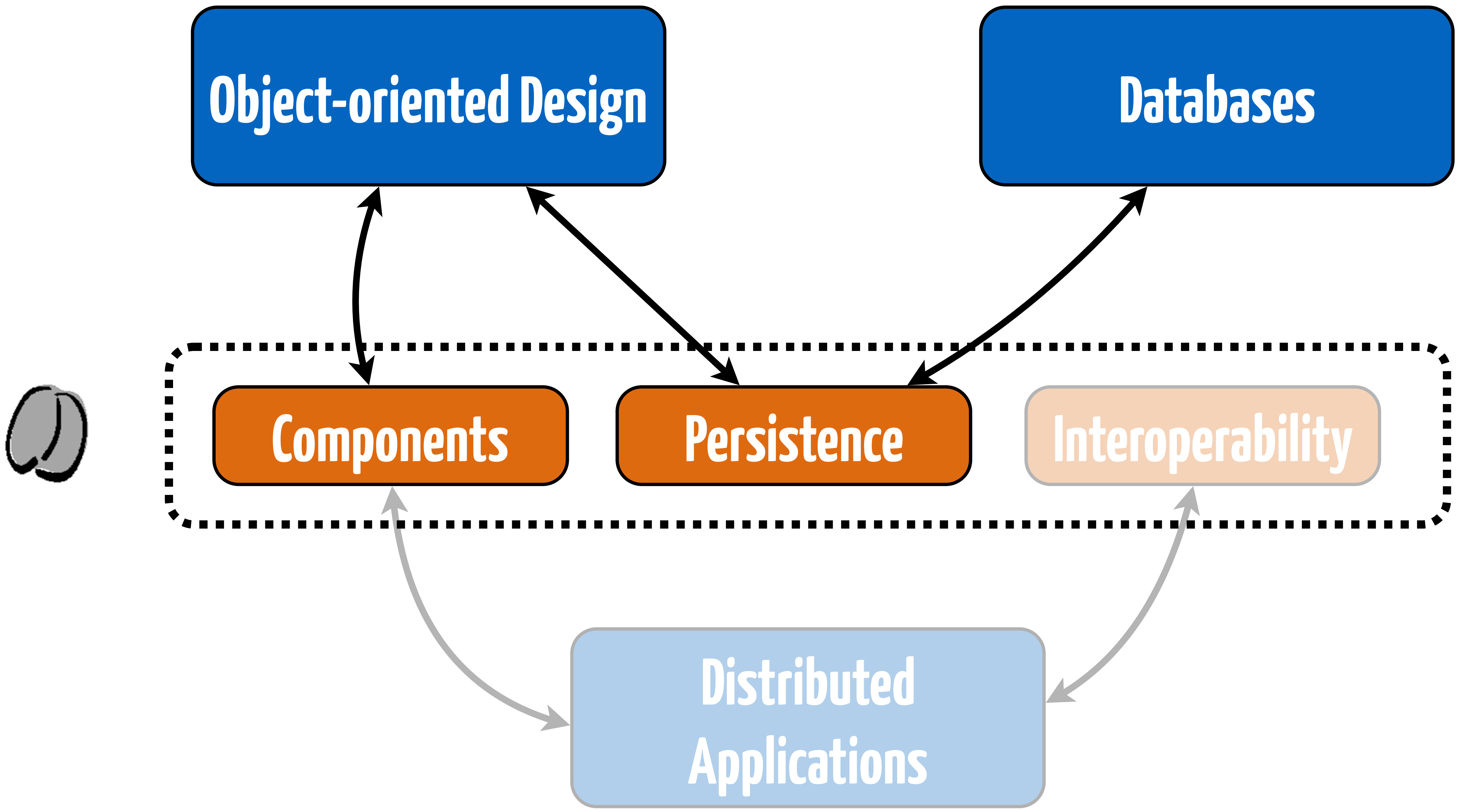
# Make your beans **persistent**

**Sébastien Mosser**

Lecture #4, 06.04.2018

# Applications Server: Dependencies



1 **Domain** Modeling

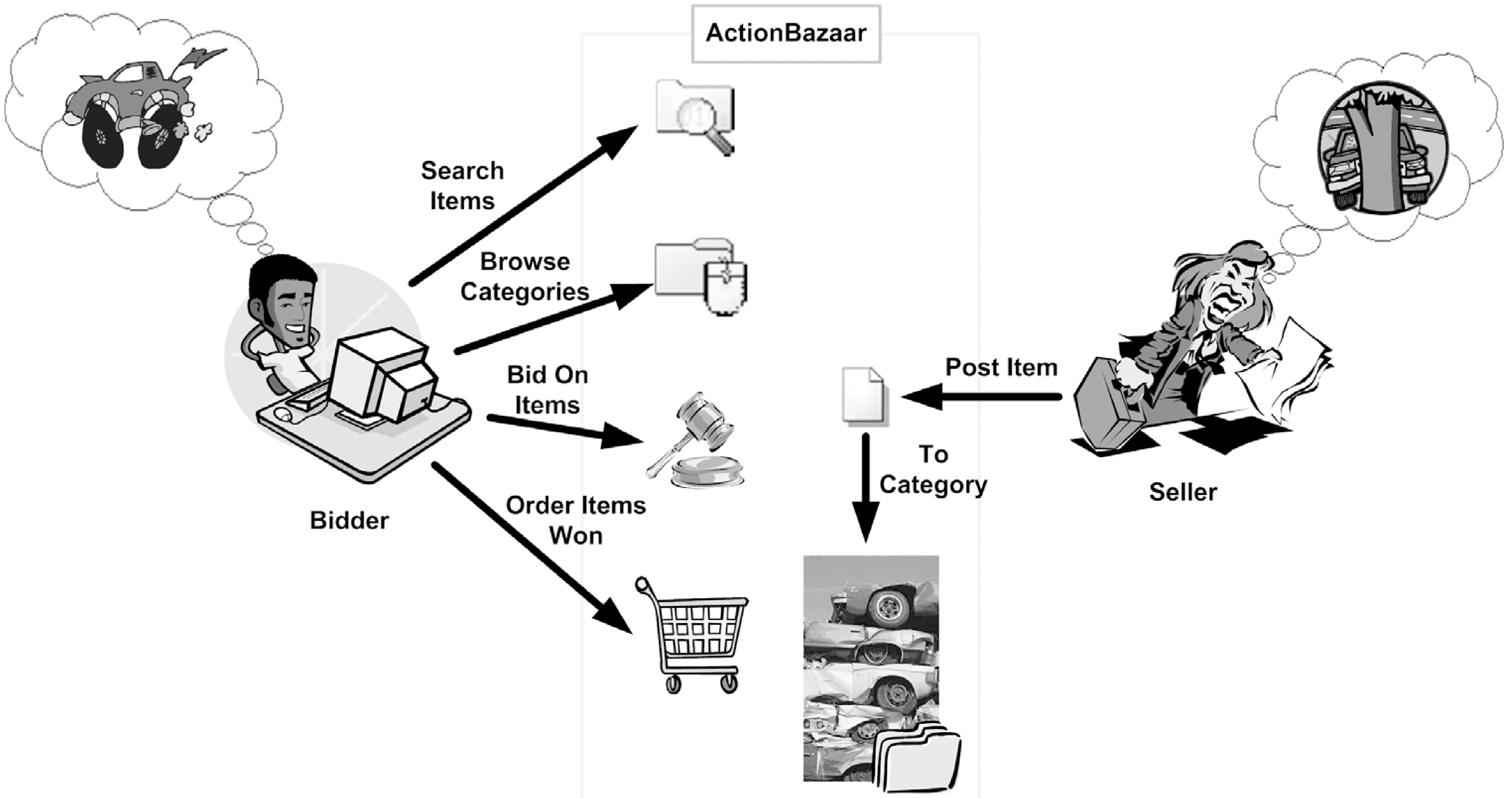
2 Modeling **relationships**

3 The **Entity Manager**

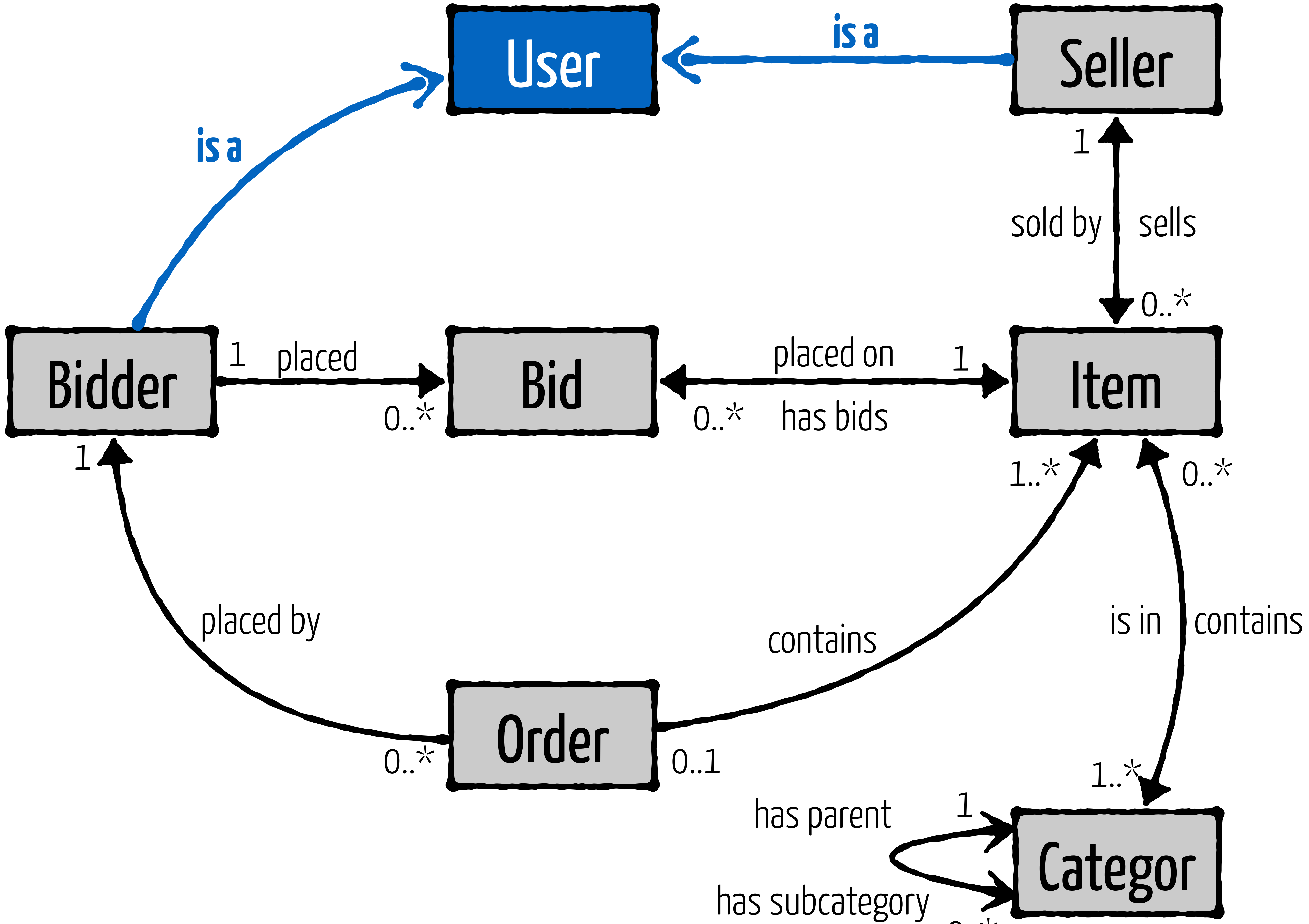
4 **From the trenches**

1

# Domain Modeling


ActionBazaar
Search Items
Browse Categories
Bid On Items
Order Items Won
Bidder
Post Item
To Category
Seller

[EiA]


User
is a
Seller
is a
1
sold by
sells
0..*
Bidder
1
placed
0..*
Bid
placed on
1
0..*
has bids
Item
1
1..*
0..*
placed by
contains
is in
contains
Order
0..*
0..1
1..*
has parent
1
Categor
has subcategory
0..*

[EiA]

```
@Entity
public class Category {

  public Category() { … }

  protected String name;

  public String getName() {
    return this.name;
  }

  public void setName(String n) {
    this.name = n.toUpperCase();
  }
}
```

property-based access

(JPA)

[EiA]

Categor

```
@Entity
public class Category {

  public Category() { … }

  public String name;
}
```

field-based
access

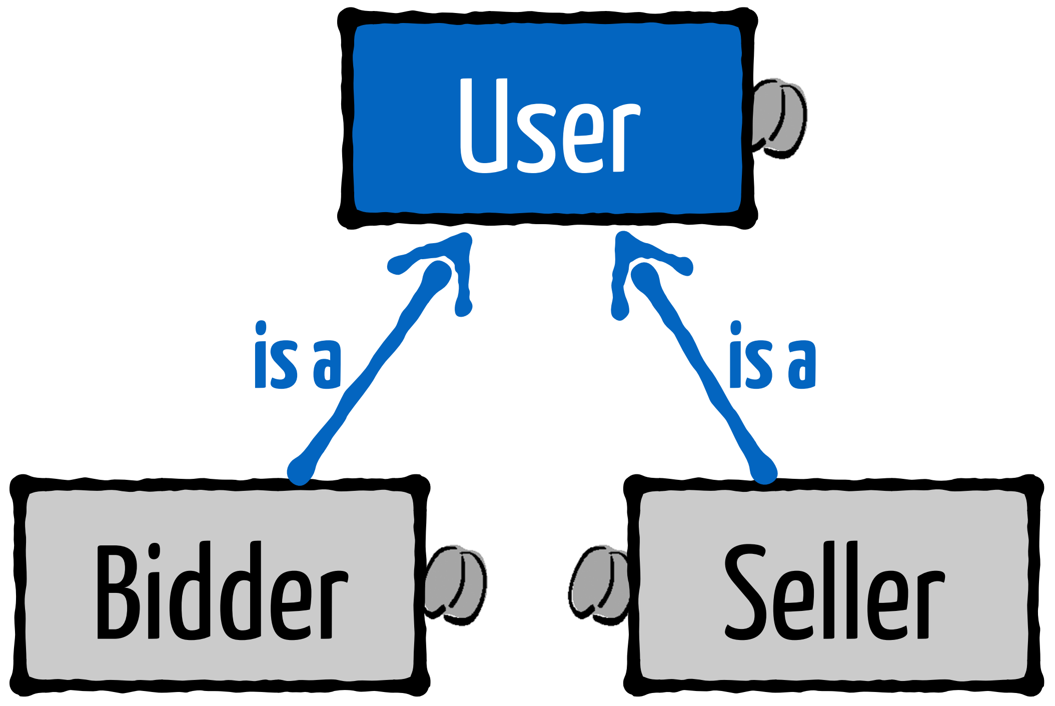
**Fields are simple but forbid encapsulation**

## Do not use fields

We're doing this here just to have examples that fit in a single slide

The container will behave badly with public attributes. Annotate getters.

```
@Entity
public abstract class User {
  // ...
}
```



```
@Entity
public class Bidder extends User {
  // ...
}

@Entity
public class Seller extends User {
  // ...
}
```

[EiA]

## Simple Primary Key: @Id

```
@Entity
public class Category {
   // …

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   public Long id;

}
```

**Identifiers must define an "equals" method**

[EiA]

## Composite Key: @IdClass

```
public class CategoryPK extends Serializable {
  String name;
  Date createDate;
}


@Entity
@IdClass(CategoryPK.class)
public class Category {

  @Id
  protected String name;

  @Id
  protected Date createDate;
}
```

## Identifiers must define an "equals" method

```
public class CategoryPK extends Serializable {

  public boolean equals(Object other) {

    if (other instanceOf CategoryPK) {
      final CategoryPK that = (CategoryPK) other;
      return that.name.equals(name) &&
             that.createDate.equals(createDate);
    return false;

  }

  public int hashCode() {
    return super.hashCode();
  }
}
```

