Persistence

Philippe Collet, contains 78,3% of slides from Sébastien Mosser

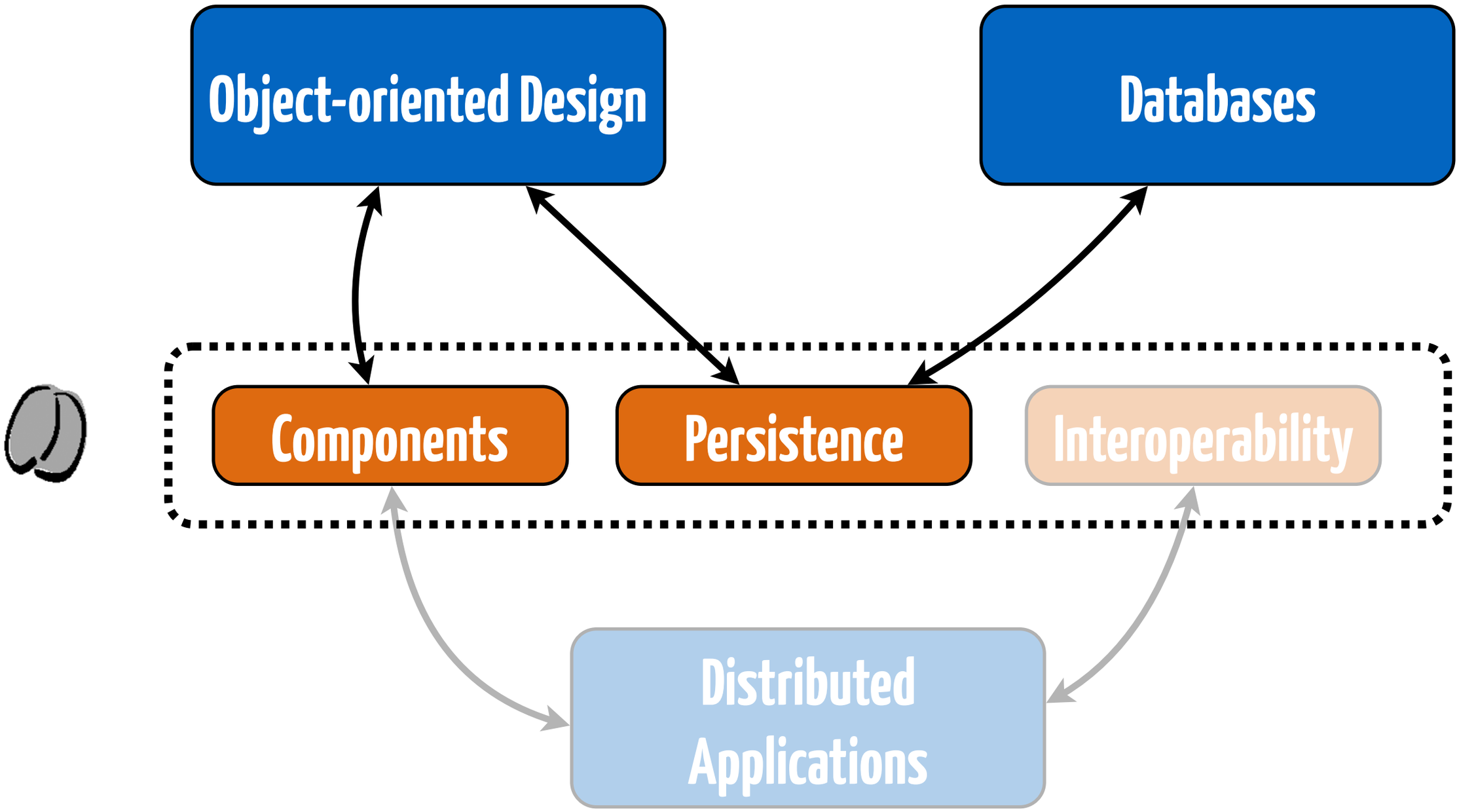
Lecture #X 29.03.2019

pictures: sxc.hu, pixabay

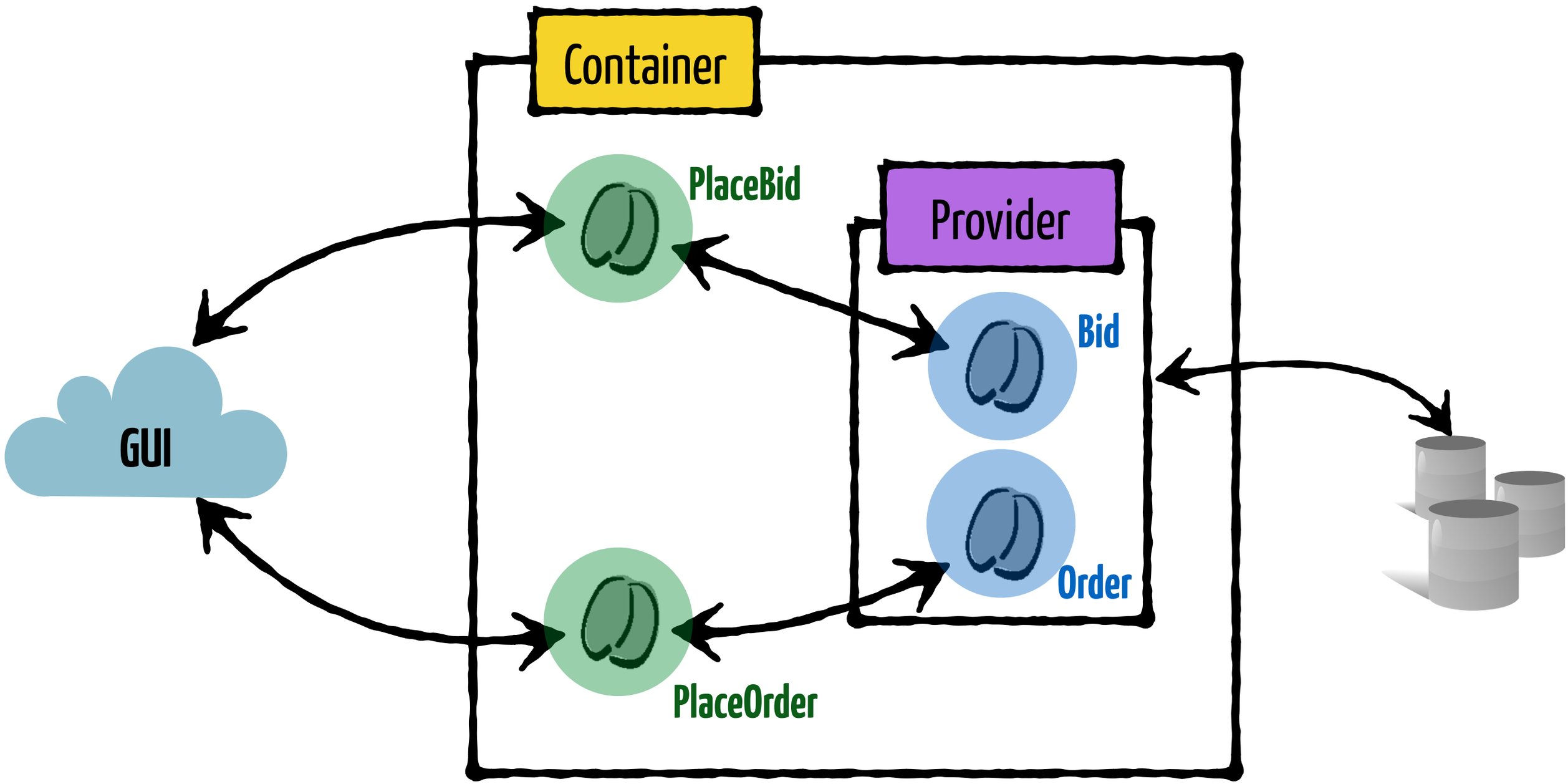
Object-Relational Mapping

