```
@Entity
class Compte{
  @Id
  int ide
  @onetoone
  Etudiant e;
  //
}
```

```
@Entity
class Etudiant{
  @Id
  int ide;
  :
  //Association
  @Manytoone
  Groupe gp;
  //Constructeur vide
  +
  Getters et Setters
  @onetoone
  CompteC;
```

```
@Entity
class Groupe {
  @Id
  int idgp;
  String Nom;
  @onetoMany;
  List<Etudiant> le;
}
```

```
Groupe gpe1=new Groupe(1,"GCR");
Etudiant e = new Etudiant(1,"Ali",gpe1);
// insertion de l'objet e ?
```

```
class GestionImp implements
    Session session;
    public GestionImp(){

    }
```

Isave : insert
merge : update
Load : find

```
Etudiant getEtudiantById(int id){
  Etudiant e = this.session.load(Etudiant.class, id)
  return e;
}
```

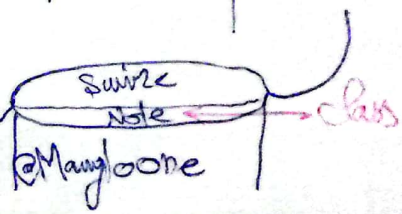
```
List<Etudiant> getEtudiantByAge(int age){
  String req = "select e from Etudiant e where
               e.age =:a";
  Query q = this.session.createQuery(req);
  q.setParameter("a", age)
  List<Etudiant> le = q.getResultList();
  return le;
}
```

```
@Entity
class Etudiant{
@ManytoOne
Groupe gp;
@ManytoMany
List<Matiere>
```

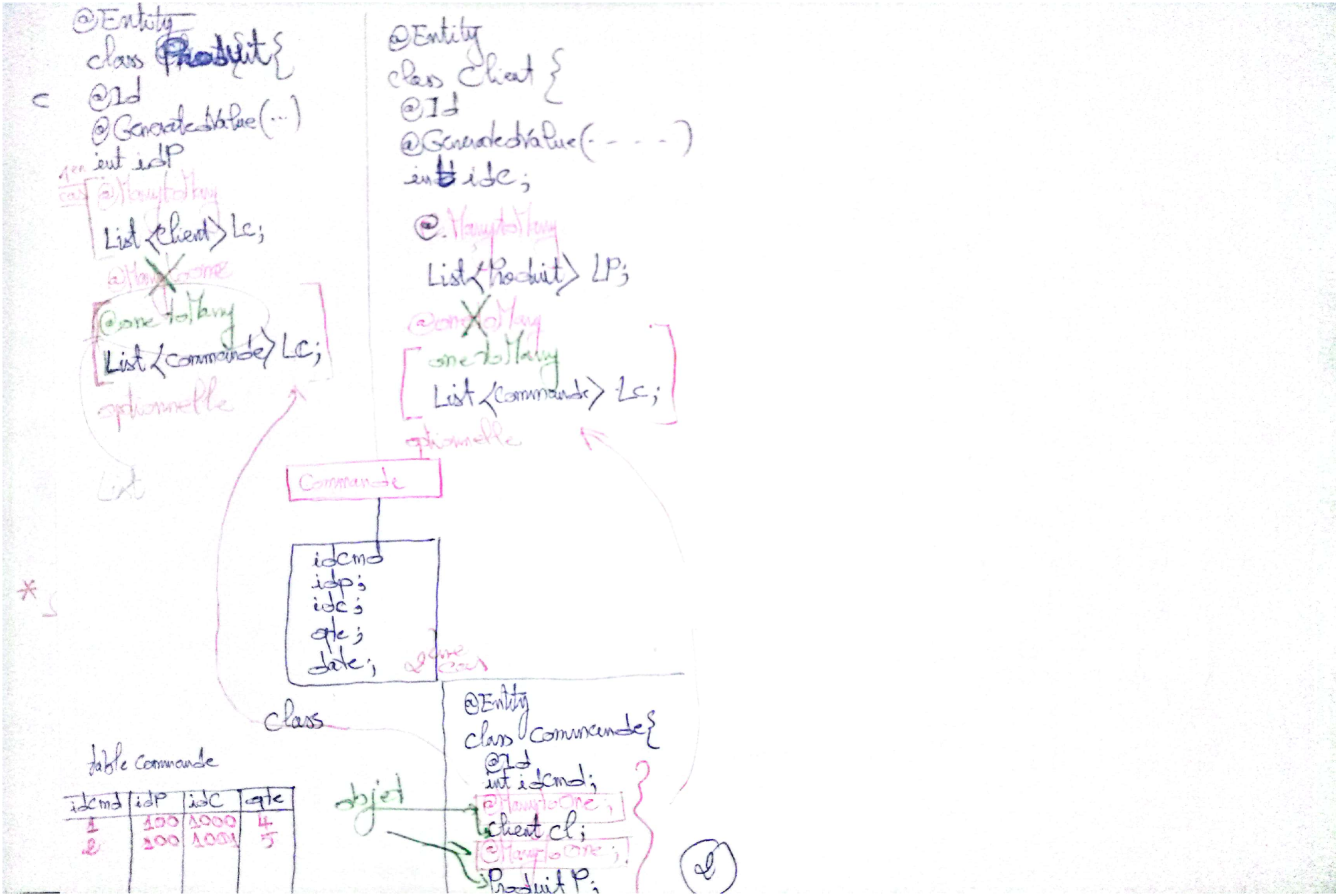
```
@Entity
class Groupe{
  :
  :
  :
```

```
class Matiere
@ManytoMany
```

```
class Enseignant
```



