01: Entities and Context

- pom.xml
  dependencies: #hibernate-core #mysql-connector-java {#jaxb-api (java.xml.bind) #jaxb-impl #jaxb-core (com.sun.xml.bind)} obligatory in version 10 and above ... (of java)

persistance
<eclude-unlisted-class>false — מסיר את הצורך להכריז על ה-class של ה-entities
<provider>org.hibernate.jpa.HibernatePersistence.Provider

Entity

- @ID: every entity must ① Have or ② Inherit a primary key
  date: LocalDate type. from java.time.LocalDate supported with jpa 2.2+
  jpa 2.2: the version can be found in persistence.xml
  temporal: for jpa 2.2 and below
  @Table @Column in case the col/table name in the table doesn't match the one in the entity. if there is a match, I can ommit

Entity Manager

- in order to work with the entites, I need a Entity Manager
  זה כאילו אני רוצה לבחור שירות של סוכנים. אני לא אומר להם ישירות מה לעשות, זה ה-Manager שלהם
  לאמר/לחלק להם הוראות, אותו הדבר כאן. הוא לא מנהל אחד אחד, הוא מנהל Context שלהם
  the JPA implementation creating its own context — לא בכל framework
  context: Collection of instances, of managed objects
  managed objects: Entites — Entity Manager: Manage the context
  when I want entity manager to manage something ⇒ it must be put in the context first (the Entity Manager do it by persist())

Entity Manager Factory

- Create the Entity manager: By using the class Persistence method and my persistence unit-name
  Transaction: Every operation that involve a change in the DB should involve transaction → getTransaction().begin, getTransaction().commit()
  Exception: I will use try catch and in case of exception I will rollback()
  We can work with multiple Entity Manager

persist() ! it's not insert to the DB action, it's adding the instance to the context

example

SQL actions

Insert 100
delete 100
insert 101

} 3 actions in the DB

Persist

persist(100)
delete(100)
persist(101)

} 1 action to the DB

why?

persist(100) add to the context . delete(100) remove from the context so I left with empty context. persist(101) add to the context, so I have one object in the context and only that 1 instance is written to the DB

02: @GeneratedValue

2 attributes

- Strategy
- Generator

Strategy

- AUTO: Hibernate choose for me which Generation method to use, according to my DBMS (He chooses one of the 3).

☆ Using Integer id vs int id
Both are ok. Integer allow null primitive type int doesn't allow null.
if a value isn't provided I will have zero in my table and I will never tell the diffrence between a real zero and a null zero

- IDENTITY: Auto Increment in SQL. Managed by the DB

In which cases I won't use IDENTITY

① If I don't want the DataBase Managment System to generate the ID
② if The DBMS doesn't allow Auto Increment (Oracle)

So... what options do I have?

/* if the table was created before the entity - the ID shouldn't be defined as auto increment */

- TABLE: Having a table that store the values that will be used as PK
- SEQUENCE: Having a sequence as an Entity in the DB // not supported by MySQL

- TABLE

① Creating a table in the DB with 2 columns. name=key_generator
① First Col: Sequence_name: String (it's my PK) } ← defults names
next_val: int
③ telling hibernate whats the name of the table in my DB that holds the ID's. this is done by generator = "tablename"

When I will add data to my table I will have 2 quries that generate a row in the table_id and which generate my ID, and then I will have another query to add the data, assigning the id from table_id

key_generator will always have one row

} for it to work I must use the table name and col name that indicated here

When I want to use diffrent names:

@TableGenerator (name = the name of my generator as defined in generator...
table = the name of the table in the db
pkColumnName = the name I gave to the sequence (String) column
valueColumnName = " " - next_val (int) column

- SEQUENCE: like a TABLE. I need to define a generator name
@SequenceGenerater: allow to customize the sequence
If I use a technology that allow sequence, it's better than TABLE

3° @Enumerated @Temporal

Enumuration Class

doesn't have methods, attributes. Has a private constructor
in a database table it can take the form of a: ① String - the value
② Integer - Ordinal number

@Enumerated

the default is ordinal, if I want to custm it:
@Enumerated(value = EnumType.STRING)

Disadvantages

Ordinal: if someday I will modifie my Enum class and I will change the order of the enums or insert new one in the middle it will mess-up all the previous data.

