Standard ODMG

ALBERTO BELUSSI

ANNO ACCADEMICO 2009/'10

Sommario - ODMG

- Lo standard ODMG
  - Il modello dei dati
    - Tipi e classi
    - Oggetti
    - Collezioni
    - Letterali
    - Proprietà
    - Operazioni
    - Transazioni
    - Operazioni sulla BD
  - Object Definition Language (ODL)
  - Object Query Language (OQL)

Object Management Group e Object Database Management Group

- OMG (Object Management Group)
  - associazione privata nata nel 1989 con lo scopo di promuovere l'uso di standard nell'area OO
- ODMG (Object Database Management Group)
  - Il consorzio si è costituito (1991) quando è apparso evidente che la mancanza di un modello e di un linguaggio di interrogazione standard negli OODBMS era un elemento di debolezza rispetto ad un mercato che richiede sempre più l'adozione di soluzioni portabili.

Object Database Management Group

- ODMG (Object Data[base] Management Group) è uno dei working group di OMG, che consiste dei maggiori produttori di OODBMS (circa il 90% del mercato)
  - Data General, HP, Sun, Canon, American Airlines, Unisys, Philips, Prime, Gold Hill, SoftSwitch, 3 Com +1991 AT&T, Digital, NCR, Bull, IBM

Lo standard ODMG - scopo del consorzio

- Sviluppare una serie di standard per favorire portabilità, riusabilità e interoperabilità degli OODBMS commerciali
- Successo dei RDBMS legato all'esistenza di standard, differenze tra i modelli dei vari OODBMS sono un ostacolo alla loro diffusione
- ODMG nel contesto OO stesso ruolo di SQL in quello relazionale

ODMG - risultati

- 1993: ODMG 93 standard
  - [R. Cattell, The Object Database Standard: ODMG93, MorganKaufmann, 1993]
- 1997: ODMG 2.0 standard
  - [R. Cattell et al., The Object Database Standard: ODMG 2.0, MorganKaufmann, 1997]
- 1999: ODMG 3.0 standard
  - [R. Cattell et al., The Object Database Standard: ODMG 3.0, MorganKaufmann, 1999]

ODMG 3.0 - componenti

- Object Model
  (modello dei dati ad oggetti)
- Object Specification Language
  - Object Definition Language (ODL)
    (definizione astratta di interfacce, classi, …)
  - Object Interchange Format (OIF)
    (formato per scambiare oggetti fra diversi DB, fornire una descrizione esterna degli oggetti, …)
- Object Query Language (OQL) linguaggio di interrogazione dichiarativo (la base è SQL)
- Language Bindings, per C++, Smalltalk, Java

ODMG 3.0 - Object Model (OM)

Il modello a oggetti ODMG Object Model (OM)

- Le *proprietà* e le *operazioni* di un tipo sono chiamate *caratteristiche*
- E' possibile definire una gerarchia di tipi, alcuni dei quali sono *astratti* (non istanziabili)
- L'*estensione* di un tipo è l'insieme di tutte le sue istanze
  - Contenitore di istanze
- Un tipo ha una o più *implementazioni*
  - Ogni implementazione è detta "language binding".

Il modello a oggetti ODMG Object Model (OM)

- L'ODMG Object Model specifica i costrutti supportati da un OODBMS

- ODMG OM comprende due primitive di base per la modellazione delle informazioni
  - Oggetto (object)
  - Letterale (literal)

Il modello a oggetti ODMG Object Model (OM): Oggetti e letterali

L'**oggetto** (object) è caratterizzato da un identificatore univoco ed è costituito da:

- Uno **stato** rappresentato dai valori assunti dalle proprietà dell'oggetto in un certo momento
  - Le proprietà possono consistere in **attributi** (attribute) oppure in **relazioni** (relationship) tra l'oggetto considerato e altri oggetti
- Un **comportamento** definito attraverso un insieme di operazioni che possono essere eseguite sull'oggetto o da parte di esso

Il **letterale** (literal) non ha associato un identificatore univoco.

- Oggetti e letterali sono descritti attraverso il loro **tipo** (type)
- I letterali sono costanti (immutabili) mentre gli oggetti possono essere modificati
- Un oggetto è spesso detto **istanza** (instance) del suo tipo

Il modello a oggetti ODMG Object Model (OM): Oggetti e letterali

Oggetti

- Atomici
- Collection
  - Set
  - Bag
  - List
  - Array
- Strutturati
  - Date
  - Interval Time
  - Timestamp

Il modello a oggetti ODMG Object Model (OM): Oggetti e letterali

Letterali

- Atomici
  - Long, Short, Unsigned long, Unsigned short, Float, Double, Boolean, Octet, Char, String, Enum
- Collection
  - Set
  - Bag
  - List
  - Array
  - Dictionary
- Strutturati
  - Date
  - Interval time
  - Timestamp
  - Struct (*user-defined strucutures*)

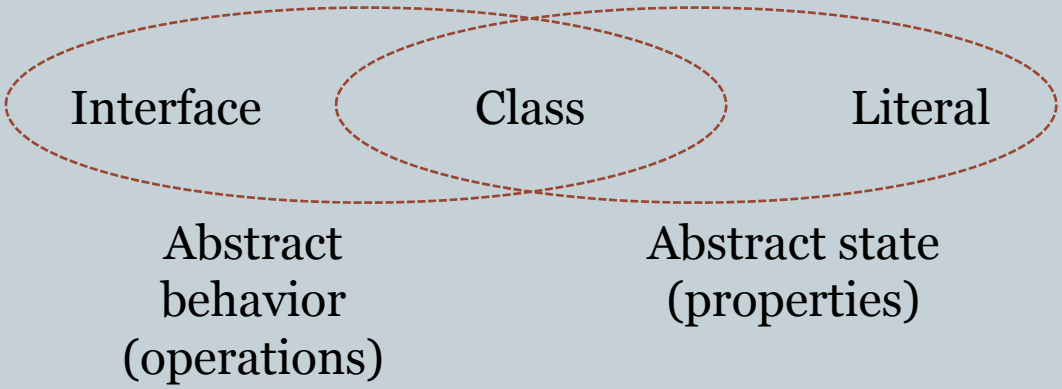
Il modello a oggetti ODMG Object Model (OM): Tipi e classi

- Tutti gli elementi di un dato tipo hanno
  - La stessa gamma di **stati** (stesse proprietà)
  - Un **comportamento** (behavior) comune (lo stesso insieme di operazioni)
- Ogni tipo è composto da
  - Una **specifica esterna** (“external specification”)
  - Una o più **implementazioni**

Il modello a oggetti ODMG Object Model (OM): Tipi e classi

Specifica esterna (definita in ODL) può consistere di:

- **Interface**: definisce le operazioni che possono essere invocate sull'oggetto di quel tipo
- **Class**: definisce le operazioni e lo stato di un tipo
- **Literal definition**: definisce solo lo stato astratto di un tipo letterale



Il modello a oggetti ODMG Object Model (OM): Tipi e classi

Interface: definisce la segnatura dei metodi (operazioni) invocabili sugli oggetti di quel tipo, cioè il comportamento astratto del tipo.

Un esempio di definizione di un'interfaccia in ODL è il seguente:

```
interface Triangolo{
    double getArea();
    double getPerimetro();
}
```

Il modello a oggetti ODMG Object Model (OM): Tipi e classi

Class: definisce le operazioni e lo stato di un tipo.

- Da notare la differenza tra il concetto di classe di un linguaggio object-oriented e il concetto di classe definito nello standard.
- Nel caso del modello ODMG il concetto di classe non fa riferimento all'implementazione dei metodi, ma viene utilizzato per definire l'insieme degli attributi e delle relazioni che caratterizzano un tipo, oltre eventualmente all'insieme delle operazioni applicabili agli oggetti di tale tipo.

Il modello a oggetti ODMG Object Model (OM): Tipi e classi

Un esempio di definizione di classe in ODL è il seguente:

```
class Automobile{
    attribute string targa;
    attribute string modello:
    attribute string colore;
    attribute integer prezzo;

    relationship <Costruttore> costruttore inverse
                 Costruttore::costruisce;}
```

Il modello a oggetti ODMG Object Model (OM): Tipi e classi

Le relazioni esprimibili attraverso il linguaggio ODL sono relazioni **binarie** tra oggetti e la loro definizione richiede di specificare sempre l'inversa in entrambi gli oggetti coinvolti. Quindi nella classe `Costruttore` si avrà una dichiarazione del tipo:

```
class Costruttore{
    ...
    relationship Set<Automobile> costruisce inverse
                 Automobile::costruttore;}
```

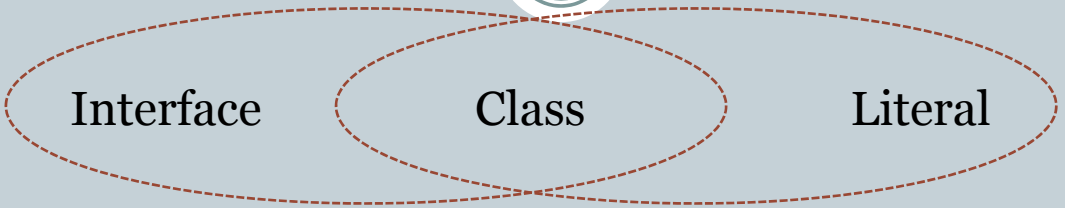
Il modello a oggetti ODMG Object Model (OM): Tipi e classi

Quindi la specifica completa di una relazione nel linguaggio ODL richiede di precisare la definizione della relazione due volte:

```
relationship <Costruttore> costruttore inverse
                    Costruttore::costruisce;}

relationship Set<Automobile> costruisce inverse
                    Automobile::costruttore;}
```

Dal momento che gli oggetti coinvolti da una relazione vengono riferiti tramite OID, risulta chiaro che i letterali non possono essere coinvolti in relazioni.

Il modello a oggetti ODMG Object Model (OM): Tipi e classi



Abstract behavior (operations)

Abstract state (properties)

```
Interface Employee{…};
```

Definisce solo il comportamento astratto degli oggetti `Employee`

```
Class Person{…};
```

Definisce sia il comportamento astratto che lo stato astratto degli oggetti `Person`

```
Struct complexNum
  {float re; float im;};
```

Definisce solo lo stato astratto dei letterali `complexNum`

Le interfacce definiscono dei tipi astratti non direttamente istanziabili.
Le classi definiscono dei tipi concreti direttamente istanziabili.

