JBoss Certified Technology Partner

Israel JBoss User Group

Session 02 / 5.6.2006

Advanced Hibernate

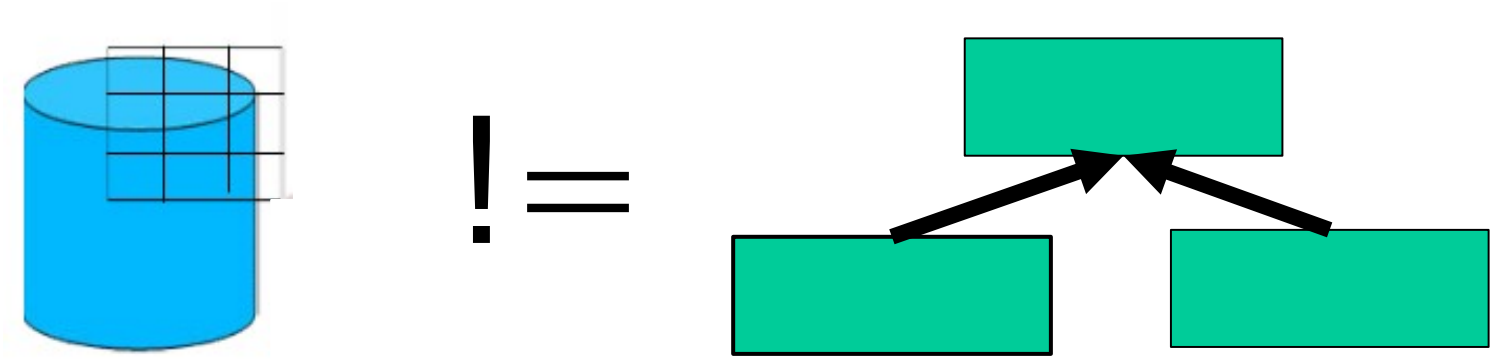
By : Yanai Franchi, Senior Software Engineer "Tikal"

Agenda

- Overview and Recap
- Advanced Mappings
- Transactional Processing
- Querying and Fetching
- Tuning

Overview and Recap

The Core Problem

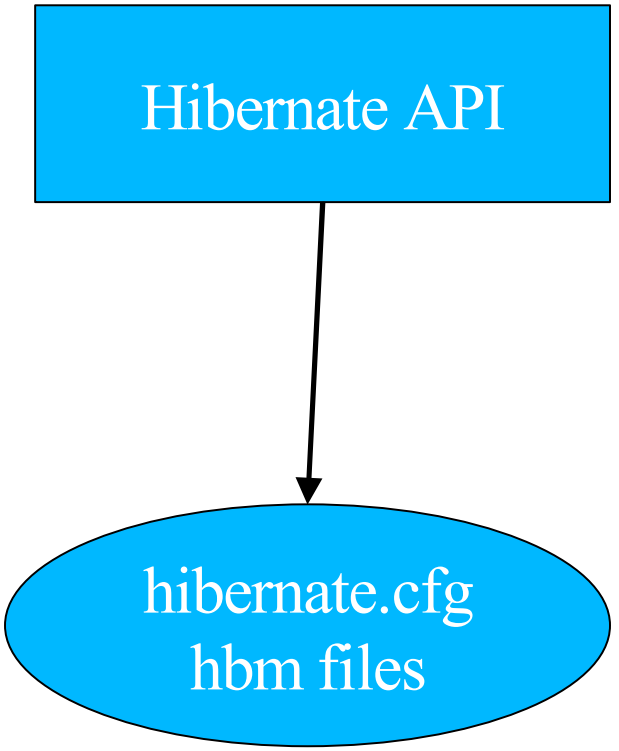


- The *Object-Relational impedance mismatch*
  - How do we map tables and column to objects
- Getting persistence right is essential
  - Failure in persistency strategy will make any non-trivial application fail

Hibernate in a Nutshell

- A solid product, widely used
- Use a mapping layer to map between objects and tables
  - XML
  - Annotations
- Dual-Layer Caching Architecture (HDLCA)
- Powerful, high performance queries
- Support for *detached* objects (no DTOs)
- …and many more cool features
- Provide a great programming model
  - Non invasive
  - Transparent and transitive

Three Ways to Achieve Persistency…



Hibernate Mapping

```
package com.tikal.sample;
public class User {
   private Long id;
   private String name;

   public User(){}
   public User(name){
       this.name=name;
    }
   public void setName(String name) {
       this.name = name;
   }
   public String getName() {
       return name;
   }
}
```

Hibernate Mapping – Cont'

