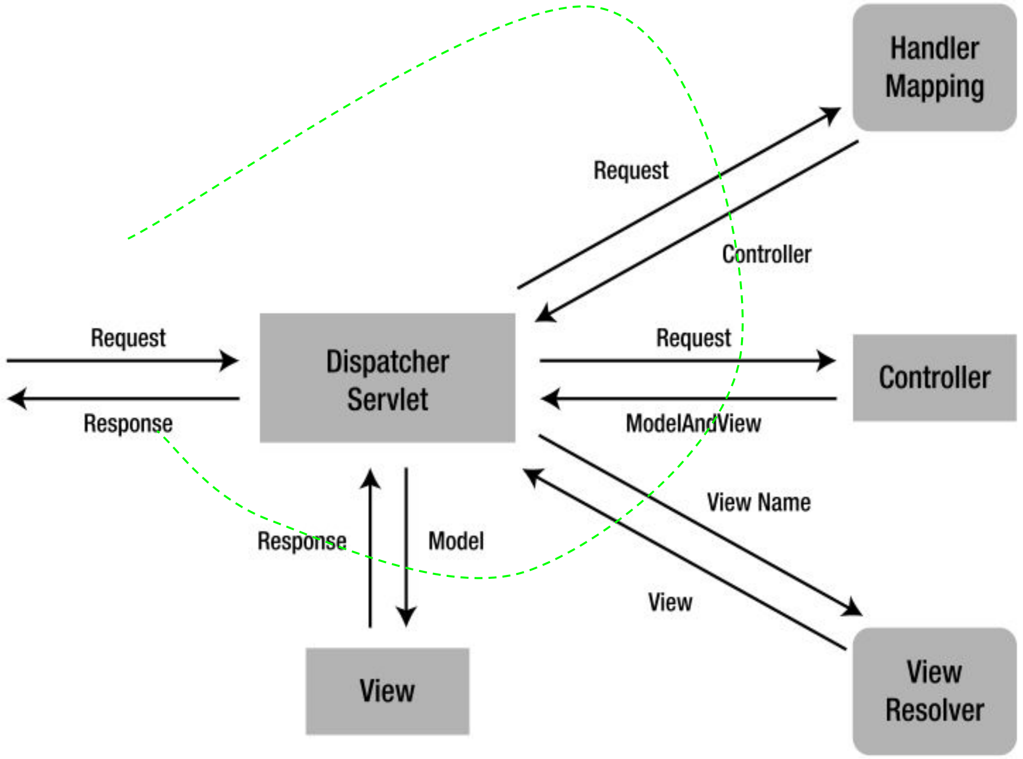
Persistence and Object-Relational Mapping

Charles Zhang

Fall 2019

Review of Spring MVC

- What is MVC and why?
- How is servlet routing configured?
- What are handler interceptors?
- What is the overall data flow for Spring MVC?



Outline

- Introduction to persistence
- JDBC (Java Database Connectivity)
  - DAO
  - Data extraction
- ORM
  - Hibernate
  - JPA, JDO
  - Data mapping

Persistence

- Definition: persistence refers to the characteristic of a state that outlives the process that created it. Without this capability, states would only exist in RAM, and would be lost when RAM loses power, such as a computer shutdown.
- Non-volatile storage
  - Hard drive
  - Flash memory
  - Tape

Can non-volatile storage really be permanent?

Relational database

- Definition: a database that stores information about both data and how they are related. Data and relationships are represented in flat, two-dimensional tables that preserve relational structuring.
- The seemingly incompatibility between relational databases and object-oriented programming

What are SQL and NOSQL?

