

PROGRAMMATION ORIENTÉE-OBJET

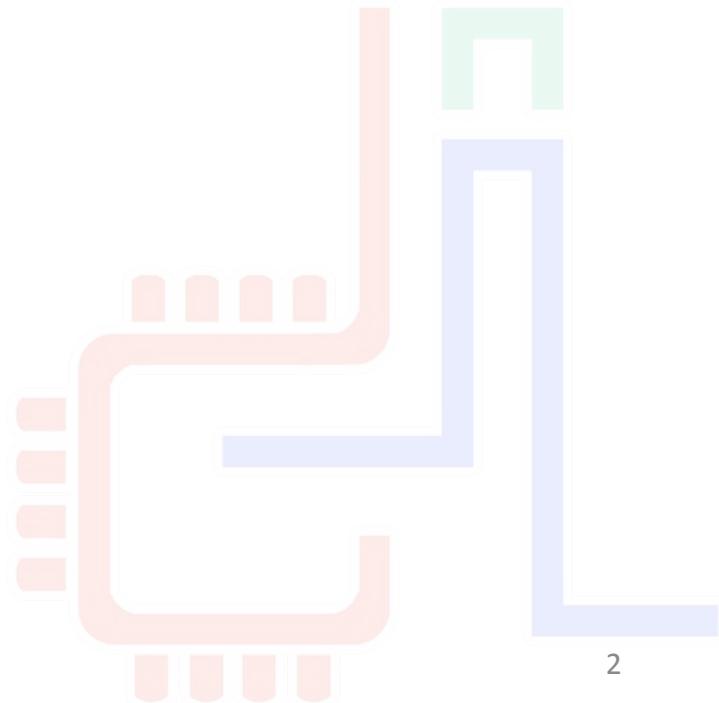
TAREK BEN MENA

2^{ème} Licence Fondamentale en
Sciences de l'Informatique



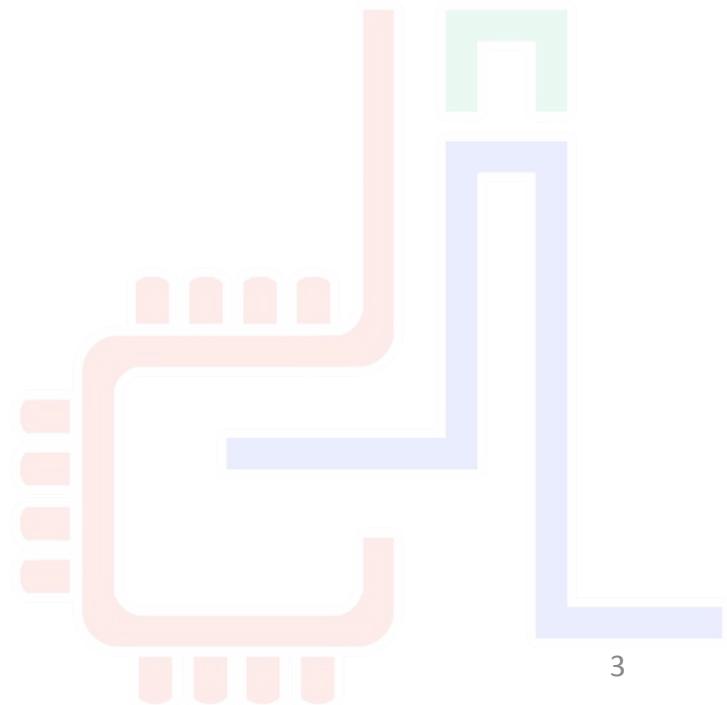
PLAN

- Introduction
 - Du paradigme procédurale à l'objet
 - Ça sera avec Java
- Concepts Orientée-Objet
 - Classes et Objet
 - Encapsulation
 - Héritage
 - Polymorphisme
 - Classe Abstraite
 - Interface
 - Exception

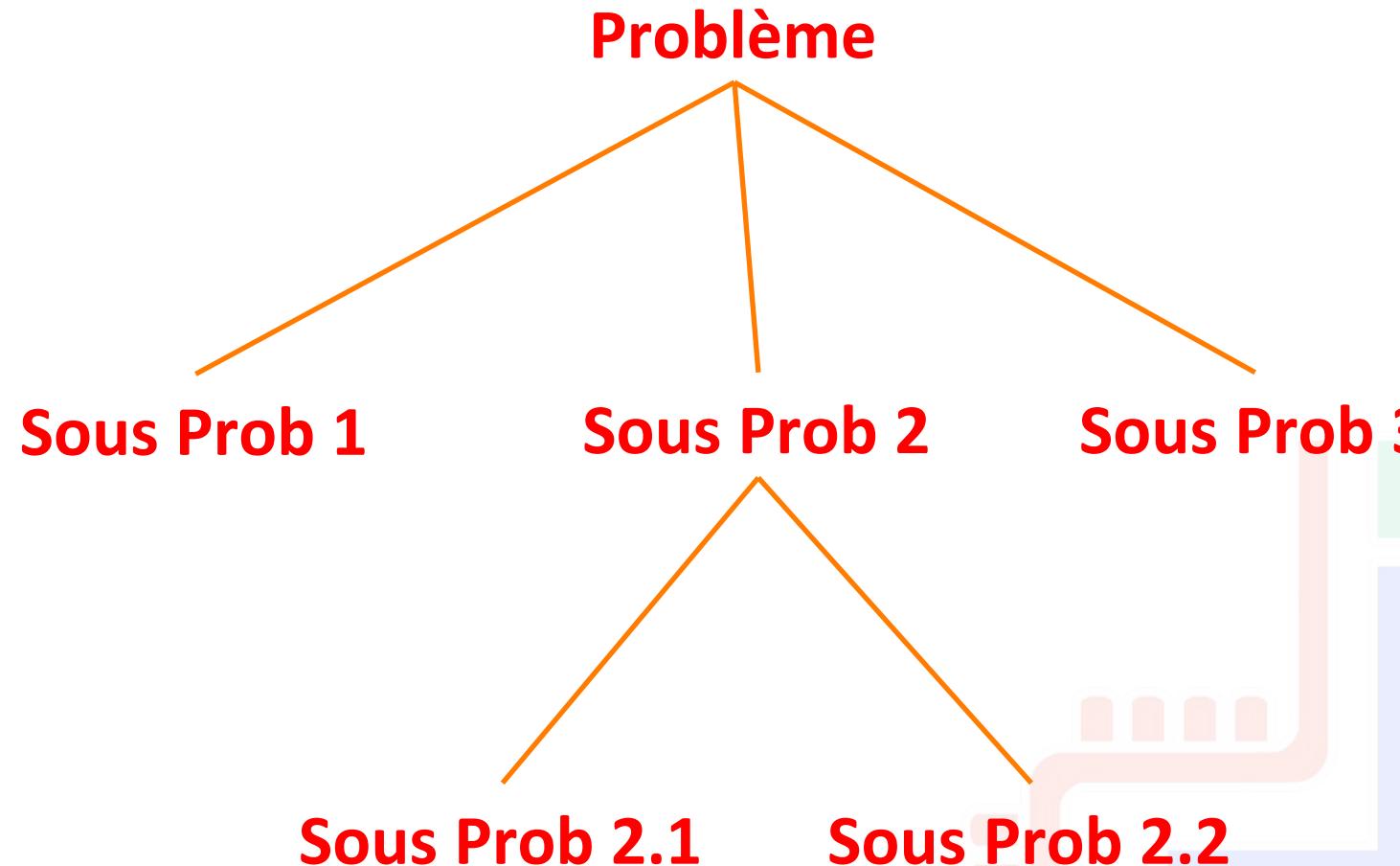


RETOUR SUR VOTRE EXPÉRIENCE DE PROGRAMMEUR EN SII

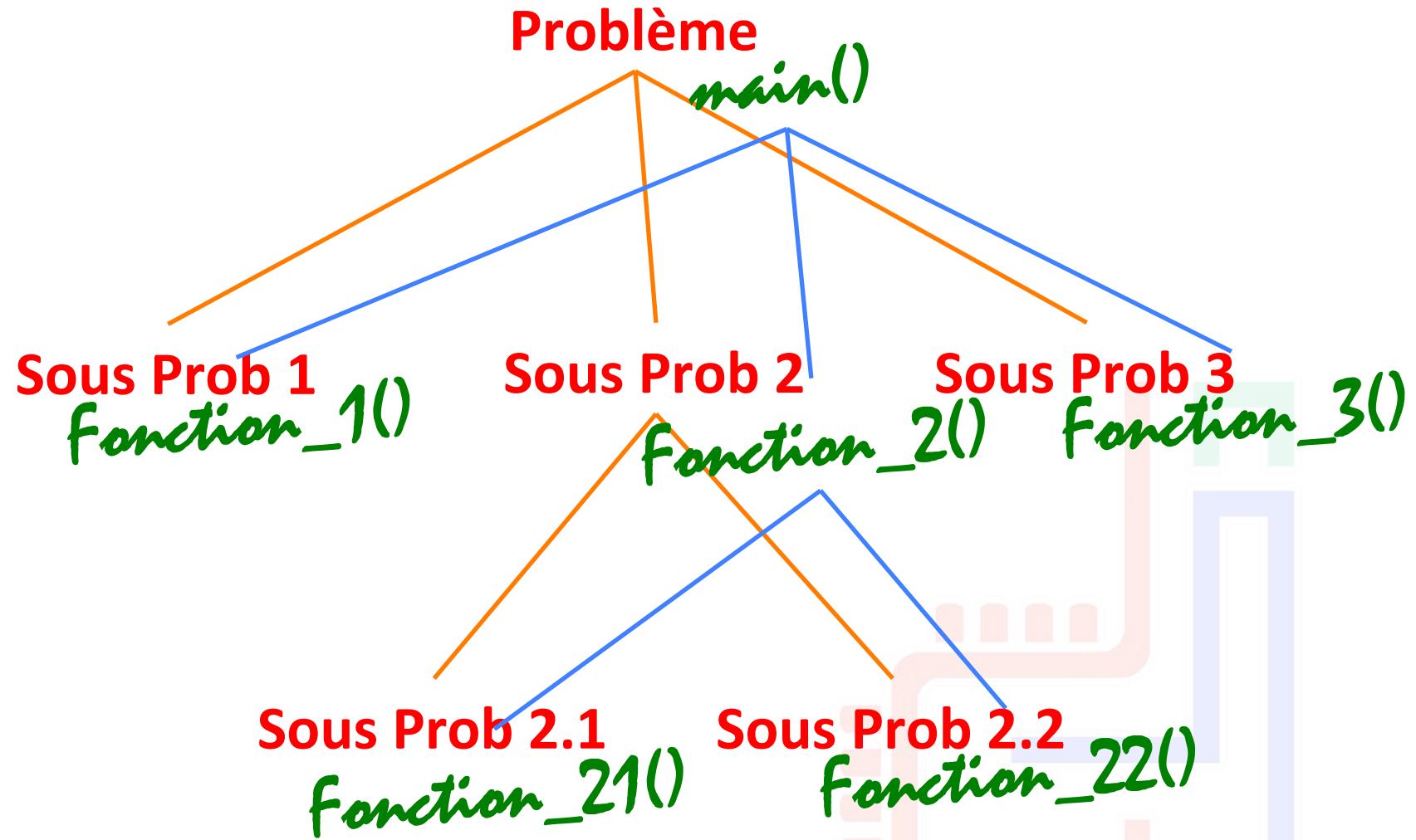
- Langage ?
- Démarche ?
- Conception ?
- Qualité ?
- Ce qui s'est bien passé ?
- Ce qui s'est mal passé ?



DÉMARCHE ?

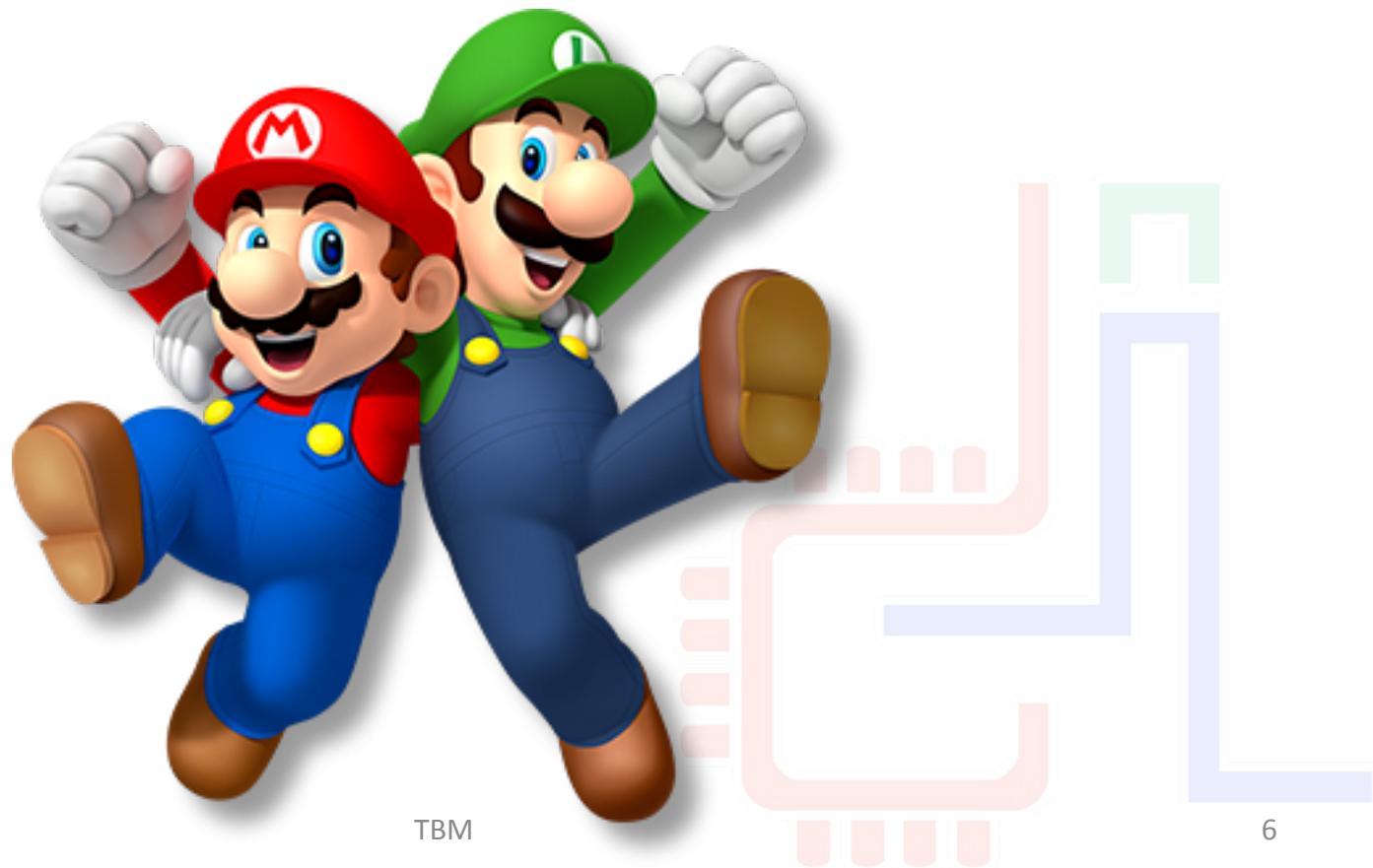


DÉMARCHE ?

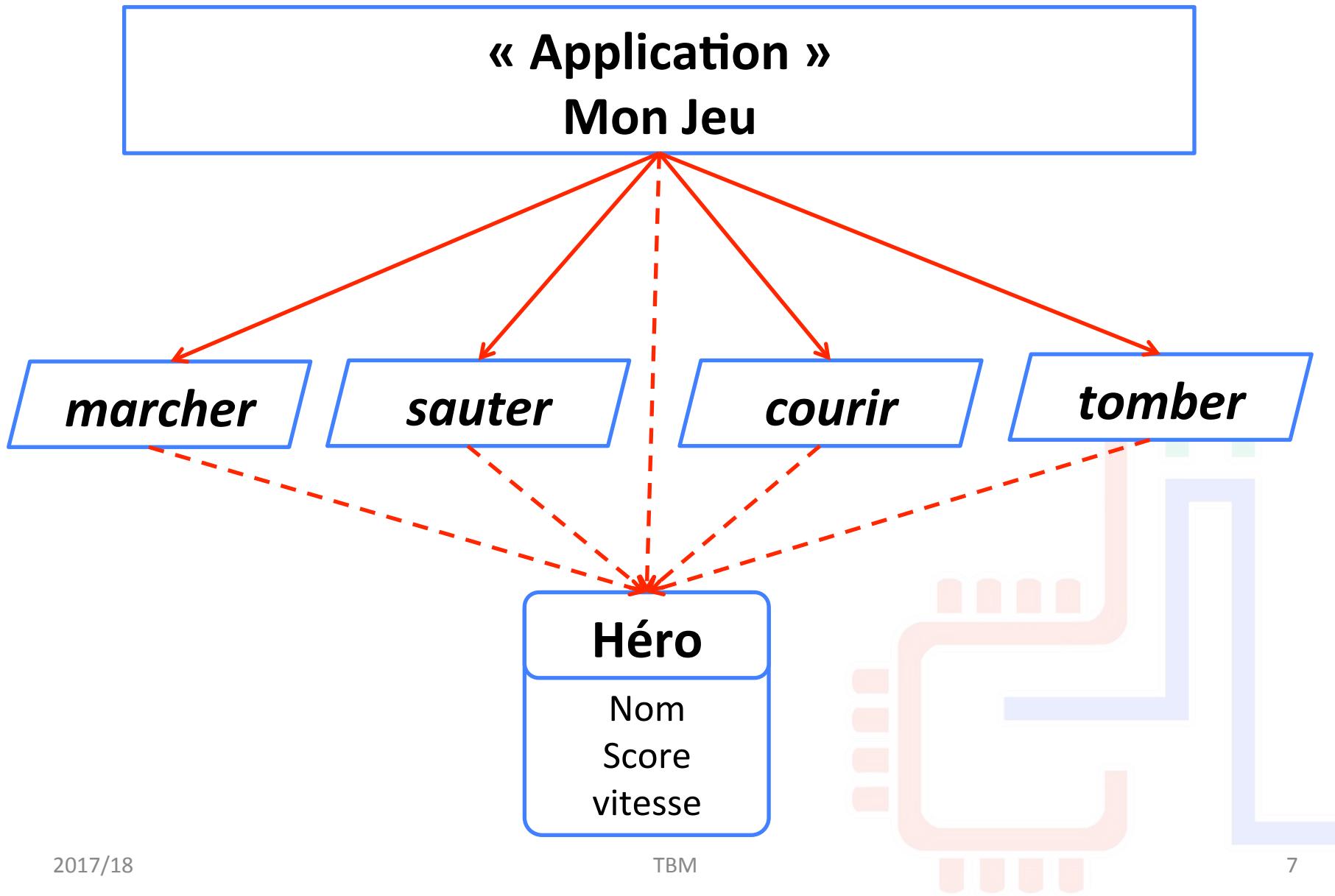


EXEMPLE POUR COMMENCER ☺

- Développer un Jeu où deux héros « Mario » et « Luigi » caractérisé par un score et une vitesse peut sauter, marcher, courir et tomber.



CONCEPTION



CODAGE

```
monJeu.c *
```

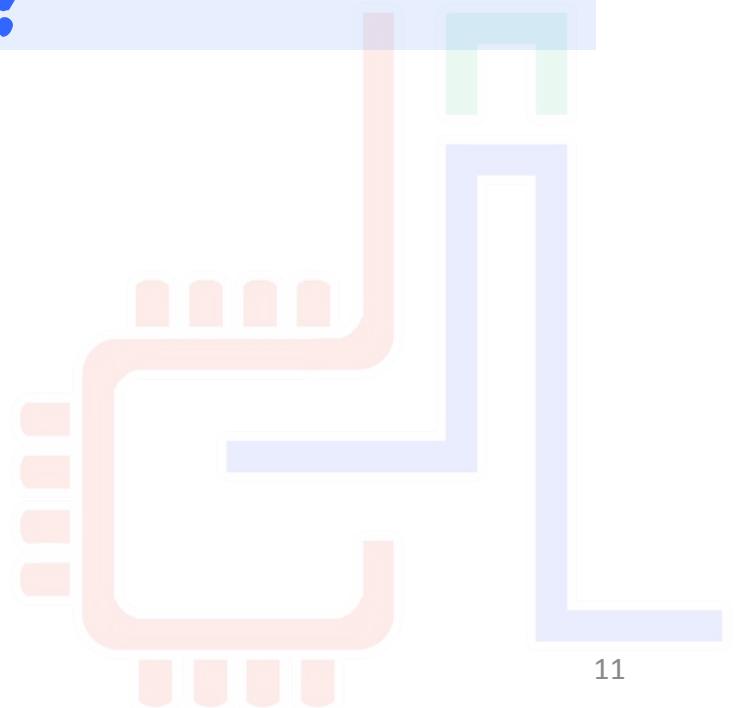
```
1 #include <stdio.h>
2
3 // décalartion des données
4 typedef struct h {
5     char nom[30];
6     int score;
7     float vitesse;
8 }Hero;
9 // décalartion des traitements
10 void marcher(Hero h){
11     // code de marcher
12 }
13 void sauter(Hero h){
14     // code de sauter
15 }
16 void courir(Hero h){
17     // code de courir
18 }
19 void tomber(Hero h){
20     // code de tomber
21 }
22
23 void main(){
24     Hero mario; // données
25     // traitement
26     marcher(mario);
27     sauter(mario);
28 }
```

APPROCHE PROCÉDURALE

- Raisonnement en terme de **fonctions** du système
 - l'accent est mis sur les fonctions et non sur les données
- Séparation des données et du code de traitement
 - Communication entre fonction Par variables globales risque de **l'effet de bord**
 - par passage de paramètres : Difficulté de réutiliser du code existant
- Ajout possible de nouvelles opérations à tout moment
- Décomposition fonctionnelle descendante

UN VOLONTAIRE POUR DÉCRIRE LA SALLE ?

UN ZÈME POUR DÉCRIRE LA SALLE MAIS
AUTREMENT ?

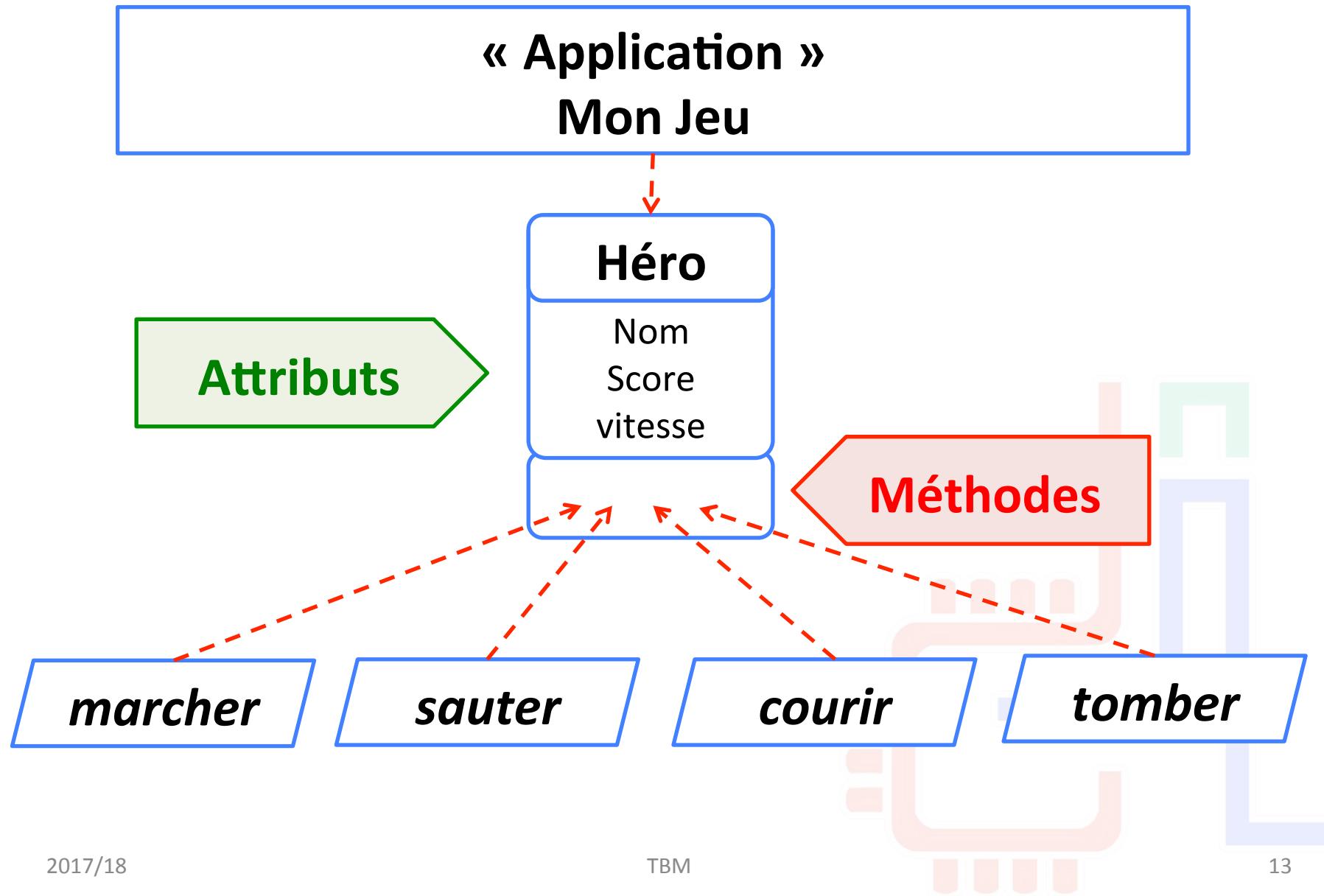


L'APPROCHE OBJET

- Diminution de l'écart entre **le monde réel** et sa **modélisation** informatique (approche naturelle)
 - le monde est avant tout objet
- Regroupement données-traitements
- Localisation des responsabilités : **encapsulation**
- Décomposition par identification des relations entre objets :
 - Association,
 - Composition,
 - Généralisation / Spécialisation



REGROUPEMENT DONNÉES-TRAITEMENTS



CODE EN JAVA

The diagram illustrates the structure of a Java program. It shows a code editor window titled "MonJeu.java" containing two classes: "Hero" and "MonJeu". The "Hero" class contains three attributes (nom, score, vitesse) and four methods (marcher, sauter, courir, tomber). The "MonJeu" class contains a main method that creates a new "Hero" object and calls its "marcher" method. A green curly brace on the right side of the screen groups the attributes of the "Hero" class under the heading "Attributs". A red curly brace on the right side of the screen groups the methods of the "Hero" class under the heading "Méthodes".

```
1  class Hero {  
2      String nom;  
3      int score;  
4      float vitesse;  
5  
6      void marcher(){  
7          // code marcher  
8      }  
9  
10     void sauter(){  
11         // code sauter  
12     }  
13  
14     void courir(){  
15         // code courir  
16     }  
17  
18     void tomber(){  
19         // code tomber  
20     }  
21 }  
22  
23 class MonJeu{  
24     public static void main(String[] args){  
25         Hero mario = new Hero();  
26         mario.marcher();  
27     }  
28 }
```

Attributs

Méthodes

LES DEUX APPROCHÉES

```
monJeu.c
```

```
1 #include <stdio.h>
2
3 // déclaration des données
4 typedef struct h {
5     char nom[30];
6     int score;
7     float vitesse;
8 }Hero;
9 // déclaration des traitements
10 void marcher(Hero h){
11     // code de marcher
12 }
13 void sauter(Hero h){
14     // code de sauter
15 }
16 void courir(Hero h){
17     // code de courir
18 }
19 void tomber(Hero h){
20     // code de tomber
21 }
22
23 void main(){
24     Hero mario; // données
25     // traitement
26     marcher(mario);
27     sauter(mario);
28 }
```

```
MonJeu.java
```

```
1 class Hero {
2     String nom;
3     int score;
4     float vitesse;
5
6     void marcher(){
7         // code marcher
8     }
9
10    void sauter(){
11        // code sauter
12    }
13
14    void courir(){
15        // code courir
16    }
17
18    void tomber(){
19        // code tomber
20    }
21
22
23 class MonJeu{
24     public static void main(String[] args){
25         Hero mario = new Hero();
26         mario.marcher();
27     }
28 }
```

UN PAS DANS LA BONNE DIRECTION



Etant donné que

- Verbes → Procédures et Fonctions
- Noms → Données

Alors

- Programmation Procédurale = Collection de Verbes Supportés par des Noms (substantifs)
- Programmation Objet = Collection de Noms Supportés par des Verbes

UN EXEMPLE POUR MIEUX COMPRENDRE

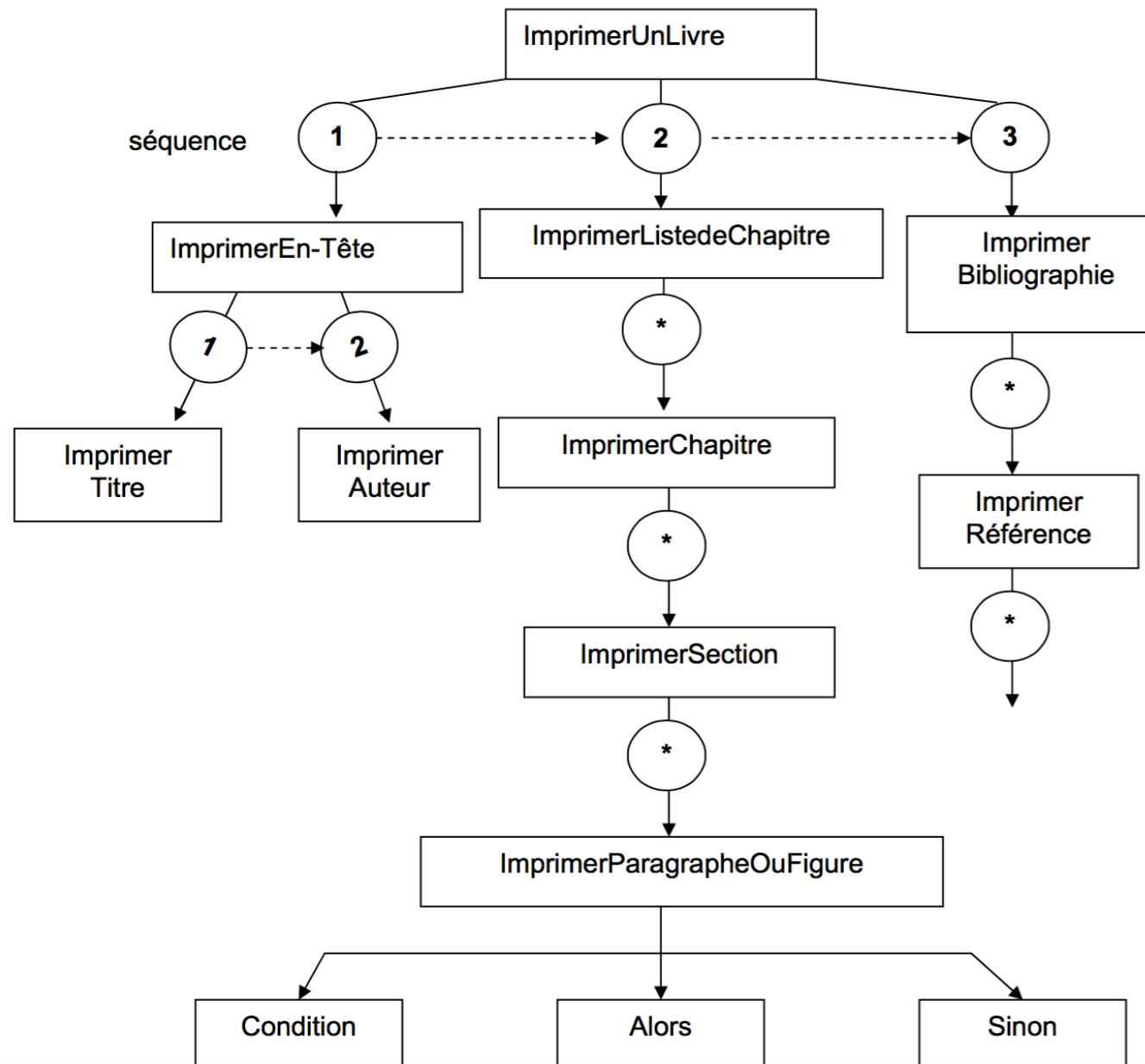
Donner l'architecture d'un programme qui permet imprimer un livre

