OBJECTSTORE

Operazioni necessarie per creare e manipolare oggetti persistenti

- creare una sessione
- creare o aprire un database
- iniziare una transazione

OBJECTSTORE - Sessioni

- creare una sessione:

  session = Session.create(null,null)

- associare il thread corrente a una sessione

  session.join()

OBJECTSTORE - Database

- creare un database

  db = Database.create(name,file_mode)

  es. di **file_mode**:
  ObjectStore.ALL_READ|ObjectStore.ALL_WRITE

- aprire un database esistente

  db = Database.open(name,access_mode)

  access_mode può essere **ObjectStore.UPDATE** o **ObjectStore.READONLY**

OBJECTSTORE - Database

esempio:

```
try {
      db = Database.open(dbName, ObjectStore.OPEN_UPDATE);
   } catch (DatabaseNotFoundException e) {
      db = Database.create(dbName,
  ObjectStore.ALL_READ | ObjectStore.ALL_WRITE);
   }
```

- chiudere un database

  db.close()

OBJECTSTORE - Transazioni

- creare (iniziare) una transazione

  tr = Transaction.begin(type)

  type può essere **ObjectStore.UPDATE** o **ObjectStore.READONLY**

- terminare una transazione

  tr.commit()

  tr.abort()

OBJECTSTORE - Roots

- creare una radice di persistenza (entry point nel database)

  db.createRoot(name,object)

  name è una stringa, **object** un oggetto che viene reso persistente

  tutti gli oggetti riferiti (raggiungibili) da **object** diventano anch'essi persistenti

  persistence by reachability

- **object** deve essere istanza di una classe *persistence capable*

OBJECTSTORE - Roots

- accedere ad una radice di persistenza

  obj = db.getRoot(name)

  l'operazione **getRoot** ritorna un oggetto di tipo **Object** ⇒ necessità di cast per manipolarlo come oggetto del tipo opportuno

  es: **tim = (Person)db.getRoot("Tim")**

- possibilità di *retainment* dei riferimenti al momento del commit di una transazione

OBJECTSTORE - Roots

- persistent garbage collector che gestisce la rimozione automatica di oggetti dal database

- possibilità di cancellazione esplicita

  ObjectStore.destroy(object)

  metodo **preDestroyPersistent()** che può essere ridefinito nelle varie classi e viene eseguito prima di **destroy** - es. per cancellare oggetti persistenti riferiti

  attenzione nel cancellare esplicitamente **String** objects (possono essere condivisi)

$\boxed{\text{OBJECTSTORE}}$

passi necessari per utilizzare ObjectStore

- la variabile d'ambiente **CLASSPATH** deve contenere
  - i file **pse.zip** o **pro.zip** e **tools.zip**
  - la source directory (Java source files e class files) e la directory contenente i file class annotati (coincidono se si annota *in place*)

OBJECTSTORE

- compilare

  javac *.java

- postprocessare

  osjcfp -inplace -dest . lista_file.class

  la lista può distinguere le classi da rendere *persistent capable* da quelle da rendere *persistent aware*

  -persistentcapable lista_file.class
  -persistentaware lista_file.class

  possibilità di usare variabile - es. **@ cfpargs**

OBJECTSTORE

- eseguire

  java nome_package.nome_file_main [params]

- nota: i file contenenti database devono avere estensione **.odb**

  per ogni database vengono creati altri due file con lo stesso nome ed estensioni **.odt** e **.odf**

OBJECTSTORE - Collezioni

- una collezione è un singolo oggetto che rappresenta un gruppo di oggetti

- ObjectStore fornisce diverse implementazioni delle interfacce **java.util.Collection** e **java.util.Map** della JDK 1.2

- **Collection** fornisce i metodi per operare su gruppi di oggetti in cui gli oggetti posso essere ordinati, possono essere duplicati, e possono essere interrogati

OBJECTSTORE - Collezioni

- la rappresentazione interna di una classe che implementa **Collection** può essere una tabella hash, un albero binario, o un'altra struttura dati
  - **java.util.List** estende **Collection** – gli elementi sono ordinati e sono permessi duplicati
  - **java.util.Set** estende **Collection** – gli elementi non sono ordinati e non sono permessi duplicati

- **Map** fornisce i metodi per operare su gruppi di coppie chiave/valore

  ad ogni chiave corrisponde al massimo un valore

  non si possono interrogare collezioni che implementano **Map**

OBJECTSTORE - Collezioni

Classe	*Implementa*
OSHashBag	**Collection**
OSHashMap	**Map**
OSHashSet	**Set**
OSHashtable	Nessuna
OSTreeMapxxx	**Map**
OSTreeSet	**Set**
OSVector	**Collection**
OSVectorList	**List**

xxx = **ByteArray**, **Double**, **Float**, **Integer**, **Long**, **String**

OBJECTSTORE - Collezioni

- **Hash**: rappresentazione a hash table
- **Vector**: rappresentazione a vettore
- **Tree**: rappresentazione a B-tree (adatta per collezioni di grandi dimensioni)

metodi per ottenere visione a **Collection** (interrogabile) di una **Map** (**keySet()**, **values()**, **entries()**)

