Programming Advanced Java

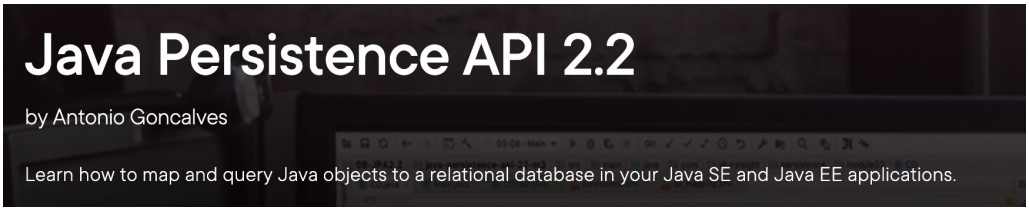
Chapter 6 - JPA

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/SampleJPAProject

HOGESCHOOL PXL

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.

HOGESCHOOL PXL

Why not using JDBC?

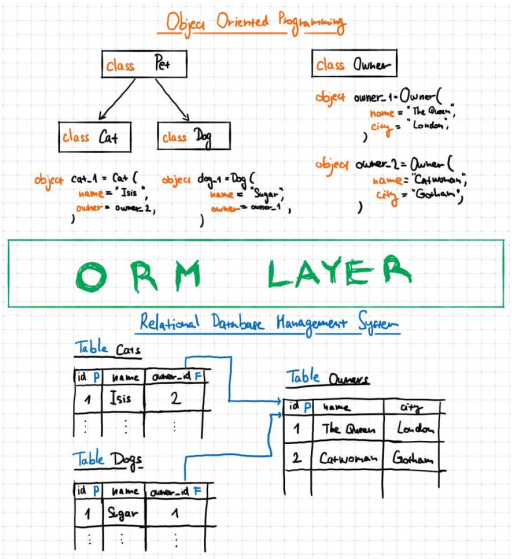
JDBC is a low level API.

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

Source:
https://dev.to/tinazhouhui/introduction-to-object-relational-mapping-the-what-why-when-and-how-of-orm-nb2

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.

HOGESCHOOL PXL

https://docs.jboss.org/hibernate/jpa/2.2/api/overview-summary.html
https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

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>5.6.5.Final</version>
</dependency>
```

HOGESCHOOL PXL

mvn dependency:tree