Java Database Connectivity (JDBC)

- Set of standard APIs to access relational databases in a vendor-independent fashion
- Support creating and executing SQL statements
  - CREATE, INSERT, UPDATE and DELETE, or,
  - Query statements such as SELECT, which return a JDBC row result set

DAO: the Data Access Object design pattern

- Common nightmare of mixing different types of logic
  - Presentation logic, business logic, and data access logic
- Data Access Object (DAO) pattern separates data access logic from business logic and presentation logic
- CRUD encapsulated in an independent module
  - Create, Read, Update, and Delete
  - Query
  - Bulk operations

A simple DAO interface

```
package com.apress.springrecipes.vehicle;

public interface VehicleDao {

    public void insert(Vehicle vehicle);
    public void update(Vehicle vehicle);
    public void delete(Vehicle vehicle);
    public Vehicle findByVehicleNo(String vehicleNo);
}
```

Implement DAO with JDBC

```java
public class JdbcVehicleDao implements VehicleDao {

    private DataSource dataSource;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(Vehicle vehicle) {
        String sql = "INSERT INTO VEHICLE (VEHICLE_NO, COLOR, WHEEL, SEAT)
                + "VALUES (?, ?, ?, ?)";
        Connection conn = null;
        try {
            conn = dataSource.getConnection();
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, vehicle.getVehicleNo());
            ps.setString(2, vehicle.getColor());
            ps.setInt(3, vehicle.getWheel());
            ps.setInt(4, vehicle.getSeat());
            ps.executeUpdate();
            ps.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        } finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {}
            }
        }
    }
```

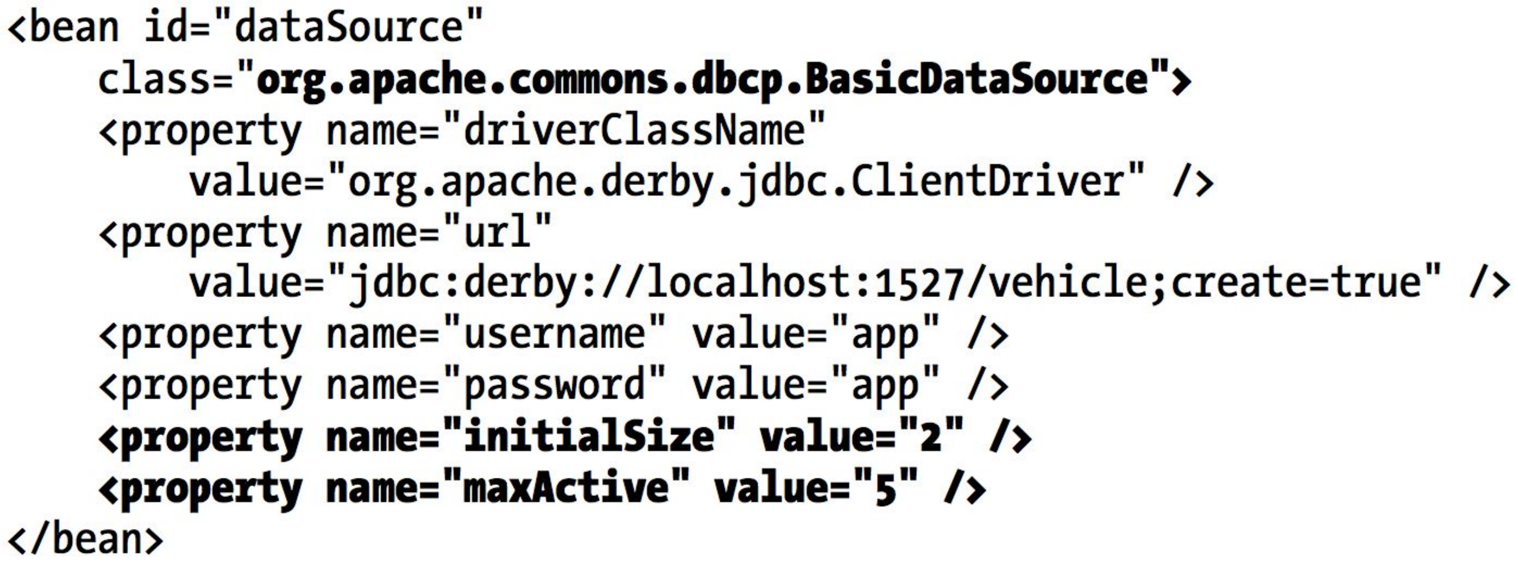
Don’t forget to release the connection in the finally block!

