Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Chapter 6 – Object Persistence, Relationships and Queries

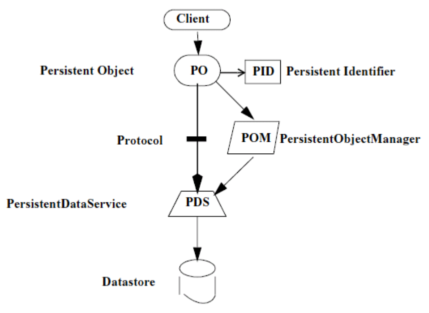
Middleware for Heterogenous and Distributed Information Systems - WS05/06

Object Persistence

- Persistent objects
  - Goals
    - simplification of programming model for data access
      - no explicit interaction with data source using SQL, JDBC, …
      - eliminate "impedance mismatch"
    - hide heterogeneity of existing data stores
      - data model, query language, API
  - Basic approach
    - application (component) interacts with objects
      - create, delete
      - access object state variables
      - method invocation
    - persistence infrastructure maps interactions with objects to operations on data sources
      - e.g., INSERT, UPDATE, SELECT, DELETE
- Variations
  - explicit vs. implicit (transparent) persistence
  - type-specific vs. orthogonal persistence

CORBA – Persistent Object Service

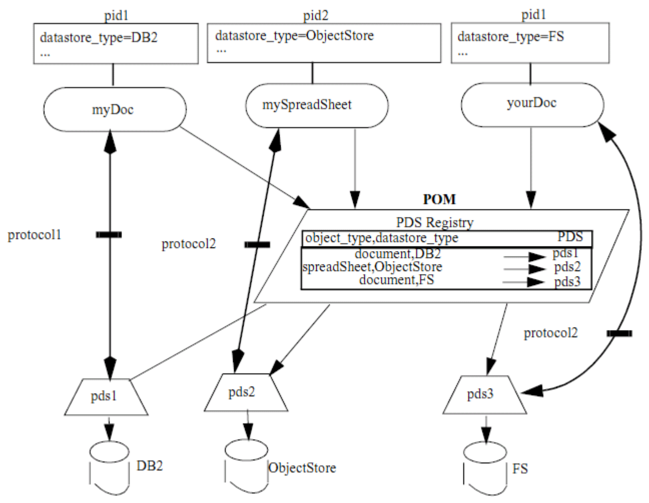
- Goal: uniform interfaces for realizing object persistence
- POS (Persistent Object Service) components
  - PO: Persistent Object
    - are associated with persistent state through a PID (persistent object identifier)
      - PID describes data location
  - POM: Persistent Object Manager
    - mediator between POs and PDS
    - realizes interface for persistence operations
    - interprets PIDs
    - implementation-independent
  - PDS: Persistent Data Service
    - mediator between POM/PO and persistent data store
    - data exchange between object and data store as defined by protocols
  - Datastore
    - stores persistent object data
    - may implement *Datastore_CLI* (encapsulates ODBC/CLI)