```
[INFO] +- org.hibernate:hibernate-core:jar:5.4.29.Final:compile
[INFO] |  +- org.jboss.logging:jboss-logging:jar:3.4.1.Final:compile
[INFO] |  +- javax.persistence:javax.persistence-api:jar:2.2:compile
[INFO] |  +- org.javassist:javassist:jar:3.27.0-GA:compile
[INFO] |  +- net.bytebuddy:byte-buddy:jar:1.10.21:compile
[INFO] |  +- antlr:antlr:jar:2.7.7:compile
[INFO] |  +-
org.jboss.spec.javax.transaction:jboss-transaction-api_1.2_spec:jar:1.1.1.Final:compil
e
[INFO] |  +- org.jboss:jandex:jar:2.2.3.Final:compile
[INFO] |  +- com.fasterxml:classmate:jar:1.5.1:compile
[INFO] |  +- javax.activation:javax.activation-api:jar:1.2.0:compile
[INFO] |  +- org.dom4j:dom4j:jar:2.1.3:compile
[INFO] |  +-
org.hibernate.common:hibernate-commons-annotations:jar:5.1.2.Final:compile
[INFO] |  +- javax.xml.bind:jaxb-api:jar:2.3.1:compile
[INFO] |  \- org.glassfish.jaxb:jaxb-runtime:jar:2.3.1:compile
[INFO] |     +- org.glassfish.jaxb:txw2:jar:2.3.1:compile
[INFO] |     +- com.sun.istack:istack-commons-runtime:jar:3.0.7:compile
[INFO] |     +- org.jvnet.staxex:stax-ex:jar:1.8:compile
[INFO] |     \- com.sun.xml.fastinfoset:FastInfoset:jar:1.2.15:compile
```

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

Objects live in memory. Entities live shortly in memory and are persisted in the database.

Field level annotations vs property (getter) annotations.

When field-based access is used, the object/relational mapping annotations for the entity class annotate the instance variables, and the persistence provider runtime accesses instance variables directly. All non-transient instance variables that are not annotated with the Transient annotation are persistent. (prefered)
When property-based access is used, the object/relational mapping annotations for the entity class annotate the getter property accessors, and the persistence provider runtime accesses persistent state via the property accessor methods. All properties not annotated with the `Transient` annotation are persistent.

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

Think to the *persistence-unit* as a box holding all the needed information for creating an `EntityManagerFactory` instance. Among this information, we have details about the data source (JDBC URL, user, password, SQL dialect, etc), the list of entities that will be managed, and other specific properties. And, of course, we have the *persistence-unit* transaction type, which can be `RESOURCE_LOCAL` or `JTA`.

The Persistence Units is also a grouping of user defined persistable classes.

Persistence units are defined by the persistence.xml configuration file.

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.

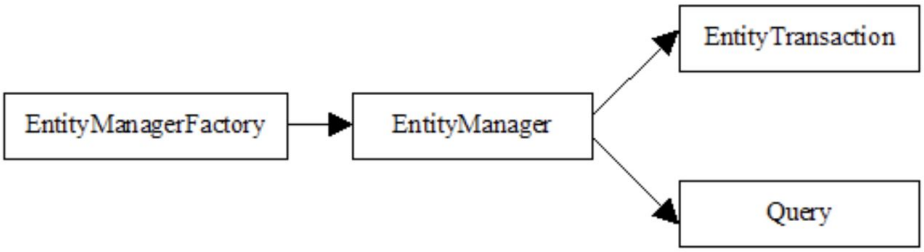
Since:

Java Persistence 1.0

HOGESCHOOL PXL

https://docs.jboss.org/hibernate/jpa/2.2/api/javax/persistence/Persistence.html

JPA interfaces



