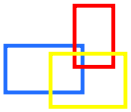
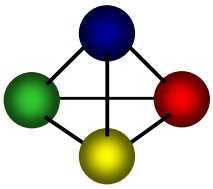
VERSANT

Java Data Objects
The Future for Java Object Persistence

Keiron McCammon
CTO
Versant Corporation
kmccammon@versant.com

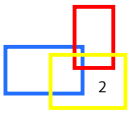


Overview



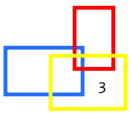
- What is JDO?
- JDO Goals
- How does JDO work?
- Using JDO
- JDO and EJB
- Looking forward…

VERSANT



What is JDO?

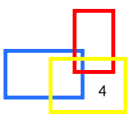
VERSANT



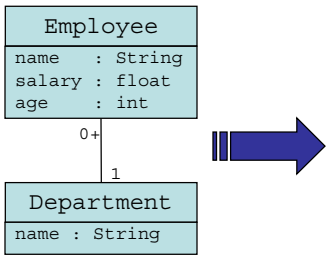
Java Data Objects (JDO)

- Standard for transparent Java object persistence
  - Provides developers with a Java-centric and object view of persistence and data store access
- Designed to allow pluggable vendor "drivers" for accessing any database/data store
- Designed to work in conjunction with Application Servers
  - "Connector Architecture" used to specify the contract between JDO Vendor and Application Server for instance, connection, and transaction management

VERSANT

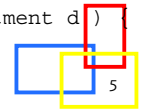


An Example - Creating the classes




```
public class Employee {
  private String     name;
  private int        age;
  private float      salary;
  private Department department;

  public Employee ( String name, int age ) {
    this.name = name;
    this.age  = age;
  }
  public String getName ( ) {
    return name;
  }
  public int getAge ( ) {
    return age;
  }
  ...
  public Department getDepartment ( ) {
    return department;
  }
  public void setDepartment ( Department d ) {
    department = d;
  }
}
```

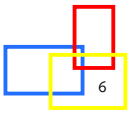
VERSANT



An Example - Adding persistence

```
static void main ( String[] args ) {

  // Need to get a database connection
  ...

  Department dept = new Department("R&D");

  Employee emp = new Employee("Joe Bloggs", 30);

  emp.setDepartment(dept);

  // Committing the transaction stores the
  // new instances in the database
  ...
}
```

VERSANT

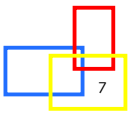


http://jcp.org/jsr/detail/012.jsp

Java Community Process

- Standard driven by the Java Community
- JDO is a Java Specification Request
  - JSR-000012
  - Specification "lead" heads expert group who propose formal specification
  - Participants & public review specification
  - Reference implementation and compatibility tests required prior to publication
  - Standard approved by JSR Executive Committee

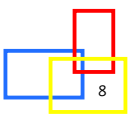
VERSANT



Expert Group Members

- Alagic
- Ericsson
- Forte Software
- IBM
- Informix Software
- Lawson Software
- LIBeLIS
- Object People
- Objectivity
- Oracle
- Poet Software
- Rational Software
- SAP AG
- Secant Technologies
- Silverstream Software
- Software AG
- Sun Microsystems
- Tech@spree
- **Versant**

VERSANT



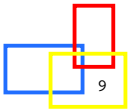
http://access1.sun.com/jdo/index.html

JDO Current Status

- JSR-000012 approved July-1999
- Specification lead selected July-1999
  - Craig Russell, SUN Microsystems
- Expert group formed August-1999
  - Expert group reviews specification before release
  - Versant is a member of the expert group
- First public review completed July-2000
- Second public review started June-2001
- Reference implementations and Technology Compatibility Kit (TCK) development underway

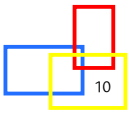
Approval Imminent

VERSANT



JDO Goals

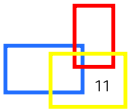
VERSANT



Goals of the JDO Architecture

- Transparent "object" persistence
  - Minimal to 0 constraints on building classes
  - No new data types or data access language
    - DDL & DML is Java
- Use in a range of implementations
  - J2ME - Embedded, device-oriented
  - J2SE - Client/server
  - J2EE - Enterprise Java Beans
- Data store independence
  - Relational, object, object-relational, hierarchical, file systems, ...