[EiA]

**Serializable requirements**

## Equality Relation definition

- equals is reflexive
- equals is symmetric
- equals is transitive
- equals is consistent
- equals uses null as absorbing element

**It's complicated!**

# Embeddable Objects

```
@Embeddable
public class Address {
  protected String street;
  protected String city;
  protected String zipcode;
}
```

does not need an UID

```
@Entity
public class User {

   @Id
   protected Long id;

   @Embedded
   protected Address address;
}
```

Shared Identifier

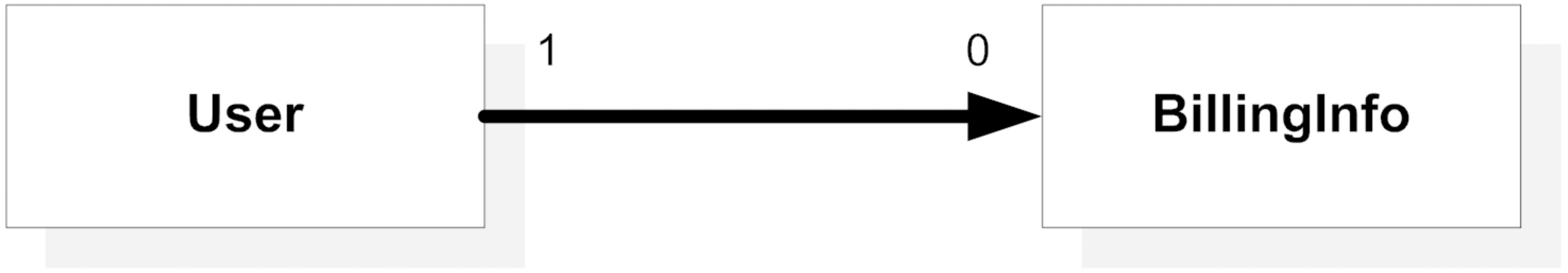
[EiA]

## Modeling relationships

2

| Type of Relationship | Annotation |
|---|---|
| 1-1 | `@OneToOne` |
| 1-n | `@OneToMany` |
| n-1 | `@ManyToOne` |
| n-m | `@ManyToMany` |

[EiA]



## Unidirectional 1-1 mapping

```
@Entity
public class User {
   @Id
   protected String userId;
   protected String email;

   @OneToOne
   protected BillingInfo billingInfo;
}
```

[EiA]

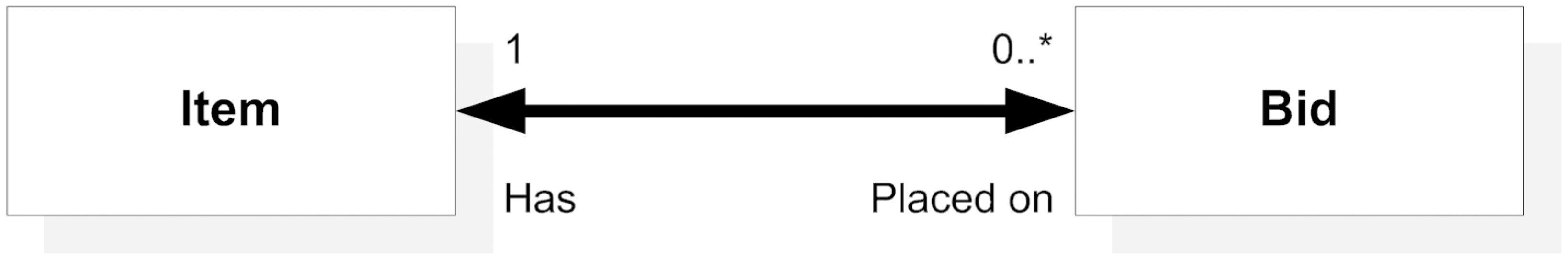
For property-based beans, annotate the getter.

## Bidirectional

## 1-1 mapping

```
@Entity
public class User {
   @Id
   protected String userId;
   protected String email;

   @OneToOne
   protected BillingInfo billingInfo;
}

@Entity
public class BillingInfo {
   @Id
   protected Long billingId;

   @OneToOne(mappedBy="billingInfo",optional=false)
   protected User user;
}
```

[EiA]

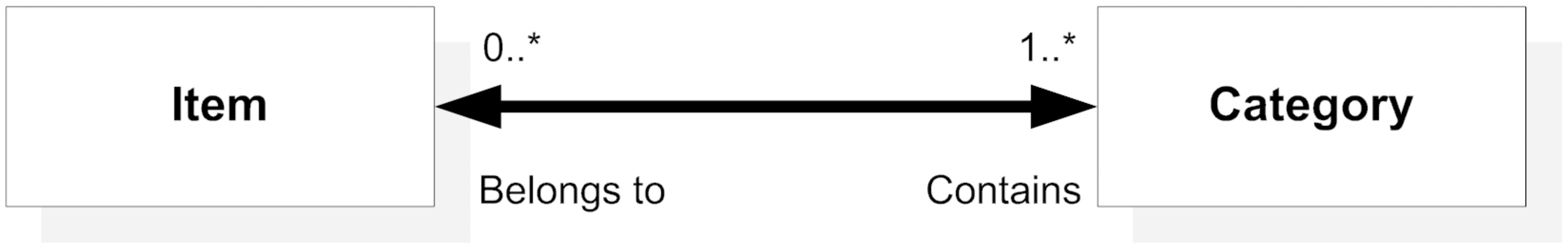


```
@Entity
public class Bid {
  @Id
  protected String bidId;
  @ManytoOne
  protected Item item;
}

@Entity
public class Item {
  @Id
  protected String itemId;
  @OneToMany(mappedBy="item")
  protected Set<Bid> bids;
}
```

Owner

1-n mapping

[EiA]



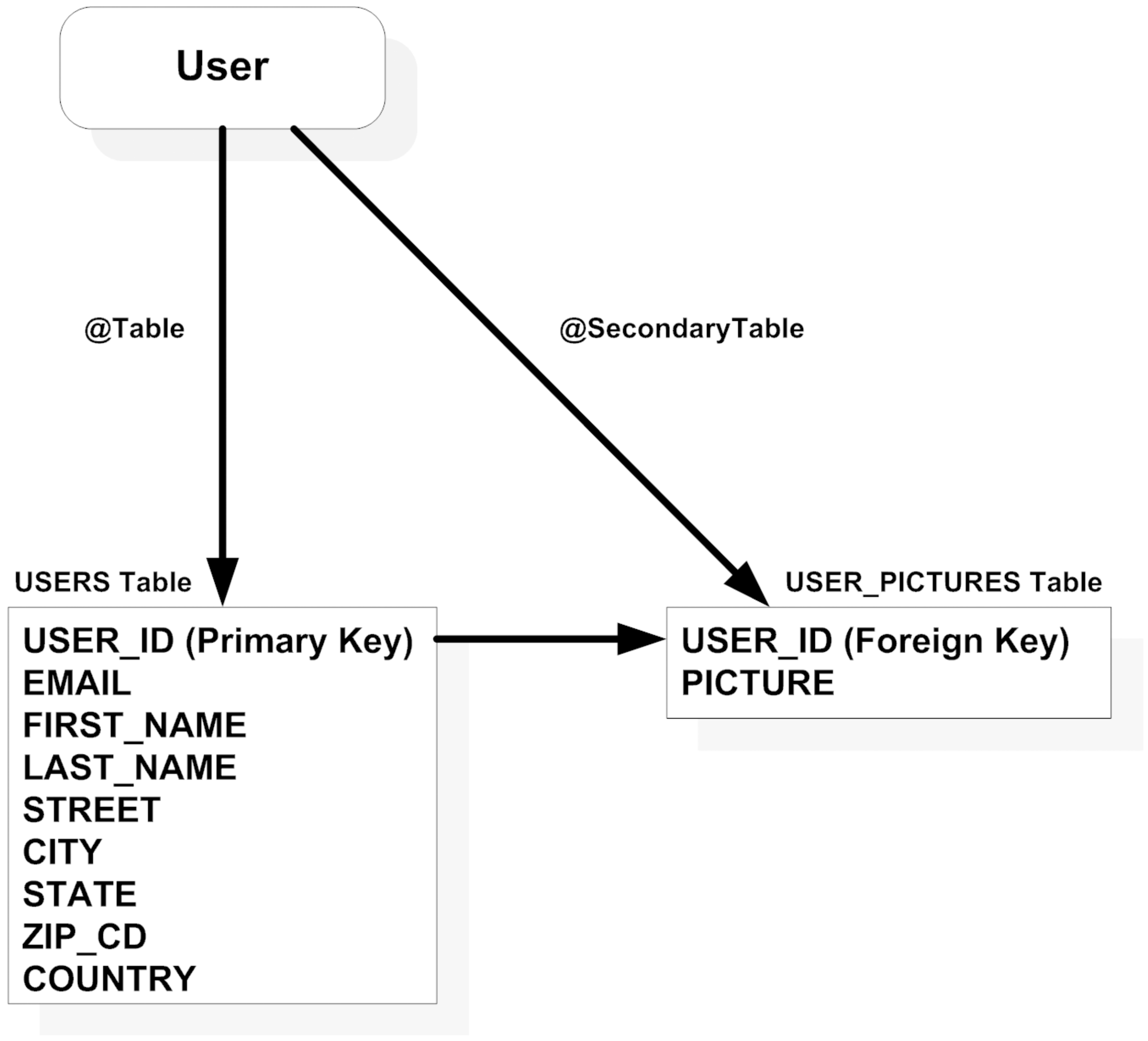
```
@Entity
public class Category {
   @Id
   protected String categoryId;
   @ManytoMany
   protected Set<Item> items;
}

@Entity
public class Item {
   @Id
   protected String itemId;
   @ManytoMany(mappedBy="items")
   protected Set<Category> categories;
}
```