Il modello a oggetti ODMG Object Model (OM): Implementazione

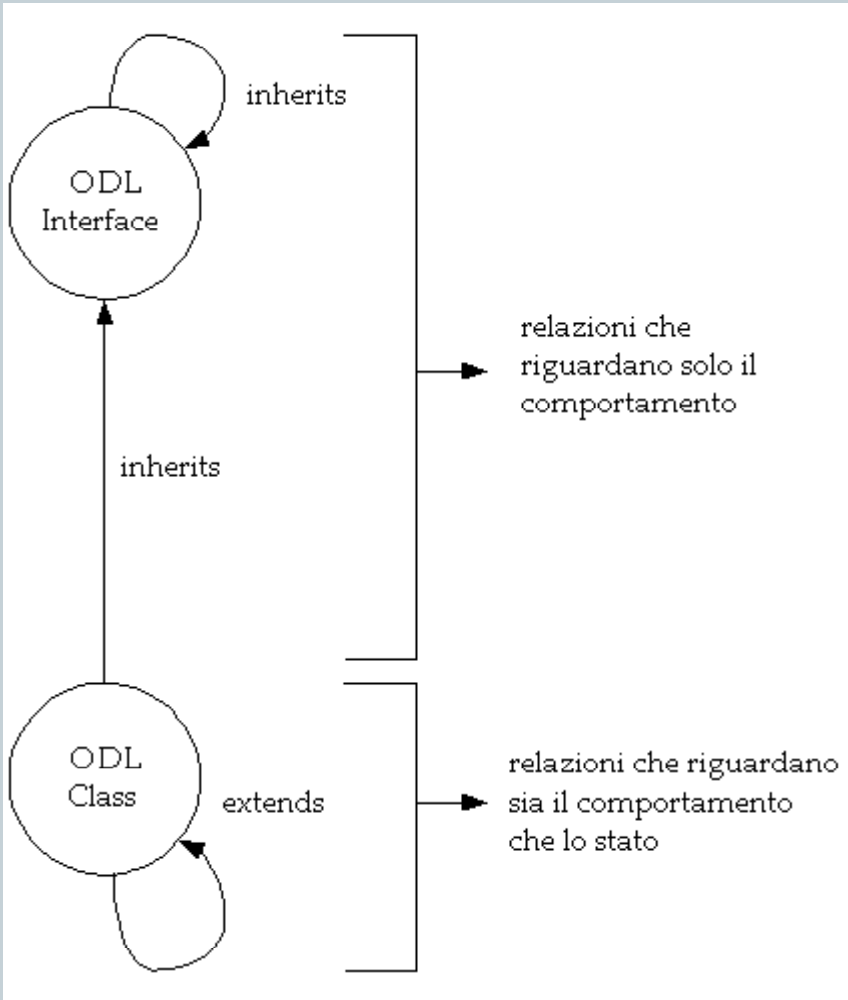
Implementazione di un tipo ("language binding") consiste di una **rappresentazione** e di un insieme di **metodi**.

- La **rappresentazione** è una struttura dati
- Un **metodo** è corpo di procedura

- È presente una variabile di un tipo opportuno per ogni proprietà che descrive lo stato di un tipo
- È presente un metodo per ogni operazione definita nell'interfaccia del tipo
  - Un metodo implementa il comportamento esternamente visibile dell'operazione ad esso associata

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

- Compatibilmente con la nozione di ODL class e ODL interface definiti dallo standard, sono stati definiti due tipi di ereditarietà
  - ereditarietà legata al comportamento
  - ereditarietà legata allo stato



Il modello a oggetti ODMG Object Model (OM): Ereditarietà

- L'ereditarietà legata al comportamento fa riferimento all'ereditarietà delle operazioni definite per un certo tipo (relazione ISA)
- ODMG supporta l'ereditarietà multipla però non sono permessi conflitti tra nomi di operazioni
- L'ereditarietà legata al comportamento si ha quando una classe eredita da un'interfaccia, oppure quando un'interfaccia eredita da un'altra interfaccia, questo tipo di ereditarietà viene definita attraverso l'operatore ":"

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

Subtyping ed ereditarietà legata al comportamento

```
interface Employee{...};
interface Professor : Employee{...};
interface AssociateProfessor : Professor{...};
```

- `Professor` è sotto-tipo di `Employee`
- `AssociateProfessor` è sotto-tipo di `Professor`

Il sotto-tipo più specifico di una gerarchia descrive tutti i comportamenti delle istanze di quel tipo, comprese quelli ereditati:

- Il tipo più specifico per un oggetto `AssociateProfessor` è l'interfaccia `AssociateProfessor`
- Un'istanza di `AssociateProfessor` è conforme a tutti i comportamenti definiti nell'interfaccia `AssociateProfessor`, nell'interfaccia `Professor` e in ogni suo supertipo.

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

Un sotto-tipo

- Eredita il comportamento dei suoi super-tipi
- Può specificare nuove operazioni rispetto a quelle ereditate
- Può specializzare comportamento ereditati
  - Polimorfismo: il comportamento è invocato run-time, rispetto al tipo dell'oggetto di volta in volta considerato

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

ODMG supporta l'ereditarietà multipla

- Un sotto-tipo può ereditare direttamente da più super-tipi

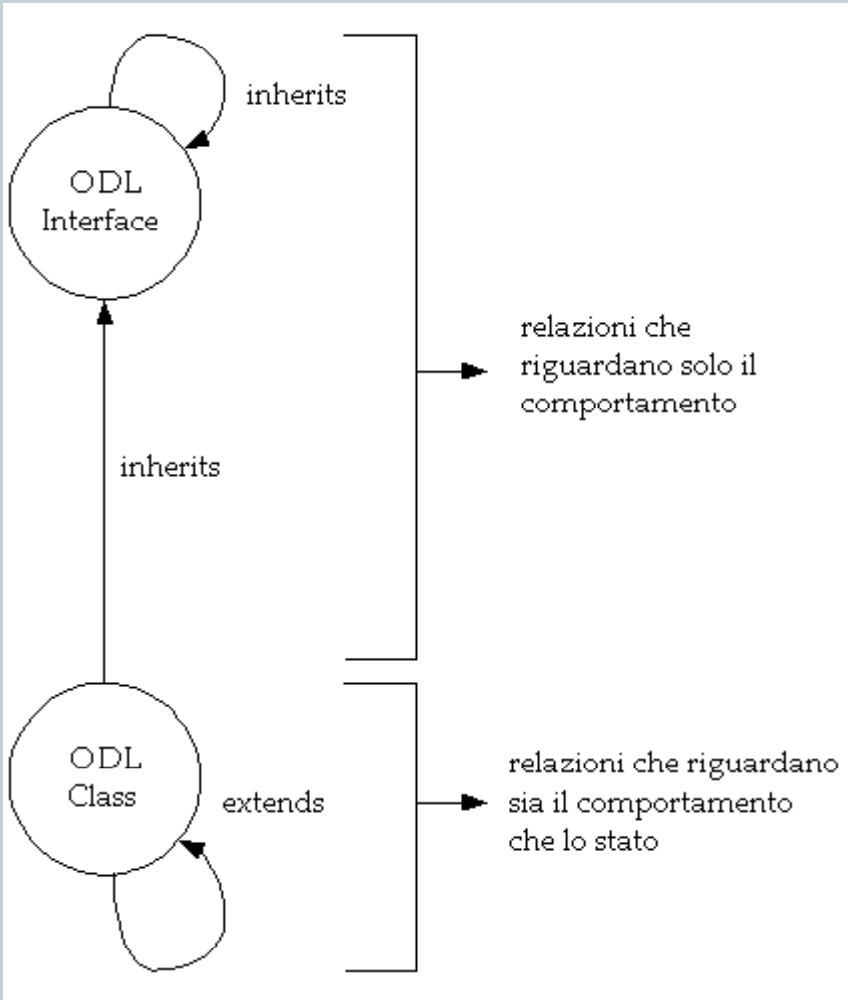
```
interface Employee{...};
interface Professor : Employee{...};
interface AssociateProfessor : Professor{...};
interface TeachingAssistant : Employee, Student{...};
```

Non sono permessi conflitti di nomi di operazioni

- Name overloading non permesso

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

- Le Interface possono ereditare (con il costrutto inherits ":") da interfacce ma non da classi
- Le classi possono ereditare da Interface (con il costrutto inherits ":") ma anche da altre classi (con il costrutto extends)



Il modello a oggetti ODMG Object Model (OM): Ereditarietà

- Le classi sono tipi concreti (direttamente istanziabili)
- Le interfacce sono tipi astratti (non direttamente istanziabili)

```
Interface Employee{...};
Class Salaried_Employee : Employee{...};
Class Hourly_Employee : Employee{....};
```

- Possono essere create istanze delle classi `Salaried_Employee` e `Hourly_Employee`, ma non della loro interfaccia super-tipo `Employee`.

Il modello a oggetti ODMG Object Model (OM): Ereditarietà


```
interface Employee{...};
interface Professor : Employee{...};
interface AssociateProfessor : Professor{...};
interface TeachingAssistant : Employee, Student{...};

class SalariedEmployee : Employee{....};
```

- Di solito si usa la convenzione di definire le operazioni esclusivamente nelle interfacce e di far ereditare tali interfacce alle classi
- Nelle classi saranno definiti esplicitamente solo attributi e relazioni
- Una classe può essere vista come un'interfaccia estesa con gli attributi che rappresentano lo stato dell'oggetto

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

- L'ereditarietà legata allo stato riguarda le proprietà degli oggetti e non può essere applicata ai letterali
- Si tratta di un'ereditarietà singola tra classi, nella quale la sotto-classe eredita tutte le proprietà della superclasse ed eventualmente ne aggiunge di nuove
- L'ereditarietà tra classi viene espressa mediante la parola chiave **extends**

Il modello a oggetti ODMG Object Model (OM): Ereditarietà

- La relazione **extends** si applica solo a tipi di oggetti (non ai letterali)

- Relazione di ereditarietà singola fra classi dove la classe subordinata eredita tutte le proprietà e tutte le operazioni della classe che viene estesa

Il modello a oggetti ODMG Object Model (OM): Ereditarietà


```
interface Employee{...};

class Person{
   attribute String name;
   attribute Date birthDate;
}

class EmployeePerson extends Person : Employee {
   attribute Date hireDate;
   attribute Currency payRate;
   relationship ManagerPerson boss inverse
   ManagerPerson::subordinates;
}

class ManagerPerson extends EmployeePerson {
   relationship Set<Employee> subordinates inverse
        EmployeePerson::boss;}
```

Relazione ISA
La classe `EmployeePerson` eredita il comportamento da `Employee`
Quindi istanze di `EmployeePerson` e `ManagerPerson` supportano il comportamento definito nell'interfaccia `Employee`

La relazione **`extends`** è transitiva, quindi ogni `ManagerPerson` avrà `name`, `birthDate`, `hireDate`, ...

Il modello a oggetti ODMG Object Model (OM): Estensione di un tipo

- **Extent**
  - L'extent di un tipo è l'insieme di tutte le istanze di quel tipo entro la base di dati considerata
    - Se un oggetto è istanza del tipo A, allora esso sarà un membro dell'extent di A.
    - Se il tipo A è sotto-tipo del tipo B, allora l'extent di A sarà un sotto-insieme dell'extent di B.