User.hbm

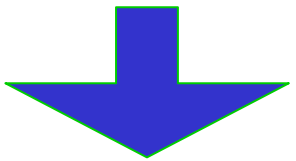
```
<hibernate-mapping package="com.tikal.sample">
   <class name="User" table="USERS">
   <id>...</id>
   <property name="name" />
</hibernate-mapping>
```

Hibernate.cfg

```
<hibernate-configuration>
   <session-factory>
        <property name="dialect">
            org.hibernate.dialect.HSQLDialect</property>
        <property name="connection.driver_class">
            org.hsqldb.jdbcDriver
        </property>
        <property name="connection.url">
             jdbc:hsqldb:mem:aname
        </property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>

        <mapping resource="model/User.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

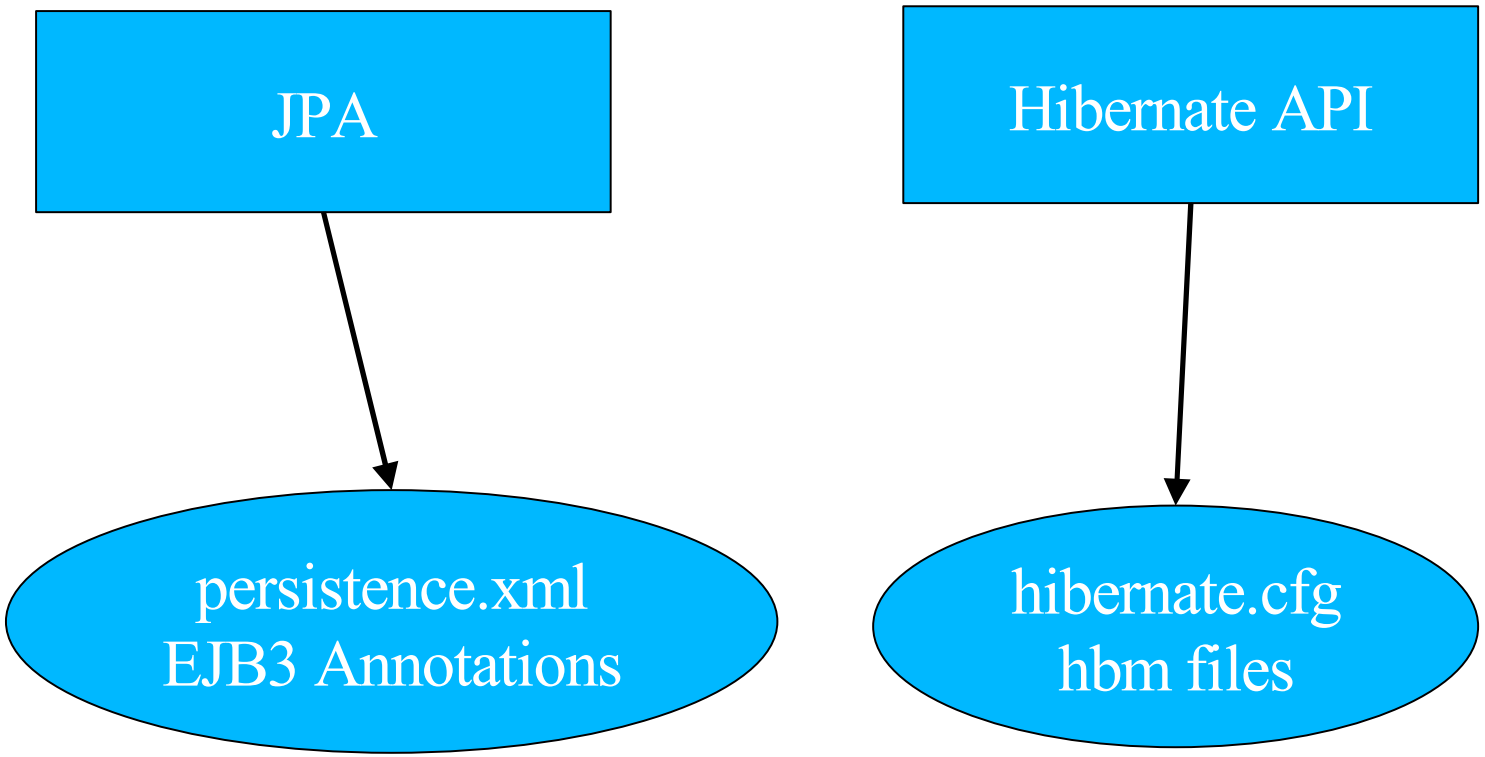
Hibernate API Example

```
SessionFactory sf =
   new Configuration().buildSessionFactory();