Owner

**n-m mapping**

[EiA]

## **Controlling** the Object-Relational mapping



```
@Entity
@Table(name="USERS")
public class User {
  @Id
  @Column(name="USER_ID")
  protected String userId;
  // …
}
```

[EiA]

## Controlling Inheritance

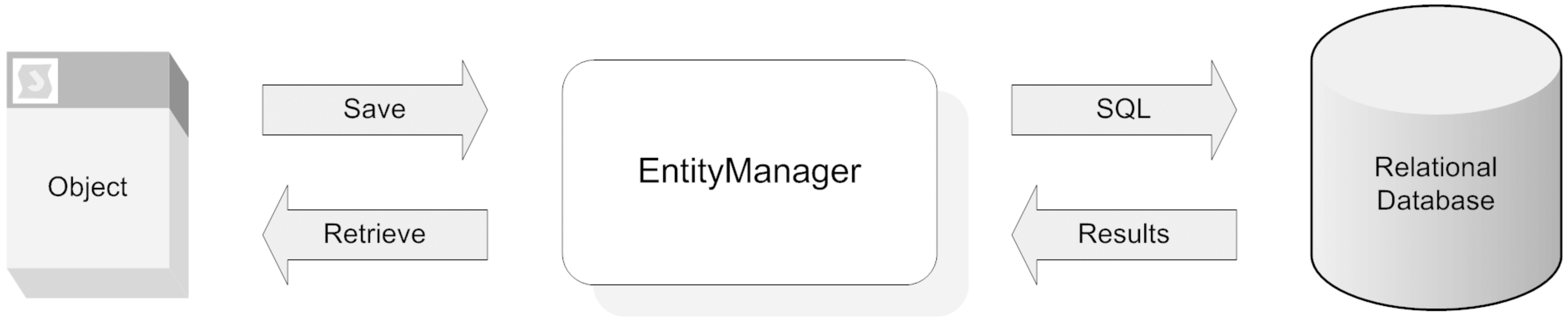
```
@Entity
@Table(name="USERS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="USER_TYPE", …)
public class User {
  // …
}

@Entity
@DiscriminatorValue(value="S")
public class Seller extends User { … }

// …
```