Principles & Patterns

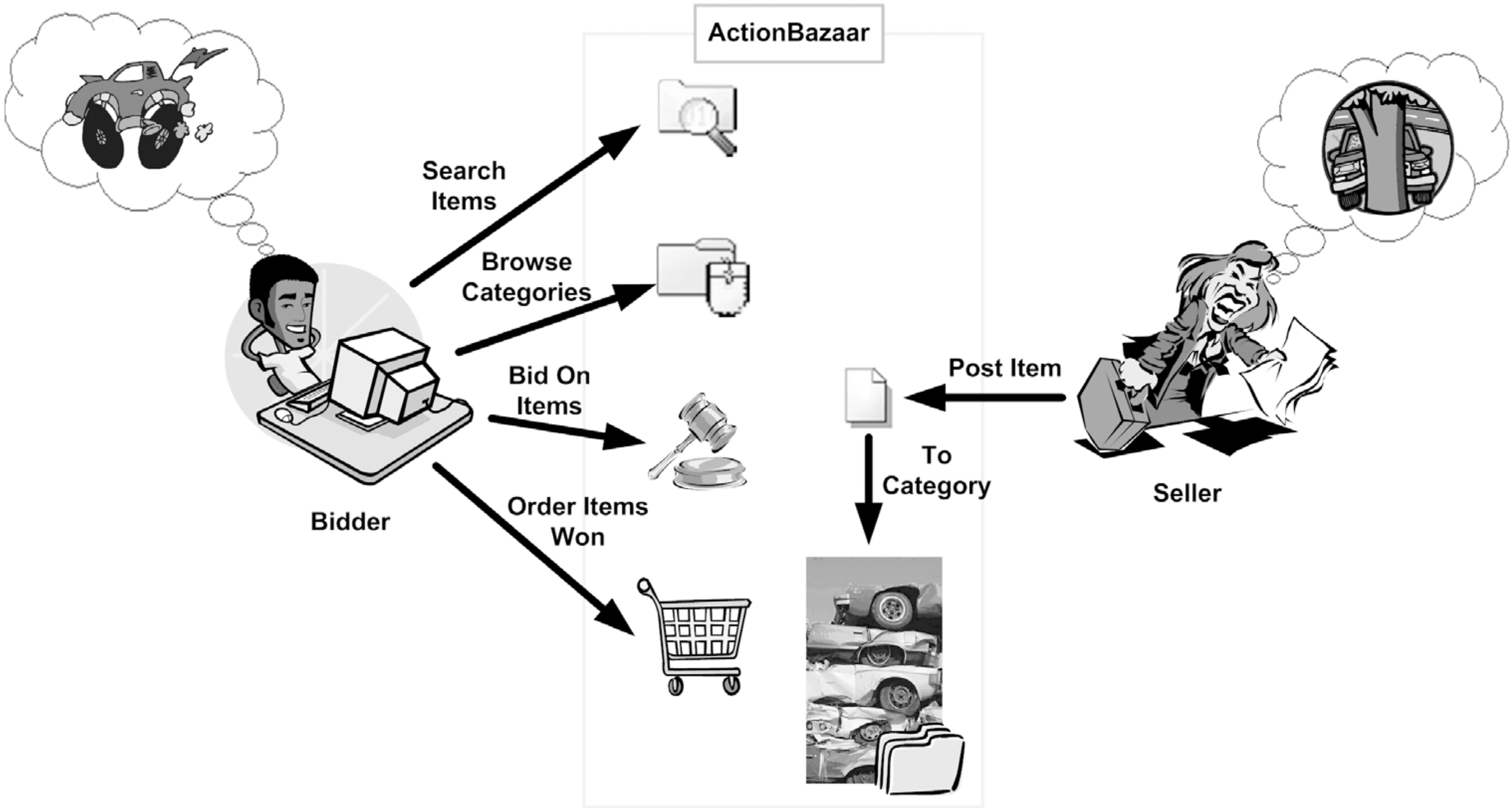
Applications Server: Dependencies



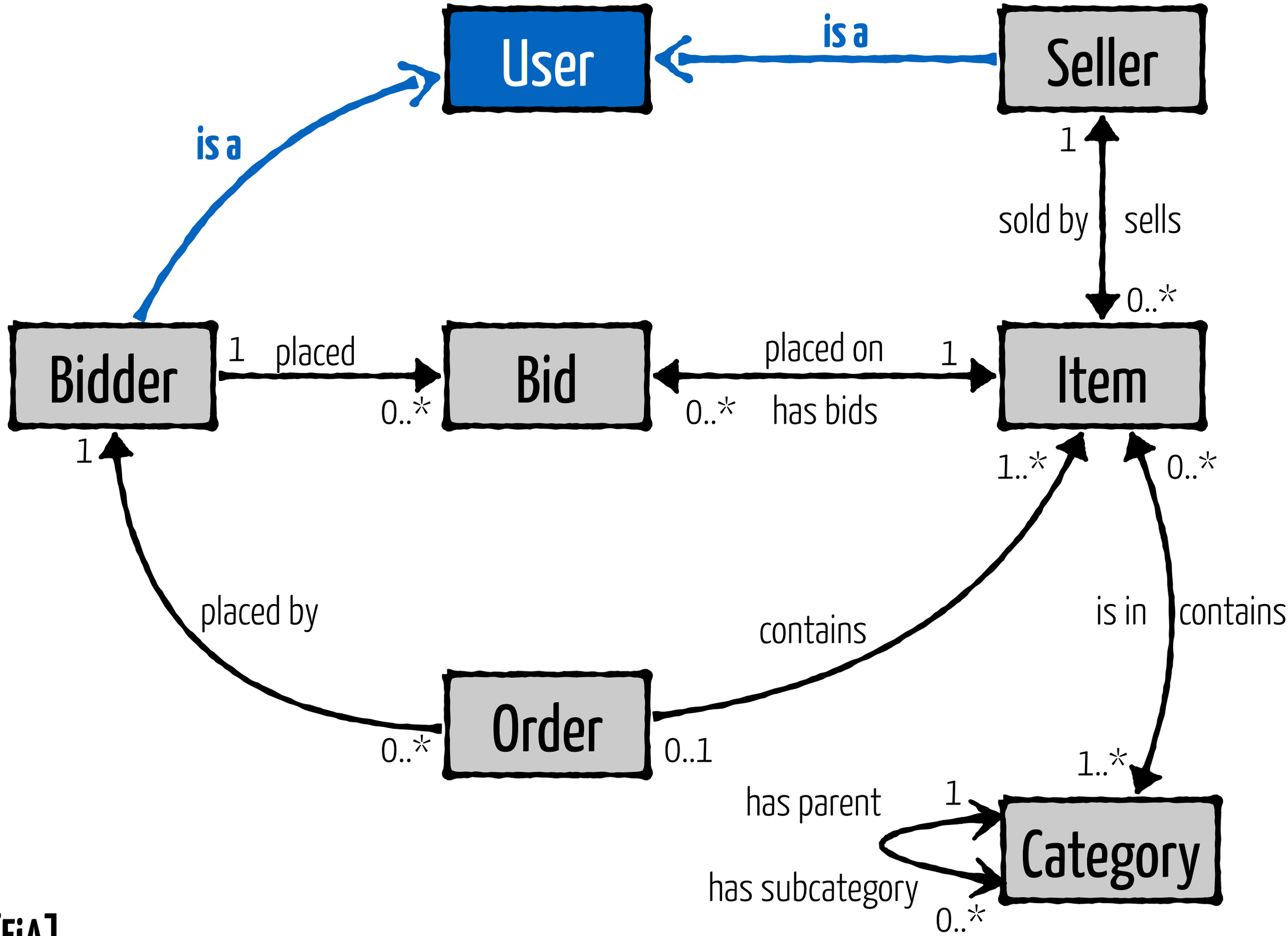
Session & Entity



[EiA]


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
Category
has subcategory
0..*

