Schedule

• Nov. 8 (TH) Object-Oriented Database Design.
  ♦ Read Sections 4.1-4.4. Project Part 5 due on Sunday night.
• Nov. 13 (T) Object-Relational, Object Queries (OQL).
  ♦ Read Sections 4.5, 9.1. Assignment 6 due. (No office hours.)
• Nov. 15 (TH) More OQL.
  ♦ Read Sections 9.2-9.3. Project Part 6 due on Sunday night.
• Nov. 20 (T) Object-Relational Queries.
  ♦ Read Sections 9.4-9.5. Assignment 7 due (no late ones).
• Nov. 22 (TH) Thanksgiving - No class scheduled.
• Nov. 27 (T) Semistructured Data, XML.
  ♦ Read Sections 4.6-4.7. Assignment 8 due. Project Part 7 due.
• Nov. 29 (TH) The Real World, Review.
Object-Oriented DBMS’s

• ODMG = Object Data Management Group: an OO standard for databases.

• ODL = Object Description Language: design in the OO style.

• OQL = Object Query Language: queries an OO database with an ODL schema, in a manner similar to SQL.
ODL Overview

• Class declarations (*interfaces*).
• Interface includes:
  1. Name for the interface.
  2. Key declaration(s), which are optional.
  3. *Extent* declaration = name for the set of currently existing objects of a class.
  4. *Element* declarations. An element is an attribute, a relationship, or a method.
ODL Class Declarations

interface <name> {  
  elements = attributes, relationships, methods  
}  

Element Declarations

attribute <type> <name>;
relationship <rangetype> <name>;

• Relationships involve objects; attributes involve non-object values, e.g., integers.

Method Example

float gpa(in string) raises(noGrades)

• float = return type.
• in: indicates the argument (a student name, presumably) is read-only.
  ♦ Other options: out, inout.
• noGrades is an exception that can be raised by method gpa.
ODL Relationships

• Only binary relations supported.
  ◆ Multiway relationships require a “connecting” class, as discussed for E/R model.

• Relationships come in inverse pairs.
  ◆ Example: “Sells” between beers and bars is represented by a relationship in bars, giving the beers sold, and a relationship in beers giving the bars that sell it.

• Many-many relationships have a set type (called a collection type) in each direction.

• Many-one relationships have a set type for the one, and a simple class name for the many.

• One-one relations have classes for both.
Beers-Bars-Drinkers Example

interface Beers {
    attribute string name;
    attribute string manf;
    relationship Set<Bars> servedAt
        inverse Bars::serves;
    relationship Set<Drinkers> fans
        inverse Drinkers::likes;
}

• An element from another class is indicated by <class>:::

• Form a set type with Set<type>.
interface Bars {
    attribute string name;
    attribute Struct Addr
        {string street, string city, int zip} address;
    attribute Enum Lic
        {full, beer, none} licenseType;
    relationship Set<Drinkers> customers
        inverse Drinkers::frequents;
    relationship Set<Beers> serves
        inverse Beers::servedAt;
}

- Structured types have names and bracketed lists of field-type pairs.
- Enumerated types have names and bracketed lists of values.
interface Drinkers {
    attribute string name;
    attribute Struct Bars::Addr address;
    relationship Set<Beers> likes
        inverse Beers::fans;
    relationship Set<Bars> frequents
        inverse Bars::customers;
}

• Note reuse of Addr type.
ODL Type System

- Basic types: int, real/float, string, enumerated types, and classes.
- Type constructors: Struct for structures and four collection types: Set, Bag, List, and Array.

Limitation on Nesting

Relationship

- class
- collection

Attribute

- basic, no class
- struct
- collection
Many-One Relationships

Don’t use a collection type for relationship in the “many” class.

Example: Drinkers Have Favorite Beers

```java
interface Drinkers {
    attribute string name;
    attribute Struct Bars::Addr address;
    relationship Set<Beers> likes
        inverse Beers::fans;
    relationship Beers favoriteBeer
        inverse Beers::realFans;
    relationship Set<Bars> frequents
        inverse Bars::customers;
}
```

- Also add to Beers:
  ```java
  relationship Set<Drinkers> realFans
    inverse Drinkers::favoriteBeer;
  ```
Example: Multiway Relationship

Consider a 3-way relationship bars-beers-prices. We have to create a connecting class BBP.

```java
interface Prices {
    attribute real price;
    relationship Set<BBP> toBBP
        inverse BBP::thePrice;
}

interface BBP {
    relationship Bars theBar inverse ... 
    relationship Beers theBeer inverse ... 
    relationship Prices thePrice
        inverse Prices::toBBP;
}
```

- Inverses for `theBar`, `theBeer` must be added to `Bars`, `Beers`.
- Better in this special case: make no `Prices` class; make `price` an attribute of `BBP`.
- Notice that keys are optional.
  - `BBP` has no key, yet is not “weak.” Object identity suffices to distinguish different `BBP` objects.
Roles in ODL

Names of relationships handle “roles.”

Example: Spouses and Drinking Buddies

```java
interface Drinkers {
    attribute string name;
    attribute Struct Bars::Addr
        address;
    relationship Set<Beers> likes
        inverse Beers::fans;
    relationship Set<Bars> frequents
        inverse Bars::customers;
    relationship Drinkers husband
        inverse wife;
    relationship Drinkers wife
        inverse husband;
    relationship Set<Drinkers> buddies
        inverse buddies;
}
```

- Notice that `Drinkers::` is optional when the inverse is a relationship of the same class.