Spring Framework 2.0 New Persistence Features

Thomas Risberg

springdeveloper.com

Introduction

Thomas Risberg

- Independent Consultant, springdeveloper.com
- Committer on the Spring Framework project since 2003
- Supporting the JDBC and Data Access code
- Co-author of "Professional Java Development with the Spring Framework" from Wrox

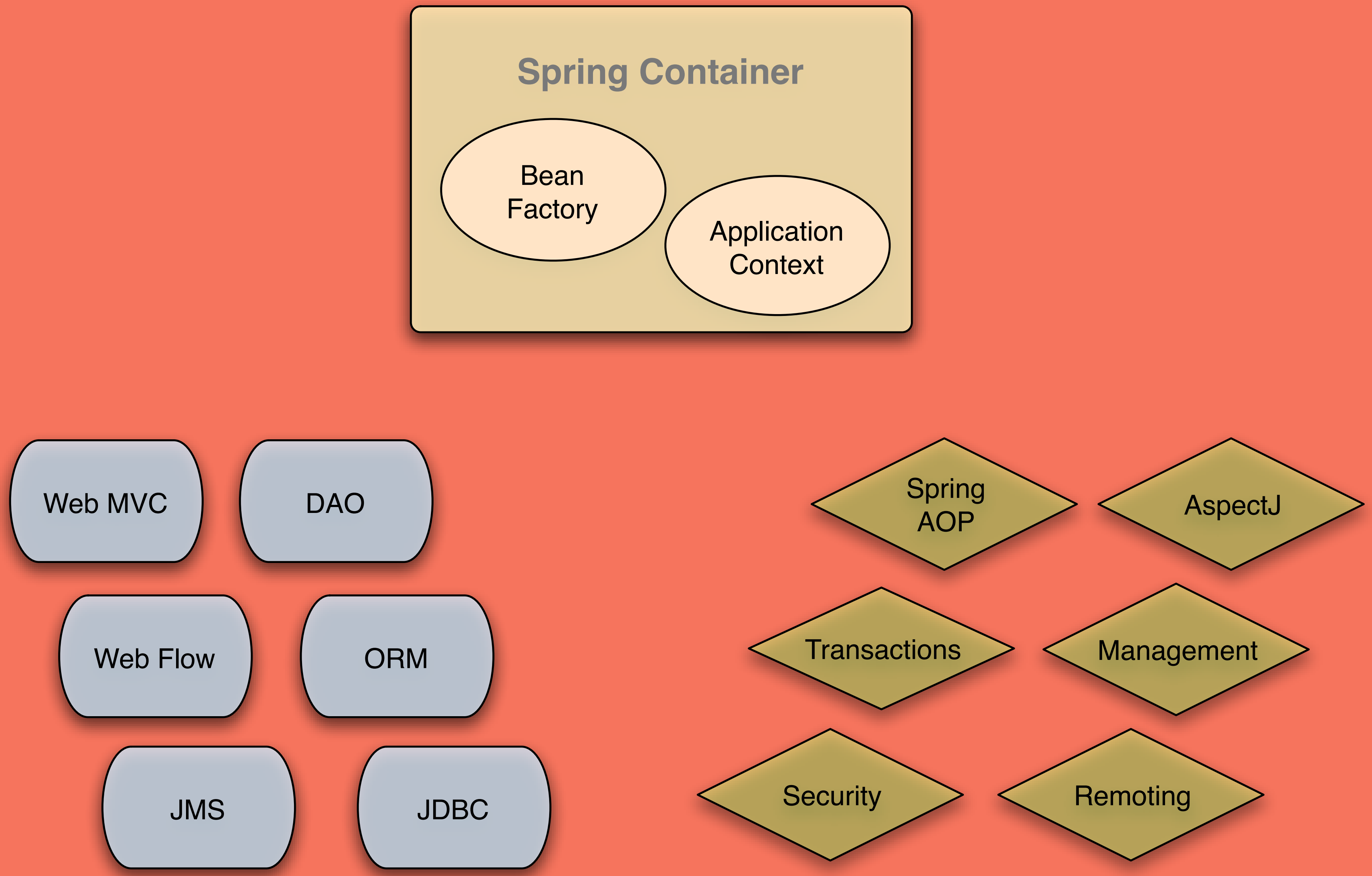
In this presentation:

- Spring Overview
- What is new in Spring 2.0?
  - Java Persistence API JSR 220
  - Spring's JDBC Abstraction Layer
- The Spring Framework Project

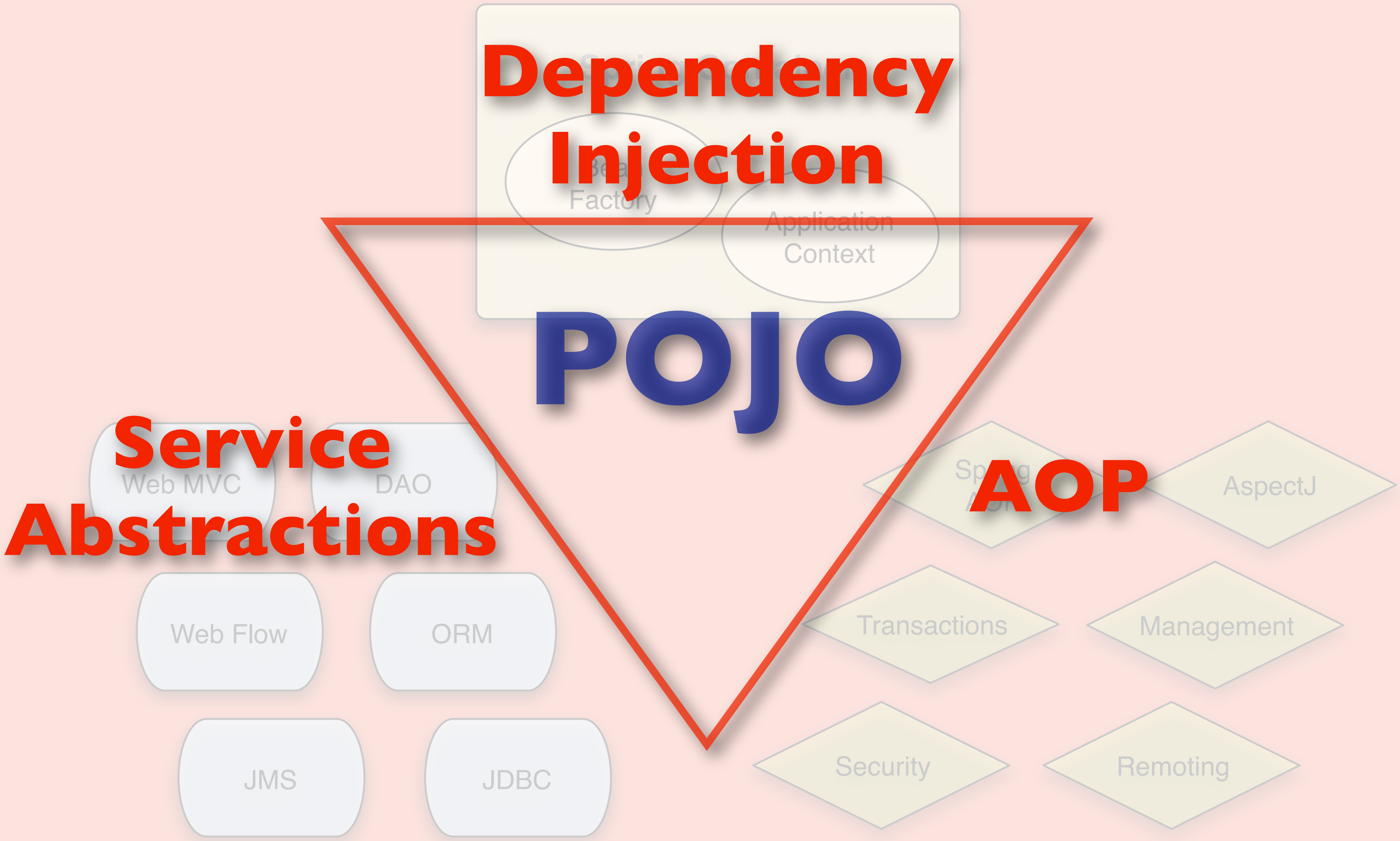
What is Spring?

