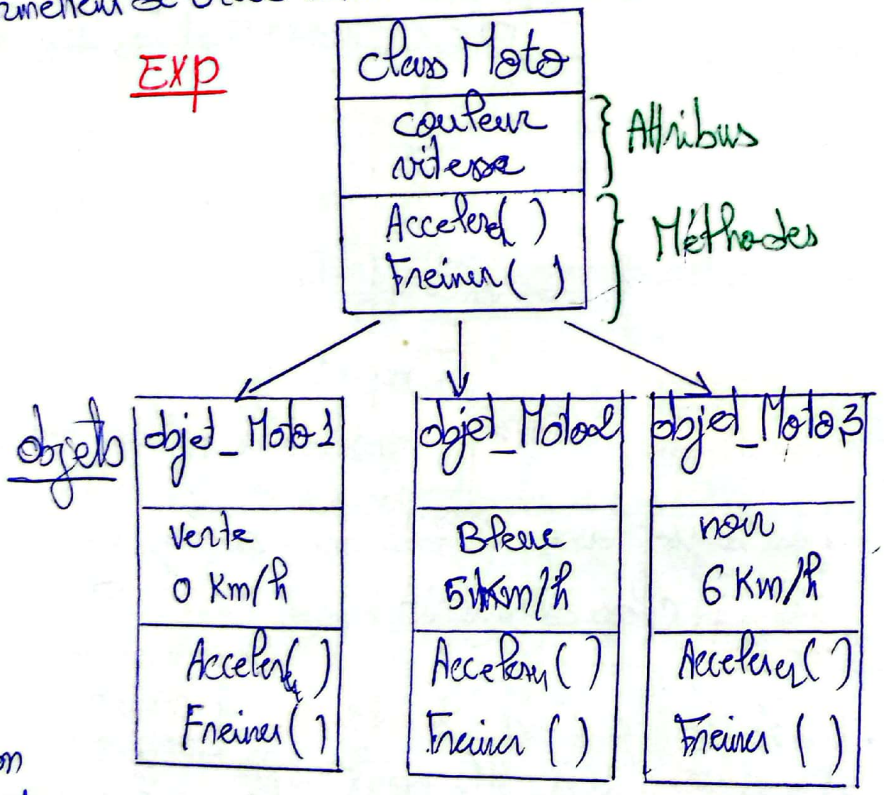
* les objets (vélo)

Chaque objet ≡ instance de classe
représentation d'une classe

(accessible ...)
(vitesse, plateaux...)
↓
attributs (caractéristique)
méthodes (comportement)
↑

* les classes
permettent de créer attributs et méthodes

EXP



notion d'objet: permet la modularité et l'encapsulation: Isoler les erreurs et facilite' la maintenance

la notion de classe: fournit un mécanisme réutilisabilité
redéfinition

(1)

Package ou bibliothèque : ensemble standard de class

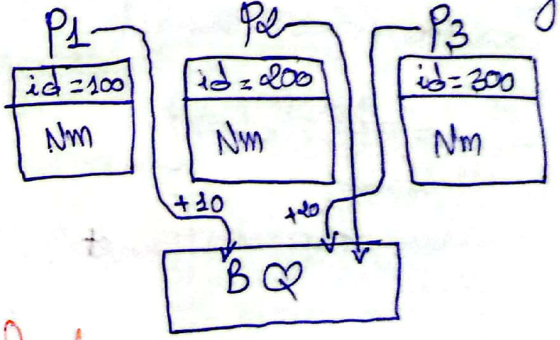
• Variables d'instance : Ex Moto : "couleur" et "vitesse"
att. d'instance

(leurs valeurs est fixé et modifiées dans l'objet)

• Variables de Classe (static) appliquer à la classe.
Ex : compteur d'objet . Ici 3motos.

Exp: class Personne { int id; String nom; static int n; }
attributs d'instances
attributs de Classe
partager sur tous les objets.

propre pour chaque objet (خاصة بكل كائن)
static (من غير ما يأثر على الآخر)



• les méthodes

• méthodes d'instances Ex: Moto → Accelerer( ) Freiner( )

• méthodes de classes (static) Ex: main( )

Scanné avec CamScanner

Une classe est définit par : (état/objet) ← les attributs
→ Les méthodes (le comportement des objets)

Constructeur par exp ← (au moment de la création d'un objet)

⇓

C'est une méthode qui admet une spécificité

Exp: très important (Heritage)

personne ( ) { } ← constructeur

personne (int id, String Nm) { } ← surdéfinition ou surcharge des méthodes

this.id = id ;
this.Nm = Nm ; }

. elle désigne l'objet courant

méthode → Public void affiche ( ) {

System.out.println (this.id this.Nm);

}

Public Class Test {

prog principale (méthode)

accès → public static void main (String[] args) {

Personne p1 = new Personne (100, "Tab")

p1.affiche( ); }

(2)

Etudiant est un Personne

* Heritage

Public class Etudiant extends Personne {

. Si n'admet pas une constructeur en classe Personne Personne() ← Constructeur

private String Groupe; (Ce pas obligatoire d'écrire ça)

public Etudiant ( ) { } (existe implicite) { super( ); }

public Etudiant (int id, String nom, String gp)

{ super (id, Nm);

this.Groupe = gp; }

Public void affiche ( ) {

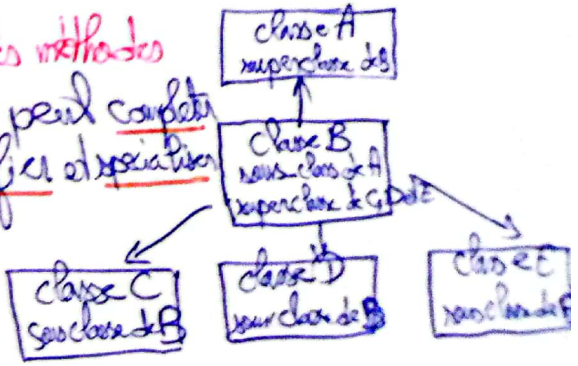
redéfinition de méthode → super.affiche( );

System.out.println (group); }

. La sous-classe va hériter des caractéristique (attributs et méthodes) de la superclasse

. String : class dans JAVA

. B hérite de toutes les méthodes et variables de A, elle peut compléter par d'autres ou modifier et spécialiser



Scanné avec CamScanner

Exp: constructeur par défaut de la classe

```
Moyenne m1 = new Moyen( )
m1.ajouteNote(14);
m1.ajouteNote(16);
```

m1	note	30
	nbre de note	2

```
double X = m1.CalculerMoyenne();
```

- pour dire pas de retour : void
- CalculeMoyenne(): double
- Type de retour

Public class Mote( ) <=> Public class Mote extends Object

- En java, tous les classes suivent une hériterit de classe Object.
- En l'absence du mot extends : La classe considéré par défaut comme héritent de classe Object

* Encapsulation : (Getters / Setters)

type de retour

```
public class Point {
    private int x, y;
    public int getX( ){
        return X;}
    public void setX (int X){
        this.X = X;}
```

accès aux méthodes

```
public class Test {
    public static void main (String[] args){
        Point p = new Point ();
        p.x = 2;  p.SetX(2)
        System.out.println (p.x);
        System.out.println (p.getX());}
```

privée ??

