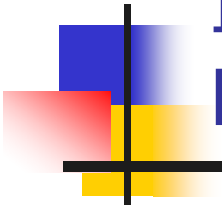


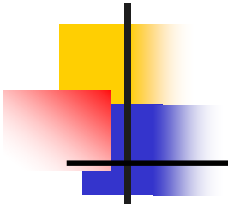
Caging the Tiger

Impact of Java™ 2 Platform, Standard Edition (J2SE™) 5.0 on Java™ Persistence

Donald Smith

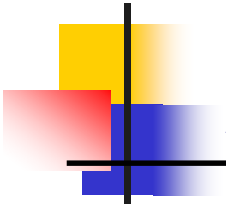
Technology Director, Oracle Corporation

blog: jroller.com/page/dsmith



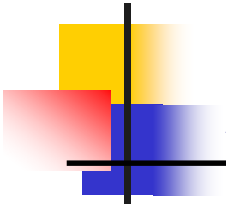
Goal

Discuss the impact J2SE™ 5.0 will have on Java™ Persistence



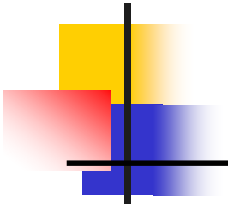
Agenda

- Quick Review of Java™ Persistence
  - Impedance Mismatch
  - Persistence Layers
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary
  - The Architects Perspective
  - The Developers Perspective



Agenda

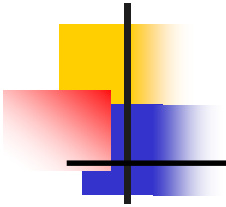
- Quick Review of Java™ Persistence
  - Impedance Mismatch
  - Persistence Layers
- Impact of J2SE™ 5.0 on Java™ Persistence
- Summary



Call To Action

Managing persistence related issues is the most underestimated challenge in enterprise Java today — in terms of complexity, effort and maintenance

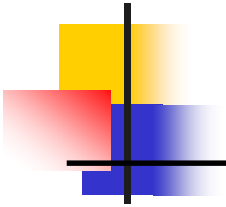
How will J2SE 5.0 help?



Problem

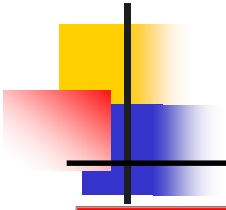
- Building Java applications with relational databases is a VERY challenging and labor intensive problem to solve
  - Fundamentally different technology
  - Different skill sets
  - Different staff/ownership
  - Different modeling and design principles

Differences must be resolved to fulfill business requirements



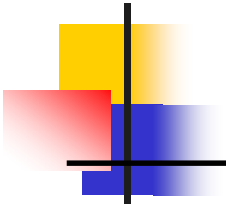
Impedance Mismatch

- Difference in relational and object technology know as "object-relational impedance mismatch"
- Challenging problem
  - Requires relational and object expertise



Impedance Mismatch

	Factor	J2EE	Relational Database
Technical	Logical Data Format	Objects, methods, inheritance	Tables, SQL, stored procedures
	Scale	Hundreds of megs	Gigabytes, terabytes
	Relationship	Memory references	Foreign keys
	Uniqueness	Internal object identity	Primary keys
Business	Key Skills	Java development, object modeling	SQL, Stored Procedures, data management, Data modeling
	Tools	IDE, Source code management, Object Modeler	Schema designer, query manager, performance profilers, database config
Political	Corporate Org. Structure	"Newer technology" often with weak organizational ties to database mgmt	Often mature infrastructure with significant legacy considerations

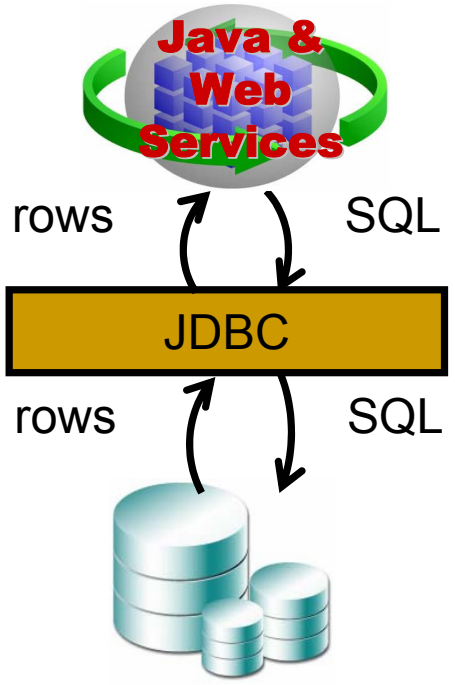


Java Access of Relational Data

- Direct JDBC
  - Direct SQL calls hard coded in Java
  - Use result sets ("rows") directly
- Persistence layer
  - Accessed as objects or components
  - Transparent that the data is stored in RDB
  - Persistence layer in middle tier handles object-relational mapping and infrastructure
  - Required if doing business logic in the middle tier!

