Spring + JPA + Hibernate

Lennard Fuller

Jasig Conference, 2009

UNICON

1. A brief overview of JPA
2. Native Hibernate vs JPA Vocabulary
3. Improper Interception
4. Inefficient Retrieval
5. equals() & hashcode()
6. QL Injection
7. Caching Concerns
8. Questions?

What is JPA?

JPA: Java Persistence API

- Unified POJO Persistence into a standard API
- Part of EJB 3.0 specification, but is separately documented
- JPA 1.0 finalized in May 2006
  - Released as part of Java EE 5 platform
- Usable within Java EE or J2SE platform

Core JPA Features

- POJO-based persistence model
  - Simple Java class files — not components
- Supports traditional O-O modelling concepts
  - Inheritance, polymorphism, encapsulation, etc.
- Standard abstract relational query language
- Standard O/R mapping metadata
  - Using annotations and/or XML
- Portability across providers (implementations)

Hibernate Native & JPA Brief Comparison

Hibernate Native & JPA Compared

- Functionally very similar.
- Vocabulary quite different
- Pitfalls for those new to technology also very similar

Native Hibernate	JPA
Session	Entity Manager
	Persistence Context
Session Factory	Entity Manager Factory
POJO	Entity

Improper Interception

Simple Example

Simple Example

Foo.java

```
@Entity
public class Foo {
 …
   @OneToMany(mappedBy=“foo”)
   public Set getBars() { return bars; }
…
}
```

Bar.java

```
@Entity
public class Bar {
 …
   @ManyToOne @JoinColumn(name=“FOO_ID", nullable=false)
    public    Foo getFoo() { return foo; }
…
}
```

Simple Example: DAO

```
@Transactional
public class FooDAO implements IFooDAO{
   private EntityManager entityManager;

   @PersistenceContext
    public void setEntityManager(EntityManager entityManager) {
       this.entityManager = entityManager;
    }

    public List query(String queryString, final Object... params) {
       …
    }

    public void removeFoo(Foo foo) {
                this.entityManager.remove(foo);
    }
}
```

Simple Example: DAO cont.

