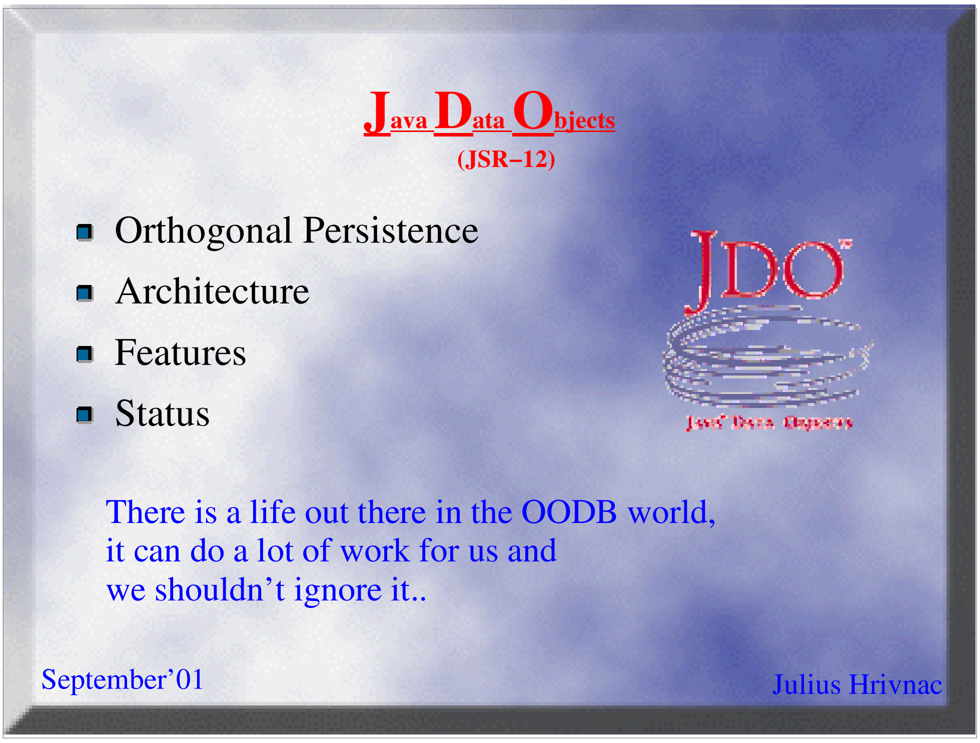


- In the Java Note (year ago), we wrote that persistency is not resolved in Java. It is no more true. Java now offers persistency APIs superior to those of C++ (JDO, JDBS, jSQL, RootIO–Java, Objy–Java, Oracle–Java). Also Java 1.4 (Merlin) delivers significant improvement in Java IO speed.
- We are doing a lot of useless work, there are products which do it much better than we can ever do.
- This talk only touches the basic features of JDO.

Main Requirements / Features

- Transparent persistence (= Transient–Peristent separation)
- No need for new language to describe persistency
- Portability (platform independence)
- Range of implementations (no lock in one DB vendor)
- Persistency for 3rd party objects
- Data store technology independence:
  - OODB
  - RDB
  - HDB
  - Files

Transparent Persistence

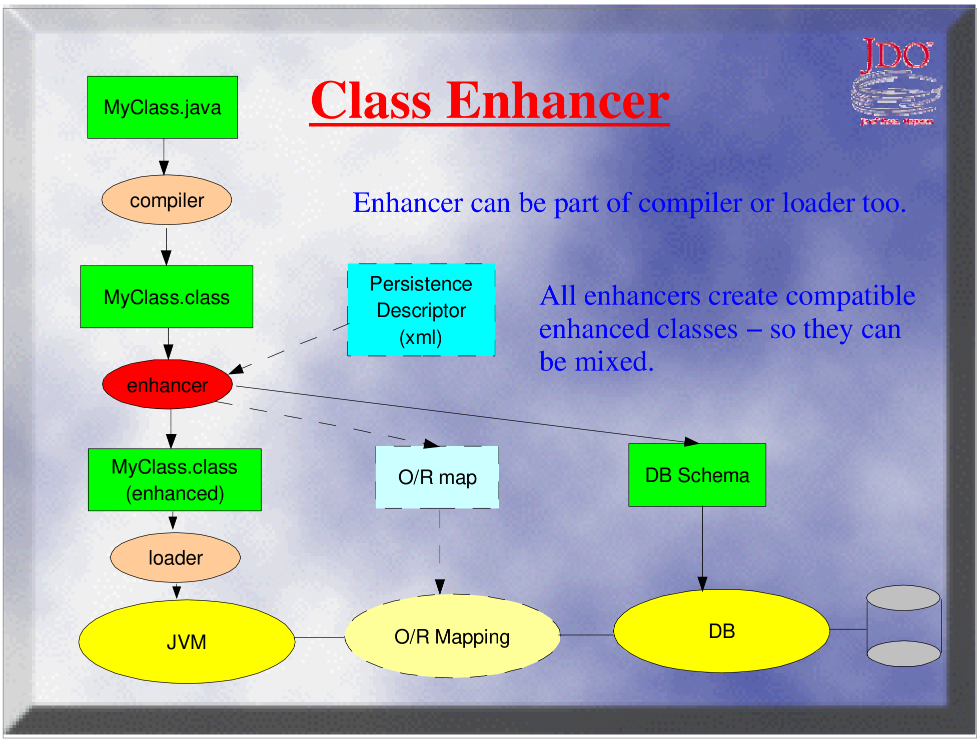
Transient class

```
public class Hit {
  public Hit(double x,
             double y,
             double z) {
    m_x = x;
    m_y = y;
    m_z = z;
    }
  public double x() {
    return m_x;
    }
  public double y() {
    return m_y;
    }
  public double z() {
    return m_z;
    }
  private double m_x = 0;
  private double m_y = 0;
  private double m_z = 0;
  }
```

