

# Standard ODMG

1

ALBERTO BELUSSI  
PRIMA PARTE  
ANNO ACCADEMICO 2011/'12

# Sommario - ODMG

2

- **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

3

- **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

4

- 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

5

- 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

6

- **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

7

- 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)

9

- 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)

10

- 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

11

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

12

## Oggetti

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

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

13

## 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 structures*)

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

14

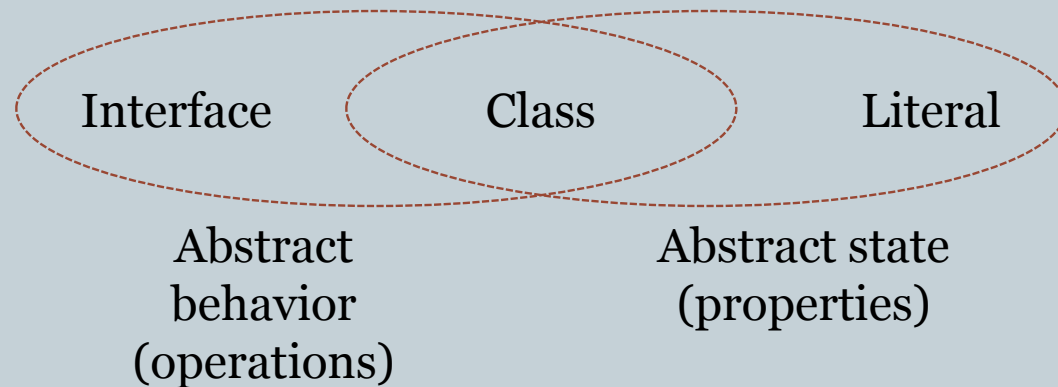
- Tutti gli elementi di un dato tipo hanno
  - La stessa gamma di **stati** (stesse proprietà)
  - Un **comportamento** (behavior) comune (lo stesso insieme di operazioni)
- Ogni tipo è composto da
  - Una **specificazione esterna** (“external specification”)
  - Una o più **implementazioni**

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

15

**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

16

**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

17

**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

18

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

19

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

20

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

21

Interface

Class

Literal

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

22

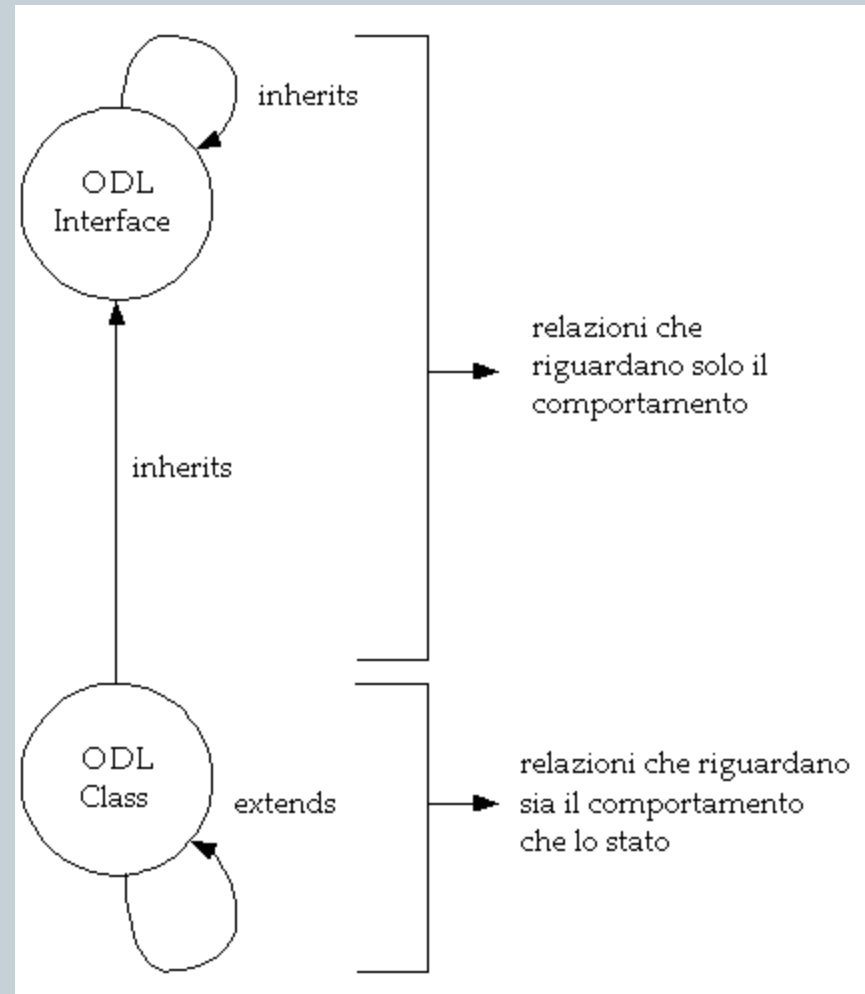
**Implementazione** di un tipo (“language binding”) consiste di una **rappresentazione** e di un insieme di **metodi**.