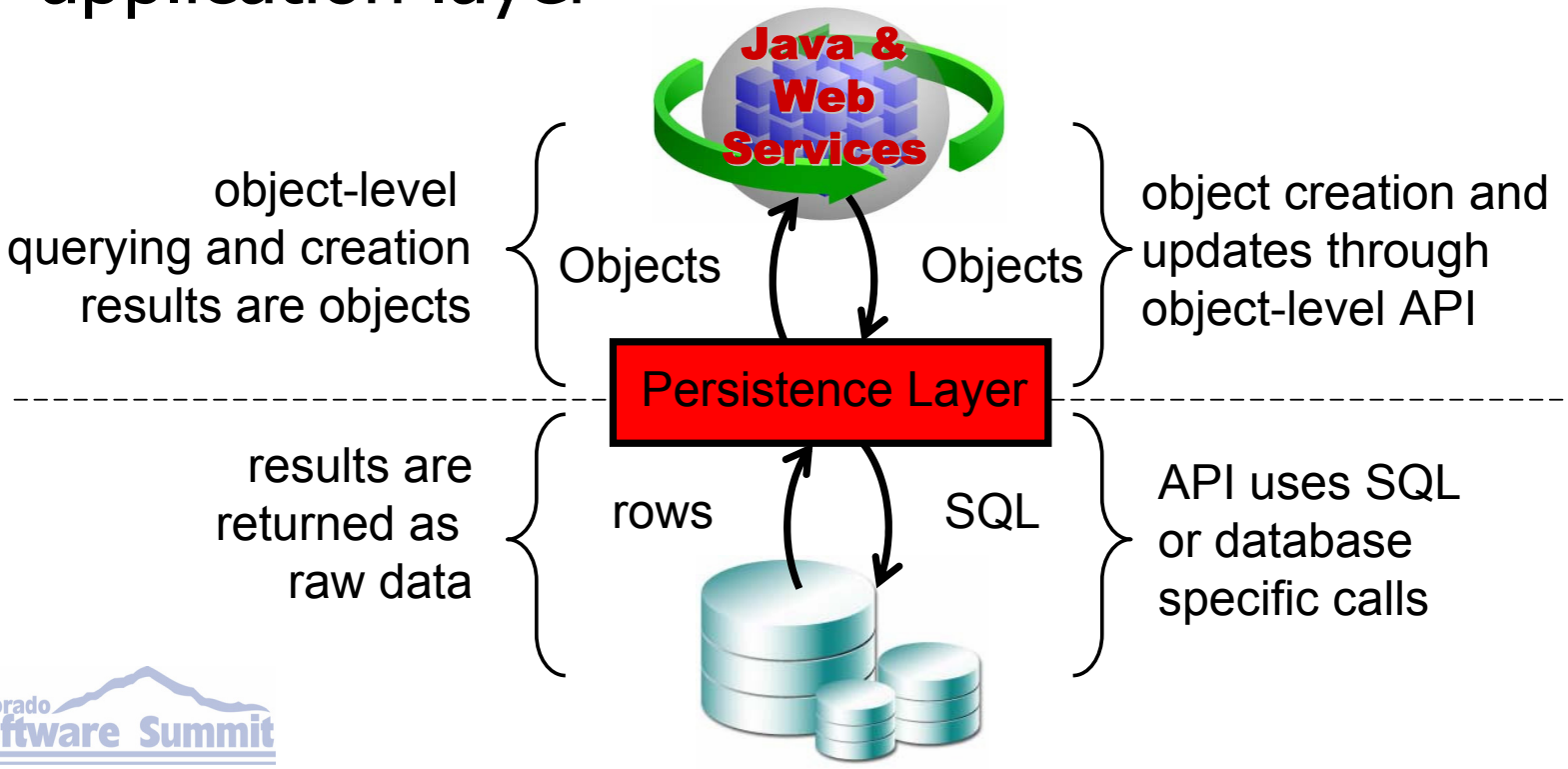
JDBC — Java Database Connectivity

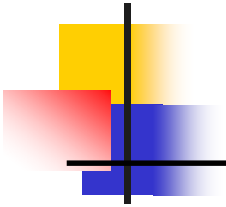
- Java standard for accessing databases
- JDBC is simply the database connection utilities Java developers need to build upon



Persistence Layer

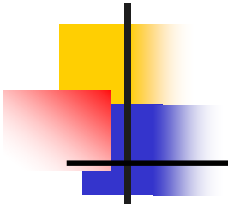
- Abstracts persistence details from the application layer





Persistence Layers

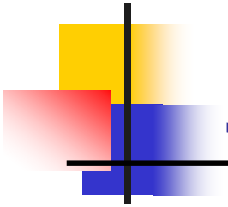
- "Old" J2EE persistence "Entity Beans"
  - BMP
    - Developer must hand code persistence "life cycle" calls generate by J2EE Container
  - CMP
    - More automatic persistence
- "Plain Ol' Java Objects" (POJO) persistence
  - EJB 3.0, Commercial and Open source, JDO



Entity Beans or POJO?

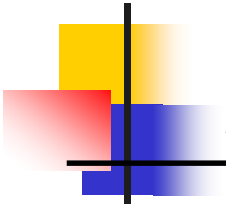
- Hot topic — should you use "Straight JDBC", "Old" Entity Beans or POJO?
- If we use Entity Beans – CMP or BMP, "old" or new?
- If we use POJO — DAO, Commercial?

Not relevant to this discussion! The issue to be discussed is: "*What impact will J2SE 5.0 have on the future of Java Persistence?*"



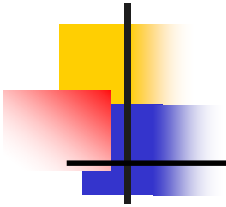
J2SE 5.0 — Tiger!

- Major revision of Java Platform and Language
- 15 Component JSRs and 100 other significant updates
- Developed through Java Community Process
- Let's examine the impact this release will have on how Java developers handle persistence!