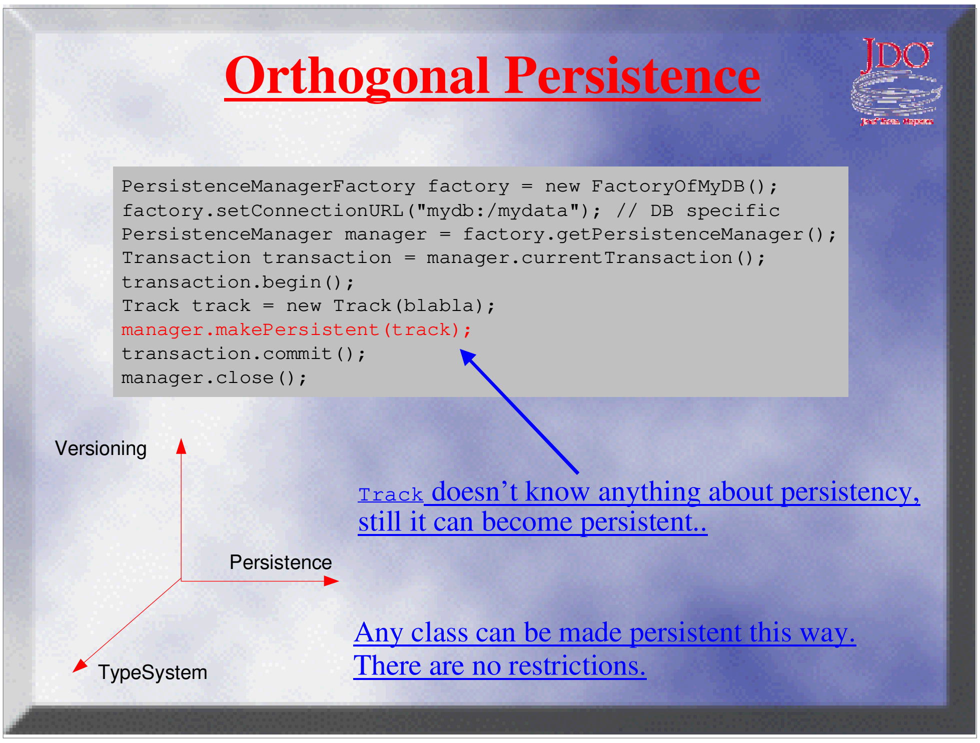
Persistent class

```
public class Hit {
  public Hit(double x,
             double y,
             double z) {
    m_x = x;
    m_y = y;
    m_z = z;
    }
  public double x() {
    return m_x;
    }
  public double y() {
    return m_y;
    }
  public double z() {
    return m_z;
    }
  private double m_x = 0;
  private double m_y = 0;
  private double m_z = 0;
  }
```

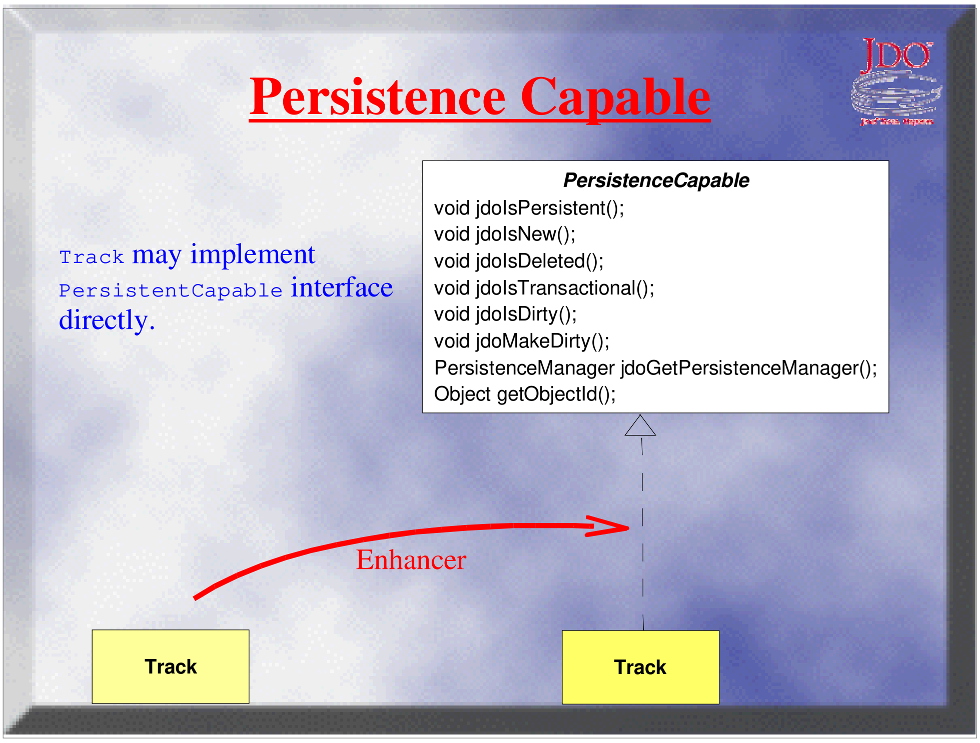
No additional code is required to make a class persistent..