- Spring is a Lightweight Application Framework covering all tiers of your typical business application
- Leverages services from underlying runtime environment (e.g. J2EE Services)
- Provides AOP services for security, transactions, management and remoting
- Integrates with other commonly used frameworks and libraries
- Greatly simplifies development effort
- Promotes modular, reusable coding practices

Features of Spring



The Spring Triangle



Why use Spring?

- Spring is not positioned to compete with J2EE or Java EE - it competes with in-house frameworks
- Many products today integrate with Spring
  - ✓ WebLogic Server
  - ✓ IntelliJ IDEA
  - ✓ ServiceMix
  - ✓ Active MQ
  - ✓ Oracle TopLink
- Next 2-3 years? Java EE 5 with EJB 3 and JSF provides part of what Spring Offers today, but most Spring users will still need the extra features provided by Spring

What is new in Spring 2.0?

- Simpler, more extensible XML configuration
- Enhanced integration with AspectJ
- Portlet MVC Framework
- Improvements in Web MVC Framework
- Additional scoping options for beans
- Ability to define beans in scripting language like Groovy or JRuby
- Message-driven POJOs

What about persistence?

- Support for Java Persistence API (EJB 3 JSR 220)
- JDBC simplifications:
  - SimpleJdbcTemplate provides support for generics, varargs and autoboxing
  - NamedParameterJdbcTemplate replaces traditional parameter placeholder with explicit parameter name
  - SqlCommand objects extends named parameter support for ease of use

Spring's Current Persistence Support

- JDBC Abstraction - provides resource management and exception translation
- Support for a growing number of O/R Mappers
  - iBATIS SQLMaps
  - Hibernate 2 and 3
  - JDO including JDO 2
  - TopLink
  - OJB
- DAO support, transaction management and exception translations for all data access choices

Java
Persistence
API
JSR 220

Works with POJOs

Standardizes:
- *ORM Metadata*
- *API*
- *Lifecycle / Callbacks*
- *Query Language*

Improves testability and removes need for DTOs

Primary API Interfaces

PersistenceContext

Transaction-scoped / Extended

EntityManager

Resource-local / JTA

Container- / Application-managed

@PersistenceContext / JNDI / emf.createEntityManager()

EntityManagerFactory

```
entityManager.getTransaction().begin();
List l = entityManager.createQuery(
  "select object(s) from ticket.domain.Show s")
  .getResultList();
entityManager.getTransaction().commit();
```

```
@PersistenceContext
...

List l = entityManager.createQuery(
  "select object(s) from ticket.domain.Show s")
  .getResultList();
```

API usage varies between JTA and Resource-local

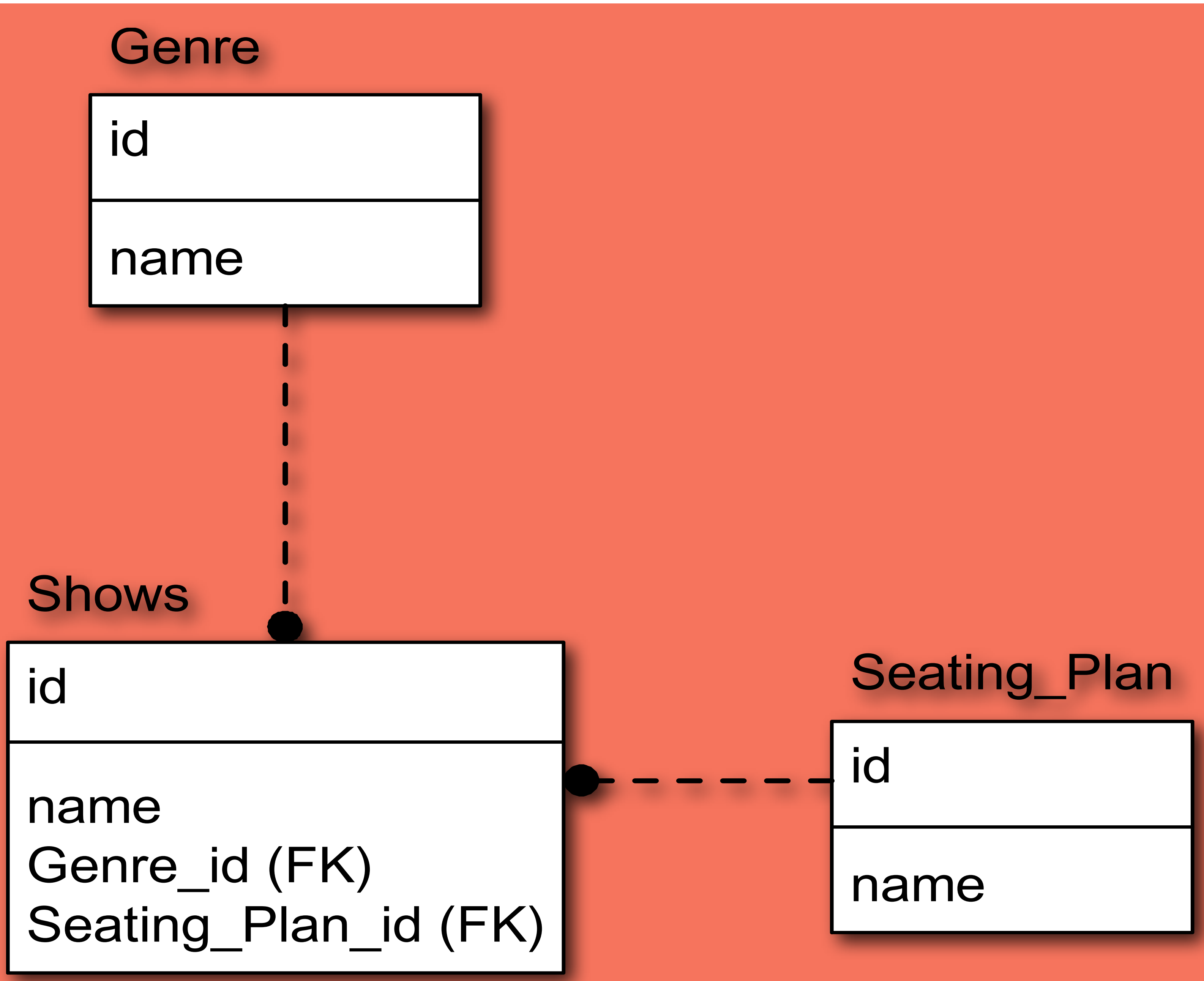
Spring will attempt to bridge these differences and provide a consistent style for programming in various environments.

This also applies when using using other O/R Mapping solutions - the same programming style is used throughout.

