TD : Ordres et treillis :

Morphismes entre 2 relations binaires.

Sur 2 ensembles $R_1 \subseteq E_1 \times E_1$ $R_2 \subseteq E_2 \times E_2$ $\subseteq$ = inclu ou égal.

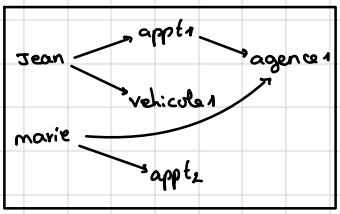
Est une application. $m : E_1 \longrightarrow E_2$ m donne une image dans E_2 à tout élément dans E_1.

Si un élément de l'ensemble de départ n'a pas d'image c'est une fonction.

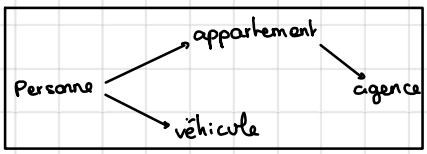
$m : E_1 \to E_2$ telle que $x, y \in E_1 . \ x R_1 y \Rightarrow m(x) R_2 m(y)$.

jean → appt1 = (jean, appt1) $\in R_1$ = jean R_1 appt1

R_1



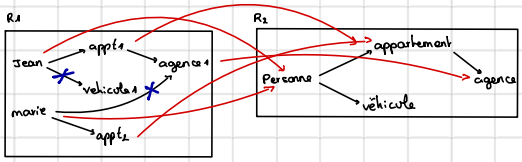
R_2



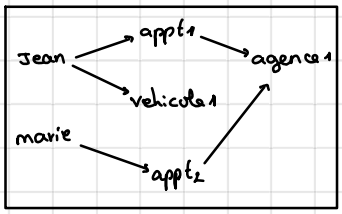
m est-elle un morphisme entre R_1 et R_2.

m(jean) = Personne
m(marie) = Personne
m(appt1) = Appartement
m(appt2) = Appartement
m(agence) = agence

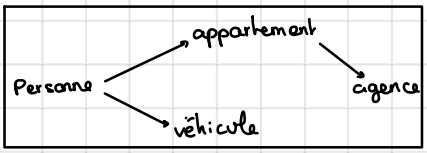
- Il n'est pas une application car véhicule 1 n'a pas d'image.
- Il manque la relation marie agence.
  Il manque m(marie) R_2 m(agence 1) et Personne R_2 Agence.



R_1



R_2

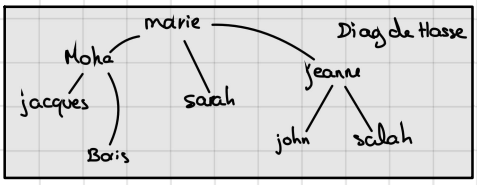


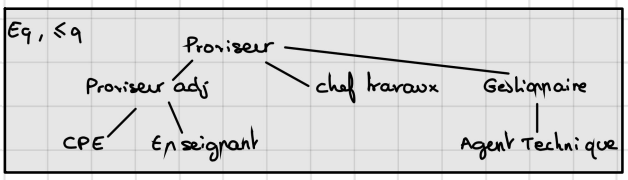
C'est une application.
C'est un morphisme. Tous les relations sont conservées.

m est-elle un morphisme entre R_1 et R_2.

m(jean) = Personne
m(marie) = Personne
m(appt1) = Appartement
m(appt2) = Appartement
m(agence) = agence
m(véhicule 1) = véhicule

Un morphisme entre 2 ensemble ordonnés $(E_p, \leq_p)$ et $(E_q, \leq_q)$
est une application $m : E_p \to E_q$ telle que $x, y \in E_p \ \ x \leq_p y \Rightarrow m(x) \leq_q m(y)$





Etablir un morphisme entre les 2 ensembles ordonnées.

m(marie) = Proviseur
m(sarah) = chef Travaux
m(jeanne) = Gestionnaire
m(john) = agent Technique
m(salah) = agent Technique
m(moha) = proviseur Adjoint
m(jacques) = CPE
m(Boris) = Enseignant

m(marie) = Proviseur
m(sarah) = chef Travaux
m(jeanne) = proviseur Adjoint
m(john) = CPE
m(salah) = Enseignant
m(moha) = Gestionnaire
m(jacques) = agent Technique
m(Boris) = agent Technique

Isomorphisme : $x \leq_P y \iff$ bijection.