Avec les 2 approches Procédurale et Objet

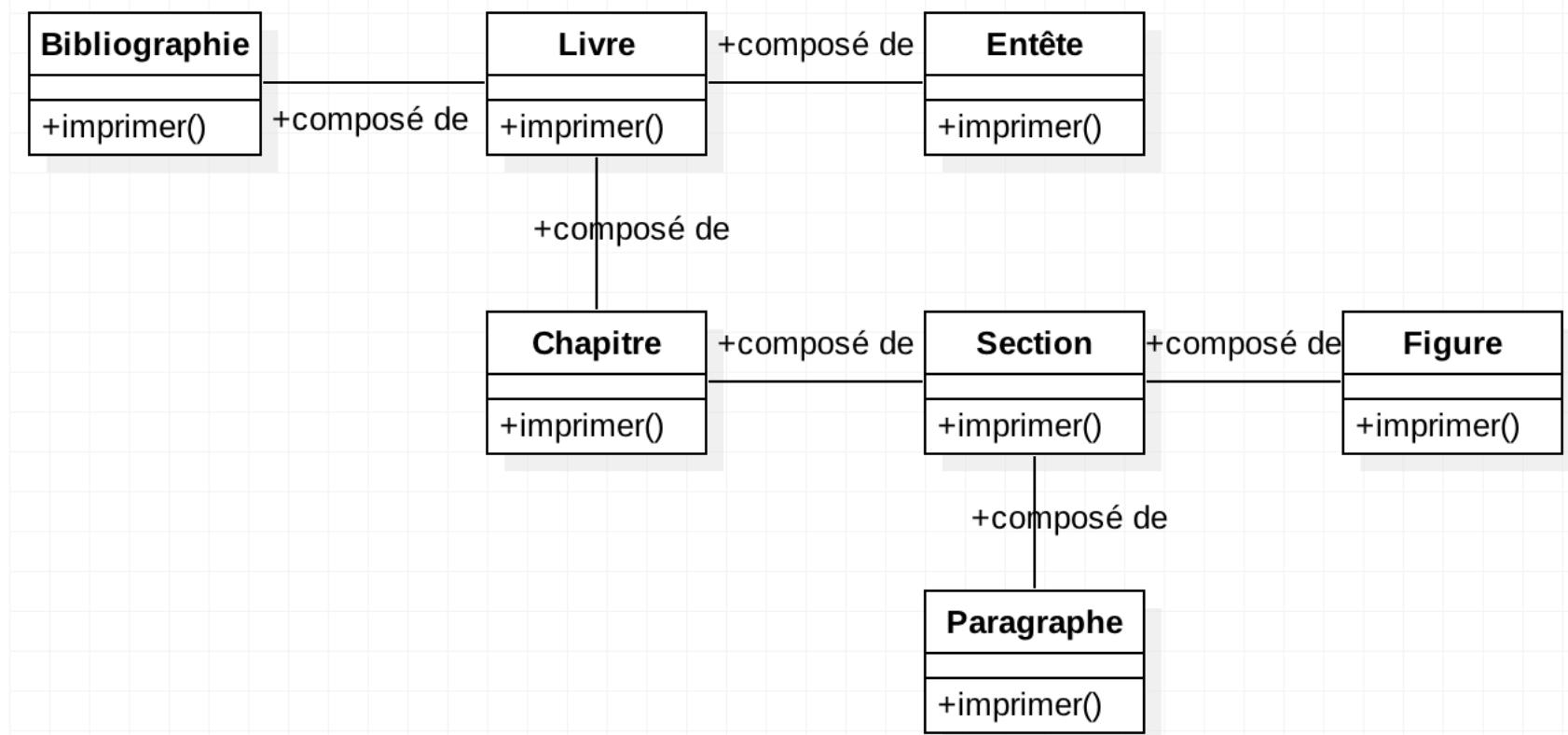
UNE ANALYSE EST INDISPENSABLE EN INFORMATIQUE AVANT TOUT

Donner l'architecture d'un programme qui permet imprimer un livre. Un livre est constitué d'un en-tête, d'une liste de chapitres et d'une bibliographie. Chaque chapitre est constitué d'une suite de sections, elles-mêmes composées de paragraphes et de figures

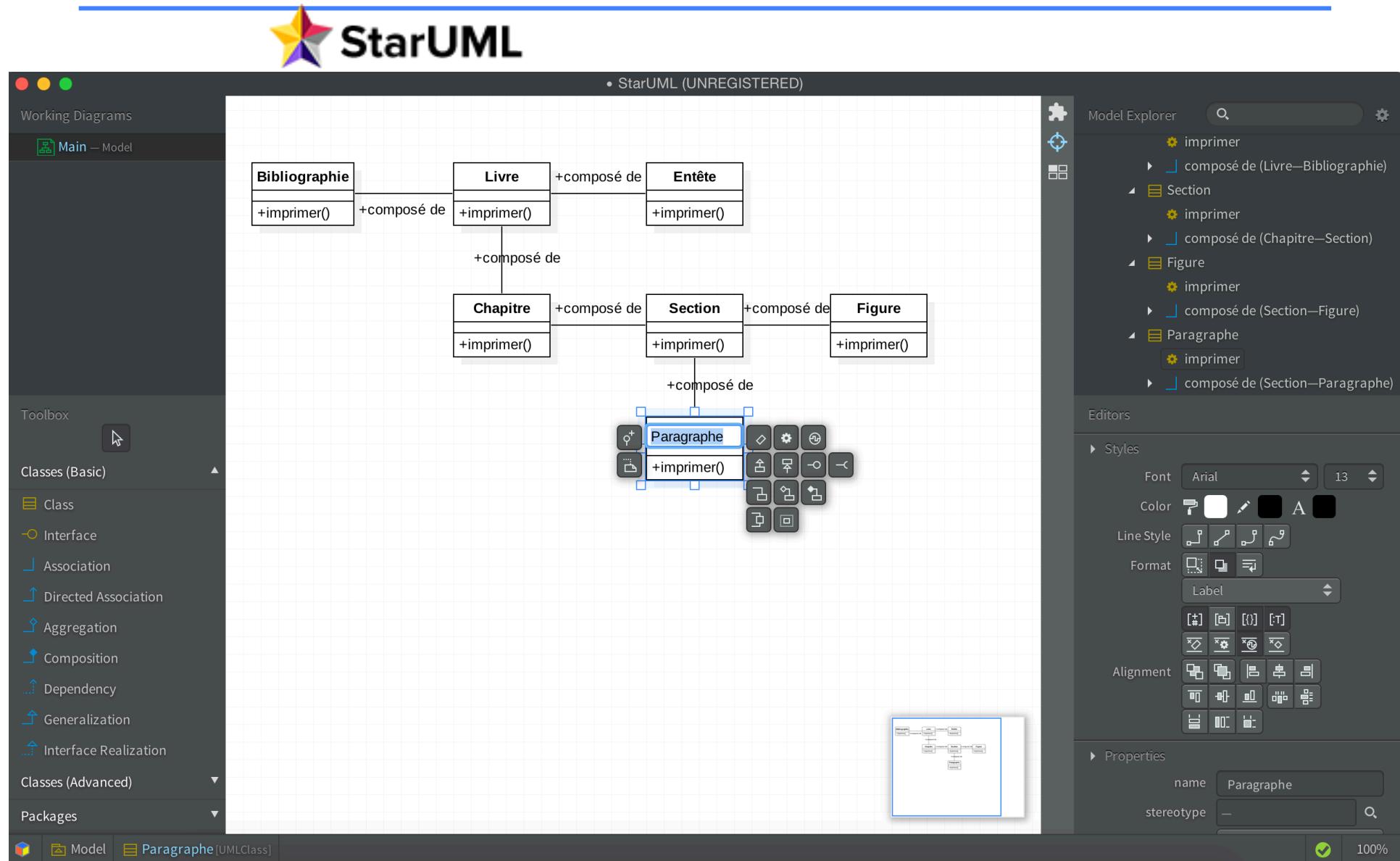
CORRECTION- APPROCHE PROCÉDURALE



CORRECTION - APPROCHE OBJET



IL Y A DES LOGICIELS POUR FAIRE CE GENRE DE DIAGRAMME



POUR RÉSUMER



L'approche Procédurale

- Que doit faire le programme?
- Collecte des verbes supporté par les noms
- Unité de décomposition la fonction
- Séparation entre les données et les traitements
- Décomposition fonctionnelle descendante (Hiérarchie)

Approche Objet

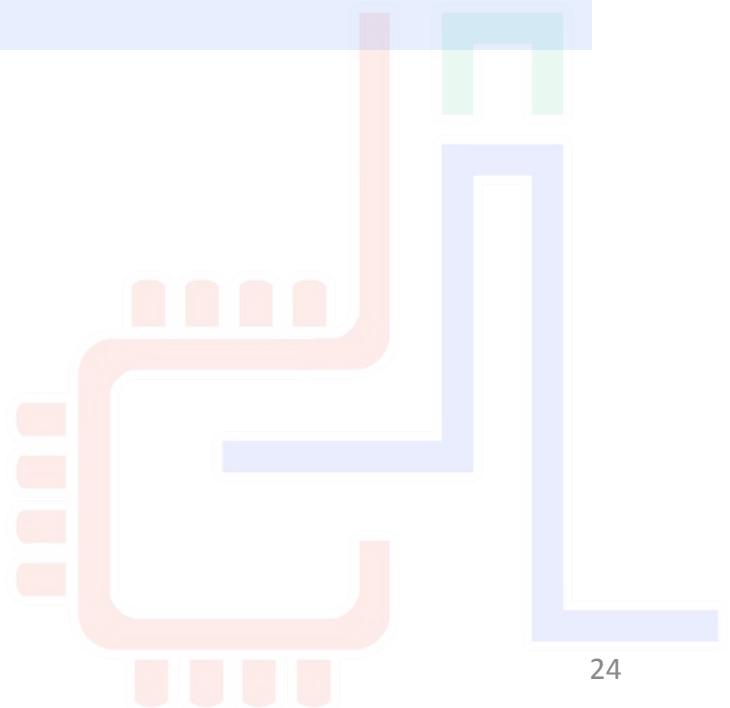
- De quoi est composé le programme ?
- Collecte des noms supportés par des verbes
- Unité de décomposition l'Objet
- Encapsulation des données + traitement
- Relations quelconques entre les objets (Réseau)

UN 2ÈME EXEMPLE

Trier un tableau de n éléments entier

Avec les 2 approches Procédurale et Objet

POUR FAIRE DE LA POO ON A BESOIN D'UN
LOO



LANGAGES ORIENTÉS-OBJETS POUR LA POO



[Smalltalk]



Ruby

Ada
2012



Java™



python



CLASSEMENT DES LANGAGES [IEEE 2017]

Language Rank	Types	Trending Ranking
1. Python	🌐💻	100.0
2. C	💻🖥️⚙️	98.4
3. C++	💻🖥️⚙️	97.7
4. Java	🌐💻	97.0
5. Swift	💻	88.6
6. JavaScript	🌐📱	87.2
7. Go	🌐💻	86.5
8. R	💻	84.4
9. C#	🌐💻	80.2
10. Ruby	🌐💻	79.1

Top 10 des langages en croissance

Language Rank	Types	Jobs Ranking
1. Java	🌐💻	100.0
2. C	💻🖥️⚙️	99.3
3. Python	🌐💻	99.3
4. C++	💻🖥️⚙️	92.6
5. JavaScript	🌐📱	90.2
6. C#	🌐💻	86.6
7. PHP	🌐	81.0
8. HTML	🌐	79.6
9. Ruby	🌐💻	77.2
10. Swift	💻	77.2

Top 10 des langages les plus demandés par pro

Language Rank	Types	Spectrum Ranking
1. Python	🌐💻	100.0
2. Java	🌐💻	99.4
3. C#	🌐💻	88.6
4. JavaScript	🌐📱	85.5
5. PHP	🌐	81.4
6. Go	🌐💻	76.1
7. Ruby	🌐💻	72.4
8. Scala	🌐📱	68.3
9. HTML	🌐	67.0
10. Perl	🌐💻	57.6

Top 10 des langages pour Web

Language Rank	Types	Spectrum Ranking
1. C	💻🖥️⚙️	99.7
2. Java	🌐💻	99.4
3. C++	💻🖥️⚙️	97.2
4. C#	🌐💻	88.6
5. JavaScript	🌐📱	85.5
6. Swift	💻	75.3
7. Scala	🌐📱	68.3
8. Objective-C	💻	45.2
9. Delphi	💻	41.1
10. Scheme	💻	19.8

TBM

Top 10 des langages pour mobile

2017/18

26

ON APPRENDRA LA POO AVEC JAVA



Java SE



Java ME



Java EE

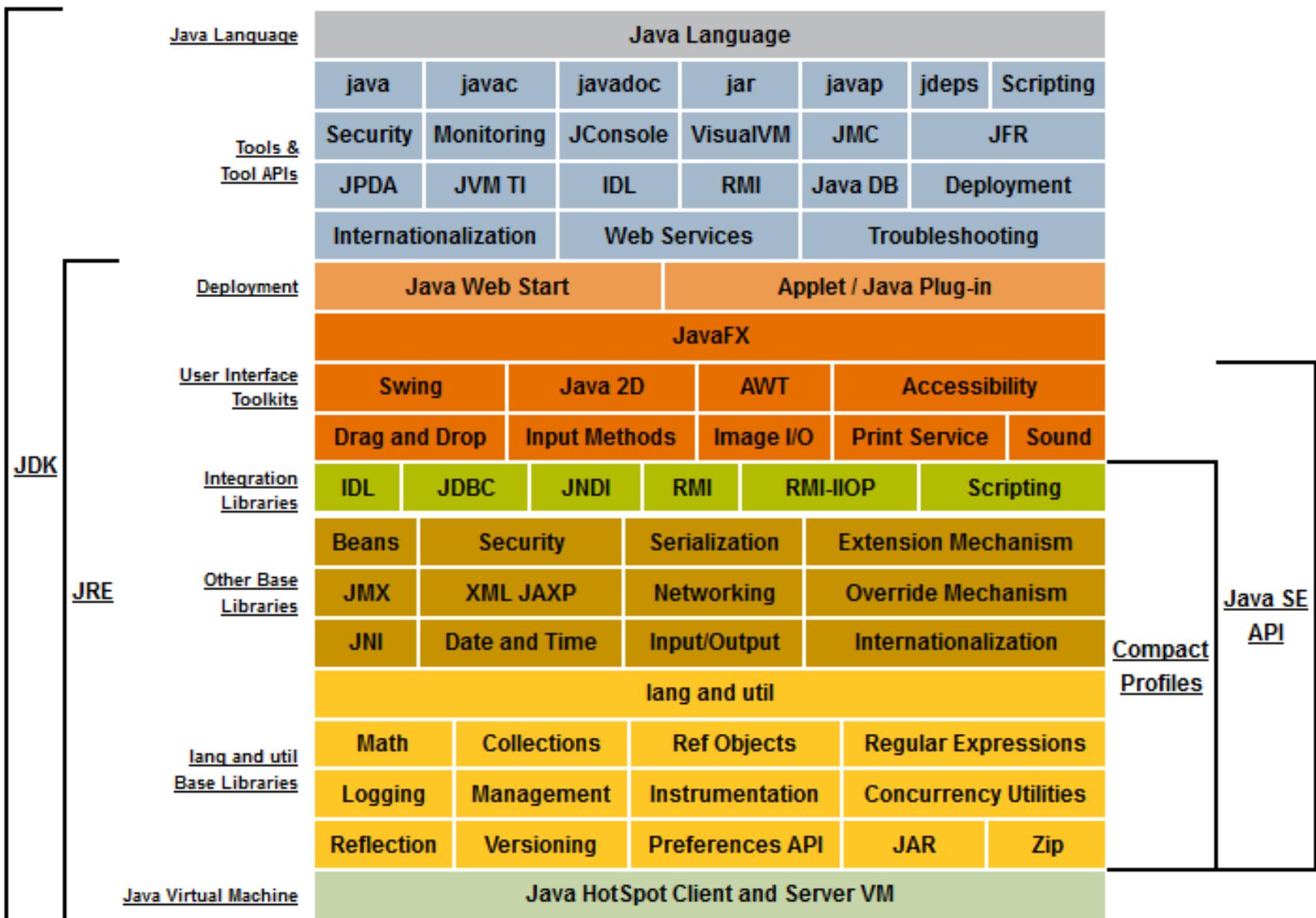


Java
Embedded

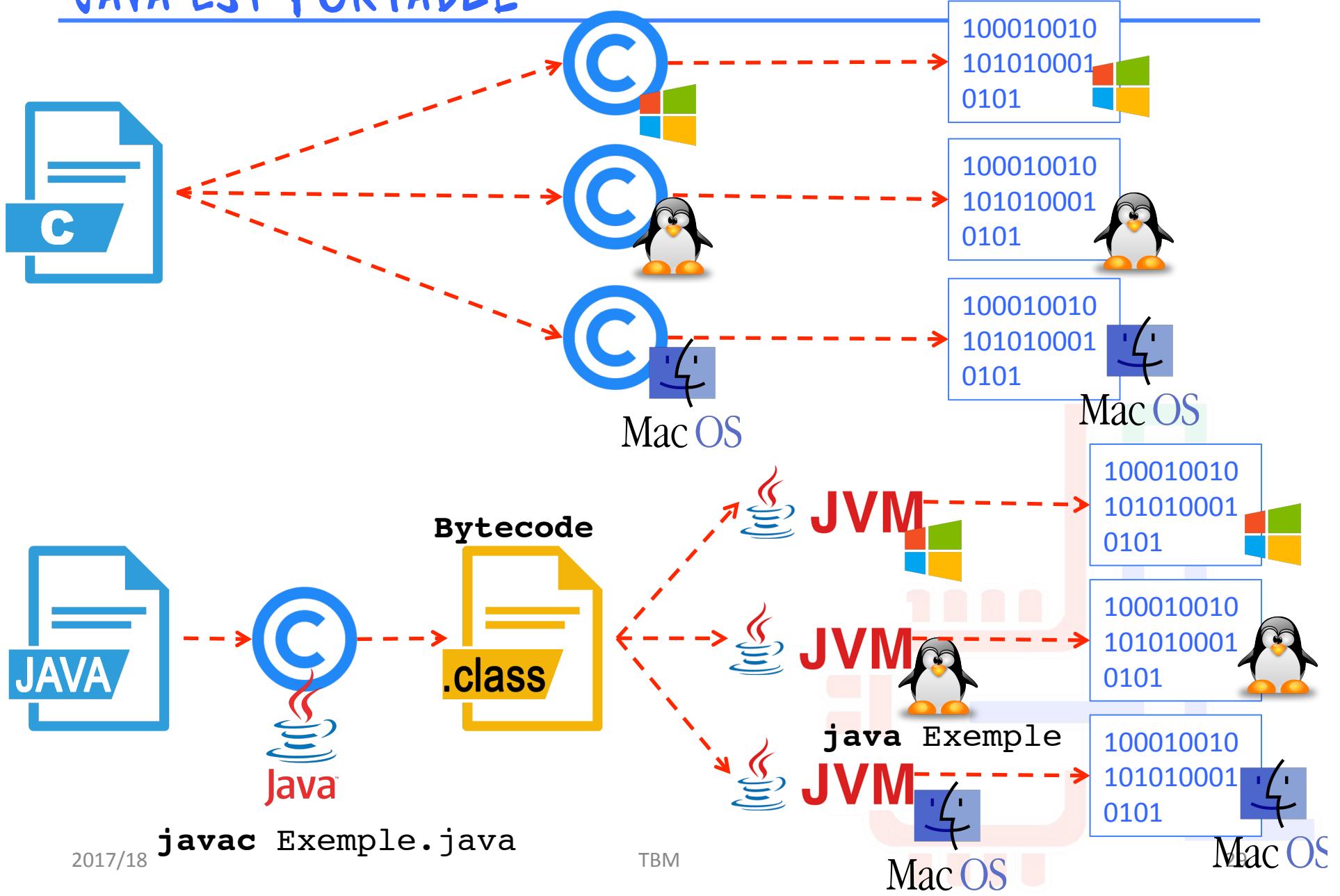


Java
Card

JAVA : PLATE-FORME JDK 8



JAVA EST PORTABLE



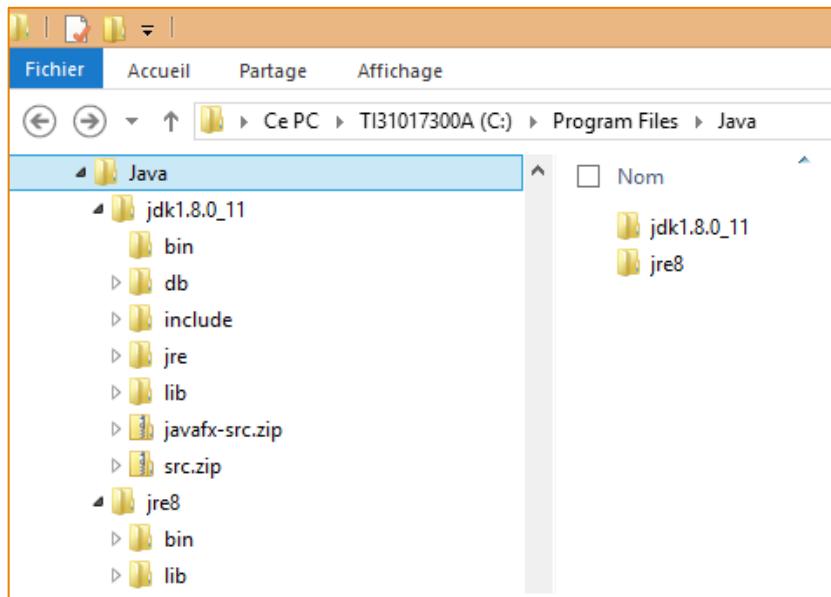
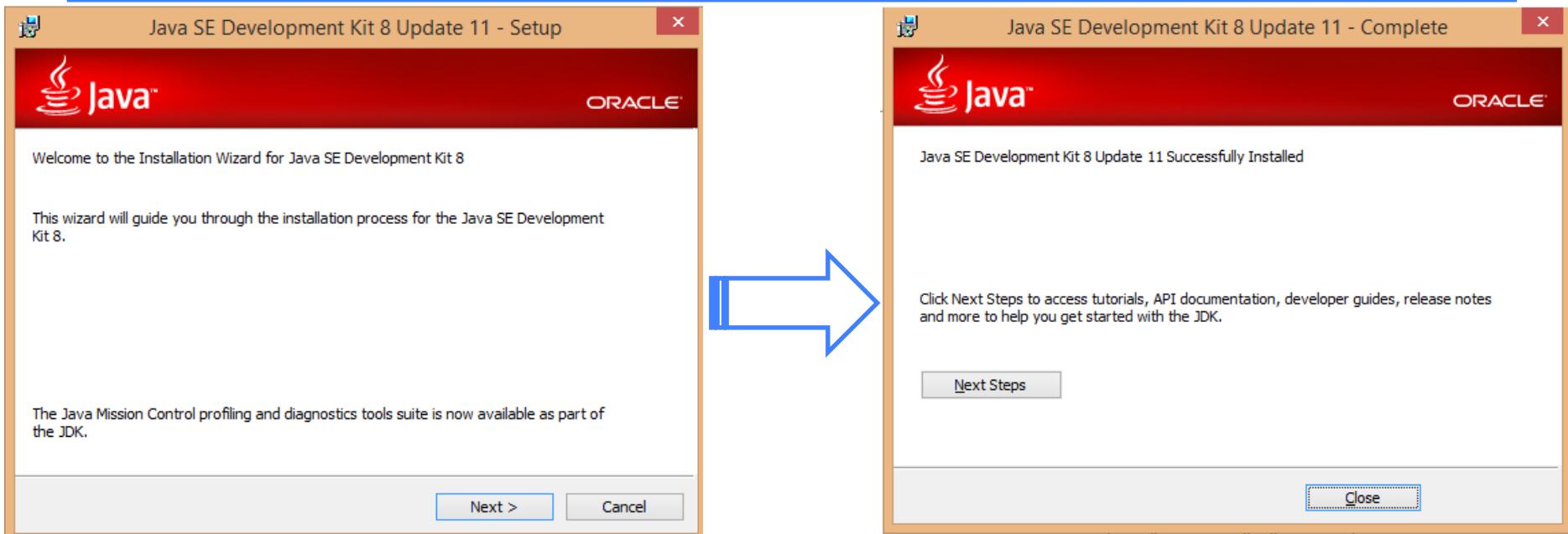
ETAPE 1 : TÉLÉCHARGEMENT DU JDK

- <http://www.oracle.com/technetwork/java/index.html>

The screenshot shows the Oracle Java Technology Network homepage. On the left, there's a large blue banner with white clouds and text about creating a free Oracle Cloud account. The right side features a "Software Downloads" section with tabs for "Top Down" (selected), "New Downloads", and "View All Downloads". A red arrow points from the text "ICI" to the "Top Down" tab. Below this, there's a section for the "Java SE Development Kit 8u11" with a license agreement checkbox. At the bottom, a table lists download links for various Java SE versions across different platforms.

Product / File Description	File Size	Download
Linux x86	133.58 MB	jdk-8u11-linux-i586.rpm
Linux x86	152.55 MB	jdk-8u11-linux-i586.tar.gz
Linux x64	133.89 MB	jdk-8u11-linux-x64.rpm
Linux x64	151.65 MB	jdk-8u11-linux-x64.tar.gz
Mac OS X x64	207.82 MB	jdk-8u11-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.66 MB	jdk-8u11-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	96.14 MB	jdk-8u11-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.7 MB	jdk-8u11-solaris-x64.tar.Z
Solaris x64	93.18 MB	jdk-8u11-solaris-x64.tar.gz
Windows x86	151.81 MB	jdk-8u11-windows-i586.exe
Windows x64	155.29 MB	jdk-8u11-windows-x64.exe

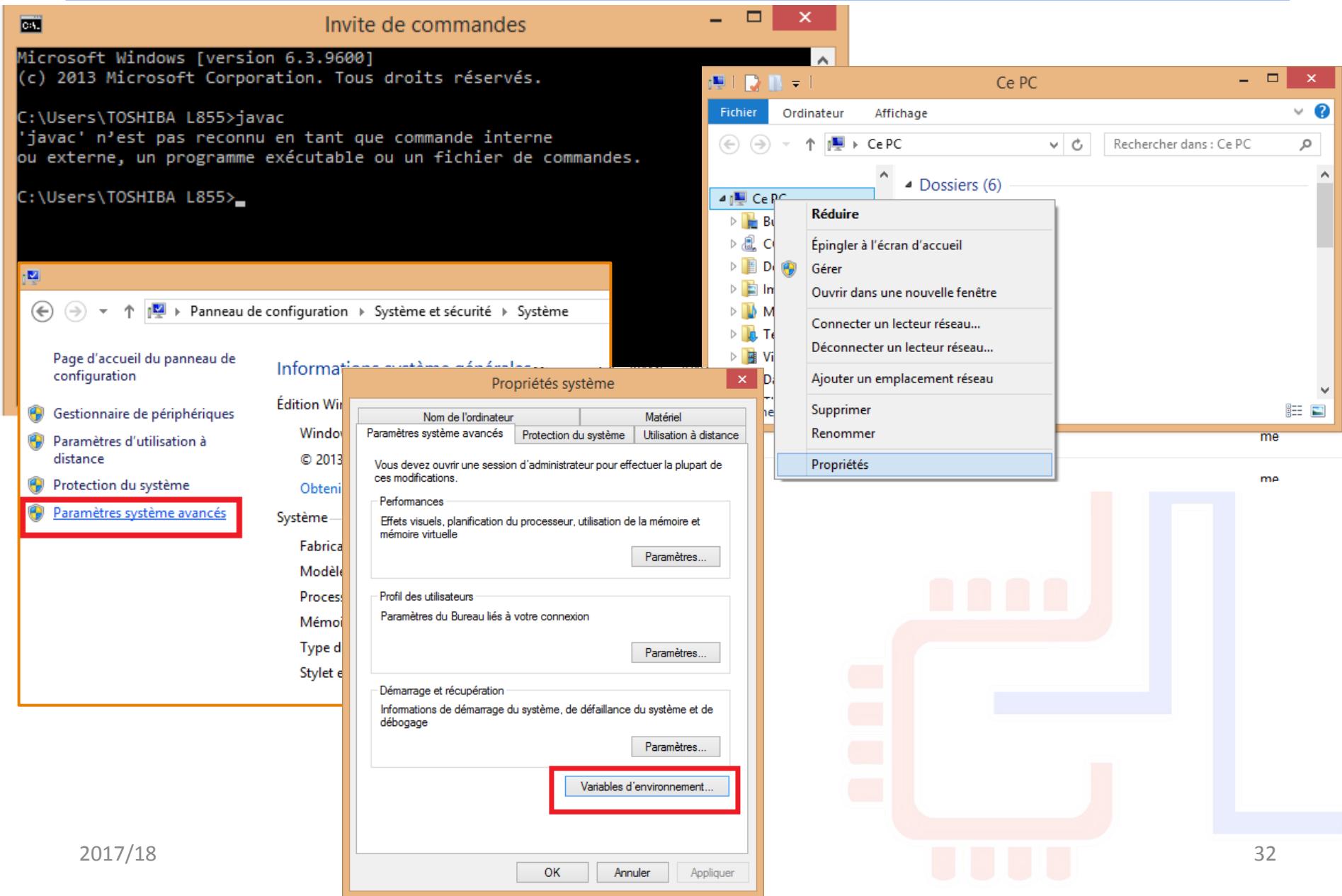
ETAPE 2 : INSTALLATION DU JDK PAR ÉTAPE



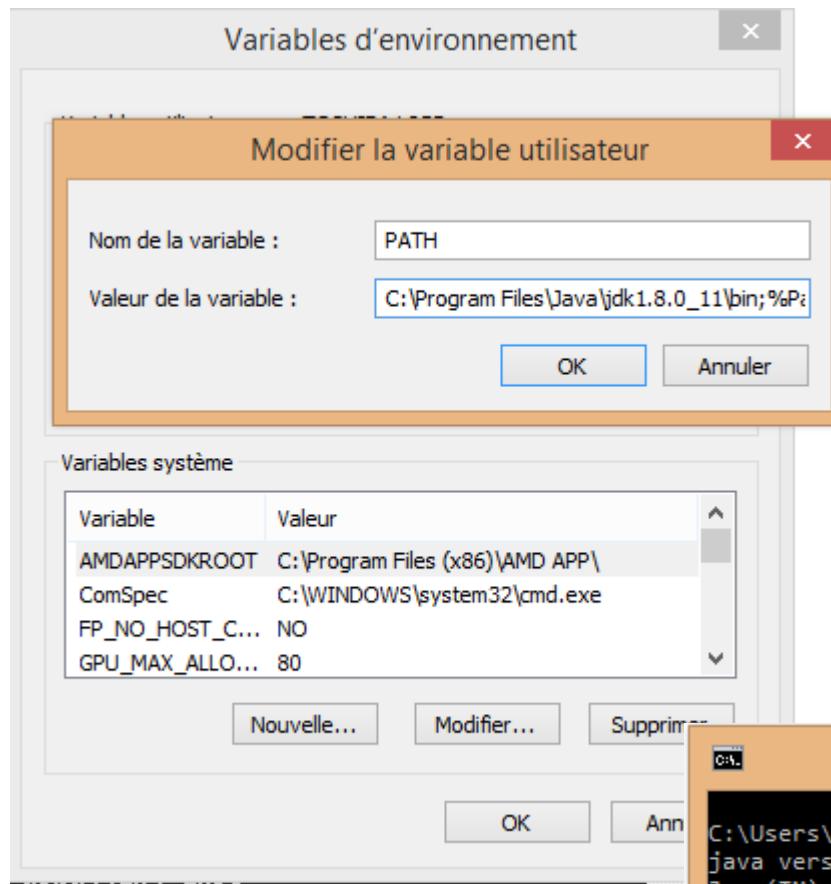
jdk1.8.0_11/bin contient :

- Le compilateur : **javac**
- La machine virtuelle : **java**
- L'outil de génération de documentation : **Javadoc**
- L'outil de création de 'livrables' pour vos clients : **jar**
- L'outil de surveillance de vos applications Java : **jvisualvm**

ETAPE 3 : CONFIGURATION



ETAPE 3 : CONFIGURATION (SUITE)



Ajoutez une nouvelle variable **PATH** et associez la valeur suivante : **C :\Program Files\Java\jdk1.8.0_11\bin ;%Path%** .

```
C:\Users\TOSHIBA L855>java -version
java version "1.8.0_11"
Java(TM) SE Runtime Environment (build 1.8.0_11-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.11-b03, mixed mode)

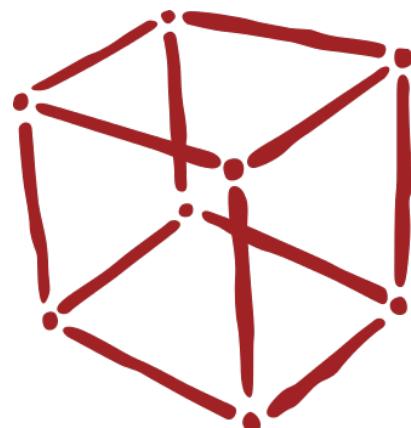
C:\Users\TOSHIBA L855>
```

ETAPE 4 : TÉLÉCHARGER UN IDE : QUE CHOISIR ?



eclipse

<https://www.eclipse.org/downloads/>



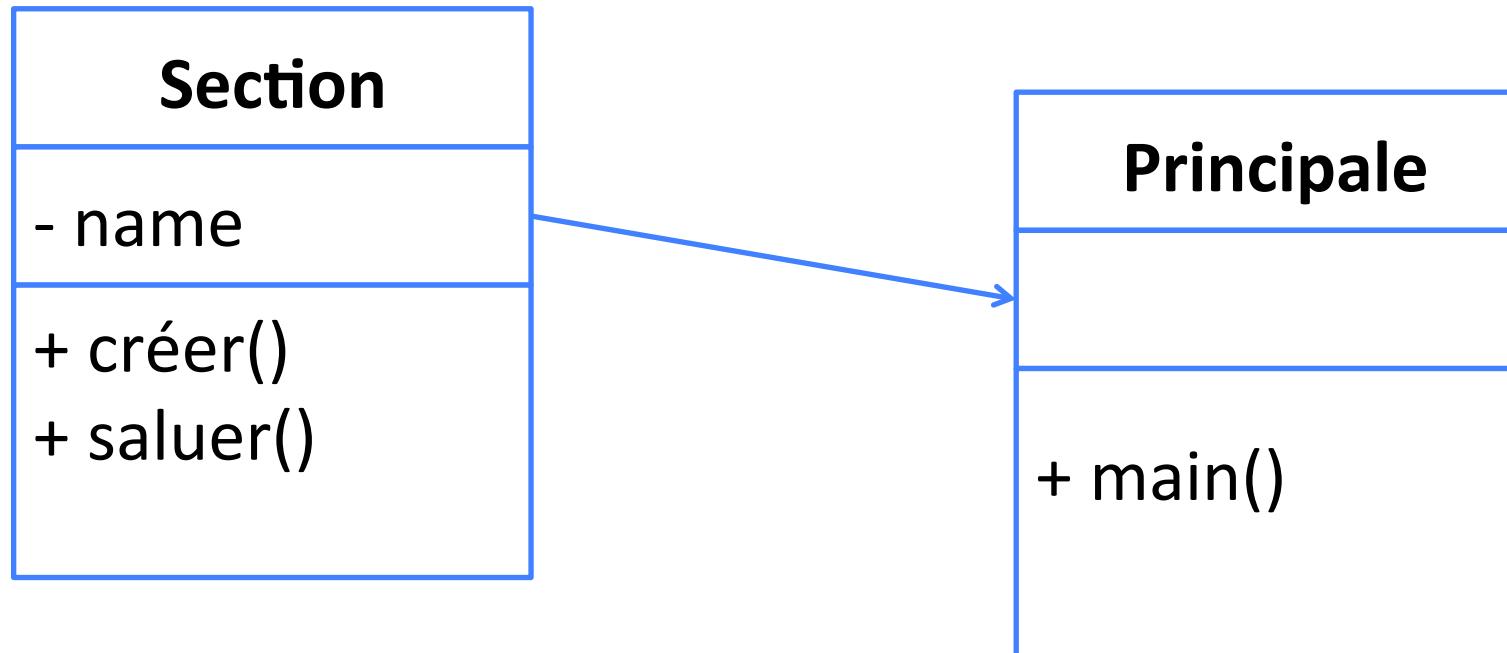
NetBeans

<https://netbeans.org/downloads/>

IL EST TEMPS POUR L'INCONTOURNABLE
« HELLO WORLD ! ON EST LES SIZ ZK17 »



UNE CONCEPTION S'IMPOSE TOUJOURS



APRÈS LA CONCEPTION, LE CODAGE

The image shows a Java development environment with two code editors side-by-side.