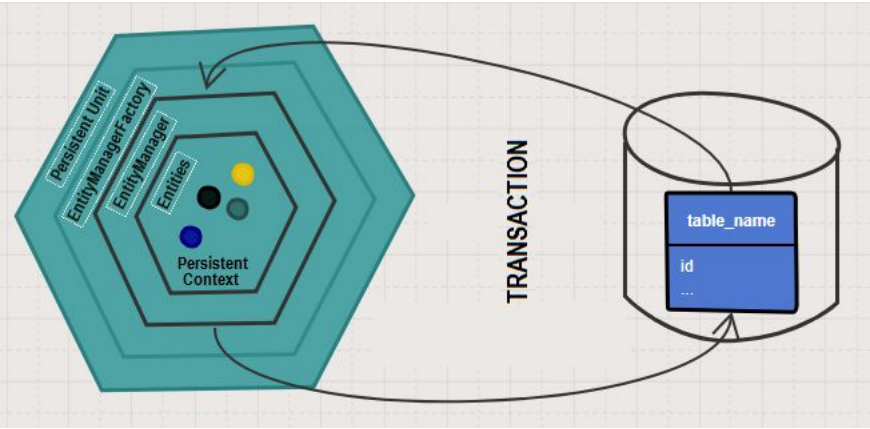
HOGESCHOOL PXL

As its name suggests, an `EntityManagerFactory` is a factory capable to create on-demand `EntityManager` instances.

Basically, we provide the needed information via the *persistent-unit*, and JPA can use this information to create an `EntityManagerFactory` that exposes a method named, `createEntityManager()`.

If we *close* an `EntityManagerFactory`, all its entity managers are considered to be in the closed state.

What is the Persistence Context?



HOGESCHOOL PXL

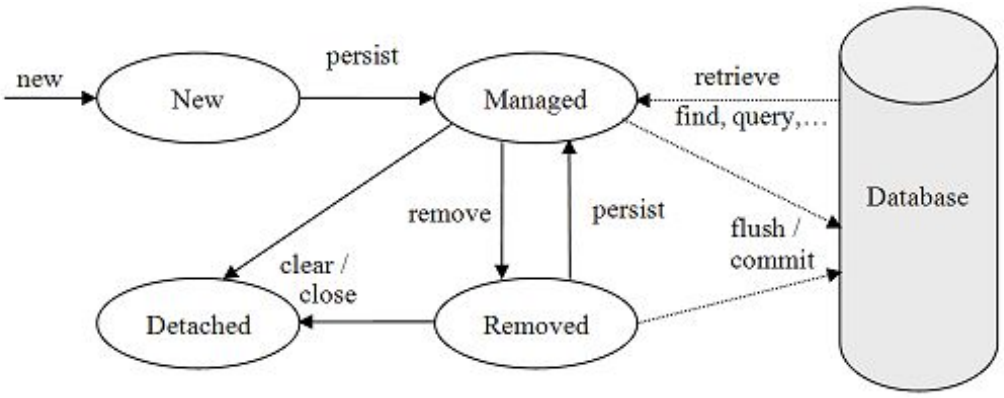
https://www.baeldung.com/jpa-hibernate-persistence-context

The persistence context is the first-level cache where all the entities are fetched from the database or saved to the database. It sits between our application and persistent storage.

Persistence context keeps track of any changes made into a managed entity. If anything changes during a transaction, then the entity is marked as dirty. When the transaction completes, these changes are flushed into persistent storage.

The *EntityManager* is the interface that lets us interact with the persistence context. Whenever we use the *EntityManager*, we are actually interacting with the persistence context.

JPA Entity Objects Lifecycle



HOGESCHOOL PXL

When an entity object is initially created its state is **New**. In this state the object is not yet associated with an `EntityManager` and has no representation in the database.

An entity object becomes **Managed** when it is persisted to the database via an `EntityManager's persist` method, which must be invoked within an active transaction. On transaction commit, the owning `EntityManager` stores the new entity object to the database.

Entity objects retrieved from the database by an `EntityManager` are also in the **Managed** state.

If a managed entity object is modified within an active transaction the change is detected by the owning `EntityManager` and the update is propagated to the database on transaction commit.

A managed entity object can also be retrieved from the database and marked for deletion, using the `EntityManager's remove` method within an active transaction. The entity object changes its state from Managed to **Removed**, and is physically deleted from the database during commit.

The last state, **Detached**, represents entity objects that have been disconnected from the `EntityManager`. For instance, all the managed objects of an `EntityManager` become detached when the `EntityManager` is closed. Detached objects can be merged back to an `EntityManager`.

More info:

https://www.objectdb.com/java/jpa/persistence/store

CRUD operations

PersistJPA
RetrieveJPA
UpdateJPA
DeleteJPA

HOGESCHOOL PXL

More info on: https://www.objectdb.com/java/jpa/persistence/crud

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 (?, ?, ?)
```

