

Etienne LANGLET.

9h00 - 12h30

14h00 - 17h00
30

Administratif:

⊗ Emargement: $\frac{1}{2}$ journée

⊗ Questionnaires: 1^{er} Jour: Mesure des connaissances initiales.

D. Jour: Mesure des acquis
Satisfaction

IMPULSE Sign In (Icône sur le bureau Windows).

Code d'accès: D1IZG2

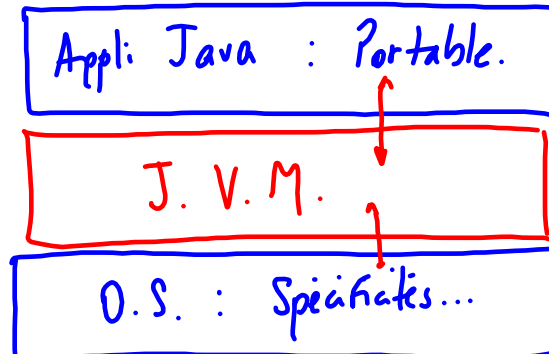
Partage de ressources:

<https://github.com/elanglet/TACFDEV1B1>

JVM : Java Virtual Machine

Exécution d'une application Java.

↳ Interprétation du Bytecode.



Obtenir une JVM:

2 packages installables.

- ⊗ JRE : Java Runtime Environment.
↳ Contient la JVM. + Bibliothèque de classes Java.
- ⊗ JDK : Java Development Kit
↳ JRE + Outils de dev.

Téléchargement : [Jdk.java.net](https://jdk.java.net)

Les plateformes Java.

Standard: Java Standard Edition (SE)

↳ La base de Java.

Enterprise: Java EE

↳ Extension de Java SE.

↳ Serveur.

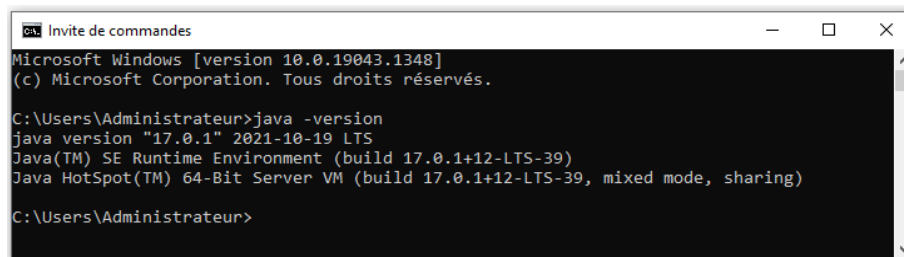
Spécificités des versions de Java.

2 types de versions:

- STS : Short-Term Support (6 mois)
- LTS : Long-Term Support (3 ~ 5 ans)

LTS: 8, 11, 17

Vérifier la version de Java installée:



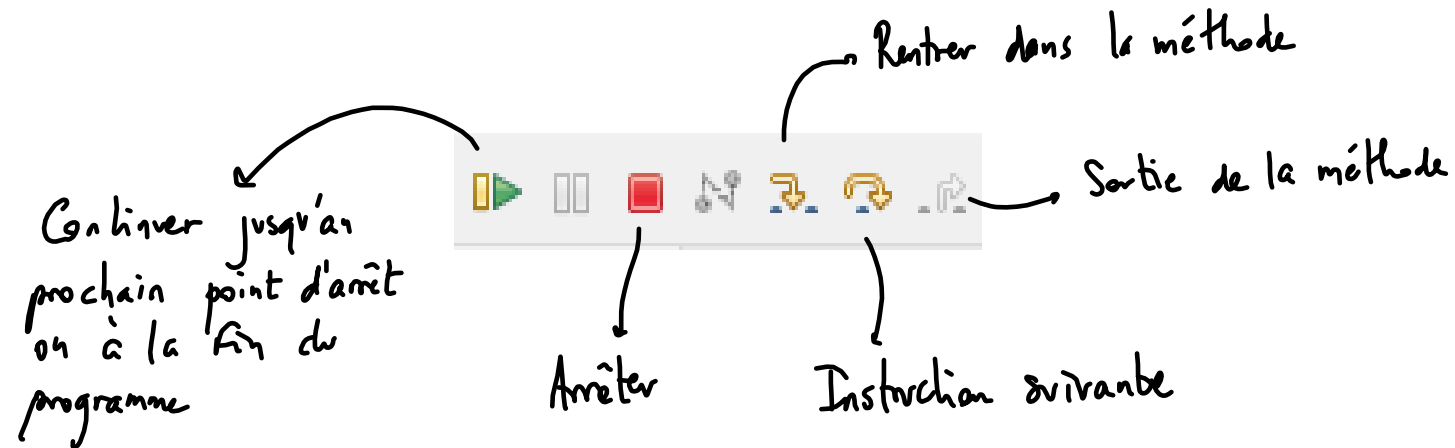
```
Invite de commandes
Microsoft Windows [version 10.0.19043.1348]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\Administrateur>java -version
java version "17.0.1" 2021-10-19 LTS
Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)

C:\Users\Administrateur>
```

Utiliser le débogueur.

- Positionner un ou plusieurs "point d'arrêt" (Breakpoint)
- Lancer l'exécution en mode "Debug"



Conventions de nommage → "Camel Case"

Classes : MaPremiereClasse / ArrayIndexOutOfBoundsException

Méthodes (fonctions) : calculerAge ()

Variables : ageDuCapitaine

Constantes : TAUX_TVA

Types de données.

→ Simples:

Entiers : byte, short, int, long

Réels : float, double.

Booléens : boolean

Caractères : char

→ Evolués / Complexes

. Tableaux

Chaines de caractères

Objets

Les tableaux.

tabEntiers1

0	100
1	200
2	300
3	400.

tabEntiers2

0	10
1	20
2	30
3	40

Tableaux à plusieurs dimensions.

```
int[] tab = new int[4][3];
```

	0	1	2
0			
1			
2		12	
3			

$tab[2][1] = 12;$

++ et --

int i = 0;

System.out.println(i++); // Affiche 0, i = 1

++ i // Affiche 2

ET / OU

ET: boolean a = false;
 boolean b = true;

if (a & b) { → f.
 ↘ f. ↘ t.
 ...
 }

if (a && b) { → f.
 ↘ f.
 ...
 }

OU:

if (b | a) { → t.
 ↘ t. ↘ f.
 ...
 }

if (b || a) { → t.
 ↘ t.
 ...
 }

- L'opérateur ternaire
 - $x ? y : z$
 - Si x vaut `true` alors l'expression vaut y sinon elle vaut z
- L'affectation
 - $x = y$
 - x est la copie de la valeur de y

```
if (a == 1) {  
    b = 2;  
}  
else {  
    b = 3;  
}
```

\Rightarrow

$b = (a == 1) ? 2 : 3;$

The diagram illustrates the logic of the ternary operator. A red arrow labeled "true" points from the condition $(a == 1)$ to the value 2. A green arrow labeled "false" points from the condition $(a == 1)$ to the value 3.



Cas d'usage : Switch / Case .

```
// Réponses possibles : "o", "oui", "n", "non", "yes", "y"
```

```
switch(reponse) {  
    case "o":  
    case "oui":  
    case "yes":  
    case "y":  
        Traitement quand réponse positive;  
        break;  
    case "n":  
    case "non":  
    case "no":  
        Traitement quand réponse négative;  
        break;  
}
```

Concepts Objets.

Objet:

Une entité identifiable du monde réel qui possède des caractéristiques et des comportements.

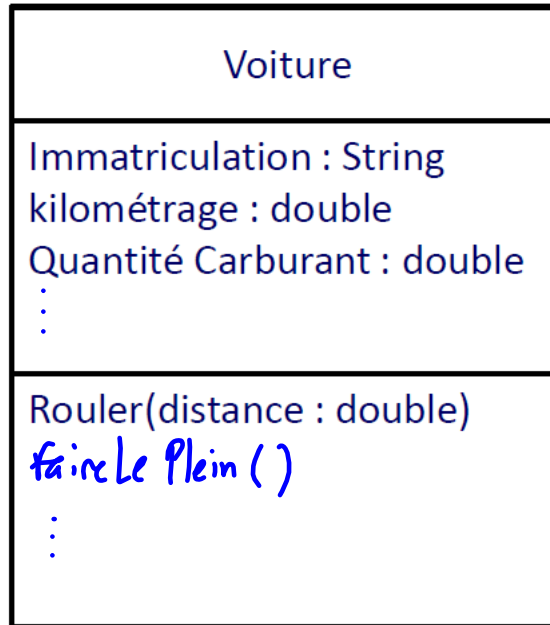
Classe:

Modèle pour la structure des objets.

La classe en UML:

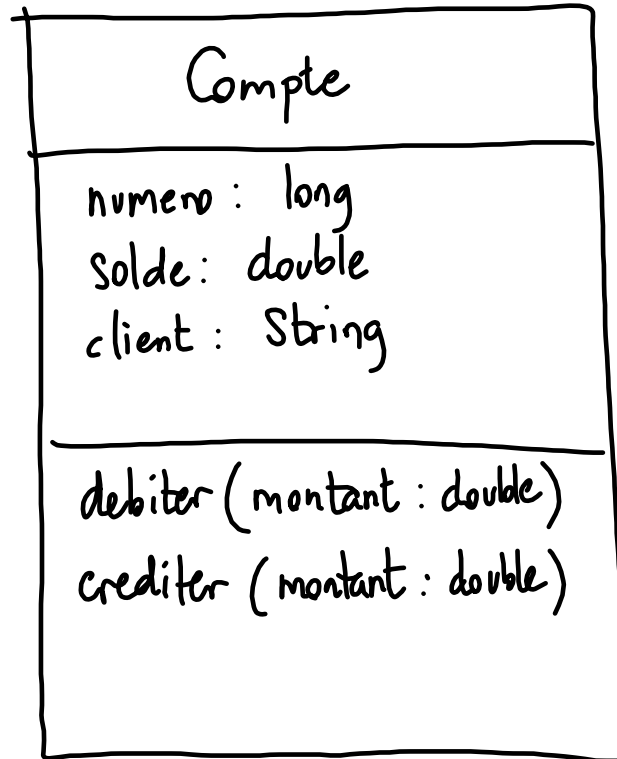
Nom
Commence par une maj.
Au Singulier.