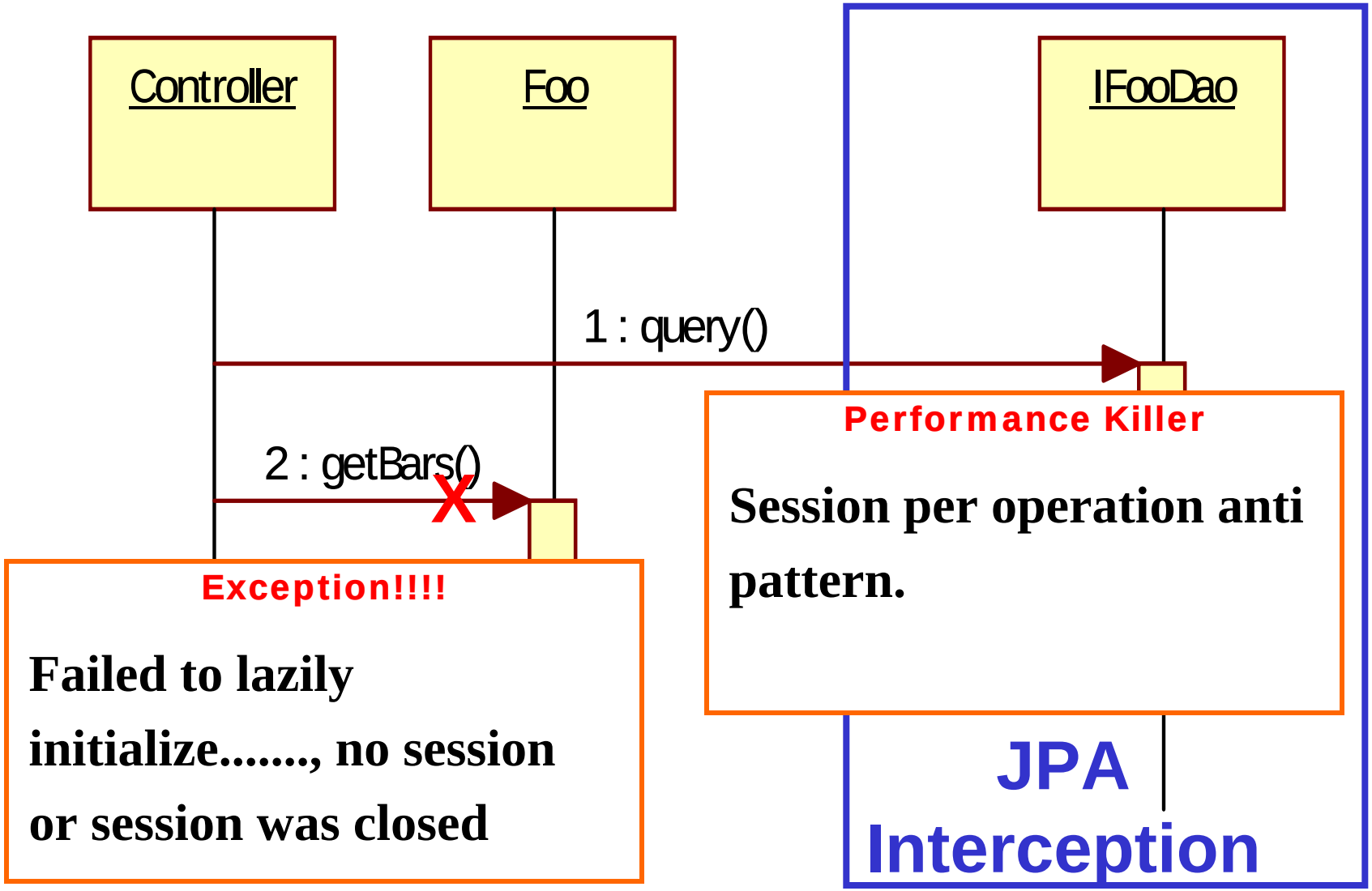
Interface for FooDAO

```
public interface IFooDAO{

    public List query(String queryString, final Object... params) ;
}
```