privée ??

```
public class Personne {
    Private int id;
    private String nom, Prenom;
    Private Adresse ad;
    Personne( ){ }
    Personne (int id, String n, String p, Adresse ad)
    { this.id=id;
      this.nom = n;
      this.prenom = p;
      this.ad = ad; }
    Public void setId (int id){
      this.id=id;}
    public void SetNom(String Nom)
      { this.nom = Nom;}
    Public Adresse getAd( ){
        return ad;}
}
```

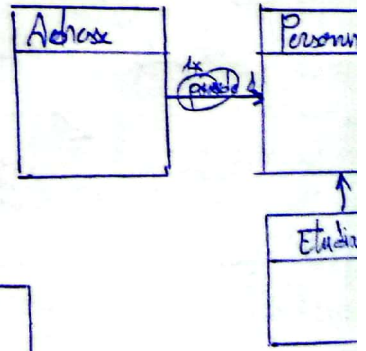
```
public class adresse {
    private int num;
    private string rue, ville;
    public void setNum (int n){
        this.num = num;}
```

```
public class Test {
    public static void main (String[] args){
        Personne p = new Personne (200);
        // P.id = 2 ;
        P.setId(2);
        Personne P1 = new Personne
        P1.setNom("Mdl");
```

- possible qu'il id est public
- si id est pri

P1

id	pri
Nom	*
P	0 ou
@	0 ou pas



Scanné avec CamScanner

```
public class Test {
    public static void main (String[] args ){
        (System.out.println())
        Adresse ad = new Adresse(10,"RueBon","Gabes");
        Personne P= new Personne (100,"BnMed","Ali", ad);
        P.ad.num =20;
        ad.setNum(20);
        P.getAd().setNum(20);
    }
}
```

création d'un objet ad

création d'un objet p

il faut que num et ad.num soient public

si ad est public

si num est privée

si ad est privée

appel de la méthode setNum() de l'objet ad

"Class" définir une classe

"public" définir les dts d'accès relatifs à cette classe. (ne pas modifier)

"main" méthode Principale de l'app

- public : précise que main est une fct publique accessible sans restriction
- static : définit que main comme une méthode de classe
- void : ce que va retourner la méthode main. rien dans ce cas

Exemple :

```
public class Moto {
    private String marque;
    private String couleur;
    private boolean moteur_en_marche = false;

    public Moto(String societe, String couleur){
        marque = societe;
        this.couleur = couleur;
    }
}
```

surcharge

Valeur par défaut : (A)

```
public Personne(){
    nom = "Wissem";
    Prenom = "Baggi";
} public Personne
```

Déclaration des attributs

Déclaration du constructeur

- "this" : représente l'objet courant
- création d'une classe → class → déclaration des attributs & méthodes

• Créer un objet :

peut regrouper ⇔

```
Moto moto_1;
moto_1 = new Moto("honda","verte");
```

(11)

```
Moto moto_1 = new Moto("honda","verte");
```

• Accès aux caractéristique : System.out.println(moto1.couleur);

• Accès aux méthodes : moto_1.demarrer_moteur();

(A) (11) `public Moto(){}` : ce constructeur construit un objet moto si les attributs possèdent de valeurs par défaut.

(14) L→ surcharge (on peut charger une méthode existante

Scanné avec CamScanner

Suncasting & sous casting :

(ycastiha : yroodha de même type)

```
public class Main{
  public static void main(String[]args){
    int my1nt = 9;
    double myDouble = my1nt;
    sys.out.println(my1nt); 9
    sys.out.println(myDouble); 9.0
```

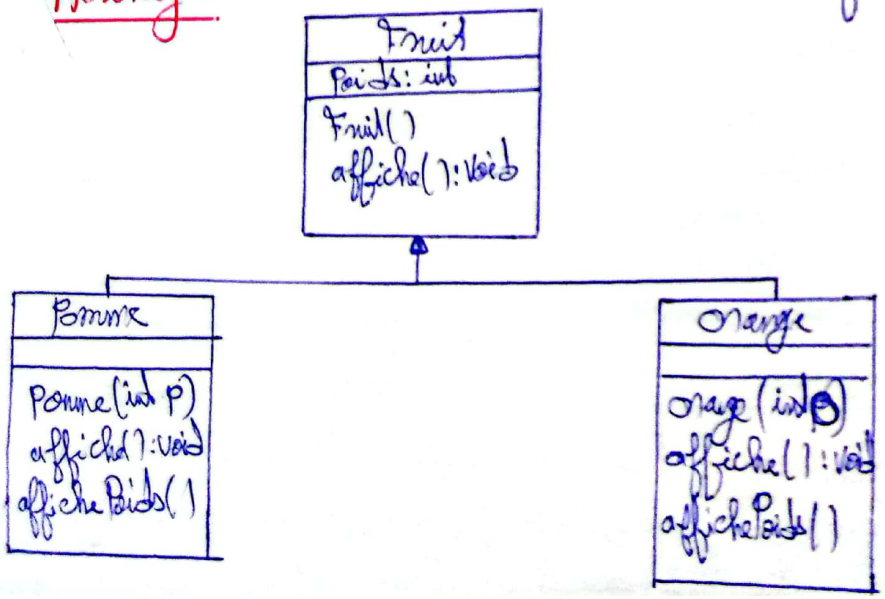
```
public class Main{
  public static void main(String[]gs)
  {double myDouble = 9.78d;
   int my1nt = (int) myDouble;
   Sys.out.println(myDouble); 9.78
   sys.out.println(my1nt); 9
```

sous-casting :

```
(fruits) orange; // orange transformée en fruits
```

EXP :

Héritage




```
Public class Fruit {
  int poids;
  Public Fruit(){
  Sys.out.println("création d'un fruit");
  public void affiche(){
  Sys.out.println("c'est un fruits");
}}
```

```
public class Pomme extends Fruit{
  Public Orange(int P){
    poids = P;
    sys.out.println("création d'une orange de "+poids+"gr");
  }
  Public void affiche(){
    sys.out.println("c'est une orange");
  }
  Public void afficheBoids(){
    sys.out.println("poids");
  }
}
```

```
Public class MonAppli{
  public static void main(String[] args){
    pomme P = new Pomme(40);
    orange O = new orange(50);
```

SunCasting :

(déjà ycébtemha comme super)

```
Fruit f1 = new orange(50); implicite
Fruit f2 = (Fruit) new orange(60); explicite
```

(Lazem ykoun fel héritage / Lazem ykébtemha)

Scanné avec CamScanner

```
main(--){
Fruit f1=new Pomme(60);
f1.afficheBoids();        Erreur
((Pomme)f1).afficheBoids();
```

erreur peut être évitée si vrai

sous-Casting (explicite)

sous: mère → fils

sur: fils → mère

Droit d'accès

Log'm'heritage

relation d'héritage lie fils-père

hérite (père → fils)