Comportements.
Méthode.



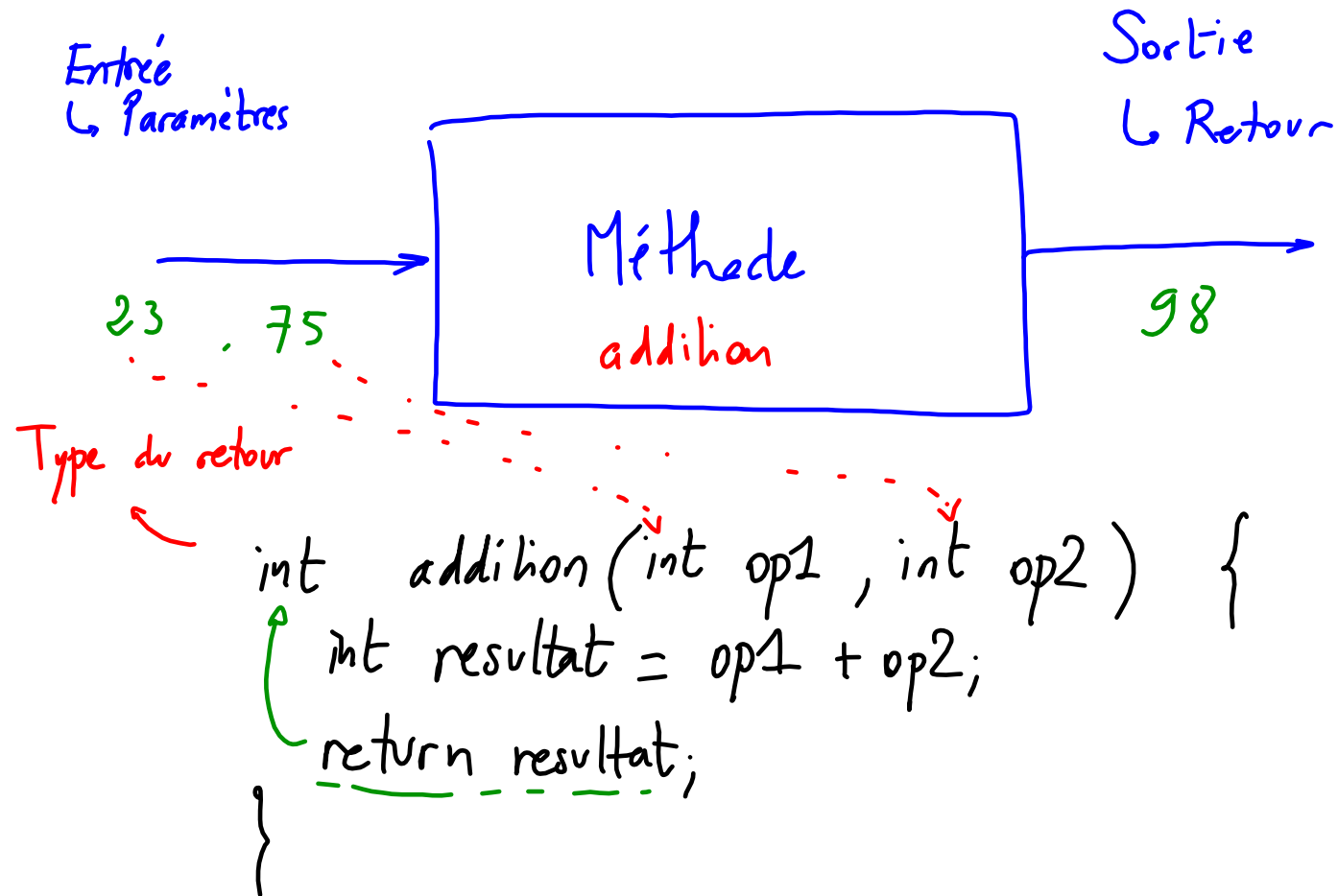
Caractéristiques
Attributs.

Mise en oeuvre : Une première classe



```
public class Compte {  
  
    long numero;  
    double solde;  
    String client;  
  
    void debiter(double montant) {  
  
    }  
  
    void crediter(double montant) {  
  
    }  
  
}
```

Les méthodes.



Ellipse : Exemple . *tableau !*

```
double moyenne(int... valeurs) {  
    int somme = 0;  
    for(int val : valeurs) {  
        somme = somme + val;  
    }  
    double resultat = somme / valeurs.length;  
    return resultat;  
}
```

*moyenne(10);
moyenne(11, 15);*

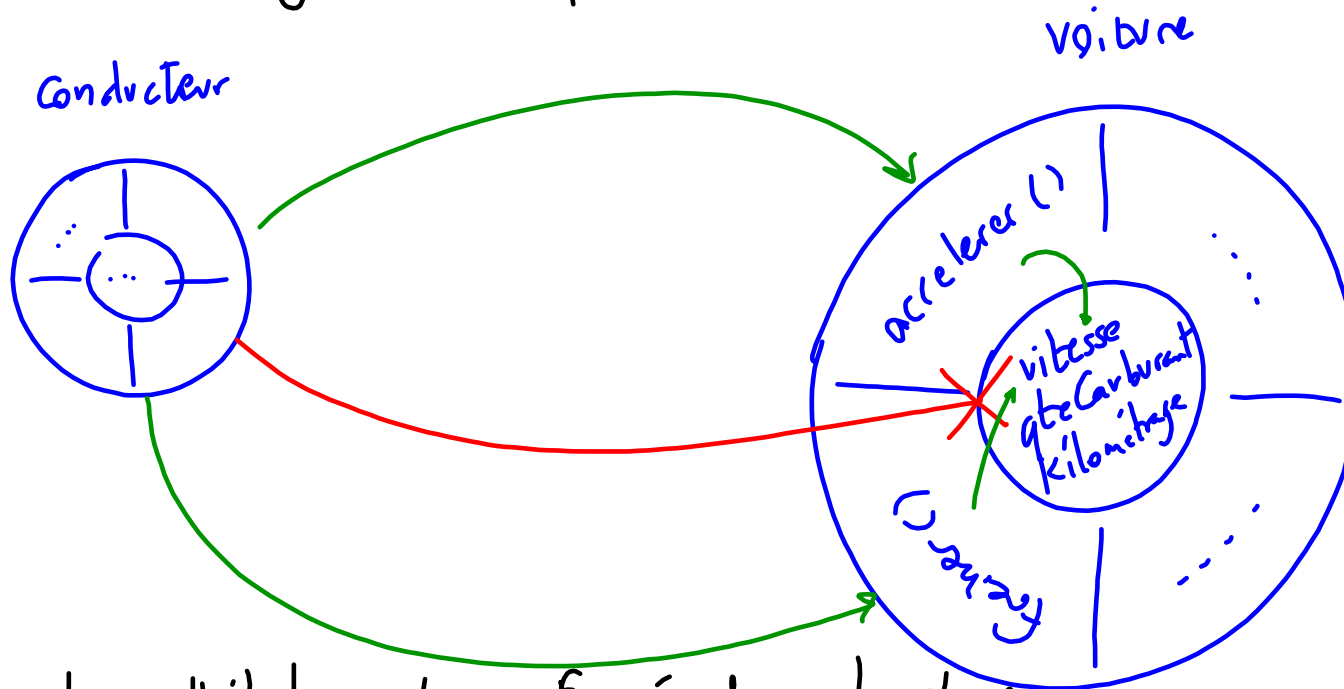
Le destructeur.

Méthode appelée automatiquement avant la destruction de l'objet.

```
void finalize () {  
}
```

Encapsulation

↳ Un objet est responsable de la cohérence de son état.



- Les attributs sont enfermés dans la classe.
- Les méthodes sont accessibles et font des contrôles.