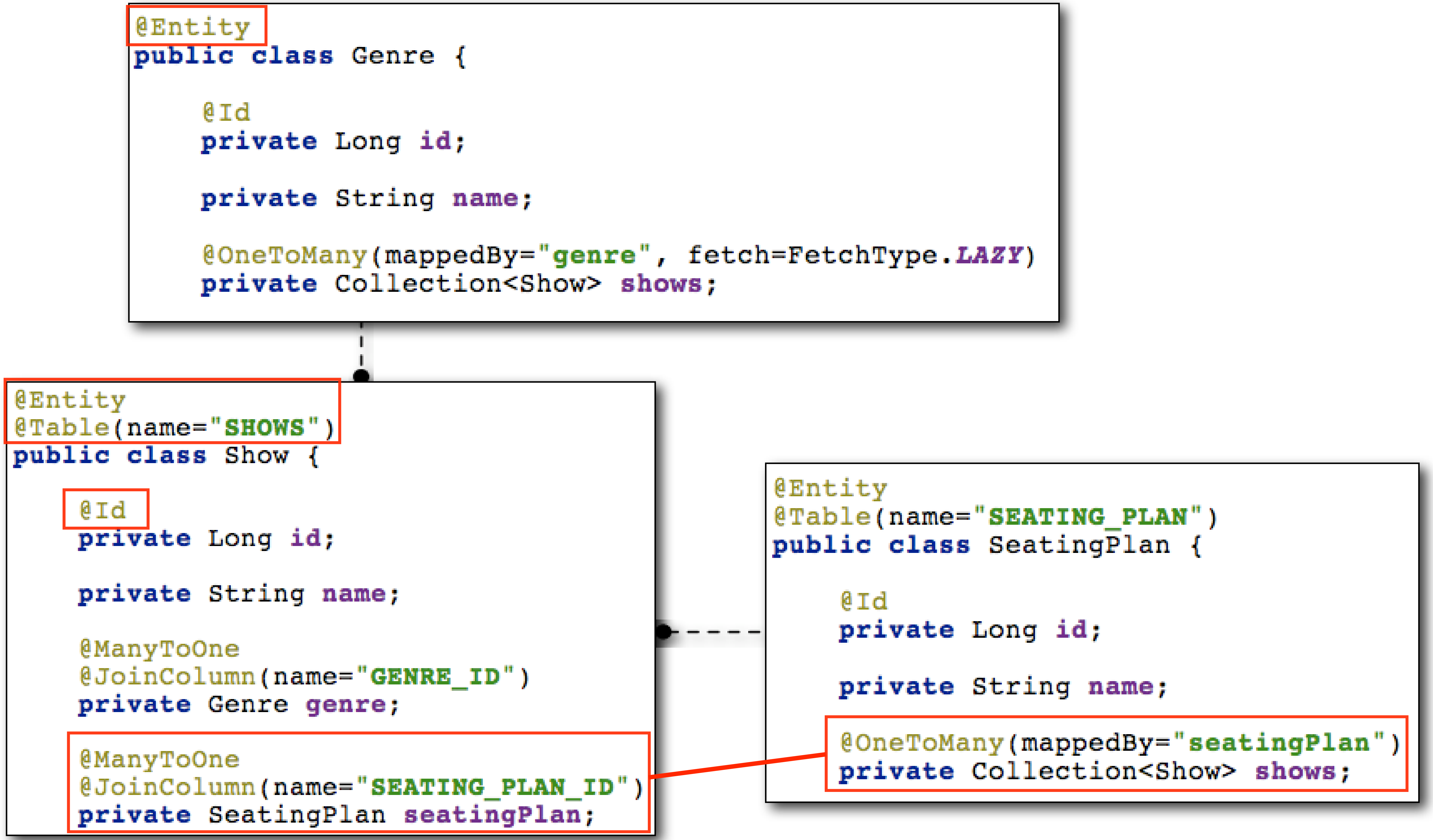
JPA Support

- In `org.springframework.orm.jpa` package
- Support classes correspond with ones for other ORM implementations like Hibernate, TopLink and JDO
- **JpaTemplate, JpaCallBack** and **JpaInterceptor** provide integration with transaction management and uses thread bound EntityManager for the persistence context
- **JpaDaoSupport** is convenience class for DAO usage
- **JpaTransactionManager** handles resource local access and JtaTransactionManager handles JTA transactions
- **LocalEntityManagerFactorybean** provides resource local bootstrapping for Java SE while JndiObjectFactoryBean does the JNDI lookups in Java EE environments

Persistence Example
Data Model



JPA Entity Mapping

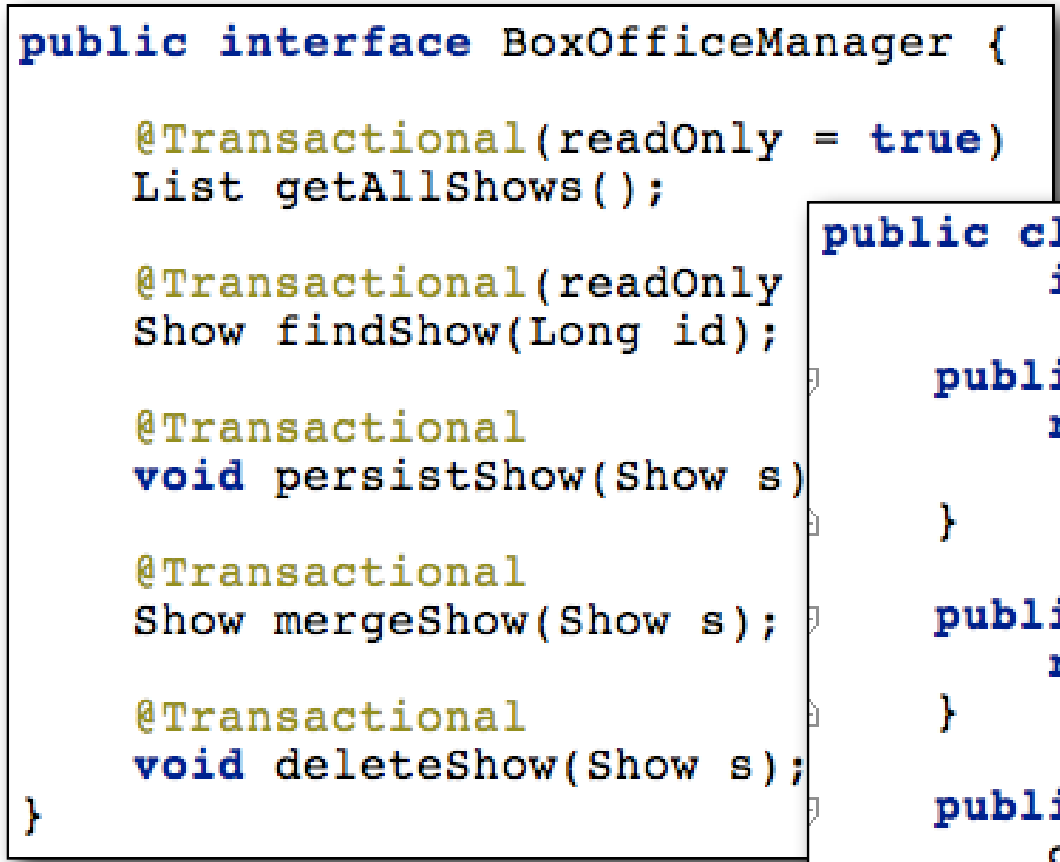


```java
@Entity
public class Genre {

    @Id
    private Long id;

    private String name;

    @OneToMany(mappedBy="genre", fetch=FetchType.LAZY)
    private Collection<Show> shows;
```

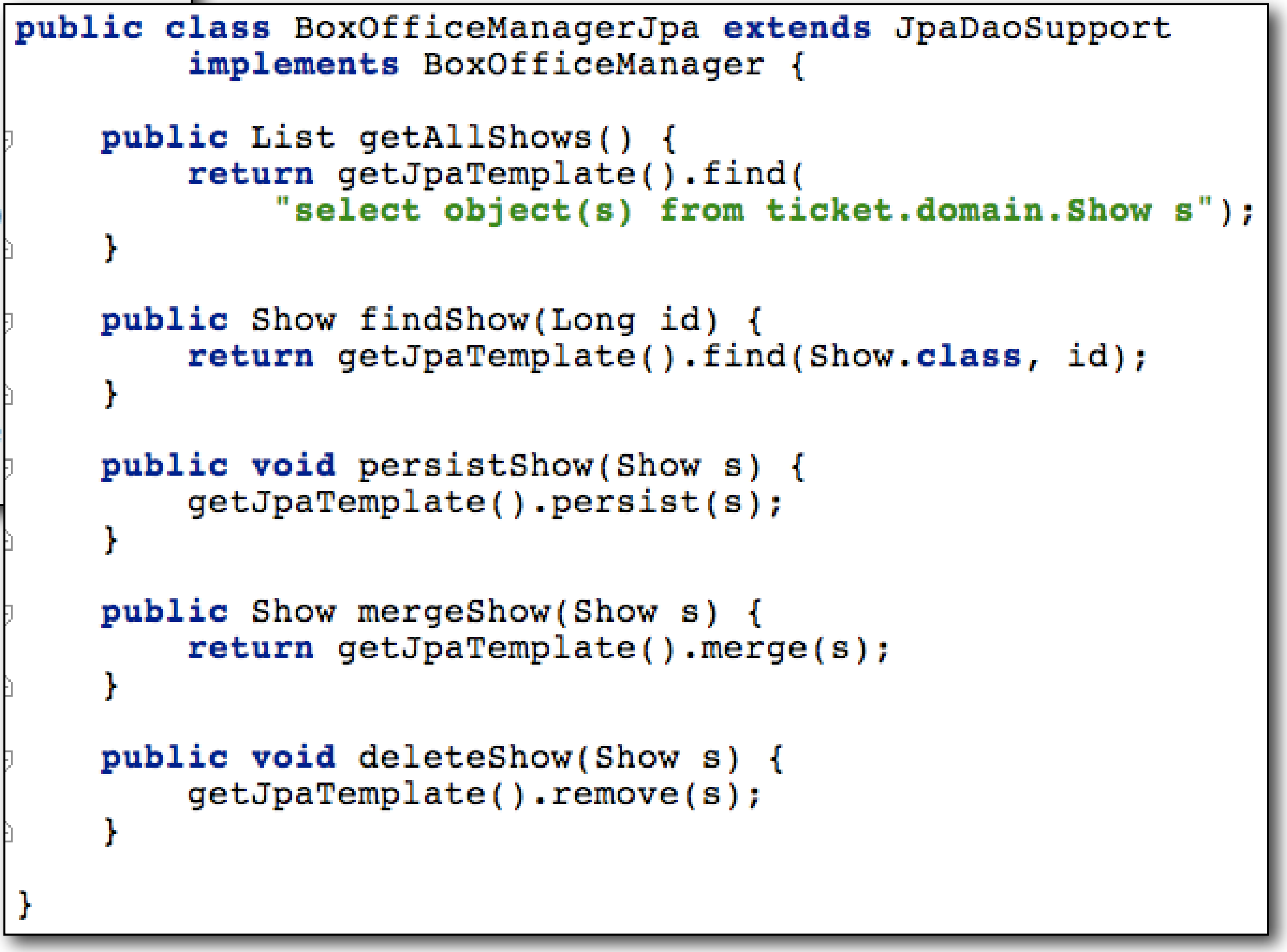
```java
@Entity
@Table(name="SHOWS")
public class Show {

    @Id
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name="GENRE_ID")
    private Genre genre;

    @ManyToOne
    @JoinColumn(name="SEATING_PLAN_ID")
    private SeatingPlan seatingPlan;
```

```java
@Entity
@Table(name="SEATING_PLAN")
public class SeatingPlan {

    @Id
    private Long id;

    private String name;

    @OneToMany(mappedBy="seatingPlan")
    private Collection<Show> shows;
```