- privé pour toutes les méthodes des classes appartenant aux autres package
- publique vis-à-vis d'ensemble des méthodes du package contenant la classe considérée

Public

(pas obligatoire)

- tjrs dispo
- meh kan methode sur ïn classe
- ma 3andhach 3alaga bel package

3la hbal dima hérite fiyh

Private

- l'attribut accessible seulement aux méthode de la classe. (il faut utiliser getters et setters)

accessible par les méthodes de cette classe

Protected

- ïn package
- les classes données (hybrides des accès précédents)
- attribut ou méthode protected est considérés comme publique ou privée

friendly

- ïn package
- accès par défaut (sans déclarer friendly) int a;

héritage

```
class A {
  private int x;
  private void f(){
      x++;
  }
  public void g(){
      f();
  }
  System.out.println(x);
}
```

```
class Test {
  public static void main(String[]a
  { A a = new A();
    a.g();
  }
```

- accessible par C1 : a, b, c, d
- ——— C2 : a, c, d
- ——— C3 : a, c

```
Package P1
public class C1 {
  public int a;
  private int b;
  protected int c;
    int d;
}
```

```
package P2
class C3 extends
  ...
}
```

friendly

```
class C2 extends C1{
  ...}
```

```
Package P1
Public class A{
  int a;
  Public int b;
  private int c;
  protected int d;
}
```

```
Package P2
public class C extends A{
  ...
}
```

```
Package P3 : sous Package P1
public class B{
  ...
}
```

pour la classe B : b

pour la classe C : b, d

- **Package** (app. développée fait appel à de nombreux cl
- package <nomDePackage>.<nomDeClasse>; La syntaxe Java permettant l'association d'une classe à un package est

Scanné avec CamScanner

```
package P1;
class A{...
}
```
A.java

```
package P1, P2;
import P1.A;
class B{ .*;
??
A a;
}
```
B.java — importe tout les classes

Rq: import <nom_Package>.<nom_class>;
import <package>;

*Les variables :

- variable d'instance (attribut d'objet)
- Variable de classe (static)
- Variable local

Remplace de variable globale

- variable local

```
void f(){
int n;
}
```

ou `for(int i=0, —)`

*Les constantes :

```
Public class A{
  final int m = 1;
```
ou bien
```
  A(int n){this.n=n};
  void set n(int n){
    this.n = n; //erreur
```

```
public class CC{ final int numcpt;
                 static int n=100;
public cc(){
  n++;
  numcpt = n;}
```
101
102
103
...

Pour déclarer une CST (final)

```
class A{
  final int x;
```
ou
Au niveau méthode/CST
```
  final void f()
```
* ما تنجمش تبدلها، أما تنجم تقرا
initialisation

Les types Primitifs :

String (objets) tableaux Vecteurs Wrappers(

Wrappers :

objet ⟷ type primitif

Création d'un Wrappers :

```
Double CA = 27.43;
```
ou
```
Double CA = new Double(27.43);
```
```
double x = CA.doubleValue();
```
objet → type primitif

ou bien
```
double x = CA
```

byte → Byte | int → Integ
float → Float | char → Charac

Cast :

Exp : int x ;

x = (int) 5.2344 ; // x = 5

x = int (5.2344) ; // autre écriture x = 5

String S = "1e3"

for (int i = 0 ; i < S.length( ) ; i++) {

S.charAt(i) ; ← retourne le caractère i

---

boolean Test = true ;

for (int i = 0 ; i < S.length( ) ; i++) {

if (! Character.isDigit (S.charAt(i))) {

méthode de la classe Character

Test = false ;

break ; }

Class Personne {

private int id ;

Private String nom ;

Public Personne (int id, String n) {

{ this.id = id ;

this.nom = n ;

}

Public String toString ( ) {

String s = "id" + id + " " + Nom ;

return s ;

}

redéfinition

Class Test {

main ( ) {

{ Personne P = new Personne (100, "BenMed") ;

System.out.println (P) ;

pour cette instruction il faut

affiche id 100, BenMed

on remplacera par la méthode toString (affiche)

taille du tableau

* Tableaux : int [ ] montableau = new int [10] ;

int [ ] t ;
ou
int t [ ] ;
int a, b [ ] ;

} t = new int [20]
int [ ] t = new int [20]

كيف نتعرف على طول tableau

tableau ( ) ← objet

length ← exp

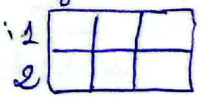
Personne tab [ ] = new Personne [20] ;

for (int i = 0 ; i < tab.length ; i++) {

tab [i] = new Personne (1000, "—") ;

* Matrice : (Tableaux de Tableaux)

int [ ][ ] Mat = new int [2][3] ;



Mat [0][0] = 2 ;

for (int i = 0 ; i < 2 ; i++)

for (int j = 0 ; j < 3 ; j++) ...

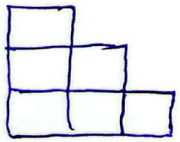
Scanné avec CamScanner

`int[] a, b[];` et `int a[], b[][];`

`a` : tableau d'entiers ; `b[]` : tableau de tableaux ; `a[]` : tableau d'entiers ; `b[][]` : tableau de tableaux

```
int[][] M = new int[3][];
for (int i=0; i<4; i++)
    { M[i] = new int[i+1]; }
```

(`M[i]` : ligne)



Initialisation d'un tableau :

```
double[] montab = {4.3, 4.6, 75}
int[][] tab = {{1,7},{23,22},{14,55}};
```

tableau de 3 tableaux de d'entiers

Copier des tableaux :

(taille du tableau ne fixe)

```
int n = x.length;
int[] y = new int[n*1.5]   // nouveau tableau plus grand 50%.
for (int i=0; i<n; i++){   // copie de valeur entre les tableaux
    y[i] = x[i];
}
```

⇔ `System.arraycopy(x,0,y,0,n);`

(source, début src, Début dest, nb éléments)

* Les collections

(= tab dynamique d'obj de type Object)

Type de collections : ArrayList, Vector, Iterator ...

Vector

(différence avec tableaux ; ils ne stockent que des objets (+ Strings))

- il faut importer :
  `import java.util.Vector`
  (désavantage : taille dynamique. n'est pas fixée)
- utilise un méthode : `monVecteur.method();`

Exp :
- void addElement(Object)
- Object elementAt(int) : retourne l'élt qui se trouve à l'indice (int)
- boolean isEmpty() : teste si vide.
- int size() : taille d'objets. (le vecteur a)

`Vector monVect = new Vector(40);` // construction d'un vecteur de taille 40

ArrayList : (classe du Package java.util.)

