Qualité logicielle

Abderrahim El Jaouhary, Frédéric Malard

codeligne@gmail.com

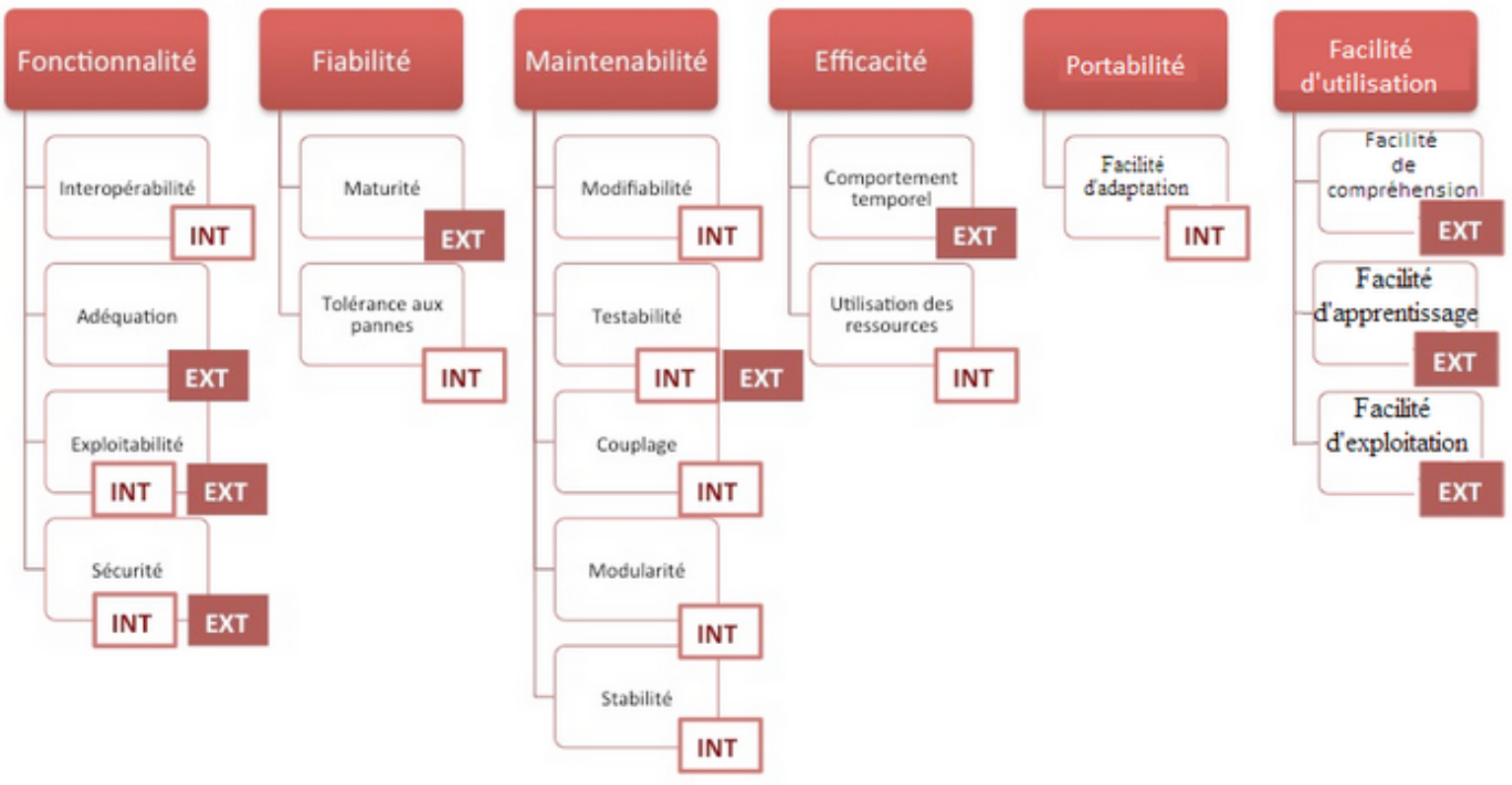
fred.malard2@gmail.com

18 décembre 2014