Datasource configuration

- New connection on every request
- One connection only: no concurrency
- Pooled connections
  - Initial and max connection size
  - Database Connection Pooling Services (DBCP) module of the Apache

Do not reinvent the wheel!

Datasource configuration example

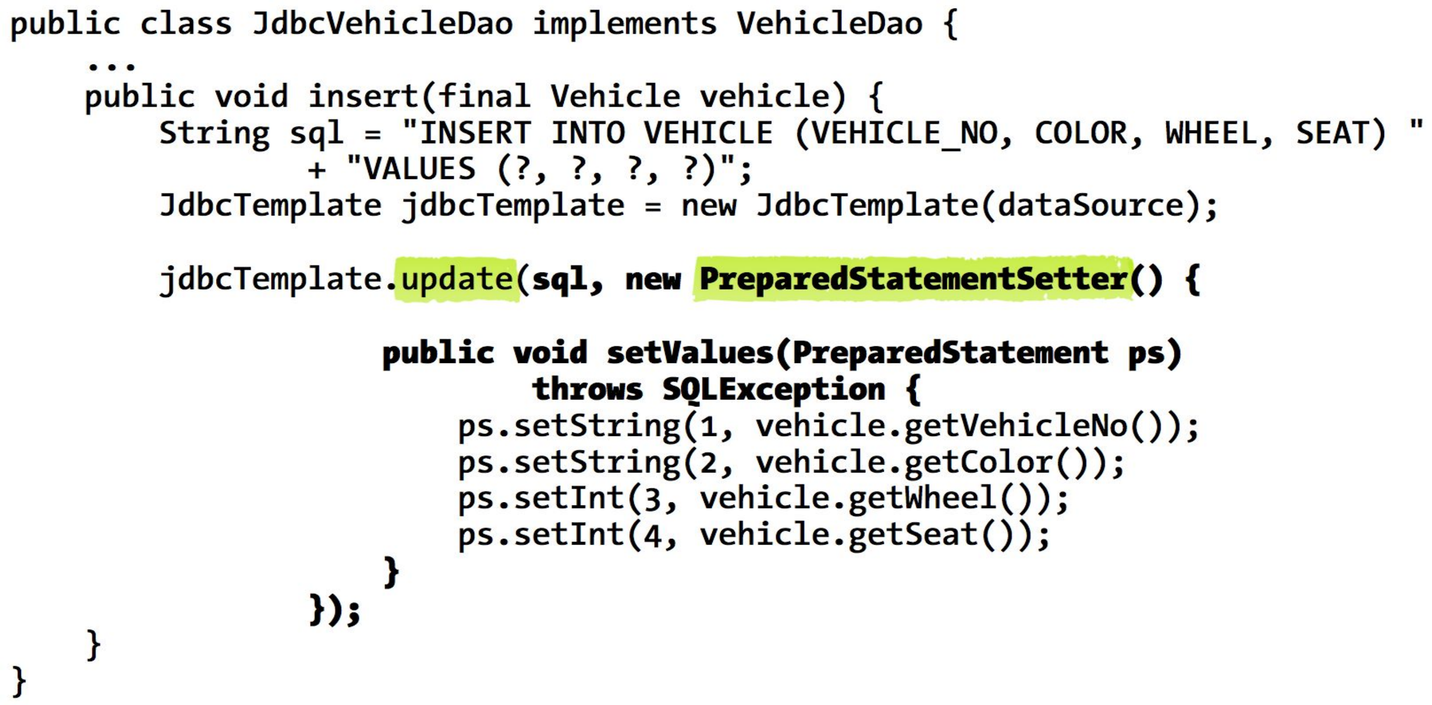


```
<bean id="dataSource"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName"
        value="org.apache.derby.jdbc.ClientDriver" />
    <property name="url"
        value="jdbc:derby://localhost:1527/vehicle;create=true" />
    <property name="username" value="app" />
    <property name="password" value="app" />
    <property name="initialSize" value="2" />
    <property name="maxActive" value="5" />
</bean>
```