Scanné avec CamScanner



Scanné avec CamScanner

ORM : résout la différence entre modèle objet et modèle relationnel

(Objet/Relational Mapping)

ORM

↳ Accès + Gestion des données

Persistant un objet ≡ Sauvegarder ses données (objet recréée lors d'un service)

Persistance des données ≡ objet se faire par BD relationnel

↳ Supporter (héritage, polymorphisme)

* JPA (Java Persistence API) BD relationnelle → Api pour la persistance des objets dans

. **Propriétés** :

. Classe ≡ table

. Variable de classe ≡ colonne de table

GlassFish : fournisseur de Persistance (implémente les classes et méthodes)

* **Les entités** : ≡ Les classes dont les instances par ↳ @Entity

```
@Entity
public class Etudiant{
}
```

=

```
@Entity(name = "etud")
public class Etudiant{
}
```

attribut clé primaire ↳ @Id

↳ @GeneratedValue (la clé générée automatiquement)

. l'annotation peut avoir un attribut strategy → indique comment générer la clé

. IDEntity

↳ autoincrementer

. Auto

↳ par défaut

. Les attributs de @Column

- name nom de l'attribut.
- unique valeur unique.
- nullable accept null
- length Longueur max

Scanné avec CamScanner

Exp :

```
@Entity
public class Etudiant{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int cin;
    @Column(nullable = false)
    private String nom;
    @Column(length = 45)
    private String prenom;
    @Column(name = "gp", nullable = false)
    private String groupe;
}
```

* Association entre les entités :

Cardinalité : 1-1, 1-N, N-1, M-N

@	Attribut ou(e)
@OneToOne	⟶ objet
@ManyToOne	⟶ objet
@OneToMany	⟶ List<>
@ManyToMany	⟶ List<>

(2)

• OneToOne (représenté par une clé étrangère ajouté dans la table qui correspond au coté propriétaire)

↑ optionnelle

Exp :

```
@Entity
public class Etudiant{
    @Id
    @OneToOne
    private Compte c;
}
```

```
@Entity
public class Compte{
    @Id
    private int idCpte;
    private String user;
    private String pswd;
```

• Association 1-N et N-1 @OneToMany @ManyToOne

(Représenté par une clé étrangère dans la table qui correspond au coté propriétaire (obligatoirement au coté Many))

Exp

```
@Entity
public class Etudiant{
@Id
private int cin;
//....
@ManyToOne
@JoinColumn(name = "idgp")
private Groupe groupe;
}
```

```
@Entity
public class Groupe{
    @Id
    private String idgp;
    private String designation;
    @OneToMany
    List<Etudiant> etudiants;
}
```