CORBA Persistence Model

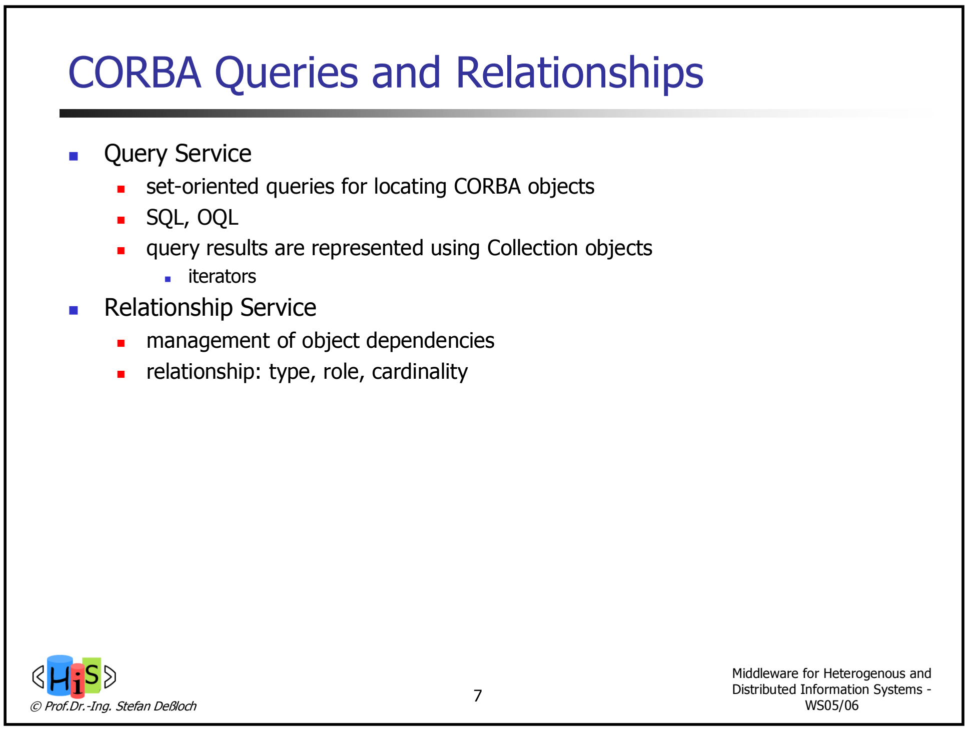
- CORBA object is responsible for realizing its own persistence
  - can use PDS services and functions
  - implicit persistence control
    - client is potentially unaware of object persistence aspects
  - explicit persistence control
    - persistent object implements PO interface, which can then be used by the client
- Explicit persistence control by CORBA client:
  - client creates PID, PO using factory objects
  - PO Interface
    - connect/disconnect – automatic persistence for the duration of a "connection"
    - store/restore/delete – explicit transfer of data
    - delegated to POM, PDS
  - caution!: CORBA object reference and PID are different concepts
    - client can "load" the same CORBA object with data from different persistent object states

