



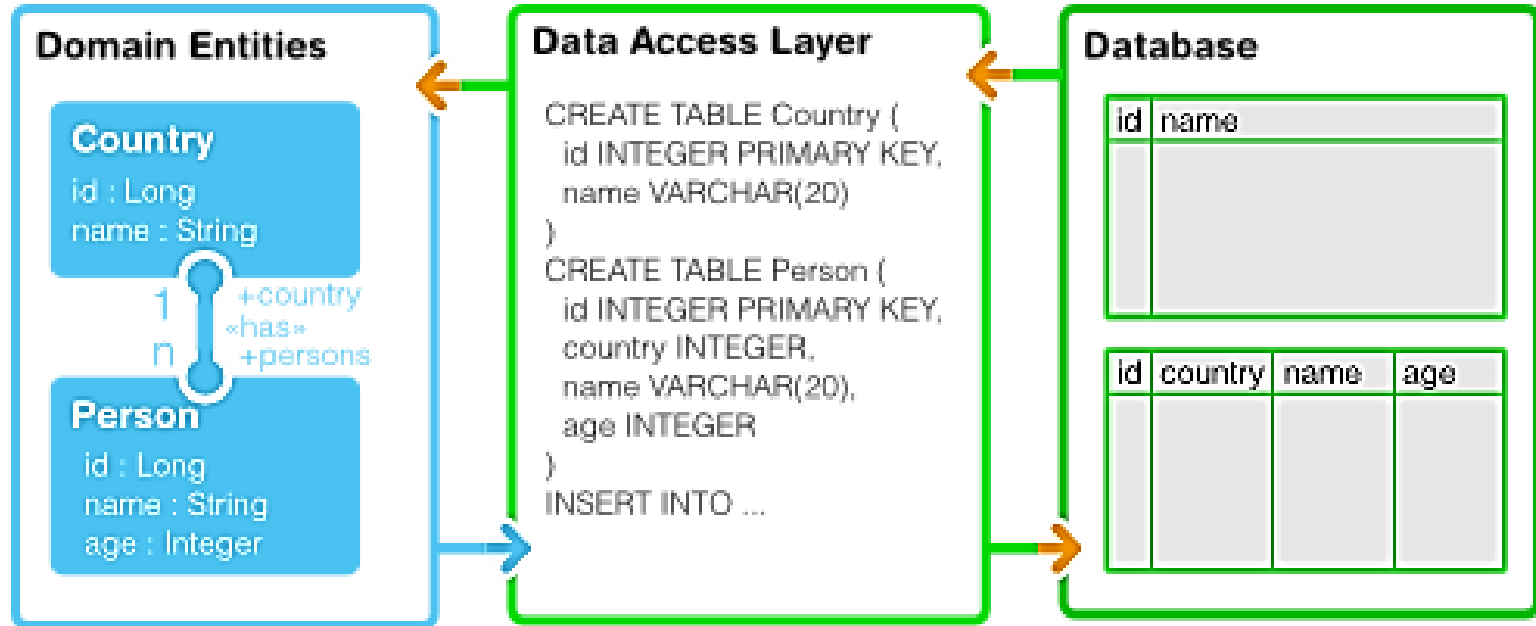
# Spring Databases

## Module 2 Spring ORM

# Spring :: ORM

## ORM - Object-relational mapping

- connects the concepts of relational database and object-oriented system



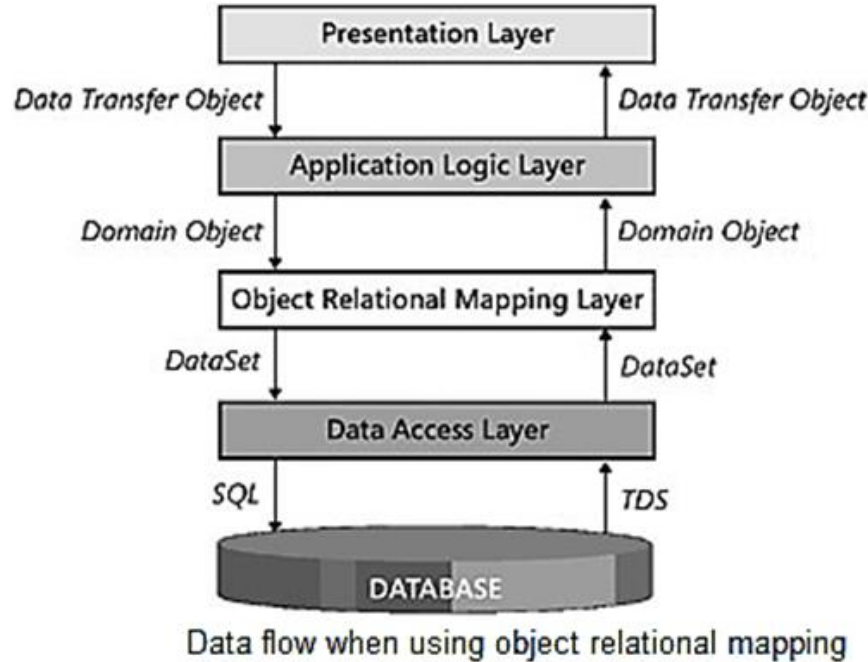
# Spring :: Overview of ORM module

- `org.springframework.orm`
- `org.springframework.orm.hibernate3`
- `org.springframework.orm.hibernate4`
- `org.springframework.orm.ibatis`
- `org.springframework.orm.jpa`



# Spring :: ORM

## ORM - Object-relational mapping



## Spring :: Benefits of Working with ORM

- Speeding development
- Making data access more abstract and portable
- Cache management
- Generating boilerplate code for basic CRUD operations (Create, Read, Update, Delete)

# Spring :: ORM :: Example

```
public interface BookDao {  
    public Book getById(int id);  
    public List<Book> getAll();  
    public void insert(Book book);  