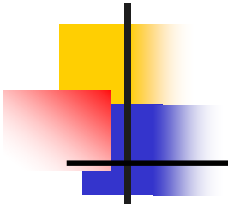
Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary



JSR 114 "RowSets"

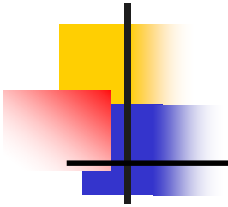
- Defines standard for implementations of the JDBC RowSet model
- Java-ized way of looking at database rows
  - Historically developers use "ResultSets" — logically a HashTable of database rows
- RowSet provides richer API than "ResultSet"
  - "Connected" and "Disconnected" semantics
  - Insert, Modify and Delete API
  - In/Output of RowSet to/from XML
  - RowSets ***are*** JavaBeans



RowSets Example

Create and Execute Query

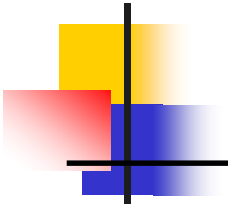
```
CachedRowSet rowset = new OracleCachedRowSet();

// Set URL
rowset.setUrl
   ("jdbc:oracle:thin:@152.23.45.32:1521:orcl");
// Set Username and Password
rowset.setUsername("HR");
rowset.setPassword("HR");
// Set SQL Query
rowset.setCommand(
   "SELECT employee_id, first_name FROM Employee”);
// Execute
rowset.execute();
```



RowSets Example

Process Results

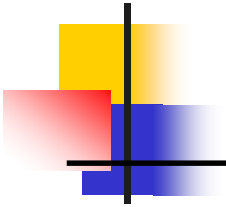
```
while (rowset.next())
{
   System.out.println(
      "empno: " +rowset.getInt (1));
   System.out.println(
      "ename: " +rowset.getString (2));
}

// Produces
 empno: 7
 ename: Donald
 empno: 36
 ename: Christian
 …
```



RowSets Example

Update Row

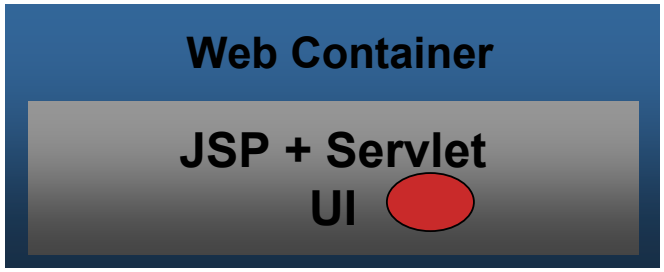
```
// Make the RowSet writable
rowset.setReadOnly(false);
// Point the rowset cursor (if necessary)
rowset.absolute(new Integer(1));
// Update the values of the selected row
rowset.updateString(2, emp.getFirstName());
.....
// Update the row in the RowSet
rowset.updateRow();
// Set the rowset for read-only
rowset.setReadOnly(true);
// Commit
rowset.acceptChanges();
```



Different Kinds of RowSets

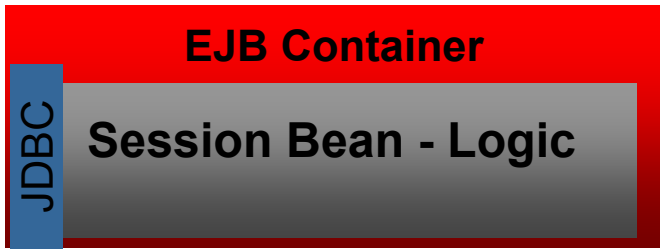
- JDBCRowSet
  - Connected and not Serializable
  - Essentially a wrapper for "ResultSet" that makes it a JavaBean, scrollable and updatable
- CachedRowSet
  - Disconnected and Serializable!
  - Server can serialize to clients, which can update and serialize back to server for commit to database
- WebRowSet
  - CachedRowSet that can be read and written to/from XML