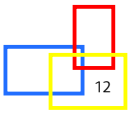
VERSANT



JDO Audience

- Java application developers
  - Transparent object persistence
  - Java-centric, no need to know how to access a database
- EJB application developers
  - Connection pooling & transaction management via Application Server
  - Transparent database access for non-CMP solutions (Session Beans & BMP)
    - No need to use JDBC directly
    - Object Queries to find instances

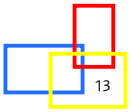
VERSANT



JDO versus JDBC

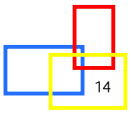
- Not meant to replace it!
  - Complimentary technology
  - Standardizes object access to data stores
  - Programmer just sees Java classes
- JDO for RDBs likely implemented on top of JDBC
  - JDBC useful for direct control over database access and connection management
  - JDBC is mature, widely accepted and understood
  - JDBC supported by major database vendors

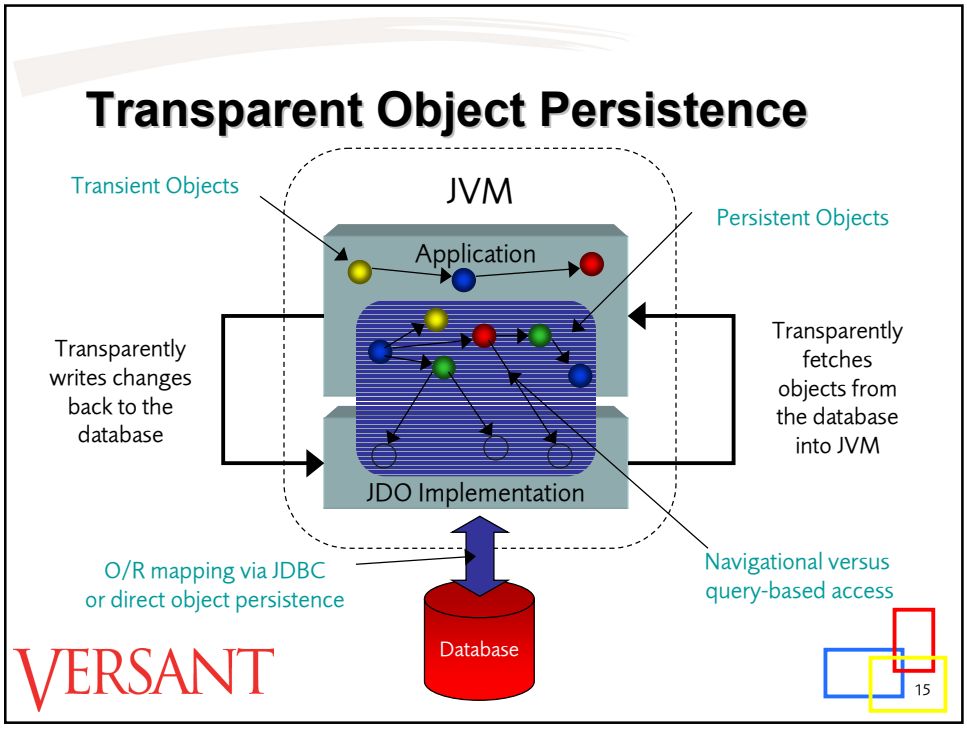
VERSANT

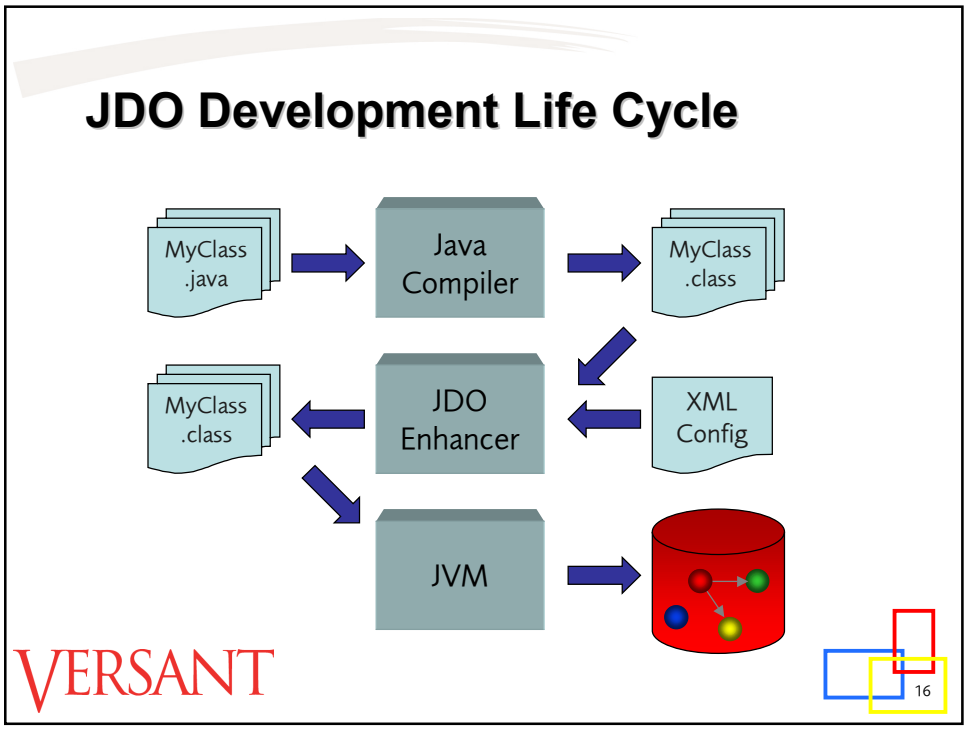


How does JDO work?

VERSANT



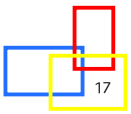

Transparent Object Persistence
Transient Objects
JVM
Persistent Objects
Application
Transparently writes changes back to the database
Transparently fetches objects from the database into JVM
JDO Implementation
O/R mapping via JDBC or direct object persistence
Navigational versus query-based access
Database
VERSANT
15


JDO Development Life Cycle
MyClass .java
Java Compiler
MyClass .class
MyClass .class
JDO Enhancer
XML Config
JVM
VERSANT
16

JDO Enhancer - What does it do?

- Reads byte code and generates new byte code
  - Adds hooks to enable JDO implementation to transparently:
    - Retrieve objects
    - Track changes to object state
    - Write changes to data store on commit
- Developer doesn't have to explicitly fetch/store objects