- 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

23

- 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

24

- 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

25

## 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, compresi 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

26

## 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

27

## 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{...};
```

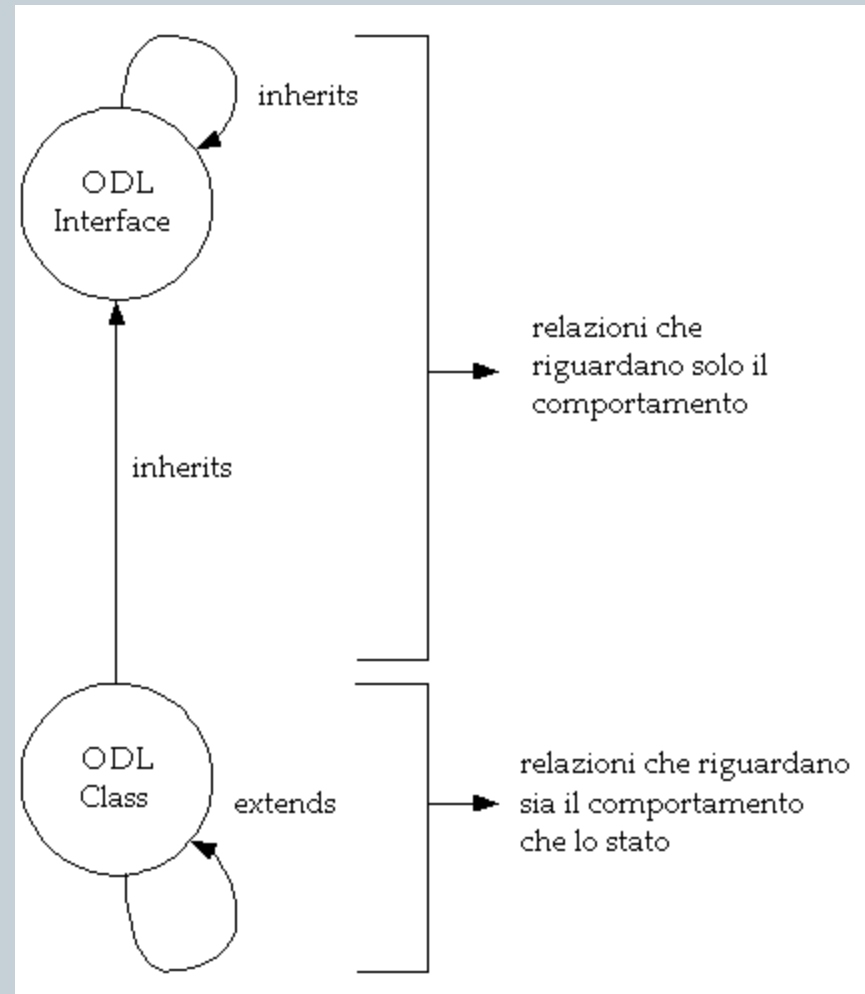
## Non sono permessi conflitti di nomi di operazioni

- Name overloading non permesso

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

28

- 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

29

- 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

30

```
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

31

- 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

32

- 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

33

```
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

34



- **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

35

- **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

36

- 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

37

- Creazione degli oggetti

- avviene mediante l'invocazione di un metodo definito in un'interfaccia, detta **factory interface**, fornita dalla specifica implementazione ("language binding") considerata

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

il cui metodo `new()` crea una 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

38

```
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

Crea un nuovo oggetto  
equivalente all'oggetto  
ricevente.  
L'oggetto creato non è  
uguale all'originale  
rispetto a "same\_as"

Confronto  
uguaglianza  
(identità) fra  
oggetti

Cancellazione dalla base di  
dati e dalla memoria

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

39

- 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

40

- 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

41

- 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

42

- 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

43

- **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

44

```
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

45

```
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

46

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

```
unsigned long  cardinality();
```

**Ritorna il numero di  
elementi contenuti nella  
collezione**

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

**Metodi che  
verificano le  
caratteristiche della  
collezione**

...

...

...

```
};
```

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

47

```
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

48

```
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

49

```
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

50

```
interface Collection: Object {
```

```
...
```

Restituisce l'oggetto che  
soddisfa il predicato

```
...
```

```
Object      select_element(in string OQL_predicate);
```

```
Iterator    select(in string OQL_predicate);
```

```
boolean     query(in string OQL_predicate,  
                  inout Collection result );
```

```
};
```

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

51

- 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

53

```
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

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

Recupera l'elemento puntato

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

## Set

54

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

**Collezione NON  
ordinata di elementi che  
NON ammette duplicati**

```
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);
};
```

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

## Set

55

```
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

56

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

```
Class Set: Collection {  
...  
...
```

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

**Confronti convenzionali tra  
insiemi**



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

## Set

57

```
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

58

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

**Collezione NON  
ordinata di elementi che  
ammette duplicati**

```
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);  
};
```

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

## Bag

59

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

60

```
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

61

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

Collezione ordinata di  
elementi che ammette  
duplicati

```
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);  
...  
};
```

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

## List

62

```
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

63

```
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

64

```
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

65

```
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

66

```
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

67

```
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

68

```
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

69

```
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

70

- **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

71

- **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

72

```
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

73

- 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

74

- **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

75

- 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

76

- **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

77

- **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

78

- **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

79

## Structured literal (user-defined structure)

- Esempio:

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

```
Address.city = "Verona"
```

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

80

- **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

81

- 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

82

- **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 letterale si usa `equals` e non `same_as`

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

83

- 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

84

## 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 appartenenza

- 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

85

- 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

86

## 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

87

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

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

Dichiarazione di traversal path

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

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

88

## 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

89

- 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

90

- **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

91

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

92

- 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

93

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

Per creare  
transazioni

Per manipolare  
transazioni

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

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

94

```
interface Transaction{  
    ...
```

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

```
void commit() raises(TransactionNotInProgress);
```

```
void abort() raises(TransactionNotInProgress);
```

```
void checkpoint() raises(TransactionNotInProgress);
```

```
boolean isOpen();
```

```
}
```

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

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

95

- 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

96

- 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

97

- 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();
};
```