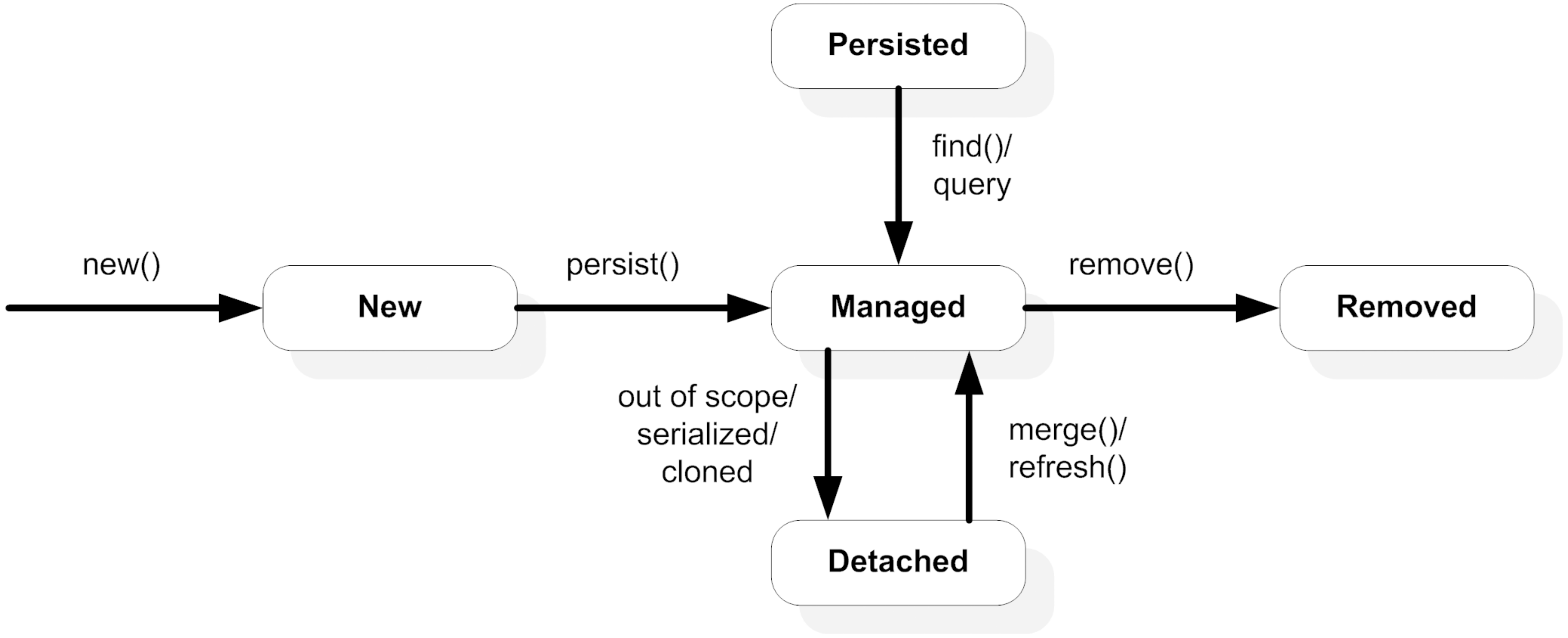
See [EiA], chapter 8

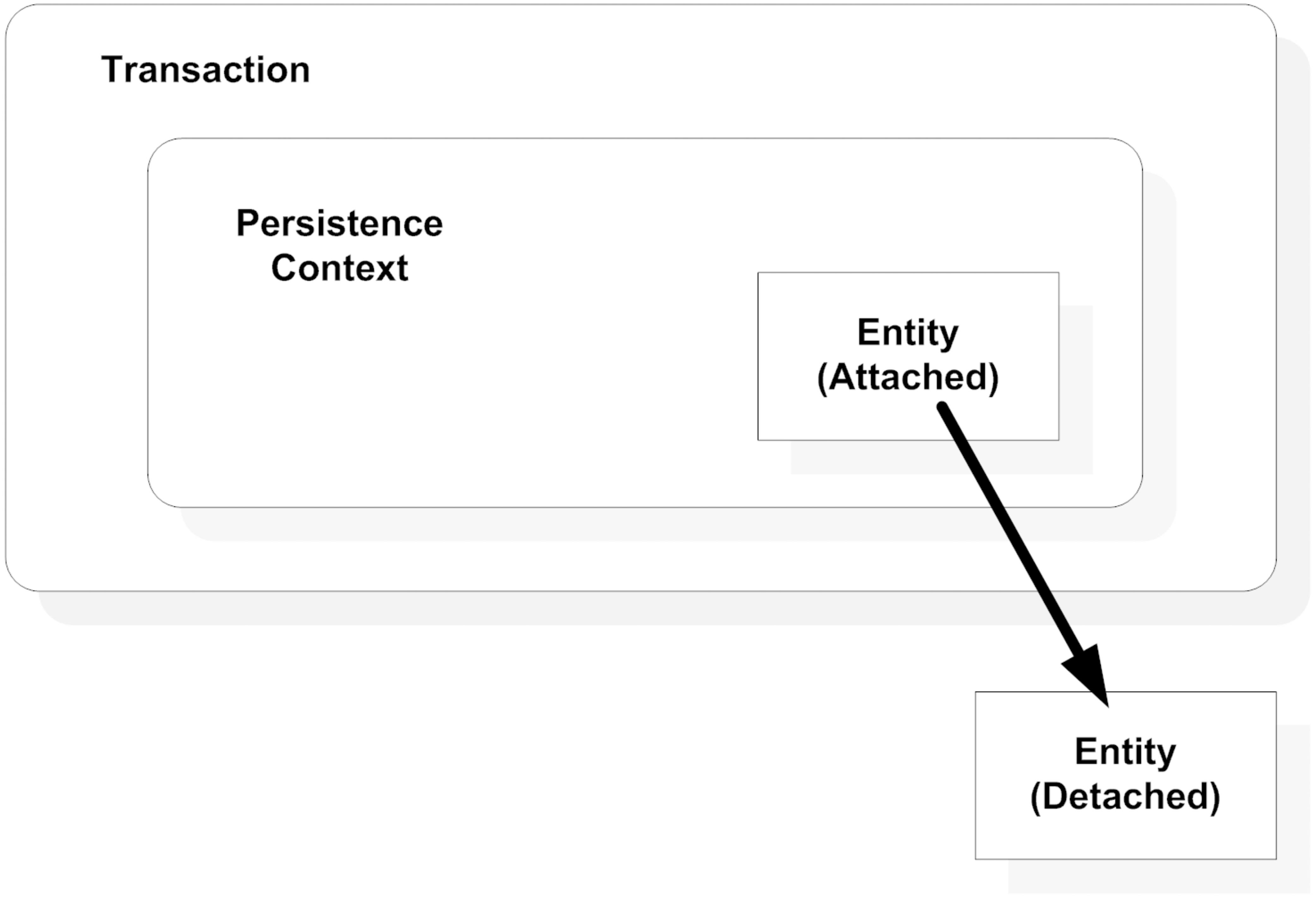
[EiA]

3

# The Entity Manager


Object
Save
Retrieve
EntityManager
SQL
Results
Relational Database


Persisted
find()/
query
new()
New
persist()
Managed
remove()
Removed
out of scope/
serialized/
cloned
merge()/
refresh()
Detached


Transaction
Persistence Context
Entity (Attached)
Entity (Detached)

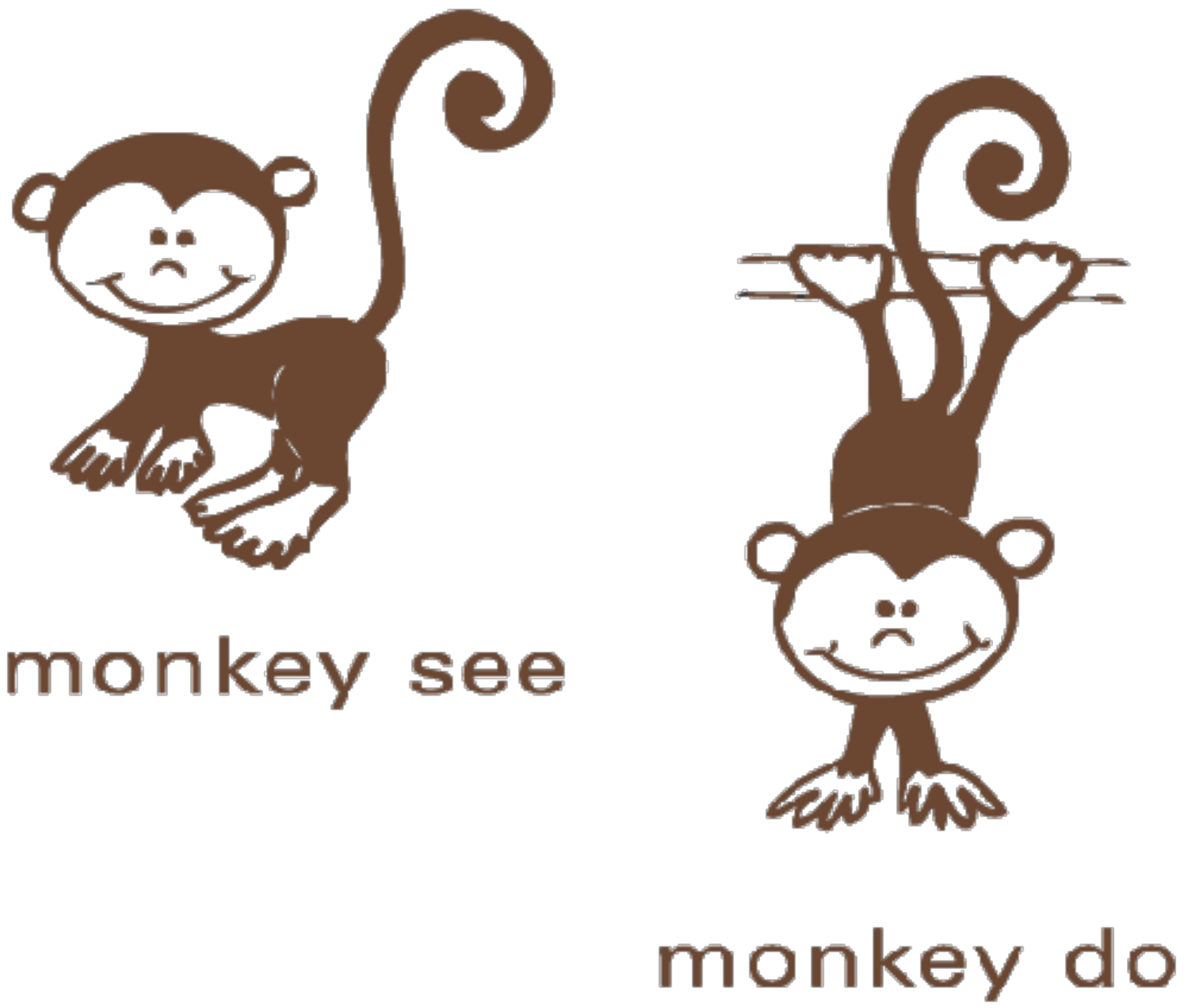
[EiA]

# Persistence context is **Injected**

```
@PersistenceContext(unitName="admin")
EntityManager manager

@Resource
private UserTransaction transaction;

public void createAndStore() {

  AnEntityBean b = new AnEntityBean("Parameters");
  transaction.begin();
  try {
    manager.persist(b);
  } finally {
    transaction.commit();
  }

}
```