JPA 2.2

before version 2.2 I could use only the Date class from java.util
From v2.2↗ I can use JavaTime (LocalTime, LocalDate...)

X @Temporal

when using LocalDate etc + working with JPA 2.2 and above
I don't need to use the @Temporal

SQL TimeStamp = Java LocalDateTime
LocalDate: setting the actual date ⇒ LocalDate.now()
LocalDateTime: setting the actual date + time ⇒ LocalDateTime.now()
ZoneDateTime: Should save in the DB the hour and the time according to the time zone that I provide as argument. But actually it always save the time zone of where the server is
Example: ZoneDateTime.now(ZoneID.of("Europe/London")) will yield the time of where I am and not the time in London

@Temporal

Before 2.2 I was obligated to use @Temporal with dates and Time
The object I deal with is of type Date

Date - Object
getTime(); give you the time that passed since 1.1.1970 in milliseconds
When using with JPA → I must tell the JPA what I want to extract from that object using the @Temporal and TemporalType

DATE TIME TIMESTAMP

Setting the value
object.setDatex(new Date());

if
if in the table my col defined as timeStamp I can use temporal type timestamp, Date. But it doesn't work with Time

TimeStamp x = new TimeStamp(System.currentTimeMillis()); => 2021-03-04 19:15:13.032

4: @Embeddable , @AttributeOveride, Composed PK

@Embeddable

when some column in my table describes another entity but this "entity" is not really an Entity cause it doesn't has a real table in the DB instead of @Entity it will be declared as @Embeddable
the perfect example -> adress

id	company_name	no_employé	capital	Street	city	ZipCode

code:
```
@Embeddable
pablic class Adress
```
⇐ they define an adress "entity" not a company entity

@Embedded

according to the previous example, in my company class "insert" the adress as a unit

code:
```
@Embedded
private Adress adress
```

Setting an Embbeded type

① object.setEmbbededType(new EmbbededType()) -> company.setAdress(new Adress());
② object.getEmbb..setAttribue("...") -> company.getAdress().setStreet("...");

@AttributeOverride

what if in my adress class I have: StreetNo, StreetName, ZipCode
and there are 3 classes that uses this embeddable class but each of them has a column name that doesn't match my Adress var
table 1: Street_no, name, zip table 2: st_name, zip_code table 3: street_n, zip_cd
Do I change all the column names to match the embeddable Class? -> NO
I will use @AttributeOverride to override the att of columns in case of embeddable and Composite PK

@AttributeOverride(name="StreetName", column=@Column(name="st_name"))
this in embbedable class refer to that in the table

What if I have more than one override to Do?
Java 8↗ JPA 2.2↗ let me repeat the @AttributeOverride so every Coloumn that need to be modified can have it's own @AttributeOverride

But if it's not the case and I don't have 1 of those?

```
@AttributeOverrides ({
        AttributeOverride (no 1°); -> פסיק
        AttributeOverride (do 2°) -> לא צריך פסיק })
```

Composed Pk

2 ways to do it

Entity//

```
@Entity   @IdClass(EntityPk.class)
public class Entity
  @ID
  type_var1Pk          My Entity
  @Id
  type_var2Pk          egal
```

\+ + + + + + + + + + + + +

Pk//

```
public class EntityPk implements Serializable
  type var1Pk
  type var2Pk
```

1. IdClass(EntityPK.class) -> [illegible]
2. every Pk compenent get @Id
3. Creating a Pk class with the Pk att which must implement Serializable

```
@Embeddable
public class EntityPk implements Seri...
  type var1Pk
  type var2Pk
```

\+ + + + + + + + + + + + +

```
@Entity
  public class Entity
@EmbeddedID
  private EntityPk id;
```

1. Creating a Pk class with the primary keys att which must implement Serialziable and @Embeddable
2. @EmbeddedID in the Entity Class

@Column	customize	@AttributeOverride
obj.setvar1PK(....)	Set	obj.set EntityPk (new EntityPk) obj.get EntityPk.setvar1PK(...)