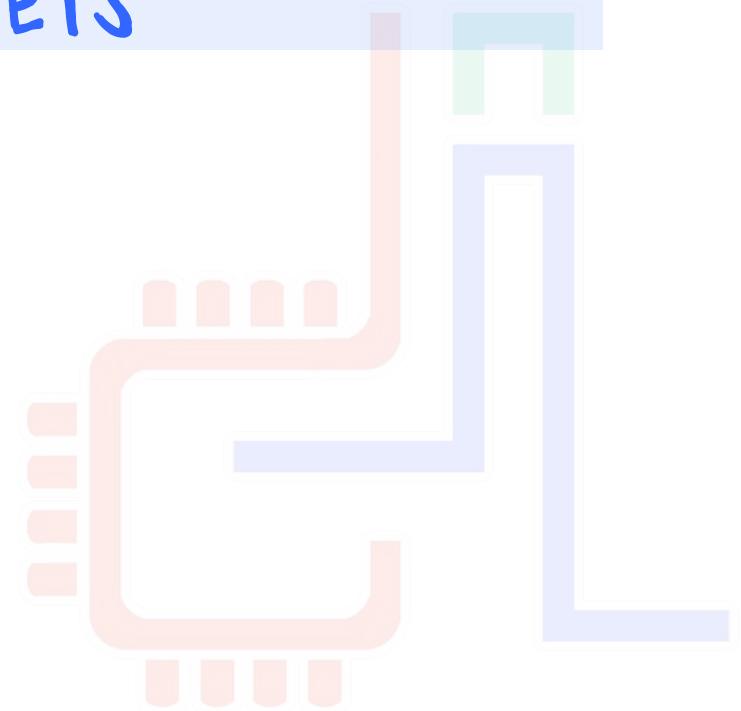
Top Editor (Section.java):

```
1 package hello;
2
3 public class Section { //début de la classe Section
4     String name;
5     // Constructeur qui permet de créer des objets
6     Section(String n){
7         name = n;
8     }
9     // méthode qui permet de saluer
10    void saluer(){
11        System.out.println("Hello world !! on est les "+ name);
12    }
13 } // Fin de la classe Section
14
```

Bottom Editor (Principale.java):

```
1 package hello;
2
3 // Programme principal
4 public class Principale {
5
6     public static void main(String[] args) {
7         Section si2 = new Section("SI2 2K17");
8         si2.saluer();
9     }
10 }
```

CONCEPTS ORIENTÉS-OBJETS



NOTION D'OBJET

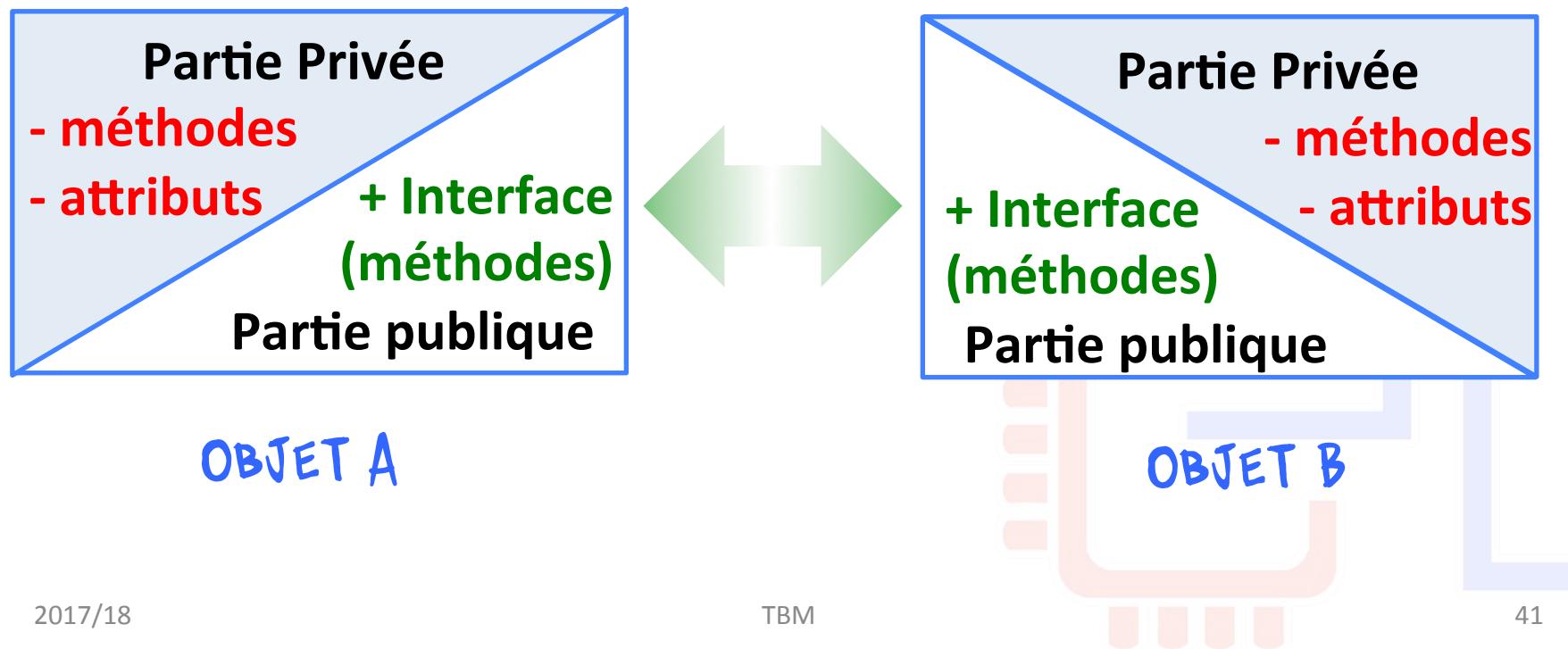
- Un objet définit une **représentation d'une entité** atomique réelle ou virtuelle, dans le but de le piloter ou de le simuler.
- Un objet encapsule données et traitements en ne laissant visible que **l'interface** de l'objet, c'est à dire les traitements que l'on peut faire dessus
- **Objet = Identité + Etat + Comportement**
 - **Identité** : caractérise de manière univoque l'existence propre de l'objet.
 - **Etat** : décrit les propriétés (**attributs**) d'un objet à un moment donné
 - **Comportement** : définit les propriétés dynamiques (**méthodes**) de l'objet : comment il agit et comment il réagit aux informations qui lui parviennent de son environnement.

COMMENT LES OBJETS COMMUNIQUENT
ENSEMBLE?

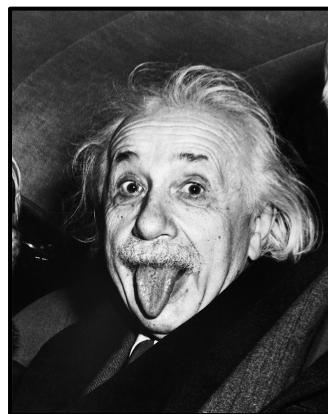


RELATION ENTRE OBJETS : INTERFACE

- Abstraction des données
 - Un objet est complètement défini par son **interface**
 - Les autres objets n'ont pas à connaître **l'implémentation interne** de l'objet pour communiquer avec lui



NOTION DE CLASSE : UN PAS VERS L'ABSTRACTION



NOTION DE CLASSE : UN PAS VERS L'ABSTRACTION

Personne
- nom
- age
- metier

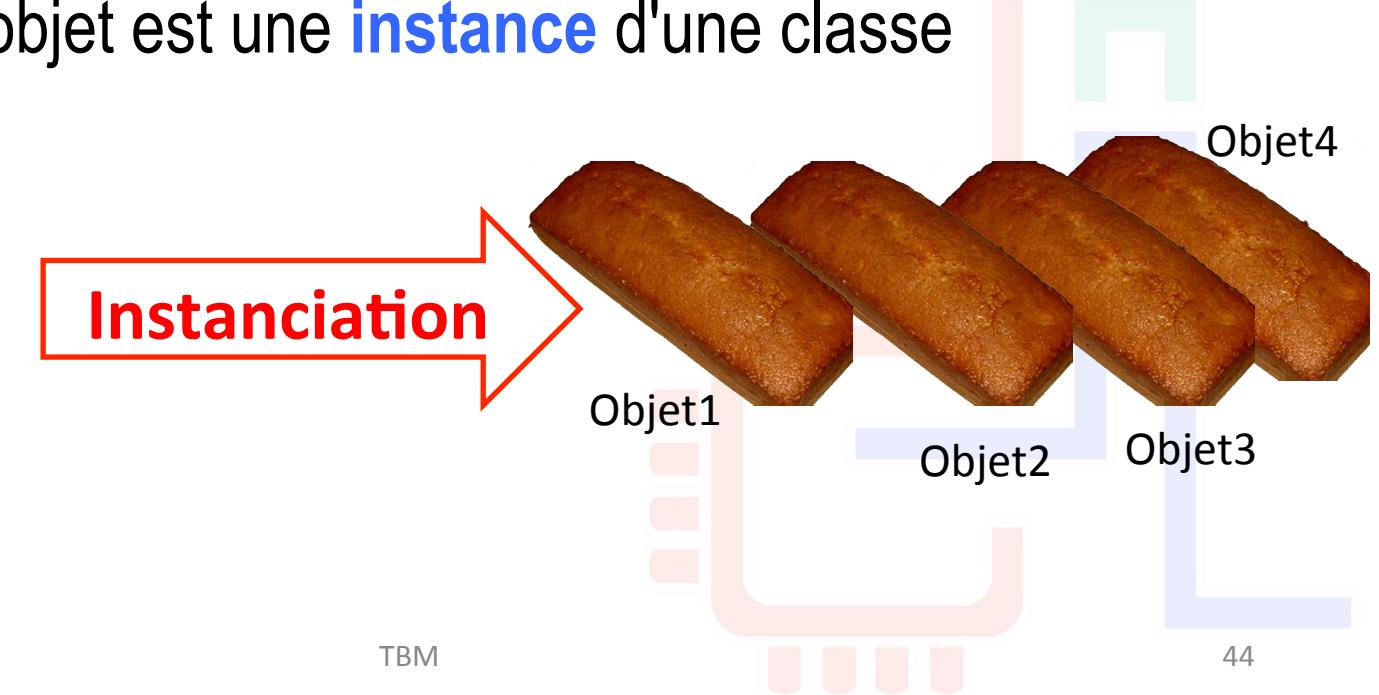


NOTION DE CLASSE : DÉFINITION

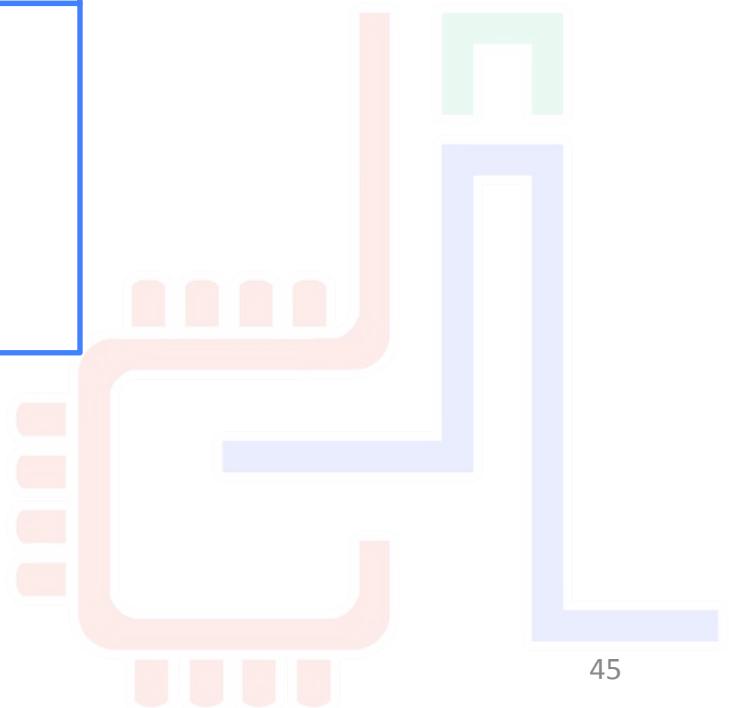
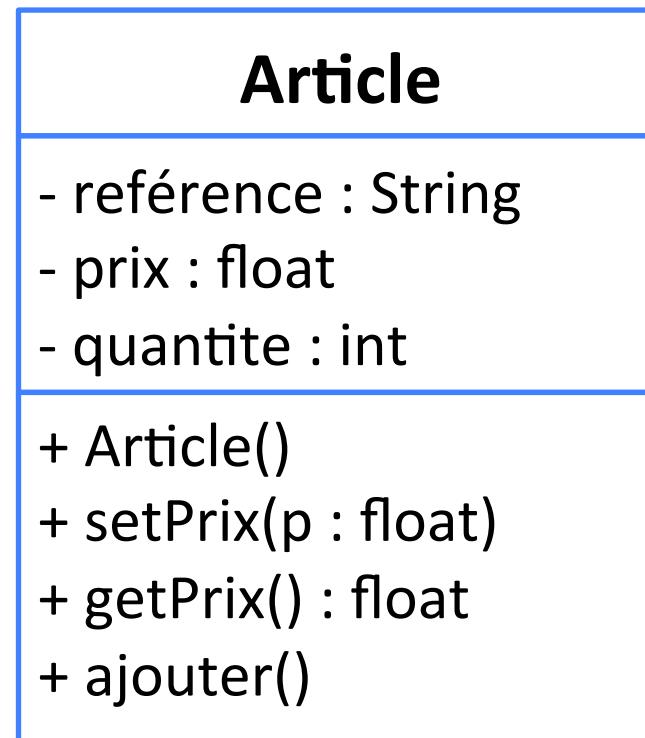
- Objet **prototypique** décrivant l'ensemble des propriétés (structure d'attribut, comportements) communs aux **éléments** de la classe.
- Un objet est donc « issu » d'une classe, c'est le produit qui sort d'un moule.
- On dit qu'un objet est une **instance** d'une classe



Classe



EXEMPLE D'UNE CLASSE ...



... EN CODE JAVA

Article

- référence : String
- prix : float
- quantite : int

- + Article()
- + setPrix(p : float)
- + getPrix() : float
- + ajouter()

```
1  public class Article { // début de la classe
2      // les attributs
3      private String reference;
4      private float prix;
5      private int quantite;
6      // méthodes
7      /* constructeur qui permet de créer les objets
8       * instances de la classe */
9
10     public Article(String r, float p, int q){
11         reference = r;
12         prix = p;
13         quantite = q;
14     }
15     /* un mutateur et accesseur qui permettent
16      * de modifier l'état d'un objet */
17     public void setPrix(float p){
18         prix = p;
19     }
20     public float getPrix(){
21         return prix;
22     }
23     /* une méthode d'implémentation */
24     public void ajouter(){
25         quantite
```

BEST PRACTICE

fichier .java doit avoir le **même** nom que la classe et le constructeur

Le Nom de la Classe doit commencer par une **Majuscule CamelCase**

L'attribut et méthode commencent par une **minuscule le 2^e mot par une Majuscule CamelCase**

PLEASE



/* Commenter votre code */

```
Article.java
```

```
1 public class Article { // début de la classe
2     // les attributs
3     private String reference;
4     private float prix;
5     private int quantite;
6     // méthodes
7     /* constructeur qui permet de créer les objets
8      * instances de la classe */
9     public Article(String r, float p, int q){
10         reference = r;
11         prix = p;
12         quantite = q;
13     }
14     /* un mutateur et accesseur qui permettent
15      * de modifier l'état d'un objet */
16     public void setPrix(float p){
17         prix = p;
18     }
19     public float getPrix(){
20         return prix;
21     }
22     /* une méthode d'implémentation */
23     public void ajouter(){
24         quantite++;
25     }
26 }
27 } TBM
```

UNE CLASSE CONTIENT :

- Un nom
- Des attributs
 - Chaque attribut représente une caractéristique de la classe
 - Décrivent l'état de la classe
- Des méthodes :
 - **Constructeur(s)** : permet de créer une instance
 - **Modificateur** ou **setter** ou **mutateur** : modifier une caractéristique
 - **Observateur** ou **getter** ou **accesseur** : voir le contenu d'une caractéristique
 - **Destructeur** : détruire l'objet
 - **Méthodes d'implémentation** : qui décrive le comportement de la classe

DANS DU CODE

Nom de la classe

Attribut

Constructeur

Setter de l'attribut prix

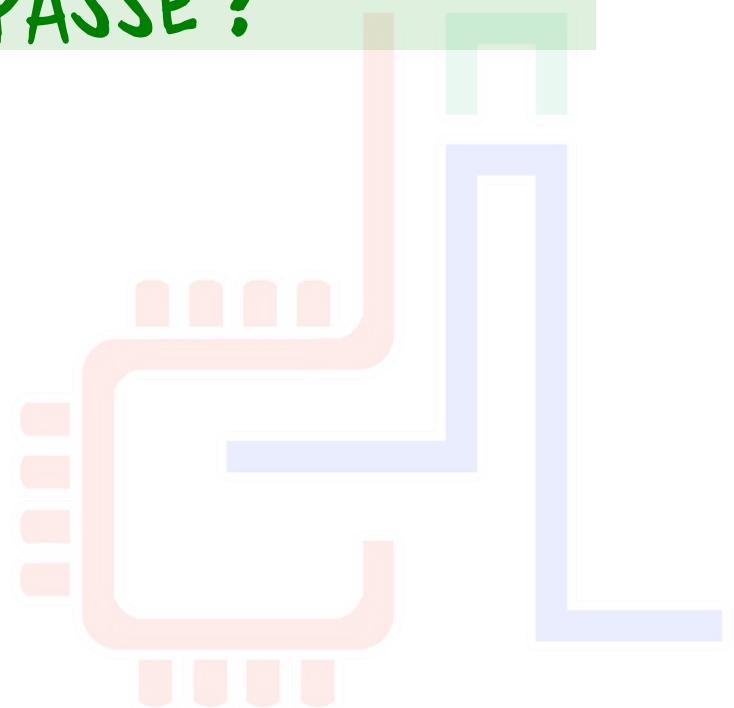
getter de l'attribut prix

Méthode
d'implémentation

```
Article.java
```

```
1 public class Article { // début de la classe
2     // les attributs
3     private String reference;
4     private float prix;
5     private int quantite;
6     // méthodes
7     /* constructeur qui permet de créer les objets
8      * instances de la classe */
9
10    public Article(String r, float p, int q){
11        reference = r;
12        prix = p;
13        quantite = q;
14    }
15    /* un mutateur et accesseur qui permettent
16     * de modifier l'état d'un objet */
17    public void setPrix(float p){
18        prix = p;
19    }
20    public float getPrix(){
21        return prix;
22    }
23    /* une méthode d'implémentation */
24    public void ajouter(){
25        quantite++;
26    }
27 }
```

ET SI ON EN CRÉÉ DES OBJETS
QU'EST CE QUI SE PASSE ?

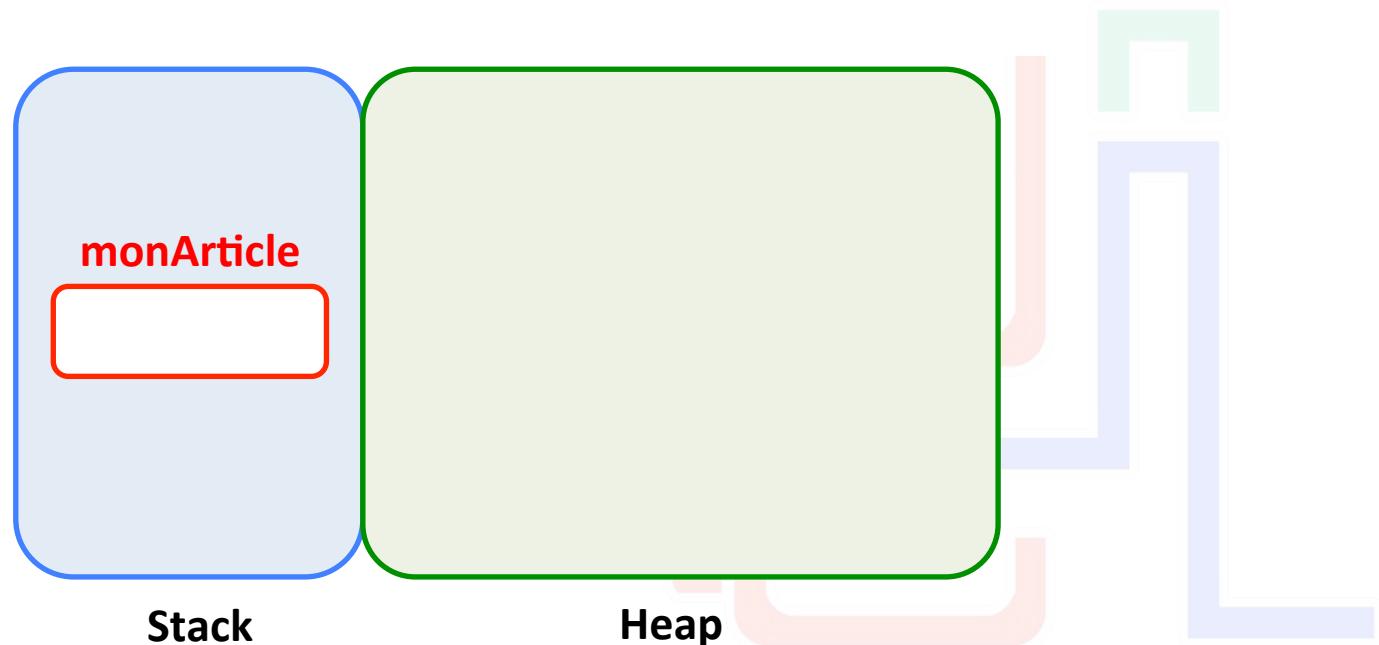


NOTION DE RÉFÉRENCEMENT D'OBJET

■ Déclaration

```
27  
28  public static void main(String arg[]){  
29      // déclaration  
30      Article monArticle;  
31      // Instanciation  
32      monArticle = new Article("EDX07", 23.0f, 100);  
33      // changement d'état  
34      monArticle.setPrix(21.0f);  
35 }
```

■ Mémoire

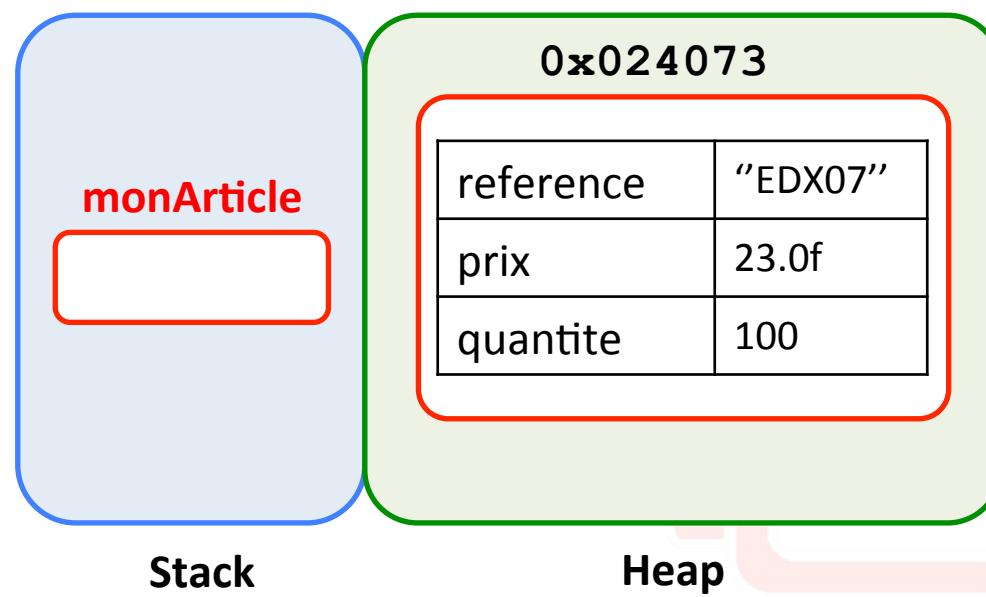


NOTION DE RÉFÉRENCEMENT D'OBJET

■ Instanciation

```
27  
28  public static void main(String arg[]){  
29      // déclaration  
30      Article monArticle;  
31      // Instanciation  
32      monArticle = new Article("EDX07", 23.0f, 100);  
33      // changement d'état  
34      monArticle.setPrix(21.0f);  
35 }
```

■ Mémoire

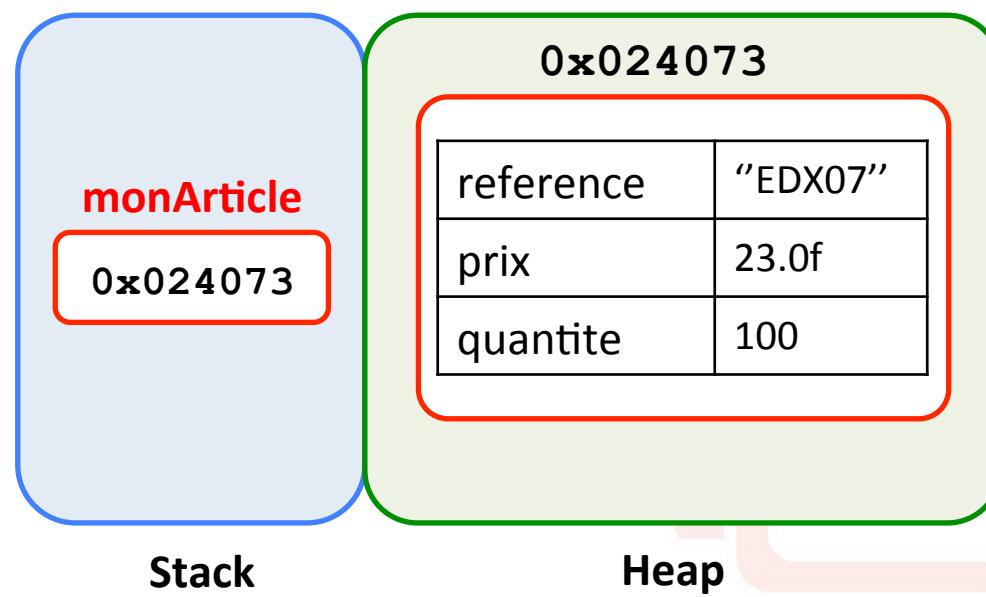


NOTION DE RÉFÉRENCEMENT D'OBJET

■ Instanciation

```
27  
28  public static void main(String arg[]){  
29      // déclaration  
30      Article monArticle;  
31      // Instanciation  
32      monArticle = new Article("EDX07", 23.0f, 100);  
33      // changement d'état  
34      monArticle.setPrix(21.0f);  
35 }
```

■ Mémoire

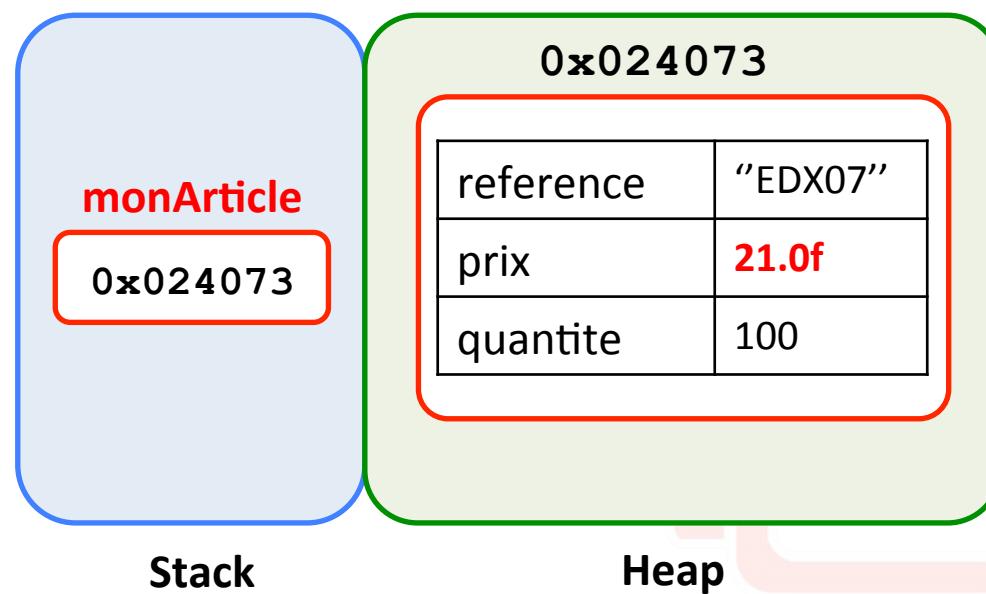


NOTION DE RÉFÉRENCEMENT D'OBJET

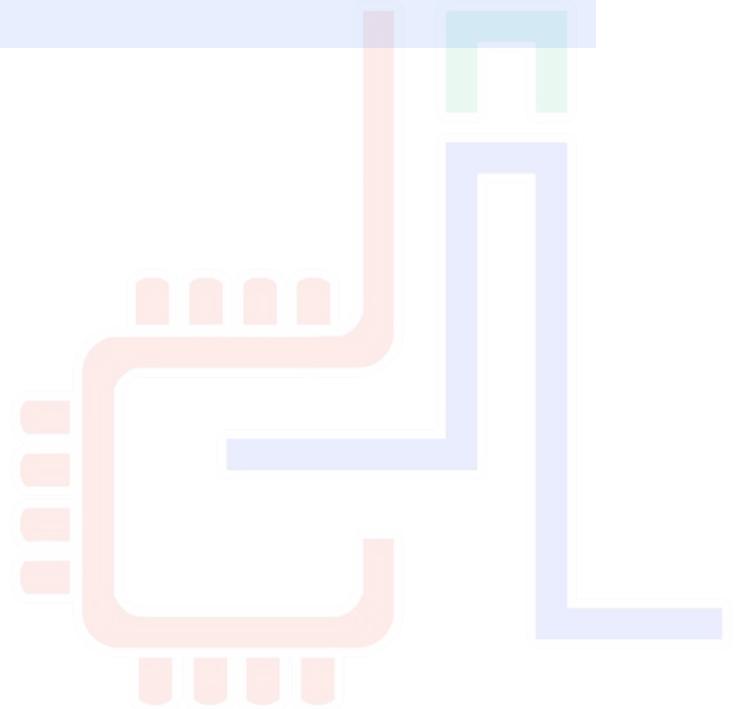
■ Changement d'état

```
27  
28  public static void main(String arg[]){  
29      // déclaration  
30      Article monArticle;  
31      // Instanciation  
32      monArticle = new Article("EDX07", 23.0f, 100);  
33      // changement d'état  
34      monArticle.setPrix(21.0f);  
35 }
```