**See [EiA], chapter 9**

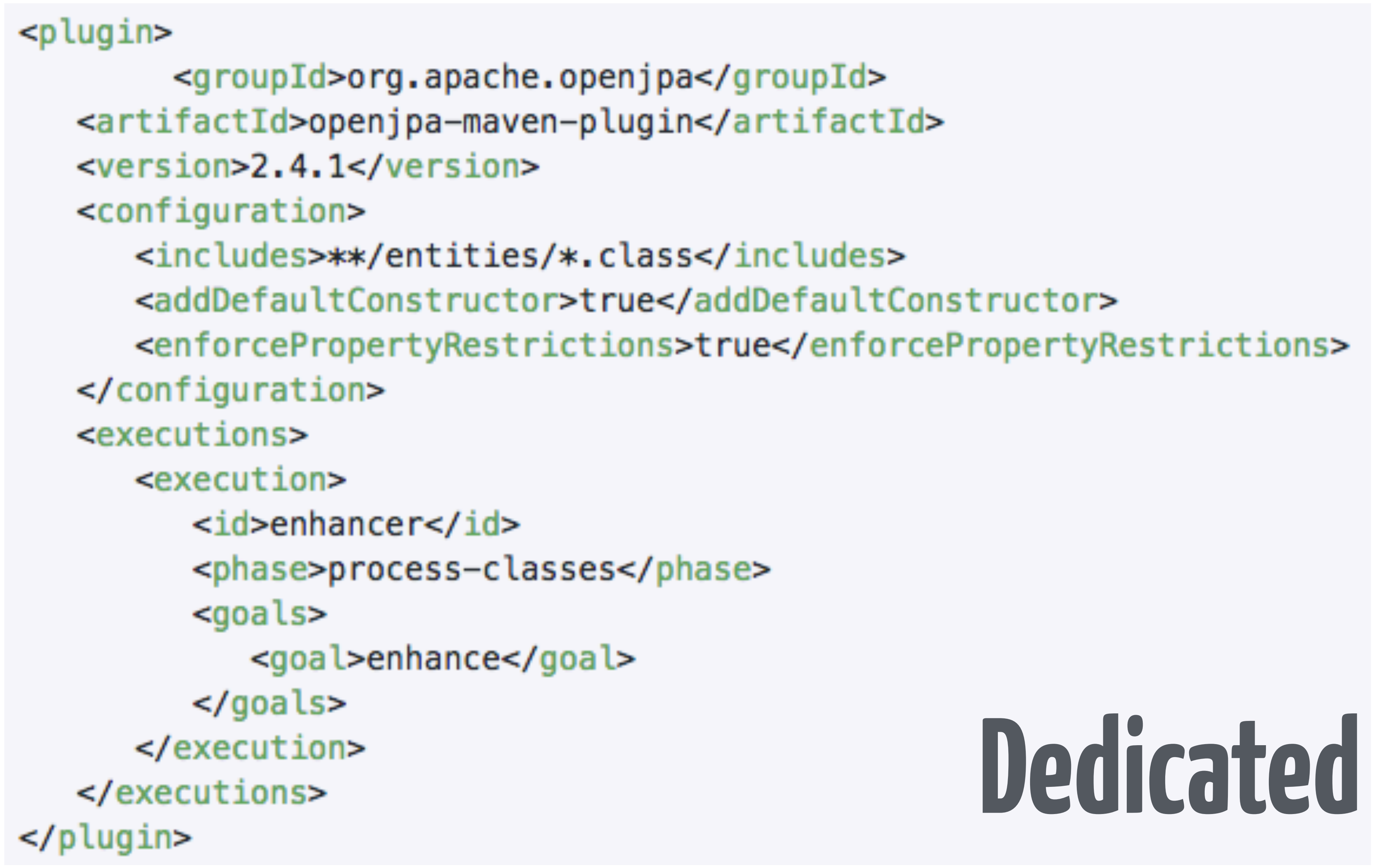
# EJB are **standard**: Learn by **Example**!



http://infiltrated.net/index.php?option=com_content&view=article&id=43&Itemid=49

# 4 From the trenches

## Bytecode enhancement



```
<plugin>
        <groupId>org.apache.openjpa</groupId>
   <artifactId>openjpa-maven-plugin</artifactId>
   <version>2.4.1</version>
   <configuration>
      <includes>**/entities/*.class</includes>
      <addDefaultConstructor>true</addDefaultConstructor>
      <enforcePropertyRestrictions>true</enforcePropertyRestrictions>
   </configuration>
   <executions>
      <execution>
         <id>enhancer</id>
         <phase>process-classes</phase>
         <goals>
            <goal>enhance</goal>
         </goals>
      </execution>
   </executions>
</plugin>
```

Dedicated Java agent

spoon-like mechanism

# Prod ≠ Test

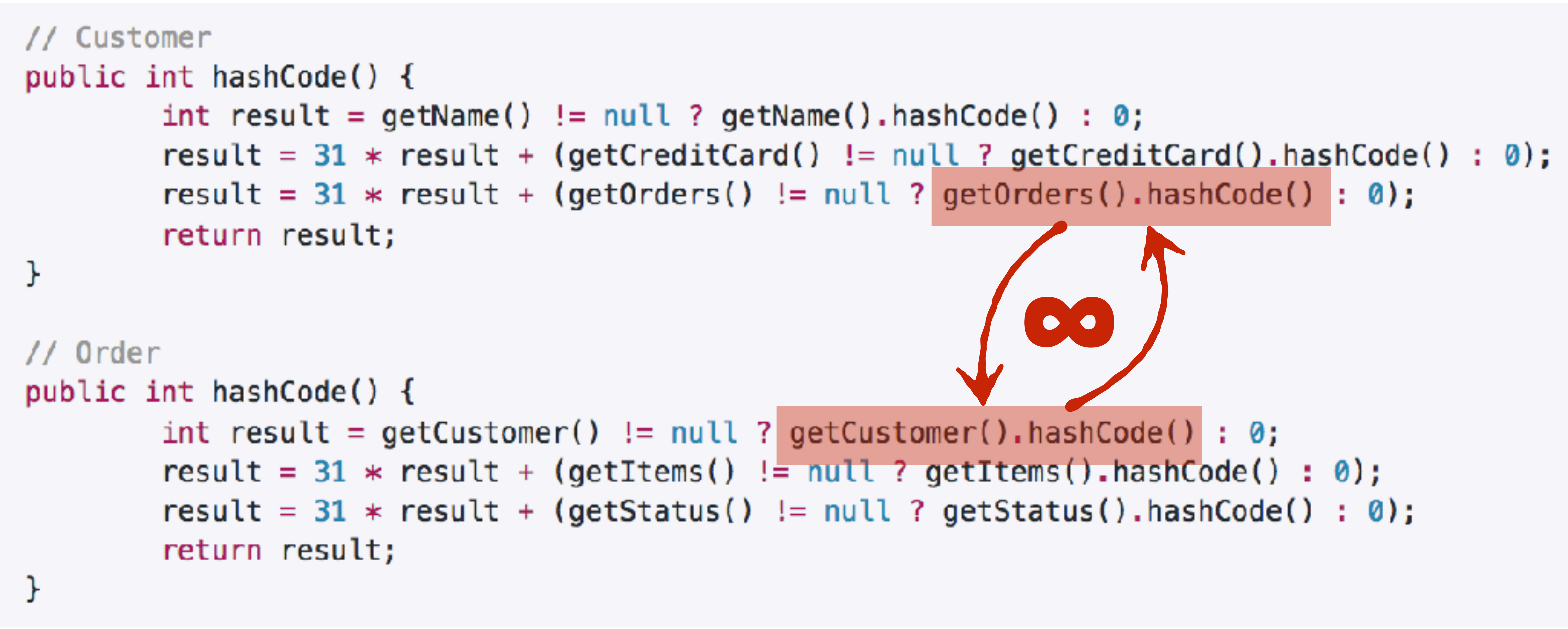
```
<resources>
    <Resource id="production" type="DataSource">
        JdbcDriver  org.hsqldb.jdbcDriver
        JdbcUrl     jdbc:hsqldb:file:proddb
        UserName    sa
        Password
        LogSql      true
        JtaManaged  true
    </Resource>
</resources>
```

