

Programació

UT14.2 Entitats i
gestor d'entitats

Entitats

Una entitat no és més que una classe els objectes de la qual persistiran en una BD. Perquè una classe es convertisca en una entitat:

- ✓ Ha de ser un **POJO** (Plain Old Java Object). És a dir, una classe simple, que no depenga d'un framework en concret i no estenga ni implemente res especialment.
- ✓ Ha de tindre un constructor públic, sense paràmetres.
- ✓ Ha de tindre els seus atributs privats, amb mètodes *getters i setters .
- ✓ No ha de ser una classe interna.
- ✓ No ha d'estar definida com una classe final.
- ✓ Ha d'implementar la interfície Serializable.

Entitats. Mapatge de taules

Qualsevol classe que necessitem persistir l'hem de convertir en entitat. Per a fer-ho utilitzarem l'anotació **@Entity**.

Tota entitat necessita un únic atribut que actue com a identificador. L'atribut identificador tindrà l'anotació **@Id**.

Podria donar-se el cas que necessitarem especificar un nom diferent per a la taula, per exemple si no estem desenvolupant l'aplicació des de zero i partim d'un model de dades ja creat. Podem fer-ho amb l'anotació **@Table** en la qual incloem el nom de la taula.

Entitats. Mapatge de taules

```
@Entity  
@Table(name = "CATEGORIAS")  
public class Categories implements Serializable {  
    @Id  
    private Integer id;  
    private String nom;  
  
    public Categories() {}  
  
    public Integer getId() {  
        return id;  
    }  
    public void setId(Integer id) {  
        this.id = id;  
    }  
    public String getNom() {  
        return nom;  
    }  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
}
```

TERMINOLOGIA:

- Anomenem **camps** (fields) als atributs privats d'una entitat.
- Anomenem **proprietats** (properties) al conjunt de getters i setters d'una entitat que modifiquen un camp.

Entitats. Mapatge de columnes

És possible anotar les característiques físiques de la columna de la base de dades en la qual es mapeja un atribut utilitzant l'anotació **@Column**. Encara que és possible especificar bastants elements, comentarem només algun d'ells (consulteu l'especificació de JPA per a obtindre més informació).

- Igual que amb les taules, és possible especificar els **noms de les columnes** amb què es va a mapejar cada atribut. El codi següent mostra un exemple.

```
@Entity  
@Table(name = "empleats")  
public class Empleat {  
    @Id  
    @Column(name = "emp_id")  
    private int id;  
    private String nom;  
    @Column(name = "sal")  
    private double sou;  
    @Column(name = "com")  
    private String comentari;  
    // ...  
}
```

empleats	
PK	emp_id
	nom
	sal
	com

Entitats. Mapatge de columnes

- És possible també obligar al fet que un **atribut no puga deixar-se a null** utilitzant l'element nullable=false. En la columna de la taula s'inclouria la restricció SQL *NOT NULL*. Per exemple:
- Quan no especificuem la longitud d'una columna que emmagatzema cadenes (String, char[] o Character[]), el valor per defecte és 255. Per a definir una altra grandària cal utilitzar l'element length.

[Més opcions.](#)

```
@Entity  
@Table(name = "empleats")  
public class Empleat {  
    @Id  
    @Column(name = "emp_id")  
    private int id;  
    @Column(nullable=false)  
    private String nom;  
    @Column(name = "sal", nullable=false)  
    private double sou;  
    @Column(name = "com")  
    private String comentari;  
    // ...  
}
```

```
@Entity  
@Table(name = "empleats")  
public class Empleat {  
    @Id  
    @Column(name = "emp_id")  
    private int id;  
    @Column(nullable=false, length=50)  
    private String nom;  
    @Column(name = "sal", nullable=false)  
    private double sou;  
    @Column(name = "com", length=200)  
    private String comentari;  
    // ...  
}
```

Entitats. Mapatge de tipus enumerats

Si no especificuem cap anotació, per a cada valor del tipus enumerat s'emmagatzemarà en la BD un ordinal, que es correspon amb l'ordre de creació. Per exemple:

```
public enum TipusEmpleat {  
    EMPLEAT_TEMPS_PARCIAL,  
    EMPLEAT_TEMPS_COMPLET,  
    EMPLEAT_EXTERN  
}
```

```
@Entity  
@Table(name = "empleats")  
public class Empleat {  
    @Id  
    @Column(name = "emp_id")  
    private int id;  
    private TipusEmpleat tipus;  
    // ...
```

El que es guardarà a la base de dades a la columna “tipus” serà un 0 per als empleats parcials, un 1 per als de temps complet i un 2 per als externs.