Persistent Object Manager



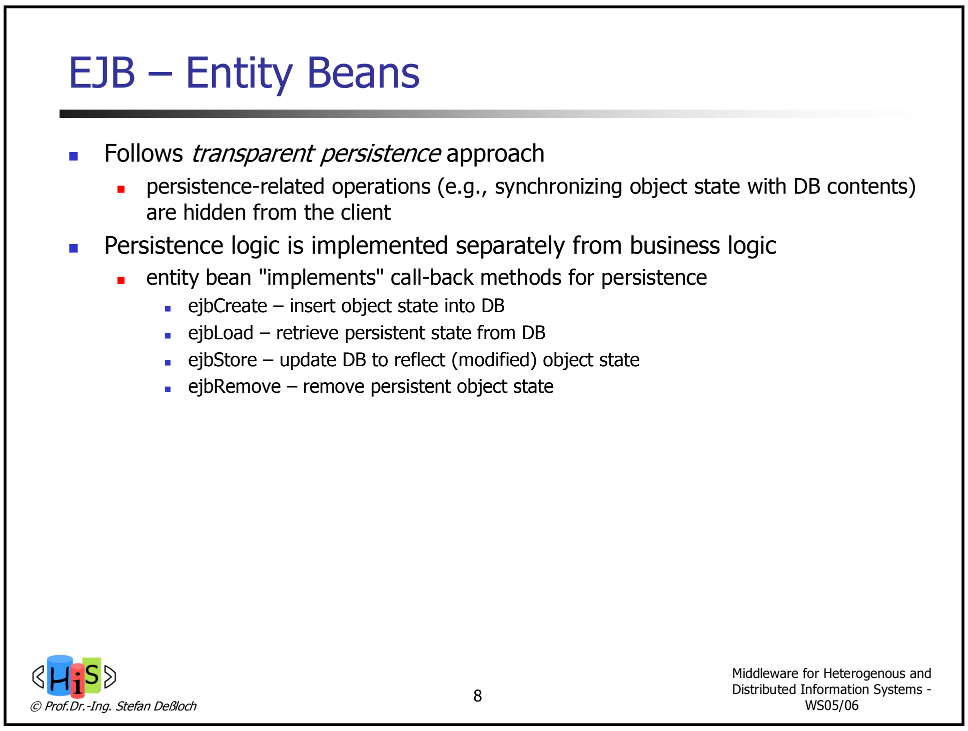
Persistence Protocols

- CORBA Persistence Service defines three protocols
  - Direct Access (DA) protocols
    - PO stores persistent state using so-called *direct access data objects* (DADOs)
      - CORBA objects whose interfaces only have attributes
      - defined using Data Definition Language (IDL subset)
    - DADOs may persistently reference other DADOs, CORBA objects
  - ODMG'93 protocols
    - similar to DA protocol (is a superset)
      - own DDL (ODL) for defining POs
    - ideal for OODBMS-based persistence
  - Dynamic Data Object (DDO) protocols
    - "generic", self-describing DO
      - methods for read/update/add of attributes and values
      - manipulation of meta data
    - used for accessing record-based data sources (e.g. RDBMS) using DataStore CLI interface
      - SQL CLI for CORBA
- Protocols are employed in the implementation of DOs



CORBA Queries and Relationships

- Query Service
  - set-oriented queries for locating CORBA objects
  - SQL, OQL
  - query results are represented using Collection objects
    - iterators
- Relationship Service
  - management of object dependencies
  - relationship: type, role, cardinality

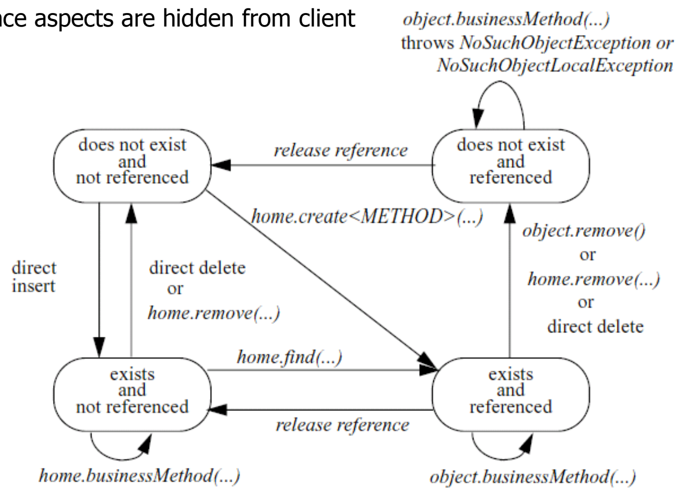


EJB – Entity Beans

- Follows *transparent persistence* approach
  - persistence-related operations (e.g., synchronizing object state with DB contents) are hidden from the client
- Persistence logic is implemented separately from business logic
  - entity bean "implements" call-back methods for persistence
    - ejbCreate – insert object state into DB
    - ejbLoad – retrieve persistent state from DB
    - ejbStore – update DB to reflect (modified) object state
    - ejbRemove – remove persistent object state

Entity Beans - Client-Perspective

- Persistence aspects are hidden from client



object.businessMethod(...) throws NoSuchObjectException or NoSuchObjectLocalException
does not exist and not referenced
release reference
does not exist and referenced
home.create<METHOD>(...)
object.remove() or home.remove(...) or direct delete
direct insert
direct delete or home.remove(...)
home.find(...)
exists and not referenced
release reference
exists and referenced
home.businessMethod(...)
object.businessMethod(...)