Session s = sf.openSession();
Transaction t = s.beginTransaction();

u = new User("Yossi");
s.save(u);

t.commit();
s.close();

sf.close();
```



```
insert into users (id, name) values (1,'Yossi')
```

Three Ways to Achieve Persistency…



What is JPA ?

- Java Persistence API is now the standard API for ORM for the Java EE platform.
  - » Simplifies the development of JEE and JSE applications using data persistence.
  - » Get the entire Java community behind a single, standard persistence API
- Originated as an improvement of EJB-CMP and became a separate persistence spec for POJOs.
- Usable both within JSE5 and JEE5
- We can use Hibernate (or other ORM implementation) as the persistence provider.

JPA Mapping

```
import javax.persistence.*;
package com.tikal.sample;

@Entity @Table(name="USERS")
public class User{

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String name;
   User(){}
   public User(name){
       this.name=name;
    }
   public void setName(String name)  {
       this.name = name;
   }
   public String getName()  {
       return name;
   }
}
```

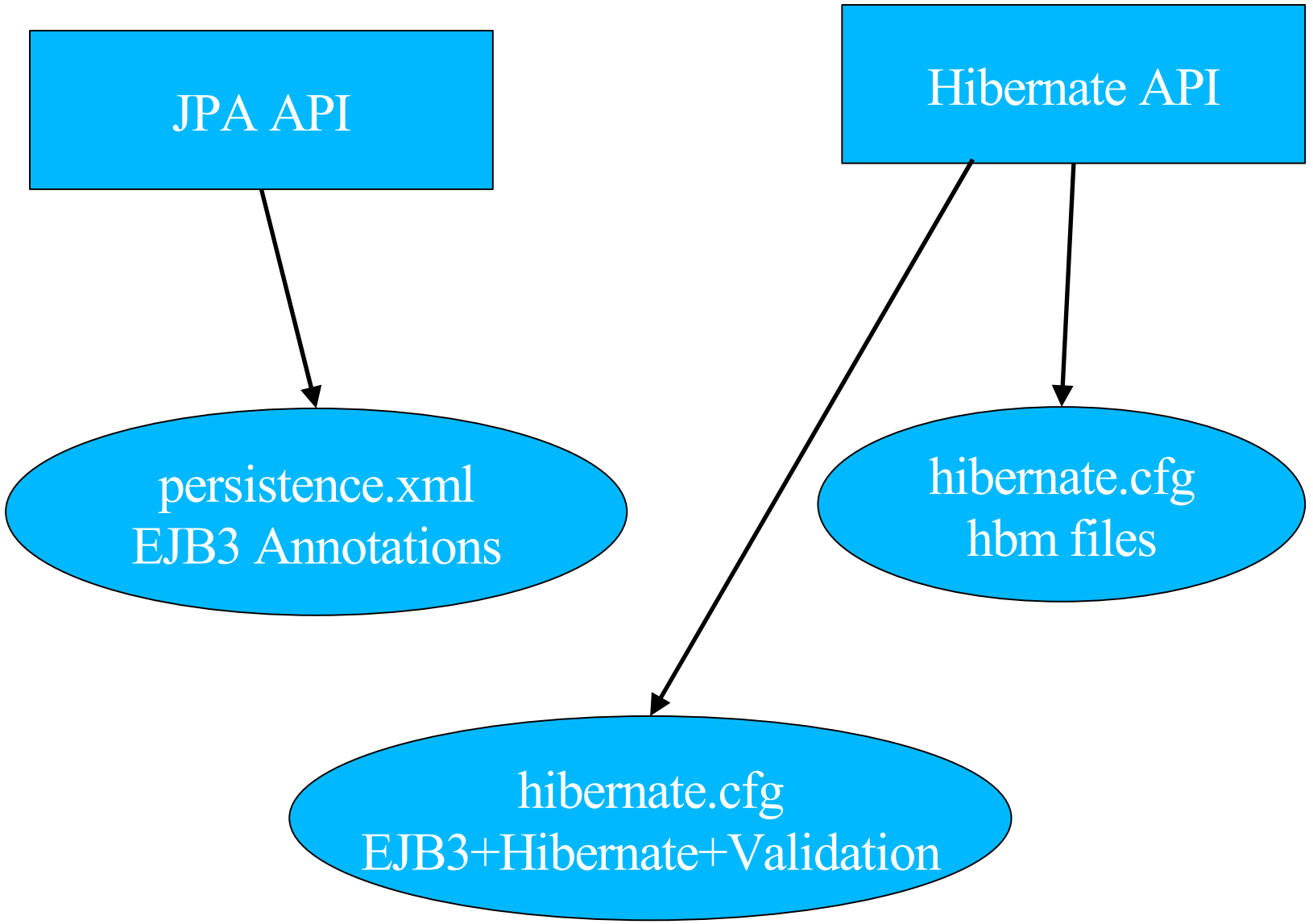
JPA Configuration – persistence.xml