$m(x) \leq_P m(y)$

Marie → Proviseur

Moh → Prov Adjoint

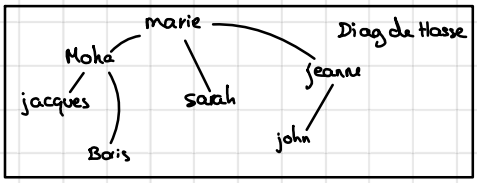
jacke → CPE

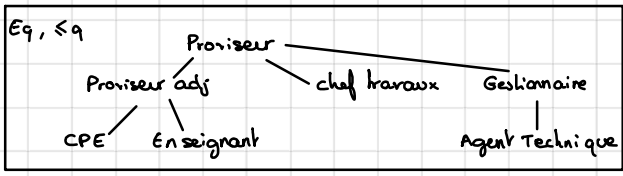
Boris → Enseignant

Sarah → chef travaux

jeanne → gestionnaire

john → agent tech





2)

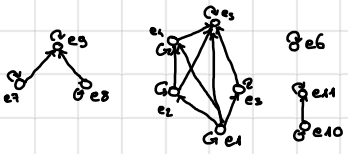


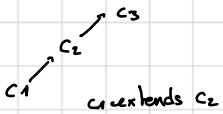
Diagramme de Hasse = Relation d'ordre.
1- Orienter les segments
2- ajouter réflexivité
3- ajouter transitivité.

Formaliser extends -

① Notation par l'ensemble des classes et l'ensemble des instances

② Notation par la relation : on peut garder extends

③ Propriétés de cette relation. Interne | réflexif
transitif
symétrique.



C_1 extends C_2

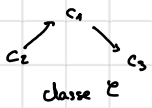
Rappels : Extends en java

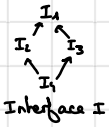
- nommer les ensembles
  - $\mathcal{C}$ classes
  - I Interface
- Etablir / formuler la relation par rapport à ces ensembles

extends $\subseteq (\mathcal{C} \cup I) \times (\mathcal{C} \cup I)$

- propriétées
  - extends $\subseteq (\mathcal{C} \times \mathcal{C}) \cup (I \times I)$
  - irréflexive
  - antisymétrique
  - non transitive

- Formaliser implements
  - nommer les ensembles





Class C_1 implements I_1

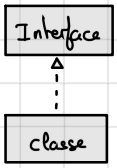
Class C_4 implements I_2, I_3 interface I_6

interface I_5 implements I_6

interface I_5 implements C_2

interface I_4 extends I_2, I_3

Implementation :



Formulation 1 $R \subseteq E \times E$

Implements $\subseteq \mathcal{C} \times I$

Implements $= \{(C_1, I_1), (C_4, I_2), (C_4, I_3)\}$

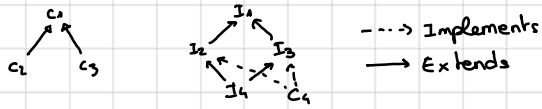
Ensemble des parties de l'ensemble (I)

Formulation 2

$P(I) = \{\emptyset, \{I_1\}, \{I_2\} \dots, \{I_1, I_2\}, \{I_1, I_3\}\}$

Implements $\subseteq \mathcal{C} \times P(I)$

Implements : $\{(C_1, \{I_1\}), (C_4, \{I_2, I_3\})\}$



Sous type : relation qui en Java se définit entre des types (on se limite aux classes et aux interfaces)
de cette manière :
Soient Tsup et Tsub des types (∈ C ∪ I)
Tsub est un sous type de Tsup
si on peut écrire une affectation du style vTsup = eTsub
où vTsup est une variable de Type Tsup et eTsub est une expression (valeur) de type Tsub

```
object o;
  o= "toto";
```

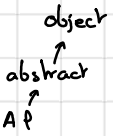
string est un sous type de object
Tsub ↓ Tsup ↓

o variable du type Tsup "toto" valeur du type Tsub

```
Tsup      Tsub
 ↓         |
véhicule v = new voiture();
```

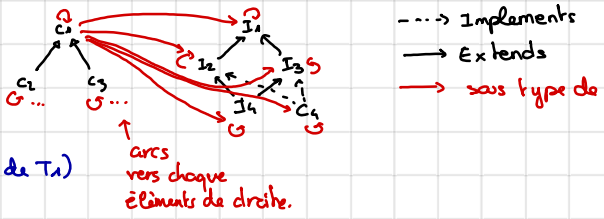
Formaliser cette relation sous type de.

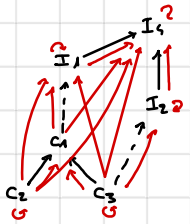
```
Object l;
l= new ArrayList<String>();
String s ="toto";
```



sous type de ⊆ (C ∪ I) × (C ∪ I)
Propriétée
Sous type_de ⊆ (C × C) ∪ (I × I) ∪ (C × I)
Il faut qu'elle soit réflexive
(sting sous type)

② transitive (Arraylist sous type de object)
③ antisymétrique (T1 sous type de T2 et T2 sous type de T1)
Sous type de = relation d'ordre
sous type de = fermeture réflexive et transitive
extends implements





Produit d'ordres :
Ici par 2 ordres
Soient 2 ensembles ordonnés $(P_1, \leq_1)$ et $(P_2, \leq_2)$
Produit direct $P = \prod_{1 \leq i \leq 2} (P_i, \leq_i)$

et un ensemble ordonnée $P = (X, \leq_P)$ $X = P_1 \times P_2$
et $(x_1, x_2) \in X$ et $(x'_1, x'_2) \in X$ $(x_1, x_2) \leq_P (x'_1, x'_2)$ si $x_1 \leq_1 x'_1$ et $x_2 \leq_2 x'_2$.