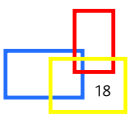
VERSANT

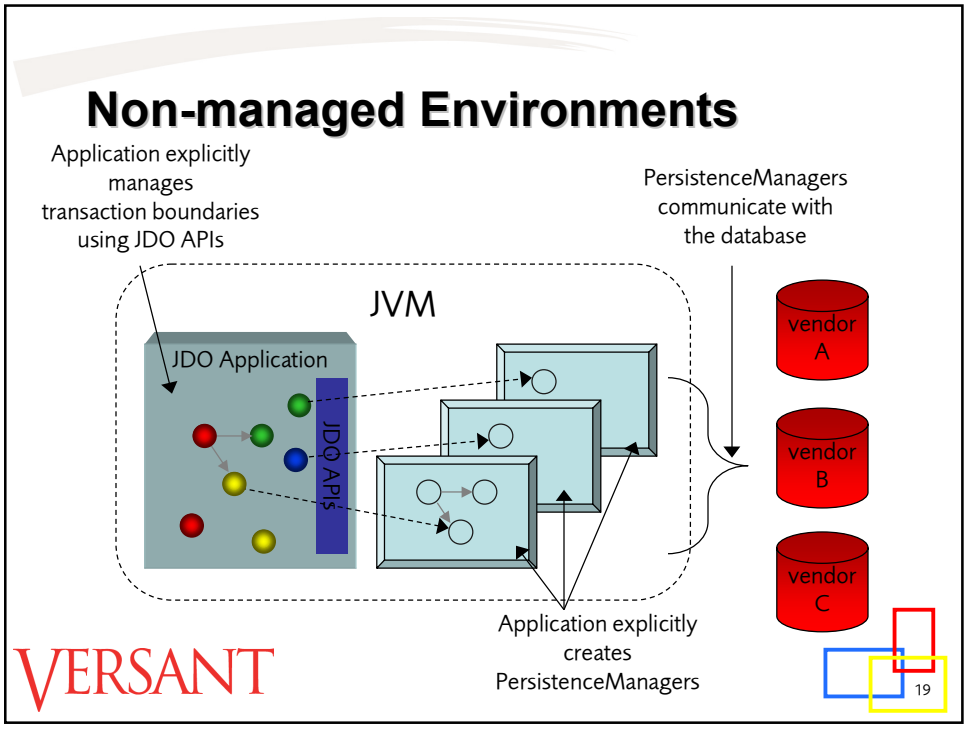


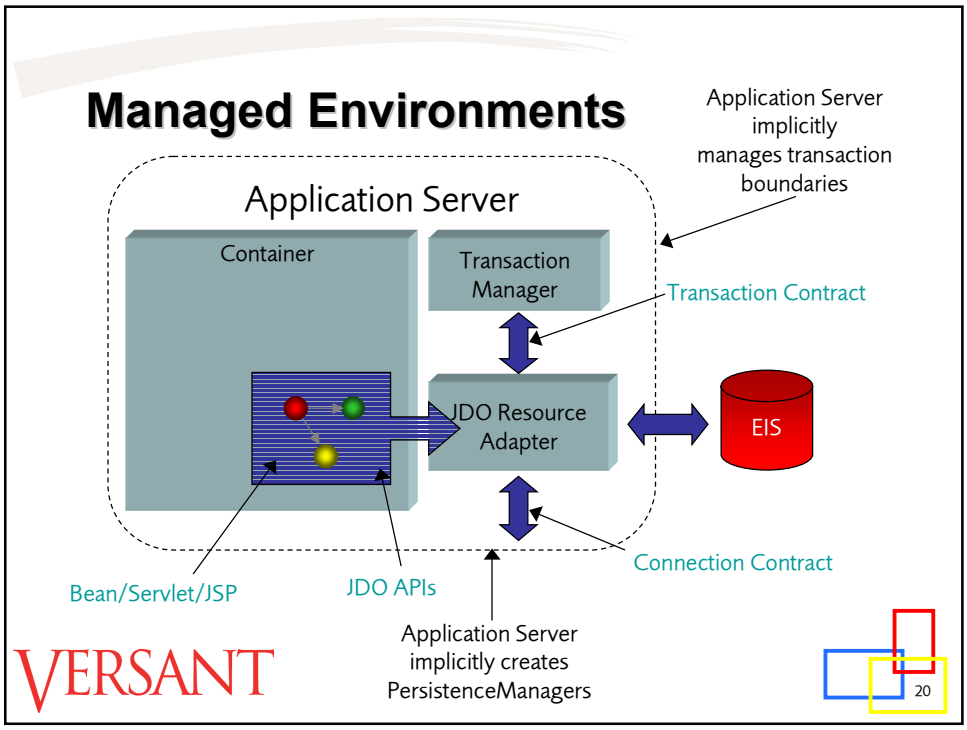
Non-managed versus Managed

- Two ways of developing JDO applications
- Non-managed Environments
  - Client/Server, 2-tier
  - Explicit connection and transaction management
- Managed Environments
  - Application Server (EJB), n-tier
  - Implicit connection and transaction management

VERSANT

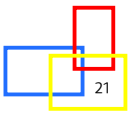



Non-managed Environments
Application explicitly manages transaction boundaries using JDO APIs
PersistenceManagers communicate with the database
JVM
JDO Application
JDO APIs
vendor A
vendor B
vendor C
Application explicitly creates PersistenceManagers
VERSANT
19


Managed Environments
Application Server implicitly manages transaction boundaries
Application Server
Container
Transaction Manager
Transaction Contract
JDO Resource Adapter
EIS
Connection Contract
Bean/Servlet/JSP
JDO APIs
Application Server implicitly creates PersistenceManagers
VERSANT
20

Using JDO

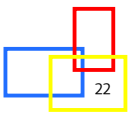
VERSANT



JDO Interfaces and Classes

- PersistenceManagerFactory (Interface)
- PersistenceManager (Interface)
- Transaction (Interface)
- Query (Interface)
- PersistenceCapable (Interface)
- InstanceCallbacks (Interface)
- JDOHelper
- JDO Exception Classes
  - …