`import java.util.ArrayList;`

- Déclaration : type : String, Integer, ... ~
  `List<Fruit> fruits;`
  `List<Integer> = new ArrayList<Integer>();`
- Création de la liste :
  `fruits = new ArrayList<Fruit>();`
- Ajouter des objets de type Fruit à la liste
  `fruits.add(new Pomme(30));`
  `fruits.add(new Orange(40));`

Scanné avec CamScanner

Des méthodes:

- .length() → renvoie la longueur de la String
- .indexOf() → la position d'un caractère
- .substring() → extraction d'une sous-chaîne
- .toUpperCase() → mise en majuscules
- .toLowerCase() → mise en minuscules

Exp 1:

```
String ch = "GCR";
ch.indexOf('G')
ch.substring(1,2)
```

(renvoie 0 (le rang du caractère G) ou, renvoie -1 si G n'est pas trouvé)

↓

retourne la sous-chaîne extraite de ch
'CR'

Exp 2:

```
String S1, S2;
S1 + S2 ; concaténation
char C1, C2;
C1 + " " + C2 ; Pour que ce résultat est String
il faut ajout " " pour 2 caractères
```

S1.equals(S2); → True / False

S.length() (String)	tab.length (tableau)	f.size() (liste/vecteur)
↓ meth	↓ variable	↓ meth
nbre de caractère dans une chaîne	taille max d'un tableau	nbre d'élt d'une liste/vecteur

Les fonctions pour les String: classe "StringBuffer"

Un objet de type StringBuffer peut modifier la chaîne (insertion, concaténation....)

Exp:

```
StringBuffer Sb = new StringBuffer("GCR");
System.out.println(Sb); // ≡ Sb.toString();
Sb.setCharAt(2,'V'); → "GCV"
Sb.append("x"); → "GCVx"
Sb.insert(3,"-"); → "GCV-x"
```

modification d'un caractère
ajout d'une chaîne en fin
insertion d'une chaîne en remplacement donné

Scanné avec CamScanner

Scanné avec CamScanner

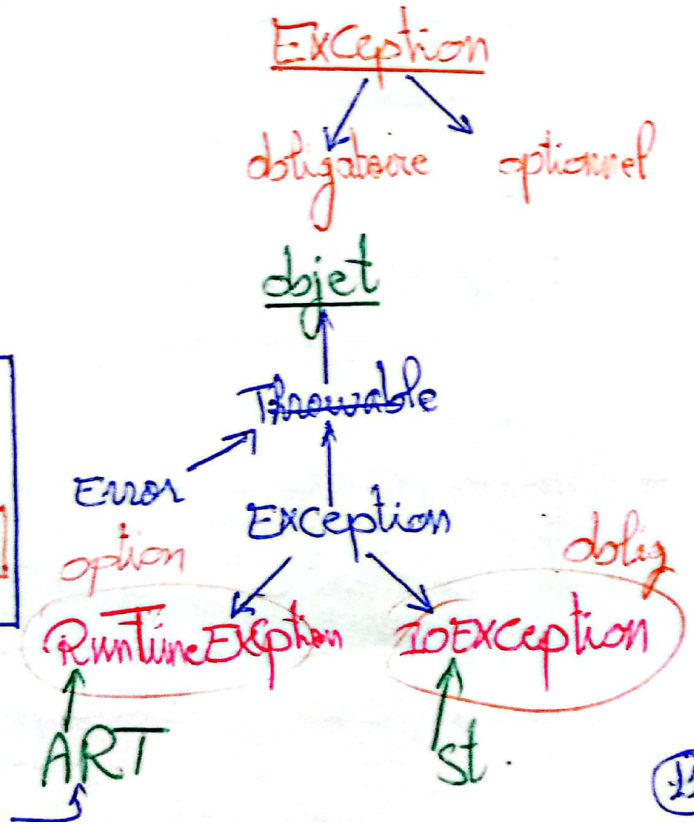
Les Exceptions :

```
class Test{
 main(){
 int a=e, b=0;
  S = a/b
  int t[] = new int[5];
  for(int i=0; i<10; i++)
   int x = Integer.parseInt("12a");
  Excep. optionnel
```

ART Exception
ART / ByZero
// Erreur inattendu
Lecture à partir

try comme cond° → si on a une erreur on va à l'exception catch

ArithmeticException



Exemples d'Exception

* IOException : fichier inconnu / protégé en écriture / lecture
* IndexOutOfBoundsException : accéder une position dans un tableau qui dépasse les limites de celui-ci.
* ArithmeticException : problème d'une expression mathématique (type, la division par zero)

la classe java.lang.exception ↳ hérite de java.lang.Throwable (java.lang.Error)

```
class A{
 void f(){
  try{ ..... ;
       ..... ;}
  catch(Exception e){
  Sys.out.println("Exception");
  Sys.out.println("message");
  sys.out.println(e.getMessage());
  sys.out.println(e.printStackTrace);
```

si erreur
si pas d'erreur

Scanné avec CamScanner

Exp complet:

```
public class TestDivision{
  public static void main(String[] args){
      int numer = 3;
      int denomin = 0;

  try{
    System.out.println(numer/denomin);
    System.out.println("message après la division");
    }
  catch(ArithmeticException e)
        (IOException e)
        (Exception e)
    {System.out.println("Exception arithmetique");
    System.out.println("message après catch");
    }
  }
```

Finally{...} un bloc catch (optionel)

Exp: class Test{
main(){
try{ _____;
_____;
_____;
catch( ){ _____ }; (1e)
finally{ co.close(); }

Throws Traitement de l'exception renvoi

```
class Test {
Public static void main(m) { int a = 2, b = 0;
    try { m1(a,b);}
    catch (ARITH) { Sys("Exception"); } ;
  }
Public int m1(int x, int y) throws ARTException{
 return m2(x,y); }
Public int m2(int i, int j) throws ARTException{
    {return (i/j)}
}
```