Il modello a oggetti ODMG Object Model (OM): Vincoli aggiuntivi

- Chiavi
  - In alcuni casi le istanze di un tipo possono essere individuate in base al valore di alcune proprietà
    - Simple key → single property
    - Compound key → set of properties

- Un tipo per avere una chiave deve avere una extent

```
interface Employee{...};

class Professor: Employee (extent professors key matricola)
{
   attribute String name;
   attribute String matricola;
   attribute Date birthDate;
}
```

Il modello a oggetti ODMG Object Model (OM): Oggetti

- Creazione dell'oggetto
- Identificatori degli oggetti
  - Usati da un OODBMS per distinguere gli oggetti e per trovarli
- Nomi di oggetti
  - Definiti dal programmatore o dall'utente finale per riferirsi a particolari oggetti
- Tempi di vita
  - Determina la gestione della memoria allocata per gli oggetti
- Struttura (atomica o no)

Il modello a oggetti ODMG Object Model (OM): Oggetti

- Creazione degli oggetti
  - avviene mediante l'invocazione di un metodo definito in un'interfaccia, detta **factory interface**, fornita dalla base di dati considerata

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

  il cui metodo `new()` crea un nuova istanza di un oggetto del tipo Object

- Tutti gli oggetti ereditano implicitamente dall'interfaccia **`Object`**

Il modello a oggetti ODMG Object Model (OM): Oggetti


```
interface Object {
enum          Lock_Type{read, write, upgrade};
exception     LockNotGranted{};
void          lock(in Lock_Type mode)
                    raises(LockNotGranted);
Boolean       try_lock(in Lock_Type mode);
boolean       same_as(in Object anObject);
Object        copy();
void          delete();
};
```

Eccezione tornata quando scade il TIME-OUT durante l'attesa di un lock

Gestione esplicita del lock sugli oggetti

Tenta di acquisire un lock

Confronto uguaglianza (identità) fra oggetti

Crea un nuovo oggetto equivalente all'oggetto ricevente.
L'oggetto creato non è uguale all'originale rispetto a "`same_as`"

Cancellazione dalla base di dati e dalla memoria

Il modello a oggetti ODMG Object Model (OM): Oggetti

- L'identificatore di un oggetto (OID) è unico e non cambia mai
- L'OID è generato dal sistema di gestione della base di dati ed assegnato in modo trasparente a ciascun oggetto.
  L'utente non può né modificarlo né usarlo per eseguire interrogazioni.
- L'OID accompagna l'oggetto per tutta la sua vita e di norma l'identificatore di un oggetto cancellato non viene riutilizzato per identificare altri oggetti.
- La rappresentazione dell'identità di un oggetto è ottenuta attraverso l'identificatore dell'oggetto
- Possono cambiare valore gli attributi di un oggetto e le sue relazioni con altri oggetti; ma l'OID di un oggetto non muta mai il suo valore
  - L'oggetto resta lo stesso oggetto
- Contrariamente al concetto di chiave primaria nelle basi di dati relazionali, l'OID non è legato al valore di alcuna proprietà dell'oggetto, pertanto esiste un concetto di identità degli oggetti slegato dal valore delle loro proprietà.

Il modello a oggetti ODMG Object Model (OM): Oggetti

- Un oggetto può avere uno o più *nomi*, significativi al programmatore o all'utente finale.
  - I nomi non sono chiavi ma sono unici
- Il nome rappresenta un modo per accedere in modo semplificato agli oggetti più importanti ("radici") della base di dati e possono essere utilizzati dall'utente durante le interrogazioni
- L'ODBMS gestisce la corrispondenza fra nomi e *OID*
- Il nome non è definito in alcuna interfaccia e non corrisponde a valori di proprietà.
- L'assegnamento di un nome ad un oggetto è uno dei meccanismi attraverso i quali si ottiene la persistenza degli oggetti
- Non tutti gli oggetti hanno un nome

Il modello a oggetti ODMG Object Model (OM): Oggetti

- Con il termine chiave si indica una o più proprietà il cui valore consente di individuare univocamente un oggetto, in piena analogia con il concetto di chiave primaria delle basi di dati relazionali.
- Se la chiave è formata da una singola proprietà si parla di *simple key*, altrimenti di *compound key*.

- Da notare che contrariamente all'OID e al nome, la chiave dipende dal valore assunto dalle proprietà dell'oggetto

Il modello a oggetti ODMG Object Model (OM): Oggetti

- Il *tempo di vita* di un oggetto determina come deve essere gestita la memorizzazione dell'oggetto stesso ed è definita alla creazione dell'oggetto
  - Transiente
  - Persistente
- Il tempo di vita di un oggetto è indipendente dal suo tipo
  - Un TIPO può avere alcune istanze TRANSIENTI e alcune PERSISTENTI

    Differenza rispetto ai RDBMS in cui TUTTO è persistente

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti

- Collezioni di oggetti
  - I tipi atomici definiti dall'utente possono essere combinati mediante l'utilizzo di generatori di tipo.
  - In particolare, nello standard ODMG sono previsti i seguenti generatori:
    - Set<t>
    - Bag<t>
    - List<t>
    - Array<t>

    che ereditano tutti dall'interfaccia Collection

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti


```
interface Collection: Object {
exception           InvalidCollectionType{};
exception           ElementNotFound{any element; };
unsigned long       cardinality();
boolean             is_empty();
boolean             is_ordered();
boolean             allows_duplicates();
boolean             contains_element(in Object element);
void                insert_element(in Object element);
void                remove_element(in Object element)
                          raises(ElementNotFound);
Iterator            create_iterator(in boolean stable);
BidirectionalIterator  create_bidirectional_iterator(in boolean stable)
                            raises(InvalidCollectionType);
Object              select_element(in string OQL_predicate);
Iterator            select(in string OQL_predicate);
boolean             query(in string OQL_predicate, inout Collection result );
};
```

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti


```
interface Collection: Object {
…
…
…
};
```

Oltre alle operazioni definite nell'interfaccia `Collection`, le collezioni ereditano le operazioni definite nell'interfaccia `Object`

`Same_as`: verifica dell'identità

`Copy`: ritorna una nuova collezione di oggetti i cui elementi sono gli stessi della collezione originale

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti


```
interface Collection: Object {
exception       InvalidCollectionType{};
exception       ElementNotFound{any element; };

unsigned long  cardinality();

boolean         is_empty();
boolean         is_ordered();
boolean         allows_duplicates();
…
…
…
};
```

Ritorna il numero di elementi contenuti nella collezione

Metodi che verificano le caratteristiche della collezione

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti


```
interface Collection: Object {
…
…

boolean       contains_element(in Object element);
void          insert_element(in Object element);
void          remove_element(in Object element)
                    raises(ElementNotFound);

…
…

};
```

Metodi per la gestione degli elementi della collezione

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti


```
interface Collection: Object {
…
…

Iterator
   create_iterator(in boolean stable);
BidirectionalIterator
   create_bidirectional_iterator(in boolean stable)
       raises(InvalidCollectionType);

…
…
};
```

Metodi per il supporto allo scorrimento degli elementi della collezione

Il modello a oggetti ODMG Object Model (OM): Collezioni di oggetti


```
interface Collection: Object {
…
…

Object          select_element(in string OQL_predicate);
Iterator        select(in string OQL_predicate);
boolean         query(in string OQL_predicate,
                      inout Collection result );
};
```

Metodi per la valutazione di predicati OQL sul contenuto della collezione

Il modello a oggetti ODMG Object Model (OM): Collezione di oggetti


```
interface Collection: Object {
…
…
Object          select_element(in string OQL_predicate);
Iterator        select(in string OQL_predicate);
boolean         query(in string OQL_predicate,
                      inout Collection result );
};
```

Restituisce l'oggetto che soddisfa il predicato

Ritorna TRUE se il predicato è soddisfatto, FALSE altrimenti

Restituisce un puntatore che consente di iterare il risultato dal primo oggetto trovato

Il modello a oggetti ODMG Object Model (OM): Iteratore

- L'interfaccia `Iterator`, utilizzata nella definizione di `Collection`, consente di accedere agli elementi della collezione
- Alla creazione di un iteratore il puntatore viene posto sul primo elemento della collezione
- La stabilità di un iteratore determina se una iterazione è sicura rispetto ai cambiamenti fatti ad una collezione durante l'iterazione stessa
  - Un iteratore è STABILE se garantisce che le modifiche apportate alla collezione durante l'iterazione non abbiano alcun effetto sullo scorrimento in corso
  - Se un iteratore non è stabile, i cambiamenti apportati alla collezione possono causare
    - Elementi mancanti
    - Doppio processamento di un elemento

Il modello a oggetti ODMG Object Model (OM): Iteratore

Supporta lo scorrimento solo in avanti di tutte le collezioni

```
interface Iterator{
exception     NoMoreElements{};
exception     InvalidCollectionType{};
boolean       is_stable();
boolean       at_end();
void          reset();
void          next_position() raises(NoMoreElements);
Object        get_element() raises(NoMoreElements);
void          replace_element(in Object element)
                    raises(InvalidCollectionType);
};
```

Il modello a oggetti ODMG Object Model (OM): Iteratore


```
interface Iterator {
…
…
void    reset();

void     next_position() raises(NoMoreElements);

Object  get_element() raises(NoMoreElements);

void     replace_element(in Object element)
              raises(InvalidCollectionType);
};
```

Riposiziona il puntatore sul primo elemento

Incrementa il puntatore passando all'elemento successivo

Recupera l'elemento puntato

Sostituisce l'elemento puntato con quello passato per argomento (valido su LIST e ARRAY)

Il modello a oggetti ODMG Object Model (OM): Set


```
interface SetFactory : ObjectFactory{
  Set new_of_size( in long size );
}

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);
Boolean       is_superset_of(in Set other_set);
Boolean       is_ proper_superset_of(in Set other_set);
};
```

Collezione NON ordinata di elementi che NON ammette duplicati

Il modello a oggetti ODMG Object Model (OM): Set


```
interface SetFactory : ObjectFactory{
  Set new_of_size( in long size );
}

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);

…
…
…
};
```

Operazioni convenzionali su insiemi

Il modello a oggetti ODMG Object Model (OM): Set


```
interface SetFactory : ObjectFactory{
  Set new_of_size( in long size );
}

Class Set: Collection {
…
…

Boolean     is_subset_of(in Set other_set);
Boolean     is_proper_subset_of(in Set other_set);
Boolean     is_superset_of(in Set other_set);
Boolean     is_ proper_superset_of(in Set other_set);
};
```

Confronti convenzionali tra insiemi

Il modello a oggetti ODMG Object Model (OM): Set


```
interface SetFactory : ObjectFactory{
  Set new_of_size( in long size );
}

Class Set: Collection {
…
…
…
};
```

`Set` raffina la semantica del metodo `insert_element` ereditato dal super-tipo `Collection` nel seguente modo:
se l'oggetto passato come argomento ad `insert_element` è già presente nell'insieme la collezione non viene modificata, altrimenti l'oggetto viene aggiunto in una qualsiasi posizione

Il modello a oggetti ODMG Object Model (OM): Bag


```
interface BagFactory : ObjectFactory{
  Bag new_of_size( in long size );
}

Class Bag: Collection {
Attribute       bag<t> value;
unsigned long   occurrences_of(in Object element);
Bag             create_union(in Bag other_bag);
Bag             create_interesection(in Bag other_bag);
Bag             create_difference(in Bag other_bag);
};
```

Collezione NON ordinata di elementi che ammette duplicati

Il modello a oggetti ODMG Object Model (OM): Bag


```
interface BagFactory : ObjectFactory{
  Bag new_of_size( in long size );
}

Class Bag: Collection {
Attribute       bag<t> value;
unsigned long   occurrences_of(in Object element);

Bag             create_union(in Bag other_bag);
Bag             create_interesection(in Bag other_bag);
Bag             create_difference(in Bag other_bag);
};
```

Calcola il numero di occorrenze di un certo elemento

Unione, intersezione e differenza di `Bag`. Ritornano un nuovo `Bag`

Il modello a oggetti ODMG Object Model (OM): Bag


```
interface BagFactory : ObjectFactory{
  Bag new_of_size( in long size );
}

Class Bag: Collection {
…
…
…
};
```

`Bag` raffina la semantica del metodo `insert_element` ereditato dal super-tipo `Collection` nel seguente modo:
Inserisce l'elemento passato come argomento e se già presente incrementa la sua molteplicità.
`Bag` raffina anche la semantica del metodo `remove_element` ereditato dal super-tipo `Collection` nel seguente modo:
Rimuove una occorrenza e decrementa la molteplicità.