WEB-INF/resources.xml

```
<property name="properties">
  my-datasource = new://Resource?type=DataSource
  my-datasource.JdbcUrl = jdbc:hsqldb:mem:TCFDB;shutdown=true
  my-datasource.UserName = sa
  my-datasource.Password =
  my-datasource.JtaManaged = true
  my-datasource.LogSql = true
</property>
```

arquillian.xml

# Auto-generated equals / hashcode



```java
// Customer
public int hashCode() {
        int result = getName() != null ? getName().hashCode() : 0;
        result = 31 * result + (getCreditCard() != null ? getCreditCard().hashCode() : 0);
        result = 31 * result + (getOrders() != null ? getOrders().hashCode() : 0);
        return result;
}

// Order
public int hashCode() {
        int result = getCustomer() != null ? getCustomer().hashCode() : 0;
        result = 31 * result + (getItems() != null ? getItems().hashCode() : 0);
        result = 31 * result + (getStatus() != null ? getStatus().hashCode() : 0);
        return result;
}
```

```
Customer seb = new Customer();
seb.setName("Sébastien");
seb.setCard("1234567890");
entityManager.persist(seb);
```

```
Customer clone = new Customer();
clone.setName("Sébastien");
clone.setCard("1234567890");
```

## seb.equals(clone) ?

Never ever use a database primary key as part of your business object equality definition

## Structural constraints / Validation

```
@Entity
public class Customer implements Serializable {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private int id;

        @NotNull
        private String name;

        @NotNull
        @Pattern(regexp = "\\d{10}+", message = "Invalid creditCardNumber")
        private String creditCard;

        @OneToMany(mappedBy = "customer")
        private Set<Order> orders = new HashSet<>();

        // ...
}
```

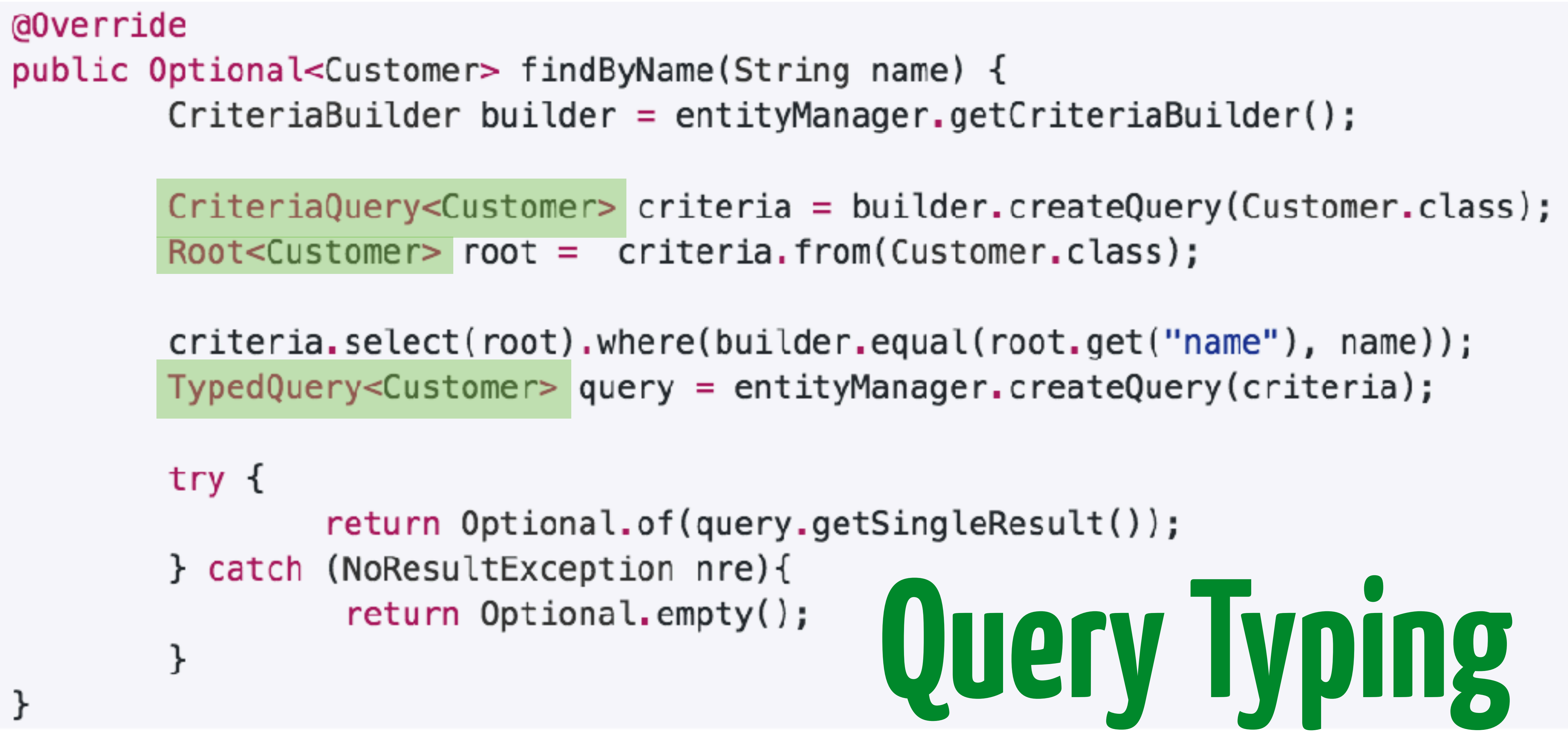
## Classical querying

```java
int id = 42;
Customer c = (Customer) entityManager.find(Customer.class, id);
```

```java
entityManager.createQuery("DELETE FROM Customer").executeUpdate();
```

## Issues?

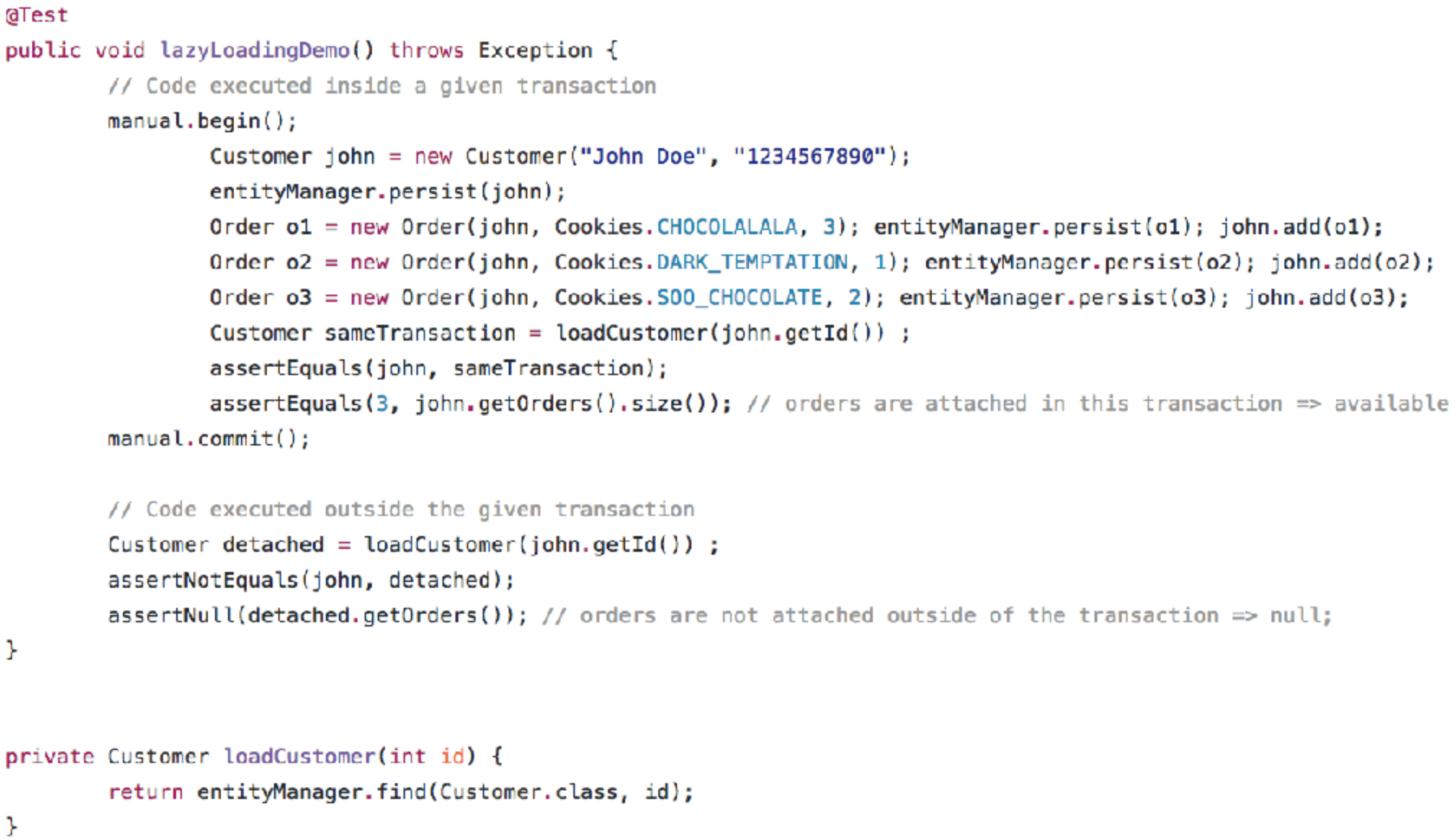
# EQL: EJB Query Language



```
@Override
public Optional<Customer> findByName(String name) {
        CriteriaBuilder builder = entityManager.getCriteriaBuilder();

        CriteriaQuery<Customer> criteria = builder.createQuery(Customer.class);
        Root<Customer> root =  criteria.from(Customer.class);

        criteria.select(root).where(builder.equal(root.get("name"), name));
        TypedQuery<Customer> query = entityManager.createQuery(criteria);

        try {
                return Optional.of(query.getSingleResult());
        } catch (NoResultException nre){
                 return Optional.empty();
        }
}
```