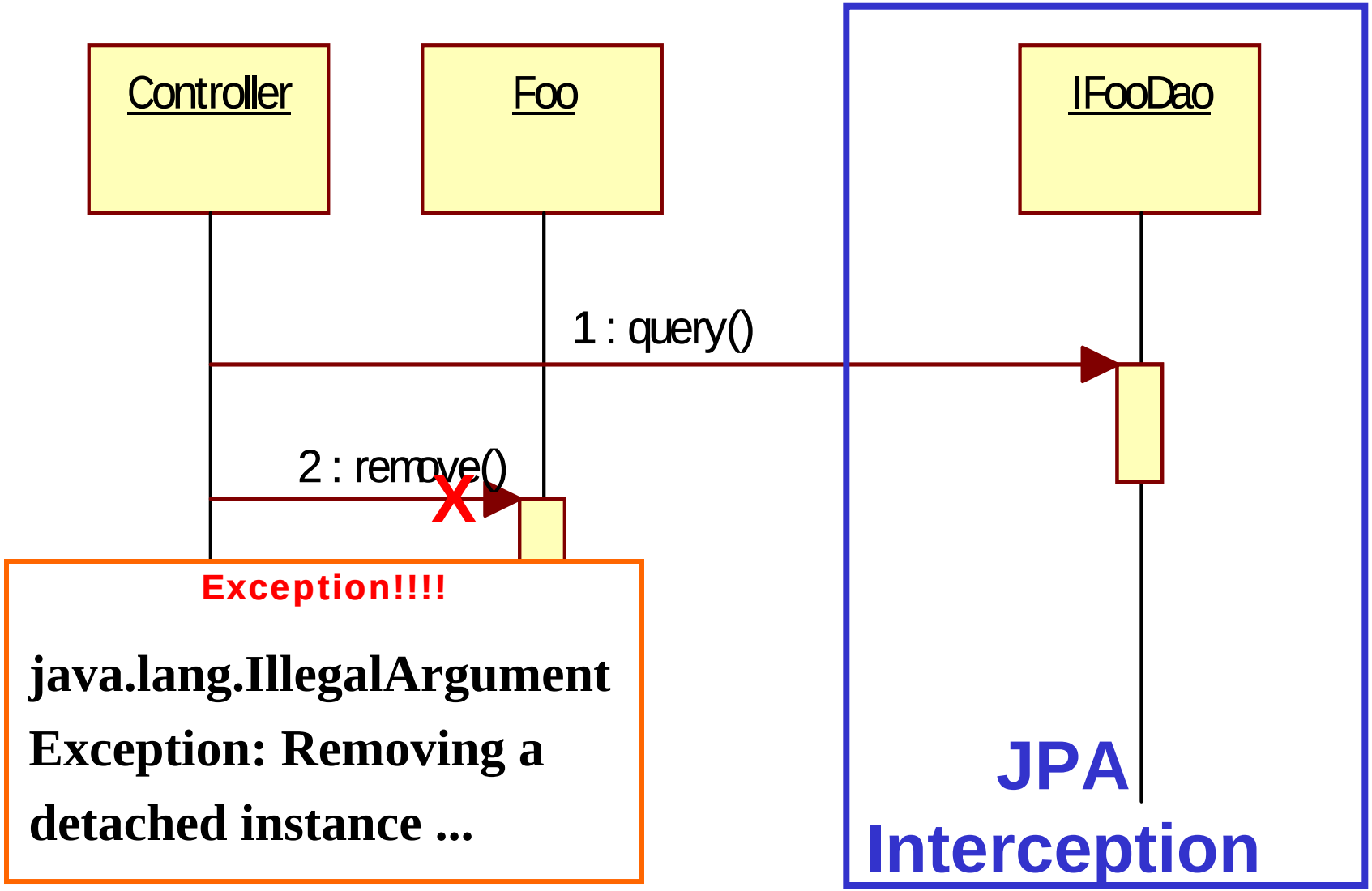
persistenceContext.xml

```
…
  <tx:annotation-driven />
  <bean

class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanP
ostProcessor" />
…
```