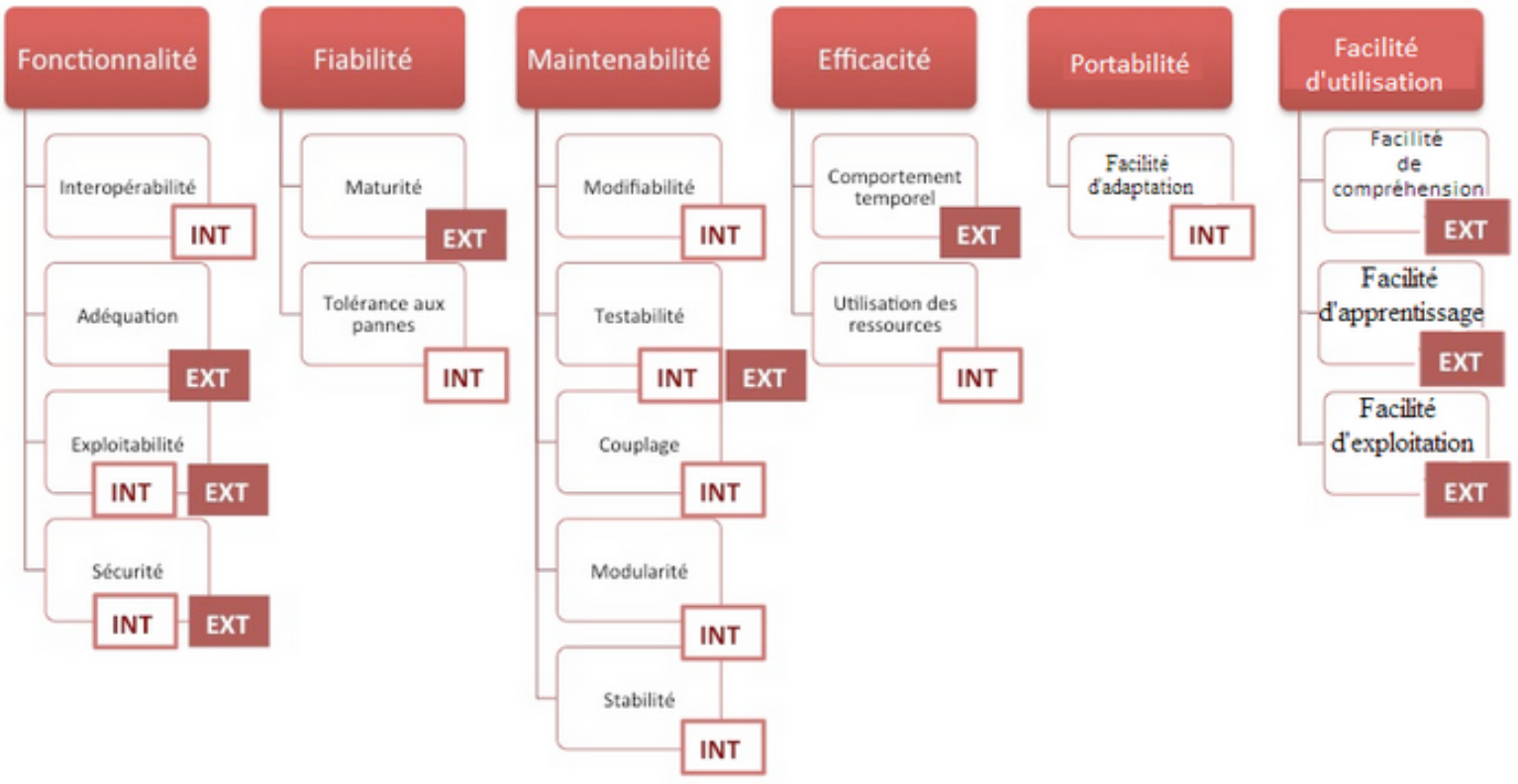
Plan

- Définition
- Historique
- Documentation et tests

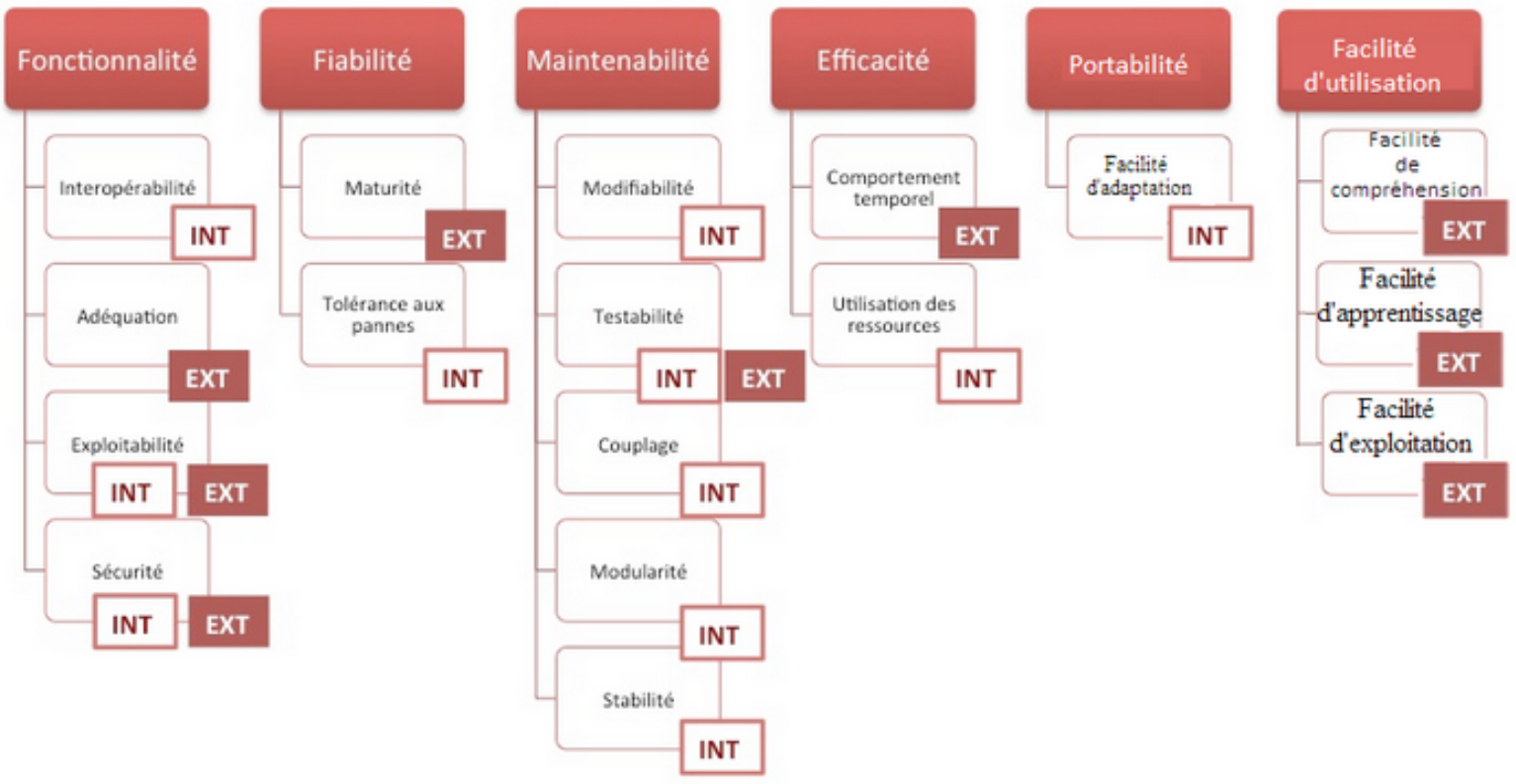
Plan



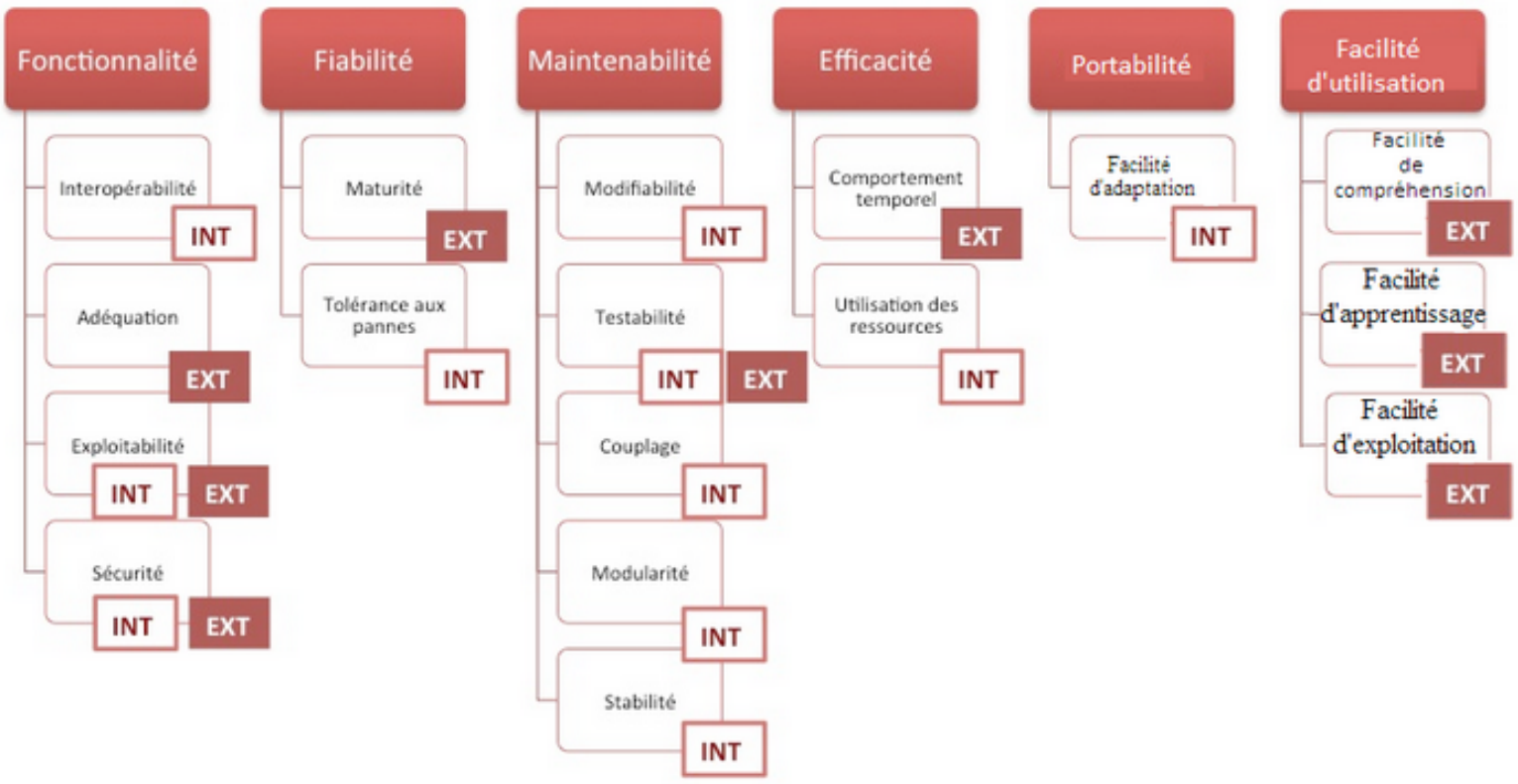