Il modello a oggetti ODMG Object Model (OM): List


```
interface ListFactory : ObjectFactory{
  List new_of_size( in long size );
}

Class List: Collection {
Exception   InvalidIndex{unsigned long index; };
Attribute   list<t> value;
void         replace_element_at(in Object element,
                                in unsigned long index)
                    raises(InvalidIndex);
void         remove_element_at(in unsigned long index)
                    raises(InvalidIndex);
Object       retrieve_element_at(in unsigned long index)
                    raises(InvalidIndex);
…
};
```

Collezione ordinata di elementi che ammette duplicati

Il modello a oggetti ODMG Object Model (OM): List


```
interface ListFactory : ObjectFactory{
  List new_of_size( in long size );
}

Class List: Collection {
…
void   insert_element_after(in Object element,
                            in unsigned long index)
             raises(InvalidIndex);
void   insert_element_before(in Object element,
                             in unsigned long index)
             raises(InvalidIndex);
void   insert_element_first(in Object element);
void   insert_element_last(in Object element);
…
};
```

Le operazioni definite per `List` sono POSIZIONALI per natura e si riferiscono ad un dato indice o all'inizio/fine della lista

Il modello a oggetti ODMG Object Model (OM): List


```
interface ListFactory : ObjectFactory{
  List new_of_size( in long size );
}

Class List: Collection {
…
void          remove_first_element()
                    raises(ElementNotFound);
void          remove_last_element()
                    raises(ElementNotFound);
Object        retrieve_first_element()
                    raises(ElementNotFound);
Object        retrieve_last_element()
                    raises(ElementNotFound);
…
};
```

Il modello a oggetti ODMG Object Model (OM): List


```
interface ListFactory : ObjectFactory{
  List new_of_size( in long size );
}

Class List: Collection {
…
List   concat(in List other_list);

void   append(in List other_list);
};
```

Ritorna una nuova lista che contiene la lista passata come parametro CONCATENATA (concat) alla lista ricevente

Modifica la lista ricevente APPENDENDO (append) la lista passata come parametro

Il modello a oggetti ODMG Object Model (OM): List


```
interface ListFactory : ObjectFactory{
  List new_of_size( in long size );
}

Class List: Collection {
…
…
};
```

`List` raffina la semantica del metodo `insert_element` ereditato dal super-tipo `Collection` nel seguente modo:
Inserisce l'elemento passato come argomento alla fine della lista (equivalente a `insert_element_last`).
`List` raffina anche la semantica del metodo `remove_element` nel seguente modo:
Rimuove la prima occorrenza dell'elemento specificato

Il modello a oggetti ODMG Object Model (OM): Array


```
interface ArrayFactory : ObjectFactory{
   Array new_of_size( in long size );
}

Class Array: Collection {
exception       InvalidSize{unsigned long size; };
exception       InvalidIndex{unsigned long index; };
Attribute       array<t> value;
void            replace_element_at(in unsigned long index,
                                   in Object element)
                       raises(InvalidIndex);
void            remove_element_at(in unsigned long index)
                       raises(InvalidIndex);
Object          retrieve_element_at(in unsigned long index)
                       raises(InvalidIndex);
void            resize(in unsigned long new_size)
                       raises(InvalidSize);
};
```

Collezione ordinata e dimensionata dinamicamente i cui elementi hanno una posizione che li identifica

Il modello a oggetti ODMG Object Model (OM): Array


```
interface ArrayFactory : ObjectFactory{
  Array new_of_size( in long size );
}

Class Array: Collection {
…
void   replace_element_at(in unsigned long index,
                          in Object element)
                raises(InvalidIndex);
void   remove_element_at(in unsigned long index)
                raises(InvalidIndex);
…
};
```

Assegna un valore nullo alla cella in posizione `index`.
Non rimuove la cella. Posizione ed indici non cambiano.
Non modifica la dimensione dell'array.

Il modello a oggetti ODMG Object Model (OM): Array


```
interface ArrayFactory : ObjectFactory{
  Array new_of_size( in long size );
}

Class Array: Collection {
…
Object  retrieve_element_at(in unsigned long index)
                raises(InvalidIndex);
void    resize(in unsigned long new_size)
                raises(InvalidSize);
};
```

Ridimensiona l'array.
Ritorna l'eccezione `InvalidSize` se la dimensione `new_size` è più piccola del numero di elementi nell'array.

Il modello a oggetti ODMG Object Model (OM): Array


```
interface ArrayFactory : ObjectFactory{
  Array new_of_size( in long size );
}

Class Array: Collection {


…
…
};
```

`Array` raffina la semantica del metodo `insert_element` ereditato dal super-tipo `Collection` nel seguente modo:
Incrementa la dimensione dell'array di `1` e inserisce l'elemento nella nuova posizione (in coda).
`Array` raffina anche la semantica del metodo `remove_element` nel seguente modo:
Sostituisce la prima occorrenza dell'elemento specificato con un valore nullo

Il modello a oggetti ODMG Object Model (OM): Oggetti strutturati

- Oggetti strutturati
  - Il modello supporta alcuni oggetti strutturati, in particolare:
    - Date
    - Interval
    - Time
    - Timestamp

    per ognuno dei quali è stata definita una particolare interfaccia.

Il modello a oggetti ODMG Object Model (OM): Oggetti strutturati

- Interfaccia che definisce le operazioni per la produzione di oggetti DATE

```
interface DateFactory : ObjectFactory{
  exception InvalidDate{};
  Date julian_date(in unsigned short year,
                   in unsigned short julian_day)
       raises(InvalidDate);
  Date calendar_date(in unsigned short year,
                     in unsigned short month,
                     in unsigned short day)
       raises(InvalidDate);
  ...
};
```

Il modello a oggetti ODMG Object Model (OM): Oggetti strutturati


```
interface DateFactory : ObjectFactory{
    ...
  boolean is_leap_year(in unsigned short year);
  boolean is_valid_date(in unsigned short year,
                        in unsigned short month,
                        in unsigned short day);
  unsigned short days_in_year(in unsigned short year);
  unsigned short days_in_month(in unsigned short year,
                               in Date::Month month);
  Date current();
};
```

Il modello a oggetti ODMG Object Model (OM): Letterali

- I tipi letterali supportati dal modello possono essere classificati in:
  - Atomic literal: numeri e caratteri (es. long, short, double, char, string, …)
  - Collection literal: set, bag, list, array
  - Structured literal: data, interval, time, timestamp, dati definiti dall'utente mediante il costrutto struct
- La caratteristica fondamentale dei letterali è che non hanno OID
  - pertanto non possono essere condivisi tra oggetti diversi ma possono essere solo copiati
  - non è possibile eseguire confronti rispetto all'identità

Il modello a oggetti ODMG Object Model (OM): Letterali

- Aspetti considerati
  - Types: descrizione dei tipi di letterali supportati dallo standard
  - Copying: modo in cui i letterali vengono copiati
  - Comparing: modo in cui i letterali vengono confrontati

Il modello a oggetti ODMG Object Model (OM): Letterali

- Atomic literal supportati dal modello
  - long
  - short
  - unsigned long
  - unsigned short
  - float
  - double
  - boolean
  - octet
  - char (abbreviato per character)
  - string
  - **enum** (abbreviato per enumeration)

Il modello a oggetti ODMG Object Model (OM): Letterali

- Atomic literal
  - Istanze di questi tipi non sono create esplicitamente ma esistono implicitamente
  - L'idea del modello OM è che con il BINDING ognuno di questi tipi venga supportato dall'analogo tipo definito dal linguaggio di programmazione considerato

Il modello a oggetti ODMG Object Model (OM): Letterali

- Collection literal
  - set<t>
  - bag<t>
  - list<t>
  - array<t>

- Generatori di tipi analoghi a quelli delle collezioni
  - Queste collezioni
    - Non hanno OID
    - I loro elementi possono essere di tipo literal o di tipo object

Il modello a oggetti ODMG Object Model (OM): Letterali

- Structured literal (structure)
  - date
  - interval
  - time
  - timestamp
  - strutture definite dall'utente:

    struct<>
- Una STRUCTURE ha un numero fissato di elementi
  - Ogni elemento ha un nome e può contenere
    - Un letterale
    - Un oggetto

Il modello a oggetti ODMG Object Model (OM): Letterali

Structured literal (user-defined structure)

- Esempio:

```
struct Address{
        string street;
        string city;
};

Address.city = "Verona"
```

Il modello a oggetti ODMG Object Model (OM): Letterali

- User-Defined Structure: altro esempio

```
struct Degree{
        string school_name;
        string degree_name;
        unsigned short degree_year;
};

Typedef list<Degree> Degrees;
```

- Le strutture definite possono essere composte liberamente
  - OM supporta
    - insiemi di strutture
    - strutture di insiemi
    - array di strutture
    - ...

Il modello a oggetti ODMG Object Model (OM): Letterali

- Copying literals
  - I letterali non hanno OID e quindi non possono essere condivisi
  - I letterali hanno comunque la possibilità di essere copiati
  - Scorrendo una collezione di letterali si ottengono copie degli elementi stessi

Il modello a oggetti ODMG Object Model (OM): Letterali

- Comparing literals
  - Non avendo OID non possono essere confrontati rispetto all'identità (`same_as`)
  - Possono essere confrontati con l'operazione `equals`
    - Nella gestione delle collezioni per inserire, rimuovere o verificare la presenza di un certo letterali di usa `equals` e non `same_as`

Il modello a oggetti ODMG Object Model (OM): Modellazione dello stato - Proprietà

- Una classe definisce un insieme di proprietà (accessibili e, in qualche caso manipolabili, dall'utente)
- ODMG OM definisce due tipi di proprietà
  - Attributi
    - Un attributo ha un tipo
  - Relazioni
    - Una relazione è definita tra due tipi, ognuno dei quali deve avere istanze referenziabili con OID
      - I letterali non possono essere coinvolti da relazioni

Il modello a oggetti ODMG Object Model (OM): Modellazione dello stato - Proprietà

Attributi

- La dichiarazione degli attributi in una classe definisce lo STATO ASTRATTO delle sue istanze

```
Class Person {
attribute short age;
attribute string name;
attribute enum E_gender{male, female} gender;
attribute Address home_address;
attribute set<Phone_no> phones;
attribute Department dept;
};
```

OID del dipartimento di afferenza

- Una particolare istanza di `Persona` avrà un certo valore (literal o object) per ogni attributo definito

Il modello a oggetti ODMG Object Model (OM): Modellazione dello stato - Proprietà

- La dichiarazione di un attributo in una interfaccia definisce solo il comportamento astratto delle sue istanze.
- Un attributo può essere implementato come struttura dati (usuale) o anche come metodo.
  - Esempio: se l'attributo `age` è definito in una interfaccia, la presenza di questo attributo potrebbe rappresentare più che lo stato, l'abilità di calcolare l'età

```
Interface i_Person {
  attribute      short age;
};

Class Person : i_Person {
  attribute      Date birthdate;
  …
};
```