Review of JDBC steps

- Obtain a database connection from the data source
- Create a PreparedStatement object from the connection
- Bind the parameters to the PreparedStatement object
- Execute the PreparedStatement object
- Handle SQLException
- Clean up the statement object and connection

JDBC Templates - Update a Database with a Statement Setter

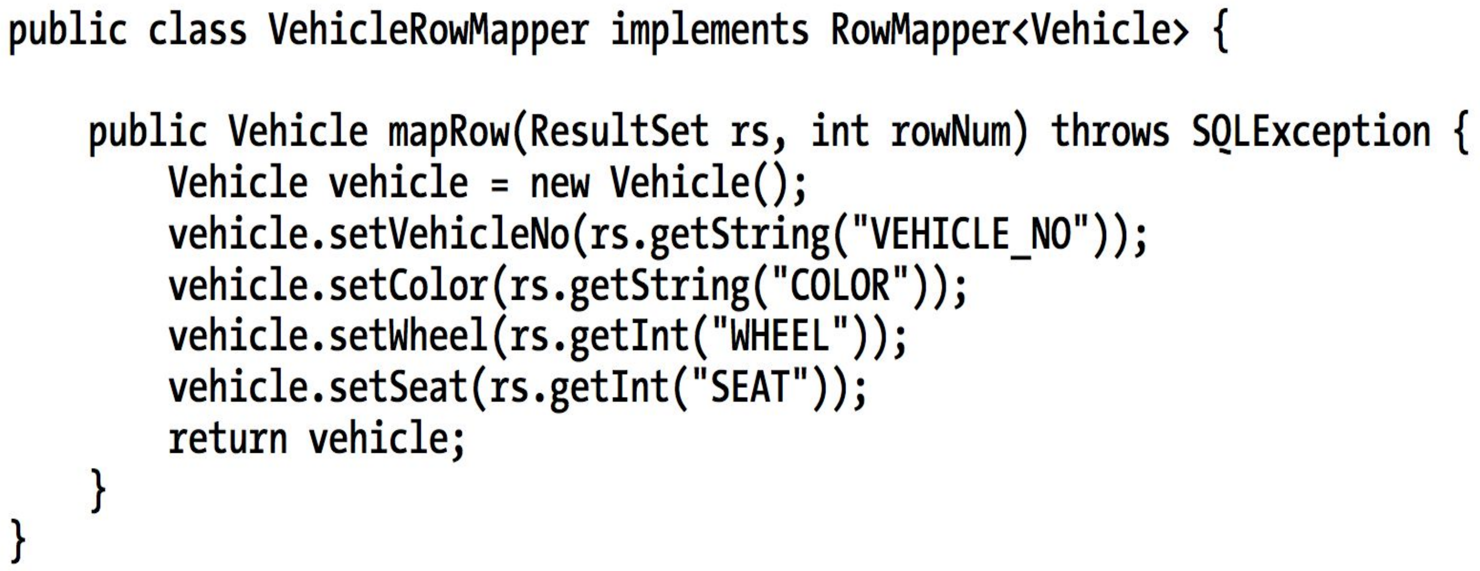


```
public class JdbcVehicleDao implements VehicleDao {
    ...
    public void insert(final Vehicle vehicle) {
        String sql = "INSERT INTO VEHICLE (VEHICLE_NO, COLOR, WHEEL, SEAT) "
                + "VALUES (?, ?, ?, ?)";
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

        jdbcTemplate.update(sql, new PreparedStatementSetter() {

                public void setValues(PreparedStatement ps)
                        throws SQLException {
                    ps.setString(1, vehicle.getVehicleNo());
                    ps.setString(2, vehicle.getColor());
                    ps.setInt(3, vehicle.getWheel());
                    ps.setInt(4, vehicle.getSeat());
                }
            });
    }
}
```