Fonctionnalité : le logiciel répond t-il aux besoins fonctionnels exprimés ?



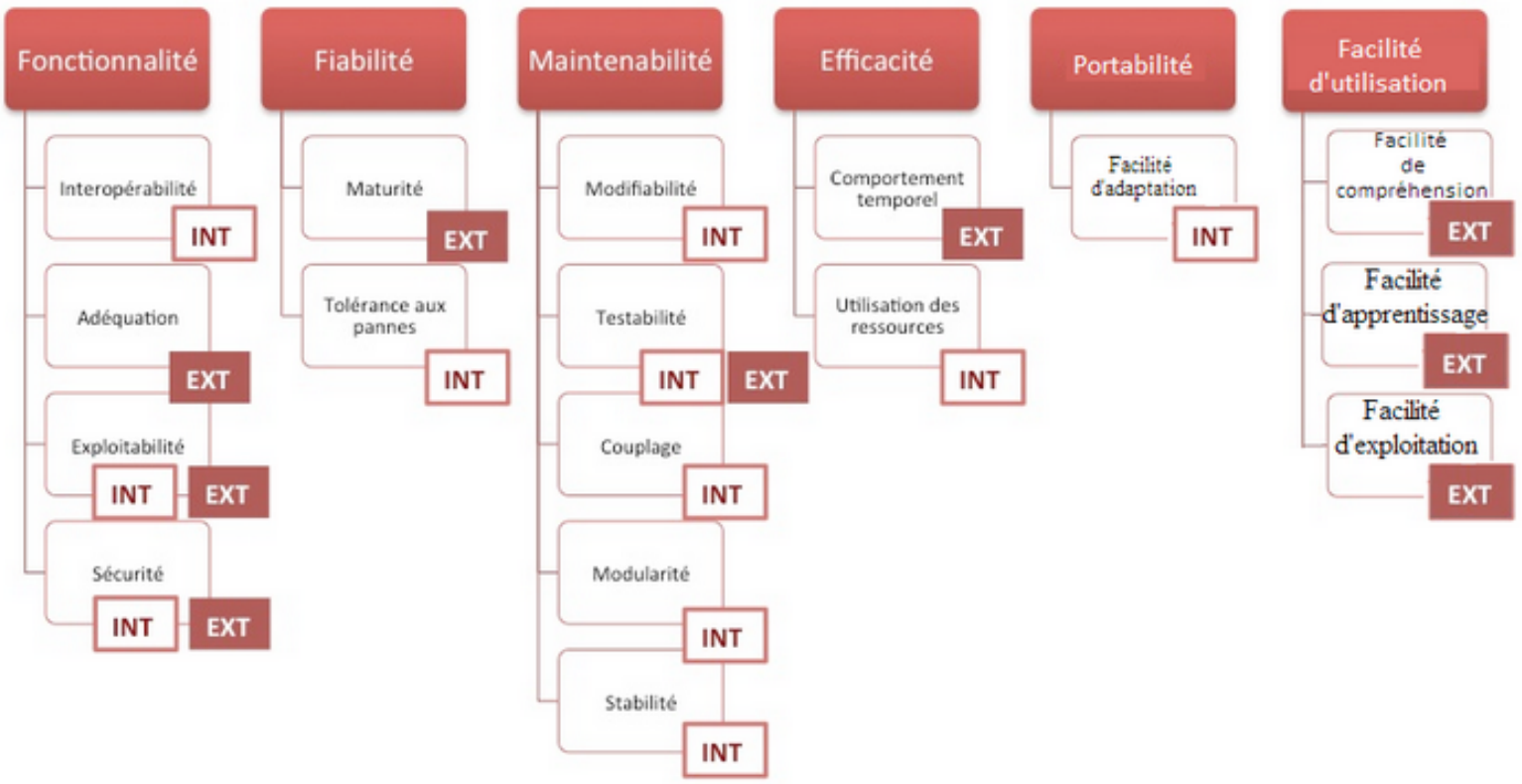
Fiabilité : conditions et durées du maintient du niveau de service