Això pot provocar problemes de manteniment si afegim nous valors de “TipusEmpleat” entre els ja existents o modifiquem l’ordre dels valor existents, ja que els valors que ja es troben persistits a la BD no s’actualitzaran i els nous correspondran al nou ordre de l’enumerat.

Entitats. Mapatge de tipus enumerats

Altra opció seria emmagatzemar el nom del valor com una cadena en lloc d'emmagatzemar l'ordinal. Podem fer això afegint una anotació **@Enumerated** en l'atribut i especificant un valor STRING.

D'esta manera no es guarda en la BD l'ordinal, sinó l'STRING del valor que tinga l'enumerat. El problema que podem tindre és si modifiquem els noms de l'enumerat.

En general, definir el tipus enumerat com un ordinal és la forma més eficient de treballar, però sempre que no siga probable haver d'afegir nous valors enmig dels ja existents.

```
@Entity  
@Table(name = "empleats")  
public class Empleat {  
    @Id  
    @Column(name = "emp_id")  
    private int id;  
    @Enumerated(EnumType.STRING)  
    private TipesEmpleat tipus;  
    // ...  
}
```

Entitats. Mapatge de tipus temporals

El mapejat dels tipus temporals del paquet `java.sql` (`java.sql.Date`, `java.sql.Time` o `java.sql.Timestamp`) no plantejen cap problema en absolut i s'emmagatzemen en la base de dades sense fer cap canvi.

Per altra banda els tipus de dades temporals de `java.util` com pot ser `java.util.Date` i `java.util.Calendar` necessiten metadades addicionals per indicar a quin tipus JDBC `java.sql` s'ha d'utilitzar quan fem la persistència. Per això tenim l'anotació `@Temporal` especificant el valor de tipus JDBC corresponent a l'enumerat `TemporalType` (`DATE`, `TIME` i `TIMESTAMP`)

```
@Id private int id;
@Temporal(TemporalType.DATE)
private java.util.Date dataNaixement;
@Temporal(TemporalType.TIMESTAMP)
private java.util.Date horaEixida;
```

Entitats. Mapatge de la clau primaria

Encara que moltes bases de dades utilitzen claus primàries naturals (nom, NIF...), la pràctica recomanable és treballar amb claus primàries generades de manera automàtica i sense cap significat fora de l'àmbit de la pròpia gestió de la base de dades.

Per a definir una clau primària única generada de manera automàtica tenim l'anotació **@GeneratedValue**. És important tindre en compte que en este cas l'identificador de l'entitat no estarà disponible fins que s'haja realitzat la inserció en la base de dades.

Entitats. Mapatge de la clau primaria

Com pots observar no cal assignar-li valors a l'atribut identificador en el constructor. Serà JPA qui s'encarregue de realizar eixe treball quan persistim l'entitat a la BD.

Existeixen quatre estratègies de generació d'identificadors que se seleccionen mitjançant l'element strategy de l'anotació. Són AUTO, IDENTITY, SEQUENCE o TABLE.

En este curs no aprofundirem sobre estes estratègies.

```
@Entity  
@Table(name = "CATEGORIES")  
public class Categories implements Serializable {  
    @Id  
    @GeneratedValue  
    private Integer id;  
    private String nom;  
  
    public Categories() {}  
  
    public Categories(String nom) {  
        this.nom = nom;  
    }  
  
    public Integer getId() {
```

Entitats. Mapatge de clau primaria composta

El primer que cal fer és definir una classe amb els atributs de la classe composta.