CachedRowSet Serialization Example



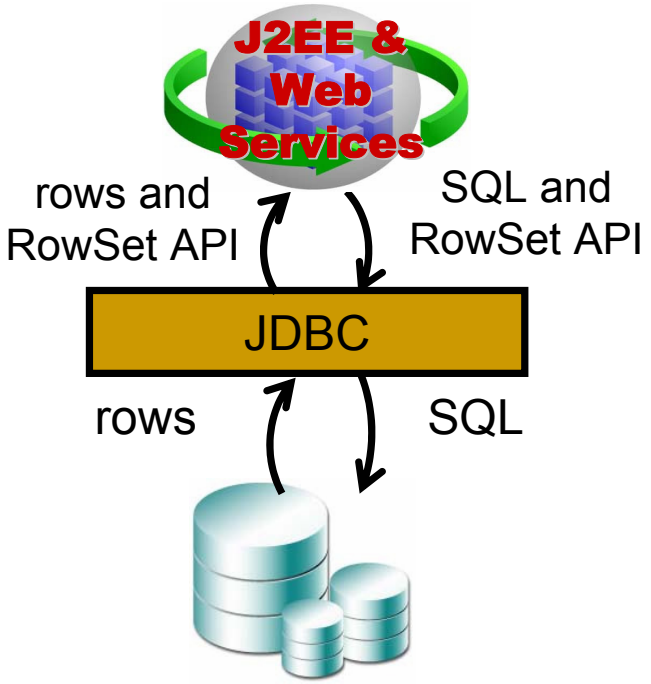
Flow of events discussed in session

remote



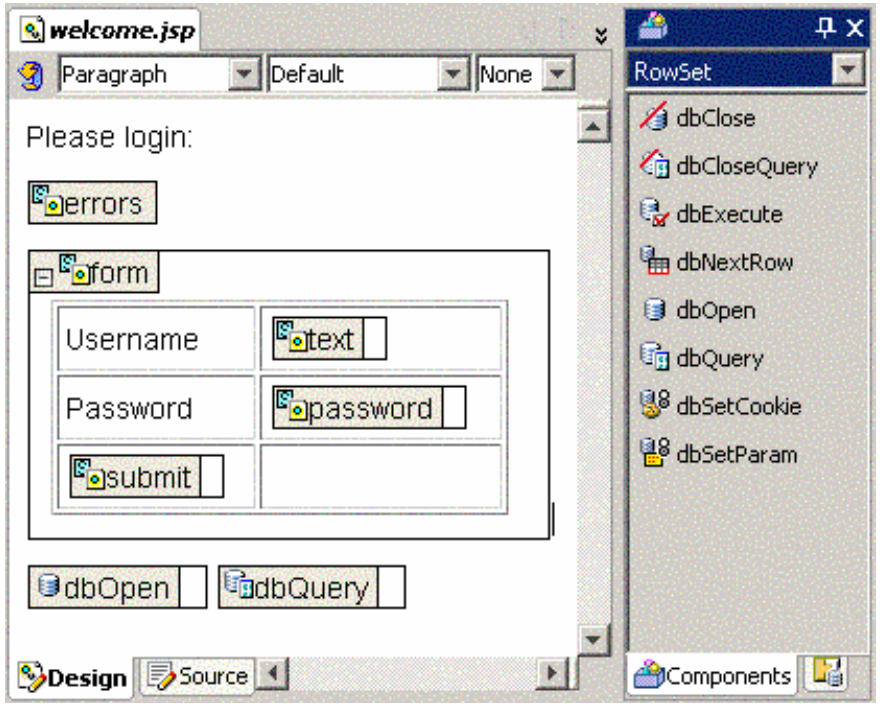
JSR 114 "RowSets"

- Improvement of "Straight JDBC" programming
  - Not a "Persistence Layer"
  - OK with:
    - "Window on data" applications
    - Business logic entrenched on database
    - J2EE nothing more than GUI tool



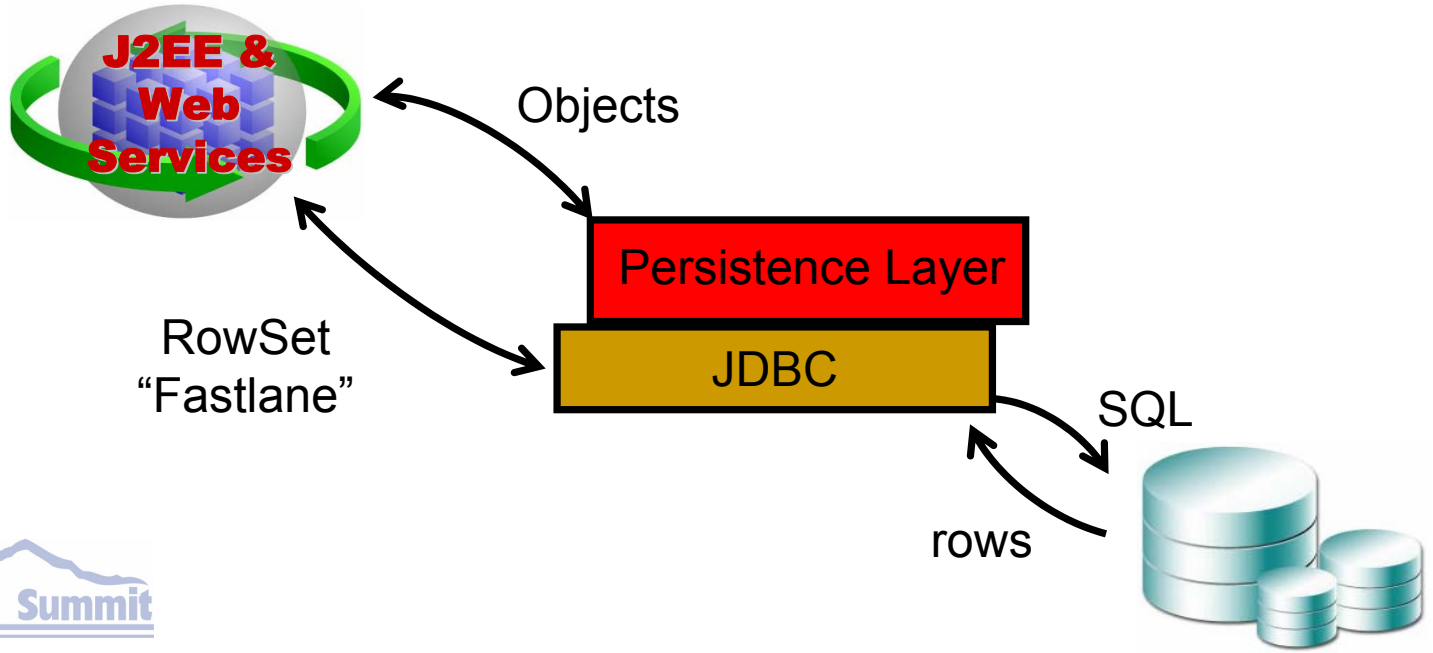
JSR 114 "RowSets"

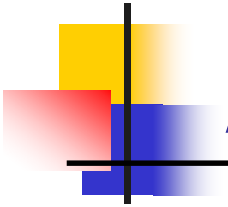
- Great for IDE vendors and for frameworks
  - JavaBean approach of RowSet makes it easier for GUI and Wizard based tools to expose DB
  - Will appeal to "VB or Microsoft style" developers