5: @SecondaryTable @OneToOne

@SecondaryTable

- it is used to connect tables but not Entities. I can link table x with Entity y without having nor Entity nor Embeddable for Table x
- what's surprising is that I can save data to table x by using the setters and getters of my Entity. Example
  1 Entity User, 2 Tables without Entity: adress and friend
  @Entity
  @SecondaryTable (name = tablex, pkJoinColumns = @PrimaryKeyJoinColumn ⇨
  (name = the name of the column in table x that I wanna store the pk of the entity)
  .... clas ....
- @Column(table=tablex)
  varxoftablex
  ⇩
  Entity.setVarXoftablex

pkJoinColumns: add the Entity ID to a column in table x.

with @SecondaryTable I have inside my Entity -attributes that comes from diffrent table(s) and affect the diffrent table(s)
It's not a JPA relationship - it's a DB relationship between tables
I can have more than one @SecondaryTable

@OneToOne

tablename ↑ pkname ↑

- Naming Convt of Fk col: JPA expect a certain way of naming → tablename_id
- relationship in JPA can be represented in 2 way: bi-directional / uni-directional
  unidirectional: I will store the relationship in the class that contains the Fk
  bi-directional: the class that doesn't contain the Fk get mappedBy. The owner has FK
- @JoinColumn(name = columnname): in case that my column that hold the Fk doesn't doesn't follow the naming convention

- MappedBy
  This class won't get any FK. if it would get one - it redundant - I already has a Fk that map to this direction of the relationship
  Owning: means that I have 2 owners in the relationship. NOT POSSIBLE

Relationship attributes

optional = If I allow (true, the default) or don't allow (false) null in this relationship (=> fk = null). If it is null -> property value exception / only in many2one

fetch = eager is the default in 1TO1 / Many2one. Lazy with collections

- eager: when I select the entity I select also all the data that is bound by relationships
- lazy: I select only the entity without the entity that is linked. no Join is used
- uses join => to link the 2 tables

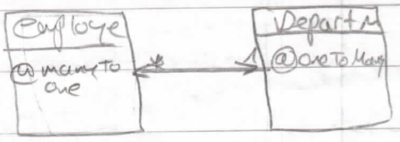
targetEntity = the entity.class in which we in relation... [illegible]

cascade = Persist. take off the need to persist linked instances.
if I persist for example a user - I add it to the context, if I want to link it to an adress I need to add also the adress to the contex. By chosing cascade.persist in the user Entity I say: whenever you persist a user, persist also the adress without me doing so

mappedBy: for bi-direction relationship

orphanRemoval: default false. in oneToMany but doesn't exist in @ManyToOne it's related to EntityManager.remove().

6: @OneToMany @ManyToOne



Use case: 1 employe work in one Department
1 Department has many employee
It important to know how it behave as uni-direction.

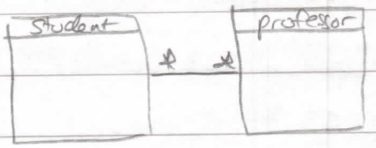
uni-direction → @JoinTable => Avoid it
The JPA let me the option not to store a FK in none of the relationed tables and to store it in another table. I will show it but avoid it;
@oneToMany: Should be on the 1 side, the owner, followed by a List/set/collection
using @oneToMany as a uni-direction require the creation of another table that will hold the fk. It doesn't sound logic cause a join table we create in manytomany. But if I work with an existing tables that I'm not able to modify?

