Advanced Java Programming Course

Java Persistence API
Hibernate OGM

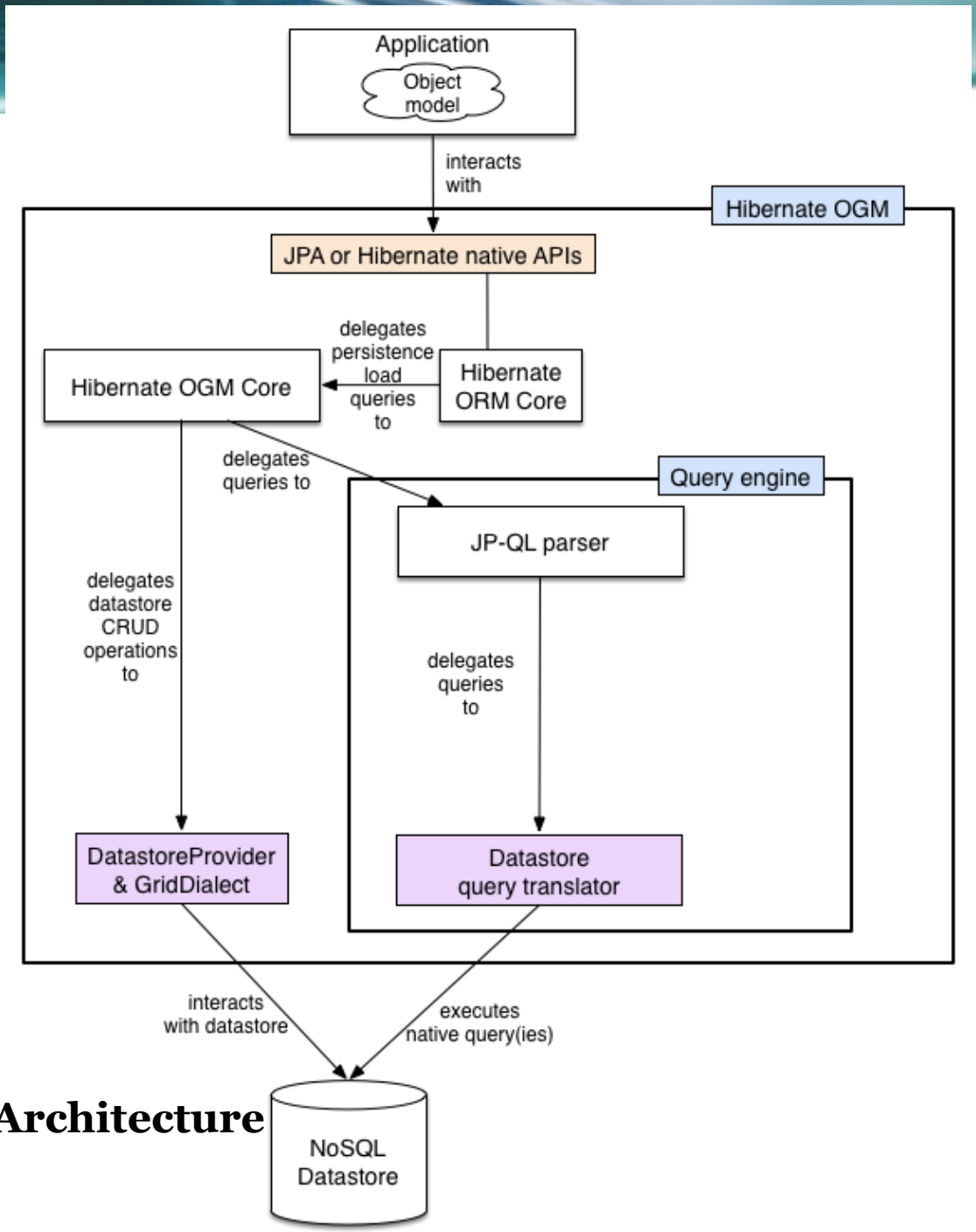
Faculty of Information Technologies
Industrial University of Ho Chi Minh City

