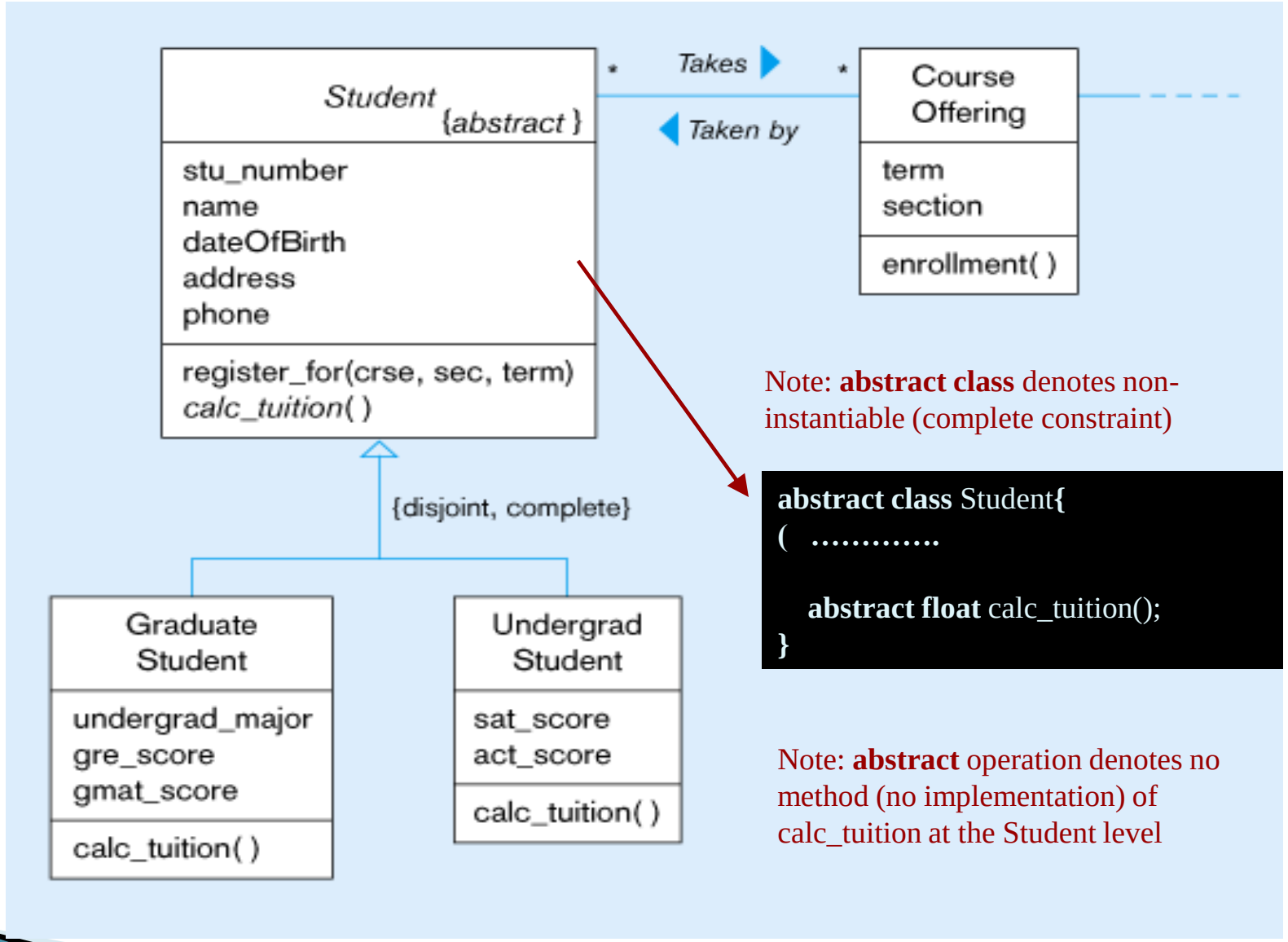


Note:
extends denotes subclassing

```
class HourlyEmployee
      extends Employee{
(   ..............

    ..............
}
```

Figure 15-5: UML class diagram showing student generalization



Creating Object Instances

- Specify a tag that will be the object identifier
  - MBA699 course ();
- Initializing attributes:
  - Cheryl student (name: “Cheryl Davis”, dateOfBirth:4/5/77);
- Initializing multivalued attributes:
  - Dan employee (emp_id: 3678, name: “Dan Bellon”, skills {“Database design”, “OO Modeling”});
- Establishing links for relationship
  - Cheryl student (takes: {OOAD99F, Telecom99F, Java99F});

Querying Objects in the OODB

- Object Query Language (OQL)
- ODMG standard language
- Similar to SQL-92
- Some differences:
  - Joins use class's relationship name:
    - Select x.enrollment from courseofferings x, ***x.belongs_to y*** where y.crse_course = "MBA 664" and x.section = 1;
  - Using a set in a query
    - Select emp_id, name from employees where ***"Database Design" in skills***;

Current ODBMS Products

- Rising popularity due to:
  - CAD/CAM applications
  - Geographic information systems
  - Multimedia
  - Web-based applications
  - Increasingly complex data types
- Applications of ODBMS
  - Bill-of-material
  - Telecommunications navigation
  - Health care
  - Engineering design
  - Finance and trading

Table 15-1 ODBMS Products

Company	*Product*	*Website*
GemStone Systems	GemStone	**www.gemstone.com**
neoLogic	NeoAccess	**neologic.com**
Object Design	ObjectStore	**www.odi.com**
Objectivity	Objectivity/DB	**www.objectivity.com**
POET Software	POET Object Server	**www.poet.com**
Versant	Versant ODBMS	**www.versant.com**
Other Links Related to ODBMS Products		
Barry & Associates		**www.odbmsfacts.com**
Doug Barry's *The Object Database Handbook*		**wiley.com**
Object database newsgroup		**news://comp.databases.object**
Rick Cattell's *The Object Database Standard ODMG 3.0*		**www.mkp.com**
Object Database Management Group		**www.odmg.org**
Chaudhri and and Zicarl's *Succeeding with Object Databases*		**www.wiley.com/compbooks/chaudhri**