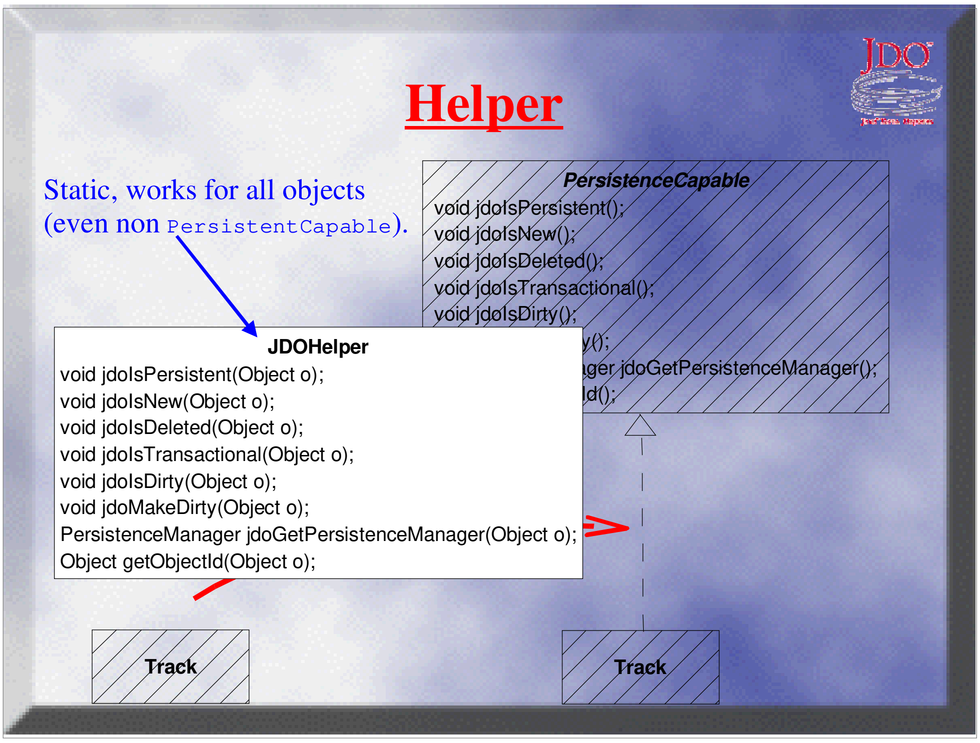
- This is the most easy (reference) implementation, but class can be enghanced at any stage:source postprocessing, while compiling, while loading,...



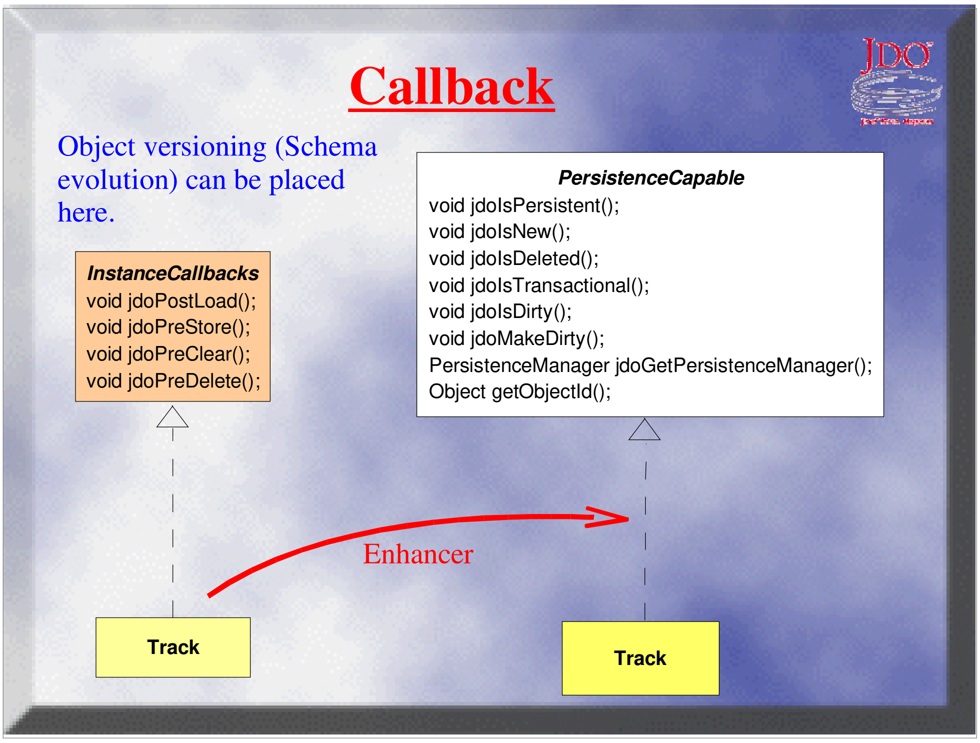
- Class can be made persistent, if it is completely defined by its fields. So, for example, classes with native interface, Threads, Sockets or Files can't be made persistent.
- Strictly speaking, JDO doesn't implement exact Orthogonal Persistency, but the differentces are minor and irrelevent for us.
- Orthogonal versioning implements Class evolotion (Scheme evolution), it uses similar architecture as Orthogonal persistence, but it is not much developed.