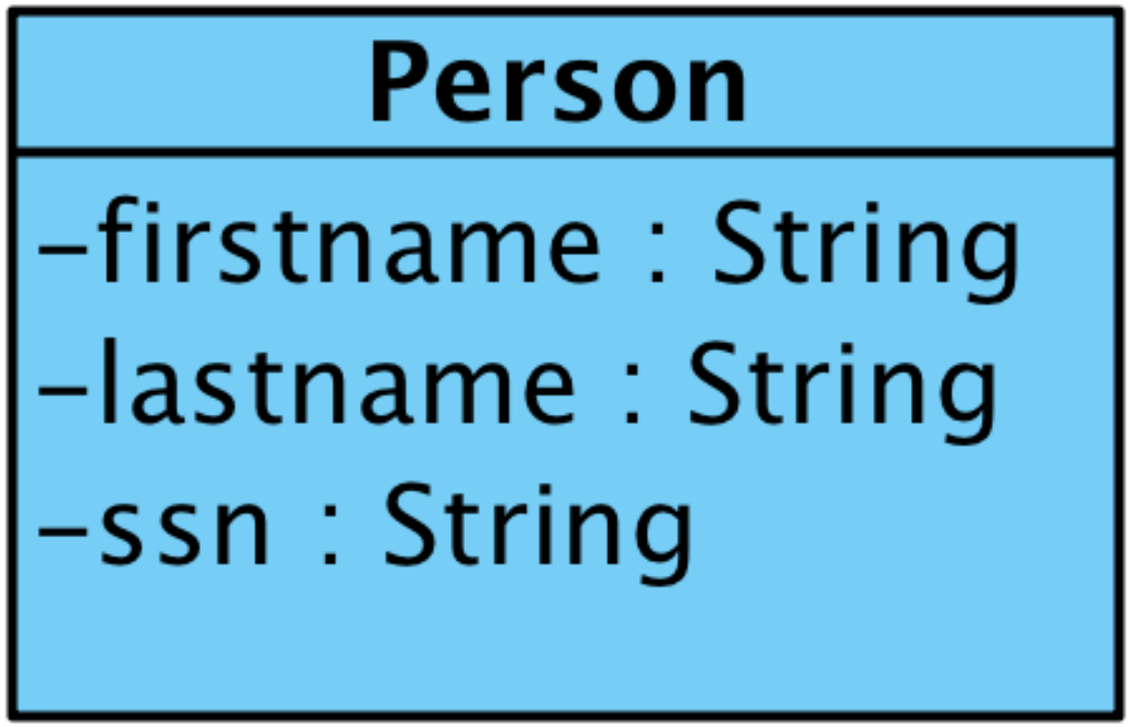
[EiA]

Impedance Mismatch

Object-Oriented	Relational
Classes	Relation (table)
Object	Tuple (row)
Attribute	Attribute (column)
Identity	Primary Key
Reference	Foreign Key
Inheritance	N/A
Methods	~ Stored Procedure

[EJB3 in Action]

Example of **Domain Model**



first_name	last_name	ssn
Sébastien	MOSSER	16118325358
...		

[PoEAA]

EJB Entities need more than simple annotations:

- An empty constructor
- A proper equals method that relies on business items
- A proper hashCode method to support objects identification in caches

Category

```
@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

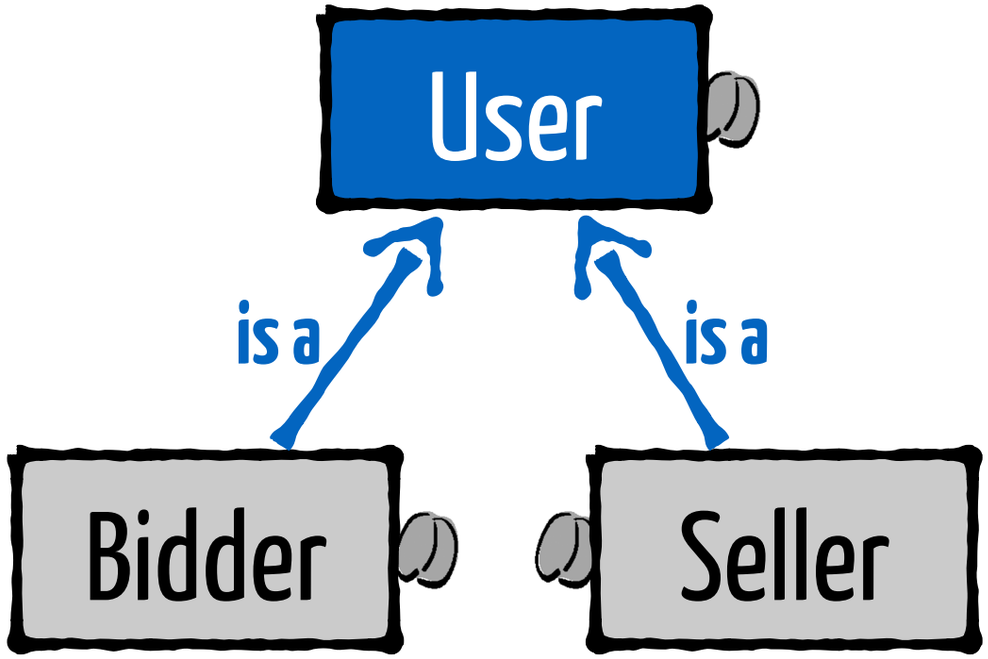
Fields are simple but forbid encapsulation

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;
}
```

[EiA]

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

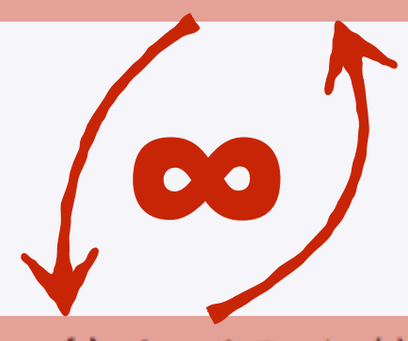
Equality Relation definition

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

It's complicated!

Auto-generated equals / hashcode

```
// 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

Embeddable Objects

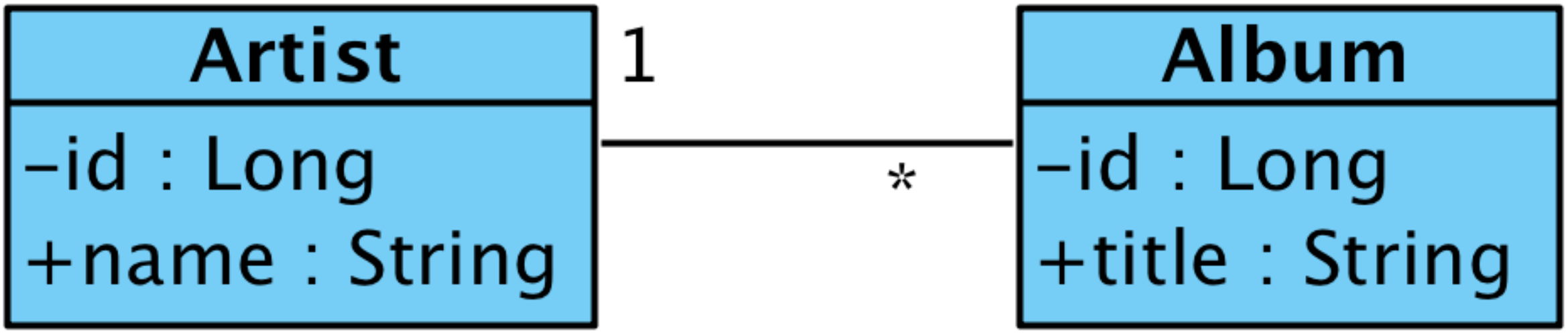
```
@Embeddable
public class Address {
  protected String street;
  protected String city;
  protected String zipcode;
}

@Entity
public class User {

  @Id
  protected Long id;