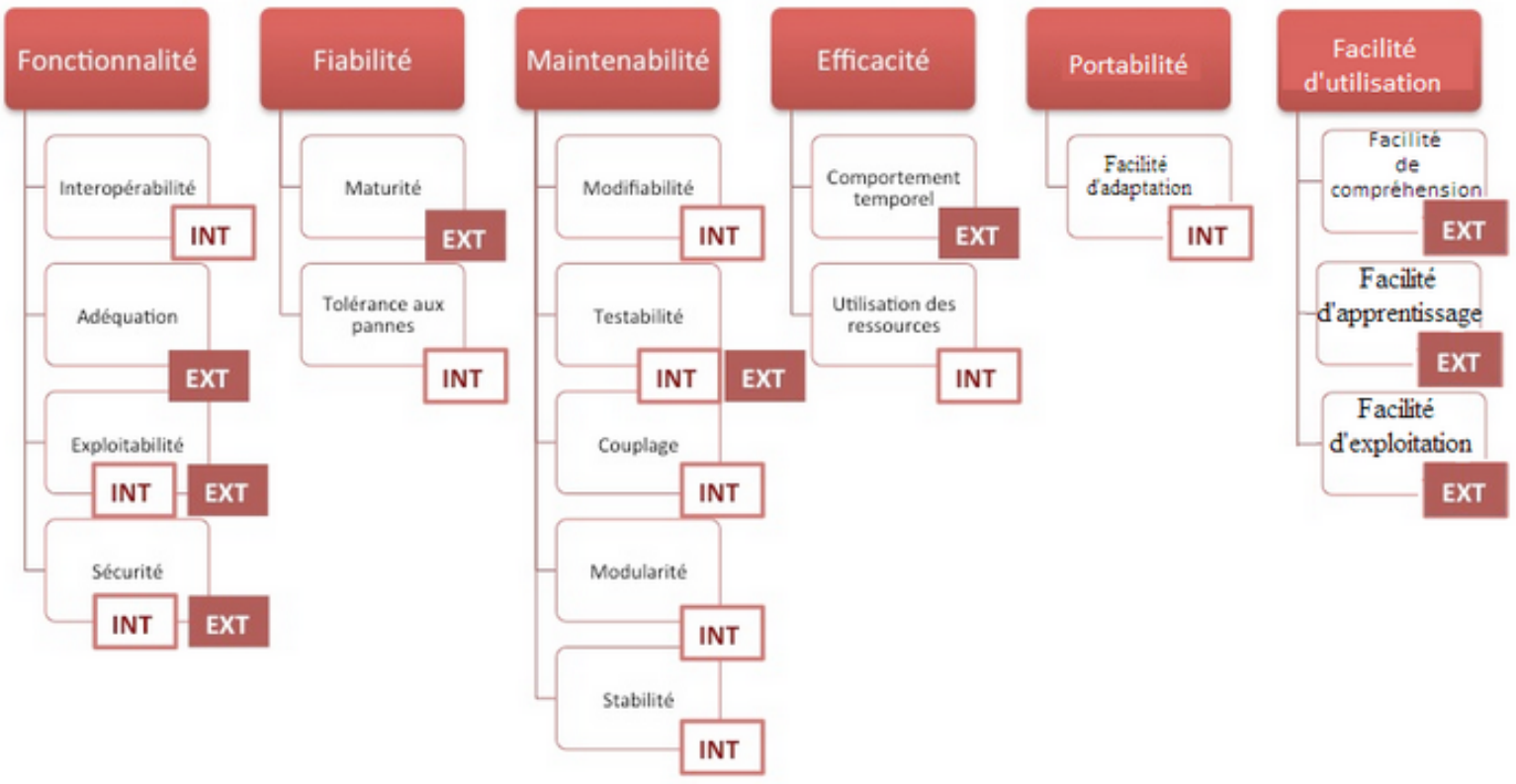
Maintenabilité : aisance a la correction et a la modification des fonctionnalités



Efficacité : résultats obtenus / ressources exigées



Portabilité : aisance a transférer le logiciel d'un environnement a un autre



Facilité d'utilisation : aisance a prendre en main le logiciel pour un nouvel utilisateur

Niv.	Environnement	Personnes	Économie	Application
D	petit dommage à la propriété	pas de risques pour les personnes	perte économique négligeable	loisirs, domestiques
C	dommage à la propriété	peu de personnes touchées	perte économique significative	alarmes de feu, contrôle de processus
B	dommage environnemental réparable	menace pour des vies humaines	grande perte économique	systèmes médicaux, systèmes financiers
A	dommage environnemental irréparable	des personnes mortes	désastre financier	systèmes de transport, systèmes du nucléaire

Figure: SCOPE

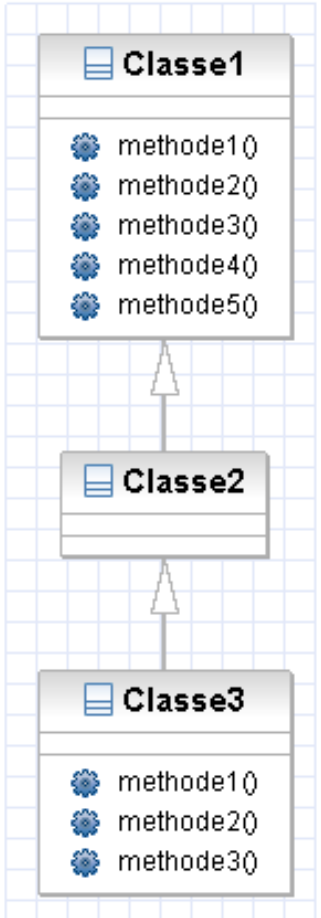
	Niveau D	Niveau C	Niveau B	Niveau A
Capacité fonctionnelle	test fonctionnel (boîte noire)	+ inspection des documents (listes de contrôle)	+ test des composantes	+ preuve formelle
Fiabilité	facilités des langages de programmation	+ analyse de la tolérance aux fautes	+ modèle de croissance de la fiabilité	+ preuve formelle
Facilité d'utilisation	inspection des interfaces utilisateurs	+ conformité aux normes sur les interfaces	+ test en laboratoire	+ modèle mental de l'utilisateur
Rendement	mesurage du temps d'exécution	+ test avec bancs d'essai (*benchmarks*)	+ complexité algorithmique	+ analyse des performances
Maintenabilité	inspection des documents (listes de contrôle)	+ analyse statique	+ analyse du processus de développement	+ évaluation de la traçabilité
Portabilité	analyse de l'installation	+ conformité avec les règles de programmation	+ évaluation des contraintes de l'environnement	+ évaluation de la conception des programmes