JSR 114 "RowSets"

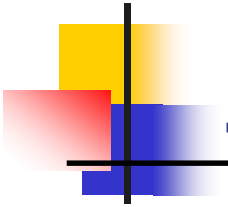
- Persistence Layer prefer fast native API, not likely to use RowSet internally
  - May expose RowSets as "fast lane pattern"





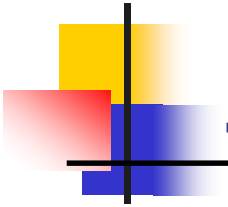
Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary



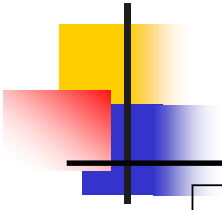
JSR 175 "Annotations"

- Common metadata infrastructure
  - Tools use metadata information to generate
    - Additional source code
    - Debugging information
  - Anything can be annotated!
    - Classes, Interfaces, Fields and Methods, Packages, Local Variables!
- Formalizes using metadata in Javadoc tags
- Hit with Aspect Oriented Programming



JSR 175 "Annotations"

- Permits "Descriptor-less Deployment"
  - O-R Mapping Information kept with source
- Reduces versioning conflicts
  - Metadata changes change "code" instead of descriptor

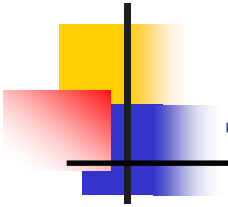


JSR 175 "Annotations"

```
// Employee.java

public class Employee {
   public int id;
   public String name;
   …
}
```

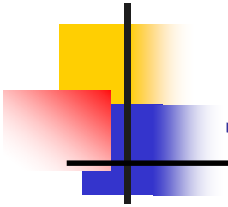
Before…

```
// OR-Mapping.xml
<mappings>
   <database-mapping>
      <attribute-name>id</attribute-name>
      <field-name>EMPLOYEE.ID</field-name>
      <read-only>false</read-only>
      <type>DirectToFieldMapping</type>
   </database-mapping>
…
</mappings>
```



JSR 175 "Annotations"

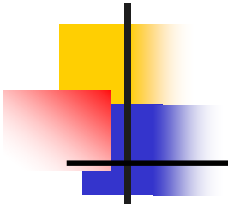
```
// Employee.java with OR Annotations

@Entity(access=FIELD)
@Table("EMPLOYEE")
public class Employee {
    @Id
    @Column(name="ID")
    protected int id;

    @Basic
    @Column(name="NAME")
    protected String name;
    …
}
```



JSR 175 "Annotations"

- OR is more sophisticated than first appears
  - Potential for annotations to take over!

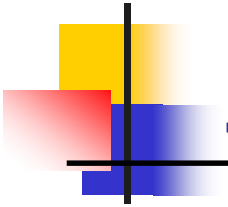
```
public class Customer {
    @ORMapping (
        reference-class = com.domain.Order.class,
        type = "OneToMany",
        read-only = false,
        private-owned = false,
        indirection-policy = "Transparent",
        container-class = ArrayList.class,
        source-key-field = "CUSTOMER.ID",
        target-key-field = "ORDER.CUST_ID",
        order-by-target-field = "order_date",
        batch-read = false,
        …)
    public Collection orders;
    …
```



Minimizing Annotations

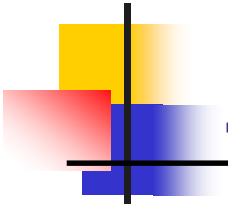
- EJB 3.0 strategy for minimizing annotation work is by using "defaults"
- Could be as simple as:

```
// Employee.java with OR Annotations

@Entity(access=FIELD)
public class Employee {
    @Id
    protected int id;

    protected String name;
    …
}
```



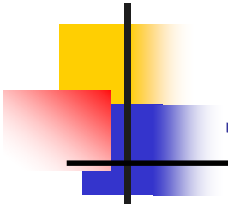
JSR 175 "Annotations"

- Complicates dependencies for tools
  - Mapping GUIs interacting with source files
  - Mapping Tools and IDE
    - Increased value of coupling
- What about dynamic annotations?
  - Changing annotations requires recompile
  - Target table for class dynamic based on dev versus testing, *etc.*



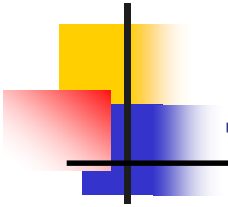
JSR 175 "Annotations"

- Mapping Tools responsible for more than mapping, where does this metadata go?
  - Queries
  - Object retrieval optimizing
    - (eliminate N+1 reads)
  - Persistent Object Lifecycle event handling
  - Insert/Update/Delete Analyzer and optimizer
  - Stored Procedure handling
  - Cache Strategies



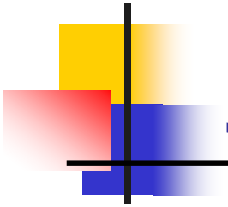
JSR 175 "Annotations"

- "Descriptorless Deployment" an exaggeration…
- Package annotation — not "clean"
  - Likely not viable
- Need Session descriptor
  - Database platform and version
  - Data Source Information (Connection Pool)
  - Transaction Controller
  - Logging and Profiling
- Logging and profiling *via* JMX an option



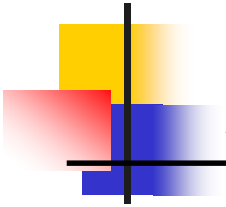
JSR 175 "Annotations"