  @Embedded
  protected Address address;
}
```

does not need an UID

Shared Identifier

[EiA]

Problem: Representing associations



artists

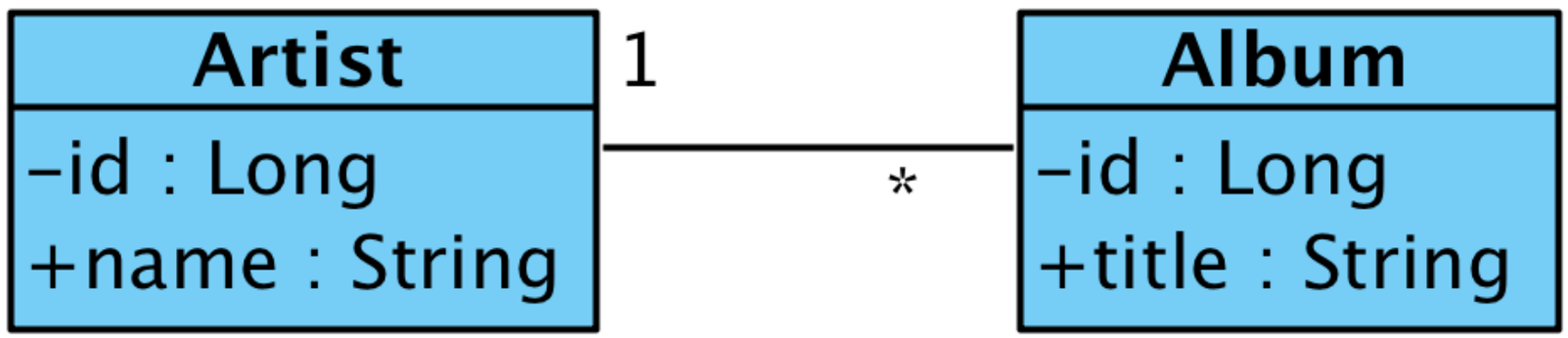
id	name
1	Linkin Park
	...

albums

id	title
1	A Thousand Suns
2	Minutes to Midnight
	...

[PoEAA]

Solution #1: **Association Table** [M-N]



artists

id	name
1	Linkin Park
...	

albums

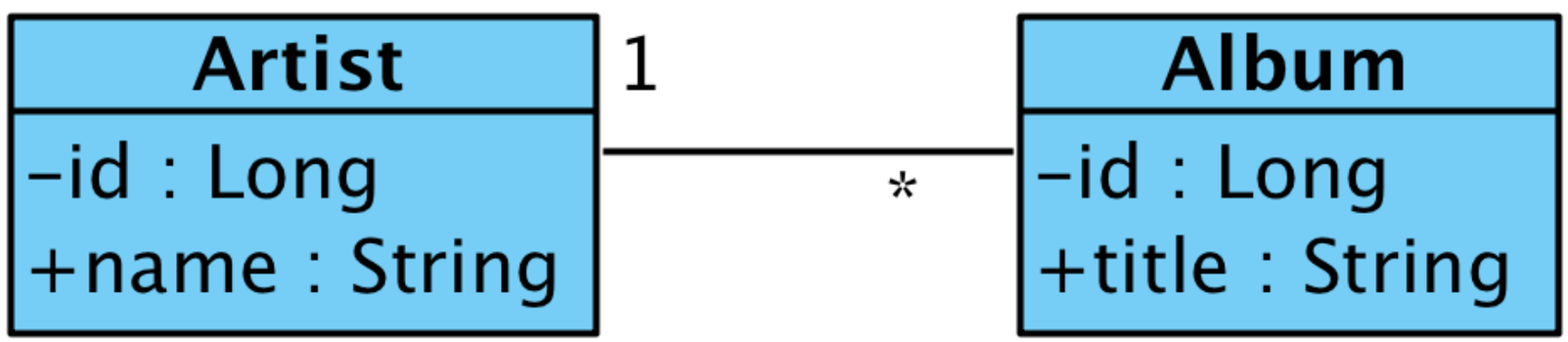
id	title
1	A Thousand Suns
2	Minutes to Midnight
...	

artists_to_albums

artist_id	album_id
1	1
1	2
...	

[PoEAA]

Solution #2: **Foreign Key** [1-N]



artists

id	name
1	Linkin Park
...	

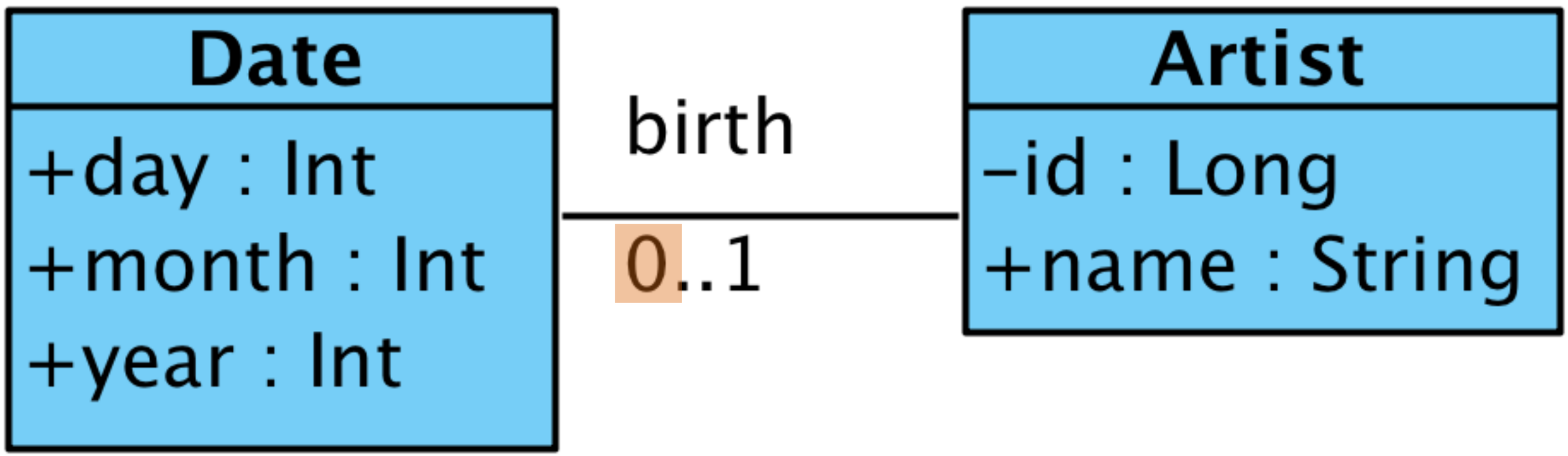
albums

id	title	artist_id
1	A Thousand Suns	1
2	Minutes to Midnight	1
...		

or **[1-N]** ≡ **[M-N]** when N =1

[PoEAA]

Solution #3: **Relation Merge** [1-1]



artists

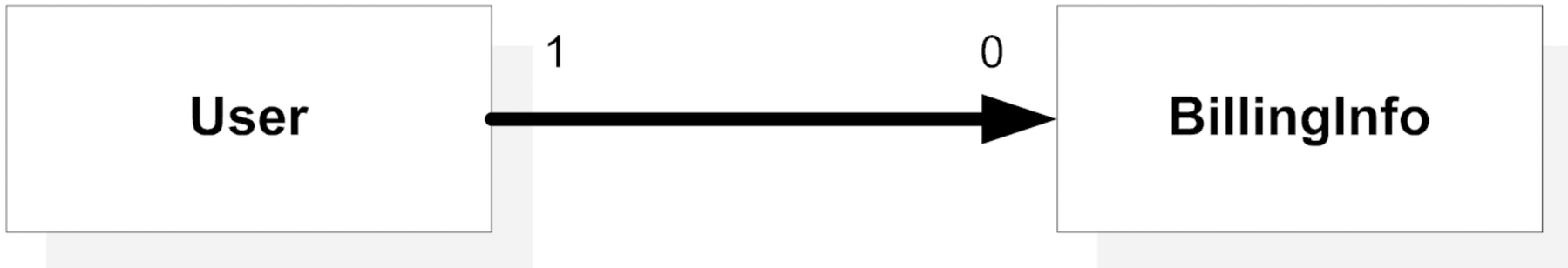
id	name	birth_day	birth_month	birth_year
1	Linkin Park	-1	-1	-1
...				

or **[1-1]** ≡ **[1-N]** when N =1

[PoEAA] or **[1-1]** ≡ **[M-N]** when M = 1 and N =1

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]

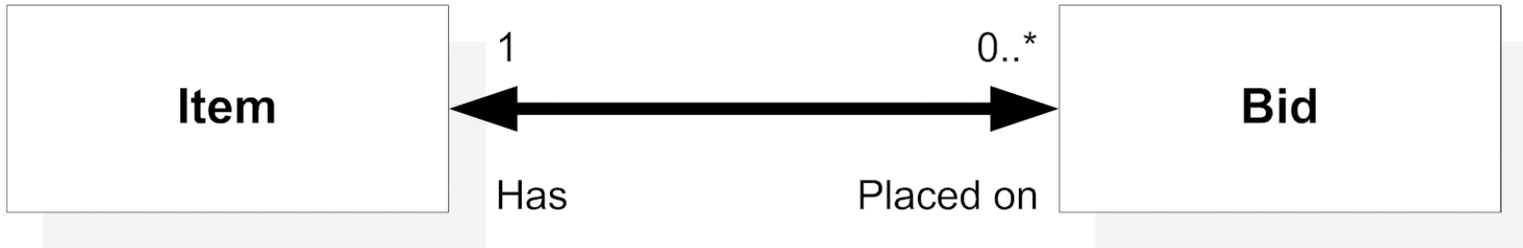
For property-based beans, annotate the getter.

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]

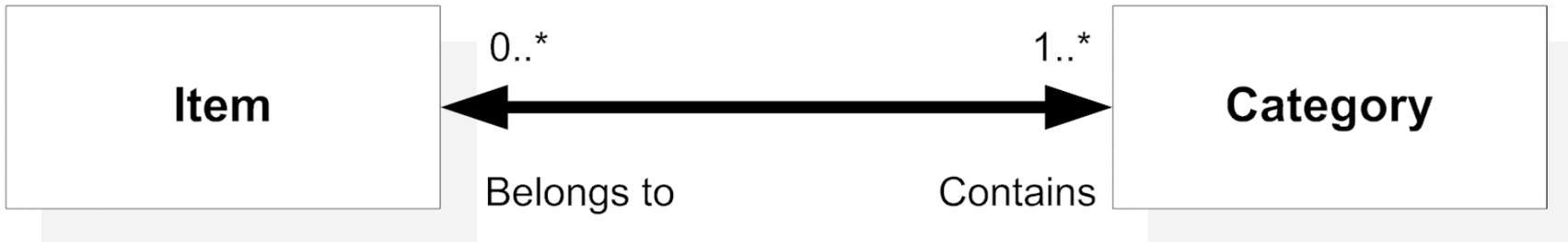