Modificateurs de visibilité.

public : Visible en dehors de la classe

private : Invisible _____

<défaut> : Visible par les classes du même package (répertoire)

protected : Visible par les sous-classes. (Héritage)

Méthodes d'accès : Accesseurs.

private int vitesse;

En lecture (getter):

```
public int getVitesse() {  
    return vitesse;  
}
```

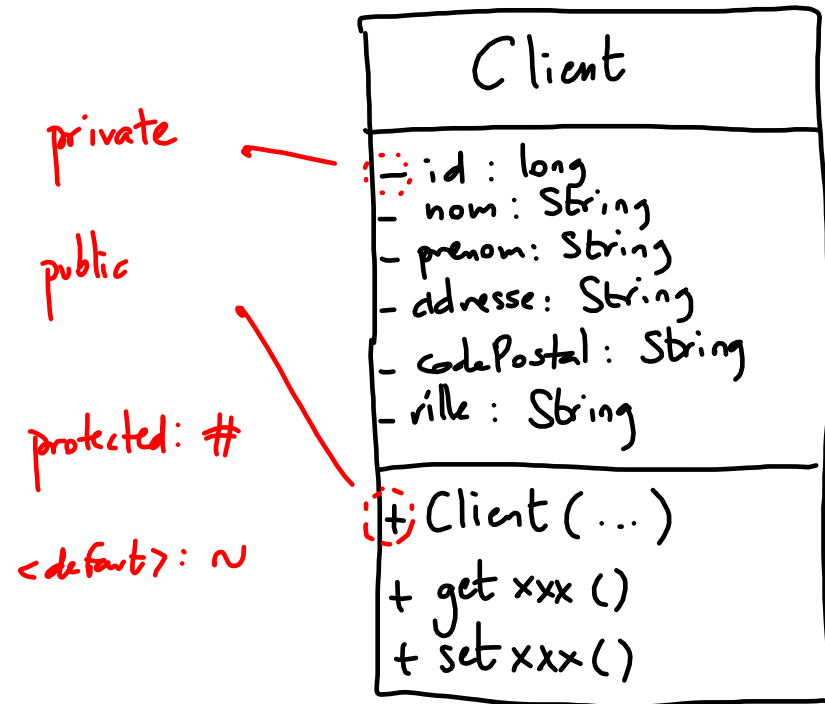
En écrire (setter):

```
public void setVitesse(int vitesse) {  
    if(.....) {  
        this.vitesse = vitesse;  
    }  
}
```

Génération avec Eclipse:

Menu "Source" → "Generate getters and setters".

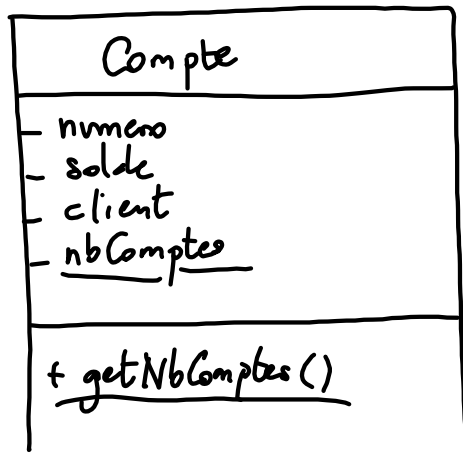
Mise en œuvre: Nouvelle classe & génération avec Eclipse



Membres de classes

Attributs & méthodes | de classes
| "static"

↳ Informations propre à la classe et pas à chaque objet !

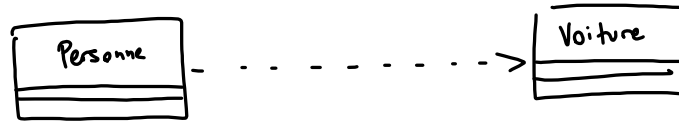


⊗ Constantes de classes.

```
private static final int NB_ROWS = 4;
```

Collaborations

Dépendance : Un objet peut avoir besoin d'un autre pour réaliser ses opérations (dépo. Mo)



Composition : Les cycles de vie des objets sont liés !



Agrégation : Conteneur / Contenu, mais les cycles de vie sont indépendants.



Association : Relation permanente entre les objets, mais le cycle de vie des objets est indépendant.

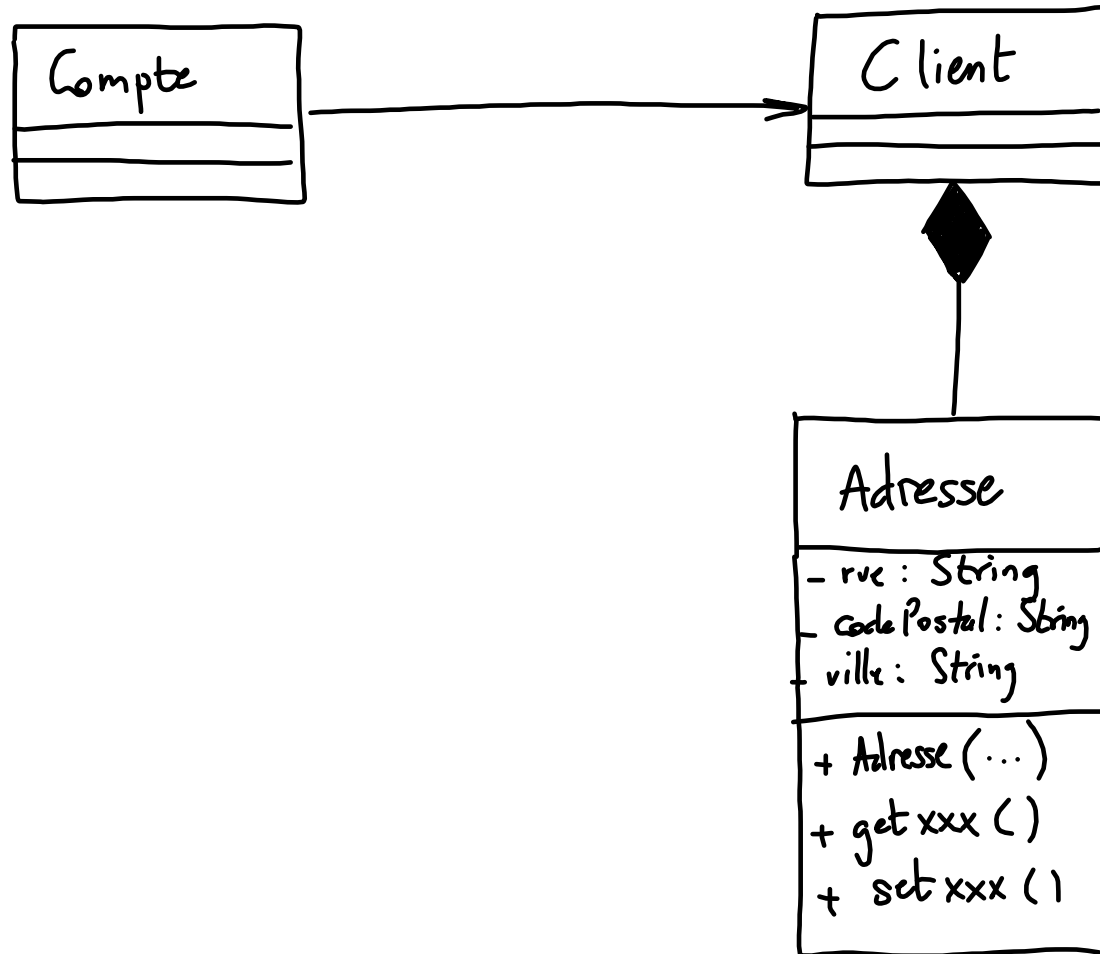


```

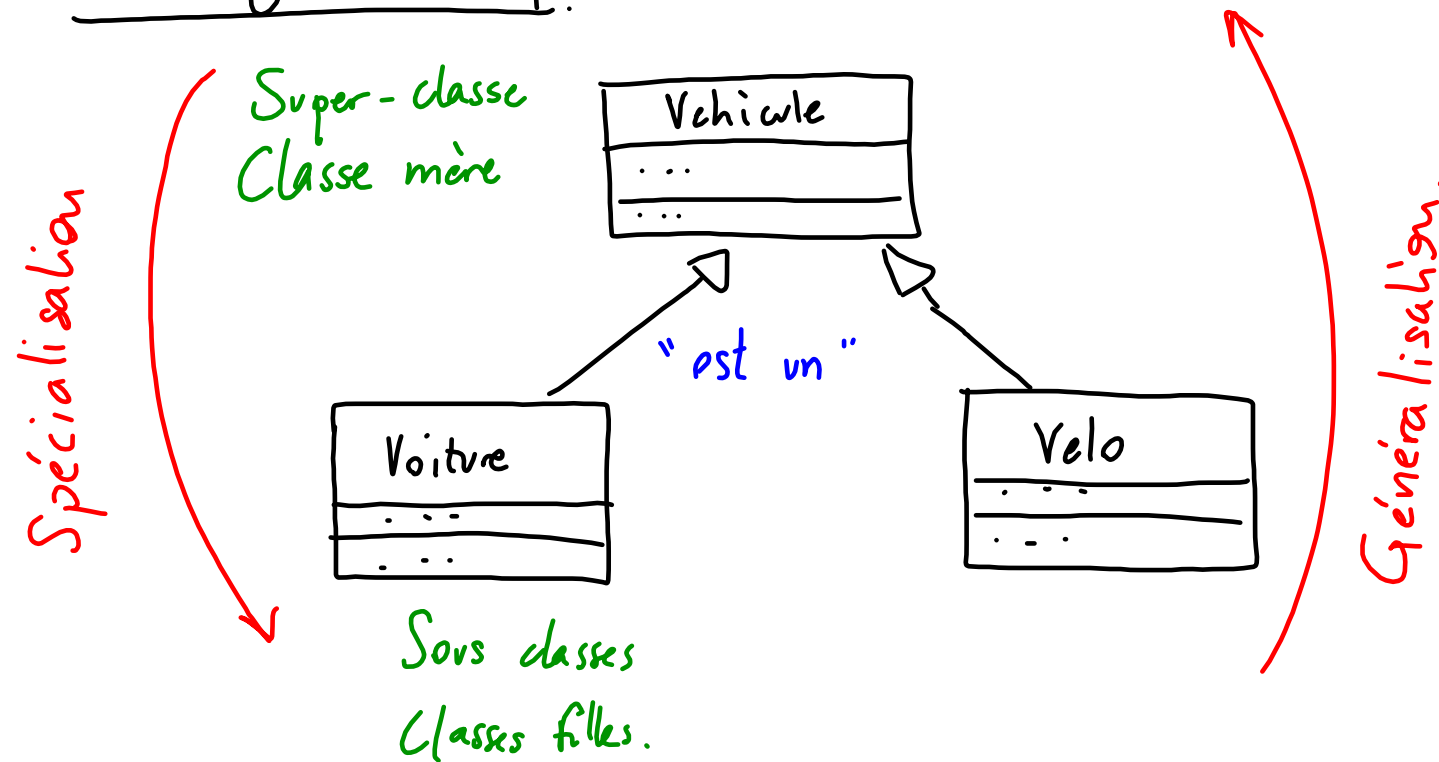
public class Voiture {
    private CarteGrise certificat;

    public Voiture(..., CarteGrise certificat) {
        this.certificat = certificat;
    }
}
  
