Programming Advanced Java

WEEK 5 - JPA

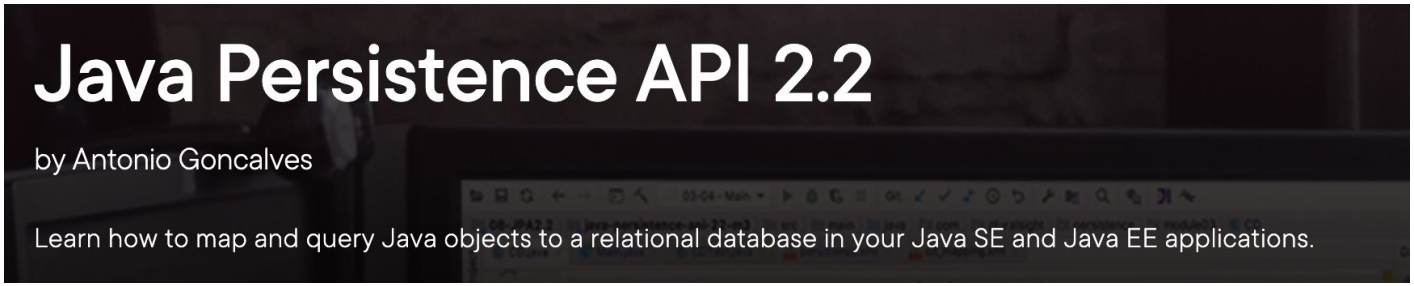
PXL DIGITAL

Goals

The **junior-colleague**

- can explain the benefits of JPA over JDBC.
- can describe the concept of ORM.
- can explain what JPA is, and what it's not.
- can denominate different JPA providers.
- can describe what a persistence unit is.
- can explain the fundamental interfaces of JPA.
- can explain what the persistence context is.
- can implement entity classes.
- can describe the entity objects' lifecycle.
- can implement different types of relationships between entity classes.
- can implement CRUD operations.
- can implement queries.
- can implement named queries.
- can explain, identify and solve the N + 1 query problem.

PLURALSIGHT



https://app.pluralsight.com/library/courses/java-persistence-api-21/table-of-contents

HOGESCHOOL PXL

Coding examples

Code: https://github.com/custersnele/JA2_introduction_JPA

Overview

1. Why not JDBC?
2. What is ORM?
3. What is Java Persistence API?
4. Different JPA implementations.
5. What is an Entity class?
6. What is a Persistence-Unit?
7. The bootstrap class Persistence.
8. Interfaces of JPA.
9. The JPA Entity Object Lifecycle.
10. Implementing CRUD operations.
11. Implementing relations.
12. Implementing queries.
13. N + 1 query problem.
14. Exercises.

Why not using JDBC?

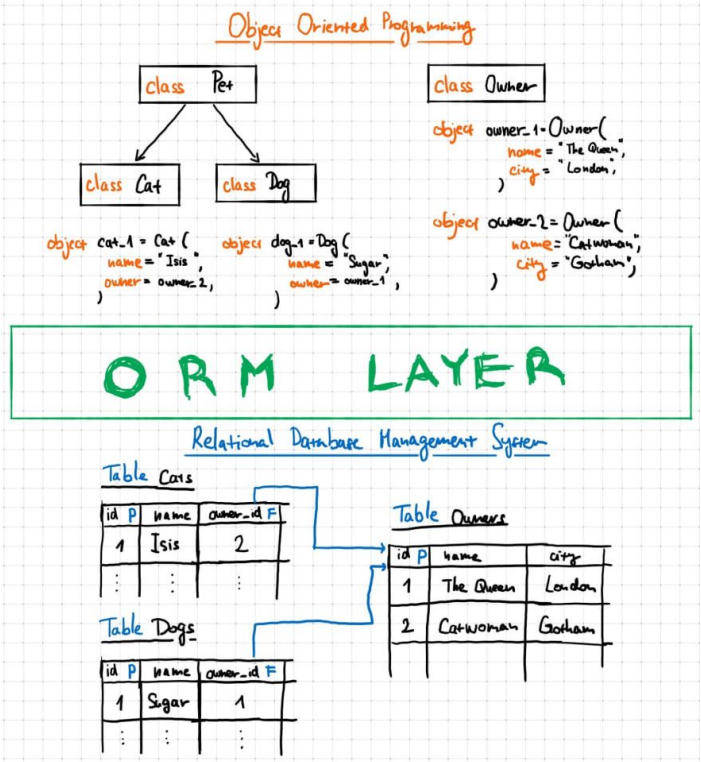
JDBC is a low level API.