Bi-directional => @JoinColumn
@oneToMany in the 1 side
@ManyToOne in the * side → it will get the FK

The Many side must be the owning
@JoinTable: JPA specification expect @JoinTable in 1ToMany uni-directional
why? Cause JPA doesn't know where is the FK and It doesn't proived option to specify it
@ManyToOne
the manySide must be the owning side hence he can't have the mappedBy it will be the one that hold the FK
Name of FK: must be exctaly as follow. if it's not exactly the same in my table
→ @Column(name=...). The name is (name of the entity var)_(the name of pk)
~~if I try~~ to
@JoinColumn = customize a fk column name

Exception: Table zzz.xxx_yyy doesn't exist!
why? if I have bidirectional @oneToMany and on the other side I have @ManyToOne - and I forgot to put @OneToMany(mappedBy){on this side
Since the @ManyToOne is the owner} => Hibernate will not know who is the owner and will take @oneToMany side as the owner => which require an Intermediate Join Table. It will consider it @oneToMany uni-directional

4° @ManyToMany



Result in a Join Table that stores the id

Uni-directional

proffesor know about his students, students dont know about thier proffesor
the proffesor is the owner

owner	Join Table	non-owner
@ManyToMany	owner_nonowner	—
Collection<Type>	(proffesor_student)	

ID naming Convention: the var name_id (List<Student> eleves => eleves_id)
What if I dont follow the naming convention? my join table has another name and also my columns

```
@JoinTable(name = tableName , joinColumns = @JoinColumn(name(name = owner),
                                inverseJoinColumns = @JoinColumn(name = non-owner))
```

Bi-directional

owner		non-owner
@ManyToMany	don't forget to add to the collection sides or both	@ManyToMany (mappedBy)
Collection<non-owner Type)		collection<owner Type>

Remember
@JoinTable @JoinColumn => only in the owner Entity
mappedBy => only in the non-owner Entity

In ManyToMany any of them can be the owner of the relationship and it doesn't impact the location of the fk since it an external (Join Table)
So I need to remember: the class that will get JoinTable will never get "mappedBy" and vise versa

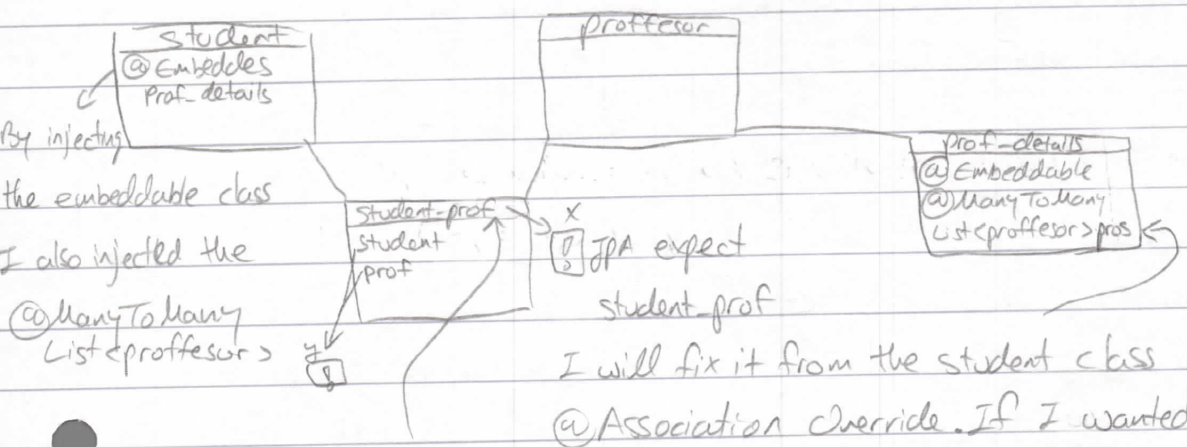
8. @AssociationOverride @ElementCollection

- **@AssociationOverride**: customizing OUTSIDE the relationship
  it's like @AttributeOverride but for columns that are part of relations
  @AssociationOverride (name = the var name, joinColumns = @JoinColumn = th column name
  @joinColomns is for a relation column
  @Colomns is for a "normal" column