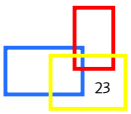
VERSANT



An Overview

- Use PersistenceManagerFactory to get a PersistenceManager
  - PersistenceManager embodies a database connection
- Use a PersistenceManager to create a Transaction or a Query
- Use a Transaction to control transaction boundaries
- Use a Query to find objects by value
- Enhanced classes implicitly implement PersistenceCapable
- PersistenceCapable classes can implement InstanceCallbacks

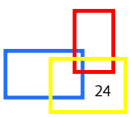
VERSANT



JDO Object Model

- Support for all Java field types
  - Primitives, object references, interfaces
  - Exception: References to system-defined classes
- Support for all Java class and field modifiers
  - Public, private, protected, static, transient, abstract, final, synchronized, volatile
- Support for all user-defined Java classes
  - Exception: any classes that depend on state of inaccessible or remote objects
    - java.net.SocketImpl
  - Exception: any classes that use native methods

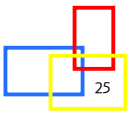
VERSANT



PersistenceManagerFactory (Interface)

- Standard mechanism to get PersistenceManager instances
  - May implement resource pooling and connection management
- Implements java.io.Serializable
  - Support for lookup via JNDI
- Uses JavaBeans pattern for get/set Properties
  - Standard properties
    - ConnectionUserName
    - ConnectionPassword
    - ConnectionURL
    - ...

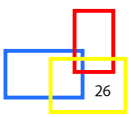
VERSANT



PersistenceManager (Interface)

- Primary interface to the "object cache"
  - Cache management methods
    - Refresh/release objects
- Provides management of PersistenceCapable objects
  - Identity management methods
  - Life-cycle management methods
- Acts as factory for other JDO classes
  - Query creation methods
  - Transaction creation methods
- Use to get Collection of all instances of a class
  - Class extent methods

VERSANT



PersistenceManager Methods

- Identity Methods
  - Get the JDO Identity of a JDO Instance

```
Object getObjectId ( Object pc )
```

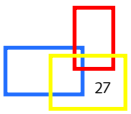
  - Get a JDO Instance given its JDO Identity

```
Object getObjectById ( Object oid )
```

- Lifecycle Methods

```
void makePersistent ( Object pc )
void deletePersistent ( Object pc )
void makeTransient ( Object pc )
void makeTransactional ( Object pc )
void makeNontransactional ( Object pc )
```

VERSANT

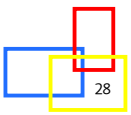


Class Extents

- Collection of all object instances of a given class managed by the data store
- Available for any PersistenceCapable class

```
Extent getExtent ( Class   pc,
                   boolean subclasses )
```

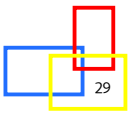
VERSANT



JDO Query Objectives

- Query language neutral
  - Optional support for SQL, OQL, etc.
  - Optimizations possible for specific query languages
- Multi-tier architecture
  - Entirely in-memory
  - Server-side (data store query engine) execution
- Support for Large result sets
- Support for "compiled" queries

VERSANT



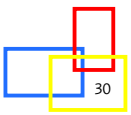
Query (Interface)

- PersistenceManager is the Query factory

```
Query newQuery ( Class      cls,
                 Collection cln,
                 String     filter )
```

- Queries filter Collections and return Collections
- Required elements in a query
  - Collection of candidate instances
    - May be a class extent
    - May be a Collection in the JVM
  - Class (type) of the result set
  - Filter (Java boolean expression)
- Optional elements in a query
  - Parameter & variable declarations; Imports; Ordering

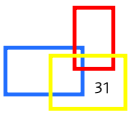
VERSANT



Query "Filters"

- Filters are Java boolean expressions
- Identifiers are class attributes
- Navigation via '.' notation
  - Support for single object navigation
  - Support for collections via "contains()" method
  - Support for wildcards via "startsWith()" & "endsWidth()"
- Support for parameter substitution and variables

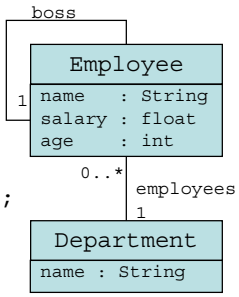
VERSANT



"Filters": Simple example

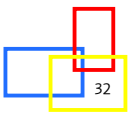
- Find well compensated employees

```
String filter = "salary > 100000";
```