renvoyer pour ; faciliter la l'exception

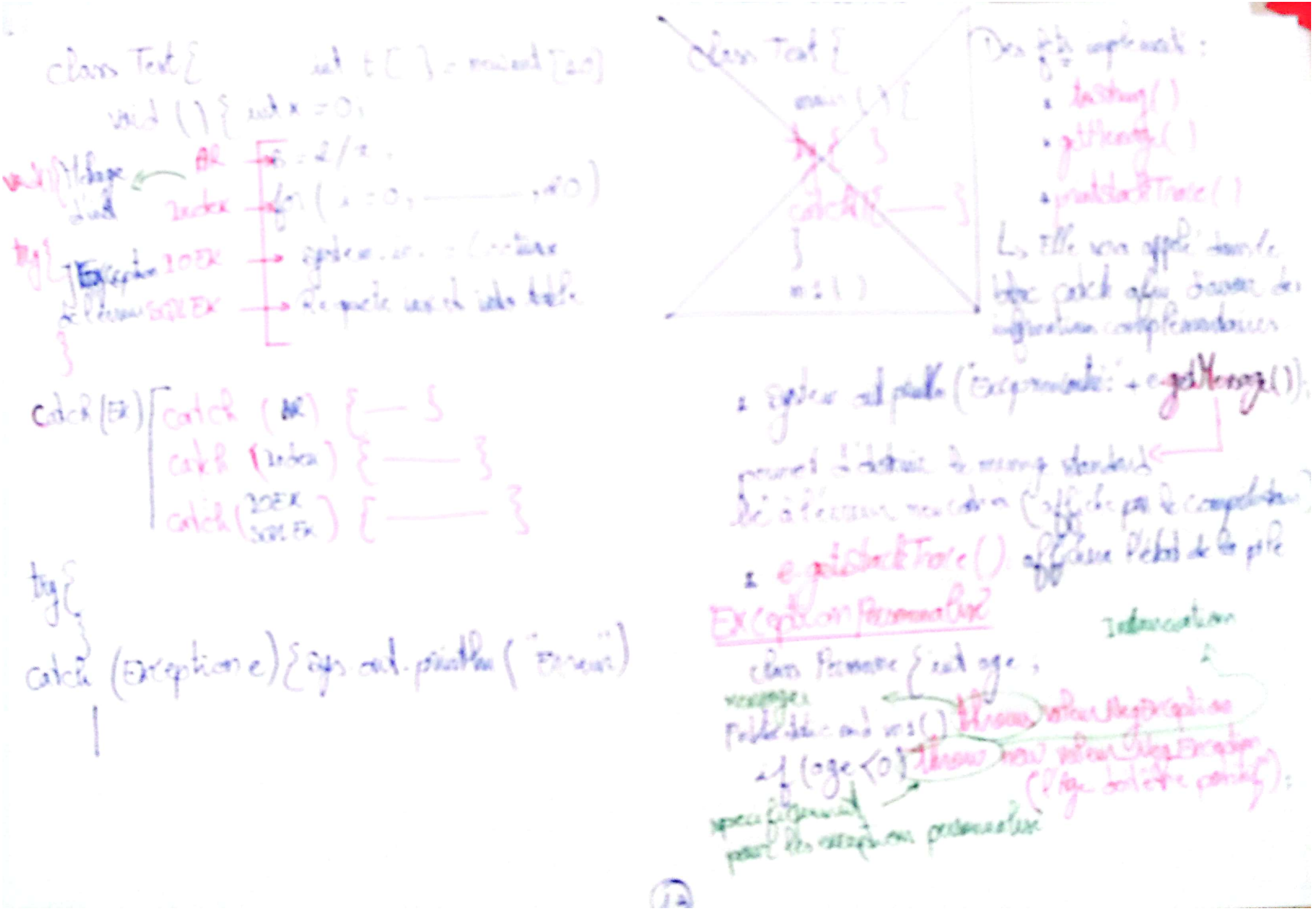
2/0

Exception ≡ Erreur inattendu (Erreur issue de l'utilisateur (n'issue de codage) codage à utiliser)
≠ mauvais codage

* l'appel de la méthode m1 entraîne de la méthode.

on ajoute "," après throws Exception

si on a plusieurs Exception


class Test {
void f() { int x = 0;
try {
Exception
Throwable
Error
ArithmeticException
NullPointerException
catch (Exception e) {
catch (Throwable t) {
catch (ArithmeticException | NullPointerException e) {
try {
catch (Exception e) { sys.out.println("Erreur")
class Test {
main() {
catch(Ex) {
m()
Des fct implémentés :
toString()
getMessage()
printStackTrace()
↳ Elle vont appel dans le bloc catch afin d'avoir des informations complémentaires
sys.out.println("Exception: " + e.getMessage());
permet d'obtenir le message standard
de l'erreur rencontrée (affiche pas le complément)
e.printStackTrace() affiche l'intégralité de la pile
Exception Personnalisé
Interaction
class Personne { int age ;
vérifier
Public static void m()
if (age < 0) throw new MonExceptionPerso
("Age saisie négatif");
Throws MonExceptionPerso
spécifier explicitement
pour les exceptions personnalisés

```
void main (){
  try {
    m1();
  catch( valeurNegException e){
      sys.out(e.getMessage());
```

```
class ValeurNegException extends Exception
    { public ValeursNegException(String s)
        { super(s); }
    }
```

```
catch (typeException e) throws typeException{
    throw new(typeException(args));
}
```

Les classes abstraite:

Scanné avec CamScanner

Scanné avec CamScanner

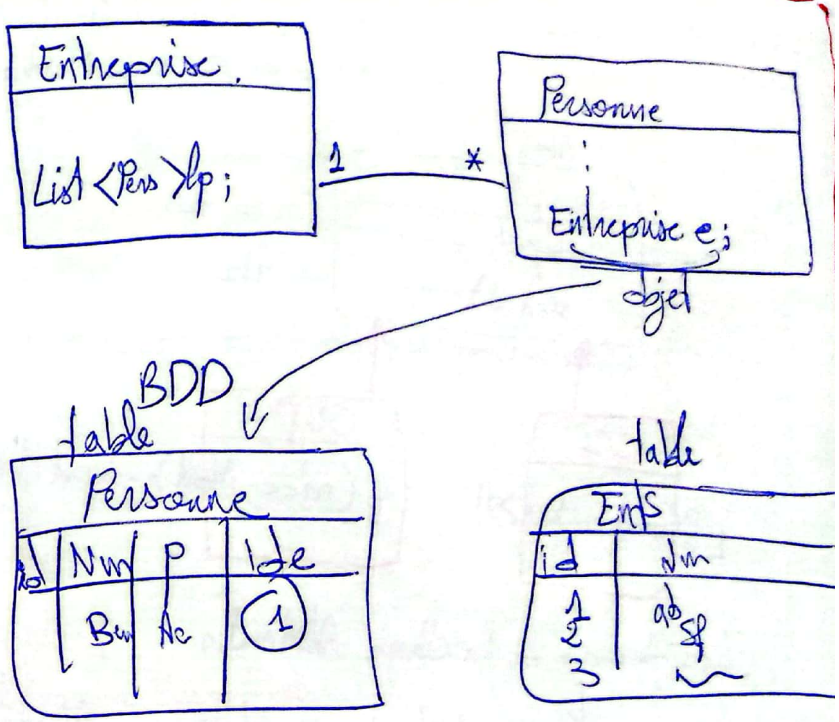
Exception Personalisé :

```
Class Personne { int age;
                 String nom;
public void m1() throws ValeurNegException
{ if (age < 0) throw new ValeurNegException("Age doit être positif");
}
```

Spécifiquement les Exception personalisé

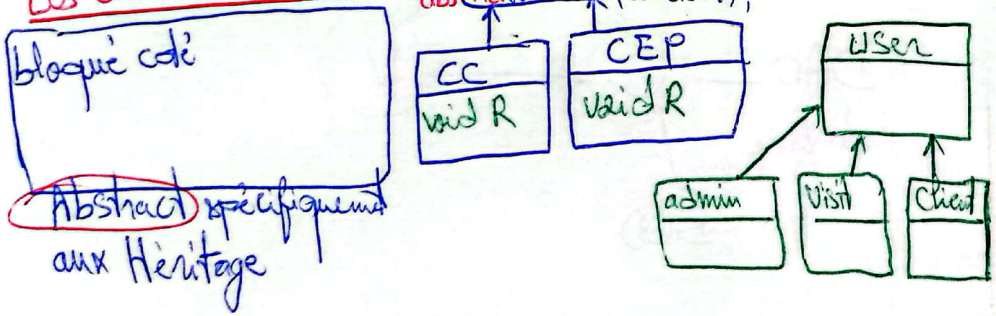
```
Class ValeurNegException extends Exception
{ String s
  public ValeurNegException(String s)
  { super(s); }
}
```