@ManyToOne
→ joinColomns

@OneToMany (uni) / @ManyToMany
→ joinTables

- in this example we created an embeddable class and we put the relation inside this class. it's bizzare and it's add couplixity for understaning so putting relation in embbedable class is not advisable but is needed to demonstrate the @AssociationOverride



I will fix it from the student class by using (from outside) @AssociationOverride. If I wanted to fix it from the pro_details class (from inside) I would have use @JoinTable

Fixing x:
in the student class:
@AssociationOverride (name = "pros", JoinTable(name = "student_prof")

Fixing y:
the jpa expect student_id and prof_id so inside the previous
@AssociationOverride (... , joinColumns = @JoinColumns(name = "student")
(non-owner FK) ← inverseJoinColumns = @JoinColumns(name = "prof")

- Outside the Relationship
  it's mean that I modify a relationship FROM a entity that doesn't own directly the relationship, it gets it by injection (@Embedded)

Remember: When working with Embeddable Class
Association Override: For relationship
Attribute Override: For Simple Fields
try to avoid AssociationOveride cause it's ->->3 ->01

@ElementCollection

relationship between Entity and non-Entity
I can't use @oneToone @ManyToMany etc ... they R Entity-Entity relationship and I have Entity-NonEntity relationship
@ElementCollection: will replace them
JpA expect that the name of the non-Entity table will be entity_non-entity var name.

Customizing table name
@CollectionTable replace The @JoinTable (works only with Entity-Entity relationship)
ex: @CollectionTable (name="nameOfTheTable", joinColumns=@JoinColumn(name=
ref to the non-Entity table <- -> thenameoftheFKsColumn
represent the FK

@Entity ...
@ElementCollection
private List<String> phoneNumbers; <-
Here I need to add @Column (name= the name of the col that represent)

Working with Objects
if it's only one column -> I can use List<String> but if it's more I need to put an List<objectName> when the Object has a @Embeddable class

9% Maps

- Map: key + value
  unique
  in JPA the ref is the value not the key
  Do I use
  Do I use relations or Element Collection with maps
  if the value is Entity => relation
  if not (String, int...) => Element Collection

@MapKeyColumn @MapKeyTemporal ...

- @MapKeyTemporal -> for the key normal @Temporal -> value

10. Inheritance @MappedSuperClass

How to inherit? Simple => extends
@Id should be written only once in the mother Entity
How do I represent it in tables? one table with all the attributes or seperate tables? ...
In JPA we have 3 strategies that are defined in @Inheritance
where: at the mother class. @Inheritance(strategy=InheritanceType...)
strategies: ① SingleTable ② Joined ③ TablePerClass (rarely used)
Default: SingleTable

.SingleTable

one column To show the type of the Entity => Discriminate Column

Id	name	color	dtype → its not a variable
1	mitzi	white	cat
2	hasmi	black	dog

Disadvantage: I store the Entity type
Advantage: Everything is in one table => make the search easier

.JOINED

Each entity has its own table. the relation to the mother class is done by an id column in the daughter class and when I want to retrive all the data I use this Id in my query to make a Join to get all the data about the child. ex: 1 Product Entity 1 chocolate Entity

id	name
1	productName
2	Velrona

Product Table

type	Kcal	id
White	350	2

I have no Id Here. this is not the table id
chocolate Table

it's this.

Advantages: I don't need a discriminator column
disadvantage: Query might be slower. I must join 2 tables. For getting for example the Data: velrona, white, 350 I need to join the chocolate table with the product table (which hold the data or the chocolates name)

10° Continue

the JPA states that the implimintations don't have to support this type.
Don't use It. I'm talking about the third strategy...