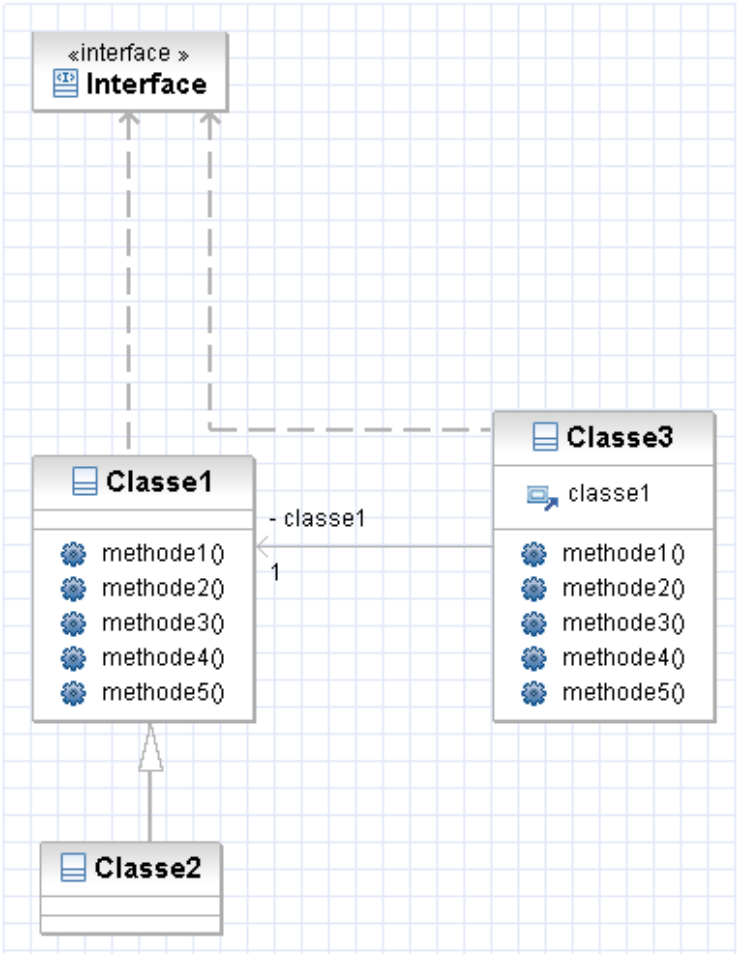
Figure: SCOPE

Plan

$$\frac{NORM \times DIT}{NOM}$$

NORM = nombre de méthodes redéfinies
DIT = profondeur de la classe dans l'arbre d'héritage
NOM = nombre de méthodes de la classe

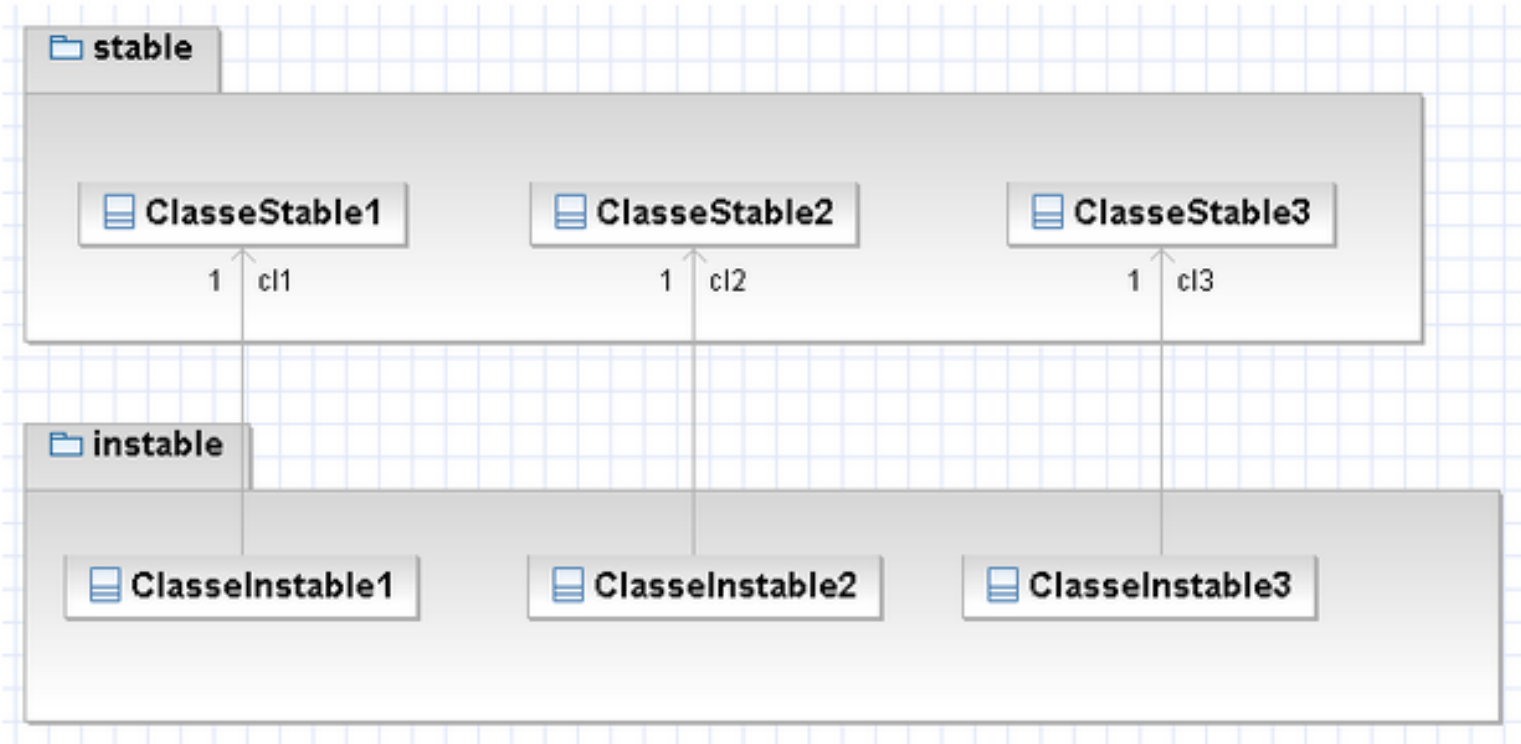

Classe1
methode1()
methode2()
methode3()
methode4()
methode5()
Classe2
Classe3
methode1()
methode2()
methode3()