```
@Entity
public class Bid {
   @Id
   protected String bidId;
   @ManytoOne
   protected Item item;
}

@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;
}
```

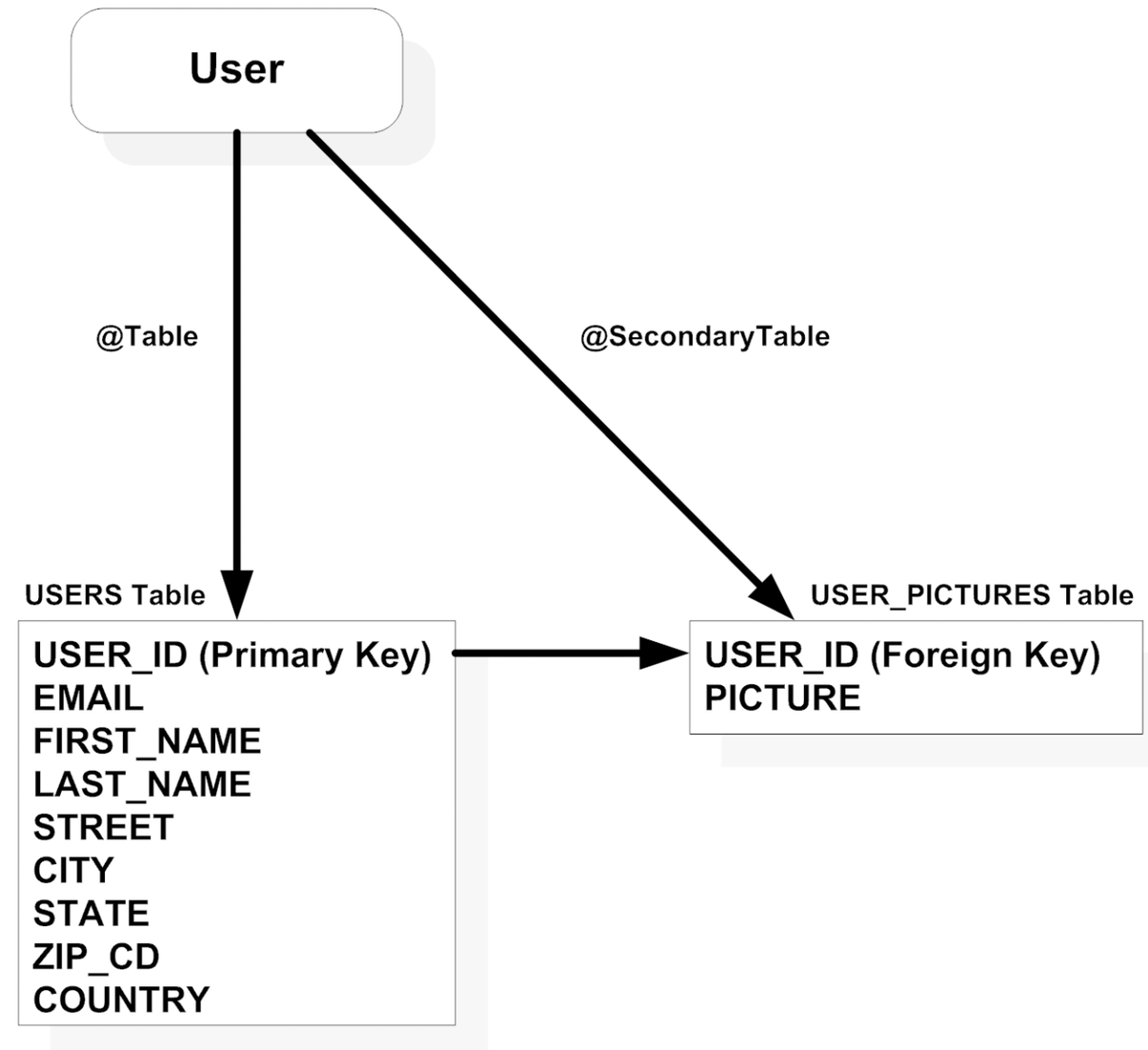
Owner

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

n-m mapping

[EiA]

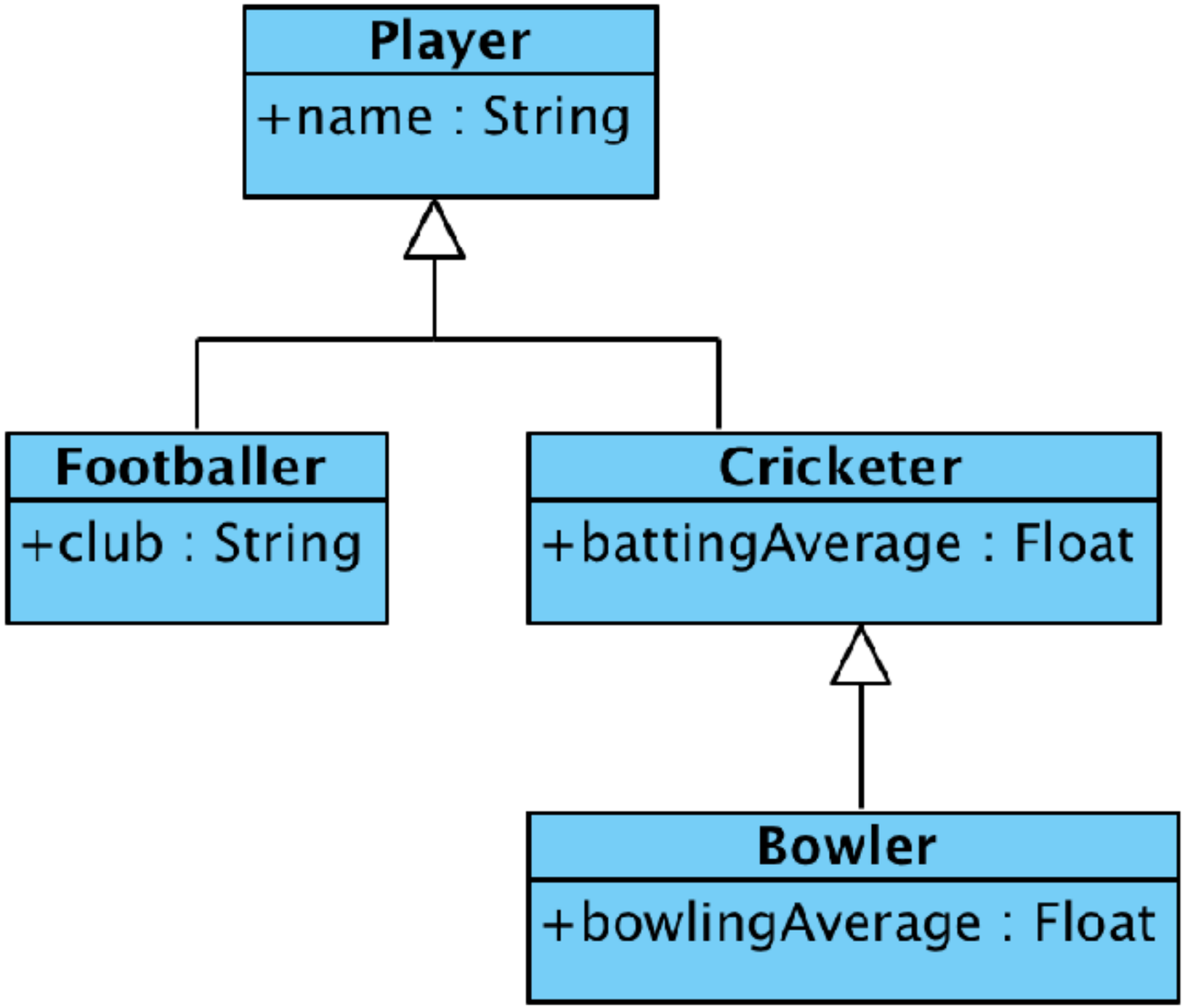
Controlling the Object-Relational mapping




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

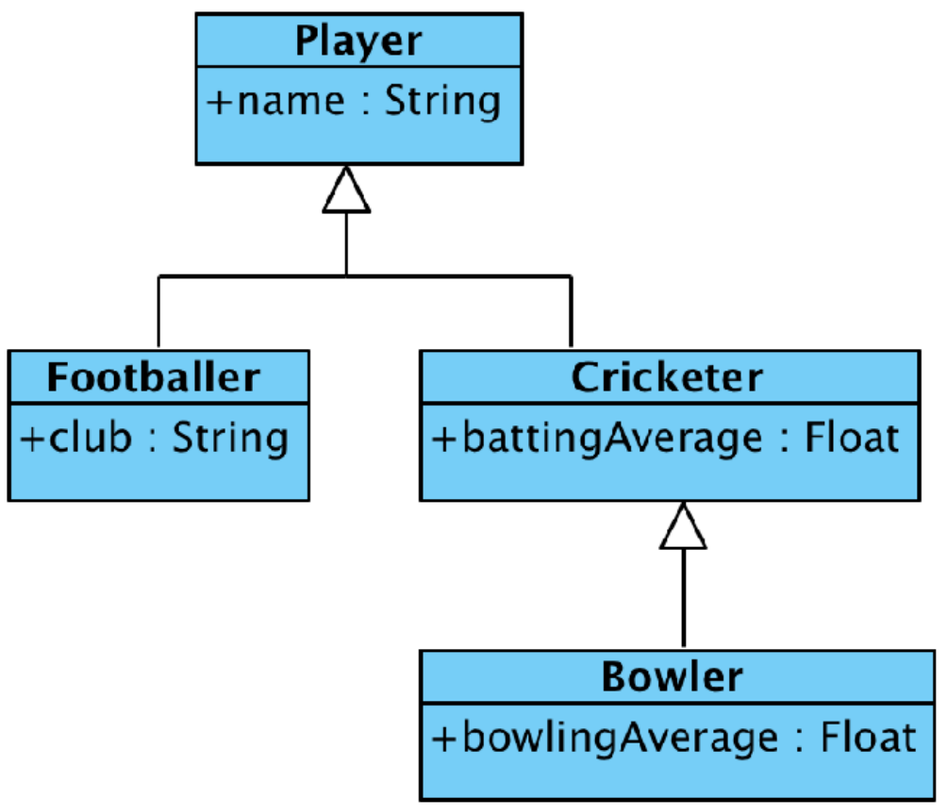
[EiA]

Problem: Implementing Inheritance



[PoEAA]

Solution #1: **Single-Table** Inheritance

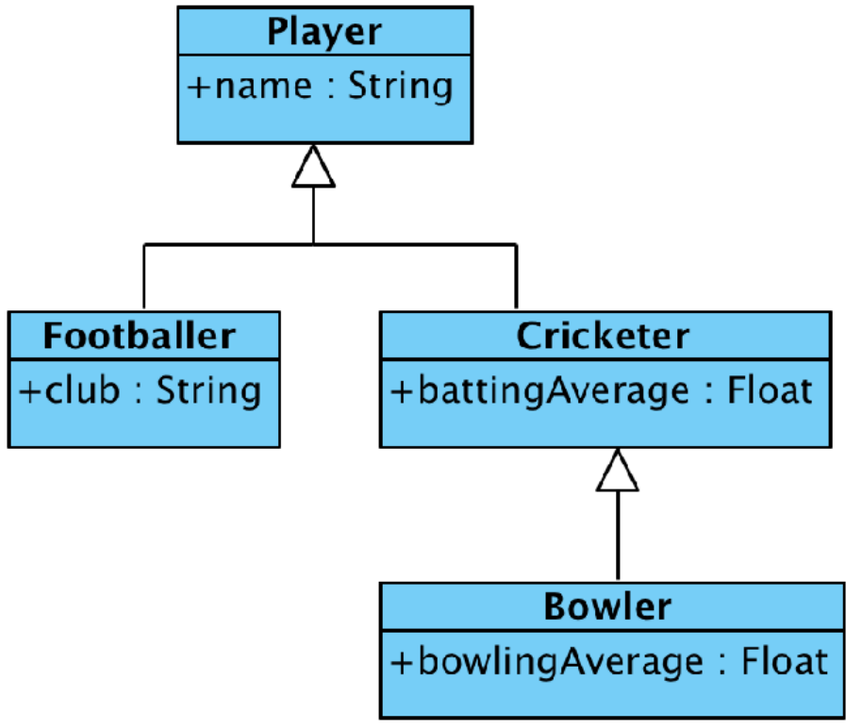


players

name	club	batting_avg	bowling_avg	type

[PoEAA]

Solution #2: **Class-Table** Inheritance



players

id	name
42	...
74	...
96	...

footballers

id	club
42	...

cricketers

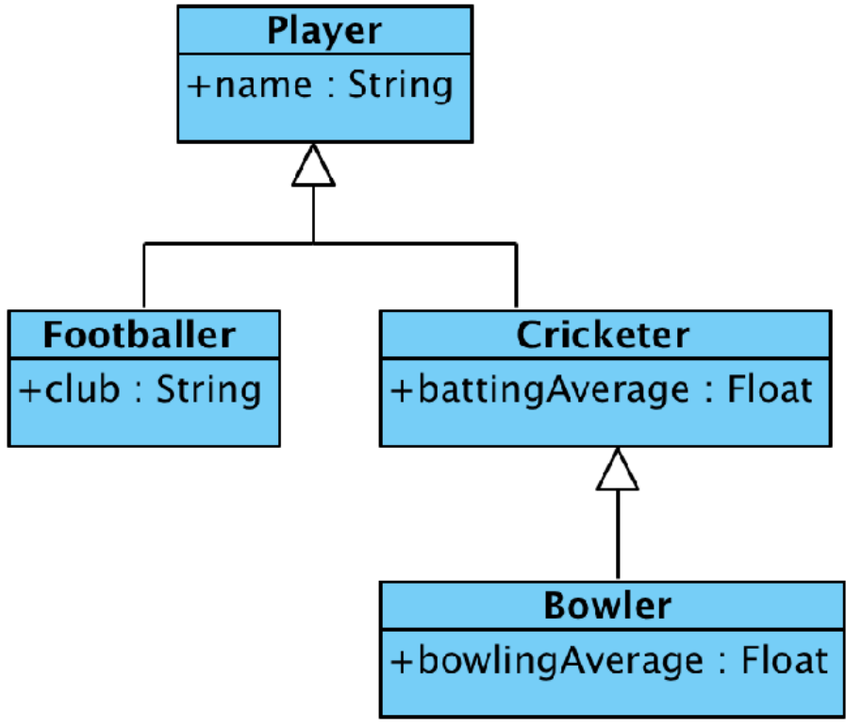