Suposem que volem mapejar una taula USUARIS amb una clau primària composta formada per USUARI i DEPT, tots dos VARCHAR. Haurem de crear una nova classe:

```
public class UsuariPK implements Serializable {  
    private String username;  
    private String dapartment;  
    //.....
```

Entitats. Mapatge de clau primària composta

I en l'entitat principal utilitzem esta classe com a clau primària, indicant l'anotació **@EmbeddedId** i els noms de les columnes amb l'anotació **@AttributeOverride**:

```
@Entity  
@Table(name = "USUARIS")  
public class Usuari {  
    @EmbeddedId  
    @AttributeOverrides({  
        @AttributeOverride(name="username", column=@Column(name="USUARI")),  
        @AttributeOverride(name="department", column=@Column(name="DEPT"))  
    })  
    private UsuariPK usuariPK;  
    //...  
}
```

Nom de l'atribut en la classe “UsuariPK”

Identificador compost (objecte embedut “UsuariPK”)

Nom de la columna en la taula “USUARIS” de la BD

Entitats. Mapatge de clau primaria composta

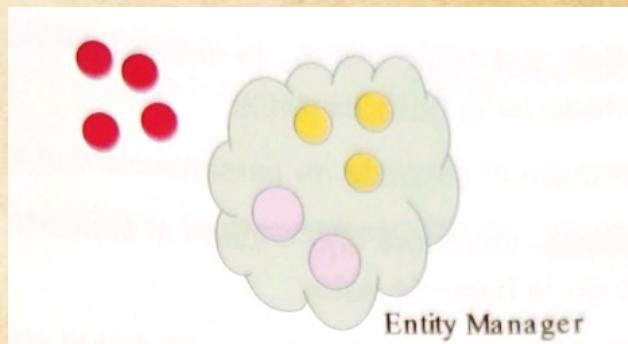
Altra opció és utilitzar l'anotació **@IdClass** si volem tindre per separat en l'entitat els dos atributs que conformen la clau primaria:

```
public class UsuariPK implements Serializable {  
    private String username;  
    private String department;  
    //.....
```

```
@Entity  
@Table(name = "USUARIS")  
@IdClass(value = UsuariPK.class)  
public class Usuari{  
    @Id  
    @Column(name="USUARI")  
    private String username;  
    @Id  
    @Column(name="DEPT")  
    private String department;  
    //..  
}
```

Gestor d'entitats

Un gestor d'entitats (interfície EntityManager) gestiona una sèrie d'objectes d'entitats pertanyents a una unitat de persistència, encarregant-se de persistir-les de manera transparent al programador.



Persistirà en la BD qualsevol canvi que es produisca en el seu context de persistència (objectes grocs i morats). Els objectes de color roig no pertanyen al context, i no seran persistits.

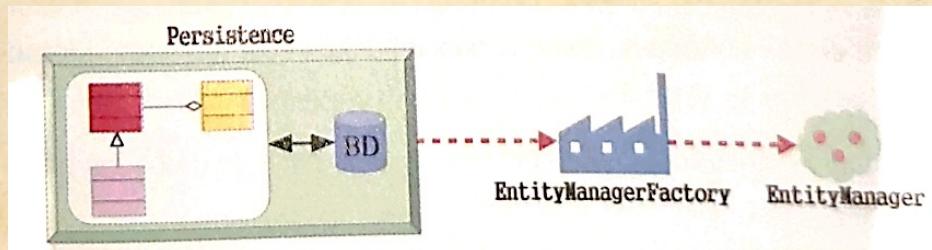
Gestor d'entitats

El gestor d'entitats té dos responsabilitat fonamentals:

- Defineix una **connexió transaccional** amb la base de dades que hem d'obrir i mantindre oberta mentre estem realitzat operacions. En este sentit realitza funcions similars a les d'una connexió JDBC.
- A més, **manté en memòria** una “caché” amb les entitats que gestiona i és responsable de sincronitzar-les correctament amb la base de dades quan es realitza un “flush”. El conjunt d'entitats que gestiona un EntityManager es denomina el seu context de persistència.