OBJECTSTORE - Iteratori

- le interfacce **Iterator** e **ListIterator** possono essere utilizzate per navigare nelle collezioni

- un *iterator* denota una posizione in una collezione

- l'interfaccia **ListIterator** estende **Iterator** (es. restituire l'indice di un elemento)

- interfaccia **IndexIterator** per scandire un indice o una **Map**

OBJECTSTORE - Interrogazioni

due passi:

1. creare l'interrogazione

2. eseguire l'interrogazione su una collezione

l'interrogazione è un predicato (espressione a valori booleani) che può includere gli operatori Java standard aritmetici, condizionali, e relazionali, oltre a metodi
+ funzionalità per pattern-matching tra stringhe

OBJECTSTORE - Interrogazioni: creazione

- si utilizza il costruttore della classe **Query** che ha come parametri un **Class** object e una stringa (l'interrogazione)
  - il **Class** object specifica il tipo degli elementi contenuti nella collezione che si vuole interrogare
  - deve essere una classe pubblica, e ogni caratteristica (attributo o metodo) specificata nell'interrogazione deve essere anch'essa pubblica

OBJECTSTORE - Interrogazioni: creazione

esempi

Query q1 =
new Query(Person.class, "getName() == \"Tim\"");

Query q2 = new Query(Person.class, "age < 40");

OBJECTSTORE - Interrogazioni: esecuzione

- per eseguire un'interrogazione su una collezione si utilizza il metodo **Query.select** con la collezione da interrogare come parametro

- esempio

  Collection youngP = q2.select(allPersons);

  (**allPersons** collezione di oggetti istanza della classe **Person**)

- uso di **pick** invece che di **select** per ottenere un solo oggetto che verifica l'interrogazione

OBJECTSTORE - Interrogazioni: uso di variabili

- nelle interrogazioni è possibile utilizzare variabili invece di costanti
  - si crea una **FreeVariables** list
  - vi si inseriscono le variabili che si intendono utilizzare, specificandone il tipo
  - nella creazione dell'interrogazione, si utilizza un costruttore con tre parametri
  - nella stringa che rappresenta l'interrogazione si utilizzano le variabili nella lista
  - il terzo parametro del costruttore è la lista di variabili
  - prima di eseguire l'interrogazione si crea una **freeVariableBindings** list che viene passata come secondo argomento al metodo **select**

OBJECTSTORE - Interrogazioni: uso di variabili

esempio

```
FreeVariables vars = new FreeVariables();
vars.put("x",String.class);
Query q = new Query(Person.class, "getName() == x", vars);
...
String name = {input dell´utente o qualche altro calcolo};
FreeVariableBindings binds = new FreeVariableBindings();
binds.put("x",name);
Collection result = q.select(allPersons,binds);
```

OBJECTSTORE - **Indici**

- è possibile definire degli indici per velocizzare l'esecuzione delle interrogazioni
- si possono definire indici su ogni collezione che implementi l'interfaccia **com.odi.util.IndexedCollection** – al momento, solo **OSTreeSet**
- un indice può essere visto come una funzione inversa dal valore di un attributo o dal valore di ritorno di un metodo a tutti gli elementi che hanno tale valore
- permette di velocizzare l'esecuzione delle interrogazioni

OBJECTSTORE - Indici: creazione

- due metodi

  addIndex(Class, String)

  addIndex(Class, String, boolean, boolean)

  - il primo argomento indica il tipo su cui si definisce l'indice
  - il secondo indica la caratteristica (pubblica) su cui si indicizza
  - i due argomenti opzionali indicano se l'indice è ordinato e se ammette duplicati (default: non ordinato con duplicati)

OBJECTSTORE - Indici: creazione

esempio

```
db.createRoot("allPersons",
  allPersons= new OSTreeSet(db));

allPersons.addIndex(Person.class,"getName()");
```

OBJECTSTORE - Indici: mantenimento

- una volta definito un indice, ObjectStore mantiene automaticamente l'indice se si aggiungono o rimuovono elementi dalla collezione

- l'utente deve aggiornare manualmente l'indice in caso di modifiche al valore della caratteristica indicizzata

  metodi **removeFromIndex**, **addToIndex** e **updateIndex**

OBJECTSTORE - Indici: cancellazione

- un indice può essere rimosso tramite il metodo

  dropIndex(Class, String)

- <u>esempio</u>

  allPersons.dropIndex(Person.class,"getName()");

OBJECTSTORE - Extents

- le applicazioni ObjectStore generate con il Component Wizard contengono già il codice per la gestione dell'estensione delle classi

- ogni classe per cui si richiede il mantenimento dell'extent ha un attributo **public static Ext** e due metodi
  - **preFlushContents()** utilizzato per assicurare che quando un'istanza della classe diventa persistente venga inserita nell'extent
  - **updateExtents(Database db, boolean add)** utilizzato per inserire un oggetto nell'extent

- i tipi possibili per gli extent sono **OSTreeSet** e **OSVector**