id	batting_avg
74	...
96	...

bowlers

id	bowling_avg
96	...

[PoEAA]

Solution #3: **Concrete-Table** Inheritance



footballers

id	name	club
42	...	...

cricketers

id	name	batting_avg
74	...	...

bowlers

id	name	batting_avg	batting_avg
96	...	...	

[PoEAA]

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 9

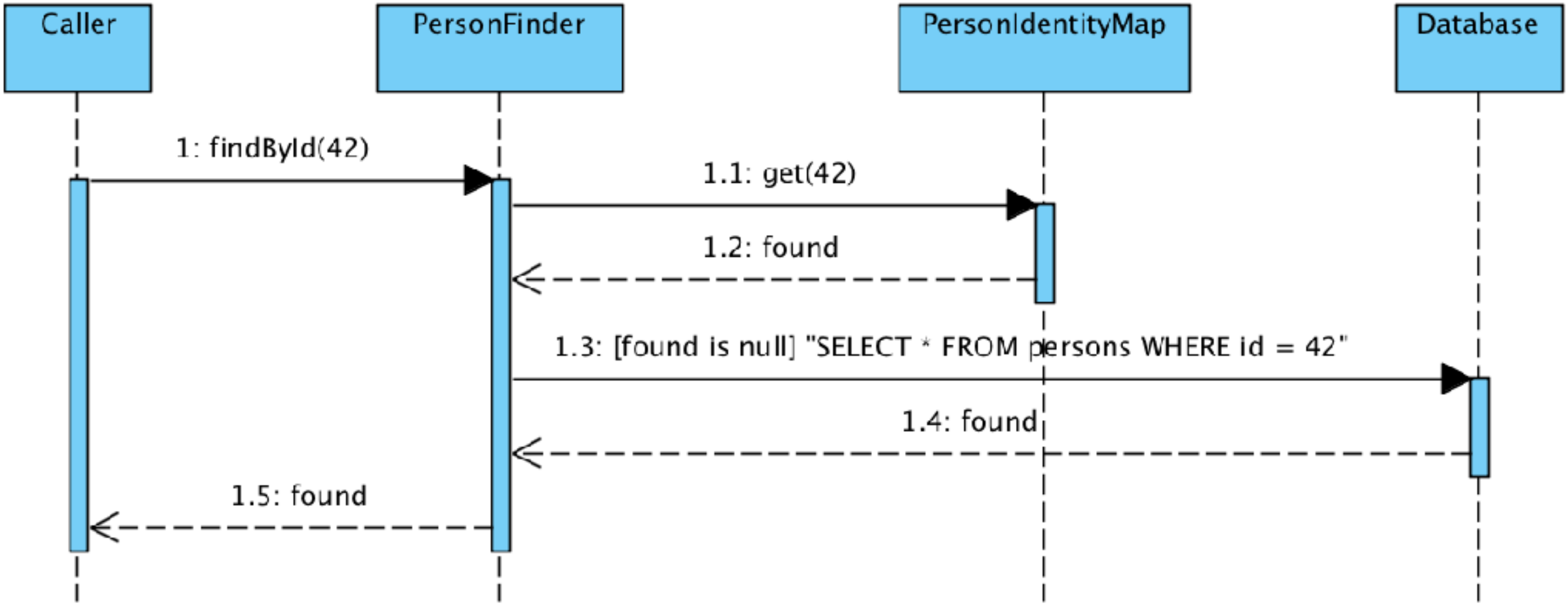
[EiA]

Make your beans **persistent**

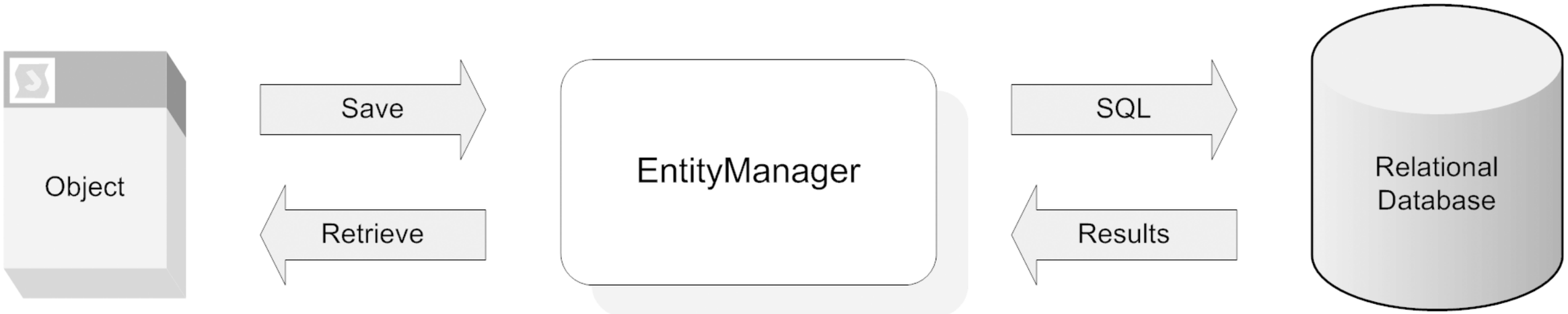
Again...

How to accelerate the access to the persistent layer?

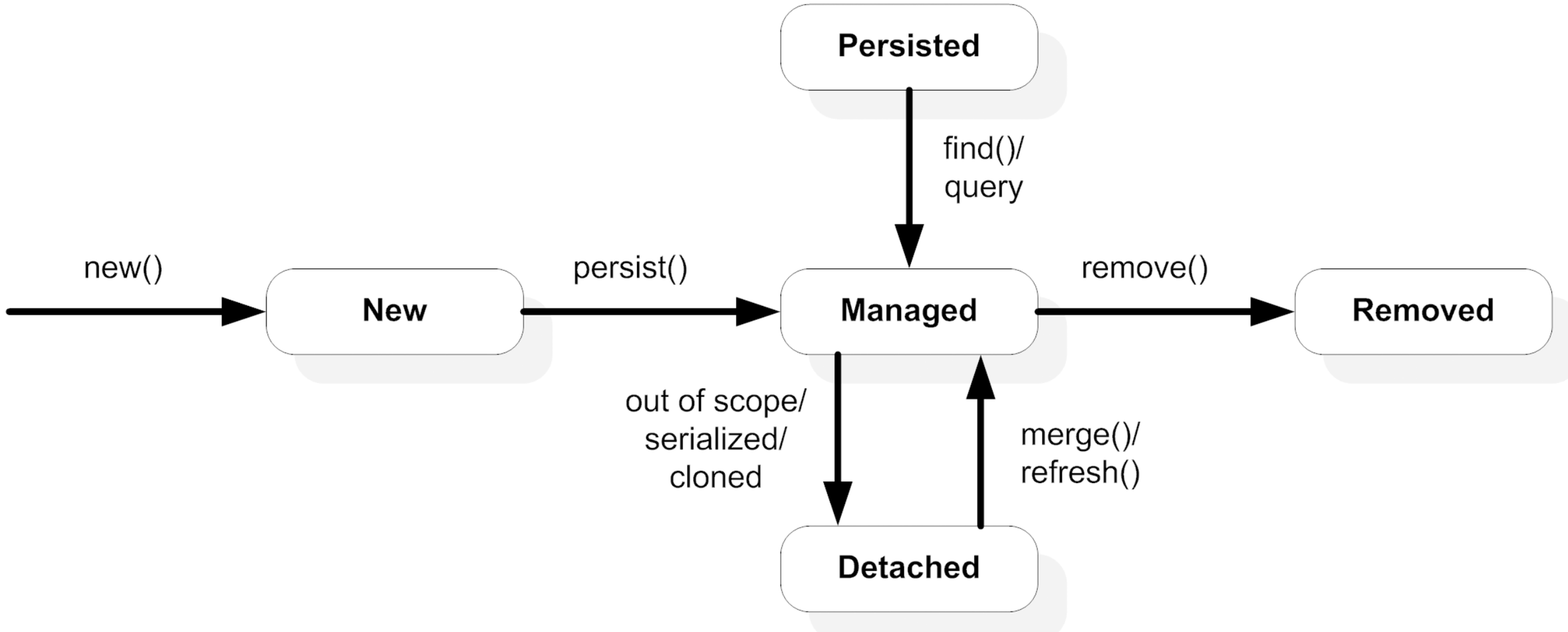
The Identity Map Pattern



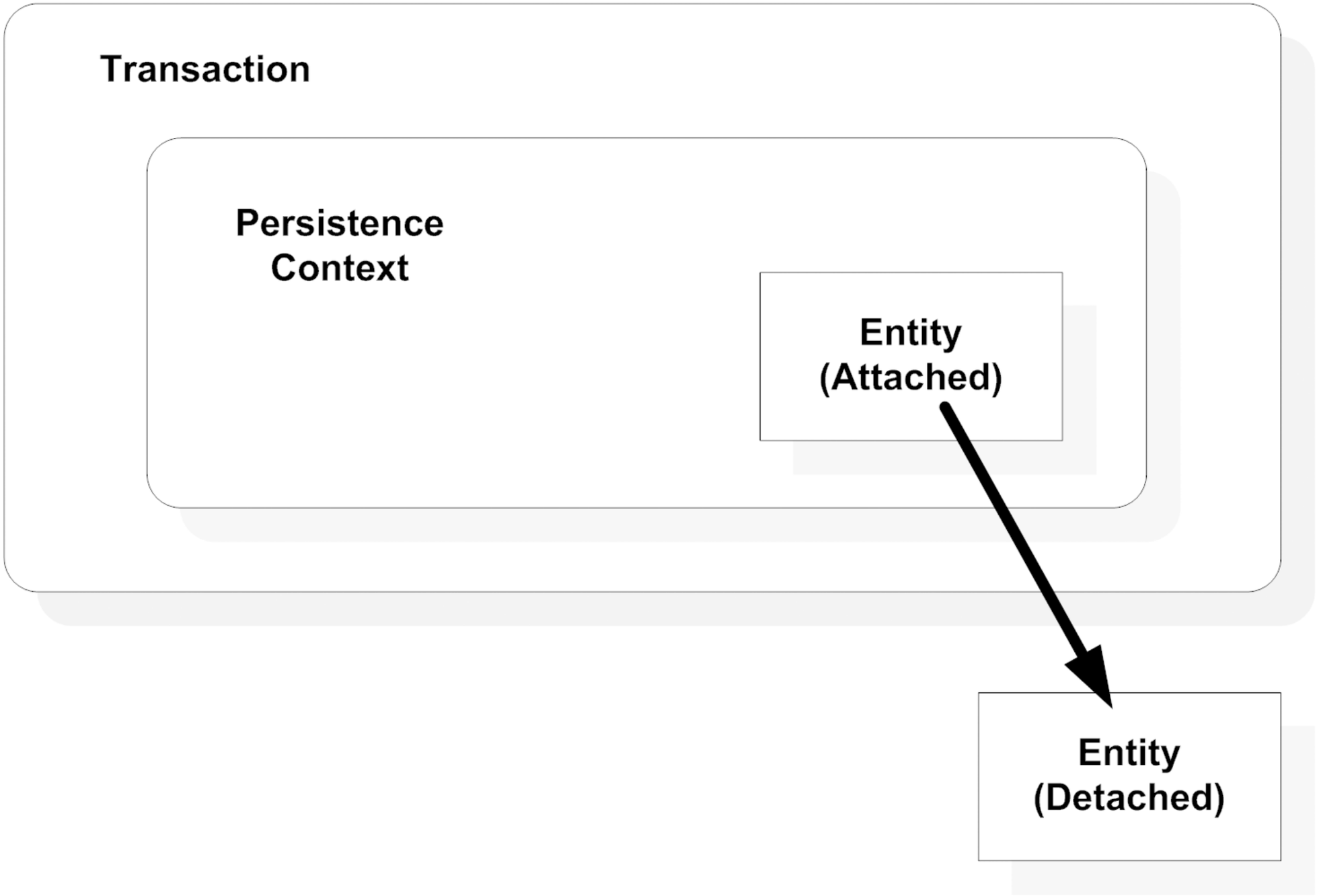
[PoEAA]


Object
Save
Retrieve
EntityManager
SQL
Results
Relational
Database

[EiA]


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

[EiA]


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 10

Advanced concepts & tricks...

Stop!

https://github.com/collet/4A_ISA_TheCookieFactory/blob/develop/chapters/Persistence.md

First

Contents

Set up: Persistence Unit (general)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

    <persistence-unit name="tcf_persistence_unit" transaction-type="JTA">
        <jta-data-source>TCFDataSource</jta-data-source>

        <class>fr.unice.polytech.isa.tcf.entities.Customer</class>
        <class>fr.unice.polytech.isa.tcf.entities.Item</class>
        <class>fr.unice.polytech.isa.tcf.entities.Order</class>

        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
            <property name="openjpa.jdbc.SynchronizeMappings" value="buildSchema(ForeignKeys=true)"/>
        </properties>
    </persistence-unit>

</persistence>
```