Session objectives

- Hibernate OGM
  - Architecture
  - Query
  - Mapping core
  - Associations

Based on: Hibernate OGM 5.1.0.Final Reference Guide

General Architecture



Configure and start Hibernate OGM

- **Using JPA**

persistence.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
6          http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
7      version="2.0">
8
9      <persistence-unit name="ogm-jpa" transaction-type="RESOURCE_LOCAL">
10         <provider>org.hibernate.ogm.jpa.HibernateOgmPersistence</provider>
11         <class>entities.Lophoc</class>
12         <class>entities.Sinhvien</class>
13         <class>entities.Monhoc</class>
14         <properties>
15             <property name="hibernate.ogm.datastore.provider"
16                 value="org.hibernate.ogm.datastore.mongodb.impl.MongoDBDatastoreProvider" />
17             <property name="hibernate.ogm.datastore.database" value="demodb" />
18             <property name="hibernate.ogm.mongodb.host" value="localhost" />
19             <property name="hibernate.ogm.datastore.create_database"
20                 value="true" />
21         </properties>
22     </persistence-unit>
23 </persistence>
```

persistence.xml file

Configure and start Hibernate OGM

Using Hibernate ORM native APIs

```
// create the StandardServiceRegistry
StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
        .applySetting(OgmProperties.ENABLED, true)
        // assuming you are using JDBCTransactionFactory
        .applySetting(AvailableSettings.TRANSACTION_COORDINATOR_STRATEGY,
                    "org.hibernate.transaction.JDBCTransactionFactory")
        // configure current session context
        .applySetting(AvailableSettings.CURRENT_SESSION_CONTEXT_CLASS, "thread")
        // assuming MongoDB as the backend
        .applySetting(OgmProperties.DATASTORE_PROVIDER, MongoDB.DATASTORE_PROVIDER_NAME)
        .applySetting(OgmProperties.DATABASE, "demodb")
        .applySetting(OgmProperties.CREATE_DATABASE, "true")
        .applySetting(OgmProperties.HOST, "127.0.0.1:27017")
        .build();

OgmSessionFactory ogmSessionFactory= new MetadataSources(registry)
        .addAnnotatedClass(Sinhvien.class)
        .addAnnotatedClass( Lophoc.class )
        .addAnnotatedClass( Monhoc.class )
        .buildMetadata()
        .getSessionFactoryBuilder()
        .unwrap(OgmSessionFactoryBuilder.class)
        .build();
OgmSession ogmSession = ogmSessionFactory.openSession();
```

Using JP-QL

```
@Entity @Indexed
public class Hypothesis {

    @Id
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    private String id;

    @Field(analyze=Analyze.NO)
    public String getDescription() { return description; }
    public void setDescription(String description) { this.description =
description; }
    private String description;
}

Query query = session
    .createQuery("from Hypothesis h where h.description = :desc")
    .setString("desc", "tomorrow it's going to rain");
```

```
1  // query returning an entity based on a simple predicate
2  select h from Hypothesis h where id = 16
3
4  // projection of the entity property
5  select id, description from Hypothesis h where id = 16
6
7  // projection of the embedded properties
8  select h.author.address.street from Hypothesis h where h.id = 16
9
10 // predicate comparing a property value and a literal
11 from Hypothesis h where h.position = '2'
12
13 // negation
14 from Hypothesis h where not h.id = '13'
15 from Hypothesis h where h.position <> 4
16
17 // conjunction
18 from Hypothesis h where h.position = 2 and not h.id = '13'
19
20 // named parameters
21 from Hypothesis h where h.description = :myParam
```

```
23  // range query
24  from Hypothesis h where h.description BETWEEN :start and :end
25
26  // comparisons
27  from Hypothesis h where h.position < 3
28
29  // in
30  from Hypothesis h where h.position IN (2, 3, 4)
31
32  // like
33  from Hypothesis h where h.description LIKE '%dimensions%'
34
35  // comparison with null
36  from Hypothesis h where h.description IS null
37
38  // order by
39  from Hypothesis h where h.description IS NOT null ORDER BY id
40  from Helicopter h order by h.make desc, h.name
```

Using the native query language of your NoSQL

```
@Entity
@NamedNativeQuery(
    name = "AthanasiaPoem",
    query = "{ $and: [ { name : 'Athanasia' }, { author : 'Oscar Wilde' } ] }",
    resultClass = Poem.class )
public class Poem {

    @Id
    private Long id;

    private String name;

    private String author;

   // getters, setters ...