Service/Manager Layer

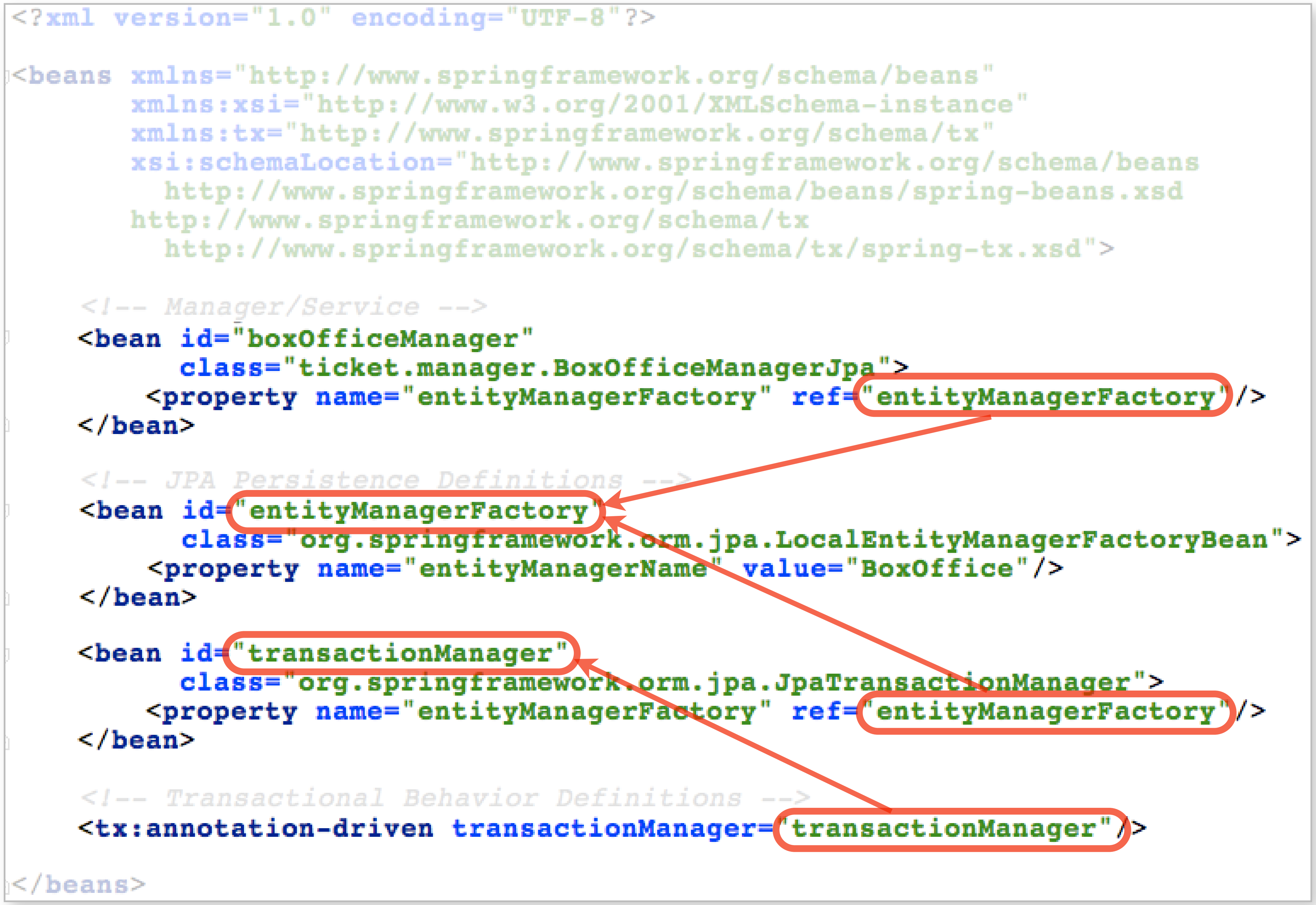


```java
public interface BoxOfficeManager {

    @Transactional(readOnly = true)
    List getAllShows();

    @Transactional(readOnly
    Show findShow(Long id);

    @Transactional
    void persistShow(Show s)

    @Transactional
    Show mergeShow(Show s);

    @Transactional
    void deleteShow(Show s);
}
```



```java
public class BoxOfficeManagerJpa extends JpaDaoSupport
        implements BoxOfficeManager {

    public List getAllShows() {
        return getJpaTemplate().find(
            "select object(s) from ticket.domain.Show s");
    }

    public Show findShow(Long id) {
        return getJpaTemplate().find(Show.class, id);
    }

    public void persistShow(Show s) {
        getJpaTemplate().persist(s);
    }

    public Show mergeShow(Show s) {
        return getJpaTemplate().merge(s);
    }

    public void deleteShow(Show s) {
        getJpaTemplate().remove(s);
    }

}
```

Application Context



```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/tx
         http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- Manager/Service -->
    <bean id="boxOfficeManager"
          class="ticket.manager.BoxOfficeManagerJpa">
        <property name="entityManagerFactory" ref="entityManagerFactory"/>
    </bean>

    <!-- JPA Persistence Definitions -->
    <bean id="entityManagerFactory"
          class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
        <property name="entityManagerName" value="BoxOffice"/>
    </bean>

    <bean id="transactionManager"
          class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory"/>
    </bean>

    <!-- Transactional Behavior Definitions -->
    <tx:annotation-driven transactionManager="transactionManager"/>

</beans>
```