- Should facilitate team development
  - Team development practices at the source code level well defined
  - Team development practices for OR-Mapping.xml files not well defined
    - 500 classes, one mapping file…



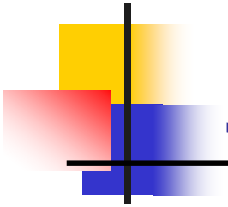
JSR 175 "Annotations"

- Decoupling of metadata and source code
  - Choice should be provided
- O-R Metadata tags require standards
  - Will be addressed by EJB 3.0
- Tags may become unwieldy
  - Persistence challenge underestimated
  - Each attribute requires dozens of tags
  - Defaults will help



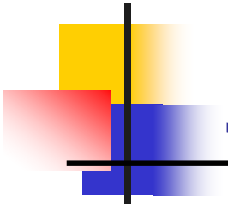
Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary



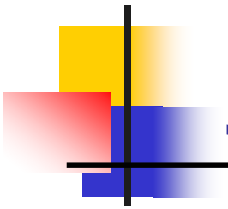
JSR 14 "Generics"

- Extension of Java to allowing parameterized types
- Typing problems caught at compile time
  - Minimizes runtime "ClassCastException"
- Simpler `for` loops and other controls



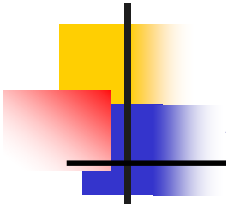
JSR 14 "Generics"

```
// Before Generics…
ReadAllQuery raq = new ReadAllQuery(Emp.class);
Collection c = session.executeQuery(raq);
for(Iterator i = c.iterator(); i.hasNext();) {
   Emp e = (Emp)i.next();
   e.giveRaise();
}

// Becomes…
ReadAllQuery<Emp> raq = new ReadAllQuery<Emp>();
Collection<Emp> c = session.execute(raq);
for(Emp e : c) e.giveRaise();
```



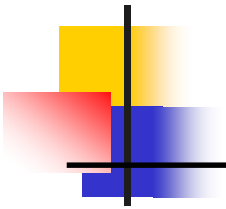
JSR 14 "Generics"

- Persistence layers must support returning appropriately typed Collections
  - Makes development easier
  - Type errors caught at compile time
- Fairly trivial to implement



Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary



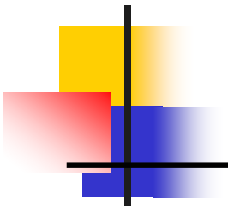
Enumerations

- "Enumerations" (part of JSR 201)
  - Self-typed constants with no public constructor

```
enum Severity {low, medium, high}
```

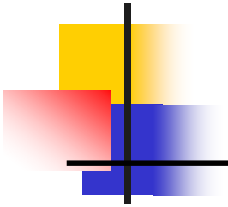
  - Enums can have attributes

```
enum Coin
 {penny(1), nickel(5), dime(10)}
```



Enumerations

- Persistence layers will need to support
  - Mapping Enumerations to tables
  - The ability to map attributes which are Enumeration types
  - Looks dead simple at first, but…
  - Let's look at an example…

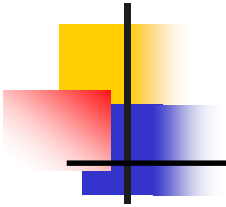


Enumerations

```
public enum Severity {low, medium, high}
public class SupportCase {
    int id;
    Severity sev;
    String customerName;
    …
}
```

Could map as simply as…

SUPPORT_CASE		
ID	SEV	C_NAME
27	medium	“ABC Inc.”
32	high	“XYZ Corp.”



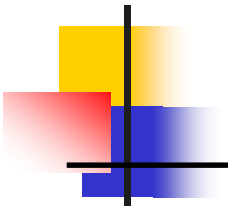
Enumerations

```
public enum Severity {low, medium, high}
public class SupportCase {
    int id;
    Severity sev;
    String customerName;
    …
}
```

Or require more consideration!

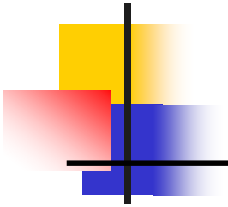
SUPPORT_CASE		
ID	SEV_CODE	C_NAME
27	102	“ABC Inc.”
32	103	“XYZ Corp.”

SEVERITY_CODES	
CODE	DECRIPTION
101	Fix within 3 days
102	Fix within day
103	Fix within hour



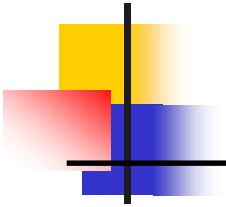
Enumerations

- Persistence layers need to consider
  - Map Enumeration value to column
  - Code table which may or may not contain literal Enumeration value
    - Requires translation in mapping
  - Validation between database and `enum`
    - Startup validates enum types with values on the database



Enumerations

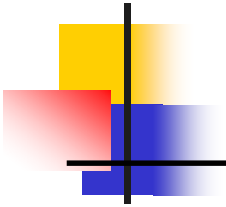
- Developers need to consider
  - Enumerations not always best choice
    - Let's not overdo it, huh?
  - If app needed "DESCRIPTION" from database, map SEVERITY_CODE as a class, not an `enum`!
  - If codes are non-static, make it a class!