```
<persistence>

  <persistence-unit name="manager1“
        transaction-type=“RESOURCE_LOCAL”>
  <class>com.tikal.sample.User</class>

  <properties>
   <property name="hibernate.dialect"
        value="org.hibernate.dialect.HSQLDialect" />
   <property name="hibernate.connection.driver_class"
        value="org.hsqldb.jdbcDriver" />
   <property name="hibernate.connection.url"
        value="jdbc:hsqldb:mem:aname" />
   <property name="hibernate.connection.username" value="sa" />
   <property name="hibernate.connection.password" value="" />
   <property name="hibernate.show_sql" value="true" />
   <property name="hibernate.hbm2ddl.auto" value="create-drop" />
   <property name="hibernate.cache.provider_class"
        value="org.hibernate.cache.NoCacheProvider" />
  </properties>
 </persistence-unit>
</persistence>
```

JPA in Action

```
EntityManagerFactory emf = Persistence
          .createEntityManagerFactory("manager1");
EntityManager em = emf.createEntityManager();

User u = new User(“Yossi”);

em.persist(u);
User u2 = (User)em.find(User.class,u.getId());

List<User> users = em.createQuery
      ("from User as u where u.name = :name")
       .setParameter(“name",”Yossi”)
       .getResultList();

em.close();
emf.close();
```

Three Ways to Achieve Persistency…



Hibernate Annotations

```
import javax.persistence.*;
@Entity @Table(name="USERS")
@org.hibernate.annotations.Entity(mutable=false)
public class User{

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String name;
   User(){}
   public User(name){
       this.name=name;
    }
   public void setName(String name) {
       this.name = name;
   }
@org.hibernate.validator.Length(max=8)
@org.hibernate.validator.NotNull
   public String getName() {
       return name;
   }
}
```

Hibernate Annotations Cont.

```
<hibernate-configuration>
   <session-factory>
      …
      <mapping class="com.tikal.sample.User"/>
   </session-factory>
</hibernate-configuration>
```

```
SessionFactory sf =
   new AnnotationConfiguration().buildSessionFactory();

Session s = sf.openSession();
Transaction t = s.beginTransaction();

u = new User("Yossi");
s.save(u); //hibernate api
s.persist(u); // Java Persistent api

t.commit();
s.close();

sf.close();
```

Dynamic Models

- Dynamic Models
  - Map of Maps
  - Dom4j trees
  - Implement your own Tuplizer

- Change The default entity model

```
hibernate.default_entity_mode=dynamic-map
```

```
pojoSession.getSession(EntityMode.MAP);
```

Dynamic-Model Example

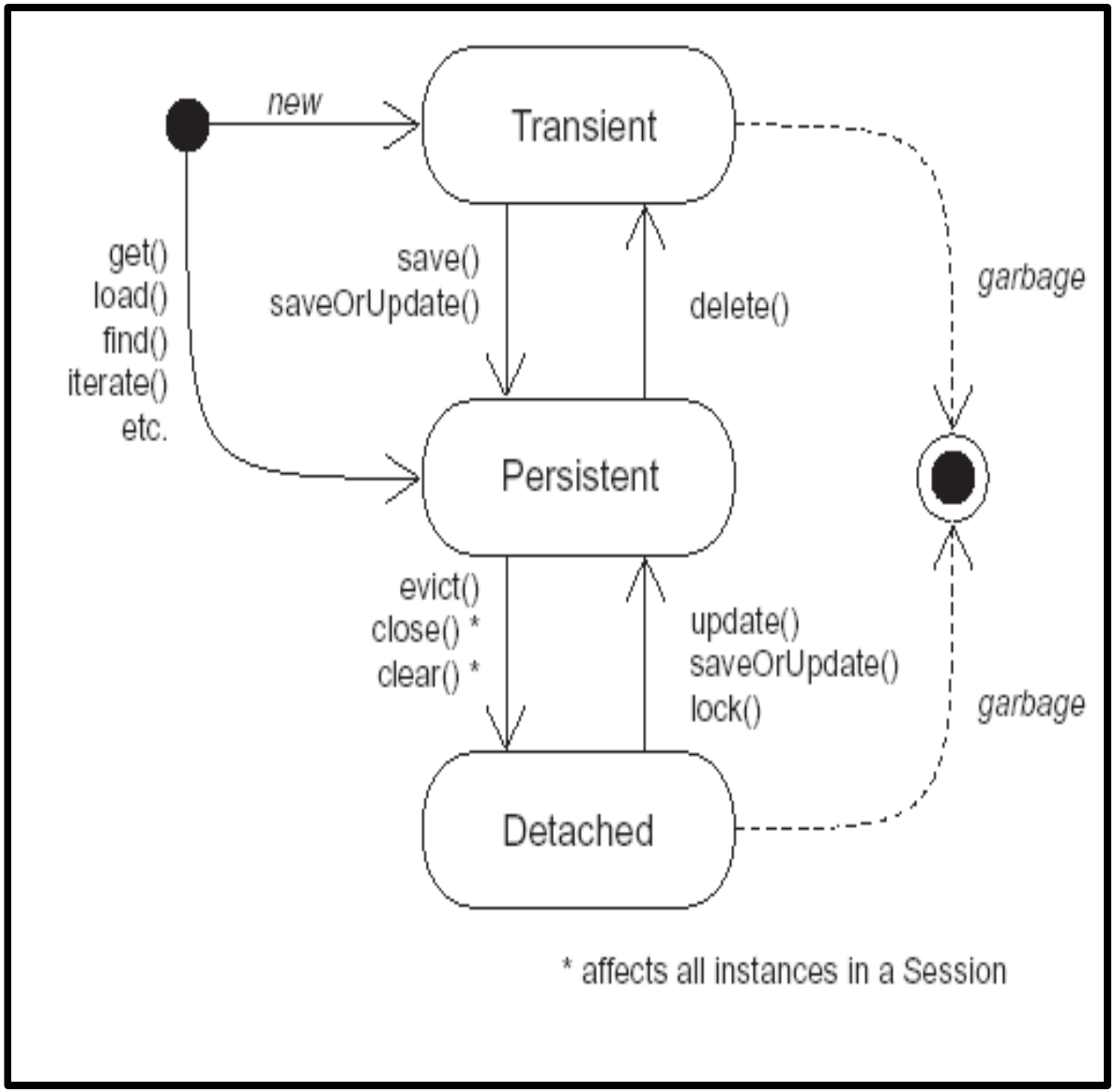