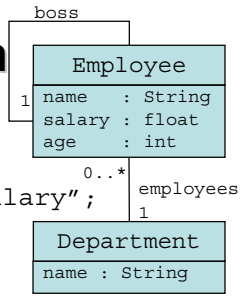



```
Extent     employees = pm.getExtent(Employee.class);
Query      query     = pm.newQuery(employees, filter);
Collection results   = (Collection) query.execute();
```

VERSANT



"Filters": Object Navigation

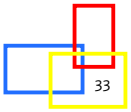


- Find over compensated employees

```
String filter = "salary > boss.salary";
```

```
Extent     employees = pm.getExtent(Employee.class);
Query      query     = pm.newQuery(employees, filter);
Collection results   = (Collection) query.execute();
```

VERSANT



Using Parameters & Variables

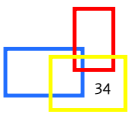
- Support for parameterized queries
  - Parameters are substituted at execution time
  - Parameters are typed
  - Parameters specified using Java syntax
  - Multiple statements separated by semicolons

```
query.declareParameters ("float salary");
```

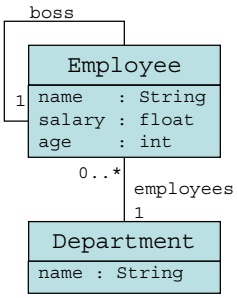
- Support for use of variables in queries
  - Variables are typed
  - Variables specified using Java syntax
  - Multiple statements separated by semicolons

```
query.declareVariables ("Employee emp");
```

VERSANT



"Filters": Complex example

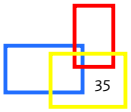


- Find all Departments with at least one well-compensated Employee

```
String filter = "
  employees.contains(emp) &&
  emp.salary > salary";
```

```
Extent     depts   = pm.getExtent(Department.class);
Query      query   = pm.newQuery(depts, filter);
query.declareParameters("float salary");
query.declareVariables("Employee emp");
Collection results =
    (Collection) query.execute(new Float(100000.0));
```

VERSANT



Imports & Ordering

- Imports
  - Imports specified using Java syntax
  - Multiple statements separated by semicolons

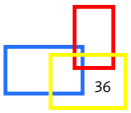
```
query.declareImports ("import example");
```

- Ordering
  - Ordering can be "ascending" or "descending"

```
query.setOrdering ("salary ascending");
```

  - Ordering can include navigation via '.' notation

VERSANT



Transaction (Interface)

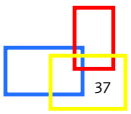
- PersistenceManager is the Transaction factory

```
Transaction currentTransaction ()
```

- Transactions can be "unmanaged" (local) or "managed" (distributed)
  - Explicit control versus implicit control
  - Designed to work in Embedded & Enterprise environments
- Support for data store (default) and optimistic (optional) concurrency models
- Methods for "unmanaged" transactions

```
isActive ()
begin ()
commit ()
rollback ()
```

VERSANT



PersistenceCapable (Interface)

- Implemented by a "Persistence Capable" Class

```
PersistenceManager
  jdoGetPersistenceManager ( )
```

Get Persistence Manager for this instance

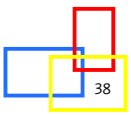
```
Object jdoGetObjectId ( )
```

JDO Object Identity

```
boolean jdoIsPersistent ( )
boolean jdoIsTransactional ( )
boolean jdoIsDirty ( )
boolean jdoIsNew ( )
boolean jdoIsDeleted ( )
```

Status Interrogation

VERSANT

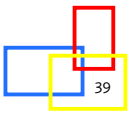


JDOHelper

- Static methods that mirror PersitenceManager & PersistentCapable interfaces
  - Delegates to appropriate interface
- Simplifies management of persistent objects

```
static Object getObjectId ( Object pc )

static boolean isPersistent ( Object pc )
static boolean isDirty ( Object pc )
static boolean isNew ( Object pc )
static boolean isDeleted ( Object pc )
…
```