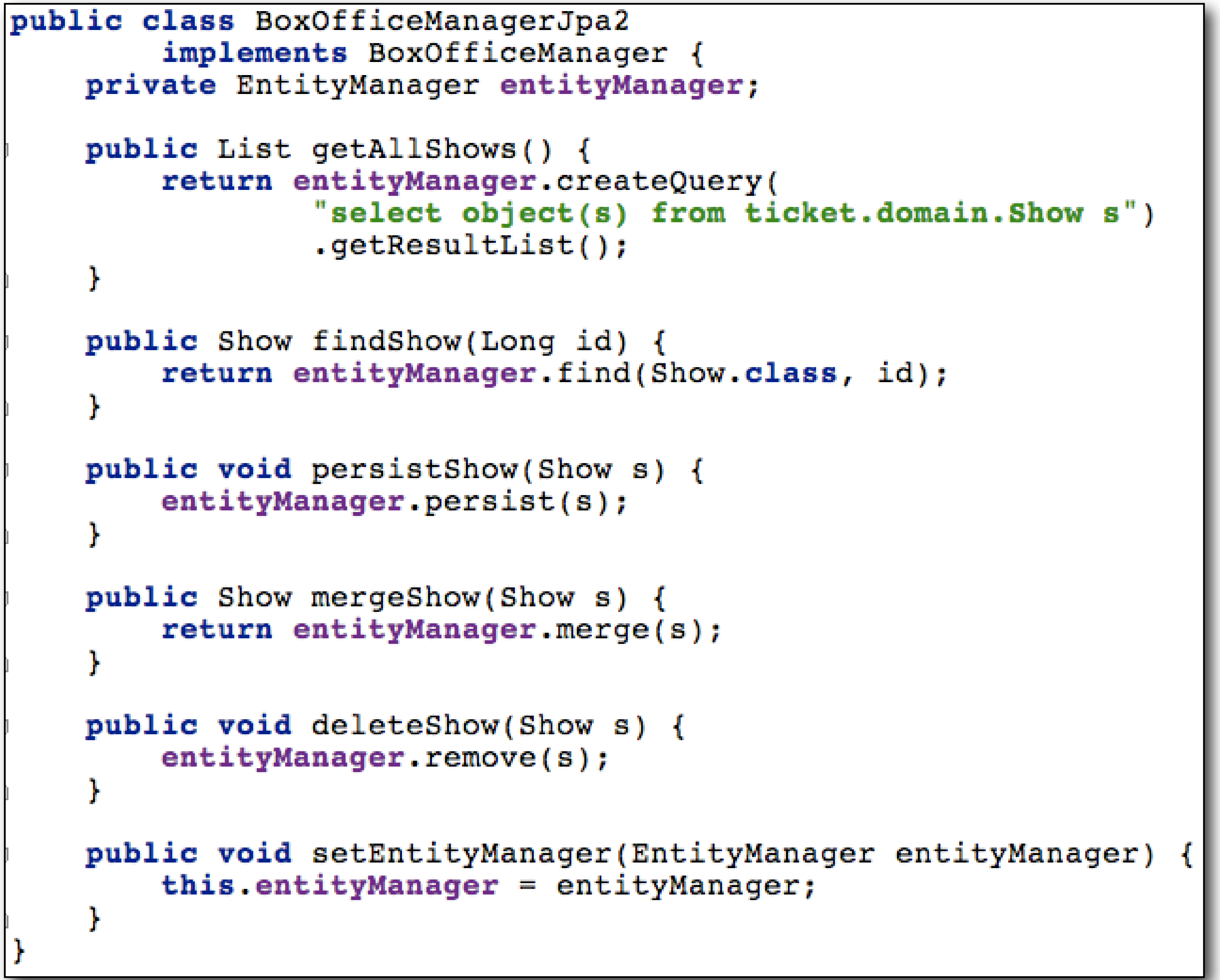
XML Configuration Simplification

```xml
<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"/>

<bean class="org.springframework.transaction.interceptor.TransactionAttributeSourceAdvisor">
    <property name="transactionInterceptor" ref="txInterceptor"/>
</bean>

<bean id="txInterceptor" autowire="byName"
      class="org.springframework.transaction.interceptor.TransactionInterceptor">
    <property name="transactionAttributeSource">
        <bean class="org.sp...k.transaction.annotation.AnnotationTransactionAttributeSource"/>
    </property>
</bean>
```

VS

```xml
<tx:annotation-driven/>
```

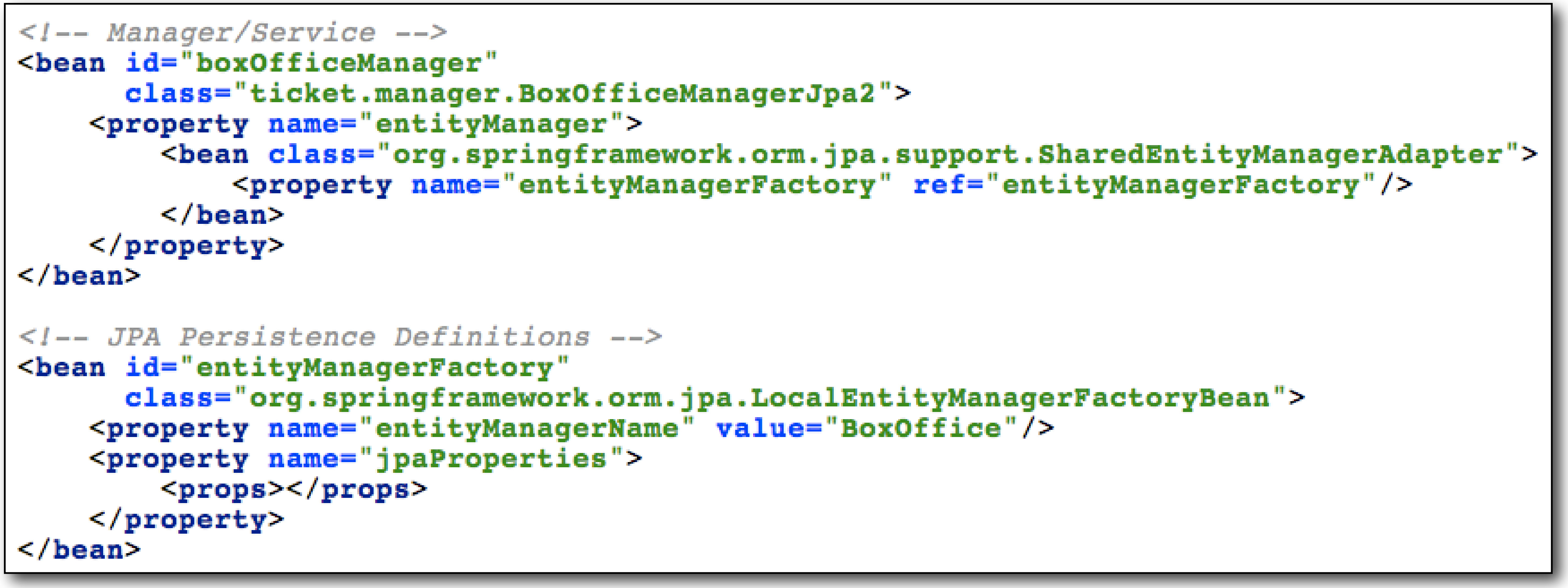
Direct use of JPA API



```java
public class BoxOfficeManagerJpa2
        implements BoxOfficeManager {
    private EntityManager entityManager;

    public List getAllShows() {
        return entityManager.createQuery(
                "select object(s) from ticket.domain.Show s")
                .getResultList();
    }

    public Show findShow(Long id) {
        return entityManager.find(Show.class, id);
    }

    public void persistShow(Show s) {
        entityManager.persist(s);
    }

    public Show mergeShow(Show s) {
        return entityManager.merge(s);
    }

    public void deleteShow(Show s) {
        entityManager.remove(s);
    }

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

Direct use of JPA API

- To get proper transaction management use JNDI lookup in a JTA environment and SharedEntityManagerAdapter for a Resource-local configuration
- No exception translation provided



```xml
<!-- Manager/Service -->
<bean id="boxOfficeManager"
      class="ticket.manager.BoxOfficeManagerJpa2">
    <property name="entityManager">
        <bean class="org.springframework.orm.jpa.support.SharedEntityManagerAdapter">
            <property name="entityManagerFactory" ref="entityManagerFactory"/>
        </bean>
    </property>
</bean>