}
```

with EntityManager

```
javax.persistence.EntityManager em = ...

// a single result query
String query1 = "MATCH ( n:Poem { name:'Portia', author:'Oscar Wilde' } ) RETURN n";
Poem poem = (Poem) em.createNativeQuery( query1, Poem.class ).getSingleResult();

// query with order by
String query2 = "MATCH ( n:Poem { name:'Portia', author:'Oscar Wilde' } ) " +
                "RETURN n ORDER BY n.name";
List<Poem> poems = em.createNativeQuery( query2, Poem.class ).getResultList();

// query with projections
String query3 = MATCH ( n:Poem ) RETURN n.name, n.author ORDER BY n.name";
List<Object[]> poemNames = (List<Object[]>)em.createNativeQuery( query3 )
                                .getResultList();

// named query
Poem poem = (Poem) em.createNamedQuery( "AthanasiaPoem" ).getSingleResult();
```

with OgmSession

- Use OgmSession.createNativeQuery or Session.getNamedQuery.
- The former form lets you define the result set mapping programmatically. The latter is receiving the name of a predefined query already describing its result set mapping.

```
OgmSession session = (OgmSession)em.getDelegate();
//OgmSession ogmSession = em.unwrap(OgmSession.class);
```

```
OgmSession session = ...
String query1 = "{ $and: [ { name : 'Portia' }, { author : 'Oscar Wilde' } ] }";
Poem poem = session.createNativeQuery( query1 )
                        .addEntity( "Poem", Poem.class )
                        .uniqueResult();
```

Using Hibernate Search (Apache Lucene)

- Hibernate Search offers a way to index Java objects into Lucene indexes and to execute full-text queries on them.
- Apache Lucene is a full-text indexing and query engine with excellent query performance. Feature wise, *full-text* means you can do much more than a simple equality match.

```
@Entity @Indexed
public class Hypothesis {

    @Id
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    private String id;

    @Field(analyze=Analyze.YES)
    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }
    private String description;
}
```

```
EntityManager entityManager = ...
//Add full-text superpowers to any EntityManager:
FullTextEntityManager ftem = Search.getFullTextEntityManager(entityManager);

//Optionally use the QueryBuilder to simplify Query definition:
QueryBuilder b = ftem.getSearchFactory()
   .buildQueryBuilder()
   .forEntity(Hypothesis.class)
   .get();

//Create a Lucene Query:
Query lq = b.keyword().onField("description").matching("tomorrow").createQuery();

//Transform the Lucene Query in a JPA Query:
FullTextQuery ftQuery = ftem.createFullTextQuery(lq, Hypothesis.class);

//List all matching Hypothesis:
List<Hypothesis> resultList = ftQuery.getResultList();
```

Using the Criteria API

- Future Support

Default JPA mapping for an entity

```java
@Entity
public class News {

    @Id
    private String id;
    private String title;

    // getters, setters ...
}
```

```
// Stored in the Collection "News"
{
    "_id" : "1234-5678-0123-4567",
    "title": "On the merits of NoSQL",
}
```

Default JPA mapping for an entity

```java
@Entity
// Overrides the collection name
@Table(name = "News_Collection")
public class News {

    @Id
    private String id;

    // Overrides the field name
    @Column(name = "headline")
    private String title;

    // getters, setters ...
}
```

```
// Stored in the Collection "News"
{
    "_id" : "1234-5678-0123-4567",
    "headline": "On the merits of NoSQL",
}
```

Rename field and collection using @Table and @Column

Define an identifier as a primitive type

```java
@Entity
public class Bookmark {

    @Id
    private String id;

    private String title;

    // getters, setters ...
}
```

```
{
  "_id" : "bookmark_1"
  "title" : "Hibernate OGM documentation"
}
```

Define an identifier using @EmbeddedId