■ Mémoire



LES ABC DE JAVA



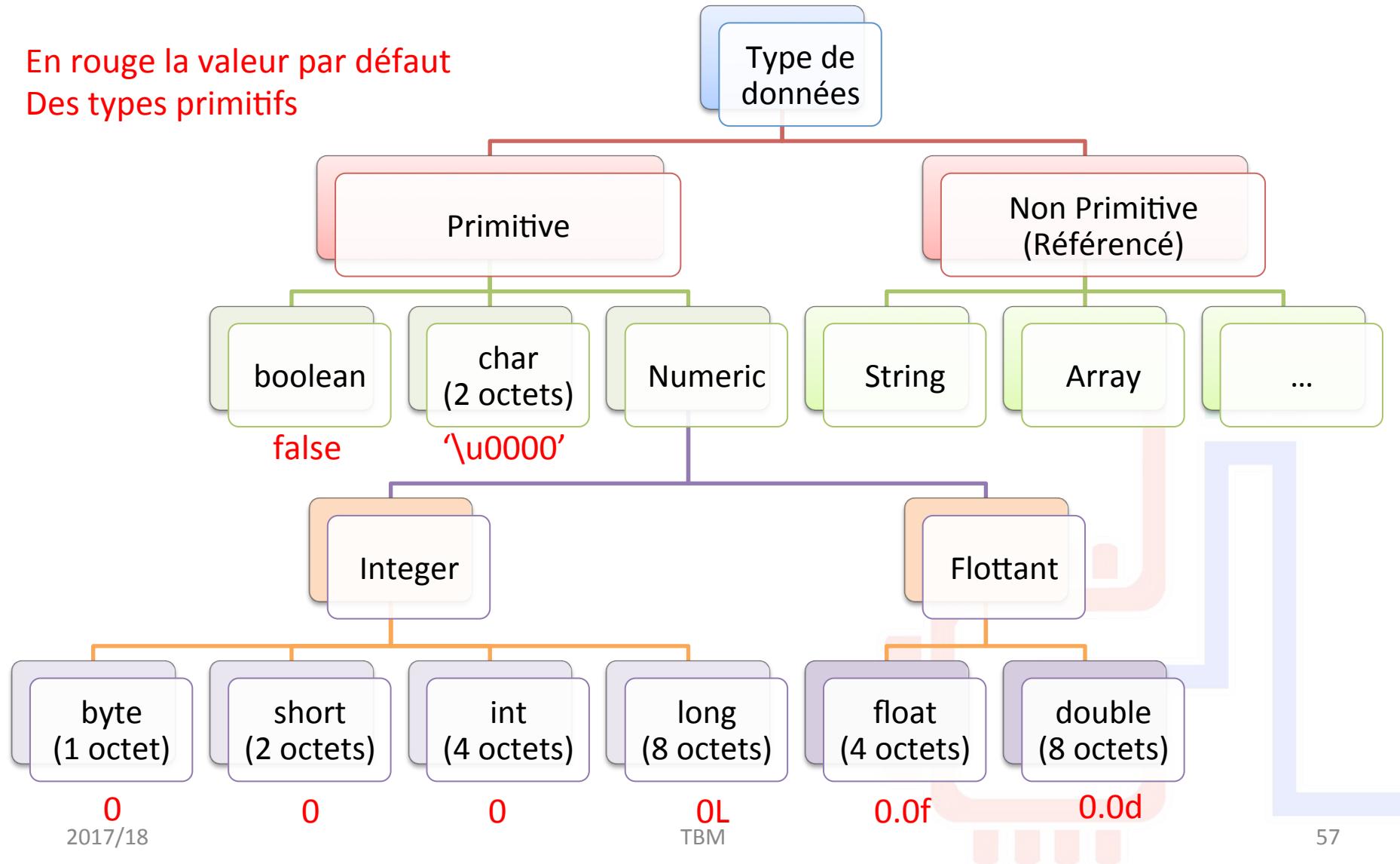
STRUCTURE GÉNÉRALE D'UN PROGRAMME JAVA

```
/* Début du fichier NouvelleClasse.java */
/* 1. Une éventuelle déclaration de package */
package nomPackage;
/* 2. Zéro ou plusieurs import */
import nomClasse; // Importer une classe sans package
import nomPackage.nomClasse; // Importer une classe d'un
package
import nomPackage.*; // Importer toutes les classes d'un
package

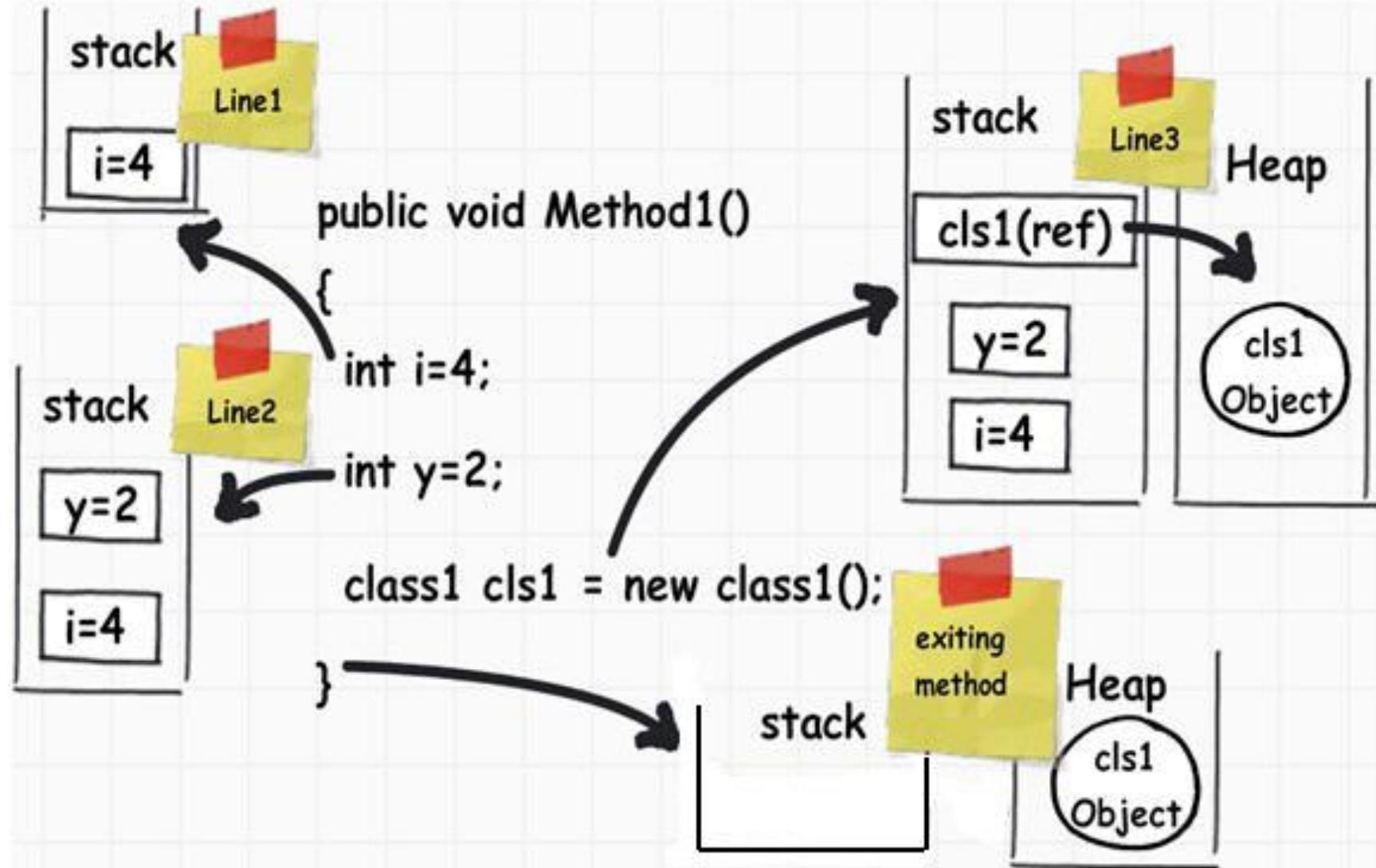
/* 3. Déclarations des classes du fichier */
// Une seule classe déclarée public et par convention qui
porte le même nom que le fichier
public class NouvelleClasse {
    // Corps de NouvelleClasse
}
class NouvelleClasse2 {
    // Corps de NouvelleClasse2
}
/* Fin du fichier NouvelleClasse.java */
```

VARIABLE : TYPE DE DONNÉES

En rouge la valeur par défaut
Des types primitifs



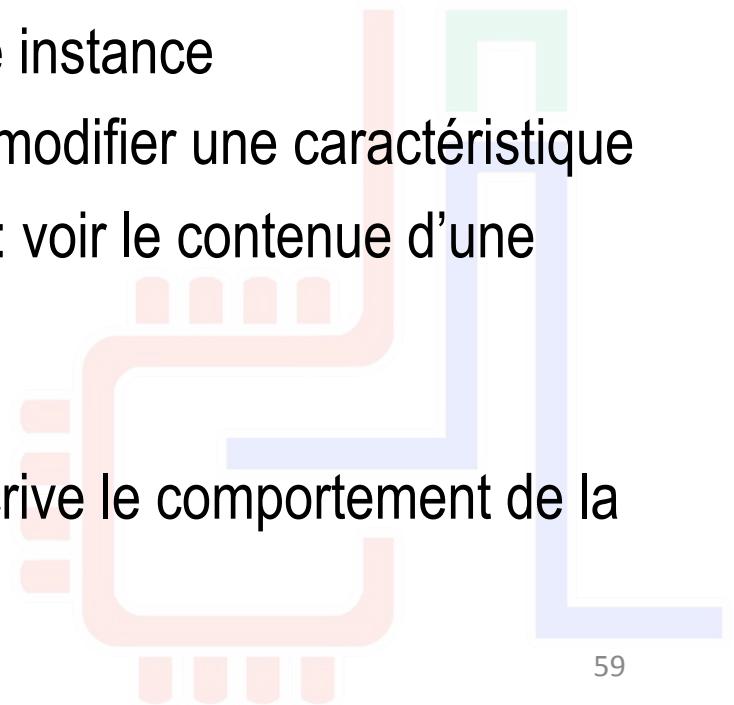
VARIABLE : PRIMITIVE VS. RÉFÉRENCE



RAPPEL : UNE CLASSE CONTIENT :



- Un **nom**
- Des **attributs**
 - Chaque attribut représente une caractéristique de la classe
 - Décrivent l'état de la classe
- Des **méthodes** :
 - **Constructeur(s)** : permet de créer une instance
 - **Modificateur** ou **setter** ou **mutateur** : modifier une caractéristique
 - **Observateur** ou **getter** ou **accesseur** : voir le contenu d'une caractéristique
 - **Destructeur** : détruire l'objet
 - **Méthodes d'implémentation** : qui décrive le comportement de la classe



DÉCLARATION DES VARIABLES (UN ATTRIBUT = VARIABLE)

■ Syntaxe :

```
Type nomVariable [= valeur];
```

■ Exemples

```
int age = 37;  
float prix;  
boolean enService = true;  
char direction = 'D';  
String reference = "EXD-074 984 A";  
Color background = new Color(255,0,0);  
Article monArticle = new Article();
```

■ Remarques

- CamelCase commence par une lettre minuscule
- Pas d'espace

CONSTANTES

■ Syntaxe :

```
final Type NOM_CONSTANTE [= valeur];
```

■ Exemples

```
final int STOCK_MAX; //il FAUT l'initialiser dans le constructeur  
final int MINIMUM_SIZE = 2;  
final float TEMPERATURE_LIMITE = 37.5f;  
final String CODE = "EXC034";
```

■ Remarques:

- Nom de constante en MAJUSCULE
- Si composé de 2 mots il faut les séparer par _
- Doit être initialiser au plus tard dans le constructeur

TABLEAU

■ Syntaxe :

```
Type[] nomTableau; ou Type nomTableau[]; // déclaration  
nomTableau = new Type[TAILLE] // création  
nomTableau[indice] = valeur // initialisation de la case  
ayant cet indice
```

■ Exemples

```
int[] monTableau = new int[3]; // déclaration et création d'un tableau de 3 entiers  
monTableau[0] = 1; // initialisation de la 1ère case ayant l'indice 0  
monTableau[1] = 37; // initialisation de la 2ème case ayant l'indice 1  
monTableau[2] = 166; // initialisation de la 3ème case ayant l'indice 2
```

```
int[] tab = {1,37,166,23,4, 6}; // déclaration, création et initialisation
```

```
Article[] stock = new Article[STOCK_MAX]; // déclaration et création d'un tableau d'Article  
stock[0] = new Article("XF0", 30.0f, 120); // remplissage de la case 0 par un article
```

■ Remarque

- Un objet tableau contient des attributs (ex. length qui contient la taille du tableau)

TABLEAU À 2 DIMENSIONS

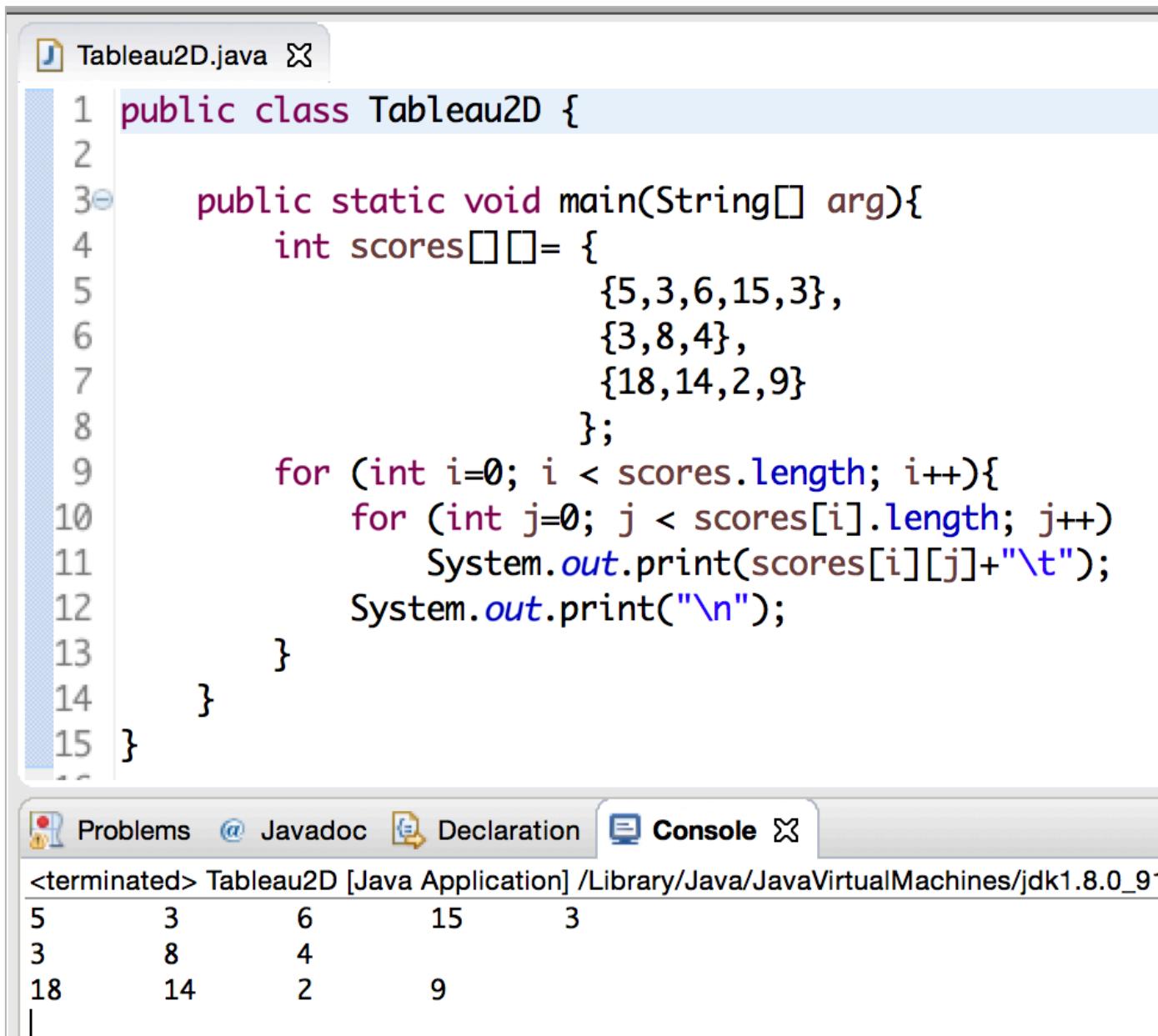
■ Syntaxe

```
Type[][] nomTableau; ou Type nomTableau[][]; //  
déclaration  
nomTableau = new Type[NB_L][NB_C] // création  
nomTableau[i][j] = valeur // initialisation de la case i è  
ligne j è colonne.
```

■ Exemple

```
int nombres[][] = {{0,2,4,6,8}, {1,3,5,7,9}};
```

TABLEAU À 2 DIMENSIONS : EXEMPLE



The screenshot shows a Java code editor with a file named "Tableau2D.java". The code defines a class "Tableau2D" with a main method that prints a 2D integer array. The array "scores" has 3 rows and 5 columns, containing the values [5, 3, 6, 15, 3], [3, 8, 4], and [18, 14, 2, 9]. The output is displayed in the "Console" tab, showing the array printed row by row.

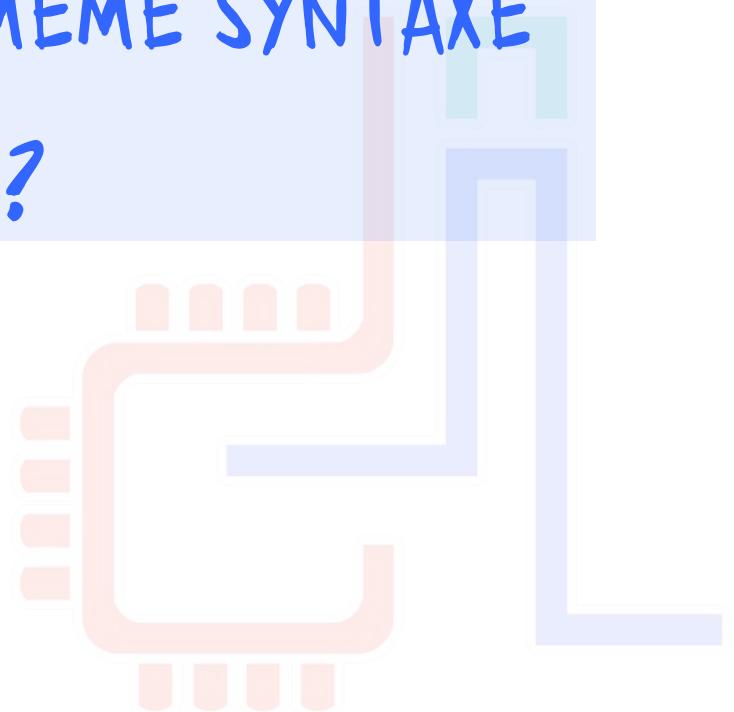
```
1 public class Tableau2D {  
2  
3     public static void main(String[] arg){  
4         int scores[][]= {  
5             {5,3,6,15,3},  
6             {3,8,4},  
7             {18,14,2,9}  
8         };  
9         for (int i=0; i < scores.length; i++){  
10             for (int j=0; j < scores[i].length; j++)  
11                 System.out.print(scores[i][j]+"\t");  
12             System.out.print("\n");  
13         }  
14     }  
15 }
```

Problems Javadoc Declaration Console

<terminated> Tableau2D [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91

5	3	6	15	3
3	8	4		
18	14	2	9	

AVEZ-VOUS REMARQUÉ DANS L'EXEMPLE
QUE LA BOUCLE FOR A LA MÊME SYNTAXE
QUE C/C++ ?



C'EST PAREIL POUR LES STRUCTURES

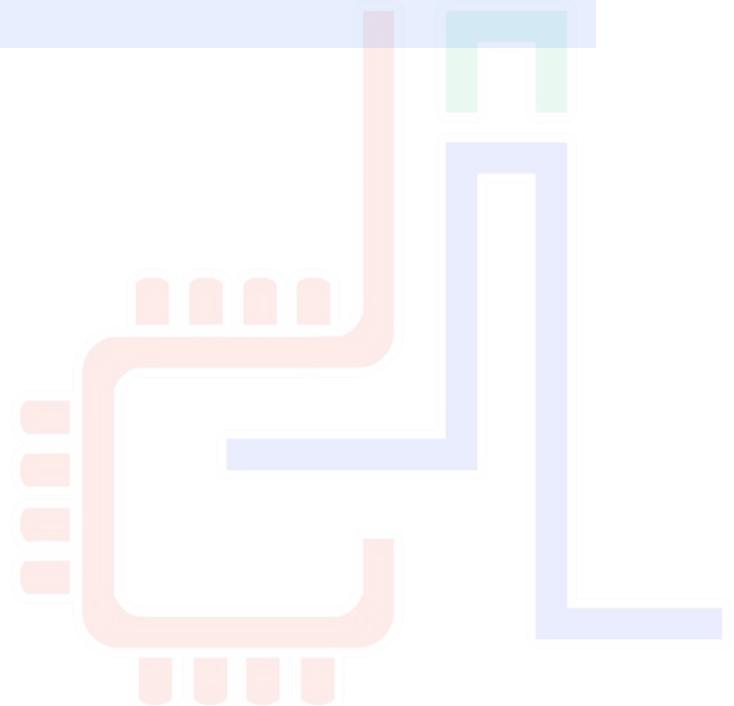
■ CONDITIONNELLES :

- Exécuter des instructions dans le cas où une condition est vraie, et exécuter d'autres instructions dans le cas contraire
 - if (même syntaxe qu'en C/C++)
 - switch-case (même syntaxe qu'en C/C++)

■ ITÉRATIVES :

- Le traitement itératif est utilisé pour exécuter une ou plusieurs instructions plusieurs fois
 - for (même syntaxe qu'en C/C++)
 - while (même syntaxe qu'en C/C++)
 - do-while (même syntaxe qu'en C/C++)

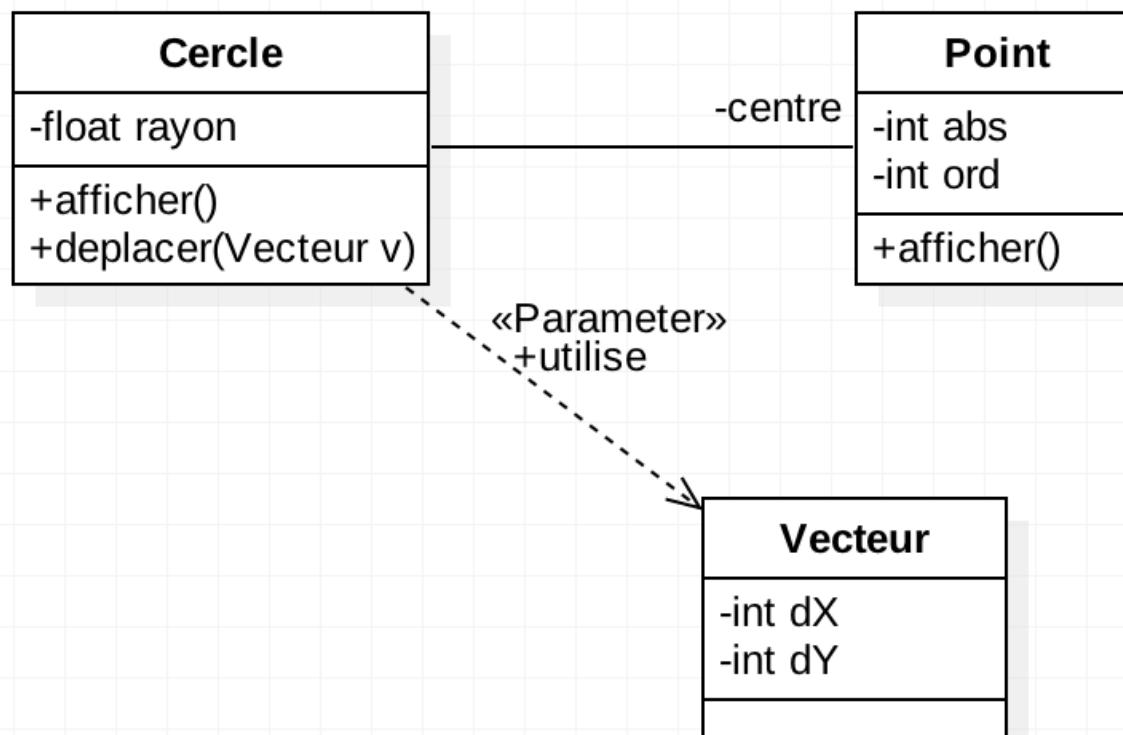
CLASSES ET OBJETS



ETUDE DE CAS VI.0

Créer et afficher deux cercles C1 et C2. Un cercle est caractérisé par un centre, un rayon. Nous souhaitons les déplacer selon des vecteurs

UNE CONCEPTION S'IMPOSE COMME D'HABITUDE



CLASSE : CERCLE

```
1 package geometrie;
2
3 public class Cercle {
4     private Point centre; // attribut privé centre de type point
5     private float rayon; // attribut privé rayon de type réel
6
7     Cercle(Point centre, float rayon){ // constructeur qui permet d'initialiser les attributs
8         this.centre = centre;
9         this.rayon = rayon;
10    }
11    public Point getCentre() { // getter de l'attribut centre
12        return centre;
13    }
14    public void setCentre(Point centre) { // setter de l'attribut centre
15        this.centre = centre;
16    }
17    public float getRayon() { //getter de l'attribut rayon
18        return rayon;
19    }
20    public void setRayon(float rayon) { //setter de l'attribut rayon
21        this.rayon = rayon;
22    }
23    public void afficher(){ // méthode qui affiche le centre et le rayon
24        System.out.print("le centre est ");
25        this.centre.afficher(); // le cercle demande à son centre de s'afficher
26        System.out.println("le rayon est "+this.rayon);
27    }
28    public void deplacer(Vecteur v){ // méthode qui permet de changer les coordonnées du centre
29        centre.setAbs(centre.getAbs() + v.getdX());
30        centre.setOrd(centre.getOrd() + v.getdY());
31    }
32 }
```

CLASSE : POINT

```
1 package geometrie;
2
3 public class Point { // début de la classe Point
4     private int abs; // attribut représentant l'abscisse du point
5     private int ord; // attribut représentant l'ordonnée du point
6
7     Point(){ // 1er constructeur permettant d'initialiser à 0 l'abs et l'ord
8         abs = 0;
9         ord = 0;
10    }
11    Point(int abs, int ord){ // 2e constructeur pour initialiser les attributs avec des paramètres
12        this.abs = abs;
13        this.ord = ord;
14    }
15    public int getAbs() { // getter de l'attribut abs
16        return abs;
17    }
18    public void setAbs(int abs) { // setter de l'attribut abs
19        this.abs = abs;
20    }
21    public int getOrd() { // getter de l'attribut ord
22        return ord;
23    }
24    public void setOrd(int ord) { // setter de l'attribut ord
25        this.ord = ord;
26    }
27    public void afficher(){ // méthode d'affichage d'un point
28        System.out.print("[ "+abs+", "+ord+"]");
29    }
30 } // fin de classe point
```

CLASSE : VECTEUR

```
1 package geometrie;
2
3 public class Vecteur { // Début de la classe Vecteur
4     private int dX; // attribut représentant le déplacement selon l'axe des X
5     private int dY; // attribut représentant le déplacement selon l'axe des Y
6
7     Vecteur(int dX, int dY){ // constructeur pour initialiser les attributs avec des paramètres
8         this.dX = dX;
9         this.dY = dY;
10    }
11    public int getdX() { // getter de l'attribut dX
12        return dX;
13    }
14    public void setdX(int dX) { // setter de l'attribut dX
15        this.dX = dX;
16    }
17    public int getdY() { // getter de l'attribut dY
18        return dY;
19    }
20    public void setdY(int dY) { // setter de l'attribut dY
21        this.dY = dY;
22    }
23
24    /* remarque */
25    // pas de méthode d'implémentation dans notre cas
26
27 } // Fin de la classe Vecteur
28
```

CLASSE : PRINCIPALE

```
1 package geometrie;
2
3 public class Principale {
4
5     public static void main(String[] args) {
6         // instantiation des deux cercles c1 et C2
7         Cercle c1 = new Cercle(new Point(3,5), 10.0f);
8         Cercle c2 = new Cercle(new Point(50,30), 5.7f);
9
10        // Création d'un vecteur de déplacement
11        Vecteur deplacement = new Vecteur(121, 345);
12        // affichage de c1 et c2 avant déplacement
13        c1.afficher();
14        c2.afficher();
15        // déplacement des cercles
16        c1.deplacer(deplacement); // avec le vecteur déjà créé
17        c2.deplacer(new Vecteur(356, 234)); // en passant un nouveau Vecteur en paramètre
18        // affichage de c1 et c2 après déplacement
19        c1.afficher();
20        c2.afficher();
21    }
22
23 }
```

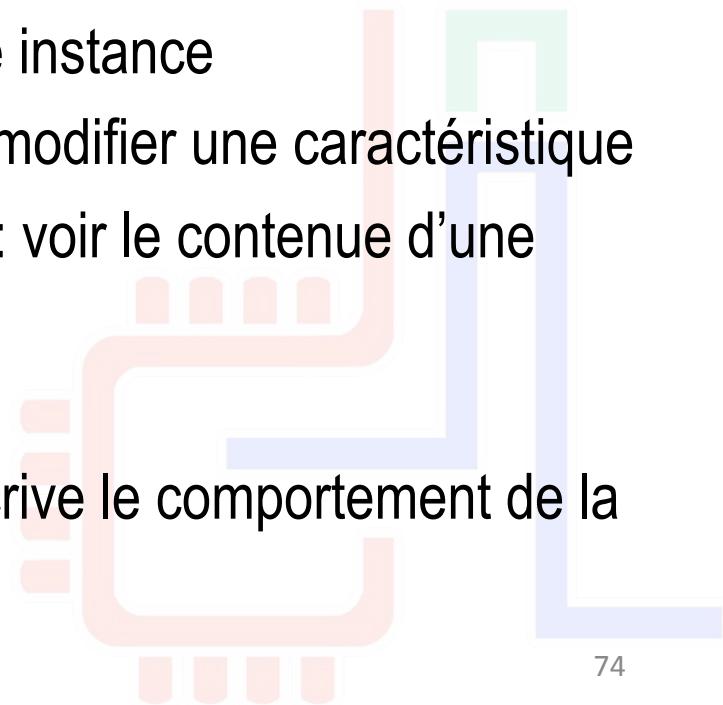
Problems Javadoc Declaration Console

```
<terminated> Principale (4) [Java Application] /library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java (23 sept. 2017 17:43:16)
le centre est [3, 5]le rayon est 10.0
le centre est [50, 30]le rayon est 5.7
le centre est [124, 350]le rayon est 10.0
le centre est [406, 264]le rayon est 5.7
```

RAPPEL : UNE CLASSE CONTIENT :



- Un **nom**
- Des **attributs**
 - Chaque attribut représente une caractéristique de la classe
 - Décrivent l'état de la classe
- Des **méthodes** :
 - **Constructeur(s)** : permet de créer une instance
 - **Modificateur** ou **setter** ou **mutateur** : modifier une caractéristique
 - **Observateur** ou **getter** ou **accesseur** : voir le contenu d'une caractéristique
 - **Destructeur** : détruire l'objet
 - **Méthodes d'implémentation** : qui décrive le comportement de la classe



CLASSE : NOTION DE CONSTRUCTEUR

- permet d'instancier des objets
- Le mot-clé **this** permet de désigner l'objet courant et d'accéder aux attributs et méthodes

```
public class Cercle {  
    private Point centre;  
    private int rayon;  
  
    Cercle (){  
        this.centre = new Point(3,5);  
        this.rayon = 10;  
    }  
    Cercle (Point centre, int rayon){  
        this.centre = centre;  
        this.rayon = rayon;  
    }  
}
```

- Pour créer un objet on utilise l'opérateur **new**

```
public static void main(String[] args) {  
    Cercle C1 = new Cercle(new Point(3,5), 10);  
}
```

CLASSE : NOTION DE GETTER / SETTER

- Getter permet d'observer la valeur d'un attribut privé

```
public TypeAttribut getNomAttribut () {  
    return (this.nomAttribut);  
}
```

- Setter permet de modifier la valeur d'un attribut privé

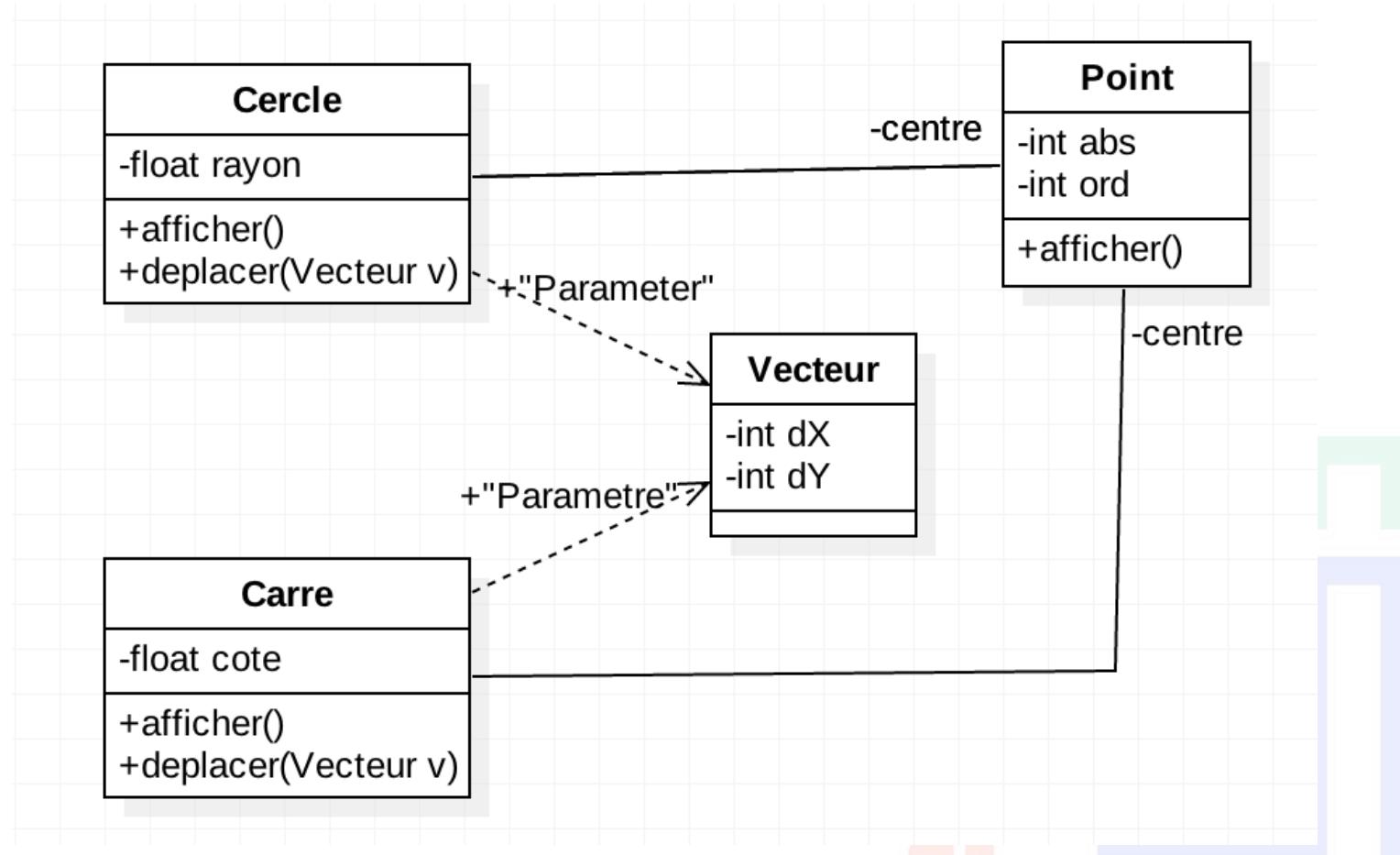
```
public void setNomAttribut (TypeAttribut a) {  
    this.nomAttribut = a;  
}
```

```
public void setCentre(Point centre){  
    this.centre = centre;  
}  
  
public Point getCentre(){  
    return this.centre;  
}
```

