Object-Oriented Databases
ODMG Standard

- Object Model and Object Definition Language
- Object Query Language (OQL)
- Programming Language Bindings

Development of OODBMS

- Many systems closely related to programming languages
  - Versant, Ontos, ObjectStore, Objectivity (C++)
  - GemStone (Smalltalk)
- Early versions had no query language support
  - ObjectStore had limited selection-based queries
- O_2 developed at INRIA (France) with large funding from European research projects
  - took more of a database approach
  - intended to be language independent
  - lot of research on query languages
  - interests also in interface and development tools support

Standards for Object Data Management

- Object Management Group (OMG)
  - architectures and tools to develop object-oriented systems
  - distributed object management
  - best known for Unified Modeling Language (UML)
- Object Data Management Group (ODMG)
  - data management support
  - complementary to OMG
  - ODMG data model based on OMG object model

Object Data Management Group (ODMG)

- ODMG formed very early in development of OODBMS
- Informal standards body involving all major vendors
  - initiated in 1991 by Rick Cattell of SunSoft
  - initially ODMG comprised five people from OODBMS vendors
- Promote portability and interoperability across products
- Not developing a standard OODBMS product
  - products will vary in terms of languages, tools, interfaces, performance, etc.
  - products may be tailored to application domains, e.g. version management for Computer-Aided Software Engineering (CASE)

ODMG Standard

- Object Model
- Object Definition Language (ODL)
- Object Query Language (OQL)
- Language bindings
  - C++ Binding
  - Smalltalk Binding
  - Java Binding

ODMG Object Model

- Based on the OMG object model
- Basic modelling primitives
  - object unique identifier
  - literal no identifier
- Object state defined by the values carried for a set of properties, i.e. attributes or relationships
- Object behaviour defined by the set of operations that can be executed
- Objects and literals are categorised by their type which defines common properties and common behaviour

Types

- Specification
  - properties, i.e. attributes and relationships
  - operations
  - exceptions
- Implementation
  - language binding
  - a specification can have more than one implementation

Type Specifications

- Interface
  - defines only abstract behaviour
  - `interface Employee {...};`
- Class
  - defines both abstract behaviour and abstract state
  - `class Person {...};`
- Literal
  - defines abstract state
  - `struct Complex { float real; float imaginary; };`

Type Implementation

- Representation
  - data structure
  - derived from type's abstract state by the language binding
- Methods
  - procedure bodies
  - derived from type's abstract behaviour by the language binding
  - also private methods with no counterpart in specification

Subtyping and Inheritance

- Two types of inheritance relationships
- IS-A relationship
  - inheritance of behaviour
  - multiple inheritance, name overloading disallowed
  - `interface Professor : Employee {...};`
- EXTENDS relationship
  - inheritance of state and behaviour
  - single inheritance
  - `class EmplPers extends Person : Employee {...};`

Extents

- Extent of a type is the set of all active instances
  - assume class **`Person`**
  - extent of class **`Person`** would be the current set of all person objects in the data management system
- Extents can be maintained automatically

Collections

- Supports both collection objects and collection literals
  - set unordered, no duplicates
  - bag unordered, duplicates
  - list ordered, elements can be inserted
  - array ordered, elements can be replaced
  - dictionary maps keys to values
- Collection objects
  - Set<t>, Bag<t>, List<t>, Array<t>, Dictionary<t,v>
- Collection literals
  - set<t>, bag<t>, list<t>, array<t>, dictionary<t,v>

Collections

- Subset containment relation defined only over sets
- Operations union, intersection and difference defined only over sets and bags
- No constraints over collections

Collections

```
interface Collection : Object {
   exception InvalidCollectionType{};
   exception ElementNotFound{ Object element; };
   boolean   is_empty();
   ...
   boolean   contains_element(in Object element);
   void      insert_element(in Object element);
   void      remove_element(in Object element) raise (ElementNotFound);
   ...
   Iterator  create_iterator(in boolean stable);
   ...
   boolean   query(in string oql_predicate, inout Collection result);
};
```

Sets and Bags

```
class Set : Collection {
   attribute set<t> value;
   Set       create_union(in Set other_set);
   Set       create_intersection(in Set other_set);
   Set       create_difference(in Set other_set);
   boolean   is_subset_of(in Set other_set);
   boolean   is_proper_subset_of(in Set other_set);
   ...
};

class Bag : Collection {
   attribute     bag<t> value;
   unsigned long occurrences_of(in Object element);
   Bag           create_union(in Bag other_bag);
   Bag           create_intersection(in Bag other_bag);
   Bag           create_difference(in Bag other_bag);
   ...
};
```

Relationships

- All relationships are binary
  - **many-to-many** relationships have a collection for both the type of the relationship and its inverse
  - **many-to-one** relationships have a collection for the type of the relationship and class for its inverse
  - **one-to-one** relationships have a class both for the type of the relationship type and for its inverse
- No support for ternary (or higher) relationships or relationship attributes
  - can be simulated by creating classes to represent relationship tuples
- System maintains referential integrity

Persistence

- Persistence by reachability
- Database gives access to global names
  - explicitly named root objects
  - types defined in schema
  - named extents of types

Other Concepts Supported

- Database operations
- Locking and concurrency control
- Transactions
- Access to metadata
- Built-in structured literals and objects
  - dates
  - times
  - timestamps
  - intervals
  - ...