```
<hibernate-mapping>
 …
 <class entity-name="Customer">
   <id name="id" type="long" column="ID">
     <generator class="sequence"/>
   </id>
   <property name="firstName" column="FIRST_NAME"/>
   <property name="lastName" column="LAST_NAME"/>
   <many-to-one name="relatedOrganization"
       cascade="save-update" class="Organization"/>
 </class>
</hibernate-mapping>
```

Dynamic Model Example Cont.

```
Session s = openSession();
Transaction tx = s.beginTransaction();

Map myOrganzation = new HashMap();
myOrganzation.put("name", "Foo inc");

Map myCustomer = new HashMap();
myCustomer.put("firstName", "David");
myCustomer.put("lastName", "Mor");
myCustomer.put("relatedOrganization", myOrganzation);

s.save("Customer", myCustomer);

tx.commit();
s.close();
```

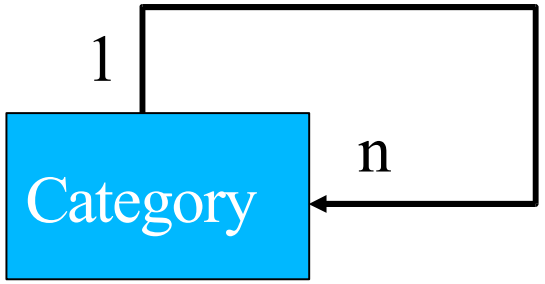
Transactional Processing

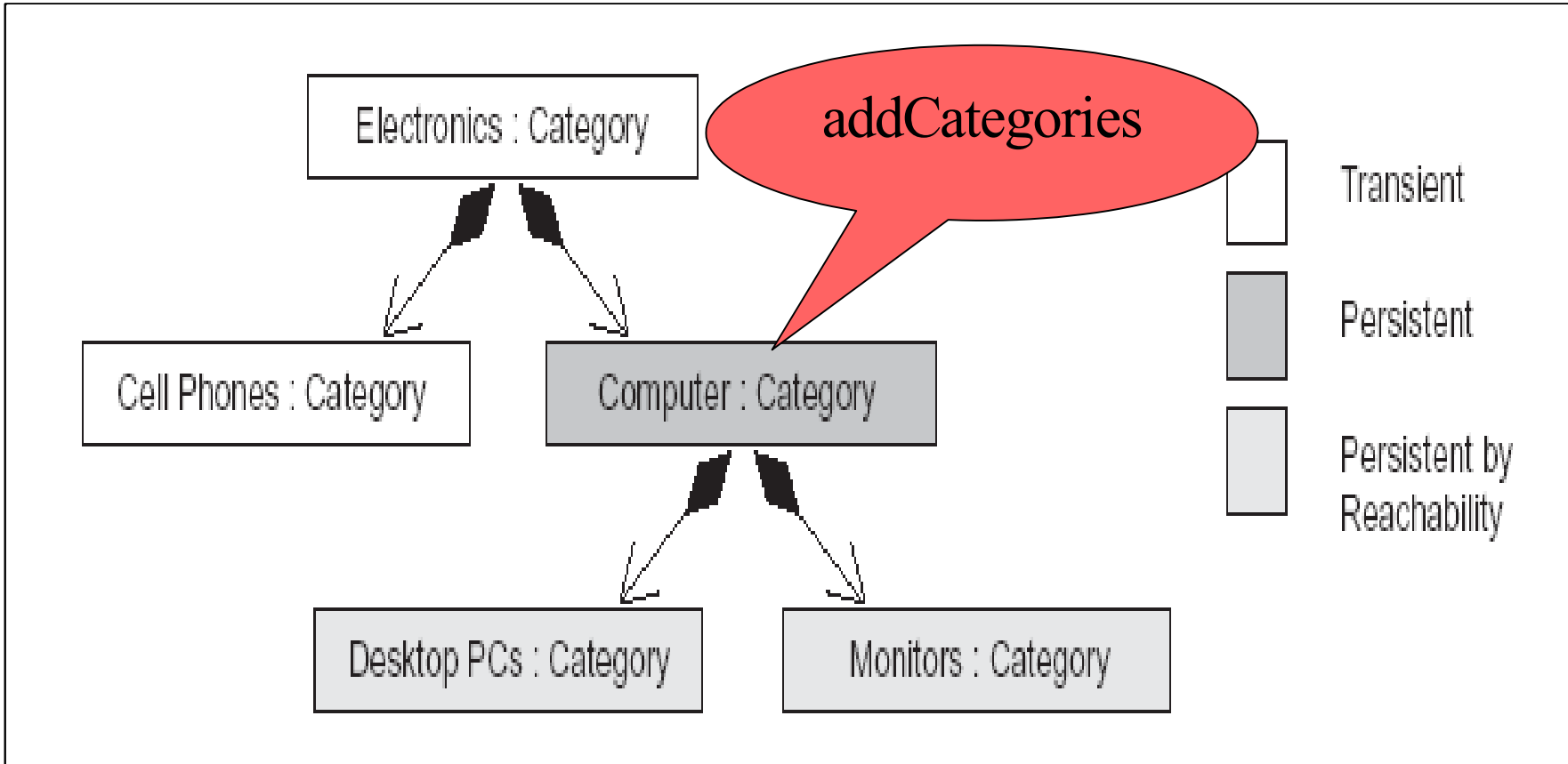
The Persistence Lifecycle



Persistence by Reachability

- An Object becomes persistence if it is referenced from a persistent instance





Transitivity Persistence Example

```
// Creating Transient categories
Category monitors = new Category("monitors");
Category desktopPCs = new Category("desktopPCs");

Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
Category computers =
   (Category)session.get(Category.class,computersId);
computers.addCategory(monitors); //-> Persistent
computers.addCategory(desktopPCs); //-> Persistent
tx.commit();