    public void update(int id, String title);  
    public void delete(Book book);  
    public Book findByTitle(String bookTitle);  
}
```

## BOOK

**ID**: integer  
**TITLE** : varchar  
**DATE\_RELEASE** : timestamp

# Spring :: ORM :: Example

Implementation with the use of Spring JDBC

```
public void setDataSource(DataSource dataSource) {  
    this.dataSource = dataSource;  
    jdbcTemplate = new JdbcTemplate(this.dataSource);  
}
```

```
@Override  
public void insert(Book book) {  
    String sql = "INSERT INTO BOOK (TITLE, DATE_RELEASE) VALUES (?, ?)";  
    jdbcTemplate.update(sql, getPreparedStatementSetter(book));  
}
```

# Spring :: ORM :: Example

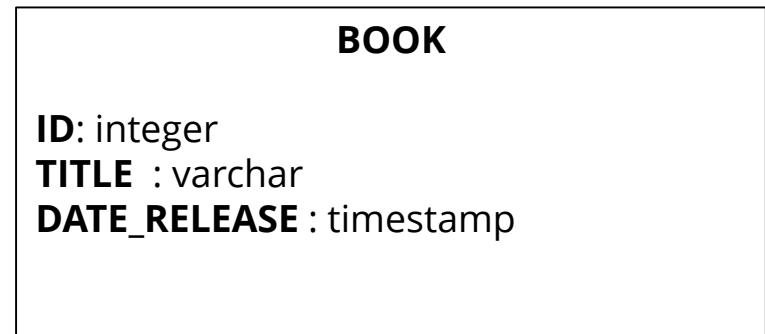
```
@Entity
@Table(name = "BOOK")
public class Book {
    private int id;
    private String title;
    private Date dateRelease;

    @Id
    @GeneratedValue
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