You need to write a lot of boilerplate code.

Hard to read and maintain.

SQL is not easy to refactor.

Object Relational Mapping



HOGESCHOOL PXL

What is Java Persistence API?

- The Java Persistence API (JPA) is the Java standard for mapping Java objects to a relational database.
- JPA is one possible approach to ORM. Via JPA, the developer can map, store, update, and retrieve data from relational databases to Java objects and vice versa.
- JPA can be used in Java-EE and Java-SE applications.
- JPA is a specification and several implementations are available.

JPA Implementations

- **Hibernate** (https://hibernate.org/)
- **EclipseLink** (https://www.eclipse.org/eclipselink/)
- **Apache OpenJPA** (http://openjpa.apache.org/)
- **DataNucleus** (https://www.datanucleus.org/)

Other approaches are:

- MyBatis (https://en.wikipedia.org/wiki/MyBatis)
- JDO (Java Data Objects, https://www.baeldung.com/jdo)

Using Hibernate

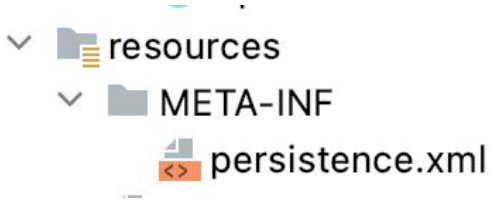
```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.4.29.Final</version>
</dependency>
```

What is an Entity class?

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class FootballPlayer {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  private String email;

  public FootballPlayer() {
     // JPA only
  }

  public FootballPlayer(String name, String email) {
     this.name = name;
     this.email = email;
  }

  // getters and setters
}
```

HOGESCHOOL PXL

What Is a Persistence-Unit?




```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             version="2.2"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
  <persistence-unit name="musicdb_pu" transaction-type="RESOURCE_LOCAL">
     <properties>
        <property name="javax.persistence.jdbc.driver"
                                              value="com.mysql.cj.jdbc.Driver"/>
        <property name="javax.persistence.jdbc.url"
                                           value="jdbc:mysql://localhost:3306/musicdb"/>
        <property name="javax.persistence.jdbc.user" value="user"/>
        <property name="javax.persistence.jdbc.password" value="password"/>
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.hbm2ddl.auto" value="update"/>
     </properties>
  </persistence-unit>
</persistence>
```

HOGESCHOOL PXL

Persistence: the JPA bootstrap class

Package javax.persistence

Class Persistence

java.lang.Object
    javax.persistence.Persistence