ETUDE DE CAS V2.0

Créer et afficher deux cercles C1 et C2 et deux carrés Ca1 et Ca2. Un cercle est caractérisé par un centre, un rayon. Un carré est caractérisé par un centre et un côté. Nous souhaitons les déplacer selon des vecteurs

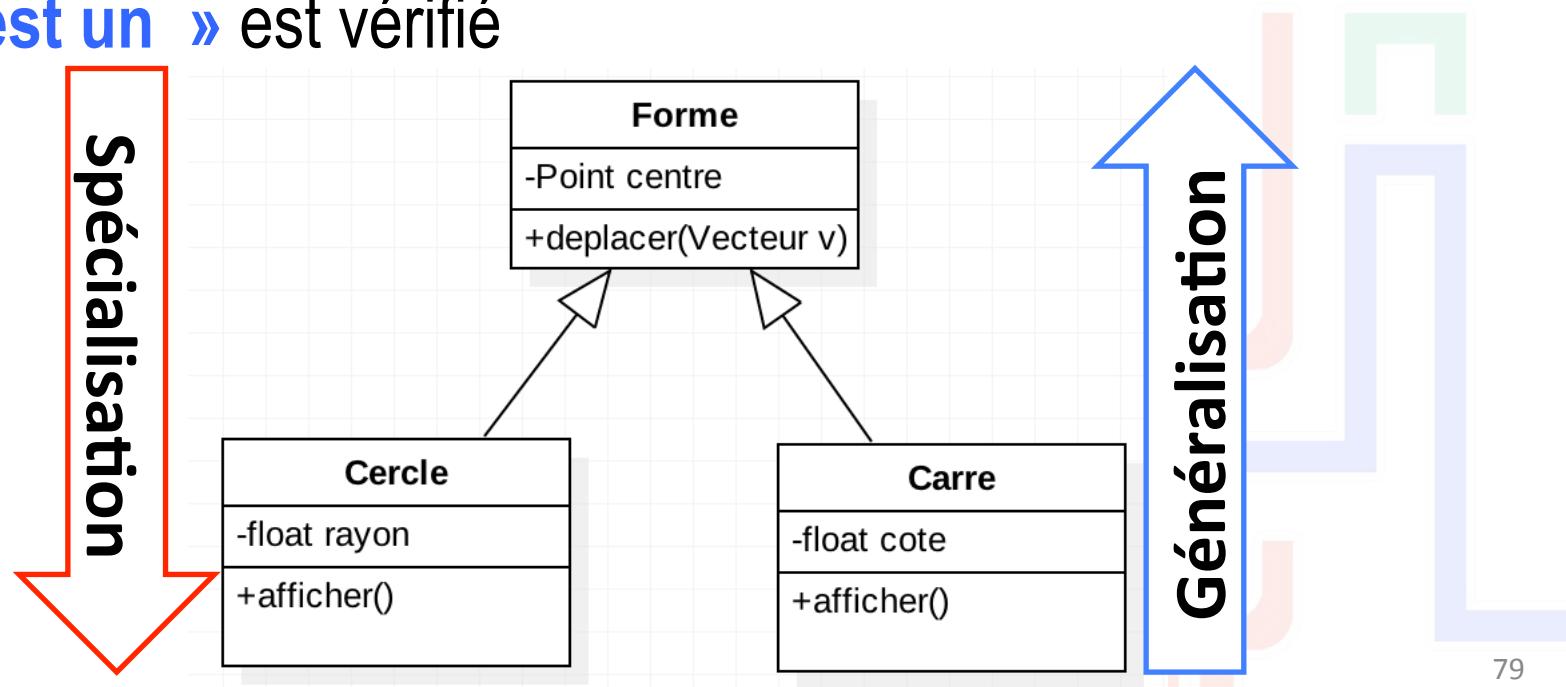
UNE CONCEPTION S'IMPOSE COMME D'HABITUDE



QU'EST CE QUE VOUS REMARQUEZ ?

POO : NOTION D'HÉRITAGE

- Quand plusieurs classes ont des caractéristiques similaires. Elles seront modélisées par une **Classe Générique** qui contiendra les caractéristiques communes et auront une relation d'héritage avec cette classe
- L'héritage ne peut avoir lieu que si le sens sémantique « **est un** » est vérifié



POO : NOTION D'HÉRITAGE

Généralisation

Construction d'une classe (classe mère) à partir d'une (ou plusieurs) autres classes (filles) en regroupant et partageant les attributs et méthodes communs

Spécialisation

enrichissement : les classes filles héritent les attributs et les méthodes de la classe mère **en y ajoutant des attributs et/ou des méthodes spécifiques au classes filles.**

HÉRITAGE : POURQUOI ET QUAND ?

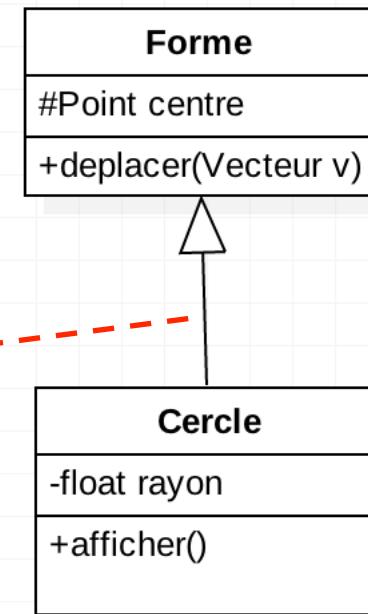
- L'héritage permet de
 - Regrouper les champs et les méthodes communs à plusieurs classes.
 - Minimiser l'écriture du code (exemple méthode déplacer écrite une fois dans Forme et utiliser avec le Cercle et le Carré.)

- On fait un héritage quand
 - une classe (fille) étend les fonctionnalités d'une autre classe (mère)
 - une classe (fille) adapte le fonctionnement d'une autre classe (mère) à une situation particulière.

L'HÉRITAGE AVEC JAVA AVEC LE MOT EXTENDS

```
public class Forme {  
    protected Point centre;  
  
    public Forme(Point centre) { <-----  
        this.centre = centre;  
    }  
  
    public void deplacer(Vecteur v){  
        centre.setAbs(centre.getAbs() + v.getdX());  
        this.centre.setOrd(this.centre.getOrd() + v.getdY());  
    }  
}  
  
class Cercle extends Forme {  
    private int rayon;  
  
    Cercle (Point centre, int rayon){  
        super(centre); ←  
        this.rayon = rayon;  
    }  
}
```

super : appel du
constructeur de la classe
mère



LE CODE DE L'ÉTUDE DE CAS 2.0

```
Forme.java Cercle.java Carre.java
1 package geometrie;
2
3 public class Forme {
4     protected Point centre;
5
6     public Forme(Point centre) {
7         this.centre = centre;
8     }
9
10    public void setCentre(Point centre){
11        this.centre = centre;
12    }
13
14    public Point getCentre(){
15        return this.centre;
16    }
17
18    public void deplacer(Vecteur v){
19        centre.setAbs(centre.getAbs() + v.getdX());
20        this.centre.setOrd(this.centre.getOrd() + v.getdY());
21    }
22 }
```

CLASSE FORME

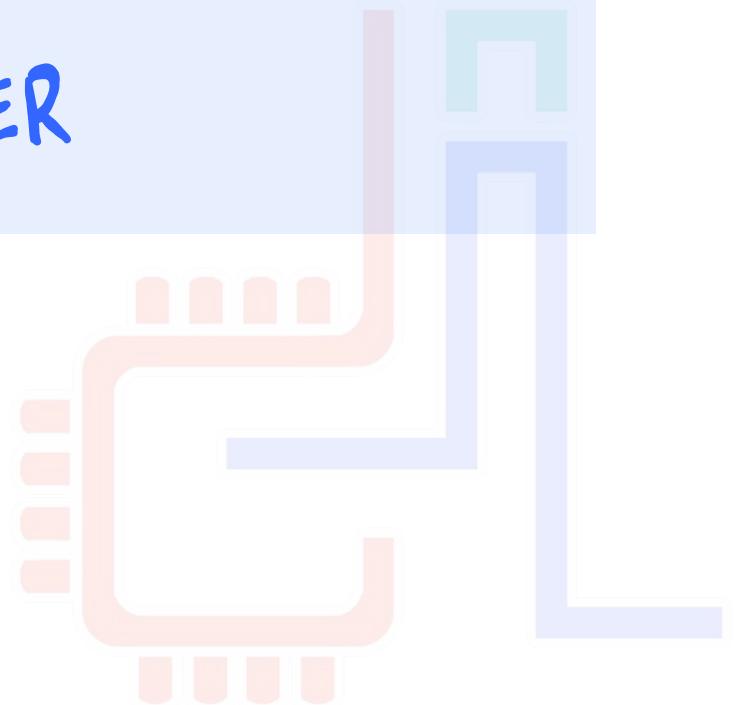
LE CODE DE L'ÉTUDE DE CAS 2.0

```
Forme.java Cercle.java Carré.java
1 package geometrie;
2
3 public class Cercle extends Forme { CLASSE CERCLE
4     private float rayon;
5
6     public Cercle(Point centre, float rayon) {
7         super(centre);
8         this.rayon = rayon;
9     }
10
11    public float getRayon() {
12        return rayon;
13    }
14
15    public void setRayon(float rayon) {
16        this.rayon = rayon;
17    }
18
19    public void afficher(){
20        System.out.print("le centre est ");
21        centre.afficher();
22        System.out.println("le rayon est "+rayon);
23    }
24}
```

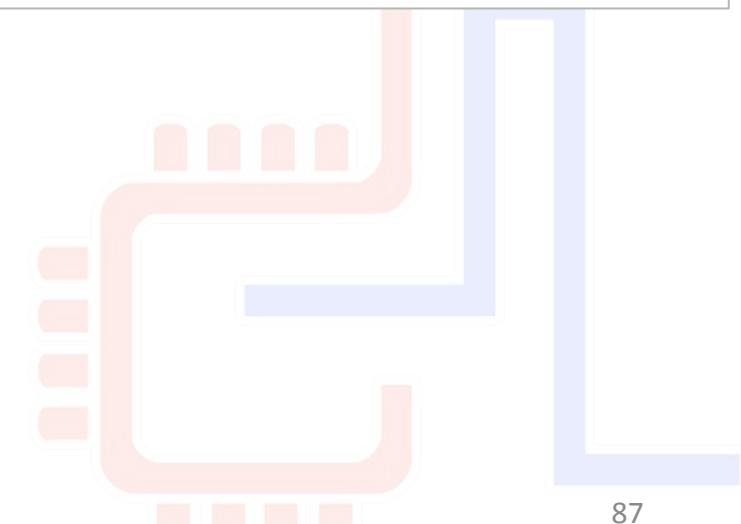
LE CODE DE L'ÉTUDE DE CAS 2.0

```
Forme.java Cercle.java Carre.java
1 package geometrie;
2
3 public class Carre extends Forme {      CLASSE CARRÉ
4     private int cote;
5
6
7     public Carre(Point centre, int cote) {
8         super(centre);
9         this.cote = cote;
10    }
11
12    public int getCote() {
13        return cote;
14    }
15
16    public void setCote(int cote) {
17        this.cote = cote;
18    }
19
20    public void afficher(){
21        System.out.print("le centre est ");
22        centre.afficher();
23        System.out.println("le coté est "+cote);
24    }
25}
```

EXEMPLE POUR MIEUX COMPRENDRE
THIS ET SUPER



QU'AFFICHE CE PROGRAMME ?



```
Principale.java Classe1.java Classe2.java
1 package geometrie;
2
3 public class Classe1 {
4     private int i;
5     Classe1(){
6         System.out.println("Classe1");
7     }
8     Classe1(int valeur){
9         this();
10    i = valeur;
11    System.out.println(this.i);
12 }
13 }

public class Principale {
    public static void main(String[] args) {
        Classe1 o1 = new Classe1();
        System.out.println("-----");
        Classe1 o2 = new Classe1(3);
        System.out.println("-----");
        Classe2 o3 = new Classe2();
        System.out.println("-----");
        Classe2 o4 = new Classe2(2);
    }
}
```

1 package geometrie;
2
3 public class Classe1 {
4 private int i;
5 Classe1(){
6 System.out.println("Classe1");
7 }
8 Classe1(int valeur){
9 this();
10 i = valeur;
11 System.out.println(this.i);
12 }
13 }

1 package geometrie;
2
3 public class Classe2 extends Classe1{
4 Classe2(){
5 super(5);
6 System.out.println("Classe2");
7 }
8
9 Classe2(int valeur){
10 System.out.println(valeur);
11 }
12 }
13 }

RÉPONSE

The screenshot shows an IDE interface with two code editors and a console window.

Code Editor 1 (Principale.java):

```
1 package geometrie;
2
3 public class Classe1 {
4     private int i;
5     Classe1(){
6         System.out.println("Classe1");
7     }
8     Classe1(int valeur){
9         this();
10    i = valeur;
11    System.out.println(this.i);
12 }
13 }
```

Code Editor 2 (Classe2.java):

```
1 package geometrie;
2
3 public class Classe2 extends Classe1{
4     Classe2(){
5         super(5);
6         System.out.println("Classe2");
7     }
8
9     Classe2(int valeur){
10        System.out.println(valeur);
11    }
12 }
13 }
```

Console Output:

```
Console Problems
<terminated> Principale (4) [Java]
Classe1
-----
Classe1
3
-----
Classe1
5
Classe2
-----
Classe1
2
```

88

LE MOT CLÉ SUPER

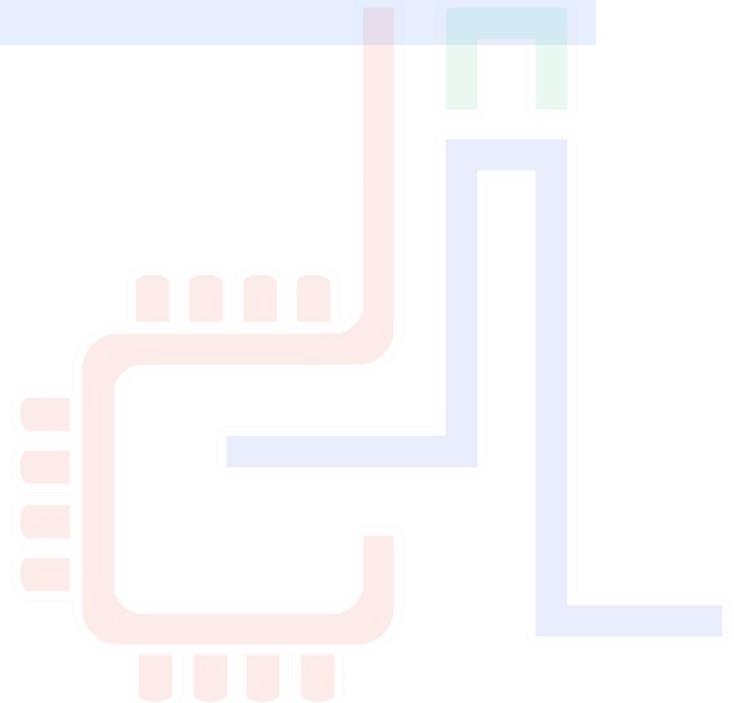
■ Super

- sert à accéder aux propriétés (attribut et/ou méthodes) de classe mère au niveau de la classe fille.
- Ce mot clé permet d'utiliser le constructeur de la classe mère dans la classe fille.

■ Remarques Importantes:

- L'appel du constructeur de la classe mère avec **super()** doit être fait à la première instruction du constructeur de la classe fille.
- En cas d'omission le système appellera comme première instruction par défaut **super()** lors de l'exécution du constructeur de la classe fille

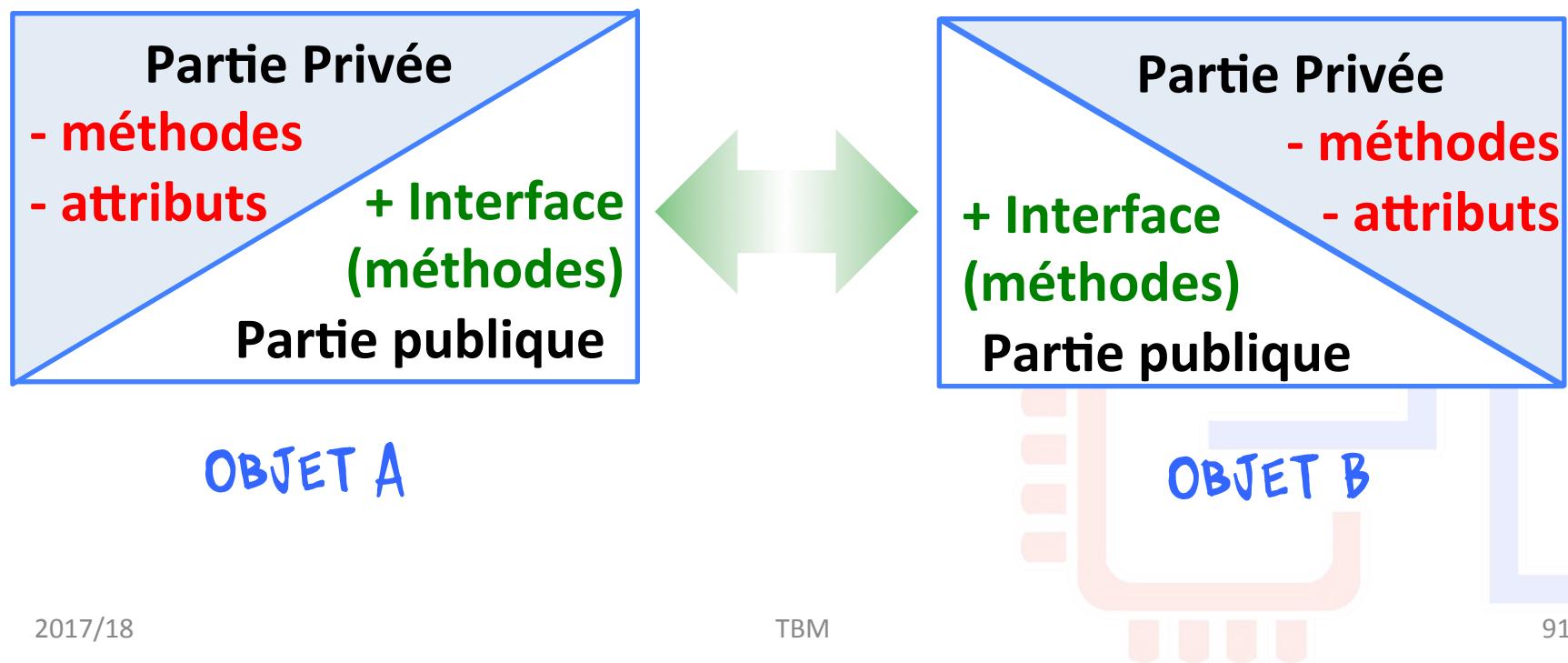
RETOUR SUR L'ENCAPSULATION



RAPPEL ET NOTION D'ENCAPSULATION

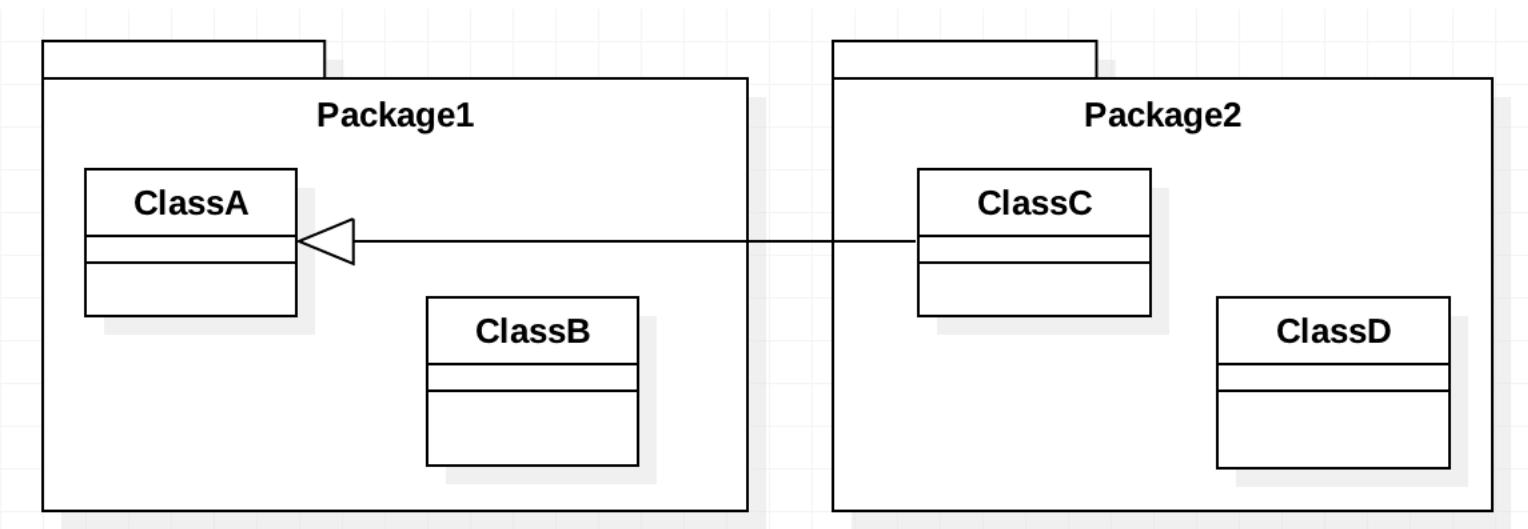
■ Encapsulation des données

- un objet ne doit pas exposer sa représentation interne au monde extérieur.
- Un objet est complètement défini par son **interface** pour les autres objets



ENCAPSULATION DE CLASSES

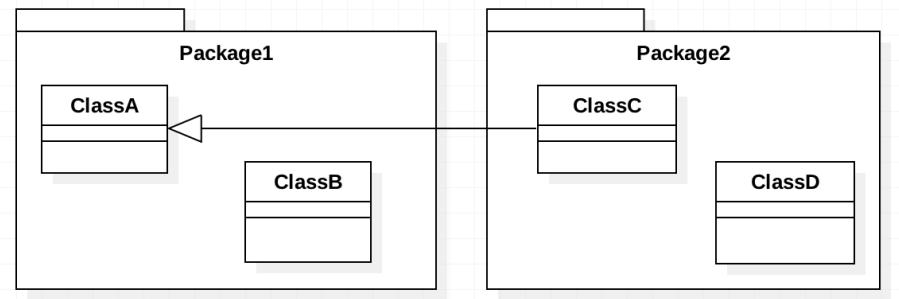
- Soit le diagramme de classes suivant :



ENCAPSULATION DES CLASSES

■ ClassA est **public**

- Elle est visible depuis n'importe quelle classe du projet



```
ClassA.java
1 package package1;
2
3 public class ClassA {
4
5 }
```

```
ClassC.java
1 package package2;
2
3 import package1.ClassA;
4
5 public class ClassC extends ClassA{
6     ClassA a;
7 }
```

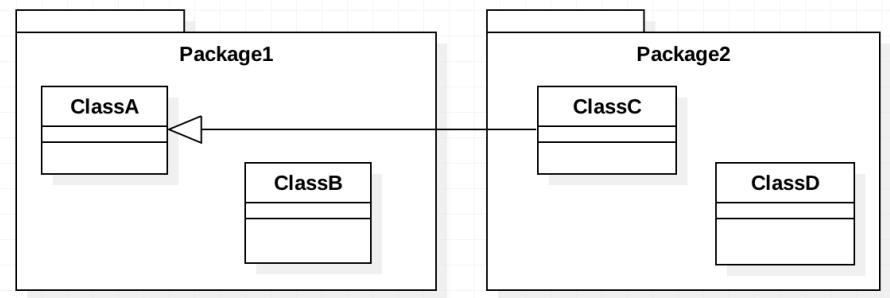
```
ClassB.java
1 package package1;
2
3 public class ClassB {
4     ClassA a;
5 }
```

```
ClassD.java
1 package package2;
2
3 import package1.ClassA;
4
5 public class ClassD {
6     ClassA a;
7 } TBM
```

2017/18

ENCAPSULATION DES CLASSES

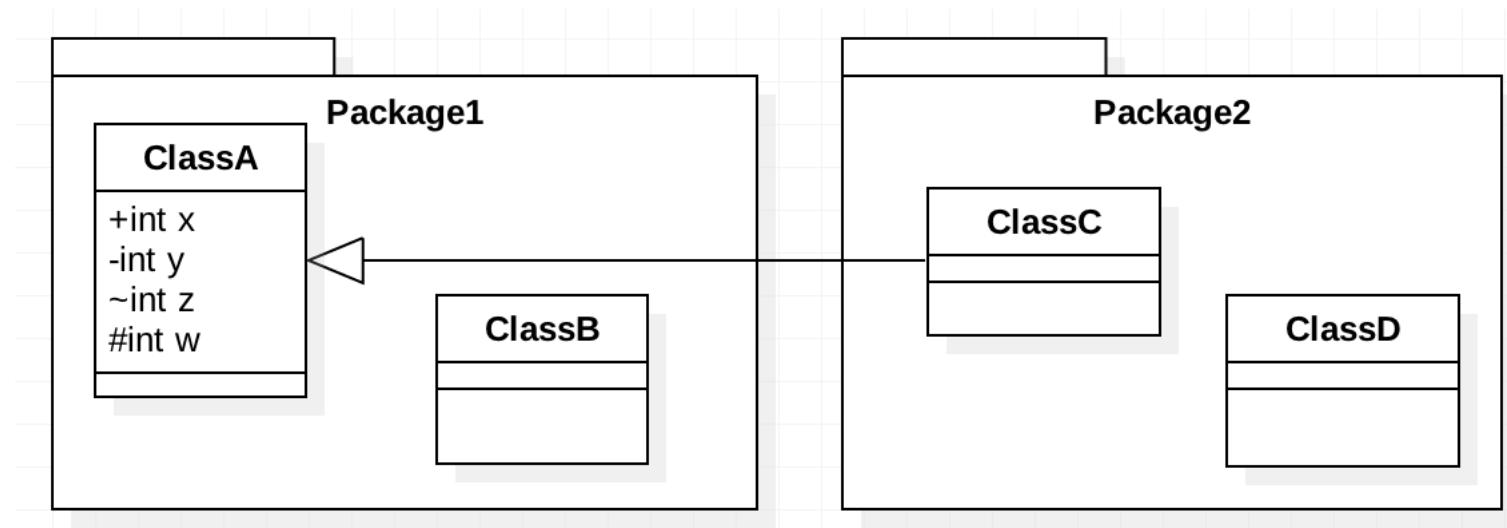
- ClassA est **package**
 - Elle est visible seulement par les classes de son package



```
ClassA.java: package package1;  
class ClassA {  
}  
ClassB.java: package package1;  
public class ClassB {  
    ClassA a;  
}  
ClassC.java: package package2;  
public class ClassC extends ClassA{  
    ClassA a;  
}  
ClassD.java: package package2;  
import package1.ClassA;  
public class ClassD {  
    ClassA a;  
}
```

ENCAPSULATION DES ATTRIBUTS

- Soit le diagramme de classes suivant :



ENCAPSULATION DES ATTRIBUTS

```
ClassA.java
1 package package1;
2
3 public class ClassA {
4     public int x;
5     private int y;
6     int z;
7     protected int w;
8 }
9
10

ClassB.java
1 package package1;
2
3 public class ClassB {
4     ClassA a = new ClassA();
5     void m(){
6         a.x = 1;
7         a.y = 1;
8         a.z = 1;
9         a.w = 1;
10    }
11 }
12
13

ClassC.java
1 package package2;
2 import package1.ClassA;
3
4 public class ClassC extends ClassA{
5     ClassA a = new ClassA();
6     void m(){
7         a.x = 1;
8         a.y = 1;
9         a.z = 1;
10        a.w = 1;
11        this.w = 1;
12    }
13 }

ClassD.java
1 package package2;
2 import package1.ClassA;
3
4 public class ClassD {
5     ClassA a = new ClassA();
6     void m(){
7         a.x = 1;
8         a.y = 1;
9         a.z = 1;
10        a.w = 1;
11        this.w = 1;
12    }
13 }
```

ENCAPSULATION DES ATTRIBUTS

public (+)

La variable *public* est visible par toutes les classes

private (-)

La variable *private* n'est accessible que depuis l'intérieur même de la classe.

package (~)

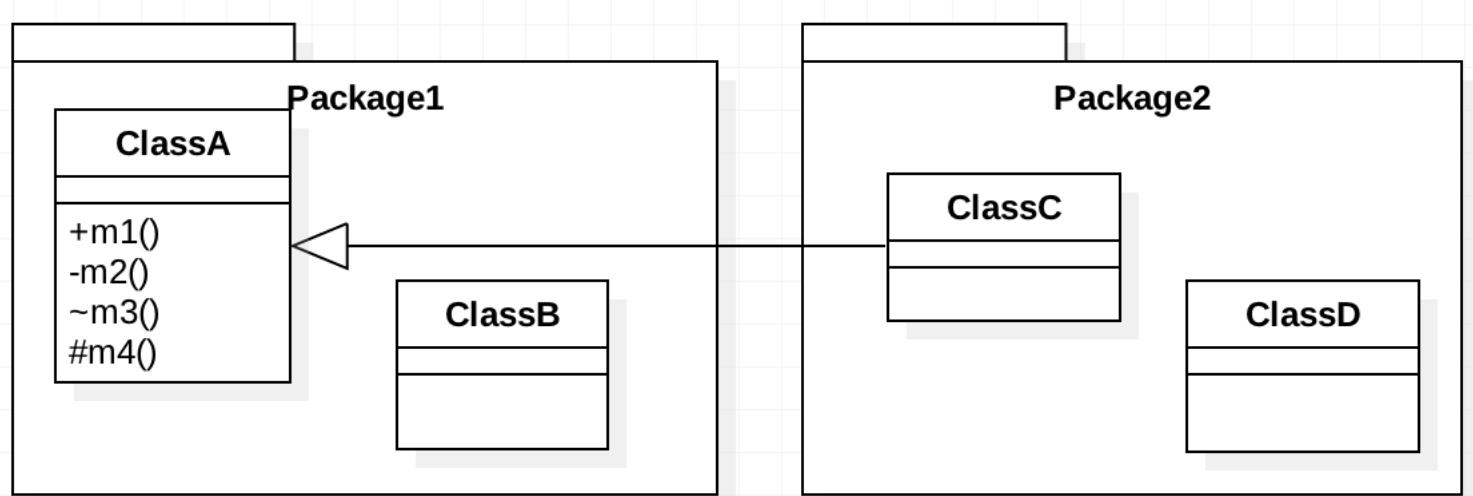
La variable *package* n'est accessible que depuis les classes faisant partie du *même package*.

protected (#)

La variable *protected* est accessible uniquement aux classes d'un *même package* et à *ses sous-classes* (même si elles sont définies dans un package différent.)

ENCAPSULATION DES MÉTHODES

- Soit le diagramme de classes suivant :



ENCAPSULATION DES MÉTHODES

```
ClassA.java
1 package package1;
2
3 public class ClassA {
4     public void m1(){}
5     private void m2(){}
6     void m3(){}
7     protected void m4(){}
8
9 }
10

ClassB.java
1 package package1;
2
3 public class ClassB {
4     ClassA a = new ClassA();
5     void m(){
6         a.m1();
7         a.m2();
8         a.m3();
9         a.m4();
10    }
11 }

ClassC.java
1 package package2;
2 import package1.ClassA;
3
4 public class ClassC extends ClassA{
5     ClassA a = new ClassA();
6     void m(){
7         a.m1();
8         a.m2();
9         a.m3();
10        a.m4();
11        this.m4();
12    }
13 }

ClassD.java
1 package package2;
2 import package1.ClassA;
3
4 public class ClassD {
5     ClassA a = new ClassA();
6     void m(){
7         a.m1();
8         a.m2();
9         a.m3();
10        a.m4();
11        this.m4();
12    }
13 }
```

ENCAPSULATION DES MÉTHODES

public (+)

La méthode *public* est visible par toutes les classes

private (-)

La méthode *private* n'est accessible que depuis l'intérieur même de la classe.