- Enhancer change normal class into PersistenceCapable class, but user can do it herself.
- Source (.java) doesn't know about persistency, run–time does know it.



- Helper is the prefered way of interaction with JDO.

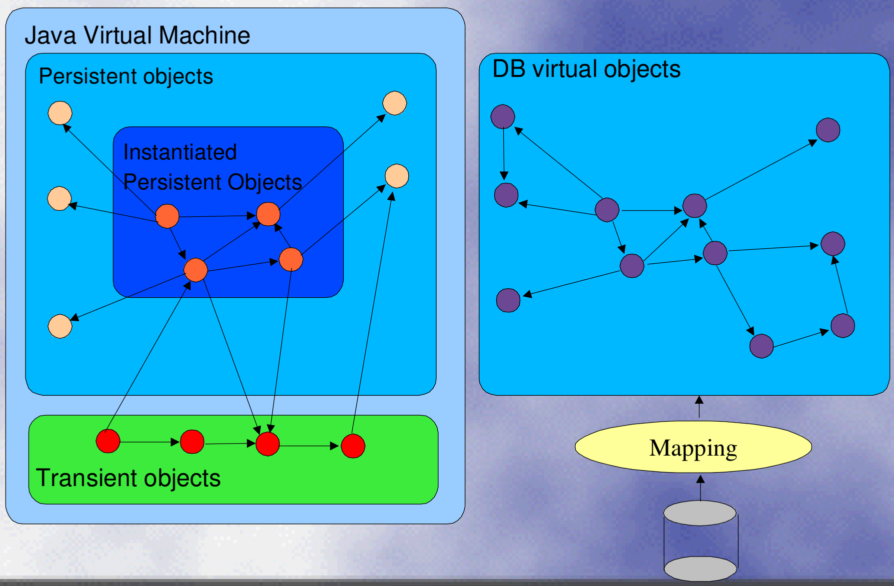


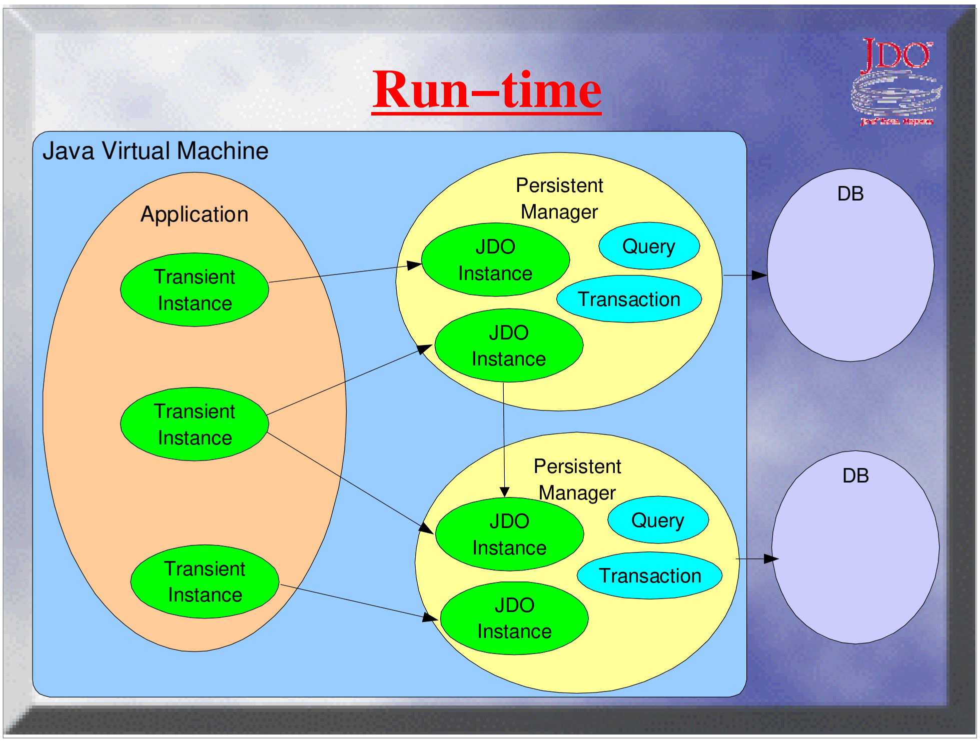
- Used for extending persistency mechanism.
- Direct implementation can be usefull for example to generate common OID.