session.close();
```

Identity Problem

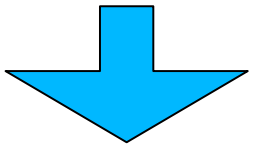
- Java objects define two different notions of sameness:
  - » Object identity (a == b)
  - » equality by value (equals() method)
- The identity of a database row is expressed as the ***primary key*** value.

Which Scope does Hibernate Implement?

```
Session session1 = sessions.openSession();
Transaction tx1 = session1.beginTransaction();
Object a = session1.load(User.class, new Long(1234));
Object b = session1.load(User.class, new Long(1234));
System.out.println(a==b);//true
System.out.println(a.equals(b)); // true
tx1.commit();
session1.close();

Session session2 = sessions.openSession();
Transaction tx2 = session2.beginTransaction();
Object b2 = session2.load(User.class, new Long(1234));
System.out.println(a==b2);//false
System.out.println(a.equals(b2)); //false
tx2.commit();
session2.close();
```

Primary Key

```
public class User {
    ...
    public boolean equals(Object other) {
        if (this==other) return true;
        if (id==null) return false;
        if ( !(other instanceof User) ) return false;
                final User that = (User) other;
                return this.id.equals( that.getId() );
        }

    public int hashCode() {
        return (id==null) ?
                System.identityHashCode(this) :
                id.hashCode();
    }
}
```



Might break `java.util.Set` contract!

Business Key

