Object Definition Language

- Design language derived from the OO community:
- Can be used like E/R as a preliminary design for a relational DB.
ODL

- **Class Declarations**
  - interface `<name>` { `elements = attributes, relationships, methods` }

- **Element Declarations**
  - attribute `<type>` `<name>` ;
  - relationship `<rangetype>` `<name>` ;

- **Method Example**
  - float gpa(in: Student) raises(noGrades)
    - float = return type.
    - in: indicates Student argument is read-only.
      - Other options: out, inout.
    - noGrades is an exception that can be raised by method gpa.
Banking Example 1

- **Keys**: ss#, loanid, branchid
- **Cardinality constraint**: each loan belongs to a single branch
interface Customer {
    attribute string name; attribute integer ss#;
    attribute Struct Addr {string street, string city, int zip} address;
    relationship Set<Loans> borrowed
        inverse Loans::borrower;
    relationship Set<Branch> has-account-at
        inverse Branch:patrons;
    key(ss#)
}

Structured types have names and bracketed lists of field-type pairs.
Relationships have inverses.
An element from another class is indicated by < class > ::
Form a set type with Set<type>.
Loans Example (III)

- interface loans {
  attribute real amount;
  attribute int loanid;
  attribute Enum loanType {house, car, general} type;
  relationship Branch belongs-to
    inverse Branch::loans-granted;
  relationship Set<Customer> borrower
    inverse Customer::borrowed;
  key(loanid)
}

- Enumerated types have names and bracketed lists of values.
interface Branch {
    attribute integer branchid;
    attribute Struct Customer::Addr location;
    relationship Set<Loans> loans-granted
        inverse Loans::belongs-to;
    relationship Set<Customer> patrons
        inverse Customer::has-account-at;
    key(branchid);
}

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.
Limitations on Nesting

Relationship

class → collection

Attribute

Basic, no class → struct → collection
ER versus ODL

- E/R: arrow pointing to “one.”
- ODL: don't use a collection type for relationship in the “many” class.
  - Collection type remains in “one.”
- E/R: arrows in both directions.
- ODL: omit collection types in both directions

- ODL only supports binary relationship.
- Convert multi-way relationships to binary and then represent in ODL
  - create a new connecting entity set to represent the rows in the relationship set.
  - Problems handling cardinality constraints properly!!
Roles in ODL

- No problem; names of relationships handle roles.”

```java
interface employee {
    attribute string name;
    relationship Set<Employee> manager
        inverse Employee::worker;
    relationship Set<Employee> worker
        inverse Employee::manager
}
```

```
manager

works for

worker
```
Subclasses in ODL

- Subclass = special case = fewer entities/objects = more properties.
- Example: Faculty and Staff are subclasses of Employee. Faculty have academic year (9 month salaries) but staff has a full-year (12 month salary).
ODL Subclasses

- Follow name of subclass by colon and its superclass.
- interface Faculty:Employee {
  attribute real academic-year-salary;
}
- Objects of the Faculty class acquire all the attributes and relationships of the Employee class.
- Inheritance in ODL and ER model differ in a subtle way
  - in ODL an object must be member of exactly one class
  - in ER an object can be member of more than one class
Keys in ODL

- Indicate with key(s) following the class name, and a list of attributes forming the key.
  - Several lists may be used to indicate several alternative keys.
  - Parentheses group members of a key, and also group key to the declared keys.
  - Thus, \((\text{key}(a_1; a_2; \ldots; a_n))\) = “one key consisting of all n attributes.” \((\text{key } a_1; a_2; \ldots; a_n\) ) =“each \(a_i\) is a key by itself.

- Keys are not necessary for ODL. Object identity and not keys differentiates objects