JDBC Templates - Update with a SQL Statement and Parameter Values

```
public class JdbcVehicleDao implements VehicleDao {
    ...
    public void insert(final Vehicle vehicle) {
        String sql = "INSERT INTO VEHICLE (VEHICLE_NO, COLOR, WHEEL, SEAT) "
                + "VALUES (?, ?, ?, ?)";
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

        jdbcTemplate.update(sql, new Object[] { vehicle.getVehicleNo(),
                vehicle.getColor(),vehicle.getWheel(), vehicle.getSeat() });
    }
}
```

Extract data with a row mapper



```
public class VehicleRowMapper implements RowMapper<Vehicle> {

    public Vehicle mapRow(ResultSet rs, int rowNum) throws SQLException {
        Vehicle vehicle = new Vehicle();
        vehicle.setVehicleNo(rs.getString("VEHICLE_NO"));
        vehicle.setColor(rs.getString("COLOR"));
        vehicle.setWheel(rs.getInt("WHEEL"));
        vehicle.setSeat(rs.getInt("SEAT"));
        return vehicle;
    }
}
```

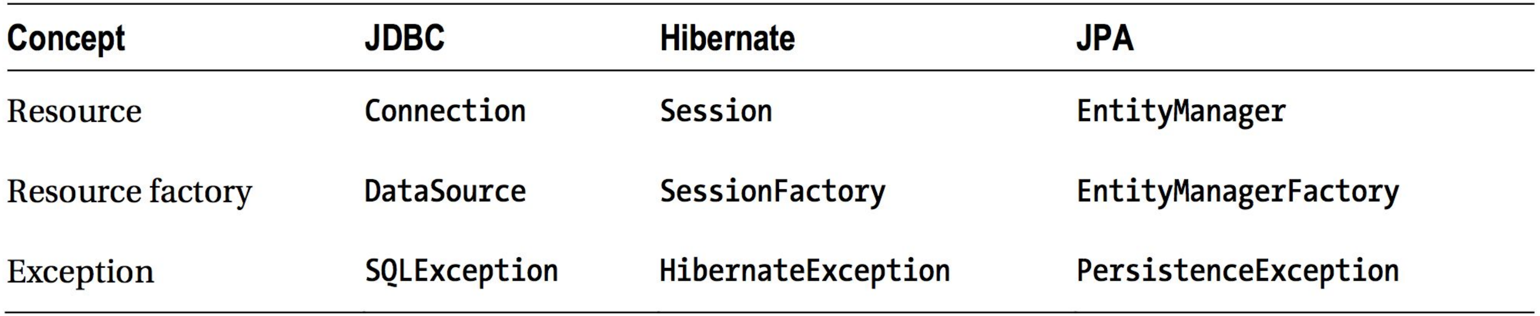
Extract data with a row mapper, continued

```
public class JdbcVehicleDao implements VehicleDao {
    ...
    public Vehicle findByVehicleNo(String vehicleNo) {
        String sql = "SELECT * FROM VEHICLE WHERE VEHICLE_NO = ?";
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

        Vehicle vehicle = (Vehicle) jdbcTemplate.queryForObject(sql,
                new Object[] { vehicleNo }, new VehicleRowMapper());
        return vehicle;
    }
}
```

ORM: Automate JDBC!