Simple Example



Remove Example



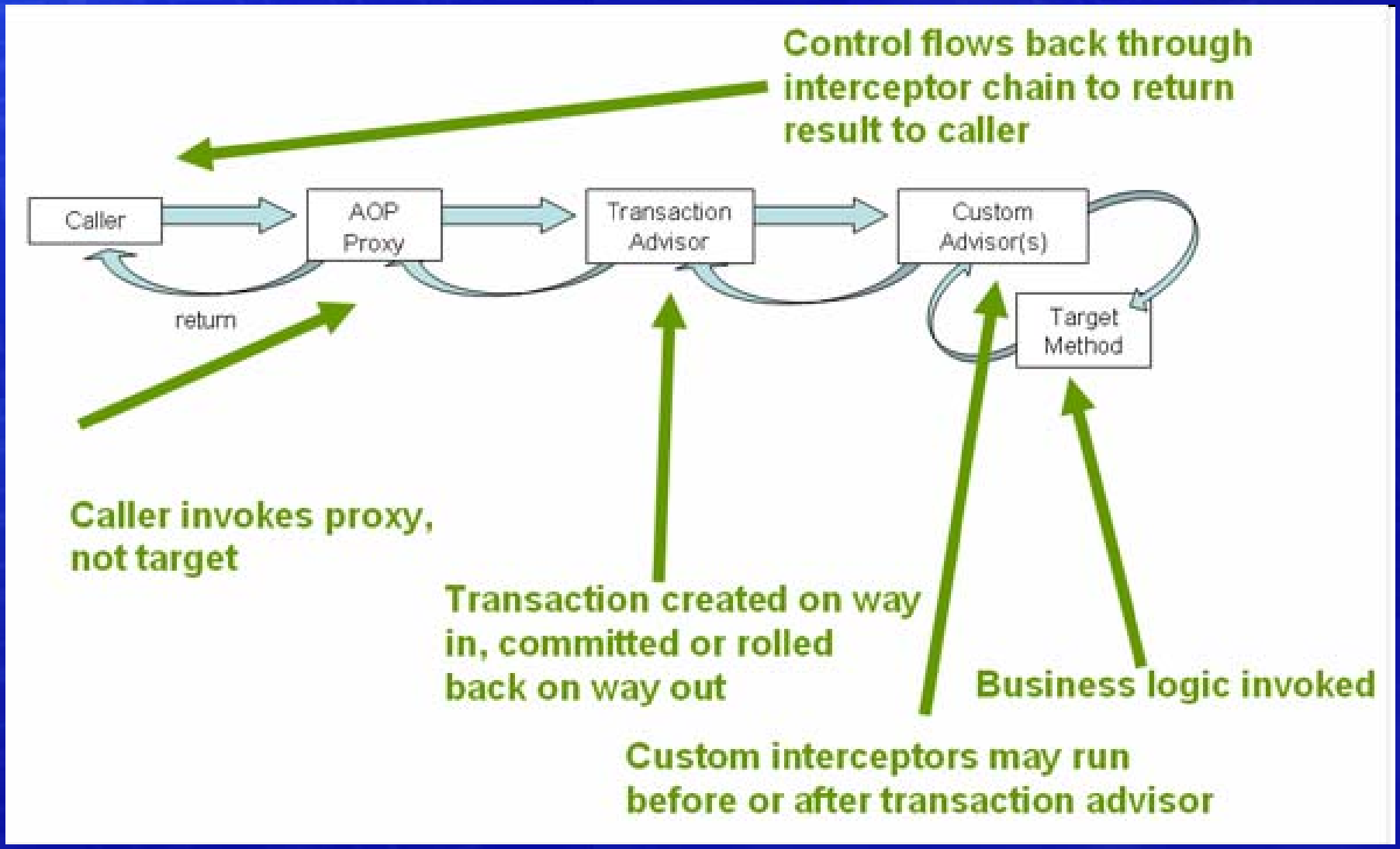
Unecessary Merge

```
@Transactional
public class FooDAO implements IFooDAO{
  private EntityManager entityManager;

   @PersistenceContext
   public void setEntityManager(EntityManager entityManager) {
      this.entit
   }

   public List                                          .. params) {
      …
   }

   public void
            this.entityManager.remove(this.entityManager.merge(foo));
   }
}
```

Performance Killer!!