«interface »
Interface
Classe1
methode1()
methode2()
methode3()
methode4()
methode5()
- classe1
1
Classe3
classe1
methode1()
methode2()
methode3()
methode4()
methode5()
Classe2

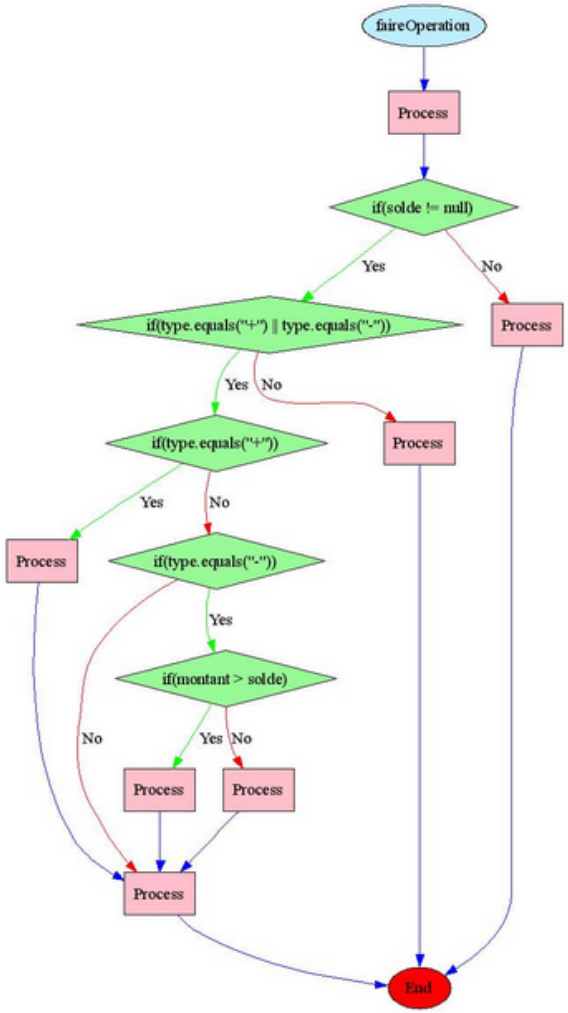
$$\frac{Ce}{Ca+Ce}$$

Ce = couplage efferent

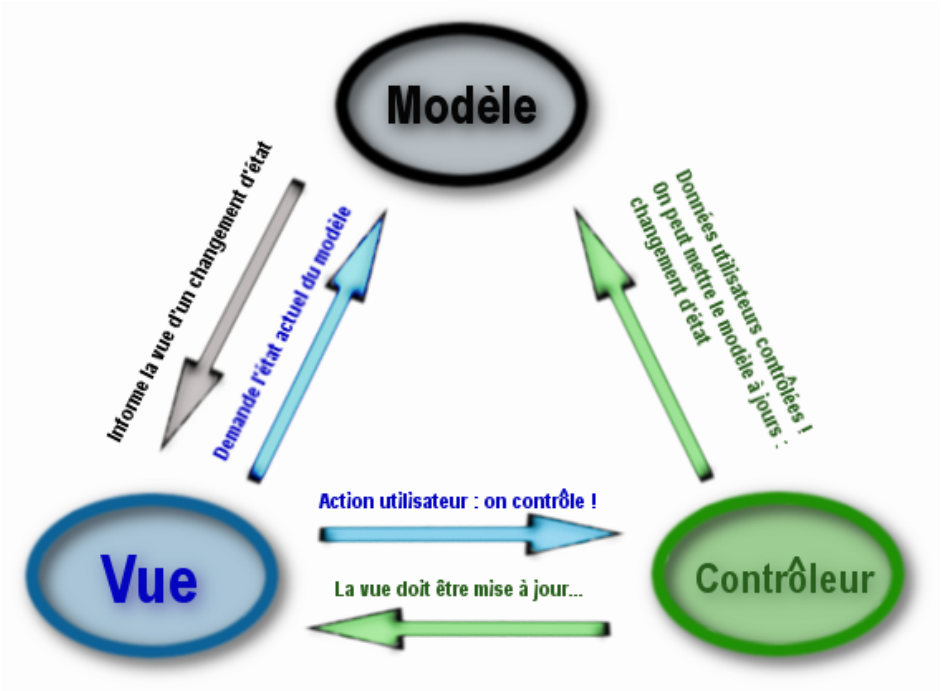
Ca = couplage afferent

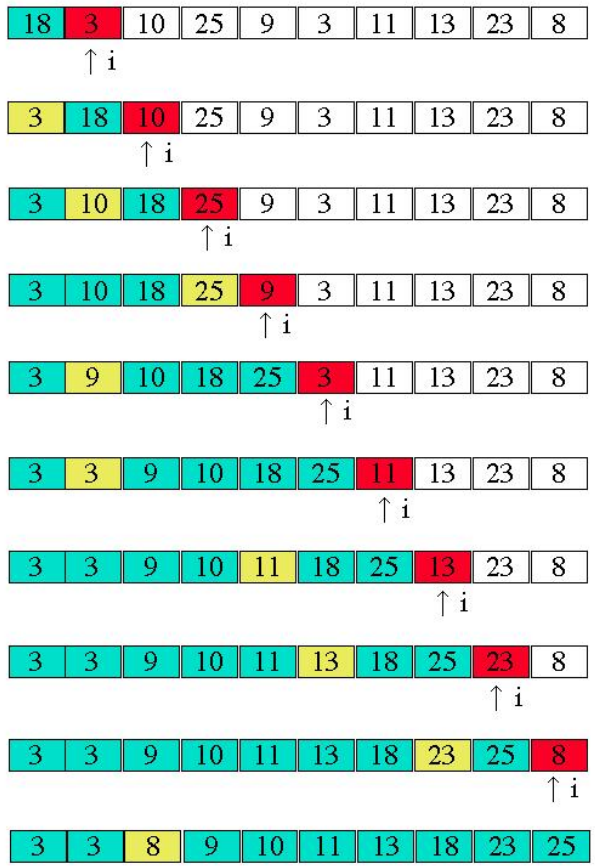

stable
ClasseStable1
ClasseStable2
ClasseStable3
1
cl1
1
cl2
1
cl3
instable
ClasseInstable1
ClasseInstable2
ClasseInstable3

```
package banque;

public class Banque {
        private Double solde;

        public void faireOperation(String type, double montant) {
                System.out.println("Début d'opération.");
                if(solde != null) {
                        if(type.equals("+") || type.equals("-")) {
                                if(type.equals("+")) {
                                        solde += montant;
                                }
                                if(type.equals("-")) {
                                        if(montant > solde) {
                                                System.err.println("Solde insuffisant !");
                                        }
                                        else {
                                                solde -= montant;
                                        }
                                }
                        }
                        else {
                                System.err.println("Type d'opération invalide.");
                        }
                }
                else {
                        System.err.println("Solde non initialisé.");
                }
    System.out.println("Fin d'opération.");
        }
}
```


faireOperation
Process
if(solde != null)
Yes
No
if(type.equals("+") || type.equals("-"))
Process
Yes
No
if(type.equals("+"))
Process
Yes
No
Process
if(type.equals("-"))
Yes
if(montant > solde)
No
Yes
No
Process
Process
Process
End