- *Stabiliamo per "convenzione nostra" di non istanziare attributi in una Interface.*

Il modello a oggetti ODMG Object Model (OM): Modellazione dello stato - Proprietà

Relazioni

- Sono definite tra tipi
- Sono simili alle relazioni del modello E-R
- ODMG OM supporta solo relazioni binarie tra due tipi
  - Relazioni uno a uno
  - Relazioni uno a molti
  - Relazioni molti a molti
- Una relazione viene definita esplicitamente dichiarando traversal path che permettono la connessione logica tra oggetti partecipanti alla relazione (*a professor teaches courses and a course is taught by a professor*)

Il modello a oggetti ODMG Object Model (OM): Modellazione dello stato - Proprietà


```
Class Professor {
  …
  relationship Set<Course> teaches
  inverse Course::is_taught_by;
  …
};

Class Course {
  …
  relationship Professor is_taught_by
  inverse Professor::teaches;
  …
};
```

Relazione uno a molti. Con `Set` faccio riferimento a MOLTI oggetti non ordinati

Dichiarazione di traversal path

Il modello a oggetti ODMG Object Model (OM): Modellazione dello stato - Proprietà

Relazioni

- Un OODBMS deve garantire l'integrità referenziale delle relazioni
  - Se un oggetto che partecipa ad una relazione viene cancellato, tutti i cammini che portano all'oggetto vanno rimossi

Il modello a oggetti ODMG Object Model (OM): Modellazione del comportamento - Operazioni

- Il comportamento di un tipo viene definito per mezzo di un insieme di operation signatures
- Ogni operation signature definisce
  - Il nome di una operazione
  - Il tipo e il nome di ogni argomento
  - Il tipo di valore tornato
  - Il nome delle eccezioni tornate in caso d'errore
- Ogni operazione è definita su un singolo tipo
  - Una operazione non può esistere indipendentemente da un tipo
  - Una operazione non può essere definita su più tipi
  - Si può avere lo stesso nome per operazioni definite su tipi diversi (overload)
    - Il nome deve essere unico rispetto alla definizione del singolo tipo

Il modello a oggetti ODMG Object Model (OM): Modellazione del comportamento - Operazioni

- Eccezioni gestite attraverso il tipo predefinito `exception`

```
interface Iterator {
exceptionNoMoreElements{ };
…
void      next_position() raises(NoMoreElements);
Object    get_element() raises(NoMoreElements);
…
};
```

Il modello a oggetti ODMG Object Model (OM): Transazioni

La gestione delle transazioni è una funzionalità fondamentale dell'OODBMS per garantire

- L'integrità dei dati
- La loro condivisione
- L'eventuale recupero

Ogni:

- Accesso
- Creazione
- Modifica
- Rimozione

di un oggetto persistente deve essere garantito da una transazione

Il modello a oggetti ODMG Object Model (OM): Transazioni

- Per ogni transazione l'OODBMS deve garantire
  - Atomicity

    La transazione termina o non ha effetto
  - Consistency

    La transazione parte da uno stato consistente del DB e termina in un altro stato consistente
  - Isolation

    Nessun altro utente vede le modifiche fatte dalla transazione prima del COMMIT
  - Durability

    L'effetto della transazione viene preservato

Il modello a oggetti ODMG Object Model (OM): Transazioni


```
interface TransactionFactory{
  Transaction new();
  Transaction current();
}

interface Transaction{
  exception   TransactionInProgress{};
  exception   TransactionNotInProgress{};
  void        begin()
                raises(TransactionInProgress);
  void        commit()
                raises(TransactionNotInProgress );
  void        abort()
                raises( TransactionNotInProgress );
  void        checkpoint()
                raises( TransactionNotInProgress );
  boolean isOpen();
}
```

Per creare transazioni

Per manipolare transazioni

Il modello a oggetti ODMG Object Model (OM): Transazioni


```
interface Transaction{
  …

  void commit() raises(TransactionNotInProgress);

  void abort() raises(TransactionNotInProgress);

  void checkpoint() raises(TransactionNotInProgress);
  boolean isOpen();
}
```

Tutti gli oggetti persistenti creati o modificati durante la transazione vengono memorizzati nel ODB

Equivalente a COMMIT seguito da BEGIN.
Fa COMMIT e tiene aperta la transazione

Il modello a oggetti ODMG Object Model (OM): Transazioni

- Il modello ODMG OM usa un convenzionale approccio basato sui lock per il controllo della concorrenza
- L'approccio basato sui lock fornisce un meccanismo per l'accesso condiviso o esclusivo agli oggetti
- Lock supportati
  - Read: accesso condiviso
  - Write: accesso esclusivo
  - Upgrade: per prevenire il deadlock dovuto a due processi che dopo aver ottenuto il lock in lettura, tentano di ottenere quello in scrittura sullo stesso oggetto

Il modello a oggetti ODMG Object Model (OM): Transazioni

- Un OODBMS può gestire uno o più database
- Ogni DB è un'istanza del tipo `Database`
- Istanze del tipo `Database` sono create usando

```
Interface DatabaseFactory{
    Database new();
}
```

Il modello a oggetti ODMG Object Model (OM): Operazioni sul DB

- Quando l'oggetto Database è stato creato, può essere manipolato usando

```
interface Database {
exception   DatabaseOpen{};
exception   DatabaseNotFound{};
exception   ObjectNameNotUnique{};
exception   ObjectNameNotFound{};
void        open(in string database_name)
              raises(DatabaseNotFoudn);
void        close();
void        bind(in Object an_object, in string_name);
Object      unbind(in string_name);
Object      lookup(in string object_name);
ODLMetaObjects::Module    schema();
};
```

ODMG Object Definition Language (ODL)

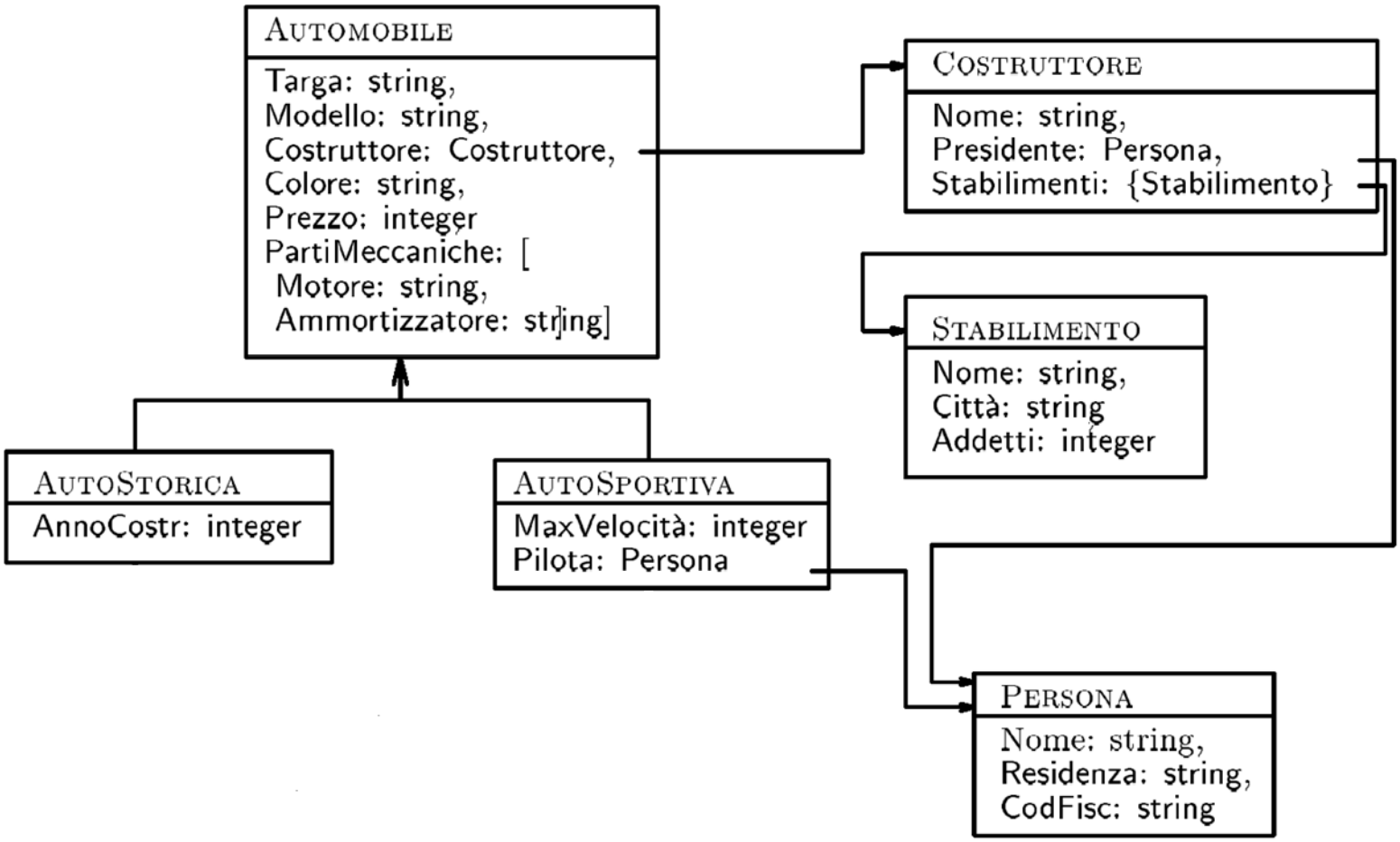
ODMG Object Definition Language

- Linguaggio per la definizione astratta di interfacce, letterali, classi, transazioni, eccezioni, …
  - ODL supporta i costrutti semantici di ODMG OM
- ODL non è legato ad alcun linguaggio di programmazione
- ODL non è inteso come un linguaggio di programmazione, ma un linguaggio di definizione per la specifica di oggetti

ODMG Object Definition Language

- ODL è un Data Definition Language per tipi di oggetti
  - ODL permette di definire le caratteristiche dei tipi: proprietà ed operazioni
  - ODL permette di definire solo la segnatura delle operazioni
- ODL consente di definire tipi di oggetti che possono essere implementati con linguaggi di programmazione diversi

ODL: Esempio 1



ODL: Esempio 1


```
struct T_PartiMec
            {String Motore,
             String Ammortizzatore};
class Automobile
( extent Automobili  key Targa )
 {     attribute String Targa;
       attribute String Modello;
       attribute String Colore;
       attribute String Prezzo;
       attribute T_PartiMec PartiMeccaniche;
       relationship <Costruttore> Costruttore
             inverse Costruttore::Costruisce;
}
```

ODL: Esempio 1


```
class AutoStorica extends Automobile
 {     attribute short AnnoCostr;
  }

class AutoSportiva extends Automobile
 {     attribute Integer MaxVelocità;
       relationship Set<Persona> Pilota
              inverse Persona::Ha_Pilotato;
  }
```

ODL: Esempio 1


```
class Costruttore
 {     attribute String Nome;
       relationship <Persona> Presidente
             inverse Persona::Presidenza;
       relationship Set<Automobile> Costruisce
             inverse Automobile::Costruttore;
       relationship Set<Stabilimento> Stabilimenti
             inverse Stabilimento::StabCostruzione;

}
```

ODL: Esempio 1


```
class Stabilimento
 {      attribute String Nome;
        attribute String Città;
        attribute long Addetti;
        relationship <Costruttore> StabCostruzione
                inverse Stabilimento::Stabilimenti;
}

class Persona
 {      attribute String Nome;
        attribute String Residenza;
        attribute String CodiceFiscale;
        relationship <Costruttore> Presidenza
                inverse Costruttore::Presidente;
        relationship Set<AutoSportiva> Ha_pilotato
                inverse AutoSportiva::Pilota;
}
```

ODL: Esempio 2

- Facciamo riferimento ad una schema che descriva:
  - Documenti: con chiave “titolo”. Ogni documento ha un titolo, un insieme di autori che sono impiegati ed è descritto dallo stato e dal contenuto
  - Articoli: sono tipi particolari di documenti per cui si memorizza la rivista su cui sono stati pubblicati e la data di pubblicazione
  - Progetti: con chiave “nome”. Ogni progetto ha un nome e viene descritto da un insieme di documenti. Inoltre, è composto da una insieme di task. Ad ogni progetto sono assegnati un insieme di impiegati ed un capo (sempre impiegato).
  - Task: ogni task viene descritto dai mesi uomo necessari per completarlo, dalla data di inizio e di fine e dal nome dell’impiegato responsabile del task.