DataSource

Setup: 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

Set up: Bytecode enhancement (OpenJPA specific)

```
<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

Annotations: 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

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

```
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

Cascading

There is a containment relationship between Customers and Orders.
Deleting a Customer should delete the associated Orders transitively

```
public class Customer {

        // ...

        @OneToMany(cascade = {CascadeType.REMOVE}, mappedBy = "customer")
        private Set<Order> orders = new HashSet<>();

        // ...
}
```

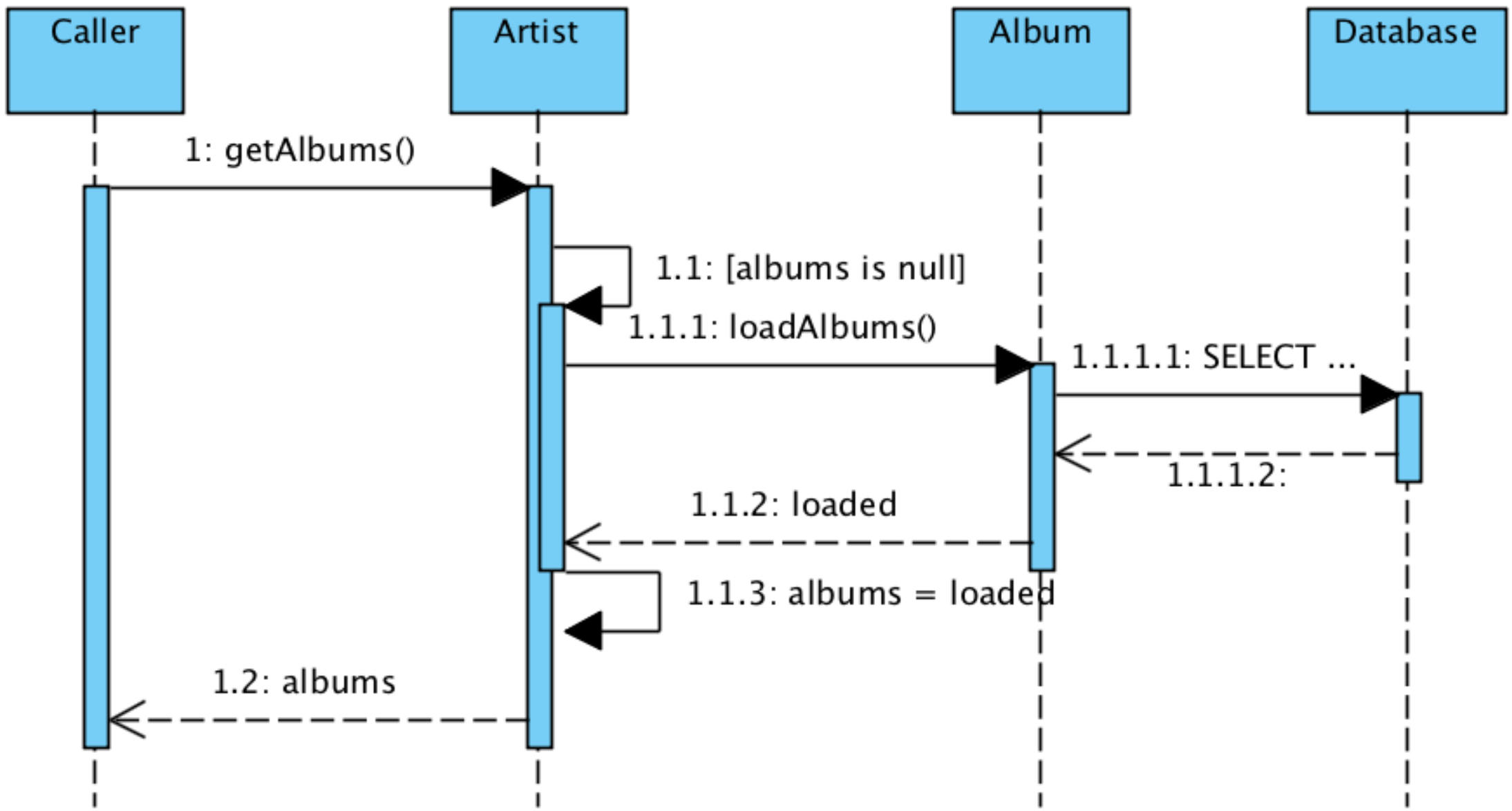
Warning: be very careful when cascading operations, especially the REMOVE one.
Deleting an Order should not delete the associated customer...

JPA Cascading Types

- **CascadeType.PERSIST** : cascade type persist means that save() or persist() operations cascade to related entities.
- **CascadeType.MERGE** : cascade type merge means that related entities are merged when the owning entity is merged.
- **CascadeType.REFRESH** : cascade type refresh does the same thing for the refresh() operation.
- **CascadeType.REMOVE** : cascade type remove removes all related entities association with this setting when the owning entity is deleted.
- **CascadeType.DETACH** : cascade type detach detaches all related entities if a “manual detach” occurs.
- **CascadeType.ALL** : cascade type all is shorthand for all of the above cascade operations.

There is no default cascade type in JPA. By default no operations are cascaded.

Lazy Loading



[PoEAA]

Lazy loading of the the orders associated to a given customer

```
@OneToMany(cascade = {CascadeType.REMOVE}, fetch = FetchType.LAZY, mappedBy = "customer")
private Set<Order> orders = new HashSet<>();
```

```
@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);
```

OUTSIDE THE TRANSACTION (DETACHMENT...)