```java
@Embeddable
public class NewsID implements Serializable {

    private String title;
    private String author;

    // getters, setters ...
}
```

```java
@Entity
public class News {

    @EmbeddedId
    private NewsID newsId;
    private String content;

    // getters, setters ...
}
```

```
{
  "_id" : {
      "author" : "Guillaume",
      "title" : "How to use Hibernate OGM ?"
  },
  "content" : "Simple, just like ORM but with a NoSQL database"
}
```

Define an id as org.bson.types.ObjectId

- Generally, it is recommended though to work with MongoDB's object id data type.
- This will facilitate the integration with other applications expecting that common MongoDB id type. To do so, you have two options:
  - Define your id property as org.bson.types.ObjectId
  - Define your id property as String and annotate it with @Type(type="objectid")

```
@Entity
public class News {

    @Id
    private ObjectId id;

    private String title;

    // getters, setters ...
}
```

```
@Entity
public class News {

    @Id
    @Type(type = "objectid")
    private String id;

    private String title;

    // getters, setters ...
}
```

Identifier generation strategies

- You can assign id values yourself or let Hibernate OGM generate the value using the *@GeneratedValue* annotation.
- There are 4 different strategies:
  - IDENTITY (suggested)
  - TABLE
  - SEQUENCE
  - AUTO

Define an id of type String as ObjectId

```java
@Entity
public class News {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Type(type = "objectid")
    private String id;

    private String title;

    // getters, setters ...
}
```

```
{
    "_id" : ObjectId("5425448830048b67064d40b1"),
    "title" : "Exciting News"
}
```

```java
@Entity
public class News {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private ObjectId id;

    private String title;

    // getters, setters ...
}
```

```
{
    "_id" : ObjectId("5425448830048b67064d40b1"),
    "title" : "Exciting News"
}
```

TABLE generation strategy

```java
@Entity
public class GuitarPlayer {

    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;

    private String name;

    // getters, setters ...
}
```

GuitarPlayer collection

```
{
    "_id" : NumberLong(1),
    "name" : "Buck Cherry"
}
```

hibernate_sequences collection

```
{
    "_id" : "GuitarPlayer",
    "next_val" : 101
}
```

SEQUENCE generation strategy

```java
@Entity
public class Song {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private Long id;

  private String title;

  // getters, setters ...
}
```

Song collection

```
{
  "_id" : NumberLong(2),
  "title" : "Flower Duet"
}
```

hibernate_sequences collection

```
{ "_id" : "song_sequence_name", "next_val" : 21 }
```

AUTO generation strategy

```java
@Entity
public class DistributedRevisionControl {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String name;

  // getters, setters ...
}
```

Careful

DistributedRevisionControl collection

```
{ "_id" : NumberLong(1), "name" : "Git" }
```

hibernate_sequences collection

```
{ "_id" : "hibernate_sequence", "next_val" : 2 }
```

Embedded objects and collections (1)

```java
@Entity
public class News {

    @Id
    private String id;
    private String title;

    @Embedded
    private NewsPaper paper;

    // getters, setters ...
}

@Embeddable
public class NewsPaper {

    private String name;
    private String owner;

    // getters, setters ...
}
```

```json
{
    "_id" : "1234-5678-0123-4567",
    "title": "On the merits of NoSQL",
    "paper": {
        "name": "NoSQL journal of prophecies",
        "owner": "Delphy"
    }
}
```

Embedded objects and collections (2)

- @ElementCollection with primitive types

```
@Entity
public class AccountWithPhone {

    @Id
    private String id;

    @ElementCollection
    private List<String> mobileNumbers;

    // getters, setters ...
}
```

```
{
    "_id" : "john_account",
    "mobileNumbers" : [ "+1-222-555-0222", "+1-202-555-0333" ]
}
```

- @ElementCollection with one attribute

```java
@Entity
public class GrandMother {

    @Id
    private String id;

    @ElementCollection
    private List<GrandChild> grandChildren = new ArrayList<GrandChild>();

    // getters, setters ...
}

@Embeddable
public class GrandChild {

    private String name;