```

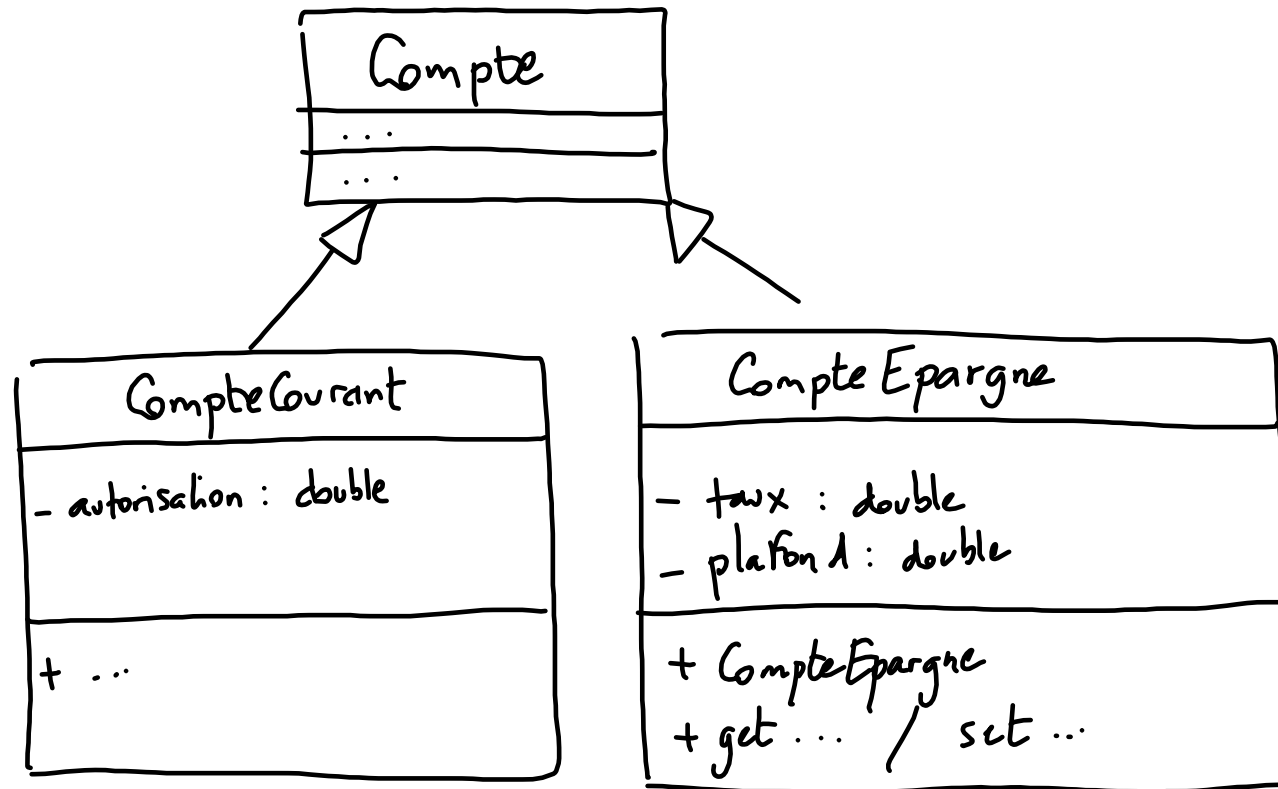
Mise en œuvre



L'héritage : Analyse



Mise en oeuvre : L'héritage



Construction d'objets dans l'héritage

En Java, on doit prendre en charge la construction complète des objets.

→ Par défaut, le constructeur de la sous-classe tente d'appeler automatiquement le constructeur par défaut de la super classe.

→ Sinon, on doit appeler explicitement un super-constructeur avec **Super ()**

↳ doit être la 1^{re} instruction.

Héritage et typage

Compte c1 = new CompteEpargne (...);

est un ?

Comment connaître la valeur du taux ?

double t = c1.getTaux();

↳ Erreur ! Pas de getTaux() dans Compte !

Solution ! Reconsidérer l'objet en tant que CompteEpargne

↳ Transstypage

CompteEpargne ce = (CompteEpargne) c1;

Est ce que l'objet
référéncé par c1
est bien un
CompteEpargne ???

double t = ce.getTaux();

Tester le type d'un objet.

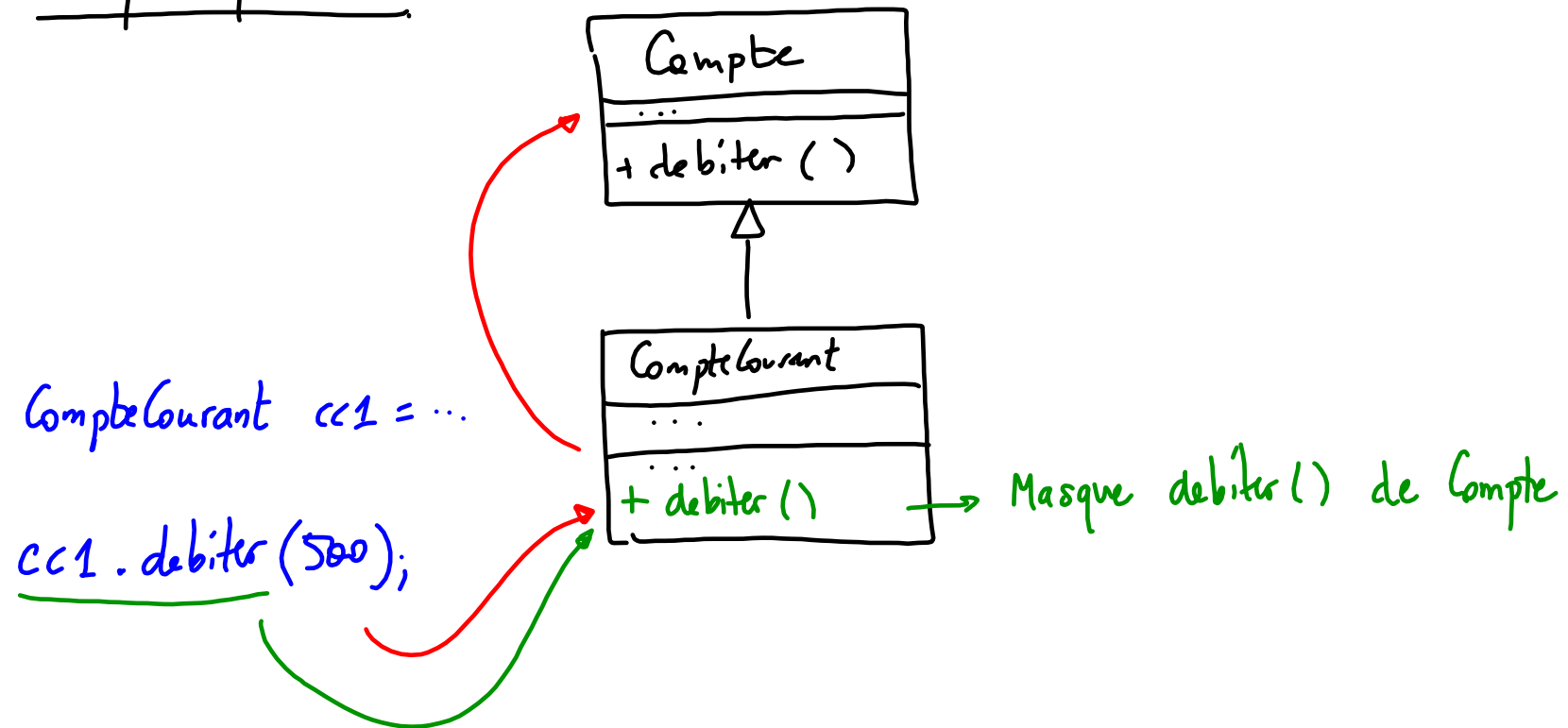
Pour vérifier que le transypage est possible !.

variable instanceof Classe

Ex. :
CompteEpargne ce1 = new CompteEpargne (...);

- ① if (ce1 instanceof CompteEpargne) ... **T**
- ② if (ce1 instanceof Compte) ... **T**
- ③ if (ce1 instanceof CompteCourant) ... **F**
- ④ if (ce1 instanceof Client) ... **F**
- ⑤ if (ce1 instanceof Object) ... **T**

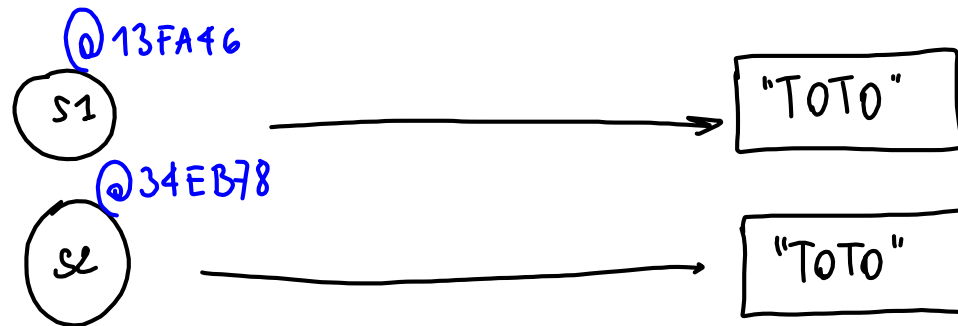
Polymorphisme



Comparaison des objets.

