Developer diary:

18.06.2025:

* Starting to split it into enitities.
* Newly learned self join
* Get into crows notation

26.06.2025:

Using jakarta.validation-api, contributes the interfaces, with the Hibernate implementation , Hibernate- validator fully programmed annotations. Since this is a bean validation I made a DTO Object Login Request, which I use to run the validation on. Jakarta Expression Language depndency is needed for dynamic String interpolation and without it, it didn’t work in my programm.

xx.xx.2025:

Discovered that if you use static methods of a class. Then you can use it through the whole application like a Java Bean in the Spring Context.

05.07.2025:

Discovered extra line of defense with a private constructor in a utility class to prevent instantion.

Still need the Throw for clarity, but it is not strictly needed.

private SceneManager() {  
 throw new UnsupportedOperationException(" This is meant as a utility class only.");  
}

06.07.2025:

Sets up Hibernate with the important points in plain Java:

1. In die POM

* Jakarte Persistance
* Hibernate-core
* Postgres

2. In die Resources eine Java Persistence XML:

is the central piece of configuration. That makes it one of the most important files of your persistence layer.

Hier wird auch die Persistance Unit bennant.

A **persistence unit** defines a set of configuration and metadata for managing entities and connecting to the database. It is declared here in the persistence.xml.

<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"

version="2.2">

**<persistence-unit name="my-persistence-unit">**

<class>com.example.model.User</class>

<class>com.example.model.Order</class>

<properties>

<property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/mydb"/>

<property name="javax.persistence.jdbc.user" value="postgres"/>

<property name="javax.persistence.jdbc.password" value="password"/>

<property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>

Ich muss sie dann verbinden wenn ich die Entity Manager Factory erstelle:

EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-persistence-unit");

3. Die Entity Manager Factory erstellen.

Zum Beispiel vorläufig in der Main:

*//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or  
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.*public class Main {  
 public static void main(String[] args) {  
 try {  
 EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("wifi-persistence-unit");  
 emf.close(); *// enough to trigger schema creation* } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 LoginView.*launch*(LoginView.class, args);  
 }  
}

Ohne dem erstellt er mir nicht die DB im PostgreSQL.

4. Die Klassen entsprechend annotieren.

08.07.2025:

Done self – referencing column in person entity.

CascadeType.PERSIST and CascadeType.MERGE. First means that the entity gets saved also the connected entitiy gets saved. Merge means the same with updated.

@ManyToOne  
@JoinColumn(name = "superior\_id")  
private Person superior;  
*//TODO try the subordinates with a set*@OneToMany(mappedBy = "superior", cascade = {CascadeType.*PERSIST*, CascadeType.*MERGE*})  
private List<Person> subordinates = new ArrayList<>();

Saving a Collection in the PostgresDatabase and the collection is also of enum elements:

Still person entity:

@ElementCollection(targetClass = Role.class)  
@Enumerated(EnumType.*STRING*)  
@CollectionTable(  
 name = "roles",  
 joinColumns = @JoinColumn(name = "person\_id")  
)  
@Column(name = "role")  
private Set<Role> roles;

@ElementCollection kennzeichnet mir das ganze als Collection vom Typ. In dem Fall role.

@Enumerated sagt speichere das als String.

@CollectionTable damit stelle ich die Normalform sicher. Hierin speichert er die Collection Elements. Ist eine eigene Tablle.

@Colum ist der Name der Tabelle