```
public class User {
    ...
    public boolean equals(Object other) {
        if (this==other) return true;
        if ( !(other instanceof User) ) return false;
        final User that = (User) other;
        return this.name.equals( that.getName() );
    }
    public int hashCode() {
        return name.hashCode();
    }
}
```

Identify the business key

Querying and Fetching

The n+1 Selects Problem

Suppose we wanted initialize items collection of each category:

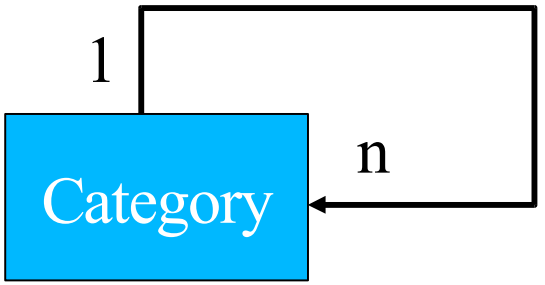
```
List<Category> categories =
   session.createQuery("from Category").list();
for (Category category : categories) {
   for (Item item :category.getItems()) {
        if(item.getPrice >123) //doSomething
   }
}
```

Eager Fetching

Keep everything lazy and use runtime fetching:

```
List<Category> categoris =
   new HashSet(session.createQuery(
        "from Category c fetch join c.items").list());

for (Category category : categories) {
   for (Item item :category.getItems()) {
        if(item.getPrice >123) //doSomething
   }
}
```

Caching Scope



Suppose we want to display the category tree frequently

```
Category root = openSession().createQuery(
   "from Category c fetch join c.categories
        where c.name =:root")
   .setString("root","Root")
   .uniqueResult();

displayCategoryTree(root);
closeSession();
```

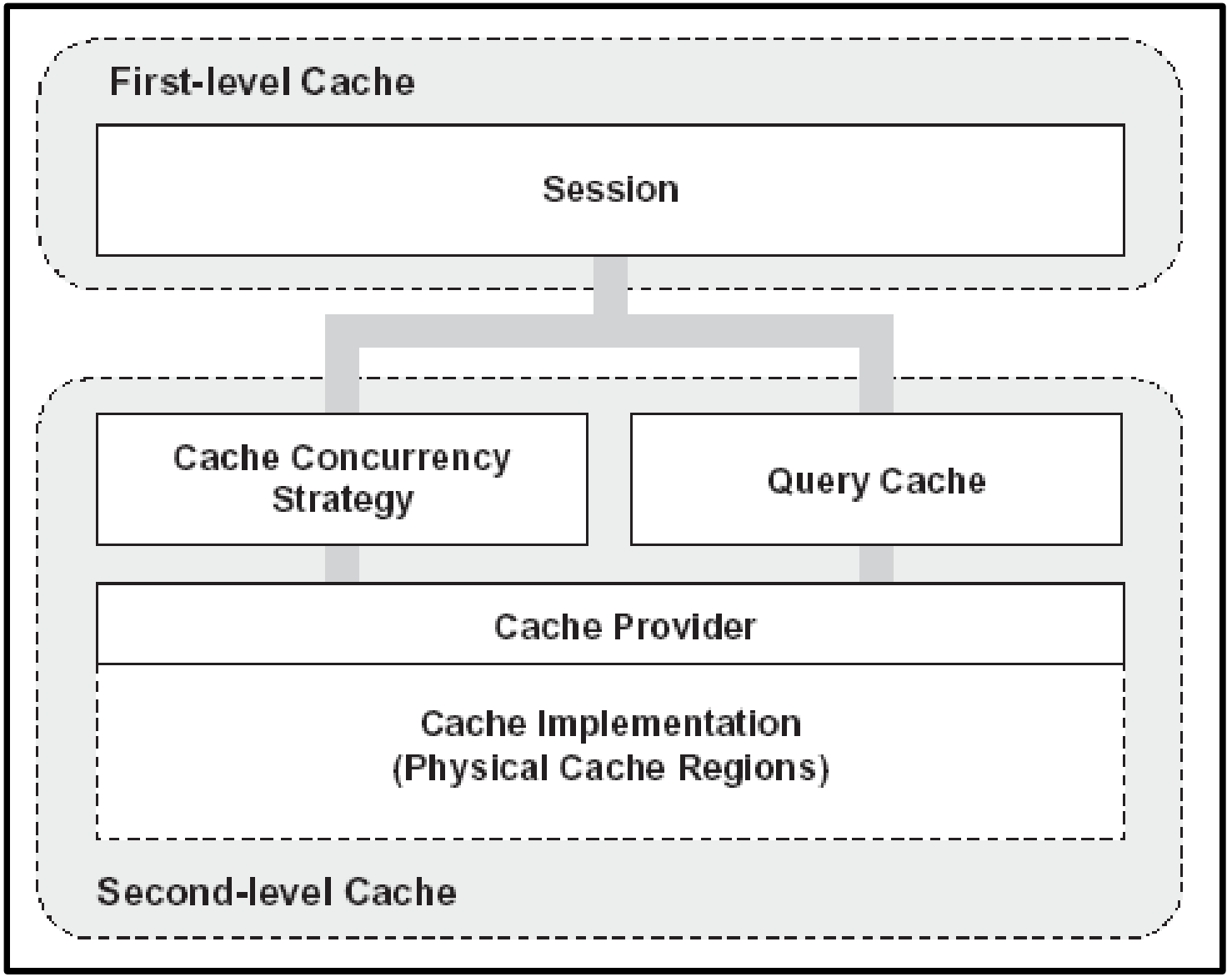
Caching Scope – Cont'