Forces another lookup and a deep merge with what is in DB. Not necessary

Transactional Proxies in Spring



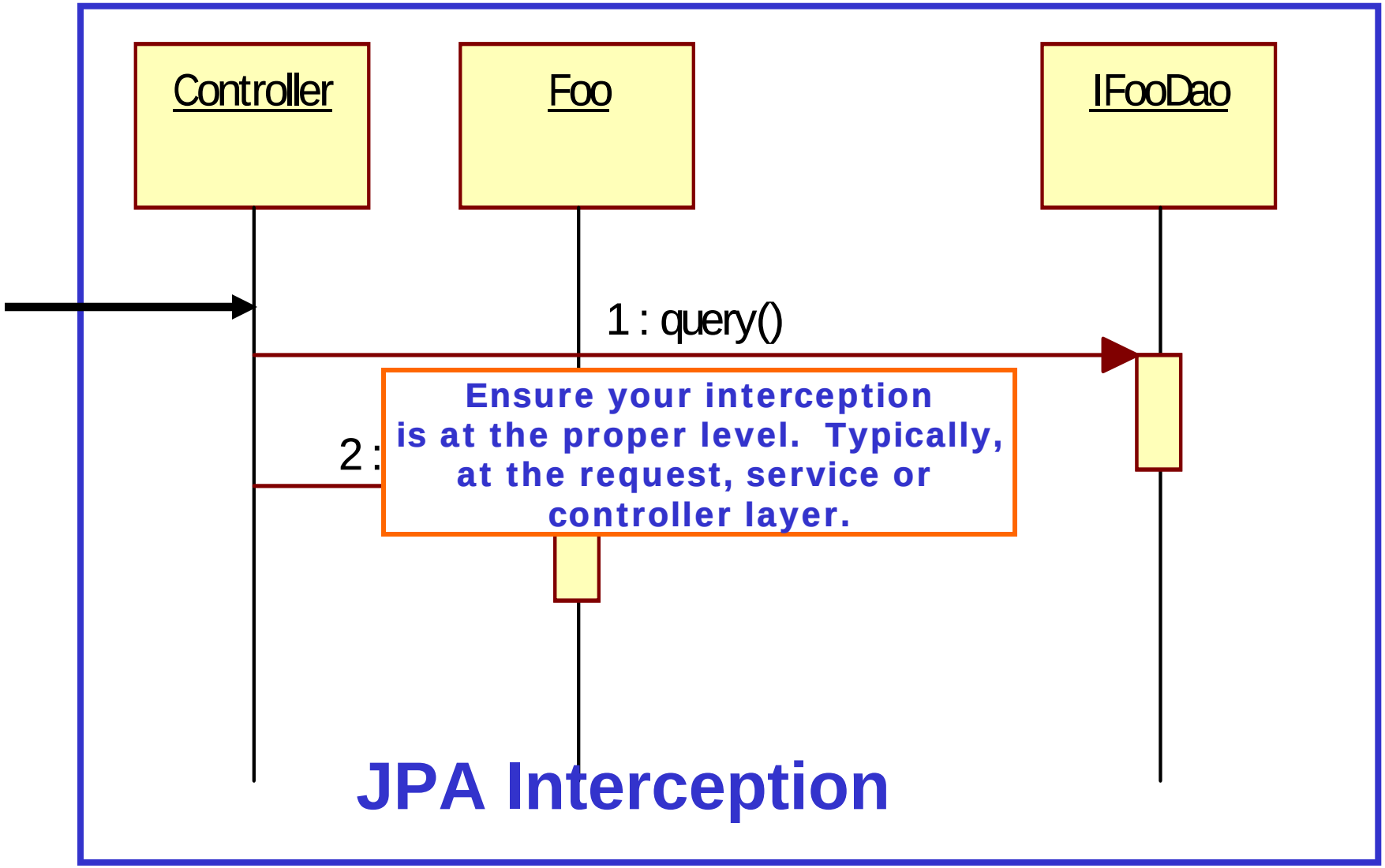
JPA Interception in Spring

- Spring makes extensive use of the Thread Local pattern.

Methods for applying JPA interception:

  - @Transactional
  - OpenEntityManagerInViewFilter
  - OpenEntityManagerInViewInterceptor
  - JpaInterceptor

Solution: Intercept at appropriate level



Means of interception

Using @Transactional

SomeService.java

```
@Transactional
public class SomeService {
        …
}
```

persistenceContext.xml

```
…
  <tx:annotation-driven />
  <bean

class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanP
ostProcessor" />
…
```

Using Filter: web.xml

```
…
   <filter>
      <filter-name>OpenEntityManagerInViewFilter</filter-name>
      <filter-
class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFil
ter</filter-class>
   </filter>
…
   <filter-mapping>
      <filter-name>OpenEntityManagerInViewFilter</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
…
```

JPA Interceptor: persistenceContext.xml

```
…
   <bean id="jpaInterceptor"
class="org.springframework.orm.jpa.JpaInterceptor">
      <property name="entityManagerFactory"
ref="entityManagerFactory" />
   </bean>
…
```