```
@Column(name = "TITLE")
public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}
```





## Spring :: ORM :: Example

**Session** is the main runtime interface between a Java application and Hibernate. This is the central API class abstracting the notion of a persistence service.

The main function of the **Session** is to offer create, read and delete operations for instances of mapped entity classes.

Instances may exist in one of three states:

- *transient*: never persistent, not associated with any **Session**
- *persistent*: associated with a unique **Session**
- *detached*: previously persistent, not associated with any **Session**

## Spring :: ORM :: Example

- Transient instances may be made persistent by calling **save()** or **saveOrUpdate()**.
- Persistent instances may be made transient by calling **delete()**.
- Any instance returned by a **get()** or **load()** method is persistent.
- Detached instances may be made persistent by calling **update()** or **saveOrUpdate()**.
- **save()** and **persist()** result in an SQL INSERT
- **delete()** results in an SQL DELETE
- **update()** results in an SQL UPDATE

# Spring :: ORM :: Example

@Repository

```
public class BookDaoImpl implements BookDao {  
    private SessionFactory sessionFactory;  
  
    private Session session;  
  
    public void setSessionFactory(SessionFactory sessionFactory) {  
        this.sessionFactory = sessionFactory;  
        session = this.sessionFactory.openSession();  
    }  
}
```

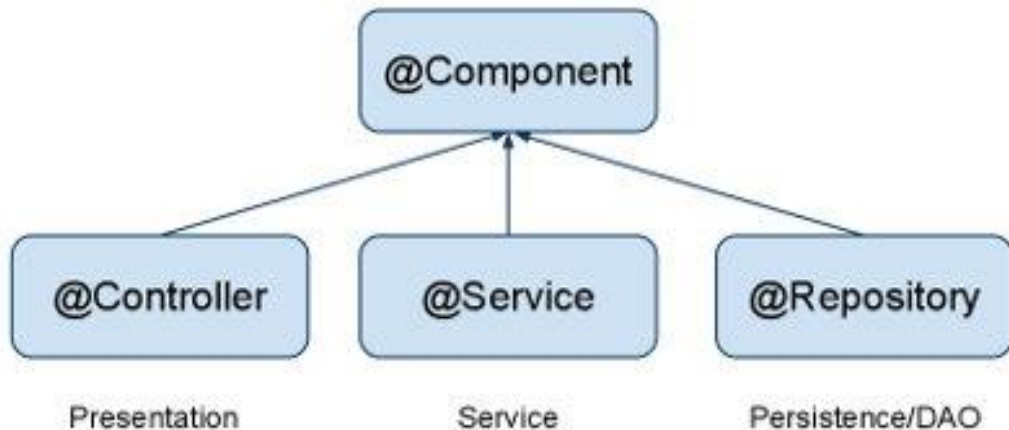
@Override

```
public void insert(Book book) {  
    Transaction tx = session.beginTransaction();  
    session.save(book);  
    tx.commit();  
}
```

In order to persist a transient object, you need to do it within a transaction and to commit it.

# Spring :: ORM :: Stereotypes

- ♦ **Stereotype** is a marker annotation denoting the role of the bean in the overall architecture (at a conceptual, rather than implementation, level).



**@Repository** – for DAO beans, translates checked exceptions to **DataAccessException** hierarchy

**@Service** – for business logic beans

**@Controller** – for controllers

# Spring :: ORM :: Example

```
<bean
class="org.springframework.dao.annotation.
    PersistenceExceptionTranslationPostProcessor" />

<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />

    <property name="hibernateProperties">
        <props>
            <prop
                key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.hbm2ddl.auto">create</prop>
        </props>
    </property>
```

# Spring :: ORM :: Example

```
<property name="annotatedClasses">
  <list>
    <value> com.luxoft.springdb.example9.Book</value>
  </list>
</property>
</bean>

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
  <property name="url" value="jdbc:h2:~/book" />
  <property name="username" value="sa" />
  <property name="password" value="" />
</bean>
```

# Spring :: ORM :: Example with HibernateDaoSupport

@Repository

```
public class BookDaoImpl extends HibernateDaoSupport implements BookDao {  
    @Override  
    @Transactional(readOnly=false)  
    public void insert(Book book) {  
        getHibernateTemplate().save(book);  
    }  
  