TABLE_PER_CLASS
all the tables/entites has all the fields
for ex: cat extends Animal
@Animal: Id, name
@Cat: Id, name, color

AUTOIncrement: isn't supported for this strategy. I can't use it => exception
they can't have the same ID cause they are part of the same hierchy
I need to set the ID myself

@MappedSuperClass

the mother class is an abstract class. I will use it wher I don't want to repeat myself. And I will mark it as @MappedSuperClass
the mother class is not Entity. Cause Entities must be able to be instances in the context. Abstract class can't have instances and therefor can't b entity
I will create my @Id in the abstract class -> like that It will be inherited and won't be reapeted
What else to put: I will put inside all the att. in common to all the class that will inherit it

3 Types of "entities" know by JPA

@Entity
@Embeddable
@MappedSuperclass
those are the 3 @annotation I use above a class to specify the kind of Entity the class is.

All the classes that extends
I can't use @Inheritance with @MappedSuperclass. Why?
Inheritance work only with between 2 or more @Entity

11: Entity Manager

working with Framework like Spring → we won't see the Entity Manager and if we see it, we won't see how it has been created

methods - summery / works only with class that are @Entity

#1 persist(): make the instance part of the context, managed by JPA
then the JPA implemantation can work with that instance (put it in the DB)

#find(): retrive an instance of an Entity from the persistance Layer (the DB) by using its primary key
#getReference(): similair to find
#contains(): Wheather an instance is in the context

#detach(): take out of the context
#clear(): take out all the instances out of the context
#remove(): Delete an instance

#merge()
#refresh(): take data from the DB and undo any modificatio I made in the context
#flush(): force the entity to enter the DB before transaction commit

Persist

in pure JPA it doesn't mean insert to the DB. it's Hibernate who does it
with EntityManager.getTransaction().commit();
relationship: if the entity that I persist is in relationship with others Entities
I need to persist the other Entity/ies also or use cascade.PERSIST

Find

2 Provide: ① Entity.Class ② ID } Product p = em.find(Product.class, 10)
returns: An object of type Entity
For non R in CRUD I don't need a transaction from the entity Manager
if the ID doesn't exist I won't get an exception, I will get a null
What Find does: SELECT from the DB and put it inside the context
when using find I may consider using if(x==null)

GetReference

Product p = em.getReference(Product.class, 10)
if I dont use p (System.print.out(p) or some action) the query (select... from) won't be excuted, it's like a lazy find()

getReference give you a proxy. If that proxy isn't use the query won't run
which query? the SELECT query that is done by hibernate to find what I want

Contains

wheater the instance is in the context
return true if the instance is from the begining in the context / was persisted (not merge)

detach and clear

take an instance out of the context without Deleting it from the DB
detach: take this specific instance out of the context
clear: empty the context. takes all instances out of the context

Remove

Detach the instance and delete it from the DB
Works only on managed instances (that are inside the context)
I will find() → than I will remove(object)
When I use find(), I put it inside the context and that's why remove() works

Merge

Merge: Taking an instance from the DB which is not in the context right now
and insert/merge it into the context
It's like persist no? Both of them take an instance and put it in the context
Not really. persist I will use with an instance that doesn't have record in the DB

Refresh

update the instance with whatever we have in the DB. it reset no matter
what value the instance has in the context, to its values in the DB

For updating queries

1. Look for the record with find()
2. use setters on the returning object

12: JPQL

- it's like SQL but it quering on objects on not on DB

Select

SQL: SELECT * FROM product (native query. can change between diffrent DBMS)

JPQL: SELECT p From Product p (not DBMS dependante)

class name start with Caftal letter

① String jpql = " ..... "

② Query query = em.creatQuery(jpql) => the method return a query

TypedQuery<Object> query = em.createQuery(jpql, Object.class)

→ object type

③ List<Object> list = query.getResultList();

- to print:

list.foreach(system.out::println)

Working with parameters

my parameter name

jpql = "select p from Product p where p.price > :price"

object type obj name attribute

...

query.setParameter("price", 10) → value

where price > 10

What if I want to get just one column?