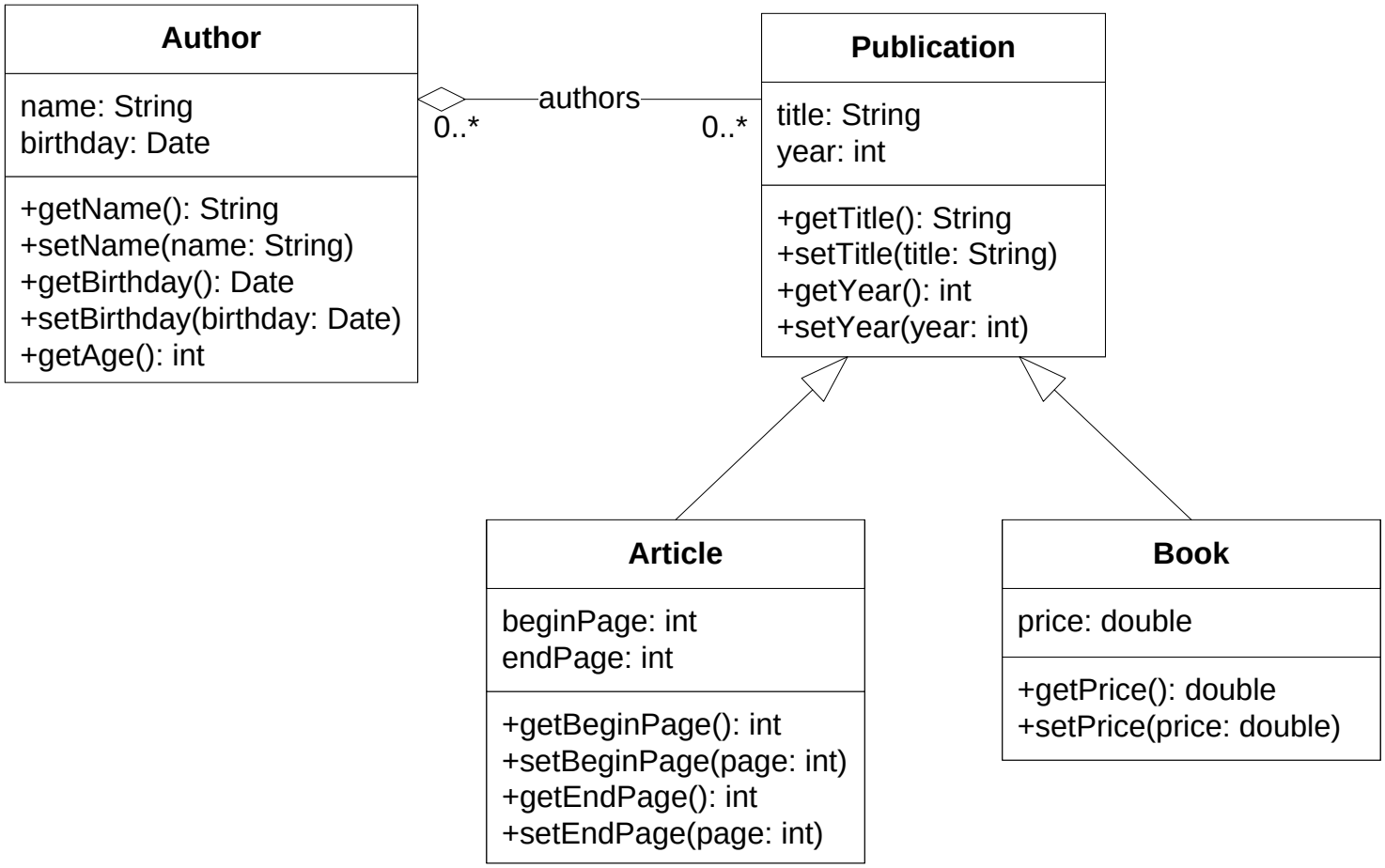
Object Definition Language (ODL)