VERSANT



JDO Identity

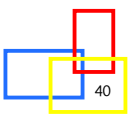
- Object identity implemented by JVM
  - Is this the same object instance?

```
obj1 == obj2
```

- Object equality implemented by class developer
  - Is this the same object?

```
obj1.equals (obj2)
```

VERSANT

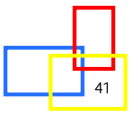


JDO Identity (cont'd)

- JDO identity implemented by JDO vendor
  - Can be based on primary key
    - Defined by application, enforced by database
  - Can be managed by the data store
    - Not related to any attribute value
  - Can be managed by JDO implementation
    - Guarantee uniqueness in the JVM but not datastore

```
obj1.jdoGetObjectId().equals(obj2.jdoGetObjectId())
```

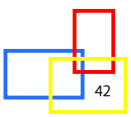
VERSANT



"Uniquing" in JDO

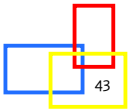
- JDO instances representing the same data store "object" exist only once per PersistenceManager
- Regardless of how the instance is obtained
  - Queries
  - Navigation
  - Other PersistenceManager methods

VERSANT

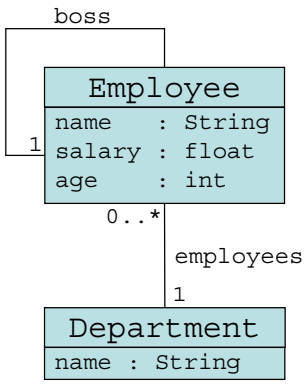


An Example

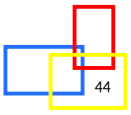
VERSANT



How does it work? An example...



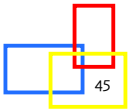
VERSANT



An Example: Employee Class

```
public class Employee {
  private String     name;
  private int        age;
  private float      salary;
  private Employee   boss;
  private Department department;

  public Employee ( String name, int age ) {
    this.name = name;
    this.age  = age;
  }
  public String getName ( ) {
    return name;
  }
  ...
  public Department getDepartment ( ) {
    return department;
  }
  public void setDepartment ( Department d ) {
    department = d;
  }
}
```

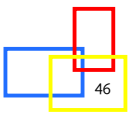
VERSANT



An Example: Department Class

```
public class Department {
  private String name;
  private Set    employees = new HashSet();

  public Department ( String name ) {
    this.name = name;
  }
  public String getName ( ) {
    return name;
  }
  ...
  public void addEmployee ( Employee emp ) {
    emp.setDeparment(this);
    employees.add(emp);
  }
}
```

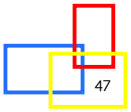
VERSANT



XML Metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
  <package name = "com.versant.jdoexample"/>
  <class name = "Employee"/>
  <class name = "Department">
    <field name = "employees">
      <collection element-type="Employee"/>
    </field>
  </class>
</jdo>
```

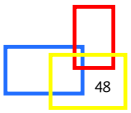
VERSANT



An Example: Connecting

```
static void main ( String[] args ) {

  // Create a Vendor specific factory
  PersistenceManagerFactory factory =
    new VersantPersistenceManagerFactory();

  // Set connection parameters
  factory.setConnectionURL(args[0]);

  // Get a PersistenceManager
  PersistenceManager pm =
    factory.getPersistenceManager();
```

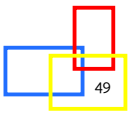
VERSANT



An Example: Find a Department

```
// Begin a Transaction
Transaction tx = pm.currentTransaction();
tx.begin();

// Construct the Query
Extent     depts   = pm.getExtent(Department.class);
String     filter  = "name = dept";
Query      query   = pm.newQuery(depts, filter);

// Execute the Query
query.declareParameters("String dept");

Collection results = pm.execute(args[1]);

// Extract the Department from the result set
Department dept =
  (Department) results.iterator().next();
```

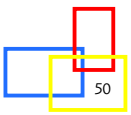
VERSANT



An Example: Create an Employee

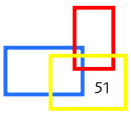
```
// Create a new Employee
Employee emp =
  new Employee(args[2], Integer.parseInt(args[3]));

// Add Employee to Department
dept.addEmployee(emp);