Plan


Modèle
Informe la vue d'un changement d'état
Demande l'état actuel du modèle
Données utilisateurs contrôlées !
On peut mettre le modèle à jours :
changement d'état
Action utilisateur : on contrôle !
Vue
Contrôleur
La vue doit être mise à jour...

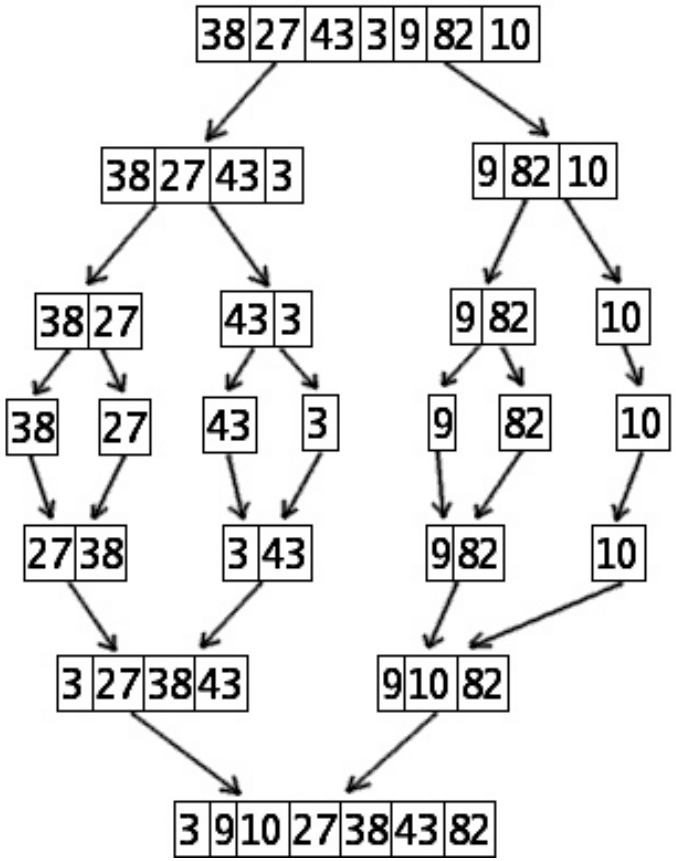


Tri par insertion

Algorithme 2 Tri d'un tableau par insertion

Requiert: un tableau tab[N] de N éléments
Fournit: le tableau trié par ordre croissant

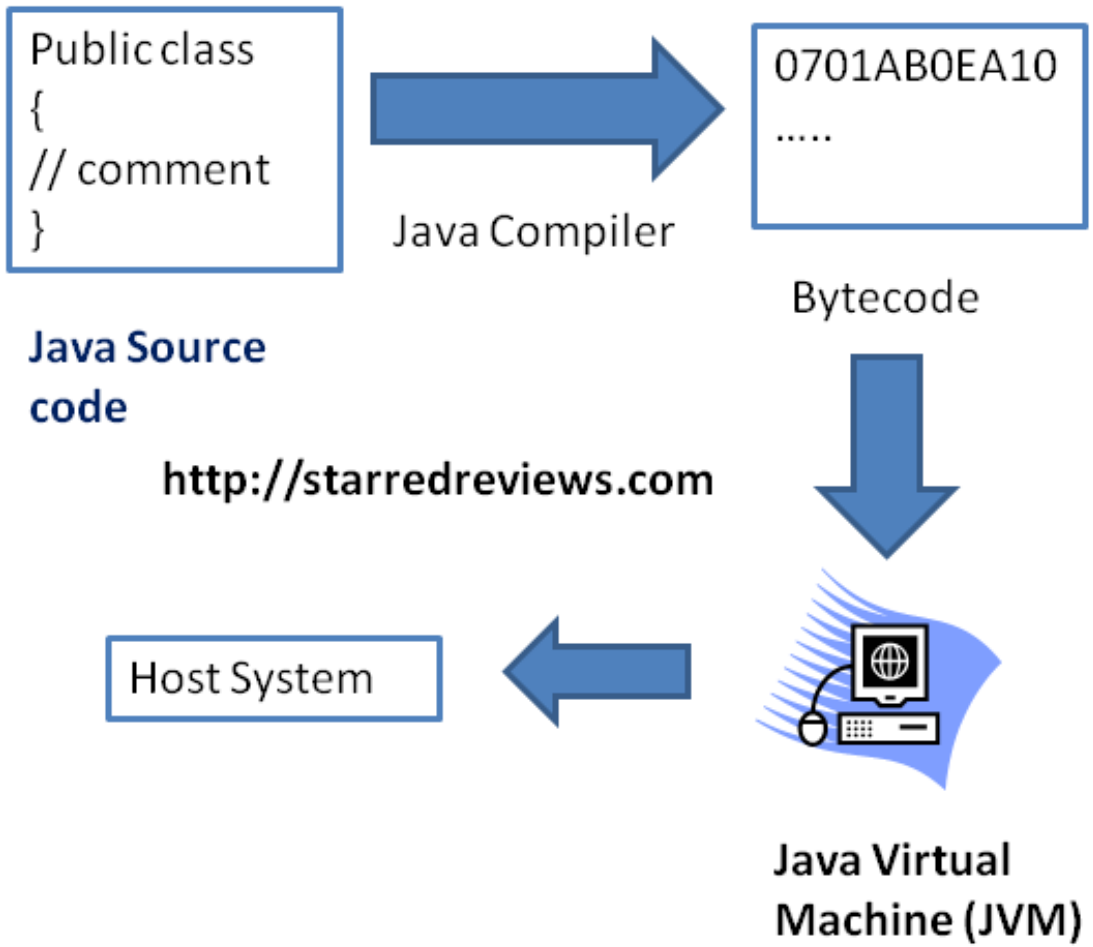
```
pour i = 1 à N faire
    tampon ← tab[i]
    j ← i − 1
    tant que j ≥ 0 ET tab[j] > tampon faire
        tab[j + 1] ← tab[j]
        j ← j − 1
    fin tant que
    tab[j + 1] ← tampon
fin pour
renvoyer tab[]
```