```
public class Persistence
extends java.lang.Object
```

Bootstrap class that is used to obtain an `EntityManagerFactory` in Java SE environments. It may also be used to cause schema generation to occur.

The `Persistence` class is available in a Java EE container environment as well; however, support for the Java SE bootstrapping APIs is not required in container environments.

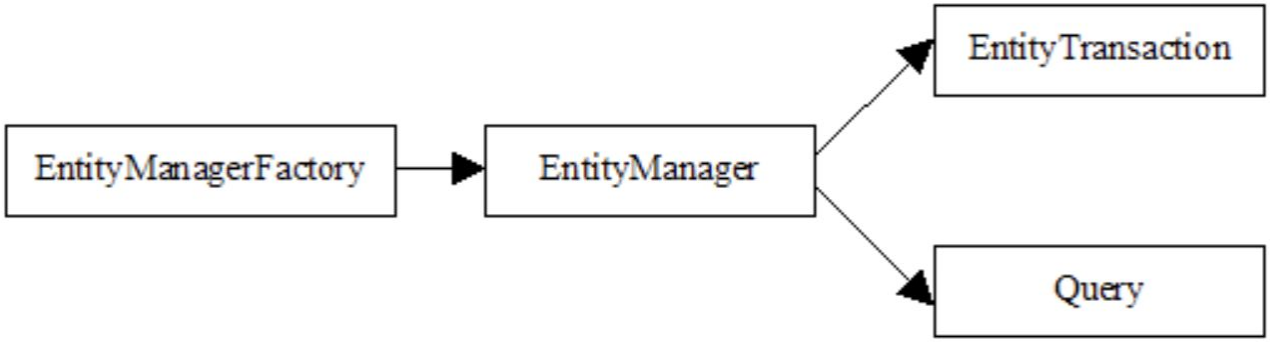
The `Persistence` class is used to obtain a `PersistenceUtil` instance in both Java EE and Java SE environments.

Since:

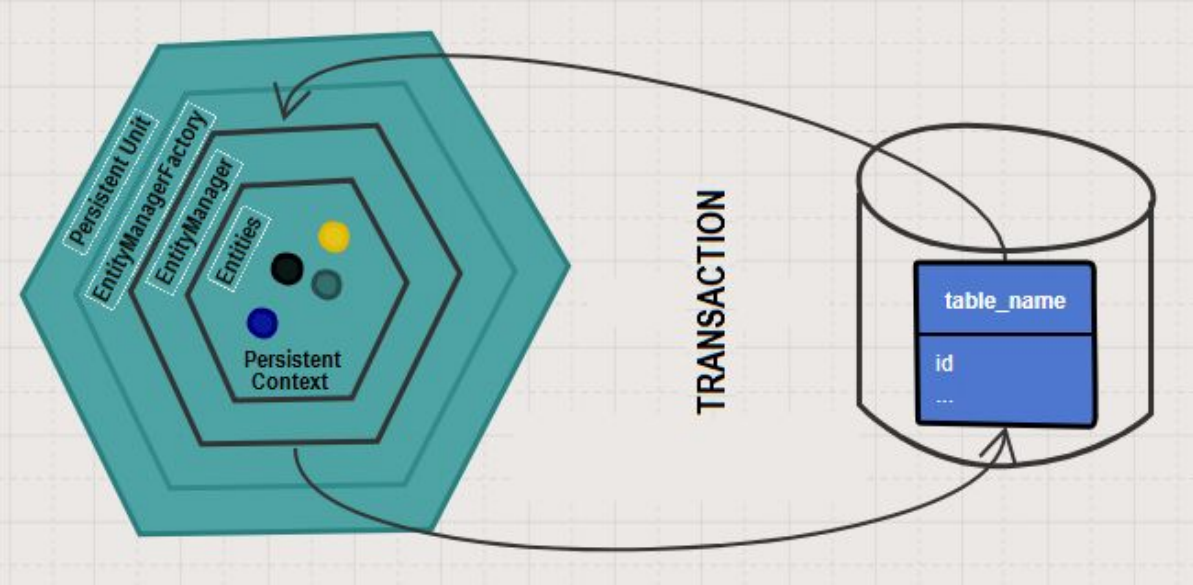
Java Persistence 1.0

HOGESCHOOL PXL

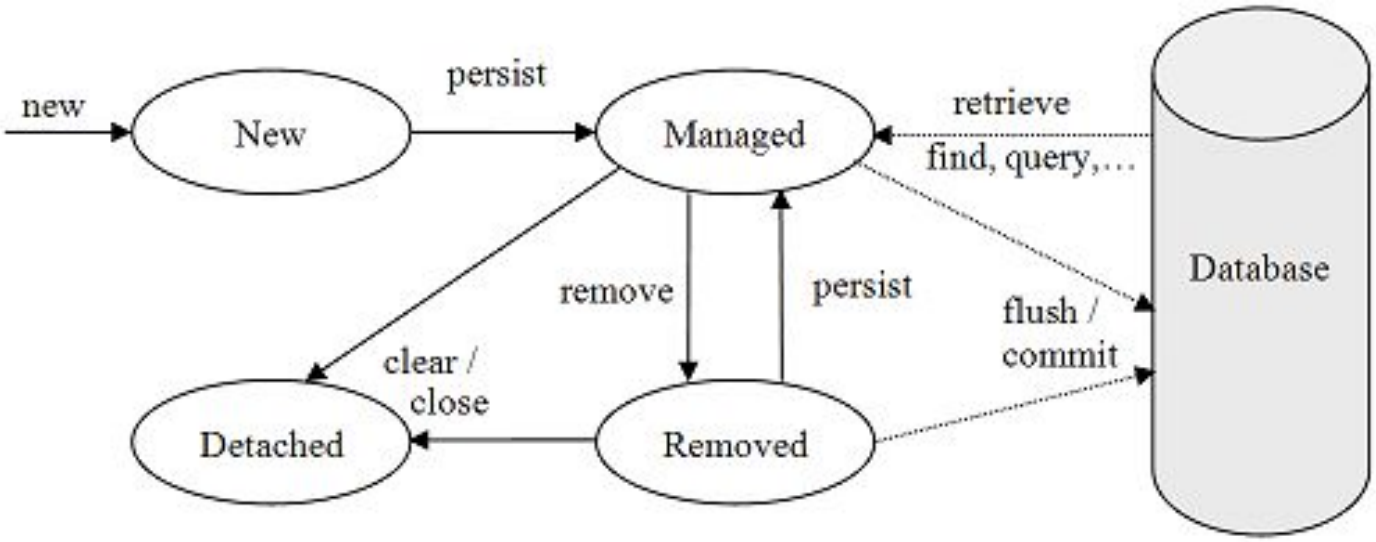
JPA interfaces



What is the Persistence Context?



JPA Entity Objects Lifecycle



CRUD operations

Demo1Find
Demo2Persist
Demo3Update
Demo4Remove

Transient fields