Opérateur de comparaison: `==` `if (a == 1) ...`

↳ Uniquement pour les types simples !
(byte, short, int, long, float, double, boolean, char)
↳ Valeur des variables.



`==` entre des objets compare les adresses mémoire !

```

String s1 = "TOTO";
String s2 = "TOT";

s2 = s2 + "O";

if(s1 == s2) { // FAUX !
    System.out.println("Les chaînes sont identiques");
}
else {
    System.out.println("Les chaînes sont différentes"); ←
}
  
```

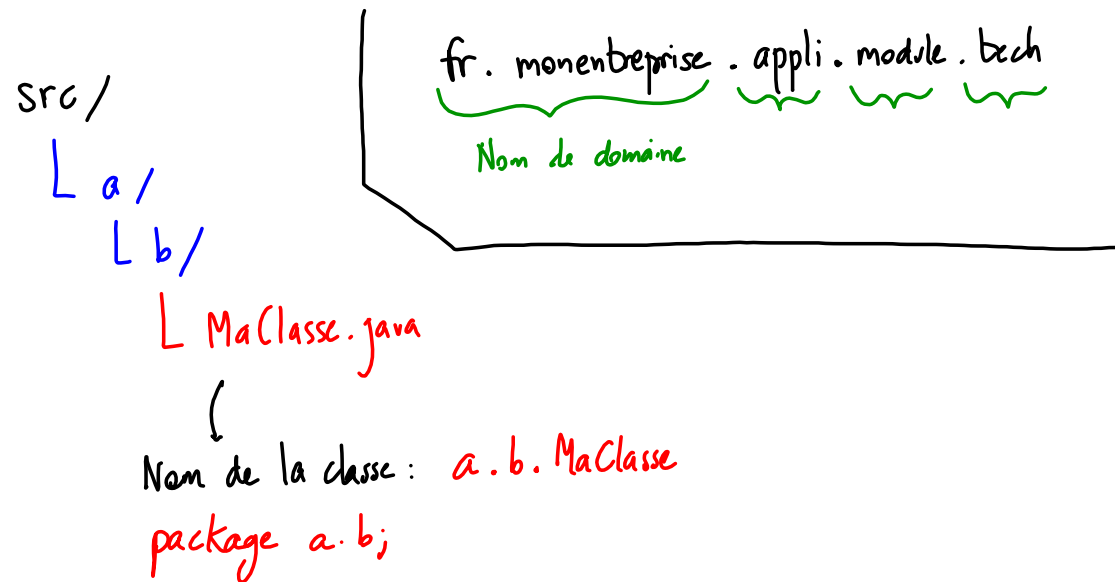
Comparer des objets avec equals()

$\text{objet1.equals}(\text{objet2}) \equiv \text{objet2.equals}(\text{objet1})$

```
String s1 = "TOTO";  
String s2 = "TOT";  
  
s2 = s2 + "0";  
  
if(s1.equals(s2)) { → Good !  
    System.out.println("Les chaines sont identiques");  
}  
else {  
    System.out.println("Les chaines sont différentes");  
}
```

Pour comparer 2 objets, il faut redéfinir la méthode `equals()` dans la classe de ces objets

Les packages : Organisation et hommage



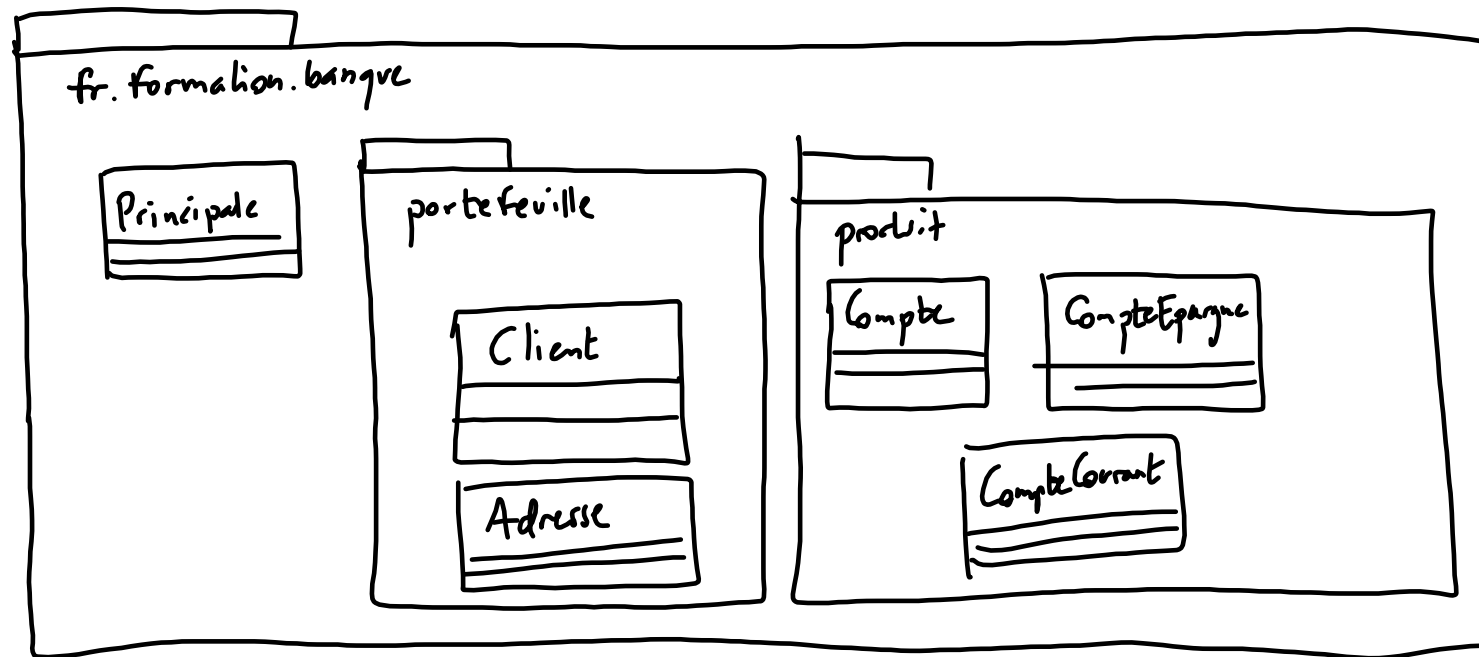
ATTENTION à l'ordre des déclarations !

- ① package ...
- ② import ...
import ...
- ③ class ... {
 }
}

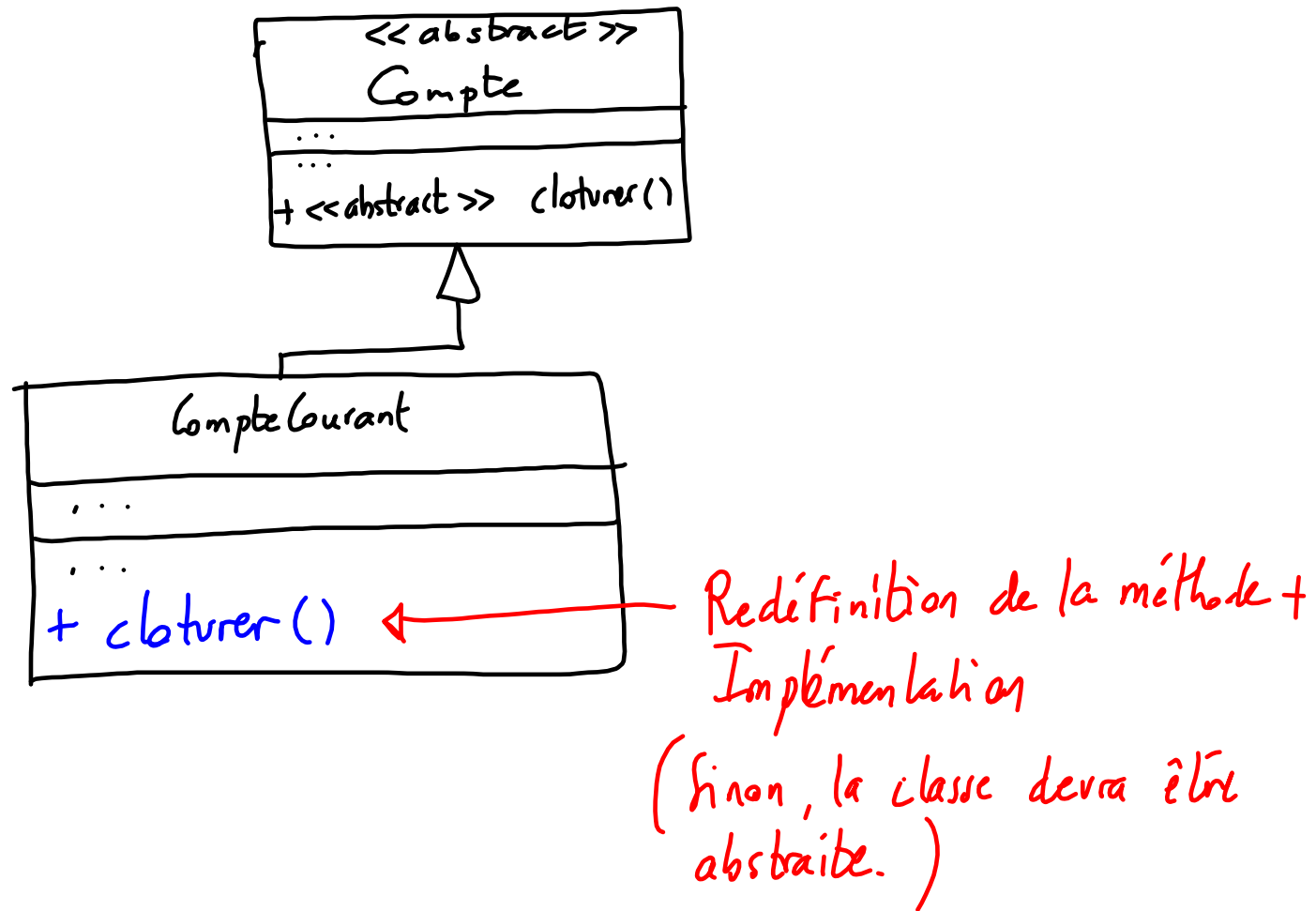
Mise en œuvre : les packages.