Scanné avec CamScanner



* @ManyToMany (M-N)

Exp : • un film a plusieurs acteurs ;

• un acteur joue dans plusieurs films ;

```
@Entity
public class Film{
  @Id
  private int idf
  .
  @ManyToMany
  List<Film> acteurs;
}
```

```
@Entity
public class Acteurs{
  @Id
  private int ide ;
  .
  @ManyToMany(mappedBy="acteurs")
  List<Film> films;
}
```

• @JoinTable ⇝ pour changer le mapping par défaut

L'autre coté doit : "mappedBy"

```
@Entity
public class Film{
  @Id
  private int idf;
  //
  @ManyToMany
  @JoinTable(name='Role', joinColumns=@JoinColumn(name="films_idf"),
  inverseJoinColumns=@JoinColumn(name="acteurs_ide"))
```

3

```
}
```

• @JoinTable : ⇝ donne les info sur la table association (Role) qui va représenter l'association

• name ⇝ nom de la table

• joinColumns ⇝ donne le nom des attributs de la table (de la clé primaire) propriétaire

• inverseJoinColumns ⇝ donne le nom des attributs de la table (de la clé primaire de l'autre côté non propriétaire)

*Mapping de l'héritage

Single_Table / Joined_Table / Table_Per_Class

Heritage représenté par :

- une seule table en BD
- jointure entre la table de l'entité mère et la table de l'entité enfant
- une table par entité

Scanné avec CamScanner

Exp :

1 Single_Table Personne

Personne
Id : int nom : String Adresse : String
parcourir toString Pass

Id	Type	Nom	@	Note	Matiere
1	ETD	Ali	Gabes	12	Null
2	ENS	Amin	Gabes	Null	Java

```
@Entity
@DiscriminationtorValue("ENS")
public class Enseignant extends Personne{

@Entity
@DiscriminationtorValue("ETD")
public class Etudiant extends Personne{
    double note;
```

2 Table_Per_class

Etudiant

Id	Nom	@	Note
1	Ali	Gabes	12

Enseignant

Id	Nom	@	Matiere
2	Amin	Gabes	Java

Etudiant
note : double

Enseignant
matiere : string

```
@Entity
@Inheritance(Strategy=InheritanceType.Table_Per_class)
Public abstract class Personne {
    @Id
    @GeneratedValue(Strategy=GeneratedType.Table)
    private Long ID;
    ⋮
}
```

(utilisé pour la portabilité relative de l'identifiant en)

```
@Entity
public class Enseignant extends ~ { ~ }
@Entity class Etudiant extends ~ { — }
```

3 Joined_Table

Etudiant

Note	Id
12	1

Enseignant

Matiere	Id
Java	2

Personne

Id	Nom	Adresse
1	Ali	Gabes
2	Amin	Gabes

Code :

```
@Entity
@Inheritance(Strategy=InheritanceType.SINGLE_TABLE)
@DescriminatorColumn(name="Type", Length=4)
public class Personne{
    @Id
    @GeneratedValue(Strategy=GeneratedType.IDEntity)
    Private Long ID;
```

```
@Entity
@Inheritance(Strategy=InheritanceType.JOINED)
public abstract class Personne {
    @Id
    @GeneratedValue(Strategy=GeneratedType.Table)
    ⋮
```

← in continue pour (2)

Scanné avec CamScanner

→ Modifier un objet Etudiant e

```
em.getTransaction().begin();
Etudiant e1 = em.find(Etudiant.class, e.getCin());
e1.setNom("Dupond");
em.merge(e1);
em.getTransaction().commit();
```

```
Public void addSoutenance(Soutenance s){
    em.getTransaction().begin();
try{ em.persist(s);
     em.getTransaction().commitUt();
   }
catch (Exception e){
    em.getTransaction().rollback();
    e.printStackTrace();
   }
}
```

→ Supprimer un objet par son id

```
em.getTransaction().begin();
Etudiant e=em.find(Etudiant.class, cin);
em.remove(e);
em.getTransaction().commit();
```

* Hibernate : solution open source de type ORM

↳ représenter une base de données en objet java et vice versa.