Objects

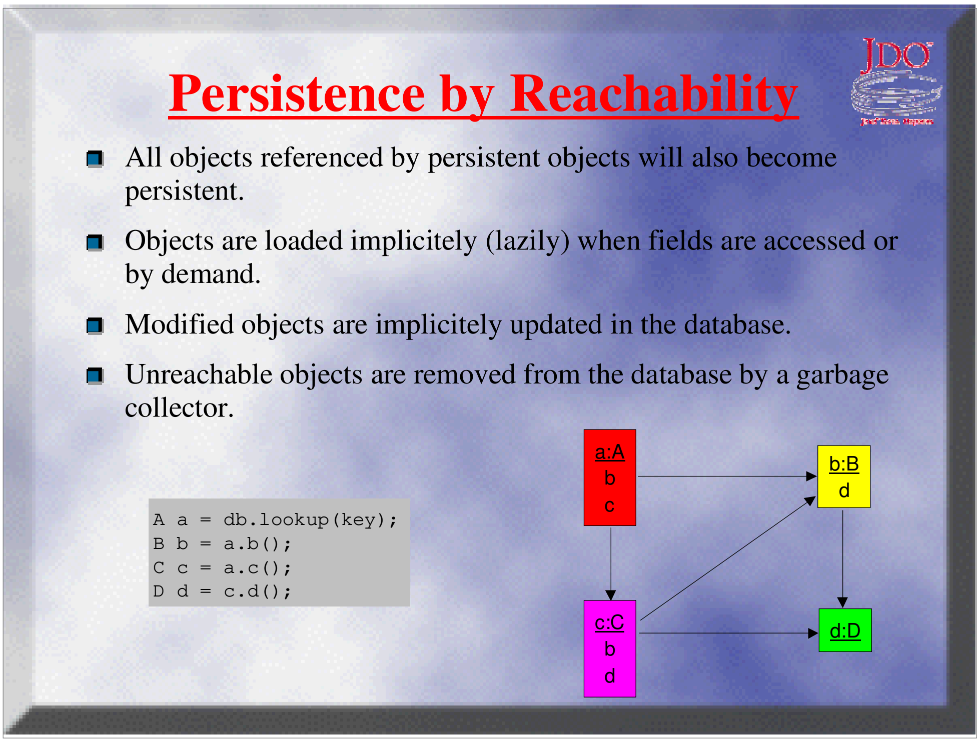
- Classes can be:
  - Persistece–capable (enhanced, can have transient or peristent instances)
  - Transient (can't be persistent, system classes like Thread, classes with native components,...)
  - Persistence–aware (enhanced, can't be persistent, but can access public data of persistent classes)
- Objects can become persistent as:
  - First Class object (persistent by itself, it has its OID)
  - Second Class object (persistent due to its relationship to some First Class object, it doesn't have its OID, like StoreGate contained object)
- Fields can be:
  - Persistent (managed by JDO)
  - Transactional non–persistent (partly manged by JDO)
  - Non–transactional non–persistent (managed by application)