```
@Entity
public class FootballPlayer {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  private String email;
  @Transient
  private int shirtNumber;
```

```
Hibernate: insert into FootballPlayer (email, name, id) values (?, ?, ?)
```

DemoFootballPlayer

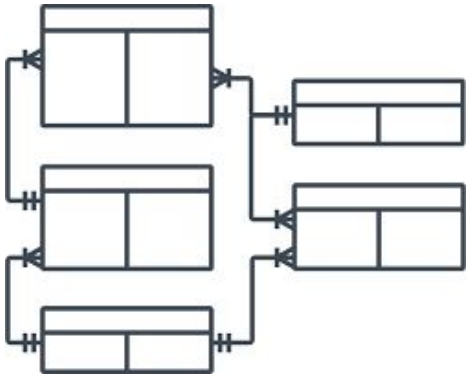
HOGESCHOOL PXL

Flush

```
1   transaction = entityManager.getTransacion();
2   transaction.begin();
3   EntityA a = new EntityA();
4   entityManager.persist(a);
5   entityManager.flush();
6   transaction.rollback();
```

Demo5Flush

Association mappings



Entity relationships

Multiplicity:

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many

Direction:

- Unidirectional
- Bidirectional

One-to-One Unidirectional

```java
@Entity
public class Patient {
  @Id
  @GeneratedValue
  private Long id;
  @Column(length = 40, nullable = false)
  private String name;
  @OneToOne(cascade = CascadeType.ALL)
  private MedicalRecord medicalRecord;
```

```java
@Entity
public class MedicalRecord {
  @Id
  @GeneratedValue
  private Long id;
  private double weight;
  private int height;
```

Demo6RelationsOneToOneUnidirectional

One-to-One Bidirectional

```
@Entity
public class User {

  @Id
  @GeneratedValue
  private Long id;
  @OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
  @JoinColumn(name = "cred id")
  private Credentials credentials;
  ...
```

Demo7RelationsOneToOneBidirectional
Demo7RelationsOneToOneBidirectionalUsage

Recommended reading

https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/

Associations: Many-to-One

```java
@Entity
public class Student {
  @Id
  @GeneratedValue
  private Long id;
  private String name;

  @ManyToOne
  private School school;
```

```java
@Entity
public class School {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  @OneToMany(mappedBy = "school", fetch = FetchType.LAZY)
  private List<Student> students = new ArrayList<>();
```

@NamedQuery