- o Convention : Définir un package racine.

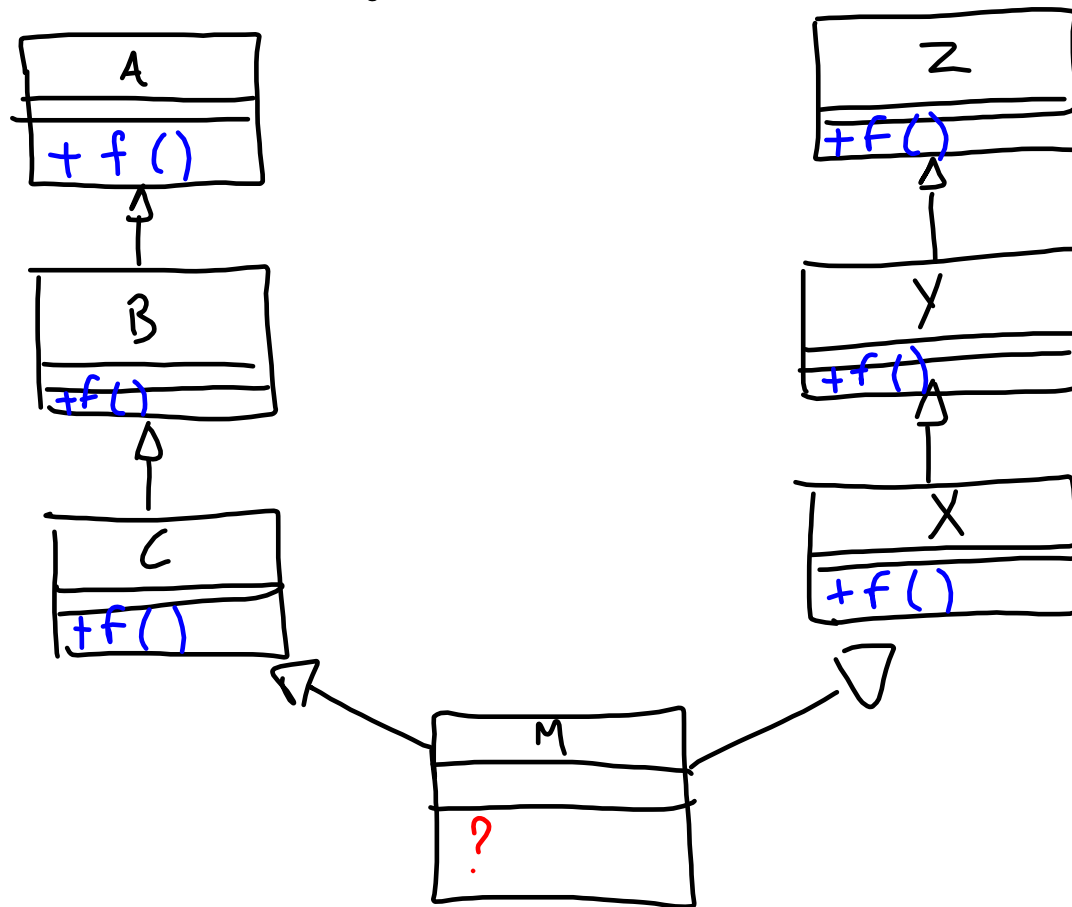
- fr. formation, banque
- produit
- portefeuille



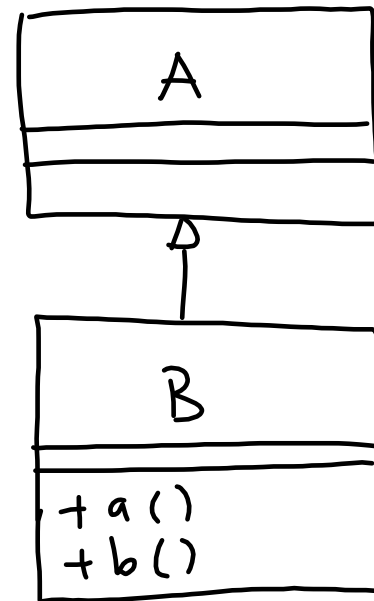
Classes et méthodes abstraites



Problème de l'héritage multiple !

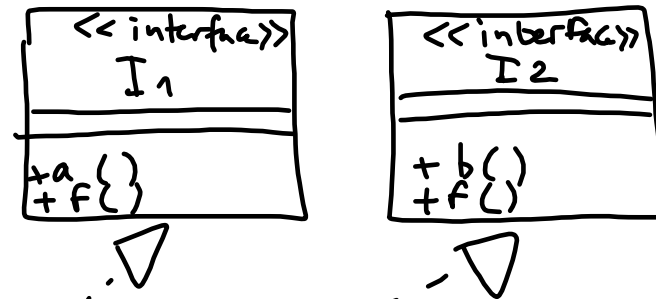


Les interfaces



```

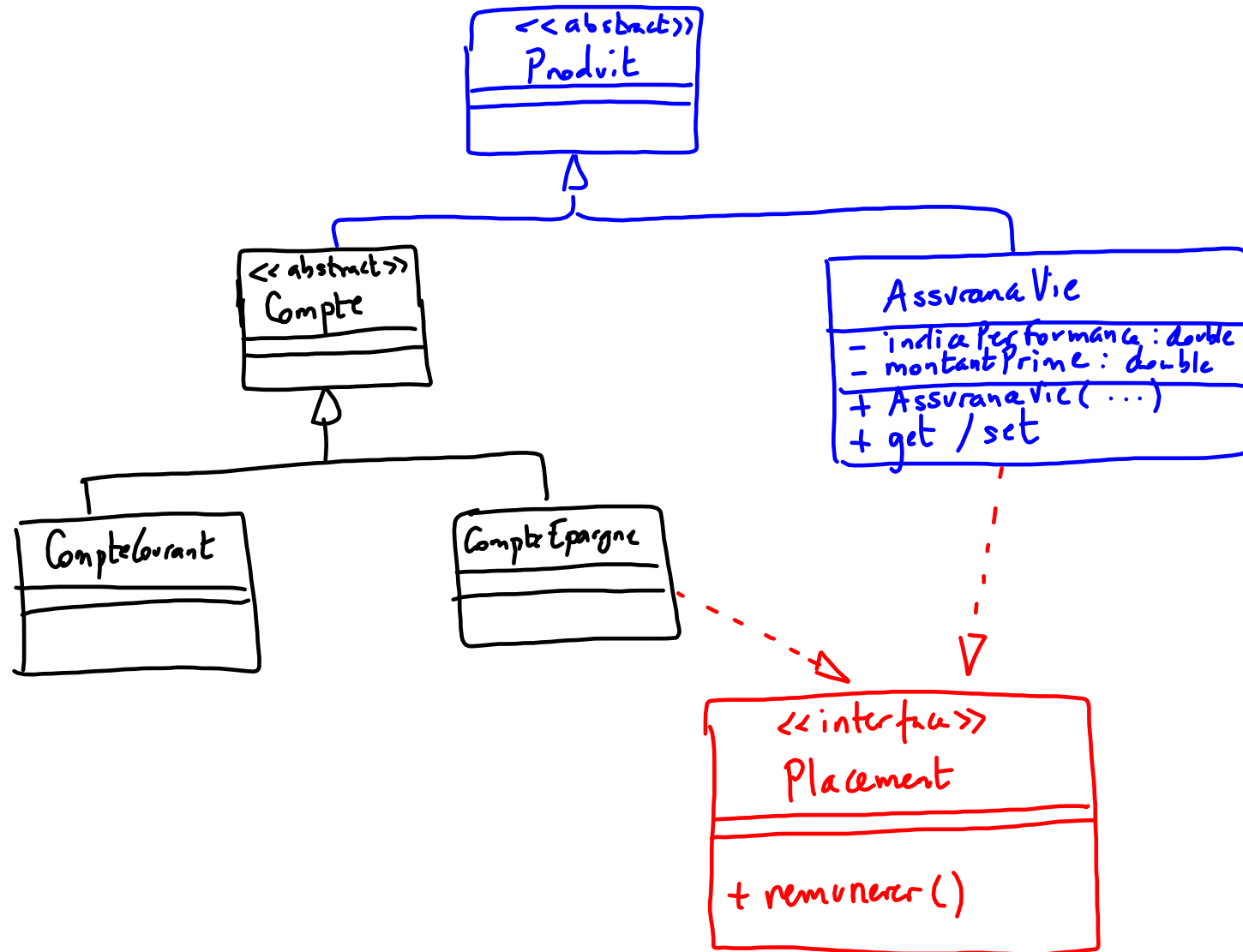
public class B extends A implements I1, I2 {
    public void a() { ... }
    public void b() { ... }
    public void f() { ... }
}
  