ODL: Esempio 2


```
class Documento
   (extent documenti key titolo)
   {
        attribute string titolo;
        attribute List<Impiegato> autori;
        attribute string stato;
        attribute string contenuto;
   }

class Articolo extends Documento
   (extent articoli)
   {
        attribute string rivista;
        attribute date data_pubbli;
   }
```

ODL: Esempio 2


```
class Progetto
(extent progetti key nome)
 {      attribute string nome;
        attribute Set<Documento> documenti;
        attribute Set<Task> tasks;
        relationship Set<Impiegato> assegnati
               inverse Impiegato::progetti;
        relationship <Impiegato> capo
               inverse Impiegato::dirige;
}
```

ODL: Esempio 2


```
class Task
(extent task)
 {      attribute unsigned short mesi_uomo;
        attribute date data_in;
        attribute date data_fine;
        attribute Impiegato responsabile;
        relationship Set<Impiegato> partecipanti
                inverse Impiegato::tasks;

  }
```

ODL: Esempio 2


```
class Impiegato
(extext impiegati key codicefiscale)
{       attribute string nome;
        attribute string residenza;
        attribute string codicefiscale;
        relationship Set<Progetto> progetti
                inverse Progetto::assegnati;
        relationship <Task> dirige
                inverse Task::capo;
        relationship Set<Task> tasks
                inverse Task::partecipanti;
}
```

ODMG Object Query Language (OQL)

ODMG Object Query Language (OQL)

- Il linguaggio OQL, originariamente sviluppato per il sistema O2, è stato adattato dal comitato ODMG, con varie modifiche, ed è considerato il linguaggio standard di interrogazione degli OODBMS

- OQL si può ritenere SQL-like
  - Usa le stesse parole chiave di SQL
  - Come SQL consente un accesso dichiarativo ai dati
  - Come SQL non è un linguaggio computazionalmente completo (non include, ad esempio strutture di controllo)

ODMG Object Query Language (OQL)

- OQL consente di invocare metodi
  - I metodi possono includere interrogazioni
- OQL permette di accedere agli oggetti attraverso il nome ad essi associato
- Una caratteristica peculiare di OQL rispetto a SQL e necessaria, vista la possibilità di strutturare gli oggetti, è la possibilità di specificare path di accesso ai dati tramite la notazione a “punto” (dot notation)

ODMG Object Query Language (OQL)

- OQL non prevede primitive per modificare gli oggetti nella base di dati
  - coerentemente al modello object-oriented le modifiche devono avvenire solo tramite i metodi
- I comandi OQL possono essere immersi in un linguaggio di programmazione

ODMG Object Query Language (OQL)

- OQL fornisce primitive per considerare insiemi di oggetti
- OQL è un linguaggio funzionale dove gli operatori possono essere liberamente composti
  - il risultato è un oggetto il cui tipo appartiene all'insieme dei tipi di ODMG OM

ODMG Object Query Language (OQL) – Schema di riferimento


```
Interface Iperson {
  exception   NoSuchPerson{};
  void        birth (in string name);
  boolean              marriage(in string person_name)
                      raises(NoSuchPerson);
  unsigned short       ancestors(out set<Person>
                                      all_ancestors);
  void                 move(in string new_address);
};
```

Aggiunge un nuovo figlio ad una persona

ODMG Object Query Language (OQL) – Schema di riferimento

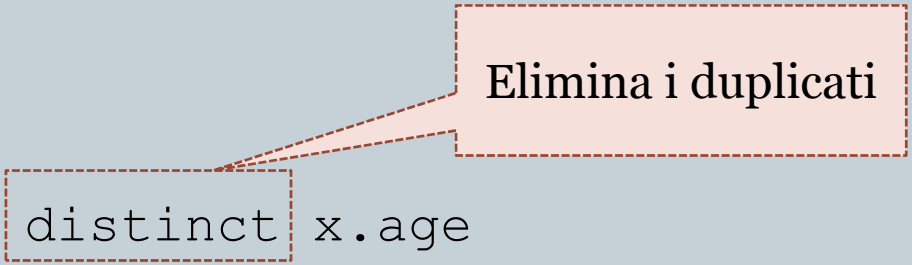

```
class Person: Iperson
(extent people)
{
  attribute string                    name;
  attribute string                    surname;
  attribute unsigned short            age;
  attribute enum
       Gender {male, female}          gender;
  attribute struct Address
   {unsigned short   number,
    string           street,
    City             city}            address;

  relationship Person spouse
       inverse Person::spouse;
  relationship set<Person> children
       inverse Person::parents;
  relationship list<Person> parents
       inverse Person::children;
};
```

ODMG Object Query Language (OQL) – Schema di riferimento


```
class City
( extent cities )
{
  attribute unsigned short     city_code;
  attribute string             name;
  attribute set<Person>        population;
};
```

ODMG Object Query Language (OQL)

- Query OQL che estrae l'età delle persone di nome "Mario"




```
SELECT distinct x.age
FROM people x
WHERE x.name = "Mario"
```

Tipo del risultato:

`set<unsigned short>` (literal)

ODMG Object Query Language (OQL)

- Query OQL che estrae l'età e il sesso delle persone di nome "Pat":

```
SELECT distinct struct(a: x.age, s: x.gender)
FROM people x
WHERE x.name="Pat"
```

Tipo del risultato:

`set<struct(unsigned short, enum)>` (literal)

ODMG Object Query Language (OQL)

Query OQL che, data una persona `p`, estrae il nome della città dove vive il coniuge della persona `p`

```
SELECT p.spouse.address.city.name
FROM people p
```

La query segue la relazione "`spouse`" usando una path expression e raggiunge un oggetto `person` che rappresenta il coniuge di `p`, quindi accede al letterale `address` che ha una struttura nella quale l'attributo `city` è un oggetto di tipo `City` nel quale esiste l'attributo `name`.

ODMG Object Query Language (OQL)

Query OQL che estrae il nome dei figli delle persone presenti nell'estensione `people` (relazione uno a molti):

```
SELECT c.name
FROM people p, p.children c
```

Tipo del risultato:

`bag<string>` (perché non c'è `distinct`)

ODMG Object Query Language (OQL)

Query OQL che estrae il nome dei figli delle persone presenti nell'estensione `people` (relazione uno a molti):

```
SELECT distinct c.name
FROM people p, p.children c
```

Tipo del risultato:

`set<string>` (perché c'è `distinct`)

ODMG Object Query Language (OQL)

Query OQL che estrae l'indirizzo dei figli di ogni persona contenuta nel DB:

```
SELECT c.address
FROM people p, p.children c
```

Tipo del risultato:

```
bag<Address>
```

ODMG Object Query Language (OQL)

Query OQL che estrae l'indirizzo dei figli delle persone che vivono in Main Street ed hanno almeno 2 figli. Si richiedono gli indirizzi dei figli che vivono in una città diversa dai loro genitori:

```
SELECT c.address
FROM people p, p.children c
WHERE p.address.street ="Main Street" AND
        count(p.children) >= 2 AND
        c.address.city != p.address.city
```

Tipo del risultato:

```
bag<Address>
```

ODMG Object Query Language (OQL)

Query OQL che estrae le persone che hanno il nome di un fiore (connessione computata tramite un join perché gli oggetti considerati non sono direttamente connessi):

```
SELECT p
FROM Persons p, Flowers f
WHERE p.name = f.name
```

ODMG Object Query Language (OQL) – Schema di riferimento

Facciamo riferimento ad uno schema con tipi

- **Impiegato** (Nome, Nascita:(Citta, Data), Stipendio, Subordinati, Età(), AssegnaSubordinato(...))

  con estensione **impiegati**
- **Dipartimento** (NomeDip, Sedi:{(NomeSede, CittaSede)}, Direttore)

  con estensione **dipartimenti**

ODMG Object Query Language (OQL) – Schema di riferimento


```
typedef struct Nascita_t {
  attribute string Citta;
  attribute date Data; }

class Impiegato
( extent impiegati key Code)
{ attribute string Code;
  attribute string Nome;
  attribute Nascita_t Nascita;
  attribute unsigned long Stipendio;
  relationship Set<Impiegato> Subordinati
       inverse Impiegato::Capo;
  relationship Impiegato Capo
       inverse Impiegato::Subordinati;
  short Eta ();
  void AssegnaSubordinato (in Impiegato Sub);
}
```

ODMG Object Query Language (OQL) – Schema di riferimento


```
typedef Struct Sede_t {
  attribute string NomeSede;
  attribute string CittaSede;
}

class Dipartimento
( extent dipartimenti )
{ attribute string NomeDip;
  attribute Set<sede_t> Sedi;
  attribute Impiegato Direttore;
  void AssegnaDirettore (in Impiegato Dir);
}
```

ODMG Object Query Language (OQL)

- Seleziona gli stipendi degli impiegati di nome Pat

```
select distinct x.Stipendio
from x in impiegati
where x.Nome = "Pat"
```

- Seleziona nome e età degli impiegati di nome Pat

```
select distinct
        struct(N: x.Nome, E: x.Eta())
from x in impiegati
where x.Nome = "Pat"
```

ODMG Object Query Language (OQL)

- Attraversamento di una relazione

```
select struct (Dip: x.NomeDip,
               Dir: x.Direttore.Nome)
from x in dipartimenti
```

- Equivalente a

```
select struct (Dip: x.NomeDip,
               Dir: y.Nome)
from x in dipartimenti, y in impiegati
where x.Direttore = y
```

Identità

ODMG Object Query Language (OQL)

- Per ciascun impiegato, seleziona il nome e l'insieme dei subordinati con stipendio maggiore di 100000

```
select distinct struct(
            Nome: x.Nome,
            Sub: (select y
                  from y in x.Subordinati
                  where y.Stipendio>100000))
from x in impiegati
```

ODMG Object Query Language (OQL)

La clausola `select` può comparire anche entro la clausola `from`:

```
select struct(N: X.Nome, E: X.Eta())
from x in (select y
           from y in impiegati
           where y.Stipendio > 100000 )
where x.Nome = “Pat”
```

ODMG Object Query Language (OQL)

- Creazione di viste — e accesso a un attributo composto

```
define Romani as
        select x
        from x in impiegati
        where x.Nascita.Citta = "Roma"
```