Enumerations

- Enumerations can have attributes which ***are*** mutable
  - Will this be considered bad style?
  - Regardless, consider the mapping implications…

```
public enum Severity {low(72), medium(24), high(1);
   Severity(int hrs) {this.resolveTime = hrs;}
   private int resolveTime;
   public int getResolveTime(){return resolveTime;}
   public void setResolveTime(int rt) {
      resolveTime = rt;
   }
}
```



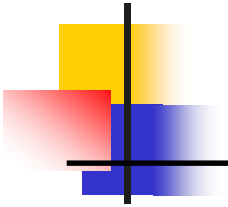
Enumerations

```
enum Severity {low(72), medium(24), high(1)
```

SUPPORT_CASE		
ID	SEV	C_NAME
27	medium	“ABC Inc.”
32	high	“XYZ Corp.”

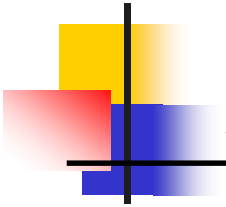
RESOLVE_TIME	
SEV	TIME_H
medium	24
high	1

- How will persistence layers do change detection of enumeration attributes?
- How will changes be reconciled with legacy data?



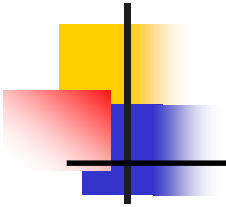
Enumerations

- Likely to be overdone initially until a proper "style" emerges
- Not so trivial when String value not used in database
- Not so trivial with mutable values!
  - Let's hope this is considered bad style



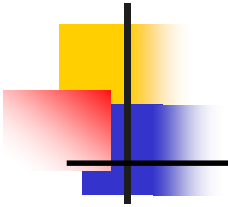
Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary



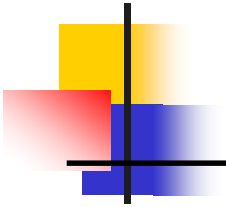
Monitoring and Management

- Java applications better monitored and managed
  - JSR 003 "JMX"
    - Management architecture of Java resources
  - JSR 160 "JMX Remote API"
    - Discovery of JMX Agents
  - JSR 163 "Platform Profiling Architecture"
    - Time and memory profiling of a JVM
  - JSR 174 "Monitoring and Management"
    - Monitoring and managing aspects of a JVM



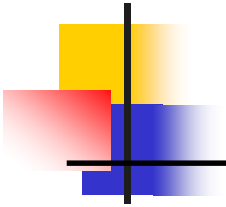
Monitoring and Management

- Common complaint — Administrators require:
  - More performance data
  - More runtime control
- JMX not new!
  - Inclusion in JDK 5.0 increases user demand
- JDK 5.0 offers more internal monitoring and management
  - Native API or through JMX



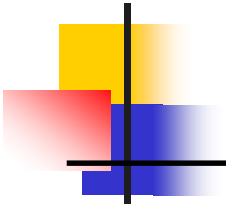
Monitoring and Management

- New hooks to tune, trace and debug
  - Memory usage
  - Garbage Collection overhead
  - Thread usage
  - Object info (count)
  - Class loading/unloading
- Vendors will associate stats with persistence functionality…



Monitoring and Management

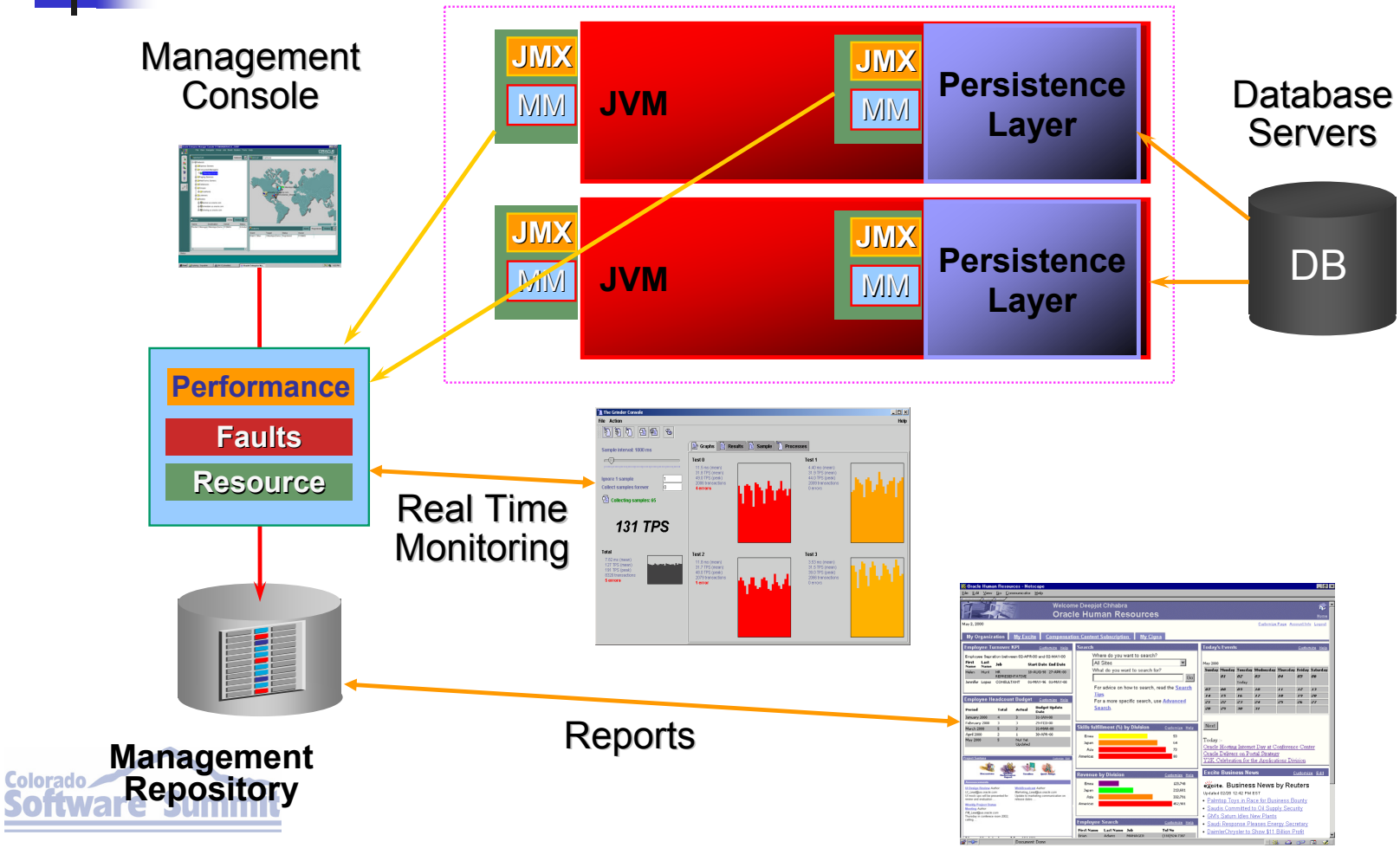
```
ReadAllQuery raq = new ReadAllQuery(Employee.class);
raq.profileResults();
session.logProfile();
session.executeQuery(raq);

// Prints
Profile results
 - 3476 business objects read in 0.87 seconds
 - 3526 new objects in heap
 - Memory usage: 524727bytes
 - 3 Classes were loaded
 - The GC ran during this query
   - It interrupted for 0.14 seconds
   - It reclaimed 31889bytes
```