Object Model

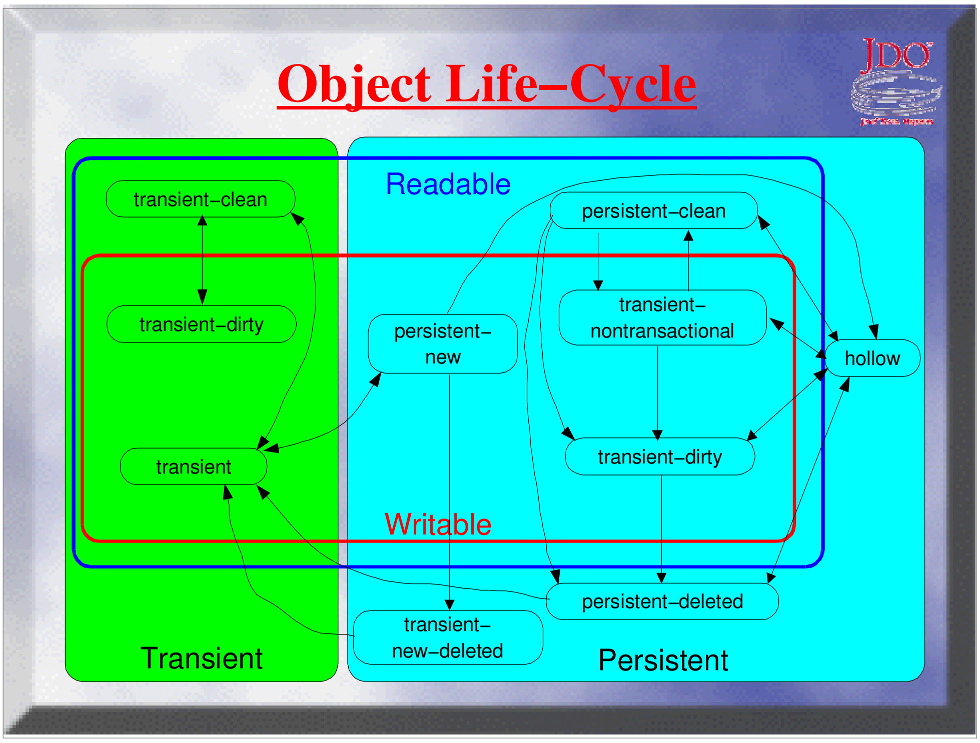




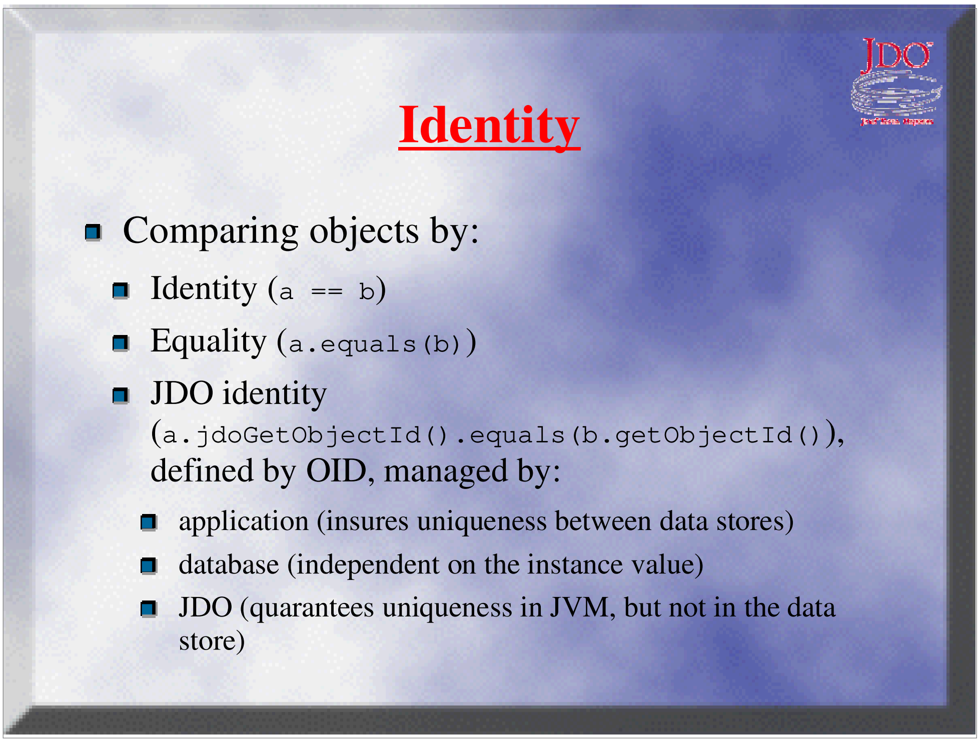
- Objects can be in different databases as all Enhancers should create compatible code.
- Database can be local or distributed (via application server).



- Common concept for Java Databases.
- b should be First Class object, otherwise it can't be shared between a and c. Otherwise, one would have two copies of b. This is similar to StoreGate problems.

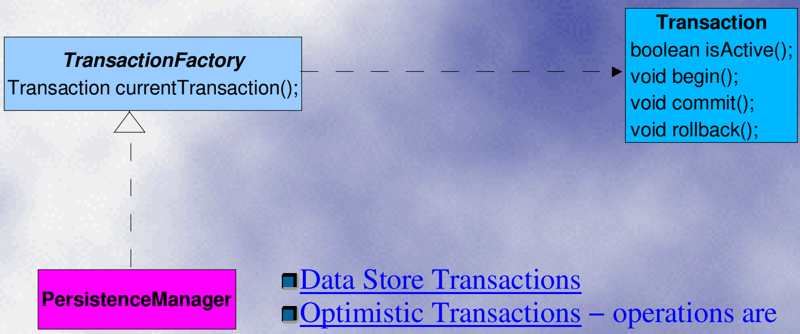


- It helps to understand and manage object' lifecycle.
- Transitions correspond to jdo methods.