<!-- JPA Persistence Definitions -->
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
    <property name="entityManagerName" value="BoxOffice"/>
    <property name="jpaProperties">
        <props></props>
    </property>
</bean>
```

Spring's JDBC Abstraction Layer

New in 2.0:

SimpleJdbcTemplate

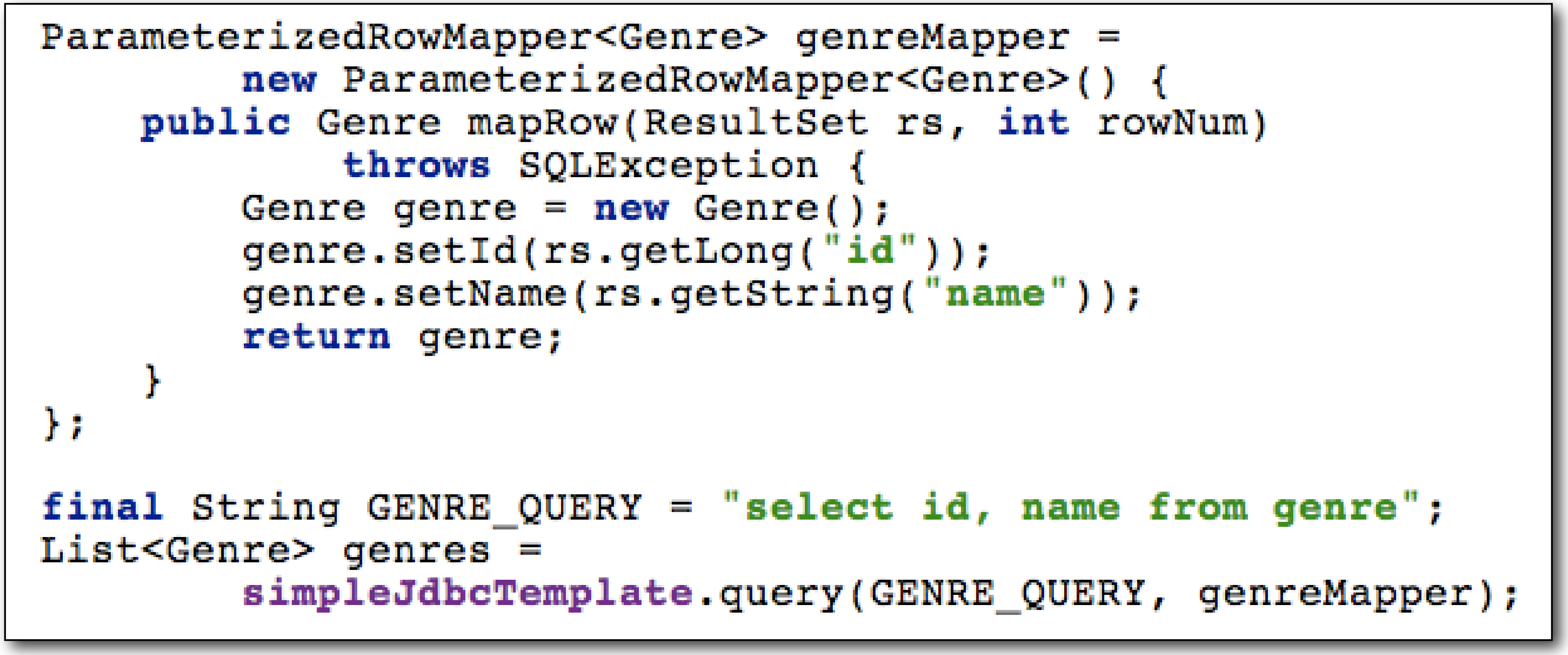
NamedParameterJdbcTemplate

SqlCommand

SimpleJdbcTemplate

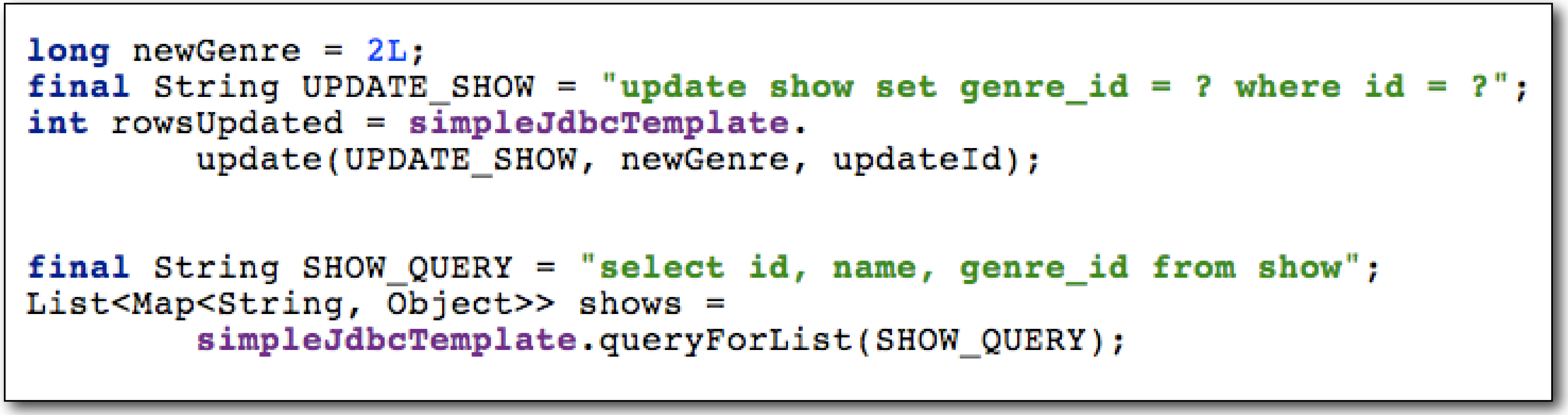
- Designed to take advantage of Java 5 features
  - generics
  - varargs
  - autoboxing
- Wraps a regular JdbcTemplate and if you need additional methods use getJdbcOperations method to access it.

SimpleJdbcTemplate



```java
ParameterizedRowMapper<Genre> genreMapper =
        new ParameterizedRowMapper<Genre>() {
    public Genre mapRow(ResultSet rs, int rowNum)
            throws SQLException {
        Genre genre = new Genre();
        genre.setId(rs.getLong("id"));
        genre.setName(rs.getString("name"));
        return genre;
    }
};

final String GENRE_QUERY = "select id, name from genre";
List<Genre> genres =
        simpleJdbcTemplate.query(GENRE_QUERY, genreMapper);
```

SimpleJdbcTemplate



```
long newGenre = 2L;
final String UPDATE_SHOW = "update show set genre_id = ? where id = ?";
int rowsUpdated = simpleJdbcTemplate.
        update(UPDATE_SHOW, newGenre, updateId);


final String SHOW_QUERY = "select id, name, genre_id from show";
List<Map<String, Object>> shows =
        simpleJdbcTemplate.queryForList(SHOW_QUERY);
```

Named Parameters

Compare this SQL statement:

```
select id, price, brand from product
 where price < ? and brand <> ?
```

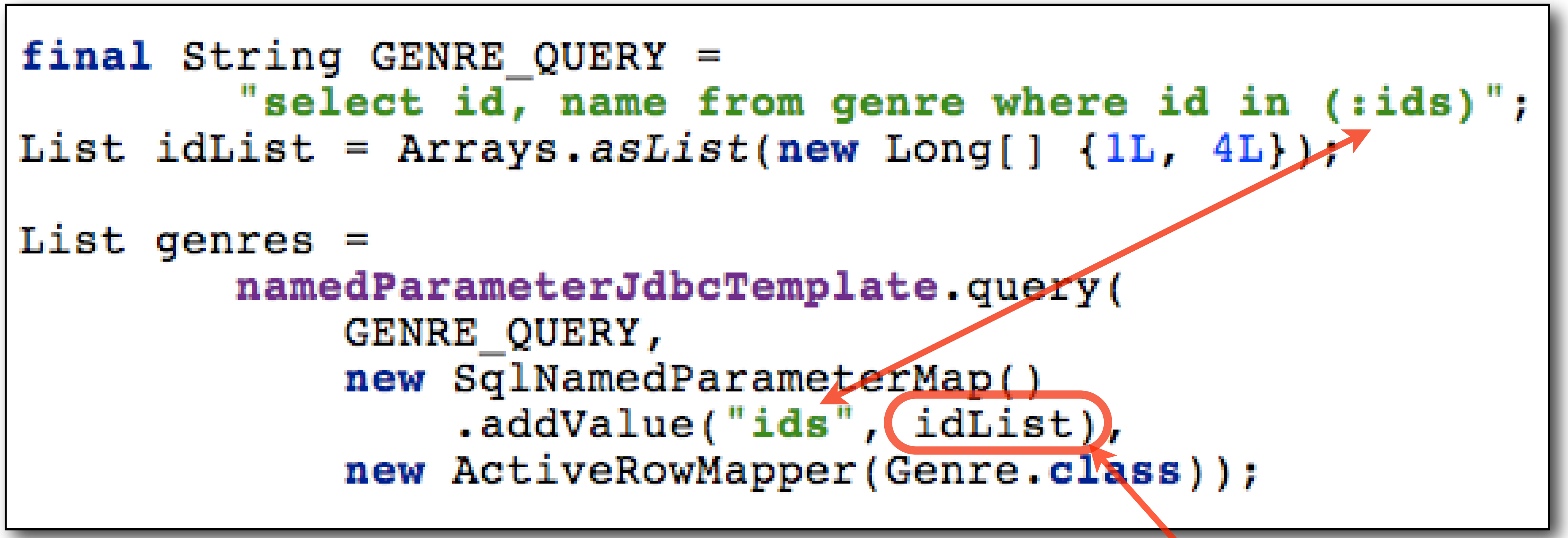
with the following