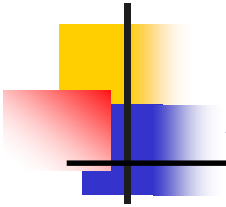


Monitoring and Management

- Administrators analyze and manage running systems
- Vendors to offer API pluggable into "Dashboards"
  - Cache monitoring
    - Force GC as cache exceeds limits
  - Data source monitoring
    - Restart or failover connections on failures
  - Session Control (Persistence Layer Parameters)
    - Profilers, Logging, Exception Handlers, External Transaction Controllers

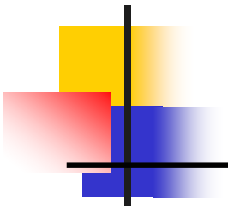
Monitoring and Management





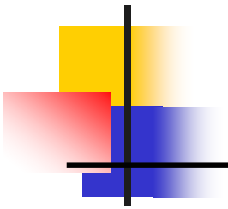
Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
  - JSR 114 "RowSets"
  - JSR 175 "Annotations"
  - JSR 014 "Generics"
  - Enumerations
  - Monitoring and Management JSR 003, 160, 163, 174
  - Memory, Threading and Concurrency JSR 133, 166
- Summary



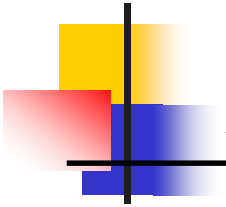
Memory, Threading and Concurrency

- Enhancements to memory, threading and concurrency options
  - JSR 133 "Memory Model and Thread Specification"
    - Describes semantics of threads, locks and data races
  - JSR 166 "Concurrency Utilities"
    - Set of utilities commonly needed in concurrent programs



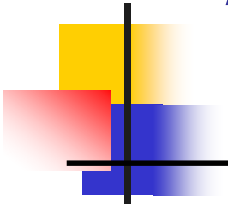
Memory, Threading and Concurrency

- Offers vendors enhanced API regarding
  - New packages java.util.concurrent, java.util.concurrent.locks and java.util.concurrent.atomic
  - Asynchronous Exceptions
  - Thread Interrupts, sleep/wait/join semantics on Thread
  - Enhanced synchronizers
- Result? Faster, safer persistence!



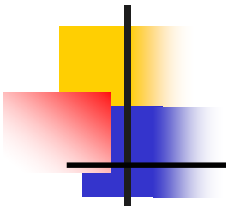
Agenda

- Quick Review of Java™ Persistence
- Impact of J2SE™ 5.0 on Java™ Persistence
- Summary
  - The Architects Perspective
  - The Developers Perspective



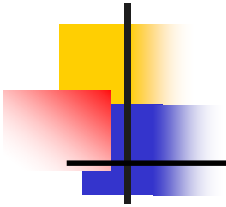
Architects Perspective Summary

- RowSet does not obviate Persistence Layers
  - More architectural support for "window on data" style applications
- Annotations reduce dependency on descriptor files
- Need to be cautious with `enum`
- Monitoring and management to offer better consoles
- Faster and safer persistence!



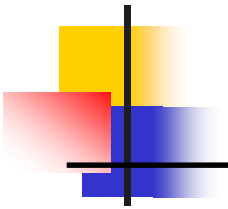
Developers Perspective Summary

- IDEs to provide richer development tools
  - JavaBean support for GUI driven development
  - Able to develop persistence "VB or Microsoft style"
- Enhanced type checking with Generics
- Annotations facilitate:
  - Team development
  - Source code management
- Easier to trace, debug and performance tune persistence



Summary

- Tiger helps Architects architect persistence!
- Tiger helps Developers develop persistence!



Q&A

- Donald Smith
- Oracle Corporation