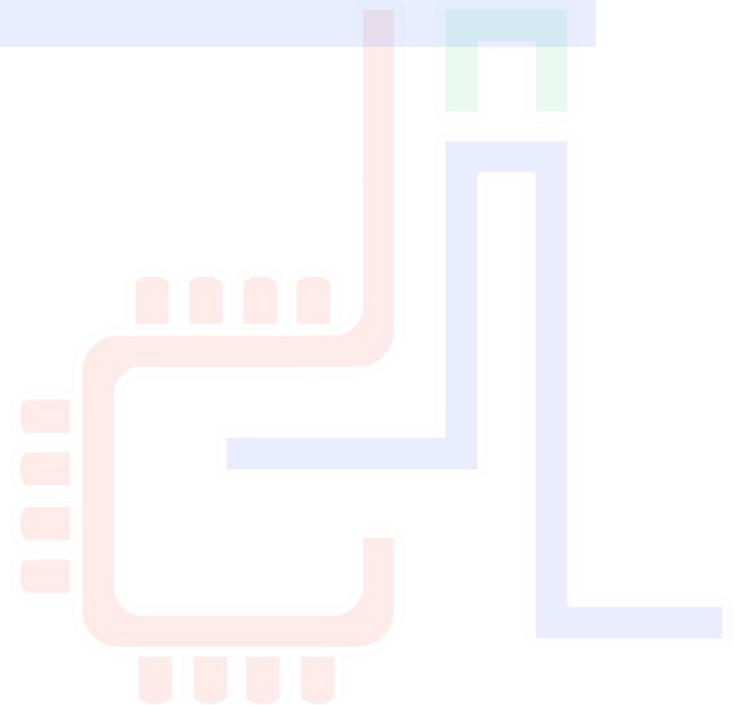
package (~)

La méthode *package* n'est accessible que depuis les classes faisant partie du même *package*.

protected (#)

La méthode *protected* est accessible uniquement aux classes d'un *même package* et à *ses sous-classes* (même si elles sont définies dans un package différent.)

APRÈS LES ABC, LES DEF DE JAVA



LA CLASSE STRING

- Offre des objets chaînes de caractères **constantes**

```
String chaine1 = new String("J'aime Java");
String chaine2 = "Ceci est une chaîne de caractère";
```

- méthodes utiles de la classe **String**:

- **static** String valueOf(**int** i); //renvoie une chaîne qui représente l'entier **i**
- **static** String valueOf(**boolean** b); //renvoie une chaîne qui représente le booléen **b**
- **static** String valueOf(**type** x); //renvoie une chaîne qui représente **x** de type primitif **type**.

- Rq: Les méthodes ne portent sur aucun objet (mot clé **static**). Pour les utiliser, on les préfixe par le nom de la classe :

- String.valueOf(1234); //retourne la chaîne "1234".

LA CLASSE STRING

- **boolean equals (String s);** /* teste l'égalité entre **this** et s. */
- **String concat (String s);** /* rend la concaténation de **this** et s. */
- **int length ();** //rend la longueur de **this**.
- **int indexOf (int c);** /* position de la première occurrence du caractère de code ASCII c, -1 si le caractère n'apparaît pas. */
- **char charAt (int i);** /* caractère en position i (numérotés à partir de 0) */
- **s1+s2 :** concaténation de **s1** et **s2**.

LA MÉTHODE TOSTRING()

- est définie dans la classe **Object** renvoie le nom de la classe de l'objet concerné suivi de l'adresse de cet objet. Pour changer l'affichage on la redéfini

```
Personne1.java
1 package geometrie;
2
3 public class Personne1 {
4     private String nom;
5     private int age;
6
7     public Personne1(String nom, int age) {
8         this.nom = nom;
9         this.age = age;
10    }
11 }
12

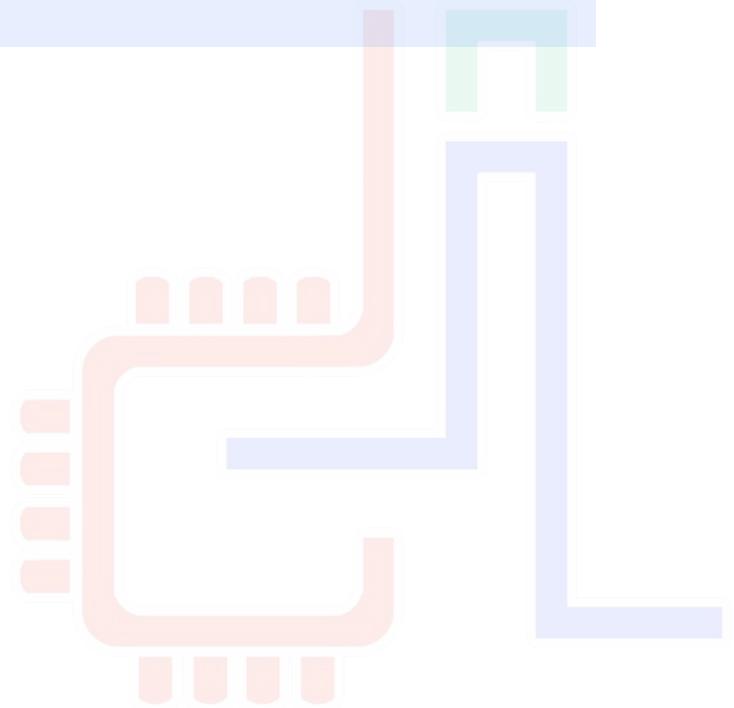
Personne2.java
1 package geometrie;
2
3 public class Personne2 {
4     private String nom;
5     private int age;
6
7     public Personne2(String nom, int age) {
8         this.nom = nom;
9         this.age = age;
10    }
11
12     public String toString(){
13         return "Je suis "+nom+" mon age est "
14             +age;
15    }
16 }

Principale.java
1 package geometrie;
2 public class Principale {
3
4     public static void main(String[] args) {
5         Personne1 p1 = new Personne1("Tarek", 37);
6         Personne2 p2 = new Personne2("Néjib", 80);
7         System.out.println(p1);
8         System.out.println(p2);
9     }
10 }
```

Console

```
<terminated> Principale (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java (16 oct. 2017 20:38:03)
201 geometrie.Personne1@7852e922
Je suis Néjib mon age est 80
```

POLYMORPHISME



DÉFINITION

- Polymorphisme (poly : plusieurs, morph : forme) est un des concepts essentiels de la POO et le moyen d'écrire les méthodes d'objets sous plusieurs formes. L'objet devient ainsi polymorphe.
- Une méthode s'écrit :

```
entête de la méthode {  
    // corps de la méthode  
}
```

- 3 types de polymorphisme
 - **Surcharge** : écrire une même méthode de plusieurs forme en modifiant son **entête**
 - **Redéfinition** : écrire une même méthode de plusieurs forme en modifiant son **corps**
 - **Surclassement** : pouvoir percevoir un objet en tant qu'instance de classes variées, selon les besoins en invoquant de plusieurs formes le **constructeur** de cette classe.

POLYMPHISME : SURCHARGE

- Définir des méthodes ayant le même nom mais pas la même signature

```
3 public class Personne1 {  
4     private String nom;  
5     private int age;  
6  
7     public void manger(){}  
8     public void manger(int t){}  
9     public void manger(char c){}  
10    public void manger(int i, int j) {}  
11    public void manger(int i, char c) {}  
12    public void manger(char c, int i) {}  
13 }
```

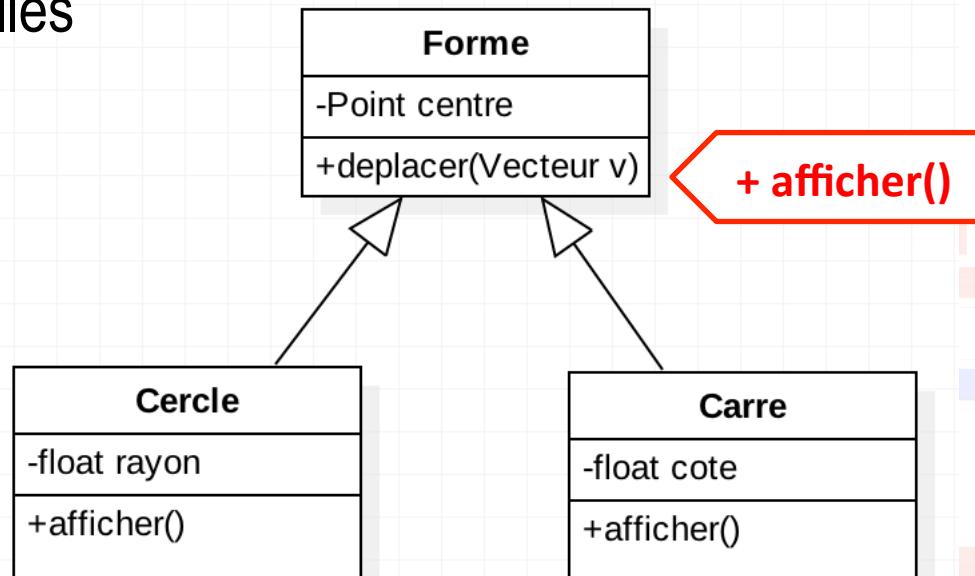
POLYMPHISME : SURCHARGE

- La signature c'est le nom de la méthode, le nombre, le type et l'ordre des paramètres

```
3 public class Personne1 {  
4     private String nom;  
5     private int age;  
6  
7     public void manger(){  
8     public int manger(){} /*méthode dupliquée en java  
malgré le type de retour pour le  
compilateur c'est la même signature*/  
10    public void manger(int t){}  
11    public void manger(char c){}  
12    public void manger(int i, int j) {}  
13    public void manger(int i, char c) {}  
14    public void manger(int x, char z) {} /* méthode dupliquée  
malgré le changement des noms paramètre parce qu'il s'agit  
des mêmes types*/  
15    public void manger(char c, int i) {}  
16    /*conclusion: la signature c'est le nom de la méthode,  
* le nombre, le type et l'ordre des paramètres */  
17  
18 }  
19  
20  
21 }
```

POLYMPHISME : REDÉFINITION

- Redéfinir dans une classe fille le code source d'une méthode déclarée dans (héritée de) la classe mère.
- Revenons à notre exemple de géométrie
 - La méthode afficher est une caractéristique commune au deux classes Cercle et Carré. Grâce à la redéfintion il est possible de la déclarée dans la classe mère (Forme) et de la redéfinir dans les classes filles



POLYMPHISME : REDÉFINITION

Redéfinition de la méthode afficher

```
Forme.java
package geometrie;

public class Forme {
    protected Point centre;

    public Forme(Point centre) {
        this.centre = centre;
    }

    public void afficher(){
        System.out.print("les centre est :");
        centre.afficher();
    }
}
```

```
Cercle.java
public class Cercle extends Forme {
    private float rayon;

    public Cercle(Point centre, float rayon) {
        super(centre);
        this.rayon = rayon;
    }

    public void afficher(){
        System.out.println("le centre est :");
        centre.afficher();
        System.out.println("le rayon est "+rayon);
    }
}
```

```
TBM Carré.java
public class Carré extends Forme {
    private int cote;

    public Carré(Point centre, int cote) {
        super(centre);
        this.cote = cote;
    }

    public void afficher(){
        System.out.println("le centre est :");
        centre.afficher();
        System.out.println("le coté est "+cote);
    }
}
```

2017/18 110

ATTENTION !!! POLYMPHISME : REDÉFINITION



```
Forme.java ✘
1 package geometrie;
2
3 public class Forme {
4     protected Point centre;
5
6     public Forme(Point centre) {
7         this.centre = centre;
8     }
9
10    public void afficher(){
11        System.out.print("le centre est :");
12        centre.afficher();
13    }
14}
15
```

ATTENTION

Si vous changez la signature de la méthode dans les classes filles C'est de la Surcharge et non de la Redéfinition

```
*Cercle.java ✘
4
3 public class Cercle extends Forme {
4     private float rayon;
5
6     public Cercle(Point centre, float rayon) {
7         super(centre);
8         this.rayon = rayon;
9     }
10    public void afficher(int i){←
11        System.out.println("le centre est :");
12        centre.afficher();
13        System.out.println("le rayon est "+rayon);
14    }
15} 2017/18
16
```

```
Carre.java ✘
4
3 public class Carre extends Forme {
4     private int cote;
5
6     public Carre(Point centre, int cote) {
7         super(centre);
8         this.cote = cote;
9     }
10    public void afficher(int i){←
11        System.out.println("le centre est :");
12        centre.afficher();
13        System.out.println("le coté est "+cote);
14    }
15}
16
```

TBM 111

POLYMPHISME : REDÉFINITION + RÉUTILISATION

Redéfinition de la méthode afficher

On remarque le même code

```
Forme.java ✘
1 package geometrie;
2
3 public class Forme {
4     protected Point centre;
5
6     public Forme(Point centre) {
7         this.centre = centre;
8     }
9
10    public void afficher(){
11        System.out.print("le centre est :");
12        centre.afficher();
13    }
14 }
15 
```

```
Cercle.java ✘
2
3 public class Cercle extends Forme {
4     private float rayon;
5
6     public Cercle(Point centre, float rayon) {
7         super(centre);
8         this.rayon = rayon;
9     }
10    public void afficher(){
11        System.out.println("le centre est :");
12        centre.afficher();
13        System.out.println("le rayon est "+rayon);
14    }
15 } 2017/18
16 
```

```
Carre.java ✘
2
3 public class Carre extends Forme {
4     private int cote;
5
6     public Carre(Point centre, int cote) {
7         super(centre);
8         this.cote = cote;
9     }
10    public void afficher(){
11        System.out.println("le centre est :");
12        centre.afficher();
13        System.out.println("le coté est "+cote);
14    }
15 }
16 
```

TBM 15 16 }

112

POLYMPHISME : REDÉFINITION + RÉUTILISATION

The diagram illustrates the concept of polymorphism through inheritance. A dashed line connects the `Forme` class at the top to its subclasses, `Cercle` and `Carre`, located below it. Red boxes highlight the `afficher()` methods in both subclasses, which are overrides of the method defined in the `Forme` class. Red arrows point from the `super.afficher()` call in each subclass's `afficher()` method back to the original `afficher()` method in the `Forme` class.

```
Forme.java
package geometrie;
public class Forme {
    protected Point centre;
    public Forme(Point centre) {
        this.centre = centre;
    }
    public void afficher(){
        System.out.print("les centre est :");
        centre.afficher();
    }
}
```

```
*Cercle.java
public class Cercle extends Forme {
    private float rayon;
    public Cercle(Point centre, float rayon) {
        super(centre);
        this.rayon = rayon;
    }
    public void afficher(){
        super.afficher();
        System.out.println("le rayon est "+rayon);
    }
}
```

```
Carre.java
public class Carre extends Forme {
    private int cote;
    public Carre(Point centre, int cote) {
        super(centre);
        this.cote = cote;
    }
    public void afficher(){
        super.afficher();
        System.out.println("le coté est "+cote);
    }
}
```

Possibilité de réutiliser la méthode afficher de la classe mère dans les classes filles avec le mot clé super

Redéfinition de la méthode afficher

2017/18 TBM 113

POLYMPHISME : SURCLASSEMENT

- Possibilité de pouvoir percevoir un objet en tant qu'instance de classes variées, selon les besoins

```
graph TD; ClassA[ClassA] --> ClassB[ClassB]; ClassB --> ClassC[ClassC]; ClassC --> ClassD[ClassD]
```

The diagram illustrates a class hierarchy with four classes: ClassA, ClassB, ClassC, and ClassD. ClassA is at the top, followed by ClassB, then ClassC, and finally ClassD at the bottom. Each class is represented by a rectangle with three horizontal lines inside, and inheritance is shown by hollow upward-pointing triangles connecting the classes.

Java code snippets are shown in five windows:

- ClassA.java**: package upcast;
public class ClassA {
}
- ClassB.java**: package upcast;
public class ClassB extends ClassA{
}
- ClassC.java**: package upcast;
public class ClassC extends ClassB{
}
- ClassD.java**: package upcast;
public class ClassD extends ClassC{
}
- Principale.java**: package upcast;
public class Principale {
 public static void main(String[] args) {
 //possible parce que le d est un c qui est un b qui est un a grâce à l'héritage
 ClassD d = new ClassD();
 ClassC c = new ClassD();
 ClassB b = new ClassD();
 ClassA a = new ClassD();
 // l'inverse n'est pas possible parce tous A n'est pas un d (Ex. Toute Forme n'est pas un Cercle)
 ClassD d1 = new ClassA();
 ClassD d2 = new ClassB();
 ClassD d3 = new ClassC();
 }
}

Annotations in the code:

- //possible parce que le d est un c qui est un b qui est un a grâce à l'héritage
- // l'inverse n'est pas possible parce tous A n'est pas un d (Ex. Toute Forme n'est pas un Cercle)

File numbers 12, 13, and 14 are crossed out.

2017/18 TBM

POLYMPHISME: RUNTIME BINDING (LIAISON DYNAMIQUE)

- Qu'affiche ce programme ?

The screenshot shows an IDE interface with four code editors:

- ClassA.java**:

```
1 package upcast;
2
3 public class ClassA {
4     public void quiSuisJe(){
5         System.out.println("je suis la classe A");
6     }
7 }
8
```
- ClassB.java**:

```
1 package upcast;
2
3 public class ClassB extends ClassA{
4     public void quiSuisJe(){
5         System.out.println("je suis la classe B");
6     }
7 }
8
```
- Principale.java**:

```
1 package upcast;
2
3 public class Principale {
4
5     public static void main(String[] args) {
6         ClassA a = new ClassB();
7         a.quiSuisJe();
8     }
9 }
10
```
- Console**:

No consoles to display at this time.

POLYMPHISME: RUNTIME BINDING (LIAISON DYNAMIQUE)

■ Résultat

The screenshot shows an IDE interface with four windows:

- ClassA.java**:

```
1 package upcast;
2
3 public class ClassA {
4     public void quiSuisJe(){
5         System.out.println("je suis la classe A");
6     }
7 }
8
```
- ClassB.java**:

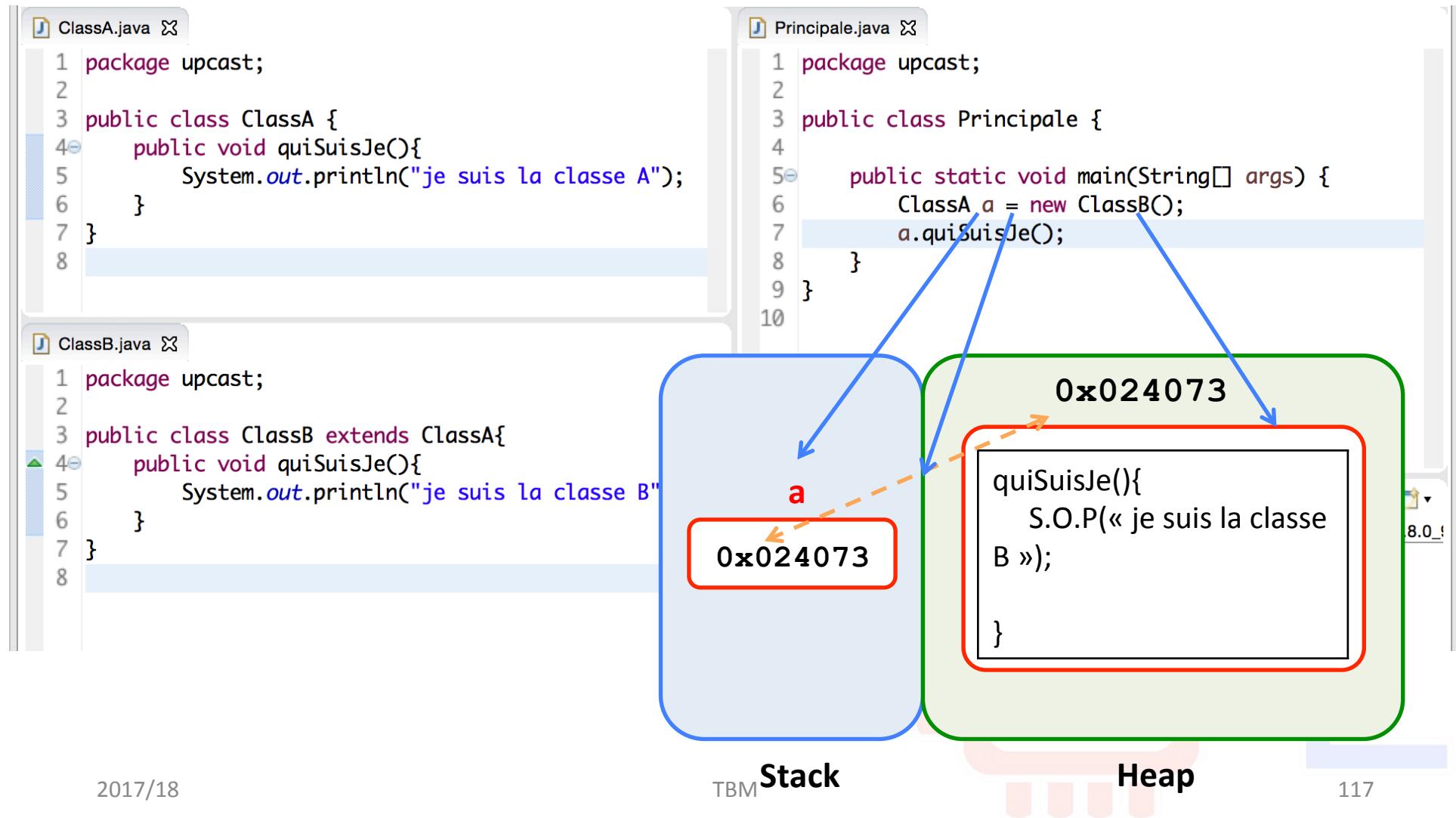
```
1 package upcast;
2
3 public class ClassB extends ClassA{
4     public void quiSuisJe(){
5         System.out.println("je suis la classe B");
6     }
7 }
8
```
- Principale.java**:

```
1 package upcast;
2
3 public class Principale {
4
5     public static void main(String[] args) {
6         ClassA a = new ClassB();
7         a.quiSuisJe();
8     }
9 }
```
- Console**:

```
<terminated> Principale (5) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_111
je suis la classe B
```

POLYMPHISME: RUNTIME BINDING (LIAISON DYNAMIQUE)

■ Explication



POLYMPHISME : SUBSTITUTION

C'est quoi le problème ?

```
3 public class Animale {  
4     public void manger(){  
5         System.out.println("je mange");  
6     }  
7     public void marcher(){  
8         System.out.println("je marche");  
9     }  
10 }
```

```
Chat.java ✘  
1 package upcast;  
2  
3 public class Chat extends Animale{  
4  
5     public void jouer(){  
6         System.out.println("je joue");  
7     }  
8 }
```

```
Principale.java ✘  
1 package upcast;  
2  
3 public class Principale {  
4  
5     public static void main(String[] args) {  
6         Animale a = new Chien();  
7         a.manger();  
8         a.marcher();  
9         a.garder();  
10    }  
11 }
```

```
Chat.java ✘  
1 package upcast;  
2  
3 public class Chat extends Animale{  
4  
5     public void jouer(){  
6         System.out.println("je joue");  
7     }  
8 }
```

Explication

l'objet Chien créé est « **surclassé** »
Il est vu de type Animale ses méthodes sont restreintes à celles de l'Animal
Il ne peut pas appeler la méthode garder()

The method garder() is undefined for the type Animale

POLYMPHISME : SUBSTITUTION

■ Solution

```
3 public class Animale {  
4     public void manger(){  
5         System.out.println("je mange");  
6     }  
7     public void marcher(){  
8         System.out.println("je marche");  
9     }  
10 }
```

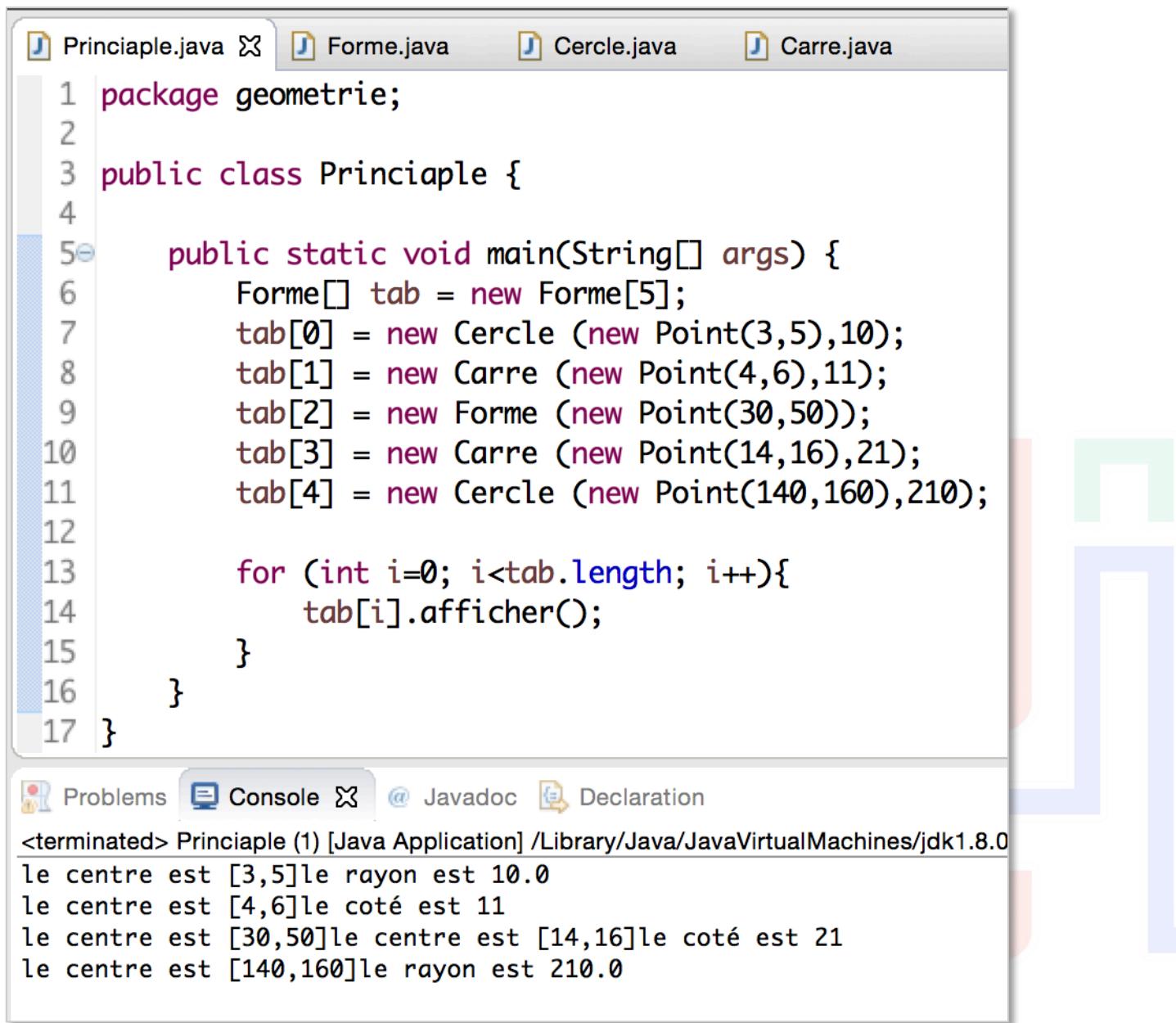
```
1 package upcast;  
2  
3 public class Chat extends Animale{  
4  
5     public void jouer(){  
6         System.out.println("je joue");  
7     }  
8 }
```

```
1 package upcast;  
2  
3 public class Chien extends Animale{  
4  
5     public void garder(){  
6         System.out.println("je tiens la ga...");  
7     }  
8 }
```

```
3 public class Principale {  
4  
5     public static void main(String[] args) {  
6         Animale a = new Chien();  
7         a.manger();  
8         a.marcher();  
9         ((Chien)a).garder();  
10 }  
11 }
```

Explication
Il faut donc substituer (casting) a par :
((Chien) a).garder();

EXEMPLE D'UTILISATION : TABLEAU POLYMORPHIQUE



The screenshot shows an IDE interface with a code editor and a console window.

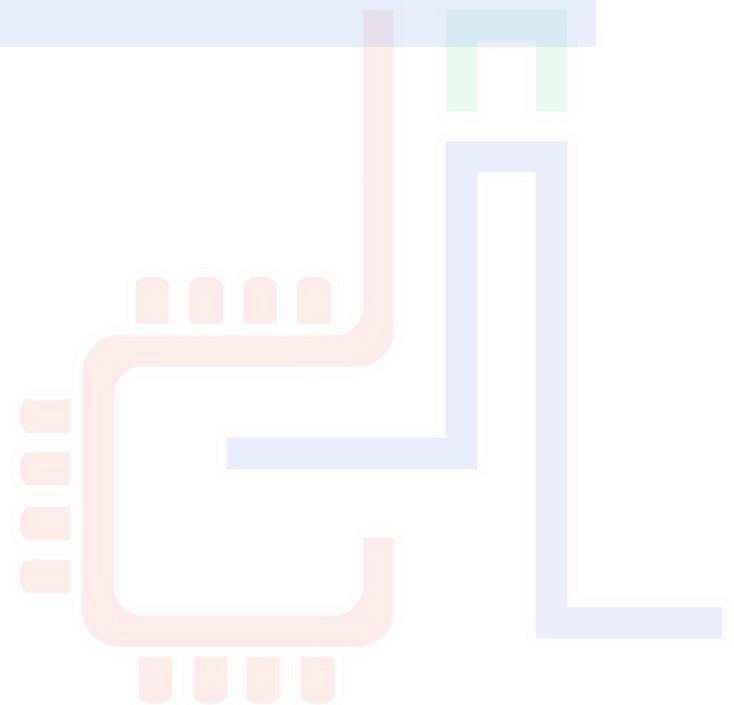
Code Editor:

```
1 package geometrie;
2
3 public class Principele {
4
5     public static void main(String[] args) {
6         Forme[] tab = new Forme[5];
7         tab[0] = new Cercle (new Point(3,5),10);
8         tab[1] = new Carre (new Point(4,6),11);
9         tab[2] = new Forme (new Point(30,50));
10        tab[3] = new Carre (new Point(14,16),21);
11        tab[4] = new Cercle (new Point(140,160),210);
12
13        for (int i=0; i<tab.length; i++){
14            tab[i].afficher();
15        }
16    }
17 }
```

Console Window:

```
<terminated> Principele (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0
le centre est [3,5]le rayon est 10.0
le centre est [4,6]le coté est 11
le centre est [30,50]le centre est [14,16]le coté est 21
le centre est [140,160]le rayon est 210.0
```

APRÈS LES ABCDEF, LES GHI DE JAVA



CONNAÎTRE LE TYPE D'OBJET ET DE CLASSE

The screenshot shows an IDE interface with three code editors and a console tab.

- Personne.java:** A class with private attributes nom and age, and a constructor taking String nom and int age, setting them to this.nom and this.age respectively.
- Etudiant.java:** A class extending Personne, with private attributes nom, classe, and age, and a constructor taking String n, int a, and String c, calling super(n, a) and setting this.classe = c.
- Principale.java:** A main method that creates Personne p, Etudiant e, and two more Etudiant objects f and g. It then prints various instanceof and getClass() results.