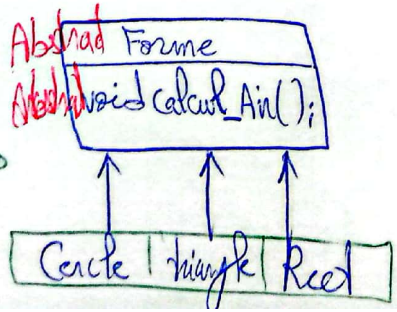
```
Void main()
{ try{
    m1();
  catch(ValeurNegException e){ System.out(e.getMessage()); }
```



40 | D

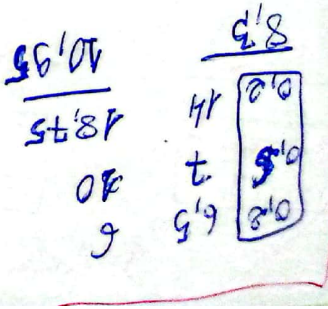
Les Classes abshaite



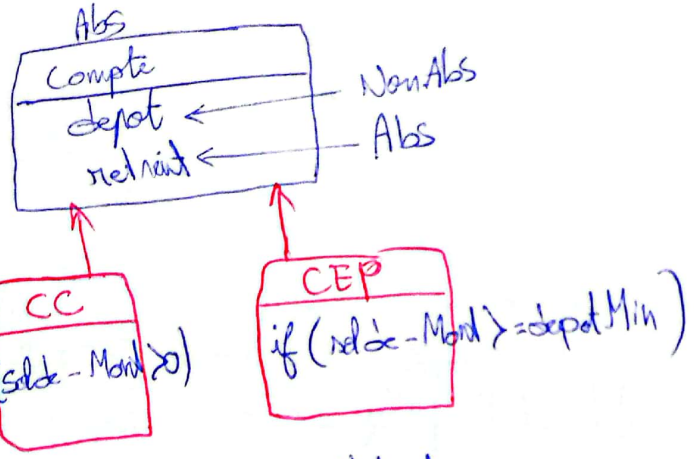


obligé de def les méthodes Abstrait pour que les Chemins A n Ab

4



Les classes Abstraites: → on ne peut pas instancier des objets
↳ ∃ une ou plusieurs Méthodes Abstraites

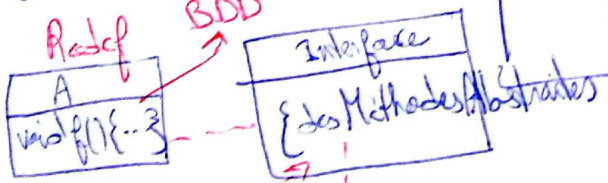


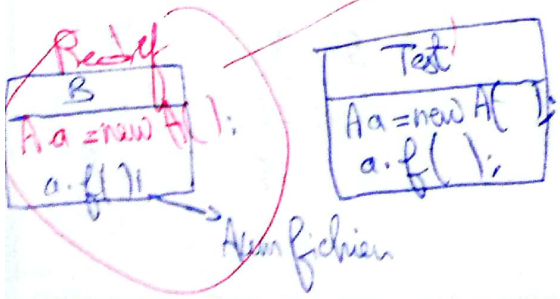
Les dérivés de la Classe Abstraites
↓
rédifinition de toutes les Méthodes Abstraites

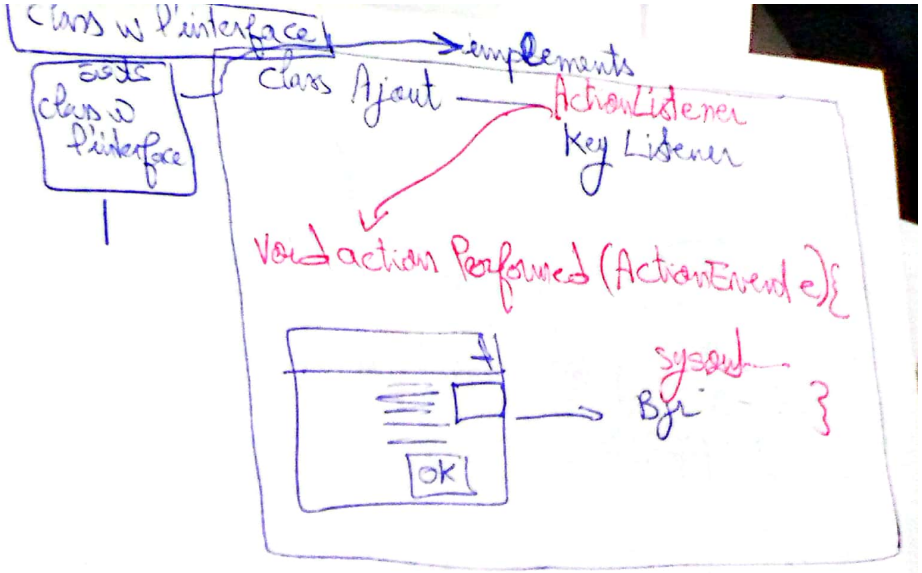
Les interfaces:

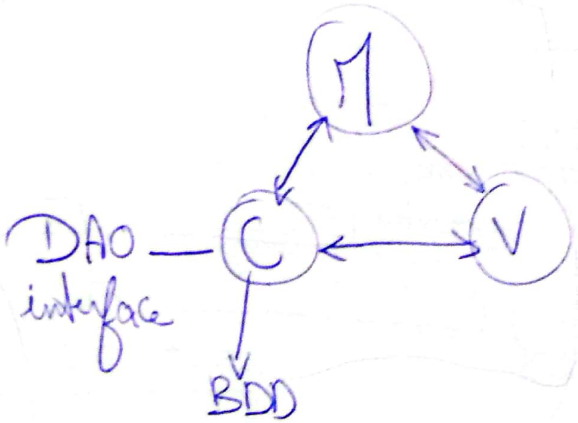
comptage fort → faible

Interface de méthode abstraite:
ActionListener
key listener









Scanné avec CamScanner

```
interface EtudiantDAO {

List<Etudiant> le = new ArrayList<Etudiant>();
Etudiant Etu = new Etudiant( );

public List<Etudiant> getAllEtudiant ( ){
    return le ;
}
public void addEtudiant (Etudiant e){
    le.add(e);
}
public Etudiant getEtudiantById (int id){
    for(Etudiant E : le){
        if (E.getId() == id)
            return E ;
    }
public
```

```
Class Etudiant {
int id;
String nom;
Groupe gp;
//Setters/Getters
}

Class Groupe{
int id gp;
String nomgp;
}
```

```
public interface IGestionEtudiant{
    void addEtudiant (Etudiant e);
    List <Etudiant> getAllEtudiant();
}
```

```
public class GestionEtudiantImp implements IGestionEtu
    List <Etudiant> le = new ArrayList<>();
public void addEtudiant (Etudiant e)
    le.add(e);
}
public list getAllEtudiant(){
    {return le;}
```