```
select id, price, brand from product
 where price < :maxPrice
   and brand <> :unwantedBrand
```

NamedParameterJdbcTemplate

- Allows the use of named parameters in any SQL statement.
  - Use a Map to pass in parameter values
  - Map key matches value with parameter name
  - If parameter value is a List then placeholders will be expanded to cover all list members - watch the size of the list!
- Wraps a regular JdbcTemplate and if you need additional methods use getJdbcOperations method to access it.

NamedParameterJdbcTemplate

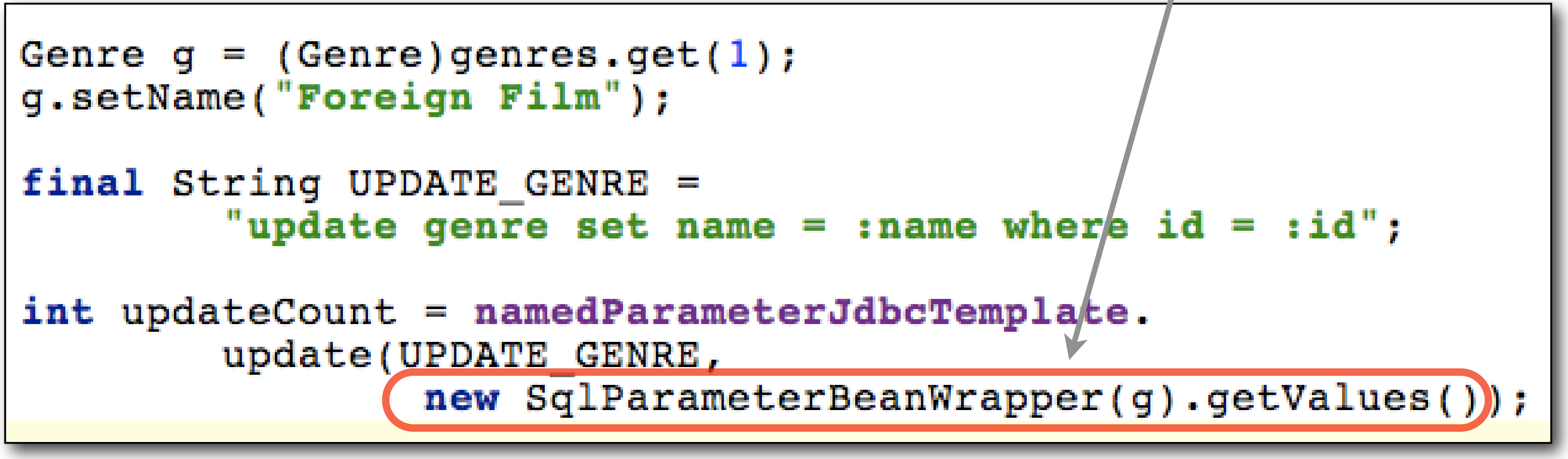


```
final String GENRE_QUERY =
        "select id, name from genre where id in (:ids)";
List idList = Arrays.asList(new Long[] {1L, 4L});

List genres =
        namedParameterJdbcTemplate.query(
            GENRE_QUERY,
            new SqlNamedParameterMap()
                .addValue("ids", idList),
            new ActiveRowMapper(Genre.class));
```

if value is a List we will expand placeholders

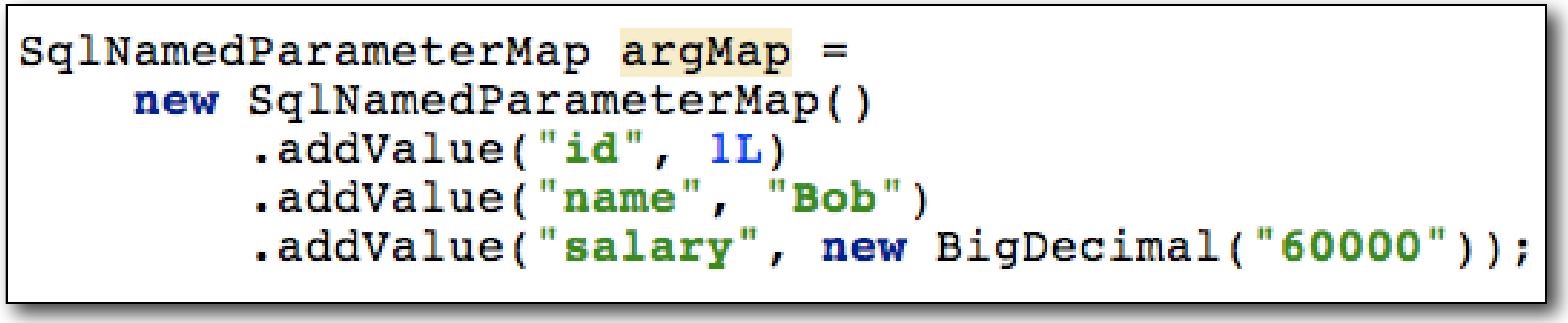
NamedParameterJdbcTemplate

creates a Map containing name and value of all public getters

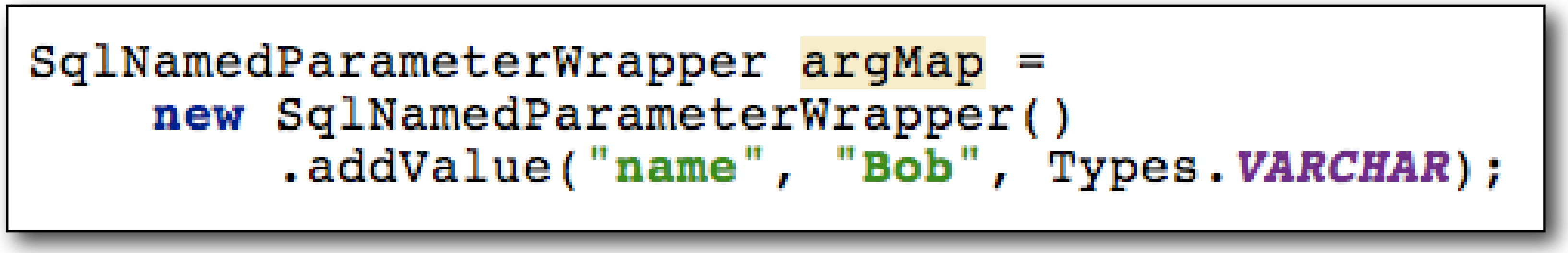


```
Genre g = (Genre)genres.get(1);
g.setName("Foreign Film");

final String UPDATE_GENRE =
        "update genre set name = :name where id = :id";

int updateCount = namedParameterJdbcTemplate.
        update(UPDATE_GENRE,
                new SqlParameterBeanWrapper(g).getValues());
```

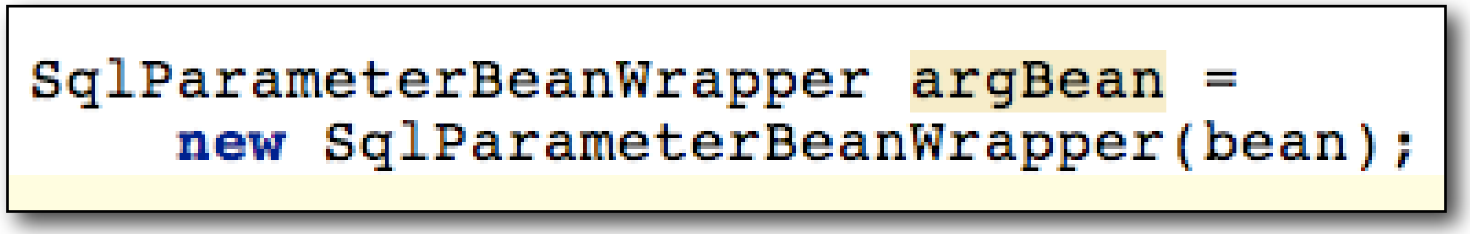
SqlNamedParameterHolder