- Manually writing all the code for each entity is getting tedious
  - SQL, connection, mapping, ...
- Let the ORM framework do it for you
  - Specify the object⇒ DB mapping
  - Declare the transactional constraints
- Popular frameworks
  - Hibernate, JPA (TopLink), JDO

JDBC, Hibernate, JPA

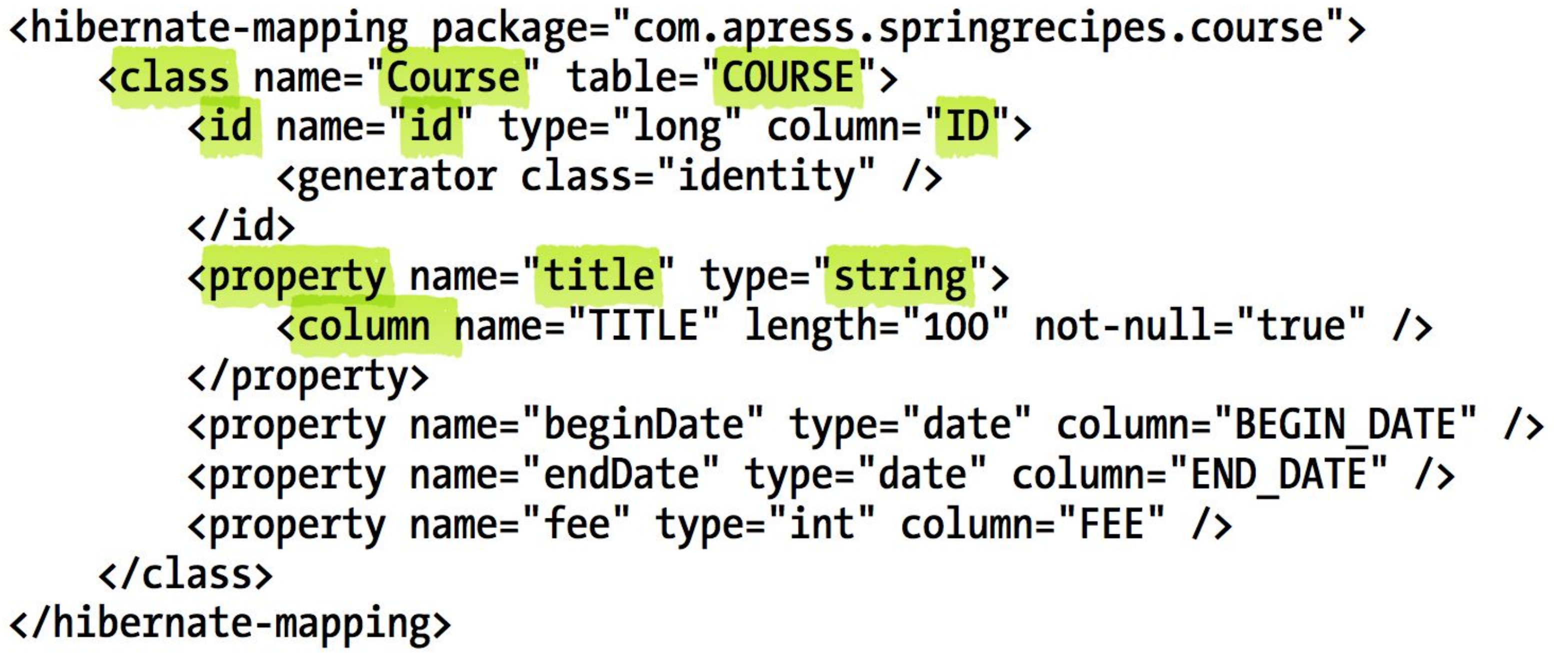


Concept	JDBC	Hibernate	JPA
Resource	`Connection`	`Session`	`EntityManager`
Resource factory	`DataSource`	`SessionFactory`	`EntityManagerFactory`
Exception	`SQLException`	`HibernateException`	`PersistenceException`