```
void displayCategoryTree(Category c){
    display(c);
    for (Category child : c.getChildren()) {
      displayCategoryTree (child);
    }
}
```

- On a Tree of n categories, this code will still have O(n) selects!

- We must use the 2nd level cache here…

Hibernate Cache Architecture



Using the 2nd level Cache

```
<hibernate-configuration>
 <session-factory>
   ...
   <property name="cache.provider_class">
          net.sf.ehcache.hibernate.EhCacheProvider
   </property>
   <property name="cache.use_query_cache">
        true
   </property>
   ...
   <class-cache class="com.myapp.Category"
        usage="read-write"/>
   <collection-cache
        collection="com.myapp.Category.children"
        usage="read-write"/>
 </session-factory>
</hibernate-configuration>
```

```
Category root = openSession().createQuery(
"from Category c where c.name =:root")
   .setString("root","Root")
   .setCacheable(true).uniqueResult();
displayCategoryTree(root);
closeSession();
```

Batch Processing Problem

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

List<Customer> customers =
   session.getNamedQuery(   "GetCustomers").list()

for (Customer customer : customers)
    customer.updateStuff(...);

tx.commit();
session.close();
```

Fail with an OutOfMemoryError somewhere around the 50,000th row

Batch Processing Example

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

ScrollableResults customers = session
    .getNamedQuery("GetCustomers")
    .setCacheMode(CacheMode.IGNORE)
    .scroll(ScrollMode.FORWARD_ONLY);
int count=0;
while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    if ( ++count % 20 == 0 ) {
        // flush a batch of updates and release memory
        session.flush();
        session.clear();
    }
}
tx.commit();
session.close();
```

StatelessSession

- Lower-level abstraction, much closer to the underlying JDBC.
- No associated persistence context.
- Does not implement a first-level cache nor interact with any second-level or query cache.
- Does not implement transactional write-behind or automatic dirty checking.
- Operations do not ever cascade to associated instances.
- Collections are ignored by a stateless session.
- Bypass Hibernate's event model and interceptors.
- insert(), update() and delete() have different semantics than save(), saveOrUpdate() and delete()

StatelessSession Example

```
StatelessSession session =
   sessionFactory.openStatelessSession();
Transaction tx = session.beginTransaction();
ScrollableResults customers = session
   .getNamedQuery("GetCustomers")
   .scroll(ScrollMode.FORWARD_ONLY);

while (customers.next()) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    session.update(customer);
}
tx.commit();
session.close();
```

DML-Style Operations

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

String hqlUpdate = "update Customer c set c.name =
   :newName where c.name = :oldName";

int updatedEntities = s.createQuery( hqlUpdate )
        .setString( "newName", newName )
        .setString( "oldName", oldName )
        .executeUpdate();
tx.commit();
session.close();
```

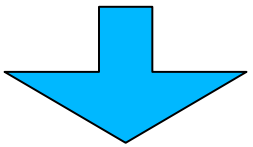
Filtering Data

```
<filter-def name="effectiveDate">
   <filter-param name="asOfDate" type="date"/>
</filter-def>
```

```
<class name="Employee" ...>
  ...
  <property name="userName"/>
  <property name="effectiveStartDate"/>
  <property name="effectiveEndDate"/>
  ...
  <filter name="effectiveDate"
    condition=":asOfDate BETWEEN
   effectiveStartDate and effectiveEndDate "/>
</class>
```

Filtering in Action

```
Session session = ...;
session.enabledFilter("effectiveDate")
         .setParameter  ("asOfDate", new Date());
List results = session.createQuery(
   "from Employee as e where e.salary > :targetSalary")
       .setLong("targetSalary", new Long(1000000))
       .list();
```



Return the currently working employees with salary above 1,000,000

Q&A

Thank You

yanai@tikalk.com