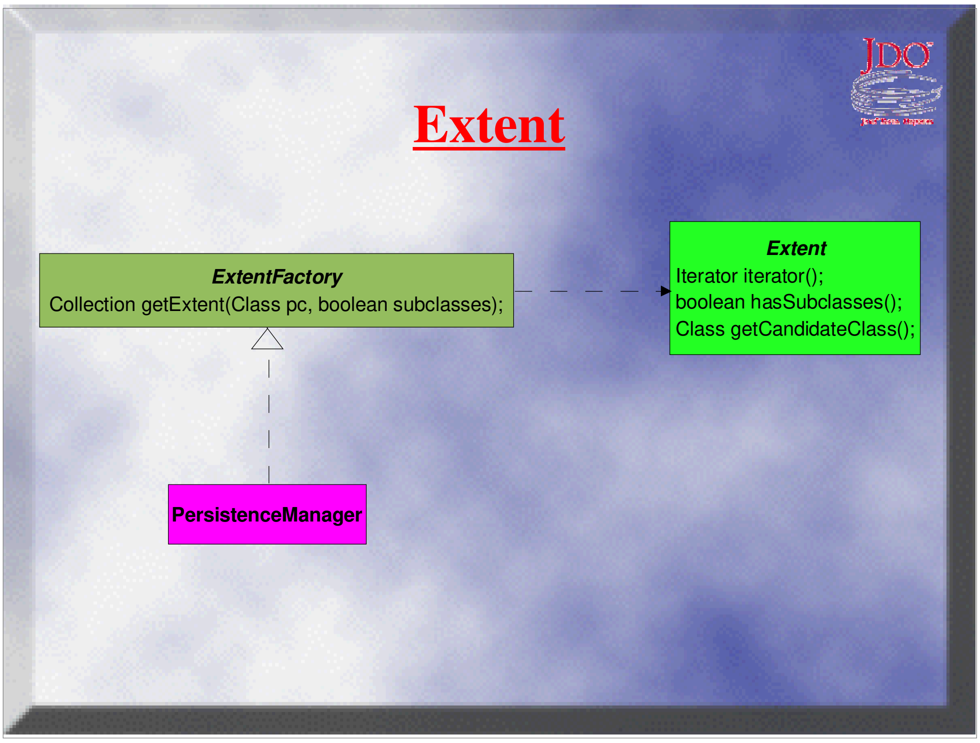


- Identity and Equality are standard Java concepts.