JPA vs JDO

Feature	JDO	JPA
JDK Requirement	**1.3+**	1.5+
Usage	J2EE, J2SE	J2EE, J2SE
Persistence specification mechanism	XML, Annotations, **API**	XML, Annotations
Datastore supported	**Any**	RDBMS only
Restrictions on persisted classes	**no-arg constructor (could be added by compiler/enhancer)**	No final classes. No final methods. Non-private no-arg constructor. Identity Field. Version Field.
Ability to persist "transient" fields	**Yes**	No
Persist static/final fields	No	Not specified
Transactions	**Pessimistic**, Optimistic	Optimistic, some locking
Object Identity	**datastore-identity**, application-identity	application-identity
Object Identity generation	Sequence, Table, Identity, Auto, **UUID String, UUID Hex**	Sequence, Table, Identity, Auto
Change objects identity	**Throw exception when not allowed**	Undefined !!
Supported types	Java primitive types, wrappers of primitive types, java.lang.String, **java.lang.Number**, java.math.BigInteger, java.math.BigDecimal, **java.util.Currency, java.util.Locale**, java.util.Date, java.sql.Time, java.sql.Date, java.sql.Timestamp, java.io.Serializable, **boolean[]**, byte[], char[], **double[], float[], int[], long[], short[]**, **java.lang.Object**, **interface**, **Boolean[]**, Byte[], Character[], **Double[], Float[], Integer[], Long[], Short[], BigDecimal[], BigInteger[], String[], PersistenceCapable[], interface[], Object[]**, Enums, java.util.Collection, java.util.Set, java.util.List, java.util.Map, **Collection/List/Map of simple types**, **Collection/List/Map of reference (interface/Object) types**, Collection/List/Map of persistable types	Java primitive types, wrappers of the primitive types, java.lang.String, java.math.BigInteger, java.math.BigDecimal, java.util.Date, **java.util.Calendar**, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.io.Serializable, byte[], Byte[], char[], Character[], Enums, java.util.Collection, java.util.Set, java.util.List, java.util.Map Collection/List/Map of persistable types
Embedded Fields	Embedded persistent objects, **Embedded Collections, Embedded Maps**	Embedded persistent objects
Access a non-detached field	**Throw exception**	Undefined !!
Inheritance	**Each class has its own strategy**	Root class defines the strategy
Operation cascade default	persist, (delete)	
Operation Cascade configuration	delete	persist, delete, refresh
Query Language	JDOQL, SQL, others	JPQL, SQL

Hibernate XML Mappings



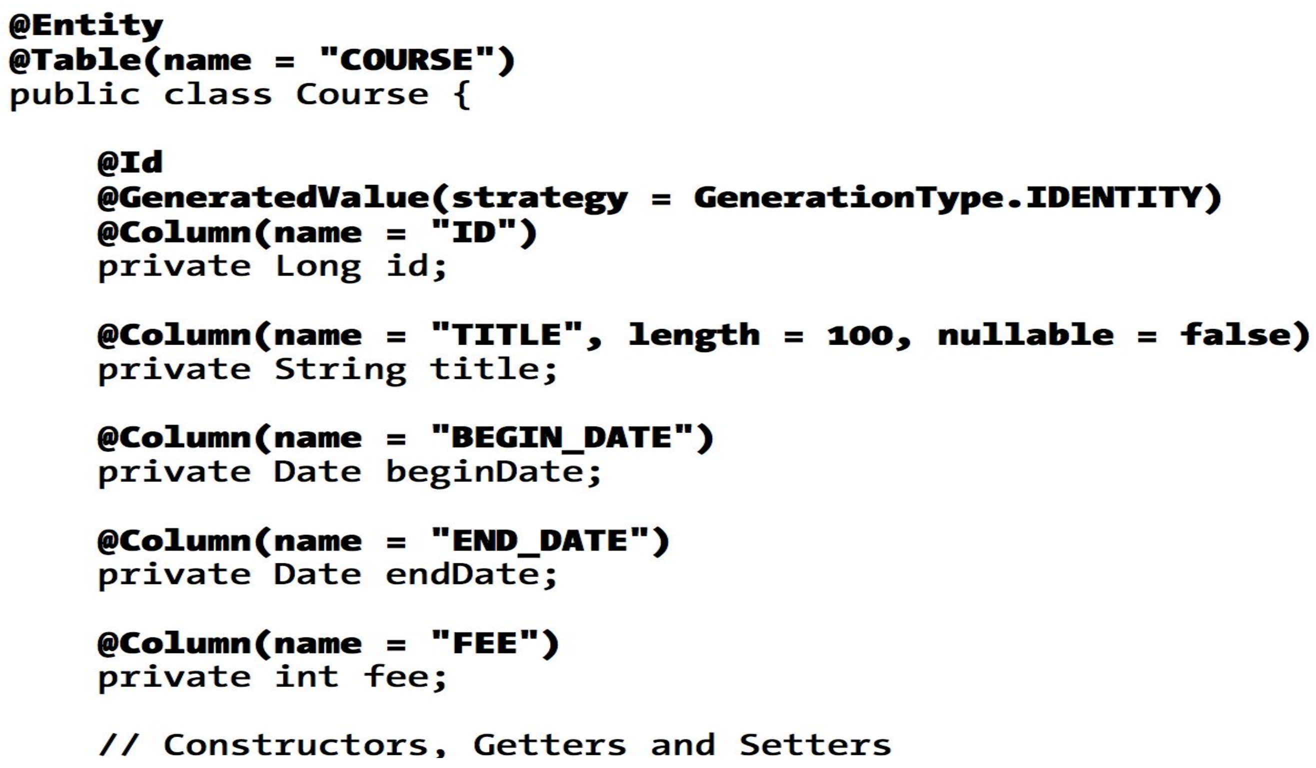
```
<hibernate-mapping package="com.apress.springrecipes.course">
    <class name="Course" table="COURSE">
        <id name="id" type="long" column="ID">
            <generator class="identity" />
        </id>
        <property name="title" type="string">
            <column name="TITLE" length="100" not-null="true" />
        </property>
        <property name="beginDate" type="date" column="BEGIN_DATE" />
        <property name="endDate" type="date" column="END_DATE" />
        <property name="fee" type="int" column="FEE" />
    </class>
</hibernate-mapping>
```