- Appartenenza

```
select x.Eta()
from x in impiegati
where x in Romani
```

ODMG Object Query Language (OQL)

Quantificatori (queste sono query booleane):

```
for all X in Impiegati:
          X.Stipendio > 200000

exists X in Impiegati:
          X.Eta() < 19
```

si può anche riscrivere così:

```
exists (select x from x in Impiegati
        where x.Eta() < 19)
```

ODMG Object Query Language (OQL)

- Subordinati di subordinati

```
select distinct struct(
            Imp: x, Subsub: z)
from x in impiegati, y in x.Subordinati,
     z in y.Subordinati
```

- Restituisce la *lista* degli impiegati, ordinata per nome

```
SELECT x
FROM x in impiegati
ORDER BY x.Nome
```

ODL: Esempio 1


```
class Automobile
(extent automobili  key Targa )
 {     attribute String Targa;
       attribute String Modello;
       attribute String Colore;
       attribute String Prezzo;
       attribute struct TPartiMeccaniche
             {String Motore,
              String Ammortizzatore} PartiMeccaniche;
       relationship <Costruttore> Costruttore
             inverse Costruttore::Costruisce;
       relationship <Stabilimento> StabilimentoCostr
             inverse Stabilimento::AutoCostruite;
}
```

ODL: Esempio 1


```
class AutoStorica extends Automobile
(extent automobiliStoriche)
{       attribute unsigned short AnnoCostr; }

class AutoSportiva extends Automobile
(extent automobiliSportive)
{       attribute unsigned short MaxVelocità;
        relationship Set<Persona> Pilota
                inverse Persona::Ha_Pilotato;
}
```

ODL: Esempio 1


```
class Costruttore
(extent costruttori key Nome)
{       attribute String Nome;
        relationship <Persona> Presidente
                inverse Persona::Presidenza;
        relationship Set<Automobile> Costruisce
                inverse Automobile::Costruttore;
        relationship Set<Stabilimento> Stabilimenti
                inverse Stabilimento::Proprietario;
}
```

ODL: Esempio 1


```
class Stabilimento
(extent stabilimenti key Nome)
{       attribute String Nome;
        attribute String Città;
        attribute unsigned long Addetti;
        relationship <Costruttore> Proprietario
                inverse Stabilimento::Stabilimenti;
        relationship Set<Automobile> AutoCostruite
                inverse Automobile::StabilimentoProd;

}

class Persona
(extent persone key CodiceFiscale)
{       attribute String Nome;
        attribute String Residenza;
        attribute String CodiceFiscale;
        relationship <Costruttore> Presidenza
                inverse Costruttore::Presidente;
        relationship Set<AutoSportiva> Ha_pilotato
                inverse AutoSportiva::Pilota;
}
```

OQL: Esempio 1

Estrarre le targhe delle automobili rosse

```
SELECT x.Targa
FROM x in Automobile
WHERE x.Colore = "Rosso"
```

Tipo del risultato:

```
set<string>
```

OQL: Esempio 1

Estrarre il nome delle persone che siano contemporaneamente piloti e costruttori delle stesse auto sportive:

```
SELECT y.Nome
FROM x in AutoSportiva,
     y in x.Piloti
WHERE
y = x.Costruttore.Presidente
```

Identità

Tipo del risultato:

`bag<string>` (possono esistere piloti omonimi)

OQL: Esempio 1

Estrarre le auto sportive Ferrari che non sono state costruite a Maranello e che superano i 250 Km/h di velocità massima

```
SELECT a.Targa
FROM a in autoSportive,
     c in costruttori, s in stabilimenti
WHERE a.MaxVelocità > 250 AND
      c = a.Costruttore AND
      c.Nome = “Ferrari” AND
      s = a.StabilimentoCostr AND
      s.Città != ”Maranello”
```

OQL: Esempio 1

Estrarre il nome dei costruttori che vendono auto sportive a un prezzo superiore a 100000 euro; per ciascuno di essi, elencare le città e numero di addetti degli stabilimenti.

```
SELECT distinct struct(
  nome: x.Costruttore.Nome,
  stab: (SELECT struct(Città: y.citta,
                       numAddetti: y.addetti)
         FROM y in Stabilimento
         WHERE y in
                x.Costruttore.Stabilimenti))
FROM x in AutoSportiva
WHERE x.Prezzo > 100000
```

Tipo del risultato:

```
Set(struct(string,bag(struct(string,unsigned long))))
```

OQL: Esempio 1

Estrarre le auto partizionate in base al costruttore, riportando per ogni costruttore il numero di auto contenute nella base di dati.

```
SELECT Costr,
  numAuto: count(select x.a from x in partition)
FROM Automobile a
GROUP BY (Costr: a.Costruttore)
```

Tipo del risultato prima della clausola select:

```
Set<struct(Costr: string,
        partition: bag<struct(a: Automobile)>)>
```

Tipo del risultato finale:

```
Set<struct(Costr: string,numAuto: unsigned long)>
```

OQL: Esempio 1

Estrarre le auto partizionate in base al prezzo.

```
SELECT *
FROM Automobile a
GROUP BY ( PrezzoBasso: a.Prezzo < 10000,
           PrezzoMedio: a.Prezzo >=10000
                        AND a.Prezzo <=25000,
           PrezzoAlto: a.Prezzo > 25000)
```

Tipo del risultato:

```
Set<struct(PrezzoBasso:boolean,
           PrezzoMedio: boolean,
           PrezzoAlto: boolean,
           partition: bag<struct(a: Automobile)>)
```

OQL: semantica delle interrogazioni

Forma base di un'interrogazione OQL

```
SELECT <espressione che produce gli oggetti/letterali
       del risultato - funzione di x1,…,xn >
FROM <collezione1> x1, …, <collezionen> xn
WHERE <espressione booleana - funzione di x1,…,xn >
```

Risultato prima della clausola SELECT:

```
bag<struct(x1: type(<collezione1>),
           …
           xn: type(<collezionen>))>
```

una ennupla $<x_1, \ldots, x_n>$ appartiene al risultato se rende vera l'espressione della clausola WHERE.

Risultato della clausola SELECT: dipende dall'espressione della stessa, ma è funzione delle ennuple prodotte nella fase precedente.

OQL: semantica delle interrogazioni

Forma di un'interrogazione OQL con il GROUP BY

```
SELECT <espressione che produce gli oggetti/letterali
        del risultato - funzione di
        partition e di g1,…,gm >
FROM <collezione1> x1, …, <collezionen> xn
WHERE <espressione booleana - funzione di x1,…,xn >
GROUP BY g1: <espressione1>, …, gm: <espressionem>
```

Risultato prima della clausola SELECT:

```
set<struct(g1: type(<espressione1>),
                …
           gm: type(<espressionem>),
           partition: bag<struct(x1: type(<collezione1>),
                                 …
                                 xn: type(<collezionen>))>
             )>
```

OQL: semantica delle interrogazioni

Forma di un'interrogazione OQL con il GROUP BY

```
SELECT <espressione che produce gli oggetti/letterali
          del risultato - funzione di x1,…,xn >
FROM <collezione1> x1, …, <collezionen> xn
WHERE <espressione booleana - funzione di x1,…,xn >
GROUP BY g1: <espressione1>, …, gm: <espressionem>
HAVING <espressione booleana su partition>
```

Una ennupla $<x_1,\ldots,x_n>$ contribuisce alla generazione dei gruppi se rende vera l'espressione della clausola `WHERE`.

Un gruppo va nel risultato se soddisfa la clausola `HAVING`.

Risultato della clausola `SELECT`: dipende dall'espressione della stessa, ma è funzione dei gruppi prodotte nella fase precedente.

ODL: Esempio 2


```
class Documento
(extent documenti key titolo)
{       attribute string titolo;
        attribute List<Impiegato> autori;
        attribute string stato;
        attribute string contenuto;
}

class Articolo extends Documento
(extent articoli)
{       attribute string rivista;
        attribute date data_pubbli;
}
```

ODL: Esempio 2


```
class Progetto
(extent progetti key nome)
{       attribute string nome;
        attribute Set<Documento> documenti;
        attribute Set<Task> tasks;
        relationship Set<Impiegato> assegnati
                inverse Impiegato::progetti;
        relationship <Impiegato> capo
                inverse Impiegato::dirige;
}
```

ODL: Esempio 2


```
class Task
(extent tasks)
{       attribute unsigned short mesiUomo;
        attribute date dataInizio;
        attribute date dataFine;
        attribute Impiegato responsabile;
        relationship Set<Impiegato> partecipanti
                inverse Impiegato::tasks;
}
```

ODL: Esempio 2


```
class Impiegato
(extext impiegati key codicefiscale)
{       attribute string nome;
        attribute string cognome;
        attribute string residenza;
        attribute string codicefiscale;
        attribute unsigned long stipendio;
        relationship Set<Progetto> progetti
                inverse Progetto::assegnati;
        relationship <Task> dirige
                inverse Task::capo;
        relationship Set<Task> tasks
                inverse Task::partecipanti;
}
```

OQL: Esempio 2

Determinare i task con almeno 20 mesi uomo il cui responsabile guadagna almeno 2000

```
SELECT t
FROM tasks t
WHERE t.mesiUomo > 20 AND
      t.responsabile.stipendio > 2000
```

il risultato è di tipo `bag<Task>`

OQL: Esempio 2

Determinare la data di inizio dei task con almeno 20 mesi uomo:

```
SELECT distinct t.dataInizio
FROM Tasks t
WHERE t.mesiUomo > 20
```

il risultato è un set di letterali: `set<date>`

OQL: Esempio 2

Determinare la data di inizio e la data di fine dei task con almeno 20 mesi uomo

```
SELECT distinct struct(
                    di: t.dataInizio,
                    df: t.dataFine)
FROM tasks t
WHERE t.mesiUomo > 20
```

il risultato è di tipo:

```
set<struct(di:date;df:date)>
```

OQL: Esempio 2

Raggruppare gli impiegati in base ai progetti a cui partecipano, selezionare i progetti con più di 10 impiegati e riportare il nome del progetto, il numero di impiegati e lo stipendio medio degli impiegati (1/2)

```
SELECT struct(progetto: ,
              numImpiegati: count(…),
              stipendioMedio: avg(…) )
FROM impiegati i
GROUP BY (prog: i.progetti.nome)
HAVING count(select x.i from partition x) > 10
```

CONTINUA ->

OQL: Esempio 2

Raggruppare gli impiegati in base ai progetti a cui partecipano, selezionare i progetti con più di 10 impiegati e riportare il nome del progetto, il numero di impiegati e lo stipendio medio degli impiegati (2/2)

```
SELECT struct(progetto: prog,
    numImpiegati: count(select x.i
                        from partition x),
    stipendioMedio: avg(select x.i.stipendio
                        from partition x)
       )
FROM impiegati i …
```

OQL: Esempio 2

Trovare i documenti che non sono stati scritti da impiegati di Verona, riportando il titolo e lo stato del documento

```
SELECT struct(titolo: d.titolo, stato: d.stato)
FROM documenti d
WHERE NOT (exists i in d.autori:
           i.Residenza = 'Verona')
```

oppure

```
SELECT struct(titolo: d.titolo, stato: d.stato)
FROM documenti d
WHERE for all i in d.autori:
           i.Residenza != 'Verona')
```