    @Override  
    public Book getById(int id) {  
        return (Book) getHibernateTemplate().get(Book.class, id);  
    }  
}
```

Insert/update/delete operations need to be transactional. The **@Transactional** annotation, to be discussed into the transactions module.

# Spring :: ORM :: Example with HibernateDaoSupport

```
<?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-3.1.xsd http://www.springframework.org/schema/tx
                    http://www.springframework.org/schema/tx/spring-tx.xsd">
  <tx:annotation-driven/>
  <bean id="transactionManager"
class="org.springframework.orm.hibernate4.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
  </bean>
```

Need to enable the configuration of transactional behavior based on annotations using `<tx:annotation-driven/>` and to create a `HibernateTransactionManager` bean.



# Spring :: ORM :: Universal DAO

```
public interface Dao<T> {  
    public void insert(final T entity);  
    public void delete(final T entity);  
    public void update(int id, String property);  
    public T findByProperty(String propertyValue);  
    public T getById(int id);  
    public List<T> getAll();  
}
```

# Spring :: ORM :: Universal DAO

@Repository

```
public abstract class AbstractHibernateDao<T> extends HibernateDaoSupport implements Dao<T> {
```

```
    private Class<T> clazz;
```

```
    public void setClazz(final Class<T> clazzToSet) {  
        this.clazz = clazzToSet;  
    }
```

@Override

```
    public T getById(final int id) {  
        return (T) getHibernateTemplate().get(clazz, id);  
    }
```

@Override

```
    @SuppressWarnings("unchecked")  
    public List<T> getAll() {  
        return (List<T>)getHibernateTemplate().find("from " + clazz.getName());  
    }
```

# Spring :: ORM :: Universal DAO

@Repository

```
public class BookDaoImpl extends AbstractHibernateDao<Book> {  
  
    public BookDaoImpl() {  
        setClazz(Book.class);  
    }  
  
    @Transactional(readOnly=false)  
    public void update(int id, String title) {  
        Book book =  
            (Book)getHibernateTemplate().getSessionFactory().openSession()  
                .get(Book.class, id);  
  
        book.setTitle(title);  
        getHibernateTemplate().update(book);  
    }  
}
```

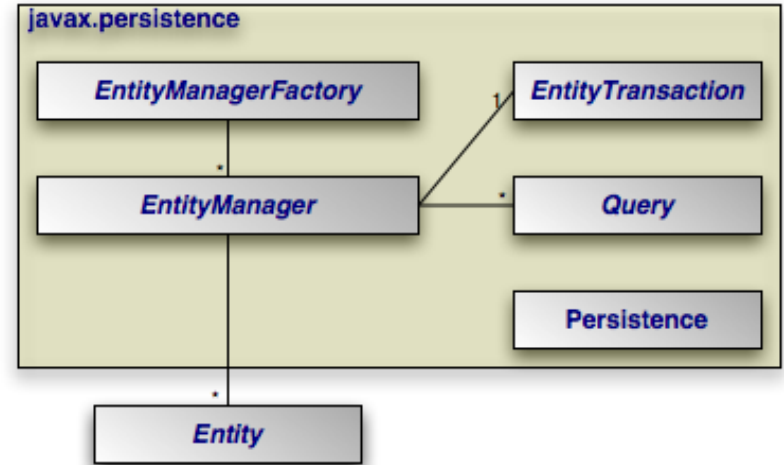
ex.11

# Spring :: ORM :: JPA

**Java Persistence API (JPA)** — API included to Java SE and Java EE since Java 5

Support of the persistency of JPA covers these areas:

- API, defined in package `javax.persistence`
- Java Persistence Query Language
- Meta-information
- Generation of DDL for entities



# Spring :: ORM :: JPA

```
public interface BookDao {  
    public Book getById(int id);  
    public List<Book> getAll();  
    public void insert(Book book);  
    public void update(int id, String title);  
    public void delete(Book book);  
    public Book findByTitle(String bookTitle);  
}
```

```
public abstract class AbstractJpaDao {  
  
    protected EntityManagerFactory entityManagerFactory;  
  