```



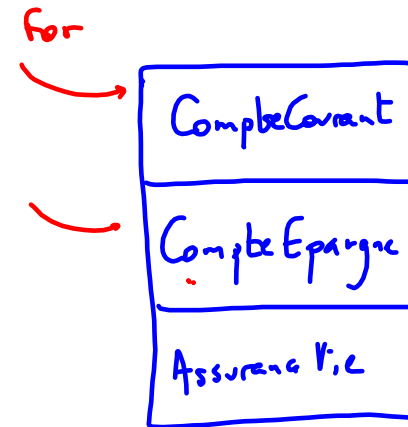
```
B b = new B();
```

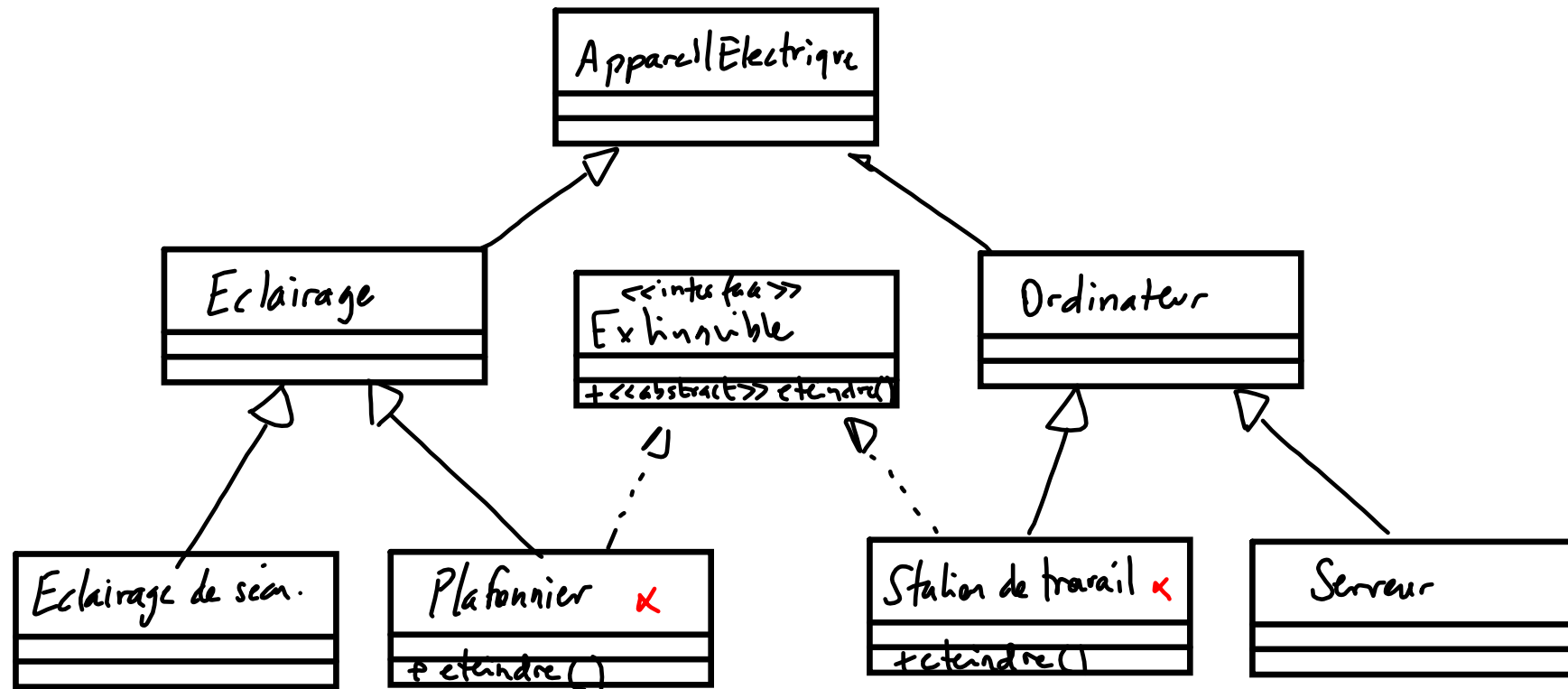
b instanceof B	→	T
b instanceof Object	→	T
b instanceof A	→	T
b instanceof String	→	F
b instanceof I1	→	T
b instanceof I2	→	T

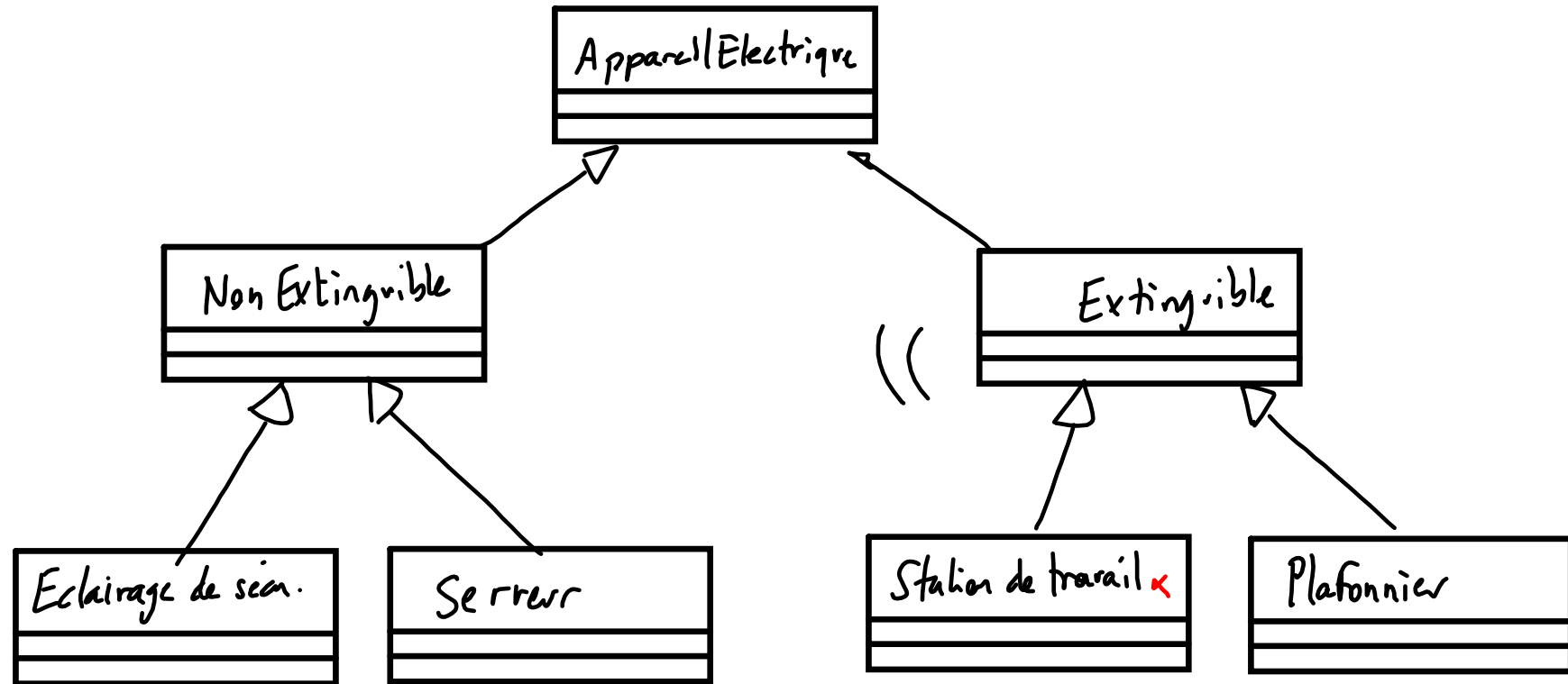
Mise en oeuvre: Les interfaces.



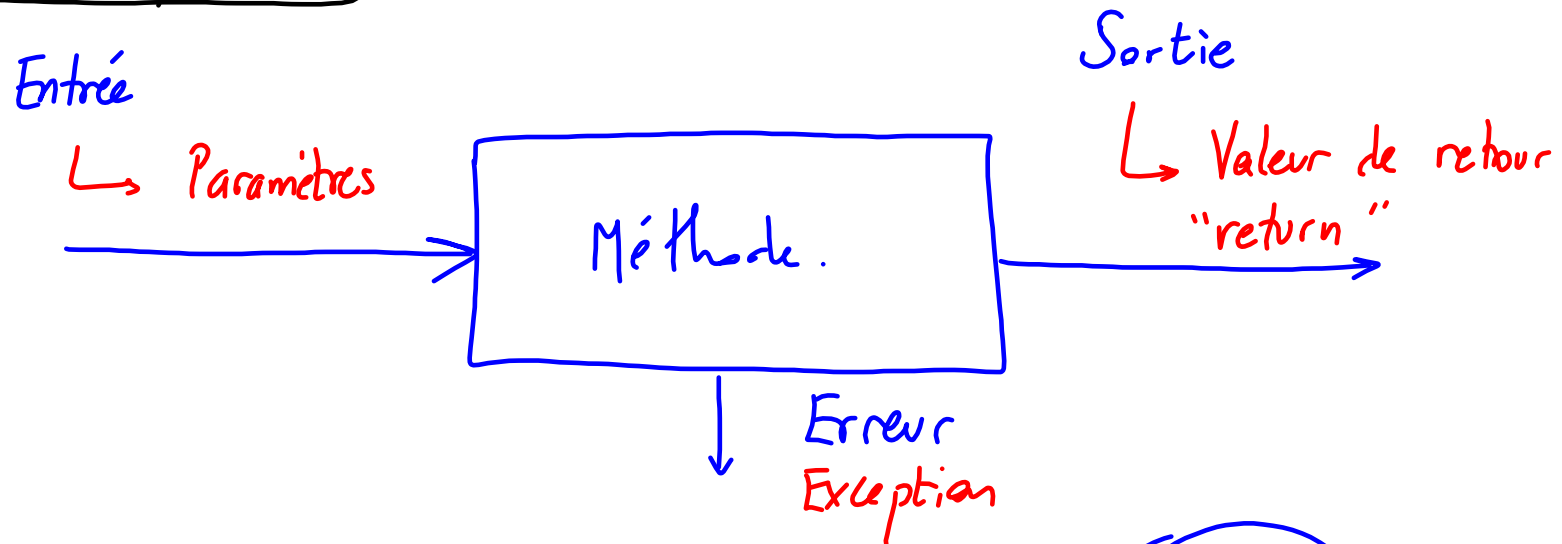
```
Produit[] produitsBancaires = { cc1, ce1, ass1 };  
  
// Pour chaque 'produit' dans 'produitsBancaires'...  
for(Produit produit : produitsBancaires) {  
    // Traitement de chaque produit ...  
  
    // On veut rémunérer tous les placements !  
    if(produit instanceof Placement) {  
        // Ce produit est un Placement  
        Placement placement = (Placement) produit;  
        placement.remunerer();  
    }  
}
```



