tutorial hibernate/jpa

This tutorial is provided to explain and highlight the main techniques

and advantages developed by JPA (Java Persistence API).

In order to use jpa within our application (we have totally implemented the middleware that manages the db connections via JAVA), it is necessary to introduce a framework that implements this ORM system.

**What do you mean with ORM ?**

ORM (Object Relational Mapping) is a programming technique that favours the integration of software systems adhering to the object-oriented programming (OOP) paradigm with RDBMS systems.

An ORM implementation provides, through an object-oriented interface, all the services related to data persistence, while abstracting the implementation features of the specific RDBMS used. His main goal is object-relational impedance mismatch resolution.

JPA is Java standard specification for ORM and we decided to use and analyze the implementation offered by the Hibernate framework.

**What is Hibernate?**

Hibernate ORM (Hibernate in short) is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate handles object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.

Hibernate's primary feature is mapping from Java classes to database tables and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

ENTITY EXAMPLES WITH HIBERNATE

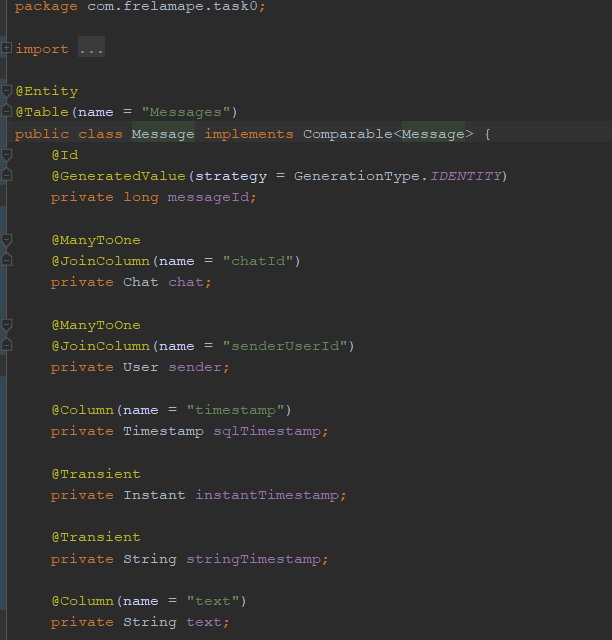


fig1: message class

This class has the task of managing the “messages” entity within the DBMS.

**@Id**: Marks the field as the key of the entity.

**@GeneratedValue(strategy=GenerationType.IDENTITY)**: Set the field to follow DB inner configuration.

**@GeneratedValue** in general it specifies the generation strategy and refers to the name of the generator.

**@Transient**: The fields marked as transient are not evaluated for persistence, just like the transient

keyword for serialization. In this example we have two fields which won’t be written on DB.

**@Column**: This annotation mentions the details of a column, if missing, the name of the field will be considered the name of the column in the table.

You can use **@basic** for not mismatching fields. A basic type maps directly to a column in the database. These include Java primitives and their wrapper classes, String, java.math.BigInteger and java.math.BigDecimal, various available date-time classes, enums, and any other type that implements java.io.Serializable.

At the bottom of the class we find all the constructors, getter and setter methods.

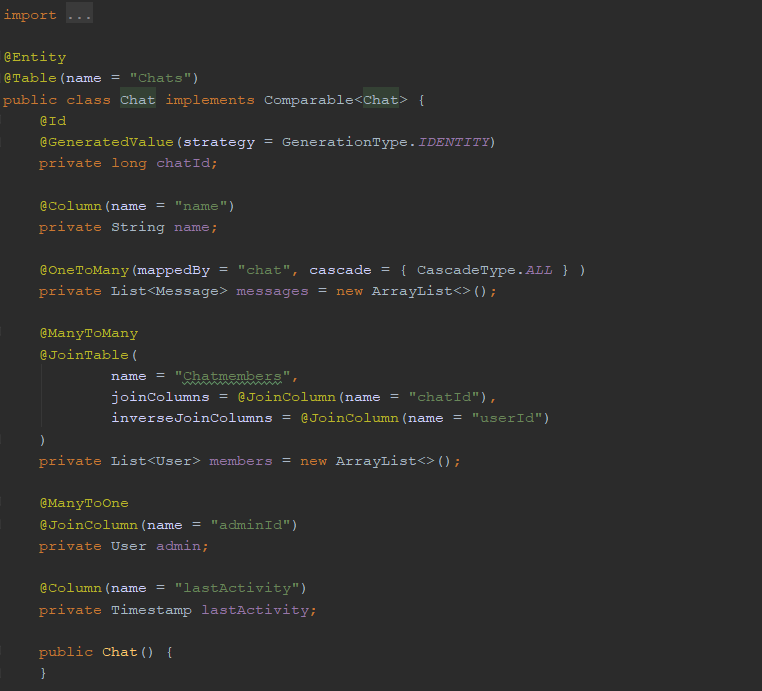


fig2: CHAT CLASS

As we can see, inside our Java classes we found different @ManyToMany, @OneToMany or @ManyToOne annotations.

This leads us to a proper discussion of the JPA Relationship Types.

JPA RELATIONSHIP TYPES

Main relationship types are:

* OneToOne
* OneToMany
* ManyToOne
* ManyToMany

ManyToMany



fig3: user class

A ManyToMany mapping describes a relationship between entities in which both can be related to multiple instances of each other.