```java
SqlNamedParameterMap argMap =
    new SqlNamedParameterMap()
        .addValue("id", 1L)
        .addValue("name", "Bob")
        .addValue("salary", new BigDecimal("60000"));
```



```java
SqlNamedParameterWrapper argMap =
    new SqlNamedParameterWrapper()
        .addValue("name", "Bob", Types.VARCHAR);
```



```java
SqlParameterBeanWrapper argBean =
    new SqlParameterBeanWrapper(bean);
```

Common methods:

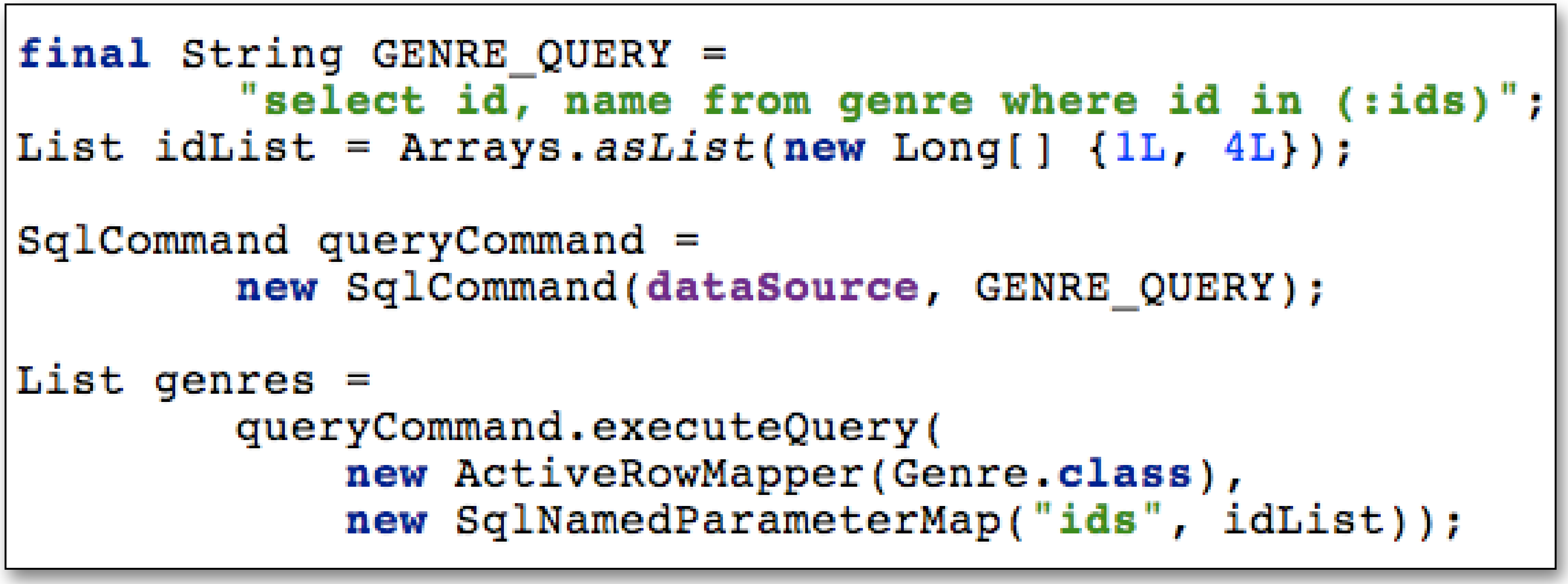
```
Map getValues()
Map getTypes()
void setValues(Map)
void getTypes(Map)
```

SqlCommand

- Alternativ to RdbmsOperation (SqlQuery, SqlUpdate...)
    - No need to explicitly declare parameters - we declare the name in the SQL statement and can declare the Type in a SqlNamedParameterHolder
    - Thread-safe, but lightweight and inexpensive to create whenever needed
    - Various execute methods depending on requested return type:

```
Object executeScalar()   Object executeObject(RowMapper)
List executeQuery()      int executeUpdate()
```

SqlCommand



```java
final String GENRE_QUERY =
        "select id, name from genre where id in (:ids)";
List idList = Arrays.asList(new Long[] {1L, 4L});

SqlCommand queryCommand =
        new SqlCommand(dataSource, GENRE_QUERY);

List genres =
        queryCommand.executeQuery(
            new ActiveRowMapper(Genre.class),
            new SqlNamedParameterMap("ids", idList));
```

SqlCommand

```
Genre g = (Genre)genres.get(1);
g.setName("Foreign Film");
SqlParameterBeanWrapper argBean =
        new SqlParameterBeanWrapper(g);
final String UPDATE_GENRE =
        "update genre set name = :name where id = :id";
SqlCommand updateCommand =
        new SqlCommand(dataSource, UPDATE_GENRE);
int updateCount =
        updateCommand.executeUpdate(argBean);
```

```
SqlInsertBuilder myData = new SqlInsertBuilder()
    .setKeyHolder(new GeneratedKeyHolder())
    .addColumnValue("name", "Presentation One")
    .addColumnValue("genre_id", 2L, Types.INTEGER);
SqlCommand insertCommand = new SqlCommand(dataSource, "show");
int rowsinserted = insertCommand.executeInsert(myData);
```

The Spring Framework Project

Started February 2003

Based on code from Rod Johnsons' book "J2EE Design and Development "

Website http://www.springframework.org/

CVS repository is on SourceForge
http://sourceforge.net/cvs/?group_id=73357
http://fisheye.cenqua.com/changelog/springframework

1.0 released March 2004
1.2 released May 2005
2.0 released Q2 2006

Spring Experience conference Dec. 2005
SpringOne conference June 2006

Development & Support

- **Development**
  - 80% of core committers work for Interface21
- **Commercial Support**
  - Interface21 -- wrote the code
  - BEA -- certified on WebLogic 9.0
  - SpikeSource -- Spike Servlet/J2EE Stack
  - SourceLabs -- SASH 1.1(certified by Oracle)

Training & Documentation

- **Training**
  - Interface21
  - Virtuas
  - ArcMind
- **Documentation / Books**

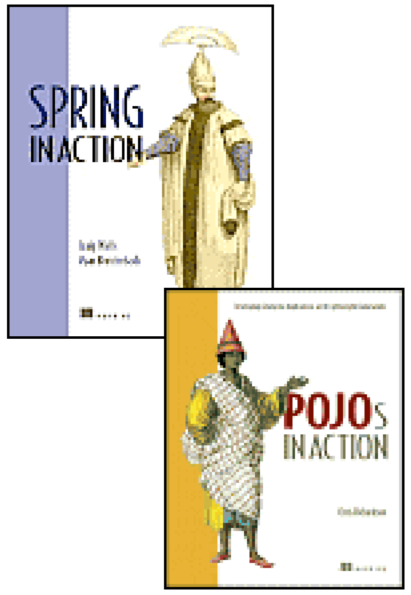
Spring

java/j2ee Application Framework

Reference Documentation

Version 2.0 M3

expert one-on-one
J2EE Design and Development
Professional
Java Development with the Spring Framework



Community

- **Support Forum -** forum.springframework.org
- **User Groups**
  - Philadelphia, PA - Dallas, TX - Sydney, Australia
- **Conferences**

SpringOne
from the Source
INTERFACE21
JUNE 15-16, 2006 AT METROPOLIS IN ANTWERP - BELGIUM
INTERFACE21 AND THE NO FLUFF JUST STUFF JAVA SYMPOSIUM SERIES BRING YOU
THE SPRING EXPERIENCE
December 7-10, 2005 in South Florida at the Sheraton Bal Harbour Beach Resort

PSUG

- **Philadelphia Spring Users Group**
  - http://springdeveloper.com/psug/
  - Meeting -- Tuesday April 4, 2006   6:00pm - 9:00pm
  - Joint meeting with the Delaware Valley BEA Users Group
  - Spring, BEA and Service-Oriented Architectures