fournissent : hibernate.cfg.xml situé dans SCR (qui comporte des informations de mapping de propriétés : ...) de reporter

```
SessionFactory sessionFactory;
ServiceRegistry serviceRegistry = new
    StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
Metadata metadata = new MetadataSources(serviceRegistry).
        getMetadataBuilder().build();
sessionFactory=metadata.getSessionFactoryBuilder().build();
```

↳ création du fabrique de Session (SessionFactory)

on va crée une Connexion par :

```
Session session = sessionFactory.openSession();
```


Scanné avec CamScanner

* Gestionnaire d'entité

Les informations → donné dans → META-INF dans le répertoire le fichier persistence.xml ou pom.xml

- EntityManager → manipuler les objet (entités) → persister

méthode pour obtenir une référence de objet EntityManager

l'unité de persistance

```
EntityManagerFactory factory =
    Persistence.CreateEntityManagerFactory("test1");
EntityManager em = factory.CreateEntityManager();
```

↳ permet de créer et exécuter les requêtes par

Les méthodes:

- persist → Insère une entité dans la BD
- merge → synchronise l'état des entités
- remove → supprime une entité de la BD
- find → trouve une instance dans la BD à partir PK
- CreateQuery ou CreateNativeQuery → Créer une requête JPQL

(5)

Exp: `Etudiant e = em.find(Etudiant.class, id);`
↳ recherche d'un objet par son id.

- Le résultat récupérer par CreateQuery par :
  - * getResultList() → récupérer le résultat sous forme Liste
  - * getSingleResult() → un seul élt de la sélection
  - * getFirstResult() → Le premier élt de la sélection

Exp :

```
Query q = em.CreateQuery("Select e from Etudiant e");
List<Etudiant> etdList = q.getResultList();
```

```
Query q = em.CreateNativeQuery("select * from Etudiant
  where num ?", model.Etudiant.class);
q.setParameter(1, 102);
Etudiant e = (Etudiant) q.getSingleResult();
```

→ Ajouter un objet Etudiant e

```
em.getTransaction().begin();
em.persist(e);
em.getTransaction().commit;
```

Scanné avec CamScanner

* Session ⟶ créer + exécuter ⟶ les requêtes grâce à :

. Les méthodes :

save ⟶ Insère une entité dans la DB
merge ⟶ synchronise l'état
delete ⟶ supprime une entité dans la DB
load ⟶ Trouve une instance dans la DB
createQuery ⟶ à partir de FK créer une requête JPQL

Exp1 :

* livre lv = this.session.load(Livre.class, id);
  ↳ recherche d'un objet par son id
* getResultList() ⟶ récupérer le résultat sous forme Liste
  getSingleResult() ⟶ un seul élt de la sélection

Exp2 : recherche avec getSingleResult()

```
this.session.getTransaction().begin();
String sql = "select d from Livre d where d.id=:id";
Query<Livre> query = this.session.createQuery(sql);
query.setParameter("id", id);
Livre liv = query.getSingleResult();
this.session.getTransaction().commit();
```

Exp3 :

```
Query<Livre> query = this.session.createQuery("select
from Livre e");
List<Livre> livs = query.getResultList();
```

Exp4 : Ajouter un objet Livre lv

```
this.session.getTransaction().begin();
this.session.save(lv);
this.session.getTransaction().commit();
```

Exp5 : Modifier un objet Livre Lv

```
this.session.getTransaction().begin();
this.session.merge(lv);
this.session.getTransaction().commit();
```

Exp6 : Supprimer un objet par son id.

```
Livre lv = this.session.load(Livre.class, id
this.session.getTransaction().begin();
this.session.delete(lv);
this.session.getTransaction().commit();
```


Scanné avec CamScanner

JPQL SQL adapté à la JPA

↳ manipuler les entités JPA

Supporte select, Update, delete

Exp : Recherche avec (where)

```
public List<Etudiant> getAllEtudiantByGp(int idgp){
  em.getTransaction().begin();
  Query q=(Query) em.createQuery("select e from
Etudiant e where e.idgp=:idgp");
  q.setParameter("idgp",idgp);
  List<Etudiant> le=q.getResultList();
  em.getTransaction().commit();
  return le;
}
```