    // getters, setters ...
}
```

```
{
    "_id" : "df153180-c6b3-4a4c-a7da-d5de47cf6f00",
    "grandChildren" : [ "Luke", "Leia" ]
}
```

@ElementCollection with @OrderColumn

```java
@Entity
public class GrandMother {

    @Id
    private String id;

    @ElementCollection
    @OrderColumn( name = "birth_order" )
    private List<GrandChild> grandChildren = new ArrayList<GrandChild>();

    // getters, setters ...
}

@Embeddable
public class GrandChild {

    private String name;

    // getters, setters ...
}
```

```json
{
    "_id" : "e3e1ed4e-c685-4c3f-9a67-a5aeec6ff3ba",
    "grandChildren" :
        [
            {
                "name" : "Luke",
                "birth_order" : 0
            },
            {
                "name" : "Leia",
                "birthorder" : 1
            }
        ]
}
```

@ElementCollection with Map of @Embeddable

```java
@Entity
public class ForumUser {

    @Id
    private String name;

    @ElementCollection
    private Map<String, JiraIssue> issues = new HashMap<>();

    // getters, setters ...
}

@Embeddable
public class JiraIssue {

    private Integer number;
    private String project;

    // getters, setters ...
}
```

```json
{
        "_id" : "Jane Doe",
        "issues" : {
                "issueWithNull" : {
                },
                "issue2" : {
                    "number" : 2000,
                    "project" : "OGM"
                },
                "issue1" : {
                    "number" : 1253,
                    "project" : "HSEARCH"
                }
        }
}
```

Associations

- Hibernate OGM MongoDB proposes three strategies to store navigation information for associations. The three possible strategies are:
  - IN_ENTITY (default)
  - ASSOCIATION_DOCUMENT, using a global collection for all associations
  - COLLECTION_PER_ASSOCIATION, using a dedicated collection for each association
- In Entity strategy
  - *-to-one associations
  - *-to-many associations

To-one associations

```java
@Entity
public class Vehicule {

    @Id
    private String id;
    private String brand;

    // getters, setters ...
}


@Entity
public class Wheel {

    @Id
    private String id;
    private double diameter;

    @OneToOne
    private Vehicule vehicule;

    // getters, setters ...
}
```

Unidirectional one-to-one

```
{
  "_id" : "V_01",
  "brand" : "Mercedes"
}
```

```
{
  "_id" : "W001",
  "diameter" : 0,
  "vehicule_id" : "V_01"
}
```

Unidirectional one-to-one with @JoinColumn

```java
@Entity
public class Vehicule {

    @Id
    private String id;
    private String brand;

    // getters, setters ...
}
```

```java
@Entity
public class Wheel {

    @Id
    private String id;
    private double diameter;

    @OneToOne
    @JoinColumn( name = "part_of" )
    private Vehicule vehicule;

    // getters, setters ...
}
```

```
{
  "_id" : "V_01",
  "brand" : "Mercedes"
}
```

```
{
  "_id" : "W001",
  "diameter" : 0,
  "part_of" : "V_01"
}
```

Unidirectional one-to-one with @MapsId and @PrimaryKeyJoinColumn

```java
@Entity
public class Vehicule {

    @Id
    private String id;
    private String brand;

    // getters, setters ...
}
```

```java
@Entity
public class Wheel {

    @Id
    private String id;
    private double diameter;

    @OneToOne
    @PrimaryKeyJoinColumn
    @MapsId
    private Vehicule vehicule;

    // getters, setters ...
}
```

```
{
  "_id" : "V_01",
  "brand" : "Mercedes"
}
```

```
{
  "_id" : "V_01",
  "diameter" : 0,
}
```

Bidirectional one-to-one

```java
@Entity
public class Husband {

    @Id
    private String id;
    private String name;

    @OneToOne
    private Wife wife;

    // getters, setters ...
}
```

```java
@Entity
public class Wife {

    @Id
    private String id;
    private String name;

    @OneToOne
    private Husband husband;