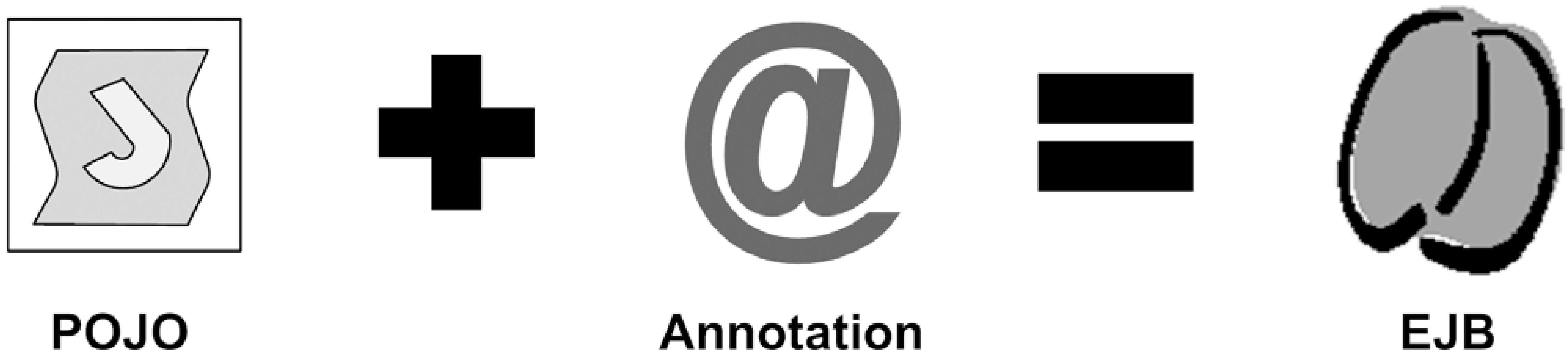
## Query Typing

# Lazy loading & Detachment



```java
@Test
public void lazyLoadingDemo() throws Exception {
        // Code executed inside a given transaction
        manual.begin();
                Customer john = new Customer("John Doe", "1234567890");
                entityManager.persist(john);
                Order o1 = new Order(john, Cookies.CHOCOLALALA, 3); entityManager.persist(o1); john.add(o1);
                Order o2 = new Order(john, Cookies.DARK_TEMPTATION, 1); entityManager.persist(o2); john.add(o2);
                Order o3 = new Order(john, Cookies.SOO_CHOCOLATE, 2); entityManager.persist(o3); john.add(o3);
                Customer sameTransaction = loadCustomer(john.getId()) ;
                assertEquals(john, sameTransaction);
                assertEquals(3, john.getOrders().size()); // orders are attached in this transaction => available
        manual.commit();

        // Code executed outside the given transaction
        Customer detached = loadCustomer(john.getId()) ;
        assertNotEquals(john, detached);
        assertNull(detached.getOrders()); // orders are not attached outside of the transaction => null;
}


private Customer loadCustomer(int id) {
        return entityManager.find(Customer.class, id);
}
```

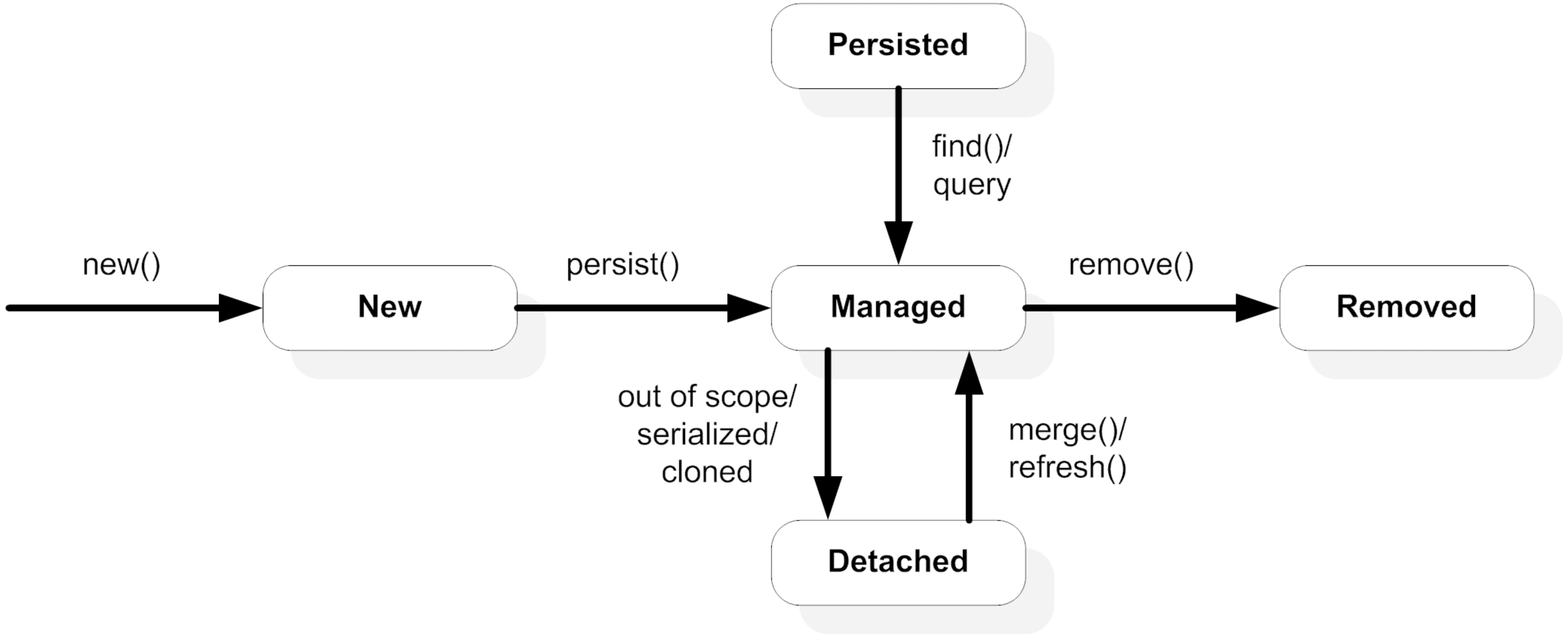
# Conclusions 5



```
@Entity
@Table(name="USERS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="USER_TYPE", …)
public abstract class User {
  // …
}
```

| Type of Relationship | Annotation |
|---|---|
| 1-1 | `@OneToOne` |
| 1-n | `@OneToMany` |
| n-1 | `@ManyToOne` |
| n-m | `@ManyToMany` |

[EiA]


Persisted
find()/
query
new()
New
persist()
Managed
remove()
Removed
out of scope/
serialized/
cloned
merge()/
refresh()
Detached