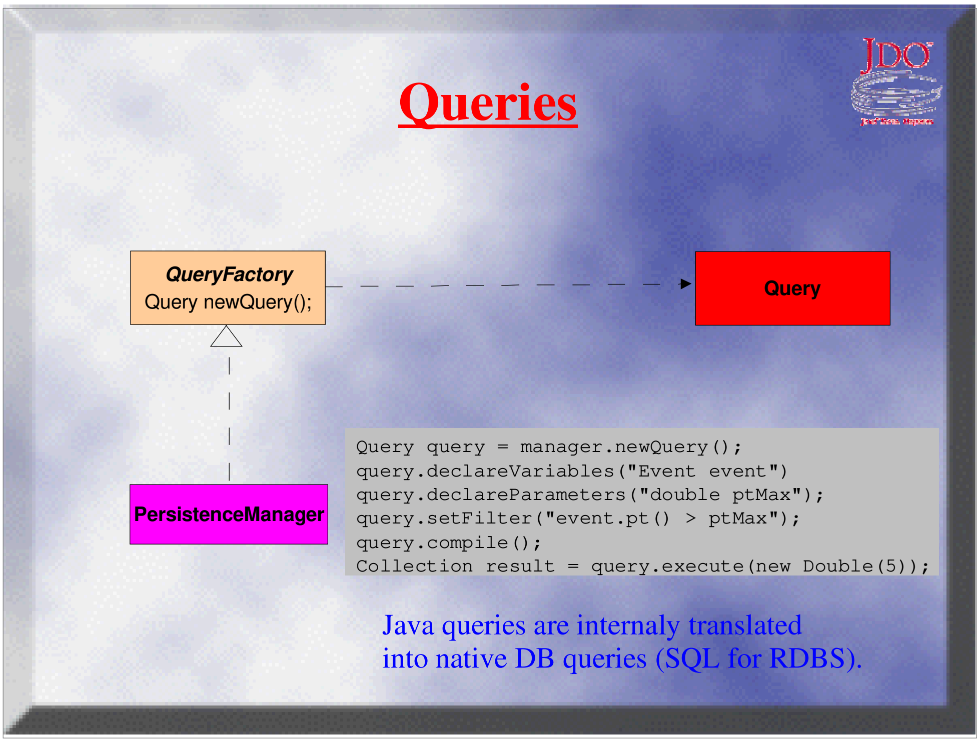
Transcriptions



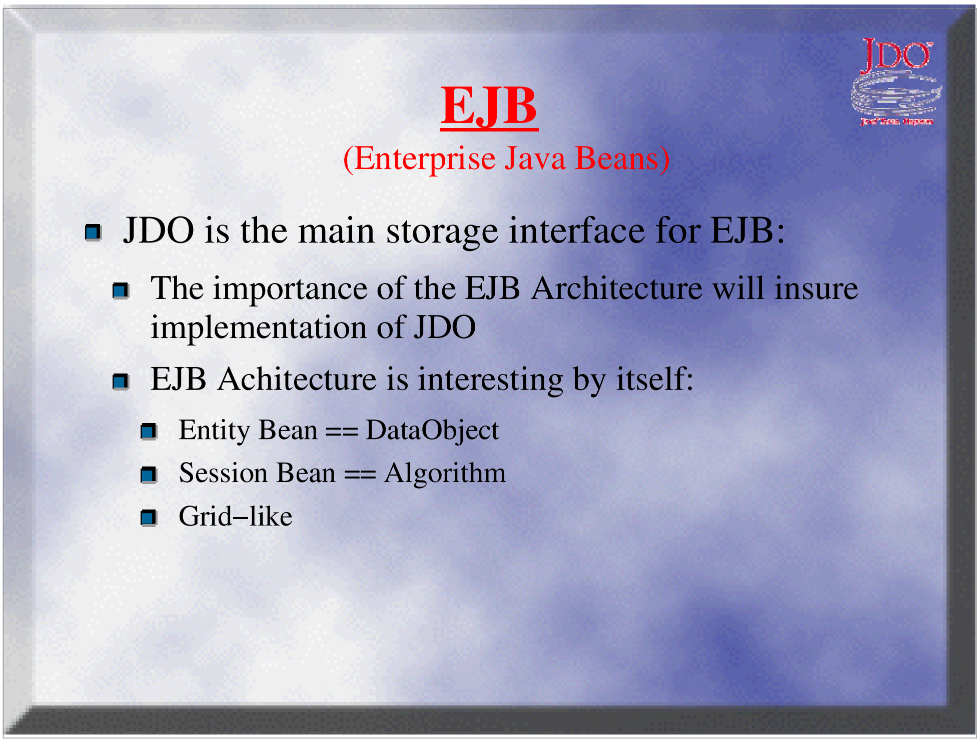
- Data Store Transactions
- Optimistic Transactions – operations are performed imemdiately using local store; during flush, consistency is verified



- Allows searching using object types.
- Similar concept exists in StoreGate.



- Queries are expressed in Java itself (not in additional language like SQL), but they have "set" semantics.
- Native DB queries for RDBS are efficient.
- Extent is used.



- Worth to look at.

Status

- Java Comunity Process Open Standard JSR–12:
  - Approved: July'99
  - Public review of specification done: July'00
  - 1.0 Proposed Final Draft (ver. 0.95)
  - Reference Implementation: Eoy'01
  - Compatibility Test Suite: Eoy'01
- Expert Group:
  - Excelon, Informix, Libelis, Oracle, Poet, Sun, Versant, Forte, IBM, Objectivity, Rational, Secant,...
- ODMG standard for Java persistency.
- Specification Leader:
  - Craig Russel (ODMG Chairman)

- Already now more developed than whatever we could ever develop.

Implementations

as of 23Aug'01

Product (Vendor)	Supported Databases	Status	Licence	Version
Reference (JSR)	Files	50%	Sun(Free, Free Source)	Beta 1.0
	SQL (JDBC)	5%		
Judo (Versant)	Versant	70%		0.5.0
OpenFusion (PrismTech)	Oracle	80%		Beta 1.0
	Sybase	80%		
	Informix	70%		
	IBM DB2	5%		
	MS SQL Server	5%		
Kodo (TechTrader)	Oracle	70%		Beta 2.1.2
	MS SQL Server	5%		
	InstantDB	5%		
	PostgreSQL	5%		
	MySQL	5%		
FastObjects (Poet)	Poet	50%		
Sparrow (SourceForge)	SQL	10%	GPL	Pre-Alpha
Orient (OrientTechnologies)	Orient	5%		2.0
Castor (Exolab)	PostgreSQL	95%	Open Source	0.9.3
	MySQL	95%		
	Interbase	5%		
	InstantDB	5%		
	Hypersonic SQL	5%		
Diamond (Gemstone)	Gemstone	5%		Alpha
LiDO (LIBeLIS)	Versant	5%	Open Source	1.0.0
...	...		...	...

- My subjective view.
- New implementations / versions are comming very quickly.
- Castor is not 100% compatible.

Summary

- JDO is an OO interface between an application and a database (OO or R), it isn’t the database itself
- JDO uses up–to–date technology
- JDO satisfies requirements on Persistent–Transient separation (more cleanly than any other existing implementation)
- JDO is well supported
- JDO doesn’t suffer from C++ problems
- JDO scalability comes directly from the interfaced DB scalability

Conclusion

- JDO is the standard technology for interfacing OODB in Java:
  - we should concider it when defining our Multi–Language strategy
- JDO uses very similar technology to Athena (Transient–Persistent Separation, StoreGate):
  - so our Architecture is not silly
  - should we realy do it ourselves ?
- JDO has real modern OODB architecture:
  - we should learn from it even when we don’t use it directly

http://access1.sun.com/jdo