// Commit transaction
tx.commit();
}
```

VERSANT



JDO and J2EE

VERSANT

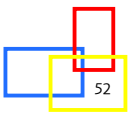


Java Connector Architecture

- Mandated as plug-in for non-JDBC data access
  - Deals with connection, transaction & security management
- Common Client Interface
  - Provides standard APIs to get a connection

```
javax.resource.cci.ConnectionFactory
javax.resource.cci.Connection
```

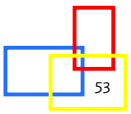
VERSANT



JDO & JCA

- PersistenceManagerFactory maps to ConnectionFactory
- PersistenceManager maps to Connection
  - Container-managed transactions
    - Management delegated to container
  - Bean-managed transactions
    - Explicit commit/rollback

VERSANT



Getting a Connection

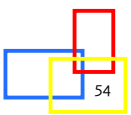
- First get a PersistenceManagerFactory
  - Done during setSessionContext()

```
// Obtain the initial JNDI Naming context
Context ctx = new InitialContext();
// perform JNDI lookup to obtain the connection factory
PersistenceManagerFactory pmf = (PersistenceManagerFactory)
  ctx.lookup("java:comp/env/jdo/VersantPersistenceManagerFactory");
```

- Then get a PersistenceManager

```
// Obtain a connection
PersistenceManager pm = pmf.getPersistenceManager();
```

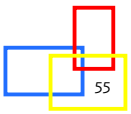
VERSANT



JDO & EJB

- Alternative to using CMP for data store access
  - Simpler & faster to develop
  - Still standard-based & database independent
- Can be used from JSP/Servlets, SessionBeans or BMP EntityBeans
  - Eliminates need to use JDBC directly
  - Simplifies development
- Facilitates an approach to development that compliments common J2EE design patterns
  - Session Façade
  - Value Object
  - Data Access Object

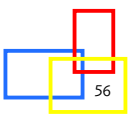
VERSANT



An Example: SessionBean

```
public void createEmployee (
  String name,
  int    age,
  String dept ) : throws RemoteException {

  // Get a PersistenceManager
  PersistenceManager pm = pmf.getPersistenceManager();

  try {
    // Construct the Query
    Extent     depts   = pm.getExtent(Department.class);
    String     filter  = "name = " + dept;
    Query      query   = pm.newQuery(depts, filter);

    Collection results = pm.execute();
```

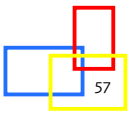
VERSANT



An Example: SessionBean

```
    // Extract the Department from the result set
    Department dept =
      (Department) results.iterator().next();

    // Create a new Employee
    Employee emp = new Employee(name, age);

    // Add Employee to Department
    dept.addEmployee(emp);
  }

  catch (Exception e) {

    throw new RemoteException(e);
  }
}
```

VERSANT

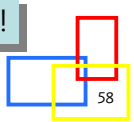


JDO & CMP

- EJB 2.0 addresses many of earlier CMP issues
  - Local interfaces
  - Container-managed relationships
  - EJBQL
- Development still order magnitude harder
  - Home & bean interfaces (local or remote)
  - Abstract bean class & dependent value classes
  - PrimaryKey class
  - Deployment descriptor
- Increasingly difficult to test

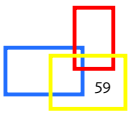
EJB 2.0 requires good tool support to make it useable!

VERSANT



Summary

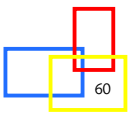
VERSANT



Looking forward...

- JDO accepted by Java Community
  - Needs your support
- JDO incorporated into Java platform
  - Standard mechanism for Java object persistence
  - Complimentary to JDBC
- JDO supported by Application Server vendors
  - Alternative to CMP
  - Disparity between JDO QL & EJB QL
    - Needs to be resolved

VERSANT



Vendor Support

O/R Mapping Tool Vendors

- Forte for Java – SUN
- TopLink - WebGain
- OpenFusion JDO – PrismTech
- Kodo JDO – TechTrader
- LiDO – LIBeLIS
- Rexip JDO – TCCybersoft

Consultants

- Object Identity
- Olgilivie Partners

Database/Middleware Vendors

- **enJin – Versant**
- FastObjects – Poet
- ObjectStore – eXcelon
- Orient ODBMS – Orient Technologies
- GemStone Facets – GemStone Systems

Others

- SAP

VERSANT