Scanné avec CamScanner

ENIG

Couche Model

```
class Etudiant {int id;   // clé primaire
                string nom;
                Groupe gp;   // BDD table Etudiant, clé étrangère
                ...
                }

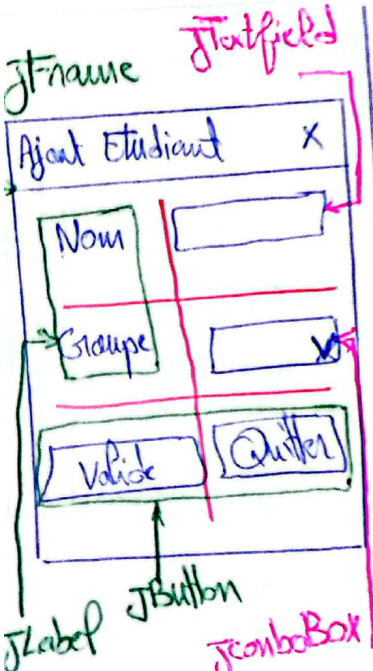
class Groupe {int id;
              string nom;
              ...
              }
```

Couche DAO

```
public interface IGestion{
    public void addEtudiant(Etudiant e);
    public List<Groupe> getAllGroupe();
}
import java.sql.*;   // ← connection
public GestionImp implements IGestion{
    Connection con;
    public GestionImp(){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            String URL="jdbc:MySQL://localhost:3306/Nom_BD";
            con=DriverManager.getConnection(URL);
        }catch(Exception e){-}
    }
}
```

Couche View

```
import javax.swing.*
class MaFenetre{
    String[] t={"Gst","GEt",...}
    JFrame f=new JFrame("Ajout Etudiant");
    JLabel lnom=new JLabel("Nom"),
           lgp=new JLabel("Groupe");
    JTextField tfnom=new JTextField(...);
    JComboBox cmbgp=new ComboBox(t),
    JButton bv=new JButton("Valider"),
            bq=new JButton("Quitter");
    public MaFenetre(){
        // Gestion des Layout;
        f.setLayout(new GridLayout(3,2)
```



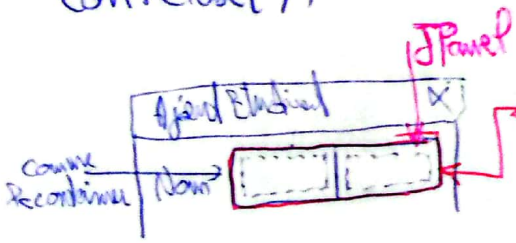
(les éléments elli T7ebhom yosiyou)

3	V
1	
2	
6	

```
public void addEtudiant(Etudiant e){
    String req="insert into Etudiant
        (nom,idgp) values (?,?)";
    try{
        PreparedStatement ps=con.prepareStatement
            (Req)
        ps.setString(1,e.getNom())
        ps.setInt(2,e.getGp().getId());
        ps.executeUpdate();
    }catch(Exception e){-}
    con.close();
```

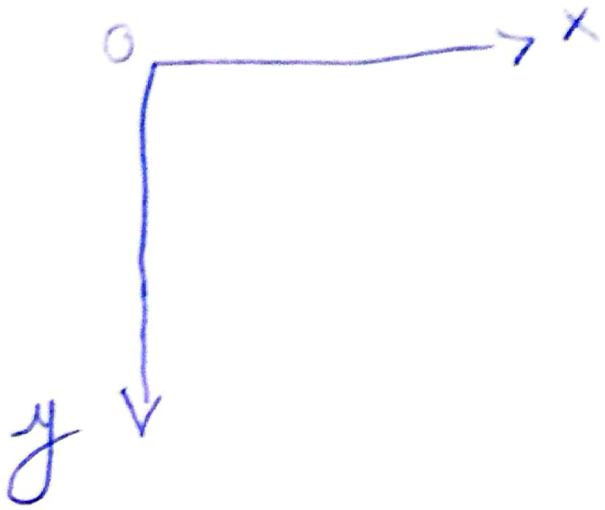
```
f.add(lnom); f.add(tfnom);
f.add(lgp); f.add(cmbgp);
f.add(bv); f.add(bq);

f.add(p); p.setLayout(new
GridLayout(1,2)); p.add(1);
p.add(2);
```




```
GestionImp gg=new GestionImp();
List<Groupe> lg=gg.getAllGroupe();
```

Scanné avec CamScanner




```
for(Graphe p:lg)
    cmbg.add(p.getName());
f.setVisible(true);
f.setBounds(200,200,300,200);
```

position — taille de la fenêtre

Gestion Events

1) Implementer l'interface d'écoute
   Exp: "ActionListener"
2) Mettre le composant à l'écoute
   Exp: "bv.addActionListener(this);"
3) Redéfinir les méthode des l'interface d'écoute

```
public void addElement(Element e) {
    String req = "insert into Element (nom, prenom) values (?,?,?)";
    values (?,?,?);
    try {
        PreparedStatement ps = con.prepareStatement(req);
        ps.setString(1, e.getNom());
        ps.setString(2, e.getPrenom());
        ps.setInt(3, e.getAge());
        ps.executeUpdate();
        con.close();
    } catch (Exception e) {
        ...
    }
}
```

```
public List<Element> getAllElements() {
    List<Element> list = new ArrayList<>();
    String req = "select * from Element";
    try {
        PreparedStatement ps = con.prepareStatement(req);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            Element e = new Element(rs.getInt(1), rs.getString(2), rs.getInt(3));
            list.add(e);
        }
    } catch (Exception e) {
        ...
    }
    return list;
}
```

```
import javax.swing.*
class Convert_EU_D {

    JFrame f = new JFrame("Convertisseur");
    JLabel le = new JLabel("Euro");
    JLabel ld = new JLabel("Dinar");
    JTextField te = new JTextField();
        String txe = te.getText();
        Double de = Double.parseDouble(txe);
    JTextField td = new JTextField();
        String txd = "" + dd;
        Double dd = Double.parseDouble(txd);
    JButton bc = new JButton("Convertir");
    JButton bq = new JButton("Quitter");
}
Public class fonction() {
        f.setLayout(new GridLayout(3,2));
```

```
b.addActionListener(this); le plus important

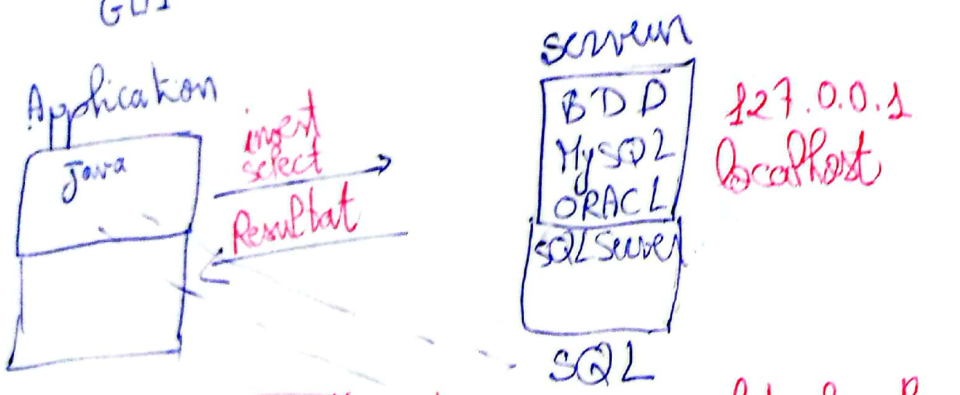
f.add(le); f.add(de); f.add(ld);
f.add(dd); f.add(p); f.add(bc)
GridLayout(1,2); f.add(bq);
f.show(); f.setBounds(100,200,200,200);
bv.addActionListener(this);
bq addActionListener(this);
}
Public void ActionPerformed(ActionEvent e){
    if(e.getSource() == bv
        String ve = te.getText();
                = td
```