DAO in Hibernate

```
public class HibernateCourseDao implements CourseDao {

    private SessionFactory sessionFactory;

    public HibernateCourseDao() {
        Configuration configuration = new Configuration().configure();
        sessionFactory = configuration.buildSessionFactory();
    }

    public void store(Course course) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            session.saveOrUpdate(course);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
```

What’s in catch and finally?

JPA Annotations



```
@Entity
@Table(name = "COURSE")
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private Long id;

    @Column(name = "TITLE", length = 100, nullable = false)
    private String title;

    @Column(name = "BEGIN_DATE")
    private Date beginDate;

    @Column(name = "END_DATE")
    private Date endDate;

    @Column(name = "FEE")
    private int fee;

    // Constructors, Getters and Setters
```

DAO with JPA

```java
public class JpaCourseDao implements CourseDao {

    private EntityManagerFactory entityManagerFactory;

    public JpaCourseDao() {
        entityManagerFactory = Persistence.createEntityManagerFactory("course");
    }

    public void store(Course course) {
        EntityManager manager = entityManagerFactory.createEntityManager();
        EntityTransaction tx = manager.getTransaction();
        try {
            tx.begin();
            manager.merge(course);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            manager.close();
        }
    }
```

Configure JPA templates

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

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

<bean name="courseDao"
    class="com.apress.springrecipes.course.jpa.JpaCourseDao">
    <property name="jpaTemplate" ref="jpaTemplate" />
</bean>
```

DAO implementation with JPA templates

```
public class JpaCourseDao implements CourseDao {

    private JpaTemplate jpaTemplate;

    public void setJpaTemplate(JpaTemplate jpaTemplate) {
        this.jpaTemplate = jpaTemplate;
    }

    @Transactional
    public void store(Course course) {
        jpaTemplate.merge(course);
    }

    @Transactional
    public void delete(Long courseId) {
        Course course = jpaTemplate.find(Course.class, courseId);
        jpaTemplate.remove(course);
    }
```

Topics for next lecture

- Collection mapping
  - One-to-one
  - One-to-many
  - Many-to-many
  - …
- Class hierarchy persistence