Container-Managed Persistence (CMP)

- Bean developer defines an *abstract persistence schema* in the deployment descriptor
  - persistent attributes (*CMP fields*)
  - relationships
- Mapping of CMP fields to DB-structures (e.g., columns) in deployment phase
  - depends on DB, data model
  - tool support
    - *top-down, bottom-up, meet-in-the-middle*
- Container saves object state, maintains relationships
  - bean does not worry about persistence mechanism
    - call-back methods don't contain DB access operations
- Manipulation of CMP fields through access methods (get*field*(), set*field*(...))
  - access within methods of the same EB
  - client access can be supported by including access methods in the remote interface
  - provides additional flexibility for container implementation
    - lazy loading of individual attributes
    - individual updates for modified attributes

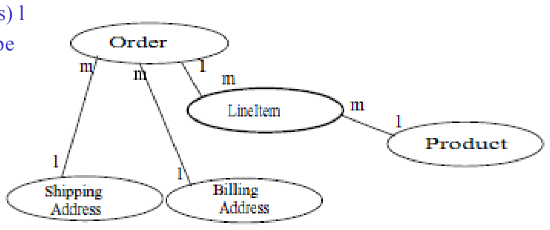
Container-managed Relationships

- Relationships can be defined in deployment descriptor
  - part of abstract persistence schema
- Relationships may be
  - uni-directional ("reference")
  - bi-directional
- Relationship types
  - 1:1, 1:n, n:m
- Access methods for accessing objects participating in a relationship
  - like CMP field methods
  - Java Collection interface for set-valued reference attributes
- Container generates code for
  - relationship maintenance
  - persistent storage
  - cascading delete (optional)

EJB Query Language

- Query language for CMP EntityBeans
  - used in the definition of user-defined Finder methods of an EJB Home interface
    - no arbitrary (embedded or dynamic) object query capabilities!
  - uses abstract persistence schema as its schema basis
  - SQL-like
- Example:

```
SELECT DISTINCT OBJECT(o)
FROM Order o, IN(o.lineItems) l
WHERE l.product.product_type
    = 'office_supplies'
```



Bean-Managed Persistence (BMP)

- Callback-methods contain explicit DB access operations
  - useful for interfacing with legacy systems or for realizing complex DB-mappings (not supported directly by container or CMP tooling)
- No support for container-managed relationships
- Finder-methods
  - have to be implemented in Java
  - no support for EJB-QL

Entity Beans

- Problems
  - Distributed component vs. persistent object
    - granularity
    - potential overhead
      - solution in EJB 2.0: local interfaces
      - but: semantic differences (*call-by-value* vs. *call-by-reference*)
    - complexity of development process
  - Missing support for class hierarchies with inheritance
  - possible performance problems
- Alternatives?
  - use JDBC, stored procedures
    - complex development
  - use O/R Mapping product
    - proprietary
  - implement own persistence framework
    - complex
  - JDO

JDO – Java Data Objects

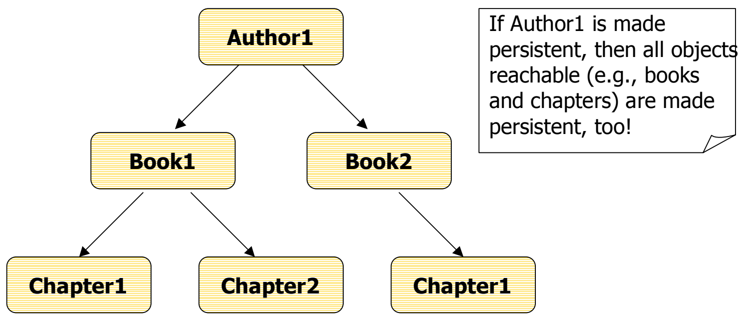
- JDO developed as new standard for persistence in Java-based applications
  - first JDO specification 1.0 released in March 2002 (after ~ 3 years) through Suns JCP (Java Community Process)
  - > 10 vendor implementations plus open-source projects
  - *mandatory features* and *optional features* in the specification (i.e., some optional features are „standardised" → promises better portability).
- Features, elements
  - orthogonal persistence
  - native Java objects (inheritance)
  - byte code enhancement
  - mapping to persistence layer using XML-metadata
  - transaction support
  - JDO Query Language
  - JDO API
  - JDO identity
  - JDO life cycle
  - integration in application server standard (J2EE)