JPA Interceptor: foo_portlet_beans.xml

```
…
   <bean id=“fooPortletBean"

class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="target">
             <ref bean=“fooPortletBeanTarget"/>
        </property>
        <property name="proxyInterfaces">
          <value>javax.portlet.Portlet</value>
        </property>
        <property name="interceptorNames">
          <list>
            <value>jpaInterceptor</value>
          </list>
        </property>
…
```

Inefficient Retrieval: N+1 and Cartesian Products

Enemies of performance

- N+1
  - Natural result of default lazy behavior
- Cartesian Product Problem
  - Only occurs in eager fetches.

How can I efficiently get those Bars?

N+1

If I use the following JPA QL to render a result for single foo associated to 1000 bars:

```
from Foo f where f.desciption like ‘%Foo%’
```

Hibernate will have to make 1001 sql queries in order for the data to be displayed IF the relationship between foo and associated bars is lazy.

```
from Foo f left join fetch f.bars where f.desciption like ‘%Foo%’
```

Adding the left join fetch overrides the default lazy relationship and allows hibernate to retrieve all the data in one sql call.

WARNING: left join fetch is NOT a silver bullet, RTM and then use.

Cartesian Product

- Opposite of N+1. Retrieves **too much** data.
- Always occurs if you retrieve “parallel” collections in an eager fashion.

- Avoidance:
  - Know your data model, only eager fetch when necessary.
  - Never ever fetch parallel collections

How will I know?

- In your dev env turn showSql on.
- **TEST** each new query and verify that that it conforms to your expectations.

```
<bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="jpaVendorAdapter">
      <bean
      class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="true" />
        <property name="generateDdl" value="true" />
        <property name="databasePlatform" value="org.hibernate.dialect.HSQLDialect" />
      </bean>
    </property>
 </bean>
```

equals() and hashCode()

equals and hashCode

- Best practice to override equals and hashCode
- Absolutely necessary if comparing detached objects
- When overriding ensure that:
  - Java equals and hashCode contract is honored.
  - Use business key (like username for user) of entity *NOT* db identifier
    - Do NOT include collections

What if I don’t?

- Scenarios involving detached entities will be risky.
- Potential issues depending on caching solution.
- Josh Bloch won’t like you.
- You may be evil.

QL injection

Not all forms of Laziness are good

- Never pass unchecked values from user into database!

```
String evilQueryString = “from user u where u.title like ‘” + search + “’”;
Query query = this.entityManager.createQuery(evilQueryString);
```

- Always use parameter binding

```
Static String happyQueryString = “from user u where u.title like :search”;
Query query = this.entityManager.createQuery(happyQueryString)
                                .setString(“search”, search);
```

Secondary Caching

Caching Pointers

- Know your cache, know its features.
  - Not all second level caches are transactional
- Understand Hibernate's 2nd level cache.
  - Query Cache
  - Entity Cache
- Protect yourself from dirty reads
- Be aware that there are many ways to cripple your cache.

How will I know?

- In your dev env turn showSql on.
- **TEST** each new query and verify that that it conforms to your expectations.
- Performance test your application, and track your results!
  - Recommend Grinder (Free, easy, powerful)
  - Compare results before and after modifications.

1. A brief overview of JPA
2. Native Hibernate vs JPA Vocabulary
3. Improper Interception
4. Inefficient Retrieval
5. equals() & hashcode()
6. QL Injection
7. Caching Concerns

Resources

- **Java Persistence with Hibernate**
- http://static.springframework.org/spring/docs/2.5.x/reference/transaction.html
- http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/support/OpenEntityManagerInViewFilter.html
- http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/support/OpenEntityManagerInViewInterceptor.html
- **http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/jpa/JpaInterceptor.html**

Questions?

Lennard Fuller

lfuller@unicon.net

www.unicon.net