Gestor d'entitats

- Un objecte EntityManager no es crea directament amb l'operador new, sinó que s'obté a partir d'un objecte EntityManagerFactory, la funció del qual és simplificar la construcció i configuració dels EntityManager.
- Al seu torn el mètode EntityManagerFactory s'obté mitjançant un mètode estàtic de la classe Persistence, que estarà configurat específicament per a la base de dades que es va definir en la unitat de persistència.



```
// Tries la unitat de persistència mitjançant el seu nom  
EntityManagerFactory emf = Persistence.createEntityManagerFactory("AntJPAPU");  
// Construim un EntityManager a partir de l'objecte "emf" anterior  
EntityManager em = emf.createEntityManager();
```

Mètodes d'EntityManager

La interfície EntityManager disposa de mètodes que permeten gestionar els objectes dels quals són responsables. Destaquem els següents:

- **void clear()**: esborra el context de persistència, disconnectant totes les seues entitats.
- **boolean contains(Object entity)**: comprova si un objecte d'una entitat es troba ja gestionat en el context de persistencia.
- **void detach(Object entity)**: elimina l'entitat del context de persistència, deixant-la disconnectada de la base de dades.
- **<T> T find(Class<T>, Object key)**: busca per clau primaria.
- **void flush()**: sincronitza el context de persistència amb la base de dades.
- **<T> T getReference(Class<T>, Object key)**: obté una referència a una entitat, que pot haver sigut recuperada de forma mandrosa (lazy).

Mètodes d'EntityManager

- **EntityTransaction getTransaction()**: retorna la transacció actual.
- **<T> T merge(T entity)**: incorpora una entitat al context de persistència fent-la gestionada, suposant que ja existeix en la BD.
- **void persist(Object entity)**: fa que una entitat siga gestionada (l'afig al context de persistència).
- **void refresh(Object entity)**: actualitza l'estat de l'entitat amb els valors de la base de dades, sobreescriuint els canvis que s'hagen pogut realitzar en l'entitat.
- **void remove(Object entity)**: elimina l'entitat

Pots trobar més informació en l'API [d'EntityManager](#)

Exemple:

```
em.persist(empleat);  
em.flush();  
em.refresh(empleat);
```

*La invocació a **flush** assegura que s'executa l'"insert" en la BD i la invocació a **refresh** assegura que l'identificador es carrega en la instància de l'entitat.*

Transaccions

Qualsevol operació que modifique la base de dades (crear, modificar o esborrar) ha de fer-se dins d'una transacció. La manera de crear una transacció és a partir d'un objecte EntityManager.

```
EntityTransaction tx = em.getTransaction();
```

Una volta disposem de l'objecte de transacció, podem realitzar les següents operacions:

- Començar una transacció, amb el mètode **begin()**
- Acabar una transacció amb èxit, amb el mètode **commit()** i portar tots els canvis realitzats a les entitats, a la base de dades.
- Desfer la transacció amb el mètode **rollback()**

Transaccions

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("EmpleatPU");
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();

Empleat e1 = new Empleat("11111111A", "Pepe", 32);
Empleat e2 = new Empleat("22222222B", "Neus", 37);
Empleat e3 = new Empleat("33333333C", "Laura", 28);

tx.begin(); // Comença la transacció
em.persist(e1);
em.persist(e2);
em.persist(e3);
tx.commit(); // Finalitza la transacció. Els canvis son persistits a la BD
```

JPA Controller

Un controlador de JPA és una classe creada expressament per a gestionar les operacions CRUD d'una entitat. També proporciona funcionalitats extra com obtindre una llista amb totes les instàncies de l'entitat, o el número d'aquestes que es troben emmagatzemades en la BD.

JPA Controller és el nom amb el qual es designa en JPA al patró de disseny DAO per a una entitat. NetBeans pot generar-los de manera automàtica per a cadascuna de les entitats del nostre model:

New > Other ... > Persistence > JPA Controller Classes from Entity Classes

És interessant que proves de crear algun controlador JPA i analitzes el funcionament dels seus mètodes.