    // getters, setters ...
}
```

```
{
  "_id" : "alex",
  "name" : "Alex",
  "wife" : "bea"
}
```

```
{
  "_id" : "bea",
  "name" : "Bea",
  "husband" : "alex"
}
```

Unidirectional many-to-one

```java
@Entity
public class JavaUserGroup {

    @Id
    private String jugId;
    private String name;

    // getters, setters ...
}
```

```java
@Entity
public class Member {

    @Id
    private String id;
    private String name;

    @ManyToOne
    private JavaUserGroup memberOf;

    // getters, setters ...
}
```

```
{
    "_id" : "summer_camp",
    "name" : "JUG Summer Camp"
}
```

```
{
    "_id" : "jerome",
    "name" : "Jerome"
    "memberOf_jugId" : "summer_camp"
}
{
    "_id" : "emmanuel",
    "name" : "Emmanuel Bernard"
    "memberOf_jugId" : "summer_camp"
}
```

Bidirectional many-to-one

```java
@Entity
public class SalesForce {

    @Id
    private String id;
    private String corporation;

    @OneToMany(mappedBy = "salesForce")
    private Set<SalesGuy> salesGuys = new HashSet<SalesGuy>();

    // getters, setters ...
}
```

```java
@Entity
public class SalesGuy {
    private String id;
    private String name;

    @ManyToOne
    private SalesForce salesForce;

    // getters, setters ...
}
```

```
{
    "_id" : "red_hat",
    "corporation" : "Red Hat",
    "salesGuys" : [ "eric", "simon" ]
}
```

```
{
    "_id" : "eric",
    "name" : "Eric"
    "salesForce_id" : "red_hat",
}
{
    "_id" : "simon",
    "name" : "Simon",
    "salesForce_id" : "red_hat"
}
```

Bidirectional many-to-one between entities with embedded ids

```java
@Entity
public class Game {

    @EmbeddedId
    private GameId id;

    private String name;

    @ManyToOne
    private Court playedOn;

    // getters, setters ...
}


public class GameId implements Serializable {

    private String category;

    @Column(name = "id.gameSequenceNo")
    private int sequenceNo;

    // getters, setters ...
    // equals / hashCode
}
```

```java
@Entity
public class Court {

    @EmbeddedId
    private CourtId id;

    private String name;

    @OneToMany(mappedBy = "playedOn")
    private Set<Game> games = new HashSet<Game>();

    // getters, setters ...
}

public class CourtId implements Serializable {

    private String countryCode;
    private int sequenceNo;

    // getters, setters ...
    // equals / hashCode
}
```

Game collection

```
{
    "_id" : {
        "category" : "primary",
        "gameSequenceNo" : 456
    },
    "name" : "The game",
    "playedOn_id" : {
        "countryCode" : "DE",
        "sequenceNo" : 123
    }
}
{
    "_id" : {
        "category" : "primary",
        "gameSequenceNo" : 457
    },
    "name" : "The other game",
    "playedOn_id" : {
        "countryCode" : "DE",
        "sequenceNo" : 123
    }
}
```

Court collection

```
{
    "_id" : {
        "countryCode" : "DE",
        "sequenceNo" : 123
    },
    "name" : "Hamburg Court",
    "games" : [
        { "gameSequenceNo" : 457, "category" : "primary" },
        { "gameSequenceNo" : 456, "category" : "primary" }
    ]
}
```

To-many associations

Unidirectional one-to-many

```java
@Entity
public class Basket {

    @Id
    private String id;

    private String owner;

    @OneToMany
    private List<Product> products = new ArrayList<Product>();

    // getters, setters ...
}
```

```java
@Entity
public class Product {

    @Id
    private String name;

    private String description;

    // getters, setters ...
}
```

```
{
  "_id" : "davide_basket",
  "owner" : "Davide",
  "products" : [ "Beer", "Pretzel" ]
}
```

```
{
  "_id" : "Pretzel",
  "description" : "Glutino Pretzel Sticks"
}
{
  "_id" : "Beer",
  "description" : "Tactical nuclear penguin"
}
```

Unidirectional one-to-many with @OrderColumn

```java
@Entity
public class Basket {

    @Id
    private String id;

    private String owner;