Autre exemple sur les interfaces



Les exceptions



```
double  diviser(double n, double d) throws Exception {  
    (S) return ...  
}
```

The code snippet is annotated with blue circles and arrows:

- A blue circle containing the letter **E** is positioned above the parameter `n`, with an arrow pointing to it.
- A blue circle containing the letter **S** is positioned to the left of the `return` statement, with an arrow pointing to it.
- A blue circle containing the word **Exception** is positioned to the right of the `throws` keyword, with an arrow pointing to it.

throw / throws.

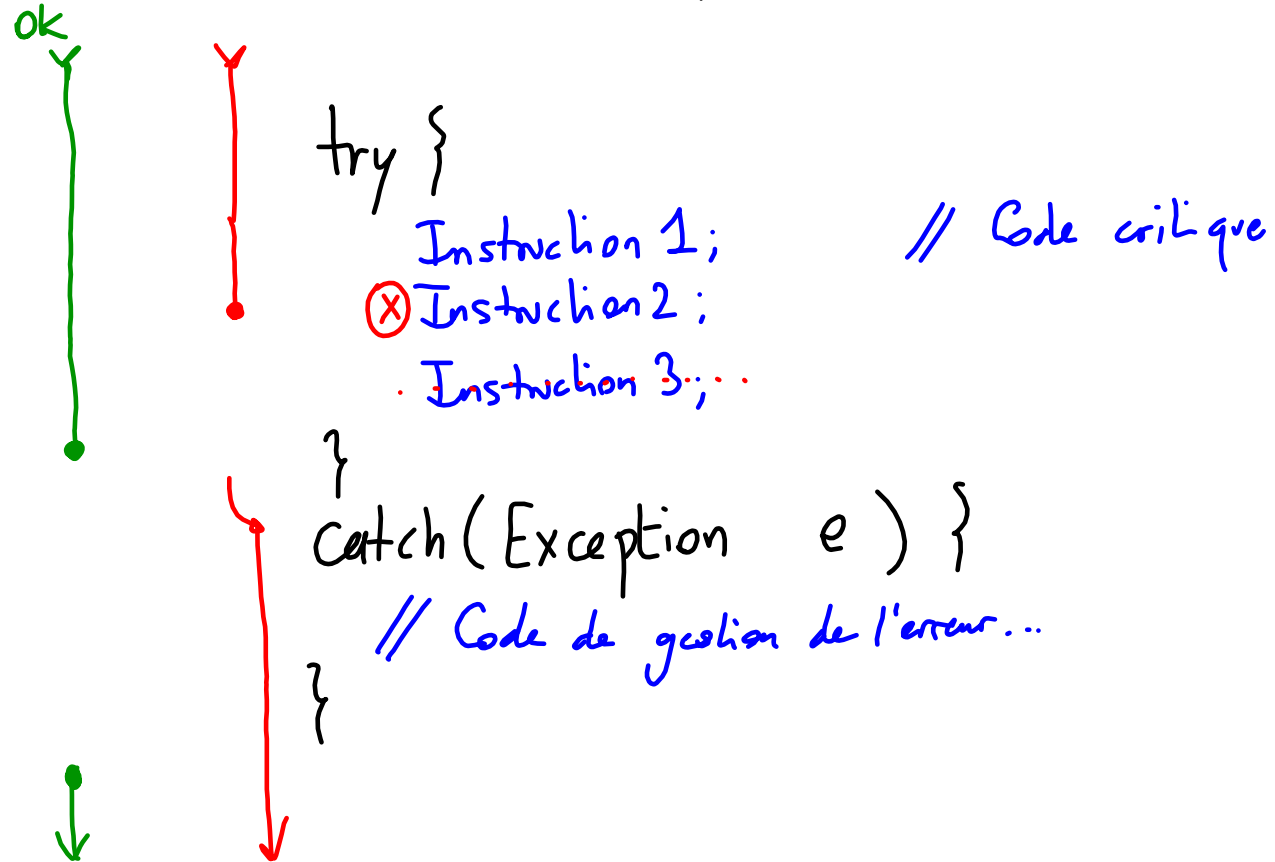
Sortie Std Entrée Sortie d'erreur

```
public double diviser(double numérateur, double dénominateur) throws Exception {  
    if(dénominateur != 0) {  
        return numérateur/dénominateur;  
    }  
    else {  
        ① Exception e = new Exception("Division par 0 impossible !");  
        ② throw e;  
    }  
}
```

- ① Création de l'objet d'erreur/exception
- ② Déclenchement de l'exception.

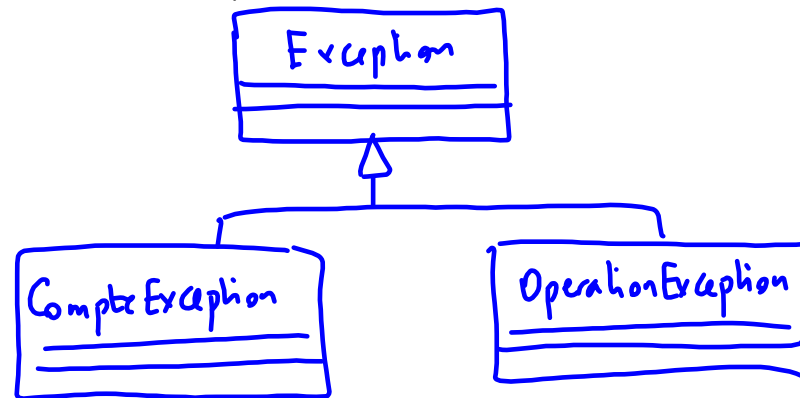
throw : Le déclenchement de l'erreur
throws : Indique une sortie d'erreur.

Gestionnaire d'exception: try ... catch ...



Mise en œuvre : Les exceptions.

- ① Créer des exceptions personnalisées.



- ② Générer des exceptions dans les méthodes où des tests sont faits et ne sont pas vérifiés.
- ③ Gérer les exceptions → try... catch...

Vector & ArrayList

↳ Tableau à taille dynamique.

↳ Varie au fur et à mesure des ajouts.

Vector est synchronisée

✓ ArrayList non

Hashtable & HashMap.

↳ Tables de hachage / Dictionnaires

↳ Ensembles de paires clés / valeurs.

Clé	Valeur
"un"	1
"deux"	2
"trois"	3
"quatre"	4
...	5

Hashtable est synchronisée

✓ HashMap non

Mise en œuvre: HashMap.

```
CompteCourant comptecourant1 = new CompteCourant(562369, 100.00, client1, 500.00);
```

On veut stocker des "CompteCourant" identifiés dans la HashMap par leur numéro ...

HashMap < k, v >

Doivent être des objets!

~~long~~ → N'est pas un objet : (

CompteCourant

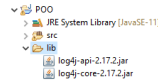
Solution: Pour chaque type simple de Java, on a une classe équivalente! → (Classes Enveloppes (Wrappers!))

byte	java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
boolean	java.lang.Boolean
char	java.lang.Character

Mise en oeuvre : Log4J.

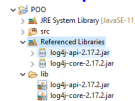
① Installer Log4J dans le projet.

a. Fichiers essentiels : log4j-api-x.x.x.jar
log4j-core-x.x.x.jar



b. Ajouter les fichiers au "Build Path"

. Sélection des fichiers
. Clique-droit → Build Path → Add to Build Path



② Fournir la configuration de Log4J → Fichier log4j2.xml dans src/

③ Générer des traces dans le code

① Créer un attribut Logger

```
static Logger logger = LogManager.getLogger(MaClasse.class);
```

② Générer des traces

```
logger.debug("Message");  
logger.info("Message");  
logger.warn("Message");  
logger.error("Message");  
logger.fatal("Message");
```

→ Générer des messages :

debug : A chaque fois qu'un objet est créé
↳ Dans tous les constructeurs.

info : Pour un débit ou un crédit sur un compte.

error : En cas d'exception.

Tests unitaires : Bonnes pratiques.

Symétrie!

CODE

```
public class A {  
    public void m1 () { .. }  
    public void m2 () { .. }  
}
```

TEST

```
public class ATest {  
    public void testM1 () { ... }  
    public void testM2 () { ... }  
}
```

Pour aller plus loin :

T443-015 - Programmation Java (Java SE) – Perfectionnement

<https://www.eni-service.fr/index.php/fomation/programmation-java-java-se-perfectionnement/>