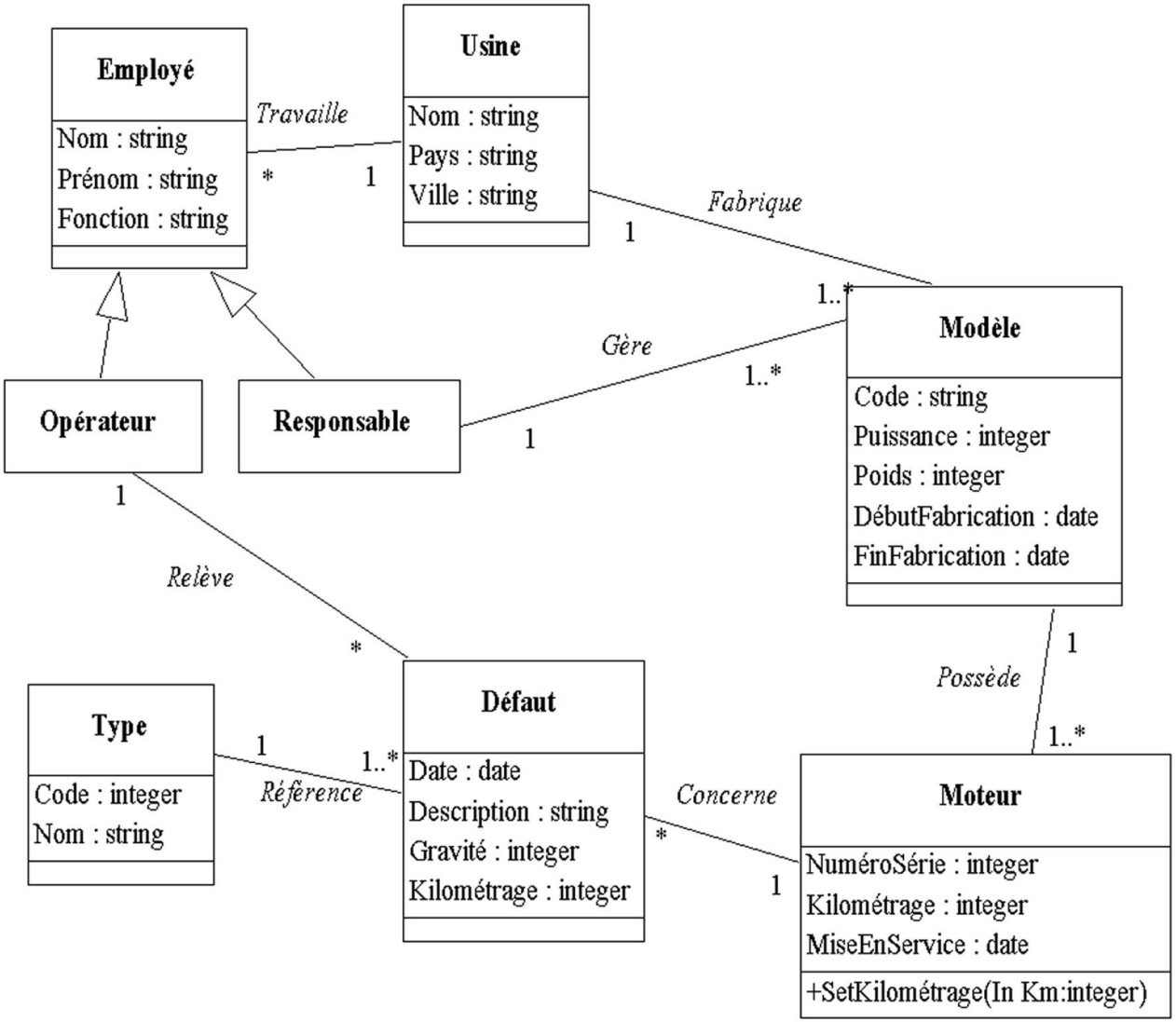

38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

```
fonction trier(p, n)
    Q := n/2 (division entière)
    P := n-Q
    si P >= 2
        q := trier(p, P)
        si Q >= 2 trier(q, Q)
    sinon
        q := p.suivant
    fin
    q := fusionner(p, P, q, Q)
    renvoyer q
fin

fonction fusionner(p, P, q, Q)
    répéter indéfiniment
        si valeur(p.suivant) > valeur(q.suivant)
            déplacer le maillon q.suivant après le maillon p
            si Q = 1 quitter la boucle
            Q := Q-1
        sinon
            si P = 1
                tant que Q >= 1
                    q := q.suivant
                    Q := Q-1
                fin
                quitter la boucle
            fin
            P := P-1
        fin
        p := p.suivant
    fin
    renvoyer q
fin
```

```java
Ville v = null;

try {
  v = new Ville("Re", 12000, "France");
}

//Gestion de l'exception sur le nombre d'habitants
catch (NombreHabitantException e) {
  e.printStackTrace();
}

//Gestion de l'exception sur le nom de la ville
catch(NomVilleException e2){
  System.out.println(e2.getMessage());
}
finally{
  if(v == null)
    v = new Ville();
}

System.out.println(v.toString());
```

- fonctions
- classes
- héritage
- macros
- et bien d'autres techniques


Public class
{
// comment
}
Java Compiler
0701AB0EA10
.....
Bytecode
Java Source code
http://starredreviews.com
Host System
Java Virtual Machine (JVM)


Employé
Nom : string
Prénom : string
Fonction : string
Travaille
*
1
Usine
Nom : string
Pays : string
Ville : string
Fabrique
1
1..*
Modèle
Code : string
Puissance : integer
Poids : integer
DébutFabrication : date
FinFabrication : date
Gère
1..*
1
Opérateur
Responsable
1
Relève
*
Possède
1
1..*
Type
Code : integer
Nom : string
1
1..*
Référence
Défaut
Date : date
Description : string
Gravité : integer
Kilométrage : integer
Concerne
*
1
Moteur
NuméroSérie : integer
Kilométrage : integer
MiseEnService : date
+SetKilométrage(In Km:integer)

- IHM (interfaces homme-machine)
- commentaires
- code propre
- etc.