    @OneToMany
    private List<Product> products = new ArrayList<Product>();

    // getters, setters ...
}
```

```java
@Entity
public class Product {

    @Id
    private String name;

    private String description;

    // getters, setters ...
}
```

Basket collection

```
{
  "_id" : "davide_basket",
  "owner" : "Davide",
  "products" : [
    {
      "products_name" : "Pretzel",
      "products_ORDER" : 1
    },
    {
      "products_name" : "Beer",
      "products_ORDER" : 0
    }
  ]
}
```

Product collection

```
{
  "_id" : "Pretzel",
  "description" : "Glutino Pretzel Sticks"
}
{
  "_id" : "Beer",
  "description" : "Tactical nuclear penguin"
}
```

Unidirectional one-to-many using maps with defaults

```java
@Entity
public class User {

    @Id
    private String id;

    @OneToMany
    private Map<String, Address> addresses
                  = new HashMap<String, Address>();

    // getters, setters ...
}
```

```java
@Entity
public class Address {

    @Id
    private String id;
    private String city;

    // getters, setters ...
}
```

```
{
  "_id" : "user_001",
  "addresses" : [
    {
      "work" : "address_001",
      "home" : "address_002"
    }
  ]
}
```

User collection

Address collection

```
{ "_id" : "address_001", "city" : "Rome" }
{ "_id" : "address_002", "city" : "Paris" }
```

Unidirectional one-to-many using maps with @MapKeyColumn

```java
@Entity
public class User {
    @Id
    private String id;
    @OneToMany
    @MapKeyColumn(name = "addressType")
    private Map<Long, Address> addresses
              = new HashMap<Long, Address>();
    // getters, setters ...
}
```

```java
@Entity
public class Address {

    @Id
    private String id;
    private String city;

    // getters, setters ...
}
```

User collection

```
{
  "_id" : "user_001",
  "addresses" : [
    {
      "addressType" : 1,
      "addresses_id" : "address_001"
    },
    {
      "addressType" : 2,
      "addresses_id" : "address_002"
    }
  ]
}
```

Address collection

```
{ "_id" : "address_001", "city" : "Rome" }
{ "_id" : "address_002", "city" : "Paris" }
```

Unidirectional many-to-many using in entity strategy

```java
@Entity
public class Student {

    @Id
    private String id;
    private String name;

    // getters, setters ...
}
```

```java
@Entity
public class ClassRoom {

    @Id
    private long id;
    private String lesson;

    @ManyToMany
    private List<Student> students = new ArrayList<Student>();

    // getters, setters ...
}
```

```
{
  "_id" : "john",
  "name" :"John Doe" }
{
  "_id" : "mario",
  "name" : "Mario Rossi"
}
{
  "_id" : "kate",
  "name" : "Kate Doe"
}
```

Student collection

ClassRoom collection

```
{                              {
  "_id" : NumberLong(1),         "_id" : NumberLong(2),
  "lesson" : "Math"              "lesson" : "English"
  "students" : [                 "students" : [
     "mario",                       "mario",
     "john"                         "kate"
  ]                              ]
}                              }
```

Bidirectional many-to-many

```java
@Entity
public class AccountOwner {

    @Id
    private String id;

    private String SSN;

    @ManyToMany
    private Set<BankAccount> bankAccounts;

    // getters, setters ...
}
```

```java
@Entity
public class BankAccount {
    @Id
    private String id;
    private String accountNumber;
    @ManyToMany( mappedBy = "bankAccounts" )
    private Set<AccountOwner> owners
                    = new HashSet<AccountOwner>();
    // getters, setters ...
}
```

```
{
    "_id" : "owner_1",
    "SSN" : "0123456"
    "bankAccounts" : [ "account_1" ]
}
```

```
{
    "_id" : "account_1",
    "accountNumber" : "X2345000"
    "owners" : [ "owner_1", "owner2222" ]
}
```

Summary

The Java Persistence API

- Entities
- EntityManager & the Persistent Context
- Persistence Units
- Exceptions
- JPA Query Language

FAQ

That's all for this session!

Thank you all for your attention and patient !