① Select p.name from Product p

- BUT I want get an object, I will get a String, int.... So I need to change TypedQuery<String> query = ...(jpql, String.class) List<String> and it will be OK

What if I want to get more than 1 but not all?

it will return as a map, as a collection → transform to objects

Functions

Select sum(p.price) From product p

of what?

when I expect single result I should use getSingleResult() ex:

- double sum = q.getSingleResult();

namedQuery

it's a query that I write in the @Entity class

@NamedQuery(name=queryName query="Select...") ⇒ Entity Class

em.createNamedQuery(queryName, typeOf the Object) ⇒ in the bean

JPA and @NamedQuery and Errors in named quries

if I write something wrong in my name query the app won't run. even if I never use it. Why?

JPA will load and validate /process by the JPA inplementation when the app START

Using numbers as parameters

jpql = "Select p from Product p where p.price > ?1"

... ⇒ q.setParameter(1, 10.0) סימן מקום זה כמו ה- 1 הנה price, יקבלו

nativeQuery

כך נראים שאני כותבת שאילתא מאוד ספציפית. מה הכוונה? אם אני כותבת ב - sql query

אם יום אחד נשנה את ה- DB ל- Oracle ⇐ יהיה לנו בעיה ולא נצטרך/ לשנות את המעבר

אם לא הייתי משתמשת ב- native Query, הייתי משנה רק את ה dependency ואת ה persistance.xml

והכל היה עובד נקי.

native Query works with Query but not with TypedQuery

Remember: Avoid Native Query. Use it only if u have no other option

Quries with Relations

1 (Department) to Many (employee)

jpql = "select e From Employee e, Department d where e.department = d

(Employee e: owner; Department d: non-owner; e.department: the ref to the non-owner in the owner's Entity; e.department = d: Join* comparing objects)

And d.id = 3" (d.id = 3: condition)

if I use toString -> I need to override it only once in one of those Entities otherwise there will be an exception

* can be written also: e.department.id = d.id ⇒ comparing attributes

14: Entity Life Cycle event + caching

- I can define in my entity thing that will happen before each:

UPDATE: @PreUpdate @PostUpdate
REMOVE: @PreRemove @PostRemove
PERSIST: @PrePresist @PostPresist
LOAD : @PostLoad

those are called Life Cycle Event

ex

```
@PrePersist
public void prePersist(){
  this.dateCreated = LocalDateTime.now(); }
```

- כל פעם שאני רוצה לשמור data ב-DB ← לפני ה-persist ← hibernate יקרא ל-method prePersist, ו-initialize את התאריך של היום לשדה dateCreated.

בפעולה כל כך פשוטה "ירוצו" ב-DB ← אני מעדכן ב-context ← בסוף של transaction הם יכנסו ל-DB

Caching

the cache that I can take from the EntityManagerFactory
Hibernate: doesn't has the caching implemented, I should do it explicitly
What we will learn
① Get the chache ② Refer to the cache
③ Enable the cache ④ configure the cache

- I will use the Factory to get the cache from (the var is emf)

① Cache cache = emf.getCache(); // I get the cache object
② the cache will give me access to
contains(Entity, id) → check if a certain instance exist (true/false)
evictAll(), evict(Entity), evict(Entity, id) ⇒ delete from the cache
evict(Entity) will evict the All the instance of this Entity + its children (if it has)

Dependency

if I want to manage the cache I must add a dependency to my

- pom.xml file:

<artifactId>hibernate-ehcache</artifactID> // make sure that the version correspond to the same version of <....>hibernate-core</...>
But it's not the end. I need to configure the persistance.xml

persistance.xml

I need to add name="hibernate.cache.use_second_level_cache" value="true"

name="hibernate.cache.region.factory_class"

value="org.hibernate.cache.ehcache.EhCacheRegionFactory" />

this is under properties

under presistance unit:

(1) <shared-cache-mode> . . . . . </...> // How I manage the cache

ALL: all the Entites are stored in the cache

ENABLE_SELCTIVE: none is stored by default. I need to activate it on the Entity to make it part of the cache by @Cacheable

DISABLE_SELECTIVE: All is stored by the default, if I want to disable I do it par Entity by @Cacheable(false)