Console Output:

```
1- true
2- true
3- false
4- true
5- class exemple.Personne
6- class exemple.Etudiant
7- false
8- true
9- false
```

NOTION STATIC : VARIABLE DE CLASSE

- n'appartient pas à une instance mais appartient à la classe.
- elle est partagée par toutes les instances de la classe

```
Personne.java
1 package exemple;
2
3 public class Personne {
4     private String nom;
5     private int age;
6     static int nbrePersonne = 0;
7     public Personne(String n, int a) {
8         this.nom = n;
9         this.age = a;
10        age++;
11        nbrePersonne++;
12    }
13
14    public String toString(){
15        return ("je suis "+nom+" mon age l'annee"
16        + " prochaine est "+ age+
17        " nombre de personne = "+nbrePersonne);
18    }
19 }
20

Principale.java
1 package exemple;
2
3 public class Principale {
4     public static void main(String[] args) {
5         Personne p1 = new Personne("Tarek", 37);
6         Personne p2 = new Personne("Néjib", 80);
7         Personne p3 = new Personne("Mohamed", 41);
8
9         System.out.println(p1);
10        System.out.println(p2);
11        System.out.println(p3);
12    }
13 }
14
15
16
17 
```

Problems Console @ Javadoc Declaration

<terminated> Principale (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java (30 oct. 2017 15:50:16)

```
je suis Tarek mon age l'annee prochaine est 38 nombre de personne = 3
je suis Néjib mon age l'annee prochaine est 81 nombre de personne = 3
je suis Mohamed mon age l'annee prochaine est 42 nombre de personne = 3
```

NOTION STATIC : MÉTHODE DE CLASSE

- Parfois on se trouve à implémenter des méthodes dont le comportement ne dépend pas des variables d'instance.
 - Exemple méthode min (calcul du minimum) ou abs (calcul de la valeur absolue) dans une classe Calcul
- Il existe des cas où une instantiation d'une classe est inutile
 - Exemple pour calculer le min on n'est pas obligé d'instancier la classe Calcul
- Le mot clé **static** permet alors à une méthode d'être invoquée **sans avoir à instancier la classe qui la contient**

EXAMPLE

```
Calcul.java
1 package exemple;
2
3 public class Calcul {
4     public static int min (int a, int b){
5         if (a < b)
6             return a;
7         else
8             return b;
9     }
10    public static int abs(int a){
11        if (a < 0)
12            return -a;
13        else
14            return a;
15    }
16 }
Principale.java
1 package exemple;
2
3 public class Principale {
4     public static void main(String[] args) {
5         int x = Calcul.min(3, 12);
6         int y = Calcul.abs(-10);
7
8         System.out.println("x="+x+", "+"y="+y);
9     }
10
11
12
13
14 }
```

Problems Console Javadoc Declaration

<terminated> Principale (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java (30 oct. 2017 18:37:33)

x=3, y=10

RÈGLES

Accède à	Variable d'instance	Méthode d'instance	Variable de classe (static)	Méthode de classe (static)
Méthode d'instance	Oui	Oui	Oui	Oui
Méthode de classe (static)	Non	Non	Oui	Oui

Dans une méthode static on ne peut pas utiliser this

CLASSE ET MÉTHODE FINAL

- Classe final ne peut avoir de classe fille

```
Classe1.java
1 package exemple;
2
3 public final class Classe1 {
4
5     public void quiSuisJe(){
6         System.out.println("je suis classe 1");
7     }
8 }
```

```
*Classe2.java
1 package exemple;
2
3 public class Classe2 extends Classe1{
4     public void quiSuisJe(){
5         System.out.println("je suis classe 2");
6     }
7 }
```

The type Classe2 cannot subclass the final class Classe1

- Méthode final ne peut pas être redéfinie

```
Classe1.java
1 package exemple;
2
3 public class Classe1 {
4
5     public final void quiSuisJe(){
6         System.out.println("je suis classe 1");
7     }
8 }
```

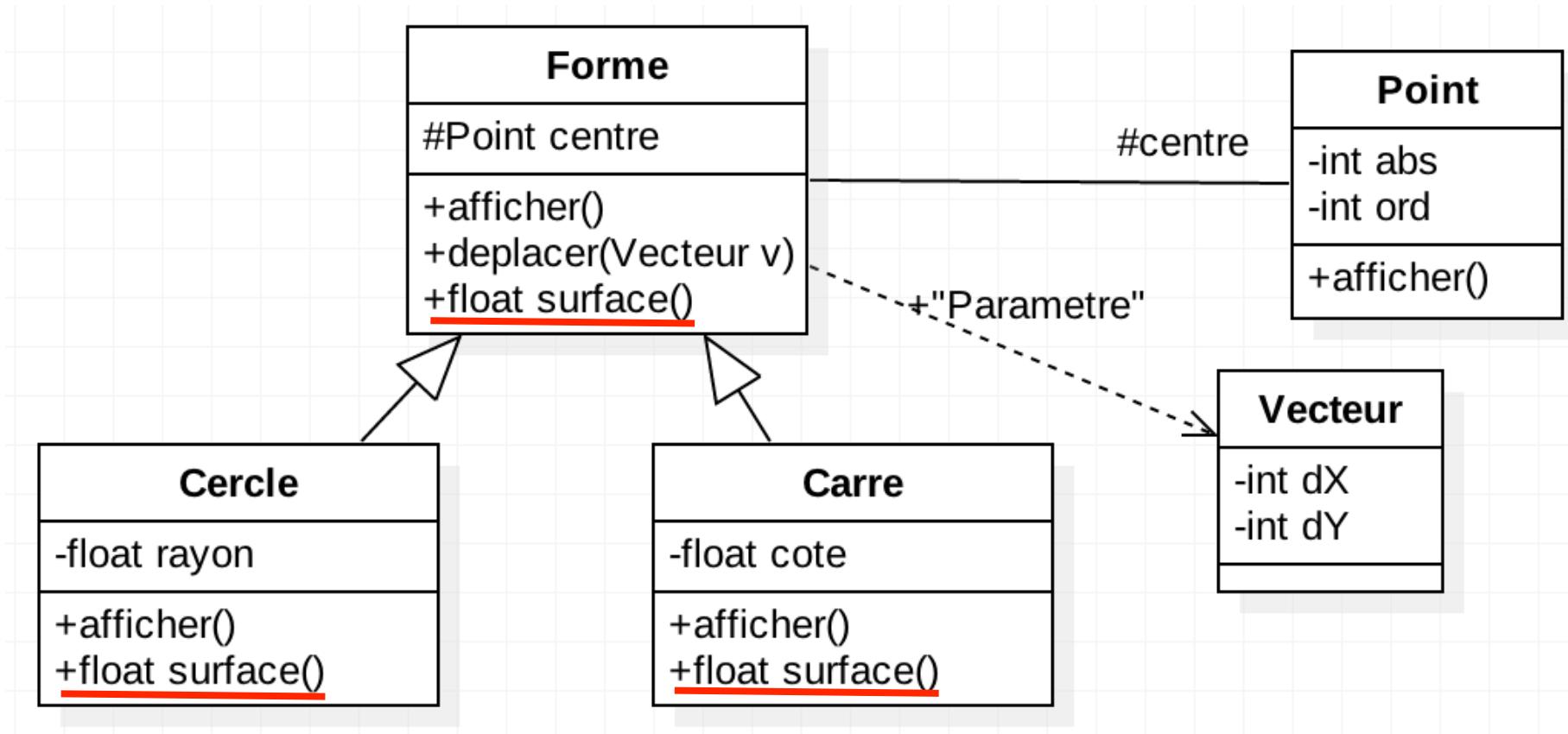
```
Classe2.java
1 package exemple;
2
3 public class Classe2 extends Classe1{
4     public void quiSuisJe(){
5         System.out.println("je suis classe 2");
6     }
7 }
```

Multiple markers at this line
- Cannot override the final method from Classe1
- overrides exemple.Classe1.quiSuisJe

ETUDE DE CAS V3.0

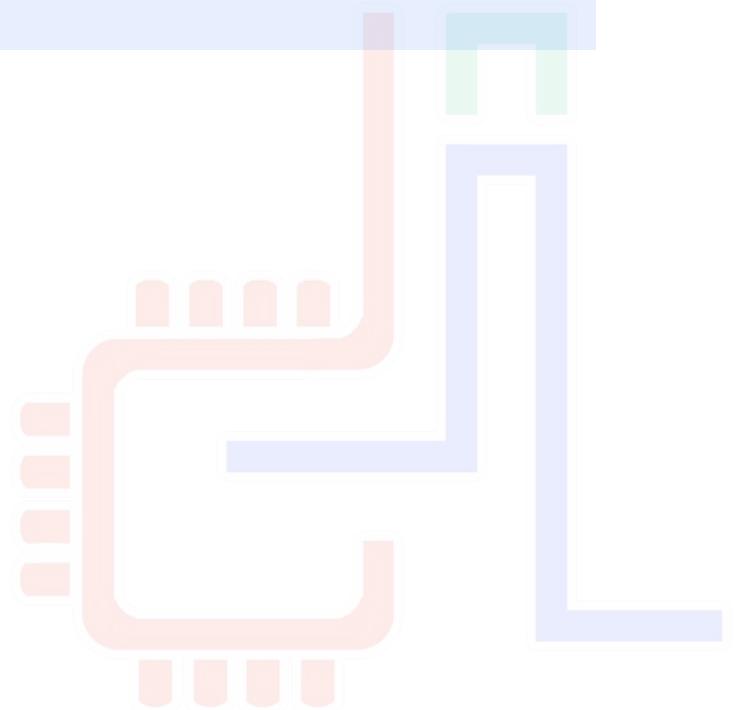
Créer et afficher deux cercles C1 et C2 et deux carrés Ca1 et Ca2. Un cercle est caractérisé par un centre, un rayon. Un carré est caractérisé par un centre et un côté. Nous souhaitons les déplacer selon des vecteurs et **calculer leurs surfaces**

UNE CONCEPTION S'IMPOSE COMME D'HABITUDE



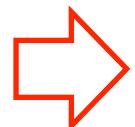
QUEL CODE METTRE DANS LA MÉTHODE SURFACE DE FORME ?

CLASSE ABSTRAITE



MÉTHODE ABSTRAITE

- Il peut donc, dans certains cas, être utile de définir une méthode **sans code source**.
- **Seule la signature** sera définie dans ce cas



la méthode sera alors déclarée **abstraite**

- En code Java
 - Il faut précéder la signature avec le mot clé **abstract** et mettre un ; après la déclaration de la signature

```
// définition d'une méthode abstraite  
abstract public float surface();
```

CLASSE ABSTRAITE

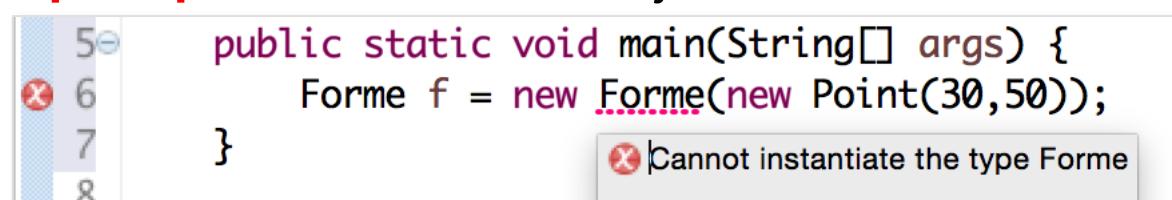
ATTENTION

On ne peut pas définir une méthode abstraite n'importe où. Cela ne peut se faire que dans une définition de classe abstraite.

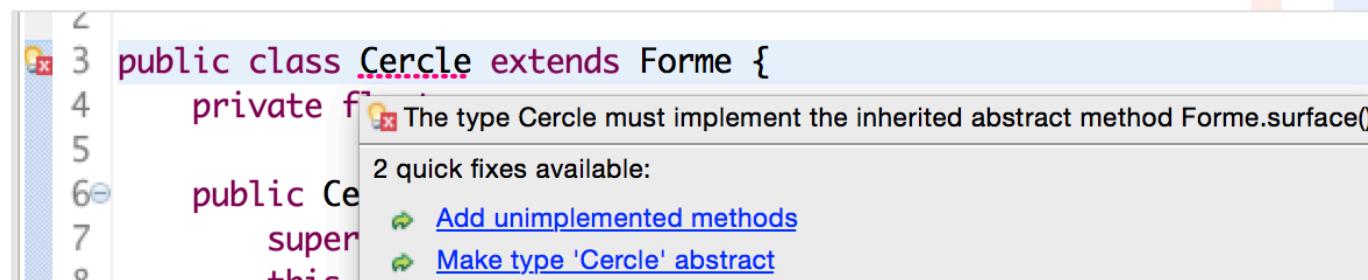
```
3 abstract public class Forme {  
4     protected Point centre;  
5  
6     public Forme(Point centre) {  
7         this.centre = centre;  
8     }  
9     public void deplacer(Vecteur v){  
10        centre.setAbs(centre.getAbs() + v.getdX());  
11        centre.setOrd(centre.getOrd() + v.getdY());  
12    }  
13    public void afficher(){  
14        System.out.print("le centre est ");  
15        centre.afficher();  
16    }  
17    // définition d'une méthode abstraite  
18    abstract public float surface();  
19 }
```

CLASSE ABSTRAITE

- Une classe abstraite est une classe qui contient au moins une méthode abstraite
- Conséquences :
 - On **ne peut plus instancier** d'objet de cette classe.



- **Il faut définir** des classes filles **implémentant** les méthodes abstraites



- Le polymorphisme (**surclassement**) **est toujours possible**

```
public static void main(String[] args) {  
    Forme f = new Cercle (new Point(3,5),10);  
}
```

DANS NOTRE EXEMPLE

```
Cercle.java
1 package geometrie;
2
3 public class Cercle extends Forme {
4     private float rayon;
5
6     public Cercle(Point centre, float rayon) {
7         super(centre);
8         this.rayon = rayon;
9     }
10
11    public void afficher(){
12        super.afficher();
13        System.out.println("le rayon est "+rayon);
14    }
15
16    public float surface(){
17        return (float) (rayon * Math.pow(Math.PI, 2))
18    }
19 }
```

```
Principe.java
1 package geometrie;
2
3 public class Principe {
4     public static void main(String[] args) {
5         Forme[] tab = new Forme[5];
6         tab[0] = new Cercle (new Point(3,5),10);
7         tab[1] = new Carre (new Point(4,6),11);
8         tab[2] = new Cercle (new Point(30,50), 20);
9         tab[3] = new Carre (new Point(14,16),21);
10        tab[4] = new Cercle (new Point(140,160),210);
11
12        for (int i=0; i<tab.length; i++){
13            tab[i].afficher();
14            System.out.println("la surface = "+
15                tab[i].surface());
16        }
17    }
18 }
19 }
```

Console output:

```
le centre est [3,5]le rayon est 10.0
la surface = 98.696045
le centre est [4,6]le coté est 11
la surface = 121.0
le centre est [30,50]le rayon est 20.0
la surface = 197.39209
le centre est [14,16]le coté est 21
la surface = 441.0
le centre est [140,160]le rayon est 210.0
la surface = 2072.617
```

QUELLE SUPERCLASSE ON VA DÉFINIR DANS CE CAS ?

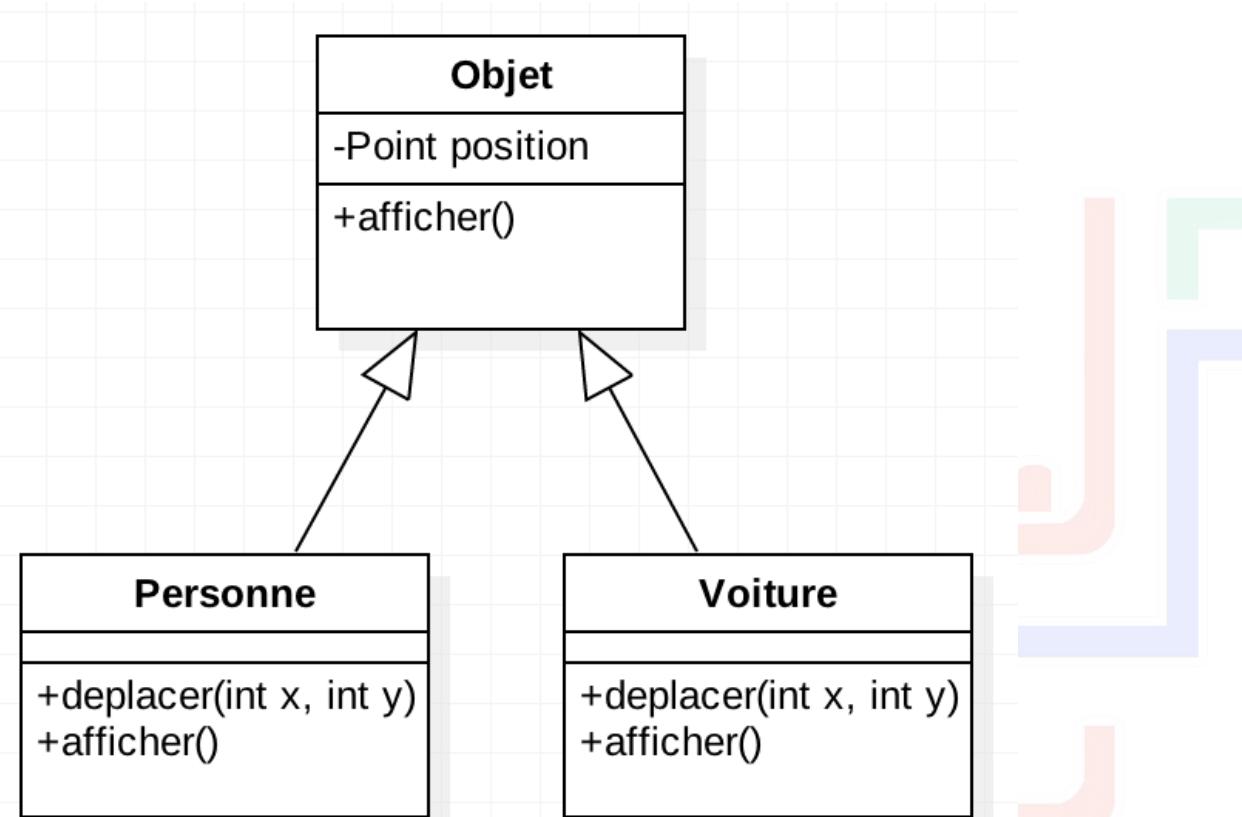
- Parce qu'il y a des caractéristiques communes
- et pouvoir bénéficier du polymorphisme

Personne
-Point position
+deplacer(int x, int y) +afficher()

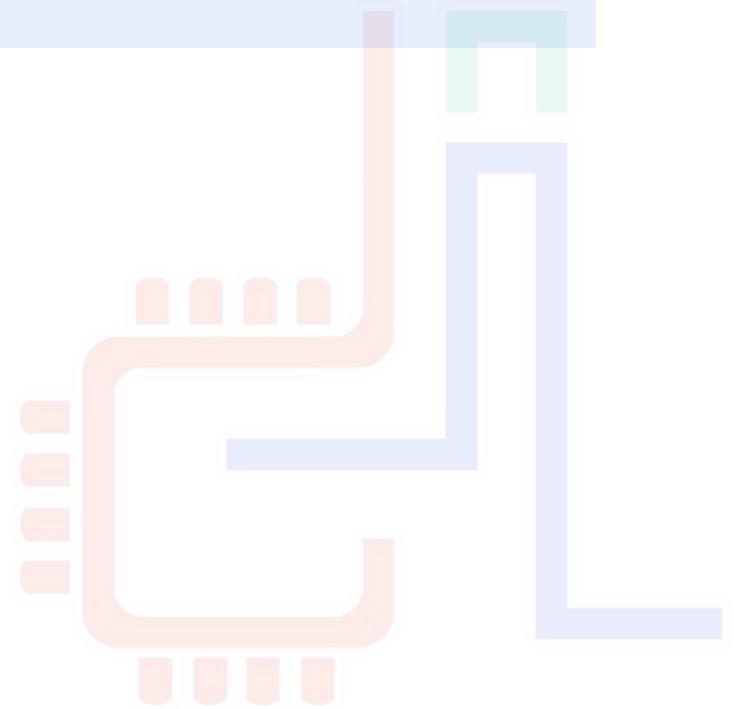
Voiture
-Point position
+deplacer(int x, int y) +afficher()

UNE SOLUTION POSSIBLE MAIS ...

- Que peux-on faire avec la méthode **déplacer(int x, int y)** qui est commune aux deux classes ?
- Parce que c'est pas tout objet qui se déplace



LES INTERFACES

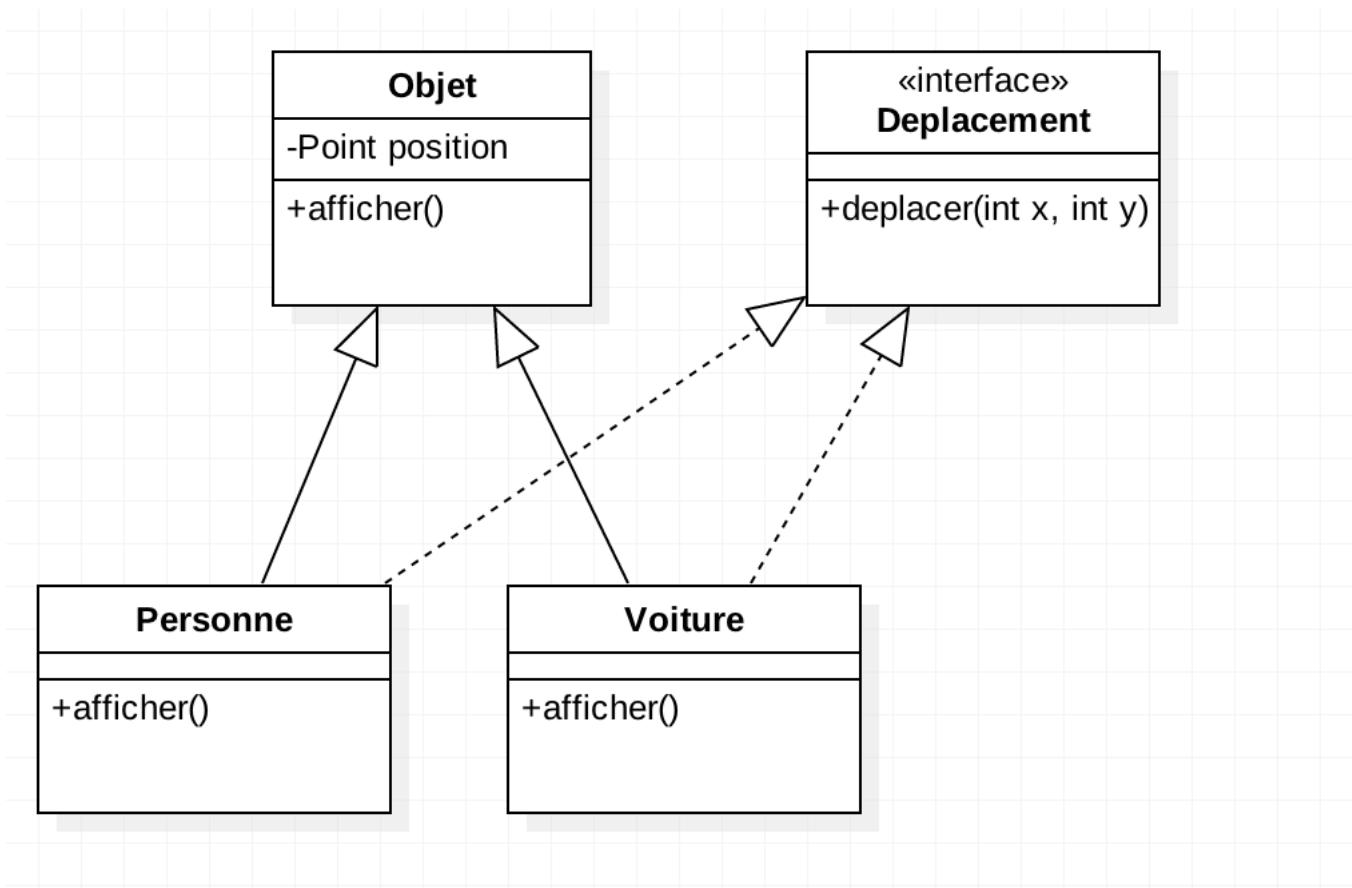


NOTION D'INTERFACE

- Une interface définit un **comportement d'une classe** qui doit être implémenté par une classe,
- Techniquement, c'est un **type, similaire à la classe**, qui contient seulement des constantes et des signatures de méthodes.
- Une interface « **est** » une classe 100% abstraite

REVENONS À NOTRE EXEMPLE

- Pour le comportement commun, Nous définissons l'**interface Déplacement** que les classes Personne et Voiture **implémenteront**



EN CODE JAVA

The diagram illustrates the relationship between four Java files:

- Objet.java**: A class named `Objet` with a protected attribute `Point position` and a constructor that initializes it. It also has a method `afficher()` that prints "je suis un objet".
- Personne.java**: A class named `Personne` that extends `Objet` and implements the `Deplacement` interface. It has a constructor that takes a `Point position` and calls the super constructor. It overrides the `afficher()` method to print "je suis une personne". It also implements the `deplacer()` method.
- Deplacement.java**: A declaration of an interface named `Deplacement` with a single method `deplacer(int x, int y)`.
- Voiture.java**: A class named `Voiture` that extends `Objet` and implements the `Deplacement` interface. It has a constructor that takes a `Point position` and calls the super constructor. It overrides the `afficher()` method to print "je suis une Voiture". It also implements the `deplacer()` method.

Dashed arrows indicate the relationships:

- A green dashed arrow points from the `position` attribute in `Objet` to the `position` parameter in the `deplacer` method of `Deplacement`.
- A red dashed arrow points from the `deplacer` method in `Personne` to the `deplacer` method in `Deplacement`.
- A green dashed arrow points from the `deplacer` method in `Voiture` to the `deplacer` method in `Deplacement`.

```
Objet.java
package exInterface;
public class Objet {
    protected Point position;
    public Objet(Point position) {
        this.position = position;
    }
    public void afficher(){
        System.out.println("je suis un objet");
    }
}

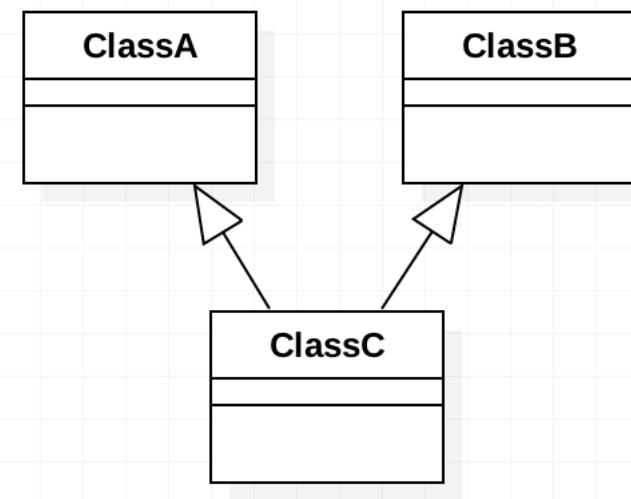
Personne.java
package exInterface;
public class Personne extends Objet implements Deplacement{
    public Personne(Point position) {
        super(position);
    }
    public void afficher(){
        System.out.println("je suis une personne");
    }
    public void deplacer(int x, int y){
        System.out.println("je marche vers la position "+x+", "+y);
    }
}

Deplacement.java
package exInterface;
// déclaration d'une interface
public interface Deplacement {
    public void deplacer(int x, int y);
}

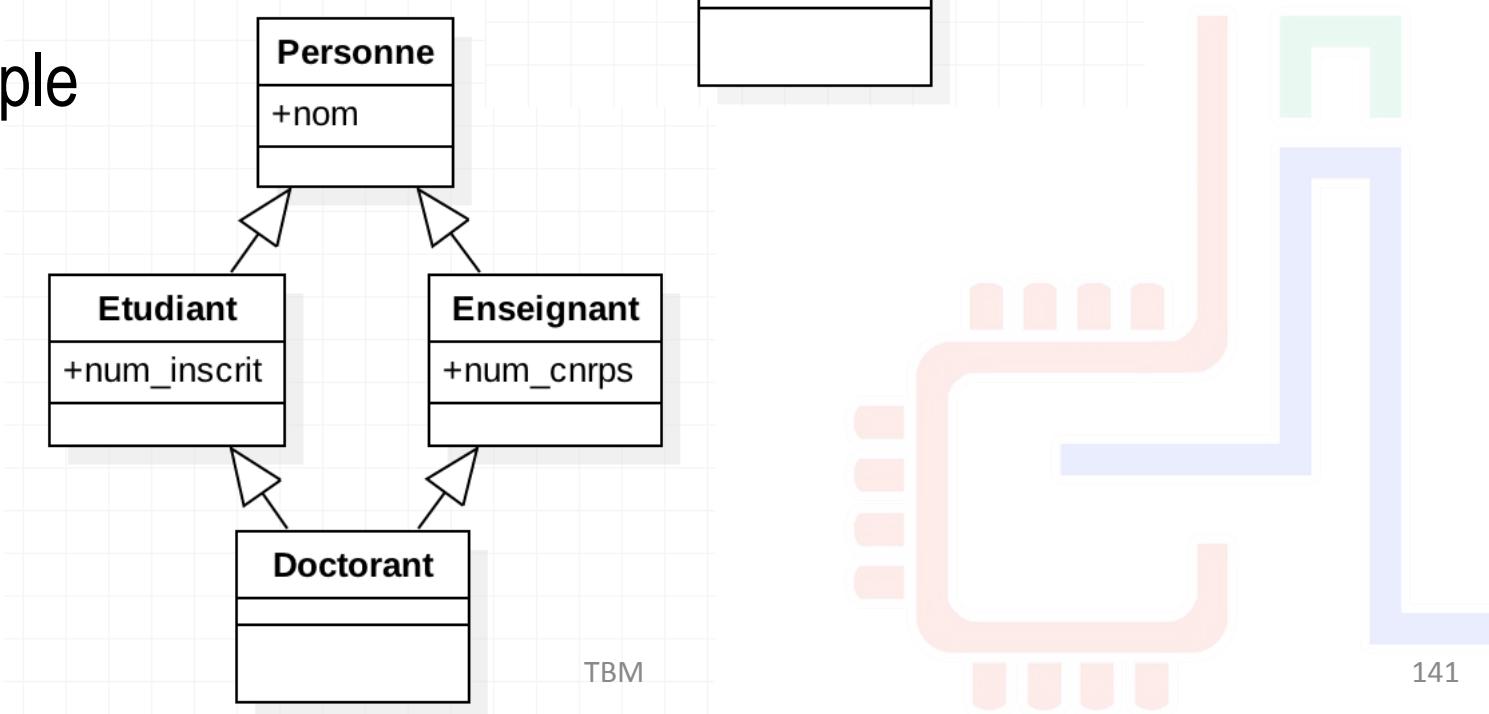
Voiture.java
package exInterface;
public class Voiture extends Objet implements Deplacement {
    public Voiture(Point position) {
        super(position);
    }
    public void afficher(){
        System.out.println("je suis une Voiture");
    }
    public void deplacer(int x, int y){
        System.out.println("je marche vers la position "+x+", "+y);
    }
}
```

L'HÉRITAGE MULTIPLE EN POO

- Une classe peut dériver d'un nombre quelconque de classes de base

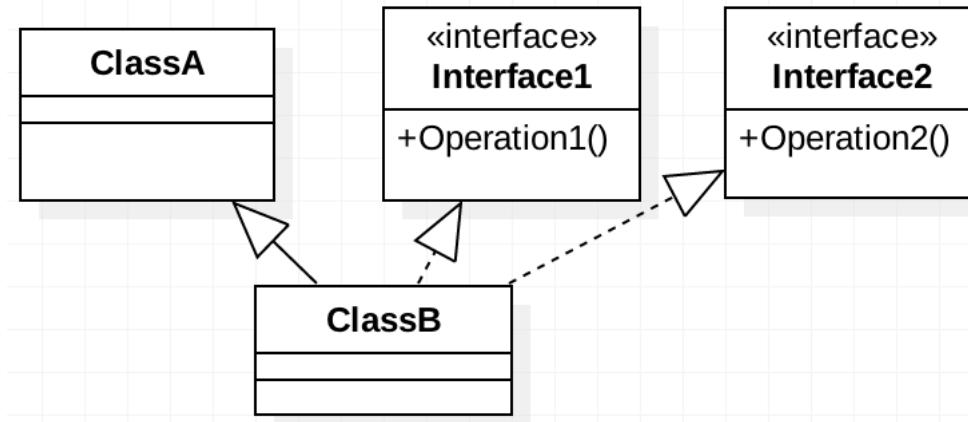


- Exemple



ATTENTION ! JAVA NE SUPPORTE PAS L'HÉRITAGE MULTIPLE

- Les interfaces permettent de le simuler



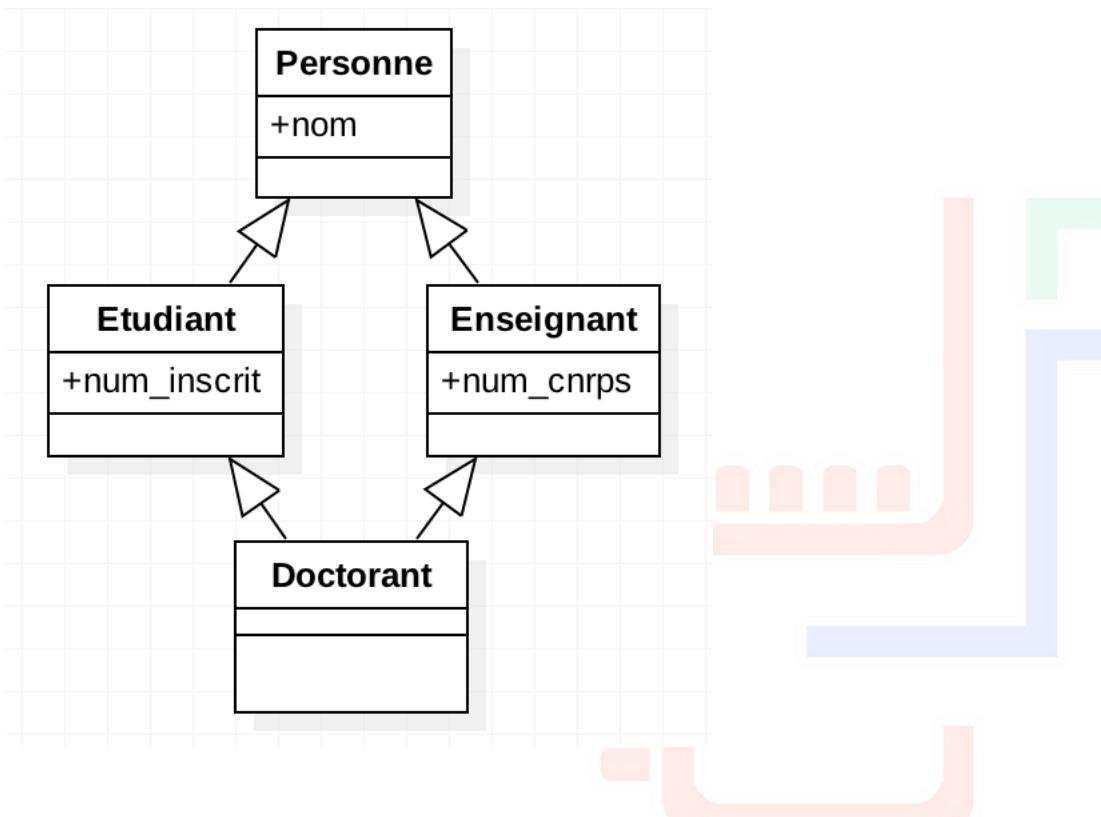
The screenshot shows four code editor panes:

- ClassA.java:** package lesInterfaces;
public class ClassA {
 public ClassA() { }
}
- Interface1.java:** package lesInterfaces;
public interface Interface1 {
 public void op1();
}
- Interface2.java:** package lesInterfaces;
public interface Interface2 {
 public void op2();
}
- ClassB.java:** package lesInterfaces;
public class ClassB extends ClassA implements Interface1, Interface2 {
 public ClassB() {}
 public void op1() {/* code op1 */}
 public void op2() {/* code op2 */}
}

Blue arrows highlight the inheritance relationship 'extends ClassA' and the implementation relationship 'implements Interface1, Interface2' in the ClassB.java code.