OQL: Esempio 2

Trovare i progetti che non hanno task il cui responsabile abbia uno stipendio maggiore di 10000 euro, riportando il nome del progetto e il cognome del capoprogetto:

```
SELECT struct(progetto: p.nome,
              capo: p.capo.cognome)
FROM progetti p
WHERE NOT (exists t in p.tasks:
           t.responsabile.stipendio > 10000)
```

OQL: Esempio 2

Trovare gli impiegati che partecipano ad almeno tre task, non dirigono nessun task e partecipano a meno di tre progetti con un budget maggiore di 100000 euro, riportando il nome e il cognome dell'impiegato:

```
SELECT struct(nome: i.nome,
              cognome: i.cognome)
FROM impiegati i
WHERE count(i.tasks) > 3 AND i.dirige.is_undefined()
  AND count(SELECT x FROM i.progetti x
            WHERE x.budget > 100000) < 3
```

Riferimenti

- **The Object Data Standard: ODMG 3.0.**

  Roderic Geoffrey Galton Cattell, Douglas K. Barry, Mark Berler

  Morgan Kaufmann, 2000

UML per la rappresentazione grafica di uno schema ODL

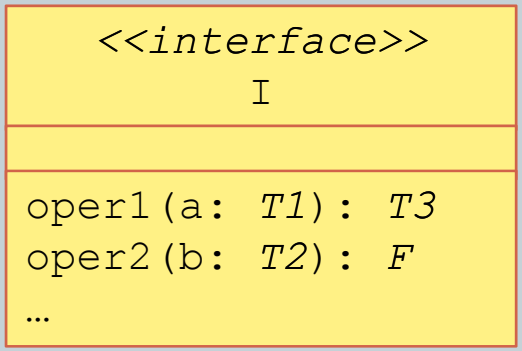
E' possibile utilizzare il modello UML e la sua sintassi grafica per specificare graficamente lo schema di una base di dati a oggetti in modo che possa essere facilmente tradotto in uno schema ODL.

Supponiamo di voler rappresentare graficamente in UML:

- Le dichiarazioni di *interface*, contenenti operazioni
- Le dichiarazioni di *typedef*, contenenti attributi
- Le dichiarazioni di *classi*, contenenti attributi e relazioni tra classi.

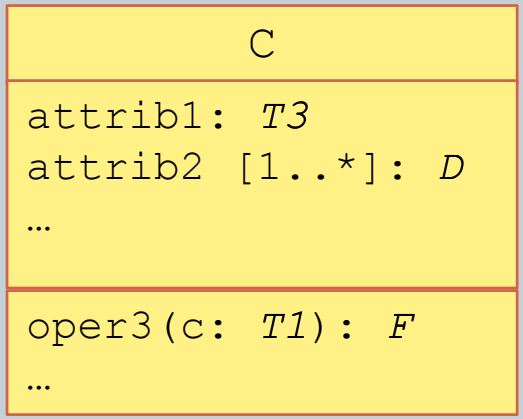
UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *Interface I* contenente due operazioni *oper1* e *oper2* con due parametri di tipo base *T1* e *T2* dove la prima operazione restituisce un tipo base *T3* mentre la seconda restituisce un oggetto della classe *F*.



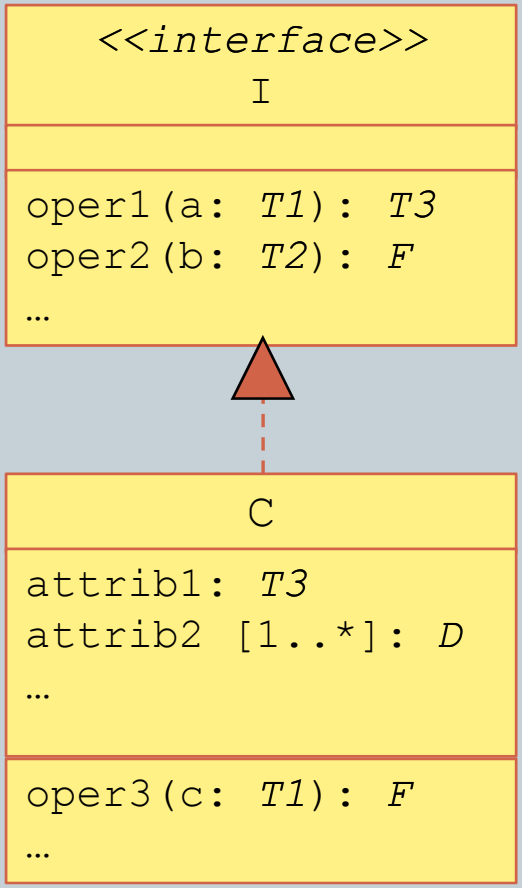
UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *Classe C* con due attributi attrib1 e attrib2, il primo di tipo base T3 e il secondo che punta ad un set di oggetti di tipo D. E' inclusa anche l'operazione oper3 che ha un parametro di tipo base T1 e restituisce un oggetto della classe F.



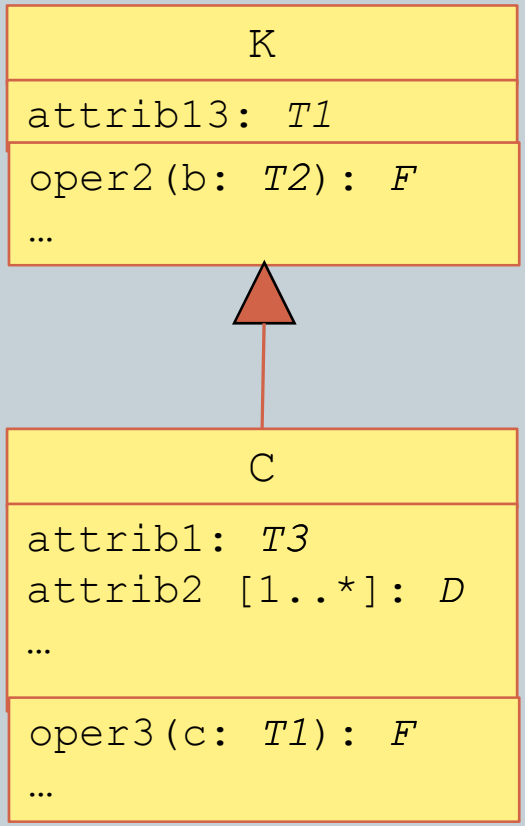
UML per la rappresentazione grafica di uno schema ODL

Una *Classe C* può anche ereditare operazioni da una *interface I*.



UML per la rappresentazione grafica di uno schema ODL

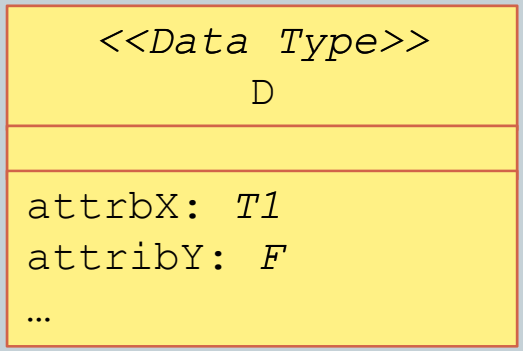
Una *Classe C* può anche ereditare un'altra *classe K*.



UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *tipo strutturato* per letterali.

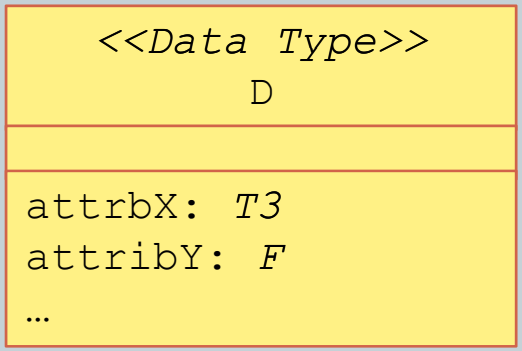
Le definizioni esterne di tipi strutturati, vale a dire quelle realizzate con il costrutto *Typedef struct*, possono essere rappresentate in UML come *DataType*.



UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *tipo strutturato* per letterali.

Le definizioni interne di tipi strutturati, vale a dire quelle realizzate con il costrutto *struct* direttamente nella specifica di classe o interfaccia, possono essere rappresentate comunque come *DataType* esterni alla classe.



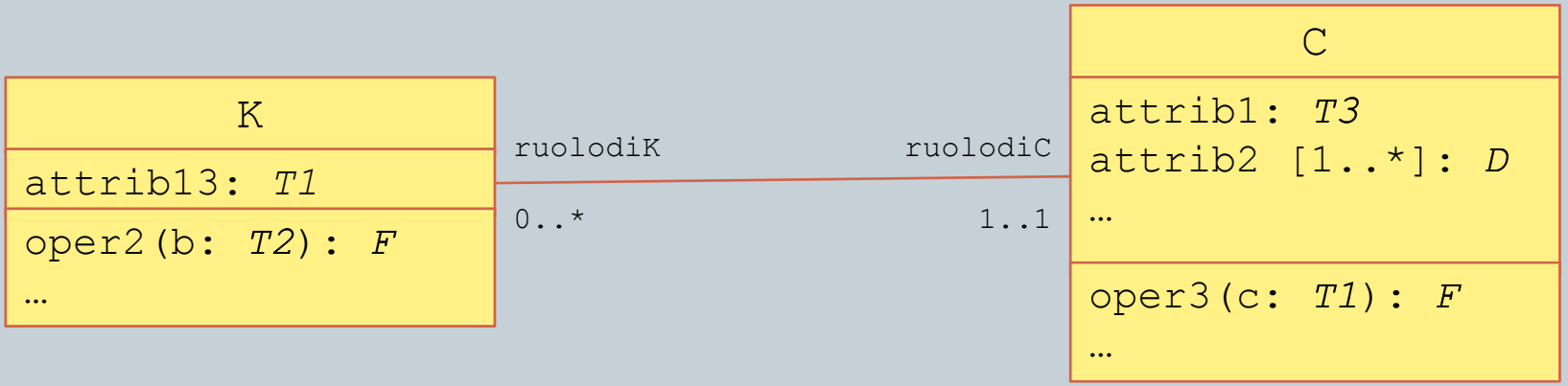
UML per la rappresentazione grafica di uno schema ODL

Rappresentazione di una *relationship* tra classi.

Per rappresentare le relationship binarie tra classi è possibile utilizzare il costrutto associazione di UML. Tale costrutto consente di precisare i ruoli e anche le cardinalità delle associazioni. Tuttavia, le cardinalità non consentono di precisare la tipologia di collezione che implementa l'associazione, tale precisazione diventa un dettaglio dello schema ODL.

UML per la rappresentazione grafica di uno schema ODL

Relationships



Class K
(extent Ks)
{ attribute T1 attrib13;
  F oper2(in b: T2);
  relationship C ruolodiC
        inverse C::ruolodiK
}

Class C
(extent Cs)
{ attribute T3 attrib1;
  attribute set<D> attrib2;
  F oper3(in c: T1);
  relationship set<K> ruolodiK
        inverse K::ruolodiC
}

UML per la rappresentazione grafica di uno schema ODL

Ulteriori dettagli della specifica ODL:

- *extent*: potrebbe diventare un tag value sulla classe
- *key*: potrebbe essere usato uno stereotipo `<<key>>` per etichettare gli attributi/ruoli che fanno parte della chiave della classe.