    @PersistenceUnit  
    public void setEntityManagerFactory(EntityManagerFactory entityManagerFactory) {  
        this.entityManagerFactory = entityManagerFactory;  
    }  
}
```

# Spring :: ORM :: JPA

@Repository

```
public class BookJpaDaoImpl extends AbstractJpaDao implements BookDao {
```

@Override

```
public void insert(Book book) {  
    EntityManager entityManager =  
        entityManagerFactory.createEntityManager();
```

```
    entityManager.getTransaction().begin();  
    entityManager.persist(book);  
    entityManager.getTransaction().commit();
```

```
    if (entityManager != null) {  
        entityManager.close();
```

```
}
```

```
}
```

# Spring :: ORM :: JPA

```
<!-- Activates @Autowired -->
<context:annotation-config />
<context:component-scan base-package="com.luxoft.springdb.ormjpa.model,
com.luxoft.springdb.ormjpa.dao" />

<jdbc:embedded-database id="dataSource"/>

<bean id="LocalContainerEntityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="LoadTimeWeaver">
    <bean
      class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"
    />
  </property>
  <property name="dataSource" ref="dataSource"/>
  <property name="persistenceProviderClass"
    value="org.hibernate.ejb.HibernatePersistence"/>
</bean>

<bean id="bookDaoImpl" class="com.luxoft.springdb.ormjpa.dao.jpa.BookJpaDaoImpl" />
```

# Spring :: ORM :: JPA

```
<persistence 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"
  version="2.0">
  <persistence-unit name="springframework.Lab.orm.jpa">
    <description>
      Persistence unit for the Spring JPA tutorial with Hibernate using as JPA provider
    </description>

    <class>com.luxoft.springdb.ormjpa.model.Book</class>

    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
    </properties>
  </persistence-unit>
</persistence>
```

orm-jpa



# Spring :: ORM :: JPA - generified

```
public interface Dao<T> {  
    public void insert(T entity);  
    public void delete(final T entity);  
    public void update(int id, String property);  
    public T findByProperty(String propertyValue);  
    public T getById(int id);  
    public List<T> getAll();  
}
```

# Spring :: ORM :: JPA - generified

```
public abstract class AbstractJpaDao<T> implements Dao<T> {  
  
    protected Class<T> clazz;  
    protected EntityManagerFactory entityManagerFactory;  
  
    public void setClazz(final Class<T> clazzToSet) {  
        this.clazz = clazzToSet;  
    }  
  
    @PersistenceUnit  
    public void setEntityManagerFactory(EntityManagerFactory  
                                       entityManagerFactory) {  
        this.entityManagerFactory = entityManagerFactory;  
    }  
}
```

# Spring :: ORM :: JPA - generified

@Override

```
public void insert(final T entity) {  
    EntityManager entityManager =  
        entityManagerFactory.createEntityManager();  
  
    entityManager.getTransaction().begin();  
    entityManager.persist(entity);  
    entityManager.getTransaction().commit();  
  
    if (entityManager != null) {  
        entityManager.close();  
    }  
}
```

# Spring :: ORM :: JPA - generified

@Repository

```
public class BookJpaDaoImpl extends AbstractJpaDao<Book> {  
    public BookJpaDaoImpl() {  
        setClazz(Book.class);  
    }  
}
```

@Override

```
public Book findByProperty(String propertyValue) {  
    EntityManager entityManager = entityManagerFactory.createEntityManager();  
    Book book = (Book) entityManager  
        .createQuery("SELECT c FROM " + clazz.getName() +  
            " c WHERE c.title LIKE :title")  
        .setParameter("title", propertyValue).getSingleResult();  
    if (entityManager != null) {  
        entityManager.close();  
    }  
    return book;  
}
```

orm-jpa-gen

# Spring :: JPA initialization

Since Spring 3.1+ we can avoid defining of persistence.xml by turning on the auto-scanning of all classes of package **packagesToScan**:

```
<bean id="LocalContainerEntityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="packagesToScan" value="com.luxoft"/>
    <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>
```

# Exercise

Lab guide:

- Exercise 2