```
@Entity
@NamedQueries(
     @NamedQuery(name="schoolByName", query = "SELECT s FROM School s WHERE s.name = :name"))
public class School {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  @OneToMany(mappedBy = "school", fetch = FetchType.LAZY)
  private List<Student> students = new ArrayList<>();
```

```
TypedQuery<School> query = entityManager.createNamedQuery( "schoolByName", School.class);
query.setParameter("name", "PXL");
School result = query.getSingleResult();
```

Demo8RelationsManyToOne

HOGESCHOOL PXL

Associations: Many-to-Many

```java
@Entity
public class Reader {
  @Id
  @GeneratedValue
  private Long id;
  @Column(unique = true)
  private String subscriberNumber;
  private String name;

  @ManyToMany
  private List<Magazine> magazineList = new ArrayList<>();
```

Demo9RelationsManyToMany

HOGESCHOOL PXL

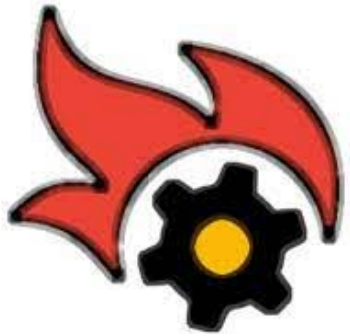
JPQL

```
Query query = em.createQuery("SELECT e FROM Employee e WHERE e.salary > 100000");
List<Employee> result = query.getResultList();

Query query = em.createQuery("SELECT e FROM Employee e WHERE e.id = :id");
query.setParameter("id", id);
Employee result2 = (Employee)query.getSingleResult();

// Query for a single data element.
Query query = em.createQuery("SELECT MAX(e.salary) FROM Employee e");
  BigDecimal result3 = (BigDecimal)query.getSingleResult();
```

Criteria API (optional)

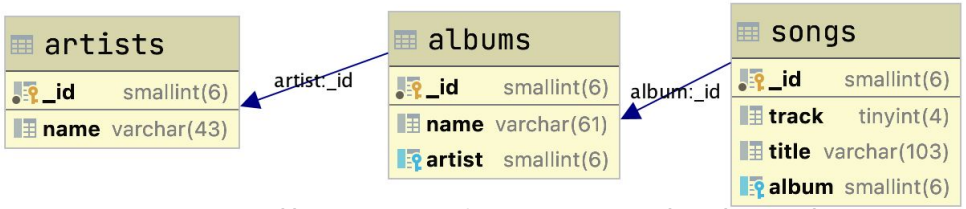
```
CriteriaQuery criteriaQuery = criteriaBuilder.createQuery();

Root employee = criteriaQuery.from(Employee.class);

criteriaQuery.where(criteriaBuilder.greaterThan(employee.get("salary"), 100000));

Query query = entityManager.createQuery(criteriaQuery);
  List<Employee> result = query.getResultList();
```

N + 1 query problem

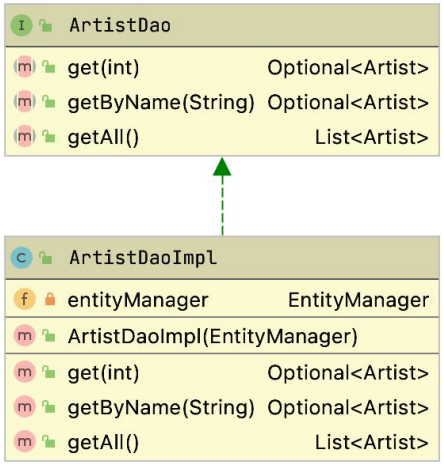


Demo10PostComments

Exercise 1



- Implement entity classes Artist, Album and Song with the above relations. Make sure all relations are bidirectional.
- Implement ArtistDao and ArtistDaoImpl.
- Optional: add unit tests for ArtistDaoImpl.
- Implement ArtistApp: given the name of an artists his albums are displayed. For each album all songs are displayed ordered by track number.



HOGESCHOOL PXL

Exercise 2

Adjust the program ExpensesOverview using JPA. Make sure all accounts and transactions are written to a database. Create a new table Category to store the transactions' category-property.