Orthogonal Persistence in JDO

- Object-based persistence, independent of type/class
  - Java class supports (optional) persistence (implements PersistenceCapable)
  - not all instances of the class need to be persistent
    - application can explicitly turn a transient object into a persistent object (and vice versa)
- Persistence logic is transparent for application
  - interacting with transient and persistent objects is the same
- "persistence by reachability"

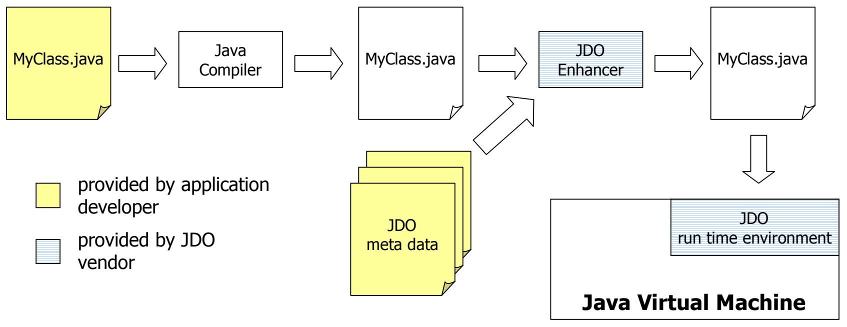
Persistence by Reachability

- all **`PersistenceCapable`** objects reachable from persistent object within an object graph are made persistent, too
- cascading delete? optional in JDO

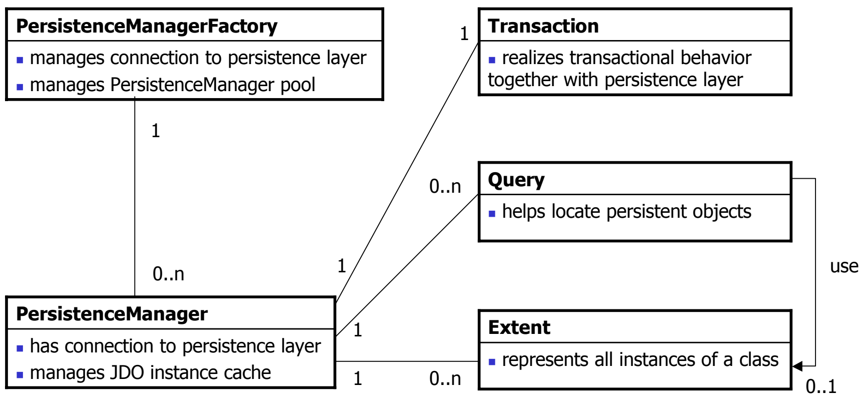

Author1
Book1
Book2
Chapter1
Chapter2
Chapter1
If Author1 is made persistent, then all objects reachable (e.g., books and chapters) are made persistent, too!

Byte-Code-Enhancement

- Java bytecode (*.class) and metadata (*.jdo)
- Same object class (now implements PersistenceCapable)
- O/R-mapping specification is vendor-specific



JDO API


PersistenceManagerFactory
manages connection to persistence layer
manages PersistenceManager pool
Transaction
realizes transactional behavior together with persistence layer
Query
helps locate persistent objects
Extent
represents all instances of a class
PersistenceManager
has connection to persistence layer
manages JDO instance cache
1
0..n
1
1
0..n
1
1
0..n
use
0..1

PersistenceManager API - Example

```
Author author1 = new Author("John", "Doe");
PersistenceManager pm1 = pmf.getPersistenceManager();
pm1.currentTransaction.begin();
pm1.makePersistent(author1);
Object jdoID = pm1.getObjectId(author1);
pm1.currentTransaction.commit();
pm1.close();

// Application decides that author1
// must be deleted
PersistenceManager pm2 = pmf.getPersistenceManager();
pm2.currentTransaction.begin();
Author author2 = (Author)pm2.getObjectById(jdoID);
pm2.deletePersistent(author2);
pm2.currentTransaction.commit();
pm2.close();
```