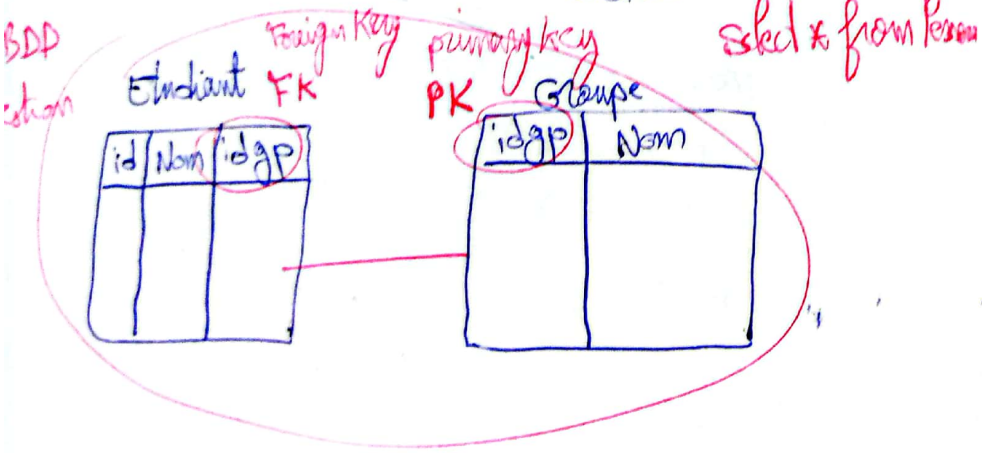
Scanné avec CamScanner

MVC ← Accès BDD

↕

GUI





0. Driver jdbc.jar

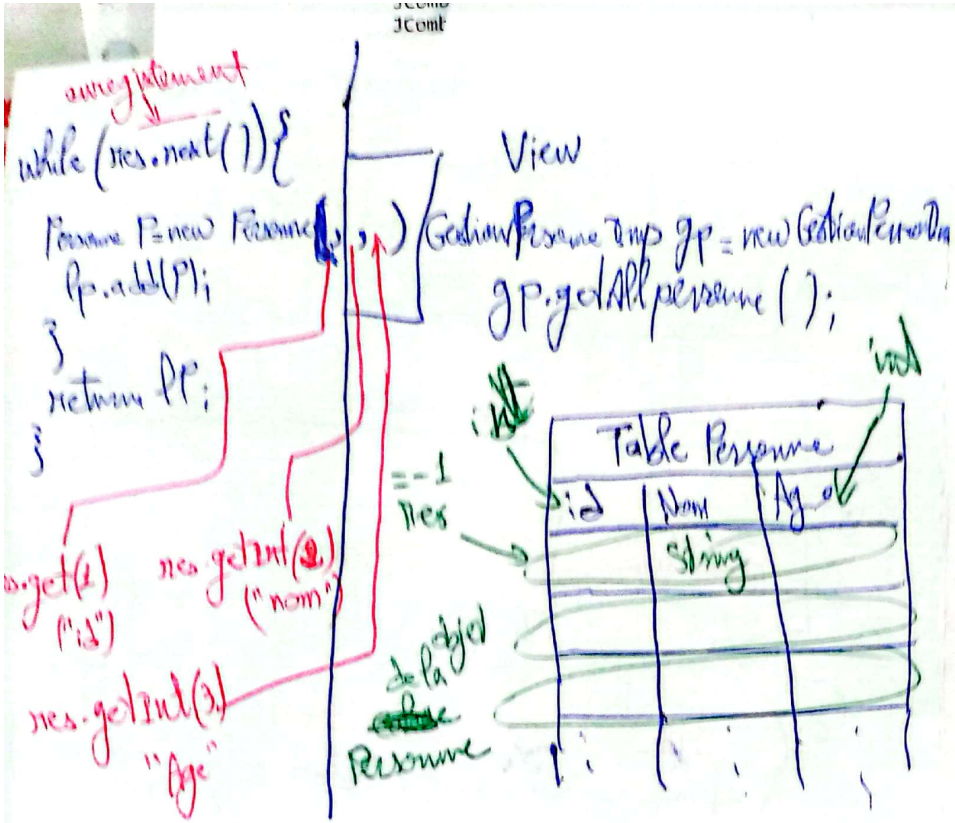
① Connexion à la BDD

② Interrogation de la BDD

;

```
Public class GestionPersonneImp implements IGestionPersonne {
    Public GestionPersonneImp( ){ }
    Public List<Personne> getAllPersonne(){
        Java.sql.*
        ① Chargement de driver (class.forName( type
        ② String url = "jdbc:mysql://localhost:port/NomBase
                       132.168.1.10   3306   par défaut: mysql
           connection con = DriverManager.getConnection(url,"root",
           catch (Exception e) { }
        Interrogation de la BDD
        ③ Statement stmt = con.createStatement();
           stmt.executeQuery("select * from personne");
```

```
Public class GestionPersonneImp implements IGestionPersonne {
    Connection con;
    Public GestionPersonneImp( ){ // { ① Chargement de Driver
                                    ② con = ... ouverture de connexion
    Public List<Personne> getAllPersonne(){
        List<Personne> p = new ArrayList<>();
        // ③ Interrogation de la BDD
        Statement stmt = con.createStatement( , );
        String req = "select * from Personne
                      nom                  where ligne
        ① ResultatSet res = stmt.executeQuery(req);
```




```
Public class GestionPersonneImp implements IGestionPersonne{
    Connection con;
    Public GestionPersonneImp(){ // { }
    Public Personne getPersonneByIdAndNom(int id, String nom) {
        String req = "select * from Personne where
                      id = " + id + " And Nom = " + nom + "
ou bien
        String req = "select * from Personne where
                      id = ? and Nom = ?"
        PreparedStatement ps = con.prepareStatement(req);
        ps.setInt(1, id);
        ps.setString(2, nom);
    ResultSet rs = ps.executeQuery();   // resultset pour le select car " Select elle renvoie une resultset
    void addPersonne(Personne P) {
        insert into Personne Values(?,?,?);
        ps.setInt(1, P.getId());
        2, P.getNom())
        3, P.getAge()).
```