Maven → Gérer + Chercher les dépendances Versions

→ Compile le code src + tests

↳ Exécute les tests

↳ Générer un jar (Librairie exécutable) (bibliothèque)

POM.xml

Projet objet Model

Scanné avec CamScanner

TD4 - JEE

User

```
@Entity
class User {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  int idU;
  private String nom;
  private String login;
  private String pswd;
  private double solde;
  @ManyToOne
  CatgAbonnement CatAb;
  // Constructeurs + Getters + Setters
```

CatgAbonnement

```
@Entity
class CatgAbonnement {
@Id
int idCat;
Private double Credit;
Private String typeAbon;
// Constructeur + Getters + Setters
```

Vod

```
@Entity (name = "video")
Public class Vod {
@Id
@GeneratedValue (strategy = GenerationType.IDENTITY);
private int idV;
@Column(name = "Genre", nullable = false, length = 45);
private String genre;
private String titre;
@ManyToOne
CatgAbonnement CatAb;
//Constructeurs + Getters + Setters
```


Scanné avec CamScanner

```
Public User verif(String login, String pswd){
    String req = "from select u from User u where u.login =: login and
                                                  u.pswd =: pswd";
    Query q = this.session.CreateQuery(req, User.class, req)
    q.setParameter("login", login);
    q.setParameter("pswd", pswd);
    User u = q.getSingleResult()
    return u;
}

Public List<Vod> getAllVodByCatAbUser(int idCat){
    String req = "select v from Vod v where V.CatAb.idCat =:idCat";
    Query q = this.session.CreateQuery(req);
    q.setParameter("idCat", idCat);
    List<Vod> lv = q.getResultList();
    return lv;
}

Public List<Vod> getAllVodByCatAbUserAndGen(int idCat, String genre){
    String req = "select v from Vod v where V.CatAb.idCat =:idCat and
    V.genre =: genre";
    Query q = this.session.CreateQuery(req);
    q.setParameter("idcat", idCat);
    q.setParameter("genre", genre);
    List<Vod> video = q.getResultList();
    return video;
}
```


Scanné avec CamScanner

```
Public List<Vod> getAllVod(){
    String req = "Select V from Vod V";
    Query q = this.session.CreateQuery(req);
    List<Vod> LV = q.getResultList();
    return LV;
}

Public double getSoldeUser(int idUser){
    Query q = session.CreateQuery("select U.solde from User U where
                                   U.id =: idUser");
    q.setParameter("idUser", idUser);
    return ((double) query.getSingleResult());
}

Public Vod getVodById(int id){
    Vod V = this.session.load(Vod.class, id)
    return V;
}

Public void UpdateSolde(int idUser, double m){
    User U = this.Session.load(User.class, idUser);
    U.setSolde(U.getSolde() + m);
    session.getTransaction.begin();
    Session.merge(U);
    session.getTransaction.commit();
}
```


Scanné avec CamScanner

```
Protected void doPost ( req , rep ) {
    String pwd = req.getParameter ("Pwd");
    String login = req. getParameter ("login");
    GestionImp gv = new GestionImp( );
    User u = gv.Verif (login, pwd);
    if (u != null){
        Session.setAttribute ("u", u);
        int id = u. getCatAb( ).getIdCat ( );
        List<Vod> Lv = gv.getAllVodsByCatAbUser(id);
        req.setAttribute ("Lv", Lv);
        req. getRequestDispatcher ("index.jsp").forward (req, rep);
    }else{
        String msg = "Invalid Login Login/pwd";
        req.setAttribute("msg", msg);
        req.getRequestDispatcher("auth.jsp").forward(req,rep);
    }
```

"ServFilter.java?genre=Idv" → doGet
"ServFilter.java?genre=Comd" → http → servlet
"ServFilter.java?genre=Science"
"ServFilter.java?genre=Action"

```
ServFilter ( ){
  doGet (req, rep){
    String genre = req.getParameter("genre");
```

GestionVodImp( ) gv = new GestionVodImp( );
gv.getAllVodsByCatAbUserAndGenre(id, genre)

Scanné avec CamScanner