- Object-oriented design and specification language
  - supports semantic constructs of ODMG object model
  - programming language independent, extensible and practical
  - compatible to OMG Interface Definition Language (IDL)
- ODL object class definition

```
class name [ ( extent name , key name ) ] {
    { exception name { { type name } } }
    { attribute type name }
    { relationship type name inverse relationship }
    { type name ( { ( in | out | inout ) type name } ) [ raises ( { exception } ) ] }
}
```

  - extent and key of a class can be specified optionally
  - relationships specify inverse to maintain referential integrity
  - methods signatures are implemented by language binding

Example Class Hierarchy



ODL Example

```
class Author (extent Authors) {
   attribute string name;
   attribute date birthday;
   relationship set<Publication> authors inverse Publication::authored_by;
   integer get_age();
};

class Publication (extent Publications) {
   attribute string title;
   attribute integer year;
   relationship list<Author> authored_by inverse Author::authors;
};

class Article extends Publication (extent Articles) {
   attribute unsigned short begin_page;
   attribute unsigned short end_page;
};

class Book extends Publication (extent Books) {
   attribute double price;
};
```

Object Query Language (OQL)

- Based on ODMG Object Model and SQL-92

```
select  list of values
from    list of collections and typical members
where   condition
```

  - collections can be extents or expressions that evaluate to collection
  - names can be used to denote a typical member of a collection
- Path expressions to navigate complex objects
  - `person.spouse.address.street.name`
- Not computationally complete
  - rather simple to use query language
- No explicit update operators
  - but can invoke update operations on objects

Basic Select Statement

- Find the titles of all publications with more than one author that were published after 1995

```
select p.title
from   Publications p
where  p.year > 1995
and    count(p.authored_by) > 1
```

- Find the titles of all publications authored by Tilmann Zäschke

```
select a.authors.title
from   Authors a
where  a.name = "Tilmann Zaeschke"
```

Illegal as the “dot” operator cannot be applied to a collection of objects

```
select p.title
from   Authors a, a.authors p
where  a.name = "Tilmann Zaeschke"
```

Correct solution based on correlated variables

Return Types

- Queries return sets, bags or lists
  - as a default, queries return a bag

```
select first: p.authored_by[1], p.title, p.year
from   Publications p
```

```
Bag<Struct { Author first, string title, integer year }>
```

  - queries with **`DISTINCT`** return a set

```
select distinct a.name
from   Authors a
```

```
Set<Struct { string name }>
```

  - queries with **`ORDER BY`** return a list

```
select   p.title
from     Publications s
order by p.year desc
```

```
List<Struct { string name }>
```

Subqueries

- Subqueries are mainly expressed in **`FROM`** clauses
- Find the names of all co-authors of Michael Nebeling

```
select distinct a.name
from   (select mp
        from   Authors m, m.authors mp
        where  m.name = "Michael Nebeling") p, p.authored_by a
```

- Find the titles of the articles that were published in the same year at the book on the ODMG 3.0 standard

```
select p.title
from   Articles p, (select o.year
                    from   Books o
                    where  o.title = "ODMG 3.0") y
where  p.year in y
```

Universal and Existential Quantification

- Boolean expressions that can be used in **`WHERE`** clauses
- Find the names of authors who have written a book that costs less than 20 €

```
select a.name
from   Authors a
where  exists b in Books:
       b.price < 20 and b in a.authors
```

- Find the names of authors who have not published anything since 2000

```
select a.name
from   Authors a
where  for all p in a.authors:
       p.year <= 2000
```

Collection Expressions

- Aggregate operators
  - **`AVG`**, **`SUM`**, **`MIN`**, **`MAX`**, and **`COUNT`** apply to collections that have a compatible member type
- Operations for sets and bags
  - **`UNION`**, **`INTERSECTION`** and **`EXCEPT`**
  - inclusion tests (subset, superset)
- Special operations for lists
- Simple coercions
  - a collection of one element can be coerced to that element using the **`ELEMENT`** operator
- Flattening a collection of collections

"ODMG 4.0"

- ODMG was dissolved in 2001
- OMG obtained rights to ODMG 3.0 in 2003
- OMG Object Database Technology Working Group (ODBTWG) was founded in 2005 in response to renewed interest in object-oriented databases
- First white paper proposes object calculus based on abstract store model and stack-based queries

Literature

- R. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, Fernando Velez (Editors): **The Object Data Standard: ODMG 3.0,** *Morgan Kaufmann 2000*
- OMG Object Database Technology Working Group: **Next-Generation Object Database Standarization**, *White Paper, September 2007*

Next Week
Commercial OODBMS: Versant

- Versant Object Database for Java
- Java Versant Interface (JVI)
- Versant Query Language (VQL)