Dessiner le produit direct de $(P_1, \leq_1)$ et $(P_2, \leq_2)$

$(P_1 \leq_1)$: 3 > 2 > 1 > 1

$(P_2, \leq_2)$: b > a

$x = \{(1,a), (1,b), (2,a), (2,b), (3,a), (3,b)\}$

Produit d'ordres :

Définition générale : Soient n ensembles ordonnés $P_1, \dots P_n$ avec $P_i = (X_i, \leq_i)$ $1 \leq i \leq x$

Leur produit direct, noté $P = (X, \leq_P)$

$X = \prod_{1 \leq i \leq x} x_i$ $\quad (x = x_1 \times x_2 \dots x_n)$

Dessiner le diagramme de Hasse du produit

de $(X_1, \leq_1)$ 3 > 2 > 1 et $(X_2, \leq_2)$ b > a $\quad X_1 = \{1,2,3\}$

Paires
(1,a)
(1,b)
(2,a)
(2,b)
(3,a)
(3,b)}

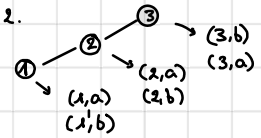
$(2,a) \leq_P (3,b)$

1 — 2 — 3

⇨

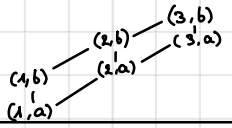
Etape 1 - Dessiner le diag de Hasse de $\leq_1$

Etape 2 - Remplacer chaque élément x de x_1 par le produit de $\{x\}$ par $\leq_2$.



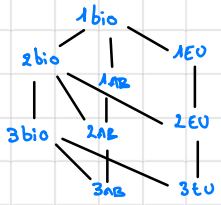
Produit de x avec $\{a,b\}$.

Etape 3 : On va relier les couples (x_1, x_2) et (x_1', x_2) lorsque $x_1 \leq_1 x_1'$

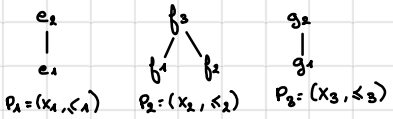


1) Prix euros : 1€ > 2€ > 3€

Label : Biocohérence — Eurofeuille, AB



Faire les étapes.

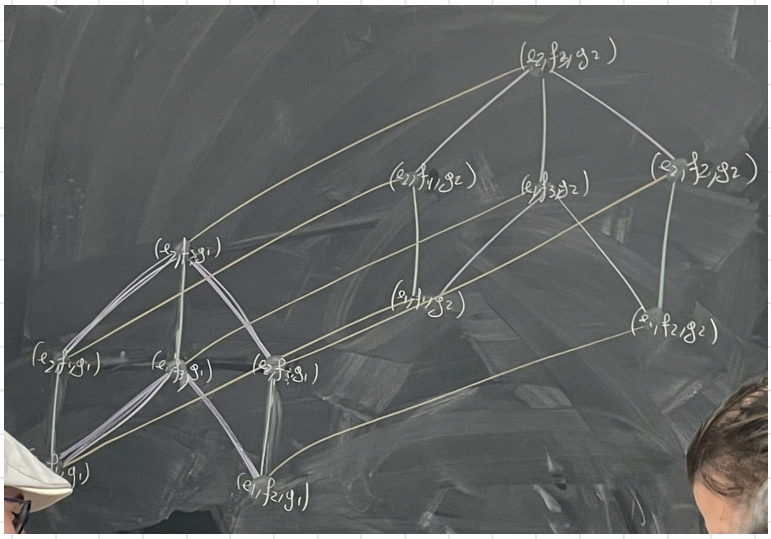


$P_1 = (X_1, \leq_1)$ $\quad P_2 = (X_2, \leq_2)$ $\quad P_3 = (X_3, \leq_3)$

$P = (P_1 \times P_2) \times P_3$.

(e_1, f_3) (e_1, f_1) (e_1, f_2) (e_1, g_2) (e_1, g_1)

(e_2, f_3) (e_2, f_1) (e_2, f_2) (e_2, g_2) (e_2, g_1)



Application en java.

```
public class C1 {
    protected C1 meth(Object o, String s)
                    Throws E1, E3, E4, Σ...}
}
```
S_1

```
public class C2 extends C1 {
    @override
    public C2 meth(Object o, String s1)
            Throws E2, E3 Σ...}
```
S_2

Règle override : elle définit un ordre partielle entre signatures. $S_1 \leq_{over} S_2$

- même nom
- même liste de types de paramètres
- Le type de retour peut être spécialisé
- La visibilité peut être augmentée
- Une exception peut disparaître
- Une exception peut être spécialisé
- La méthode n'est ni static, ni private.
- On peut passer de abstract à non abstract ou l'inverse.

On utilise 3 ensembles ordonnés

① ({public, protected, package}, $\leq_{visi}$)
public $\leq_{visi}$ protected $\leq_{visi}$ package

② $\leq$ subtype of

③ $\leq$ liste Exception