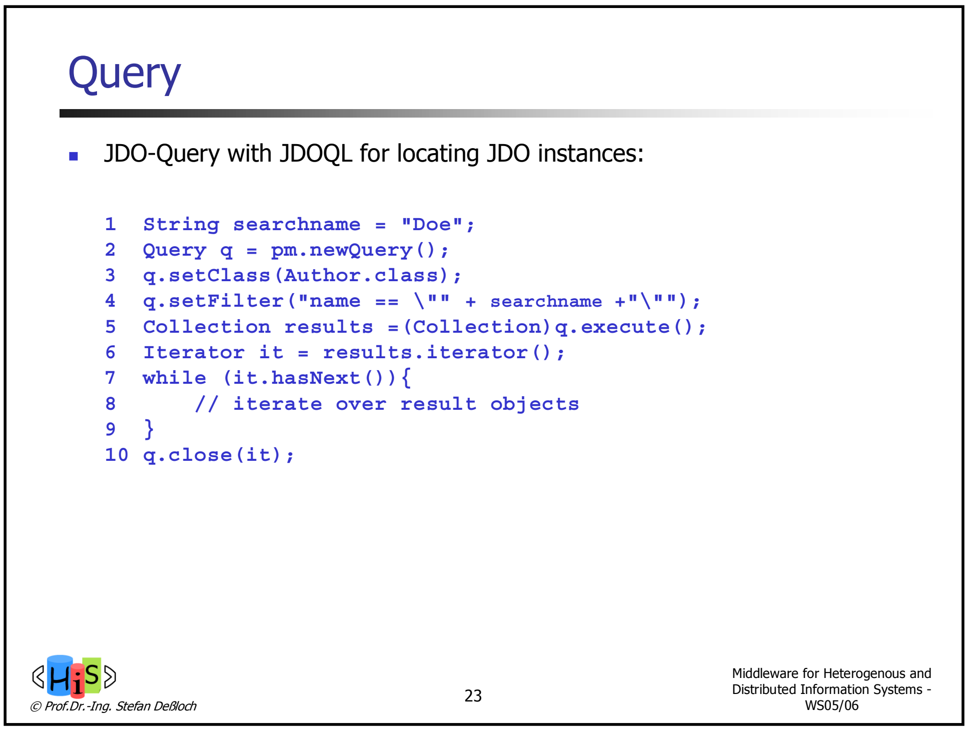
Transactions

- JDO transactions supported at the object level
- Datastore Transaction Management (standard):
  - JDO synchronises transaction with the persistence layer
  - transaction strategy of persistence layer is used
- Optimistic Transaction Management (optional):
  - JDO progresses object transaction at object level
  - at commit time, transaction is synchronized with persistence layer
- Transactions and object persistence are orthogonal

object characteristics	transactional	non-transactional
persistent	standard	optional
transient	optional	standard (JVM)