Flush

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

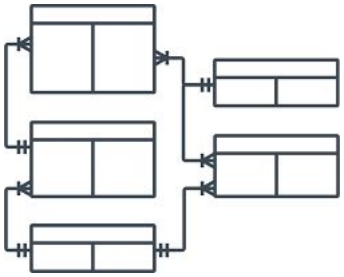
Demo5Flush

HOGESCHOOL PXL

When you call `session.save(a)` Hibernate basically remembers somewhere inside session that this object has to be saved. It can decide if he wants to issue `INSERT INTO...` immediately, some time later or on commit. This is a performance improvement, allowing Hibernate to batch inserts or avoid them if transaction is rolled back.

When you call `session.flush()`, Hibernate is forced to issue `INSERT INTO...` against the database. The entity is stored in the database, but not yet committed. Depending on transaction isolation level it won't be seen by other running transactions. But now the database *knows* about the record. What `session.flush()` does is to empty the internal SQL instructions cache, and execute it immediately to the database.

Association mappings



Entity relationships

Multiplicity:

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

Direction:

- Unidirectional
- Bidirectional

HOGESCHOOL PXL

https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate/

One-to-One Unidirectional

```java
@Entity
public class Researcher {

  private static final Logger LOGGER = LogManager.getLogger(Researcher.class);

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column(length = 40, nullable = false)
  private String name;
  @OneToOne(cascade = CascadeType.ALL)
  private ContactInformation contactInformation;
```

```java
@Entity
@Table(name = "contact info")
public class ContactInformation {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String phone;
  private String email;
  private String linkedIn;
```

HOGESCHOOL PXL

Associations: Many-to-One

```java
@Entity
public class Researcher {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column(length = 40, nullable = false)
  private String name;
  @OneToOne(cascade = CascadeType.ALL)
  private ContactInformation contactInformation;
  @ManyToOne
  private Project project;
```

```java
@Entity
public class Project {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
  private LocalDate start;
  @OneToMany(mappedBy = "project")
  private List<Researcher> researchers = new ArrayList<>();
  @Enumerated(value= EnumType.STRING)
  private ProjectPhase projectPhase;
```

@NamedQuery

```
@Entity
@NamedQuery(name = "ProjectsByPhaseAndMinNumberOfResearchers" ,
            query = "SELECT p FROM Project p WHERE p.projectPhase = :phase AND p.researchers.size >=
:numberOfResearchers")
public class Project {
   …
```

```
public List<Project> findByPhaseAndMinNumberOfResearchers (ProjectPhase phase, int
minNumberOfResearchers) {
  TypedQuery<Project> query =
           entityManager.createNamedQuery("ProjectsByPhaseAndMinNumberOfResearchers" , Project.class);
  query.setParameter("phase", phase);
  query.setParameter("numberOfResearchers", minNumberOfResearchers);
  return query.getResultList();
}
```

Associations: Many-to-Many

```java
@Entity
public class Researcher {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column(length = 40, nullable = false)
  private String name;
  @OneToOne(cascade = CascadeType.ALL)
  private ContactInformation contactInformation;
  @ManyToMany
  private List<Project> projects = new ArrayList<>();
```

JPQL

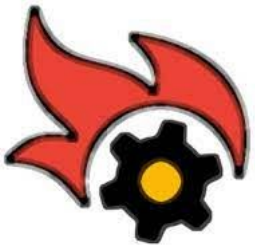
```
public Project findByName(String name) {
  TypedQuery<Project> query =
     entityManager.createQuery("SELECT p FROM Project p WHERE p.name = :name ", Project.class);

  query.setParameter("name", name);
  try {
     return query.getSingleResult();
  } catch (NoResultException e) {
     LOGGER.warn("No project found with name ["  + name + "]");
     return null;
  }
}
```

HOGESCHOOL PXL

https://en.wikibooks.org/wiki/Java_Persistence/JPQL

TODO: aanpassen naar TypedQueries!!!

N + 1 query problem

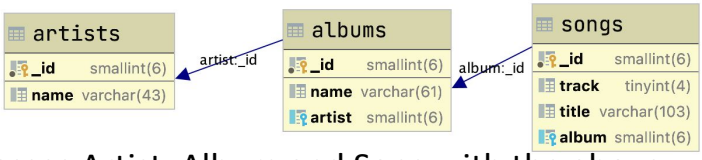


PostWithComments

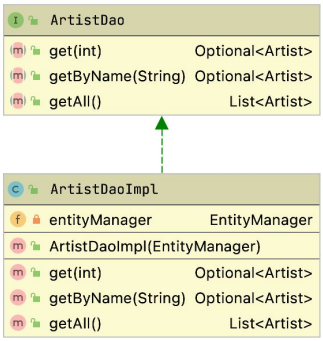
HOGESCHOOL PXL

Detailed information: https://vladmihalcea.com/n-plus-1-query-problem/

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