JDO Query Language

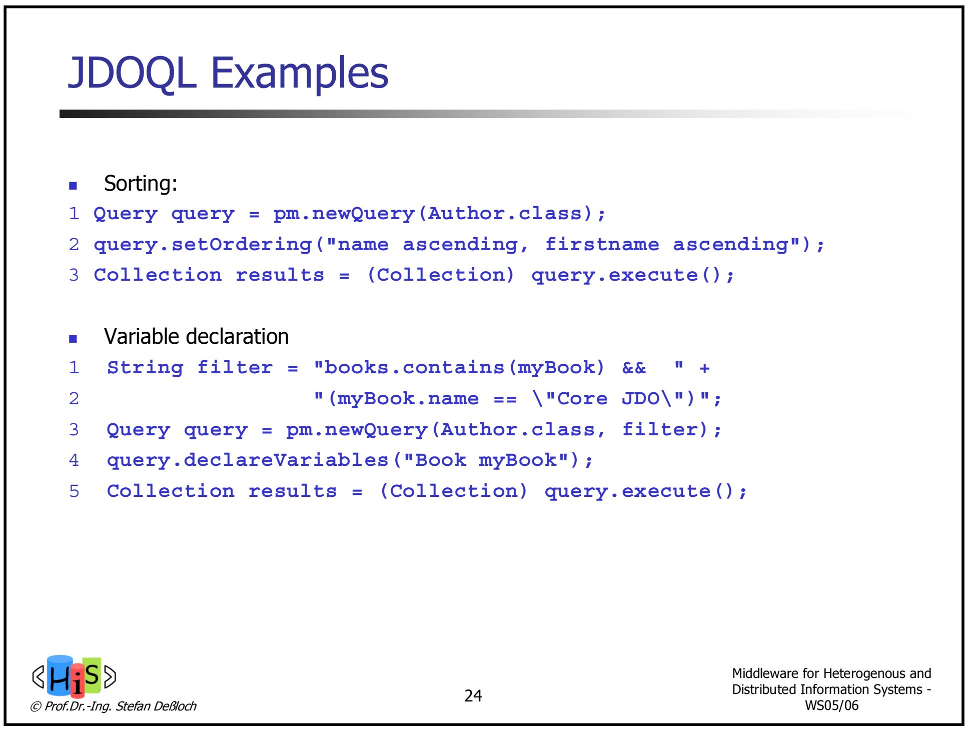
- A JDOQL query has 3 parts
  - *candidate class*: class(es) of expected result objects
    → retriction at the class level
  - *candidate collection*: collection/extent to search over
    → (optional) restriction at the object extent level
  - *filter*: boolean expression with JDOQL (optional: other query language)
- JDOQL characteristics
  - read-only (no INSERT, DELETE, UPDATE)
  - returns JDO objects (no projection, join)
  - query submitted as string parameter → dynamic processing at run-time
  - logical operators, comparison operators: e.g. !,==,>=
  - JDOQL-specific operators: type cast using "( )", navigation using "."
  - no method calls supported in JDOQL query
  - sort order (ascending/descending)
  - variable declarations



Query

- JDO-Query with JDOQL for locating JDO instances:

```
String searchname = "Doe";
Query q = pm.newQuery();
q.setClass(Author.class);
q.setFilter("name == \"" + searchname +"\"");
Collection results =(Collection)q.execute();
Iterator it = results.iterator();
while (it.hasNext()){
    // iterate over result objects
}
q.close(it);
```



JDOQL Examples

- Sorting:

```
Query query = pm.newQuery(Author.class);
query.setOrdering("name ascending, firstname ascending");
Collection results = (Collection) query.execute();
```

- Variable declaration

```
String filter = "books.contains(myBook) &&  " +
                "(myBook.name == \"Core JDO\")";
Query query = pm.newQuery(Author.class, filter);
query.declareVariables("Book myBook");
Collection results = (Collection) query.execute();
```

Summary

- Object persistence supported at various levels of abstraction
  - CORBA
    - standardised "low-level" APIs
    - powerful, flexible, but no uniform model for component developer
      - various persistence protocols
    - explicit vs. implicit (transparent) persistence
  - EJB/J2EE
    - persistent components
      - CMP: container responsible for persistence, maintenance of relationships
    - uniform programming model
    - transparent persistence
  - JDO
    - persistent Java objects
    - orthogonal persistence
- Mapping of objects to specific types of data stores (e.g., relational)
  - capabilities are not standardized, left to the vendors

Summary (2)

- Query Support
  - CORBA: queries over object collections using SQL, OQL
    - persistent object schema?
  - EJB: queries over abstract persistence schema
    - limited functionality, only for definition of Finder methods
    - more or less an SQL subset
  - JDO: queries over collections, extents
    - limited functionality
    - proprietary query language
  - queries over multiple, distributed data sources are not mandated by the above approaches