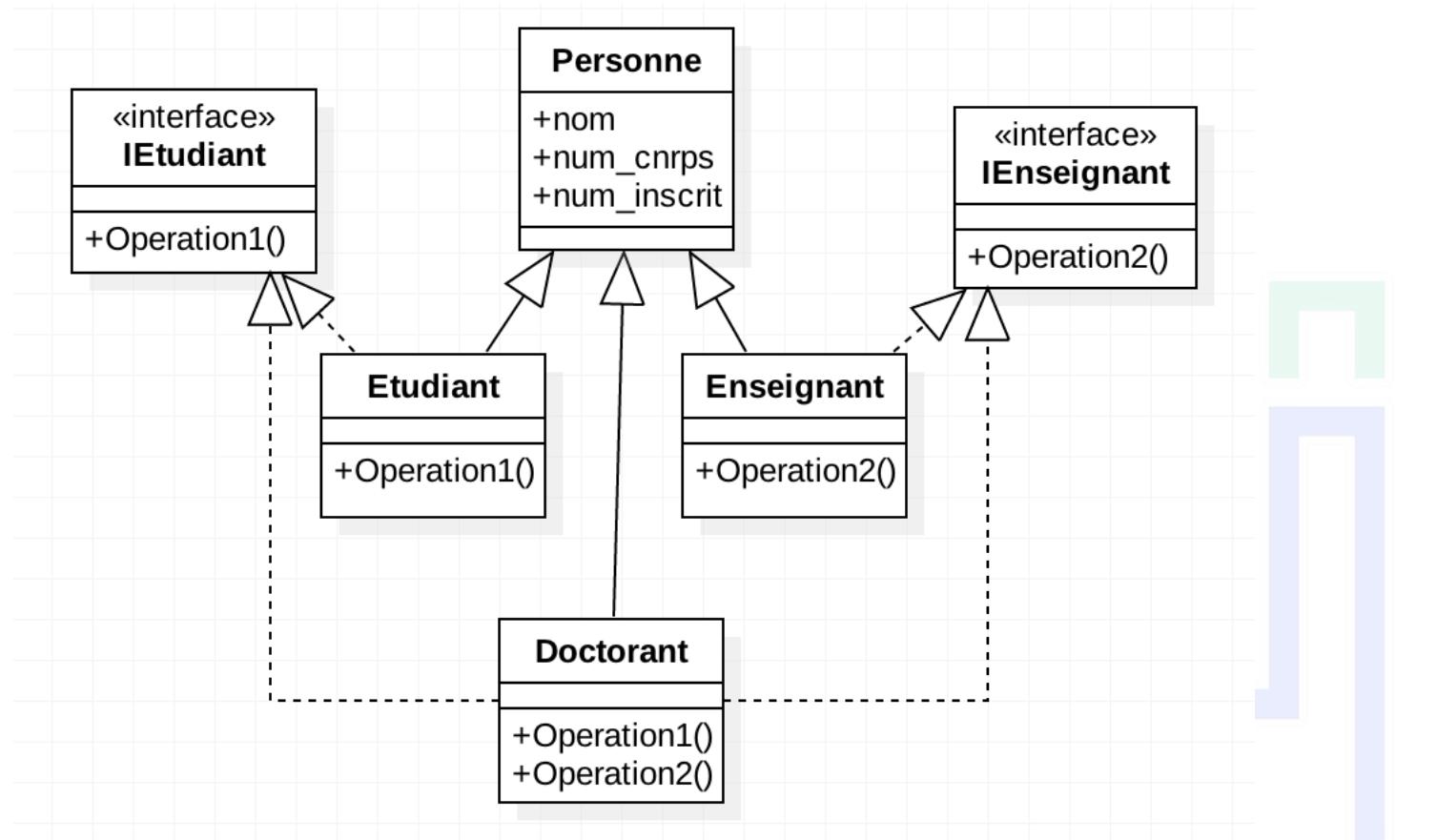
ATTENTION ! JAVA NE SUPPORTE PAS L'HÉRITAGE MULTIPLE

- Mais dans notre exemple comment faire pour l'héritage des attributs? Et pour utiliser le surclassement (Doctorant « est un » Etudiant et le Doctorant « est un » Enseignant



ATTENTION ! JAVA NE SUPPORTE PAS L'HÉRITAGE MULTIPLE

- solution possible de la simulation sera conceptuelle avec les interfaces



REMARQUES

■ Héritage entre classes

```
3 public class ClassA {  
4     public ClassA() {}  
5 }  
  
3 public class ClassB extends ClassA {  
4     public ClassB() {}  
5 }
```

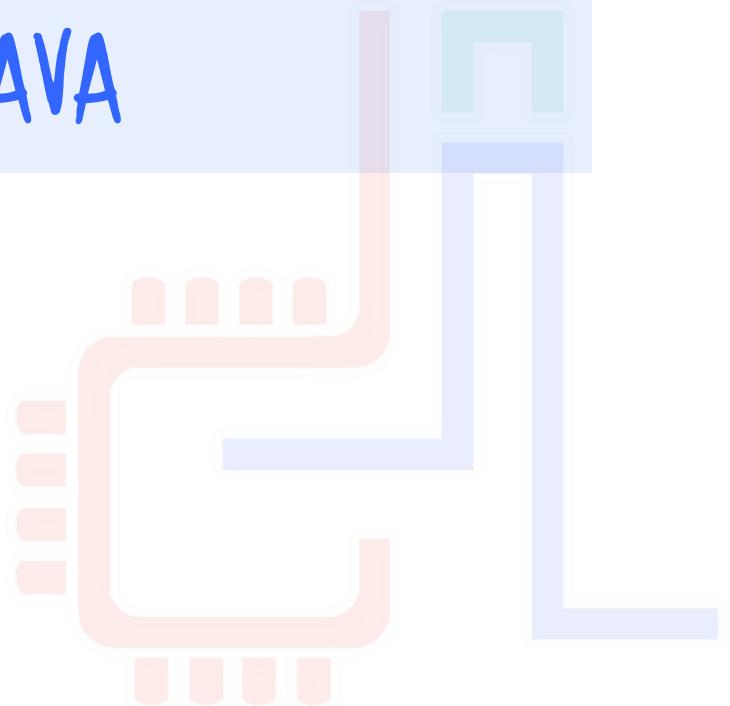
■ Héritage entre interfaces

```
3 public interface Interface1 {  
4     public void op1();  
5 }  
  
3 public interface Interface2 extends Interface1{  
4     public void op2();  
5 }
```

■ Implémentation entre classe et interfaces

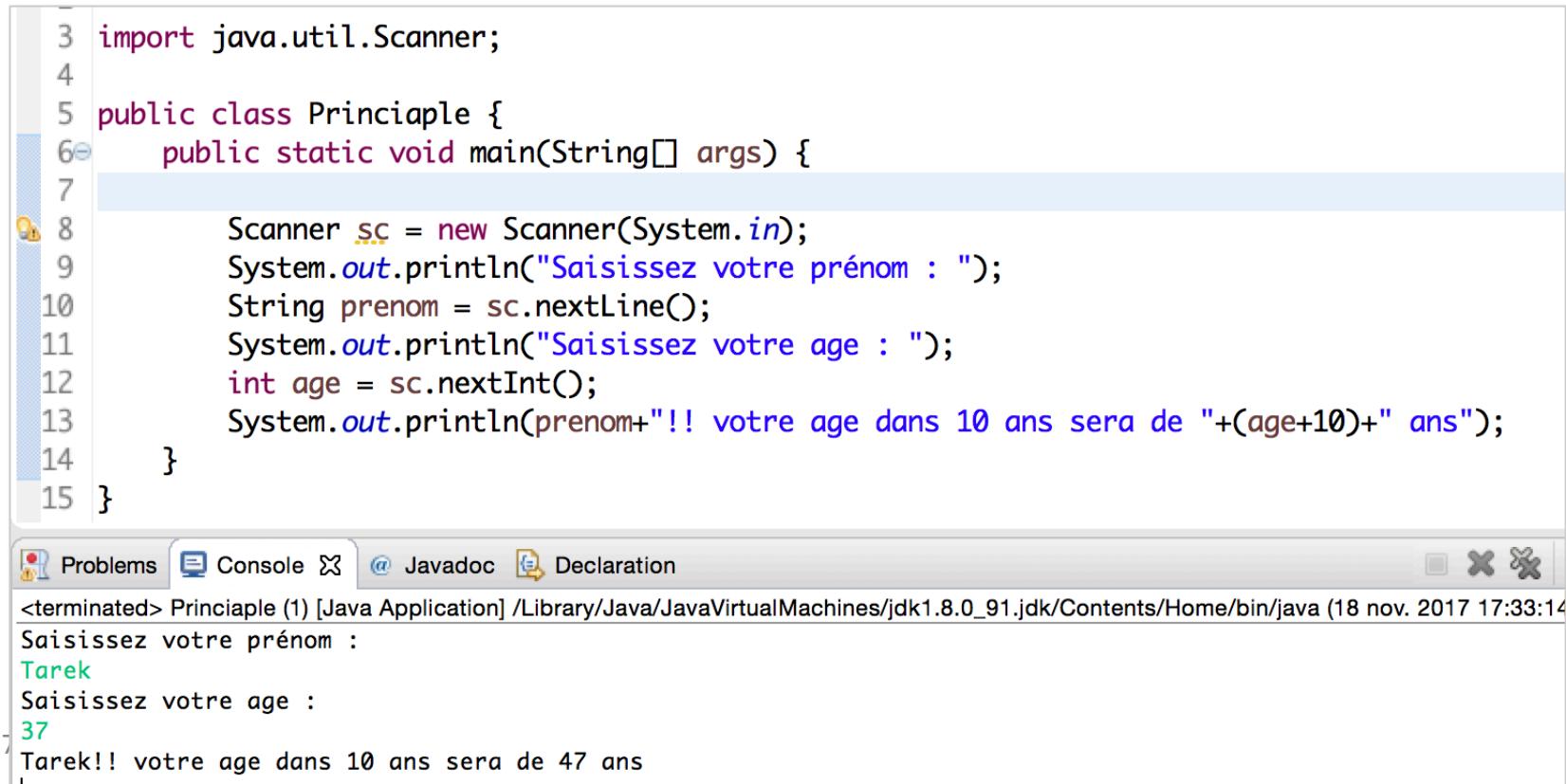
```
3 public interface Interface1 {  
4     public void op1();  
5 }  
  
3 public interface Interface2 extends Interface1{  
4     public void op2();  
5 }  
  
1 package lesInterfaces;  
2  
3 public class ClassC implements Interface2{  
4     public ClassC() {}  
5     public void op1() /* code op1 */  
6     public void op2() /* code op2 */  
7 }
```

APRÈS LES ABCDEFGHI,
LES JKL DE JAVA



LIRE AVEC LA CLASSE SCANNER

- Permet le «balayage» des chaines de caractères et des types primitifs
 - utilisée avec les flux `InputStream` ou pour lire un fichier.
 - ajoutée au package `java.util` dans la version 1.5.0 de java



The screenshot shows a Java application window with the following details:

- Code Area:** Displays a Java code snippet named "Principe".

```
3 import java.util.Scanner;
4
5 public class Principe {
6     public static void main(String[] args) {
7
8         Scanner sc = new Scanner(System.in);
9         System.out.println("Saisissez votre prénom : ");
10        String prenom = sc.nextLine();
11        System.out.println("Saisissez votre age : ");
12        int age = sc.nextInt();
13        System.out.println(prenom+"!! votre age dans 10 ans sera de "+(age+10)+" ans");
14    }
15 }
```
- Console Tab:** Shows the application's output.

```
Saisissez votre prénom :
Tarek
Saisissez votre age :
37
Tarek!! votre age dans 10 ans sera de 47 ans
```
- Bottom Status Bar:** Shows the date and time: "18 nov. 2017 17:33:14".

LIRE AVEC LA CLASSE SCANNER (SUITE)

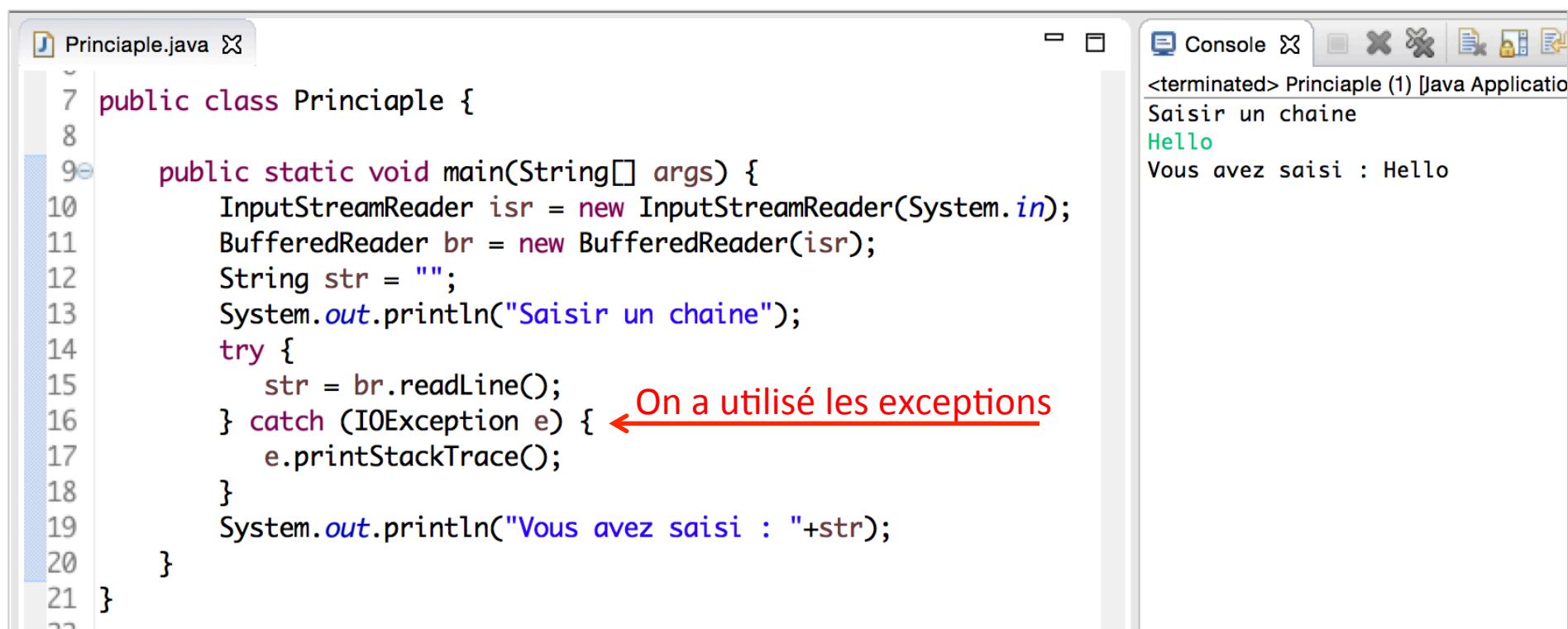
```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt(); // lire un entier
double d = sc.nextDouble(); // lire un double
long l = sc.nextLong(); // lire un long
byte b = sc.nextByte();
float f = sc.nextFloat(); // lire un réel
// pour les caractère on n'a pas un nextChar
String str = sc.nextLine(); // on lit la chaîne
char c = str.charAt(0); // on récupère le caractère
```

■ Remarque

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Saisissez un entier : ");
    int i = sc.nextInt();
    System.out.println("Saisissez une chaîne : ");
    //On vide la ligne avant d'en lire une autre à cause de nextInt()
    sc.nextLine();
    // on relie la ligne
    String str = sc.nextLine();
    System.out.println("FIN ! ");
```

COMMENT FAIRE SANS SCANNER

- **InputStreamReader** : est un pont d'un Flux d'octets à un flux de caractère. Lit les octets et les décode en caractères
- **BufferedReader** : lit le texte à partir d'un flux d'entrée de caractères pour les mettre dans une zone tampon et faciliter leur lecture



The screenshot shows an IDE interface with two panes. The left pane displays the code for a Java class named `Principle.java`. The right pane shows the `Console` output of the application.

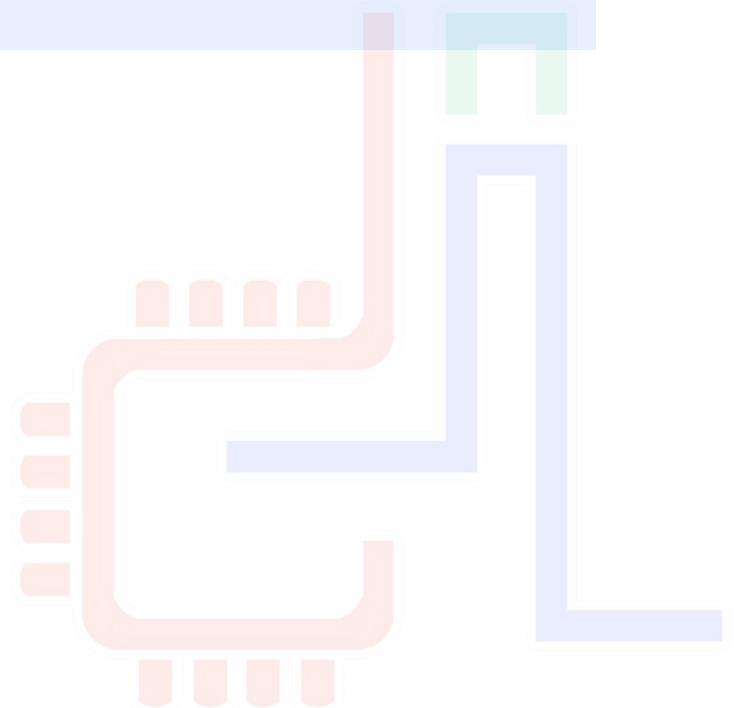
```
Principle.java
public class Principle {
    public static void main(String[] args) {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String str = "";
        System.out.println("Saisir un chaine");
        try {
            str = br.readLine();
        } catch (IOException e) { ← On a utilisé les exceptions
            e.printStackTrace();
        }
        System.out.println("Vous avez saisi : "+str);
    }
}
```

Console Output:

```
<terminated> Principle (1) [Java Application]
Saisir un chaine
Hello
Vous avez saisi : Hello
```

A red annotation with the text "On a utilisé les exceptions" and a red arrow points to the `catch (IOException e)` block in the code.

LES EXCEPTIONS



DÉFINITION

- Événement exceptionnel : erreur qui se produit lors de l'exécution → Cause l'arrêt du programme
- Exemples

The screenshot shows a Java code editor and a terminal window. The code editor displays a file named 'Principale.java' with the following content:

```
1 package mathematique;
2
3 public class Principale {
4
5     public static void main(String[] args) {
6         int x = 10, y = 0, z;
7         z = x / y;
8     }
9
10}
```

The terminal window below shows the output of the program execution:

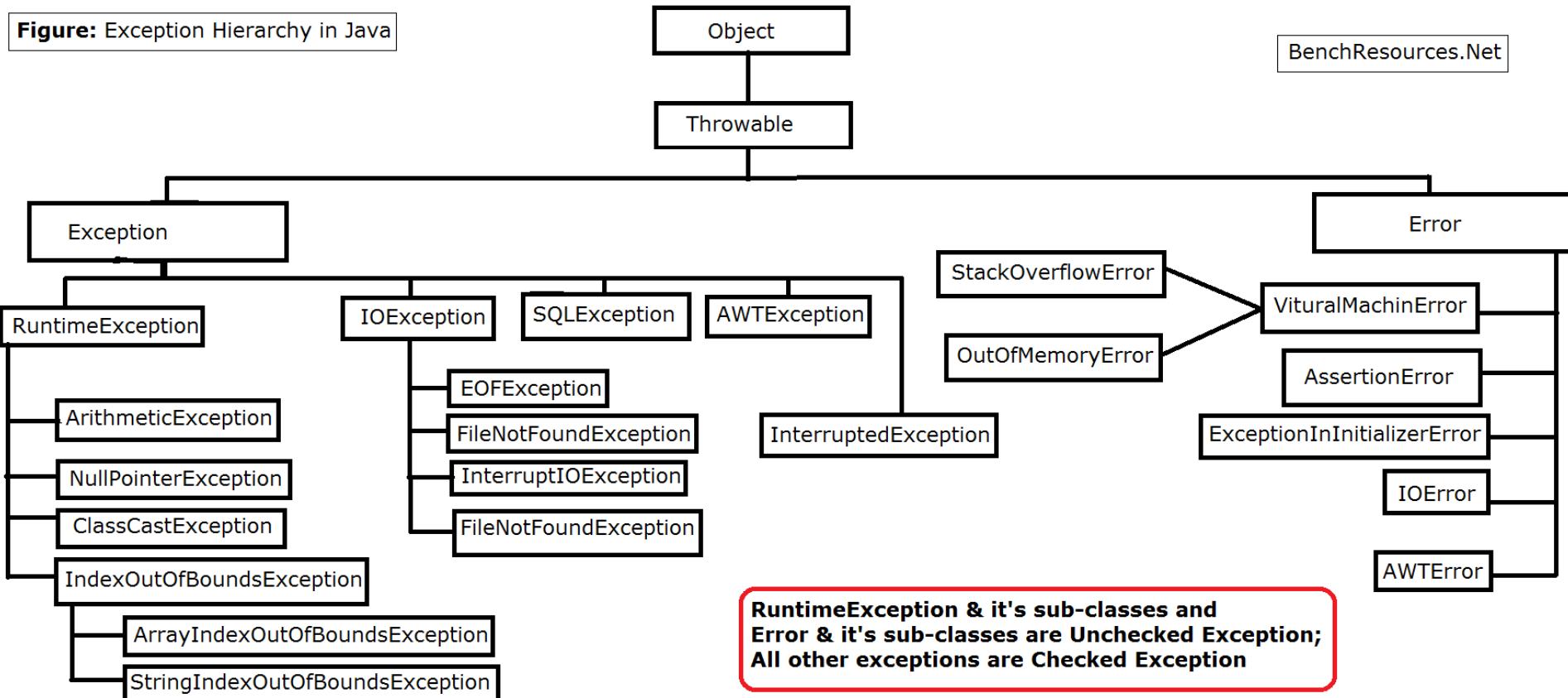
```
<terminated> Principale (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.j
Exception in thread "main" java.lang.ArithmetricException: / by zero
at mathematique.Principale.main(Principale.java:7)
```

EXEMPLES

Name	Description
NullPointerException	Thrown when attempting to access an object with a reference variable whose current value is null
ArrayIndexOutOfBoundsException	Thrown when attempting to access an array with an invalid index value (either negative or beyond the length of the array)
IllegalArgumentException	Thrown when a method receives an argument formatted differently than the method expects.
IllegalStateException	Thrown when the state of the environment doesn't match the operation being attempted, e.g., using a Scanner that's been closed.
NumberFormatException	Thrown when a method that converts a String to a number receives a String that it cannot convert.
IOException	While using file input/output stream related exception
SQLException	While executing queries on database related to SQL syntax
DataAccessException	Exception related to accessing data/database
ClassNotFoundException	Thrown when the JVM can't find a class it needs, because of a command-line error, a classpath issue, or a missing .class file
InstantiationException	Attempt to create an object of an abstract class or interface.

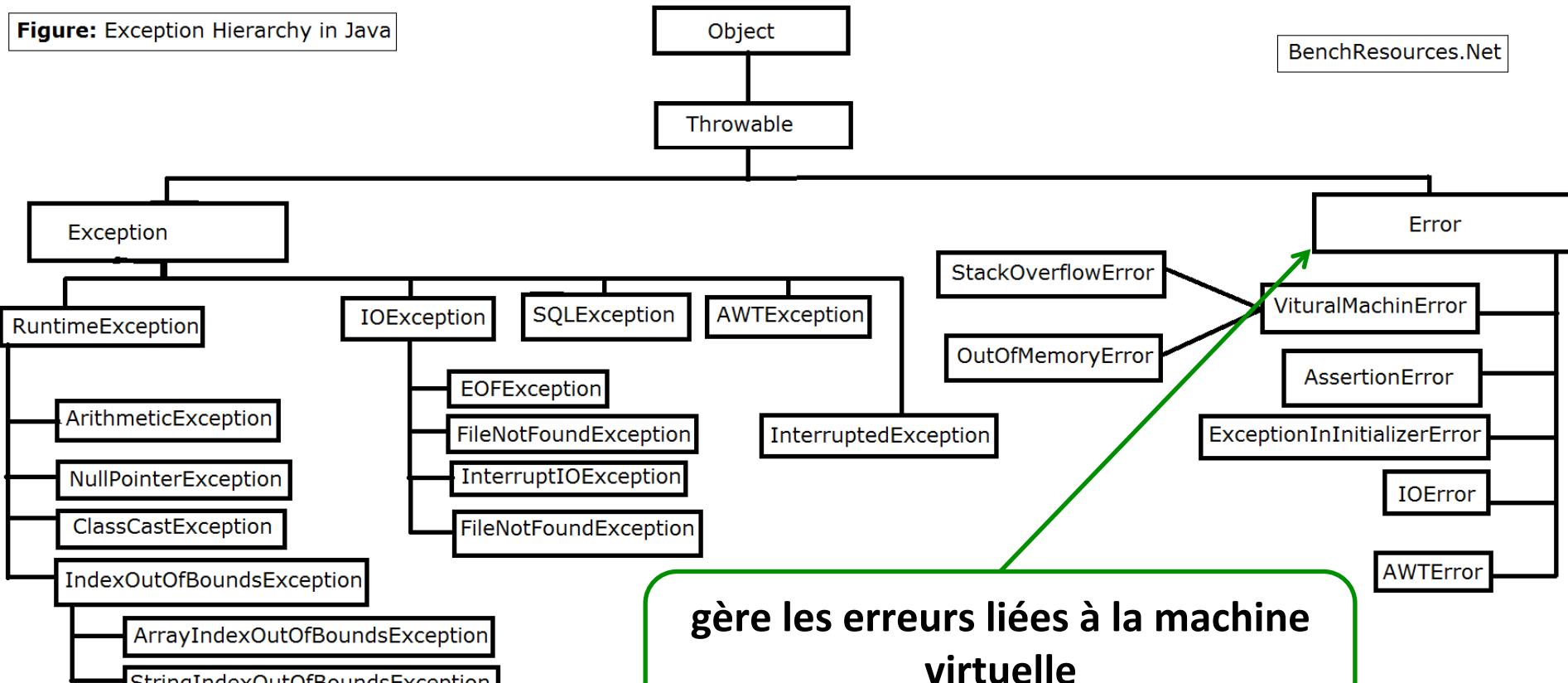
HIÉRARCHIE DES EXCEPTIONS

Figure: Exception Hierarchy in Java



HIÉRARCHIE DES EXCEPTIONS : ERROR

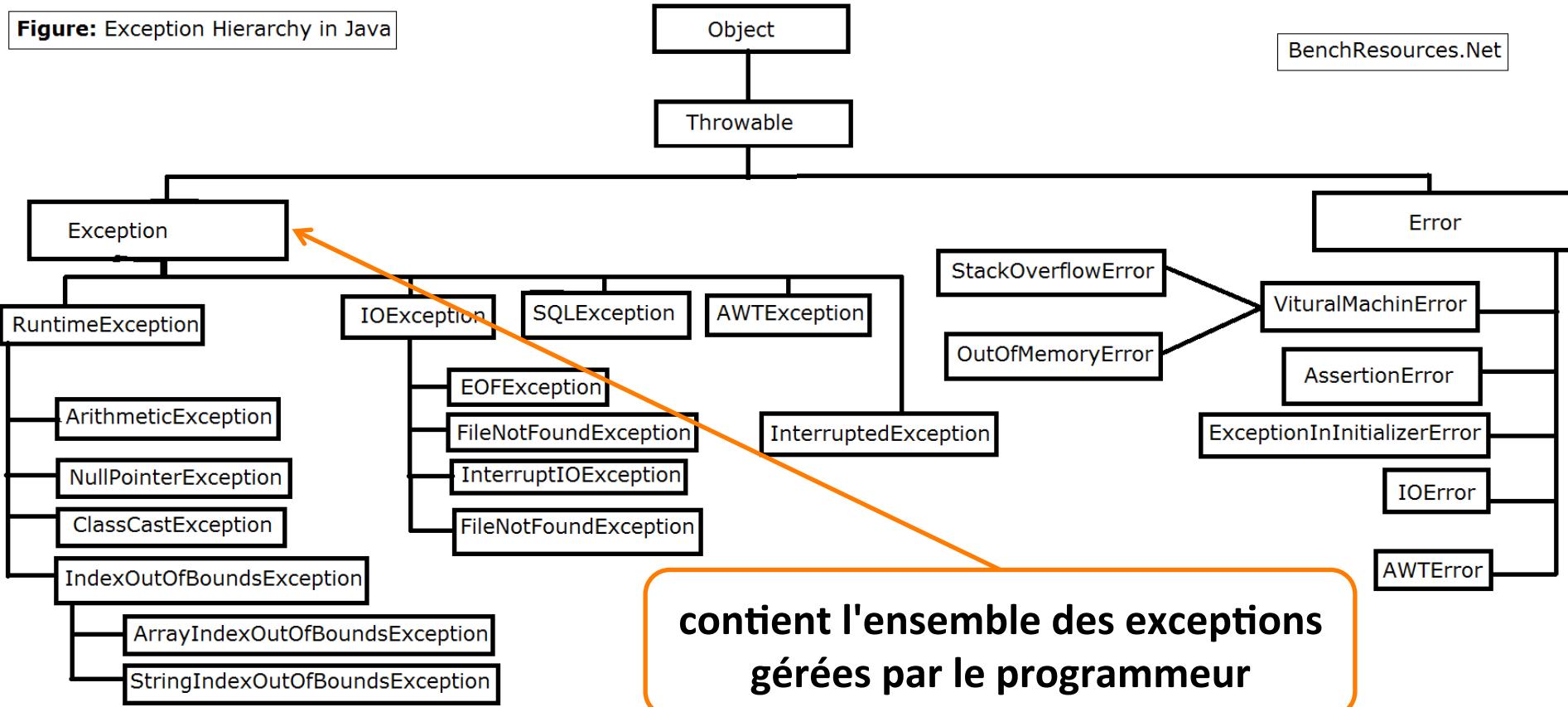
Figure: Exception Hierarchy in Java



BenchResources.Net

HIÉRARCHIE DES EXCEPTIONS : EXCEPTION

Figure: Exception Hierarchy in Java



PRINCIPE

```
Principale.java ✘
1 package mathematique;
2
3 public class Principale {
4
5     public static void main(String[] args) {
6         int tableau = {0,1,2,3,4,5};
7         for (int i=0; i < 7; i++){
8             System.out.println(tableau[i]);
9             System.out.println("FIN !");
10
11     }
12
13 }
14
15
```

Problems Console ✘ @ Javadoc Declaration

<terminated> Principale (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Content

0
1
2
3
4
5

Exécution normale Signal d'exception
Interruption

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
at mathematique.Principale.main(Principale.java:8)

Arrêt définitif du programme

```
Principale.java ✘
1 package mathematique;
2
3 public class Principale {
4
5     public static void main(String[] args) {
6         int tableau = {0,1,2,3,4,5};
7         for (int i=0; i < 7; i++){
8             try {
9                 System.out.println(tableau[i]);
10            }catch(ArrayIndexOutOfBoundsException e){
11                System.out.println("Il n y a que 6 elements");
12            }
13
14     }
15
16 }
17
18
19 }
20
```

Problems Console ✘ @ Javadoc Declaration

<terminated> Principale (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Content

0
1
2
3
4
5

Exécution normale Traitement de l'exception

Il n y a que 6 elements
FIN !

Reprise du programme

GESTION D'EXCEPTIONS : 2 SOLUTIONS

- Certaines opérations ou appel de méthodes peuvent déclencher des Exceptions. Le système exige alors leur gestion
- Exemple

The screenshot shows a Java code editor with the following code:

```
6 public class Principale {  
7  
8     public static void main(String[] args) {  
9         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
10  
11         String str = "";  
12         str = br.readLine();  
13     }  
14 }  
15  
16 }
```

An error tooltip is displayed over the line `str = br.readLine();`. The tooltip title is "Unhandled exception type IOException". It contains two quick fix options:

- Add throws declaration
- Surround with try/catch

SOLUTION 1 : AJOUTER UN BLOCK TRY/CATCH

```
try{
    /*
     *bloc de programme susceptible
     * de provoquer une exception
     * */
    }catch(Exception e){ /* TypeDException */
        /* bloc de traitement de l'exception
         * e contiendra l'instance de l'Exception
         * générée
         * */
    }
```

```
7 public class Principale {
8
9     public static void main(String[] args) {
10        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
11
12        String str = "";
13        try{
14            str = br.readLine();
15        }catch(IOException e){
16            System.out.println(e.getMessage());
17        }
18    }
19}
```

SOLUTION 2 : FAIRE MONTER L'EXCEPTION

- Au lieu de traiter l'exception avec un **block try/catch** dans la méthode, il est possible de faire monter l'exception qui sera traitée ultérieurement avec le mot clé **throws**

```
7 public class Principale {  
8  
9     public static String saisirNom() throws IOException {  
10        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
11        System.out.println("Donner votre nom : ");  
12        return br.readLine();  
13    }  
14    public static void main(String[] args) {  
15        String nom = "";  
16        try {  
17            nom = saisirNom();  
18        }catch(IOException e) {}  
19    }  
20 }
```

L'exception générée par saisirNom sera traitée dans le programme appelant

CRÉER SES PROPRES EXCEPTION

- Crédit et gestion de ses propres exceptions selon le besoin du programme au lieu d'utiliser des types d'exceptions standard
- Pour un plus haut niveau de contrôle sur le programme
- **1^{ère} étape :** Créer son exception qui **spécialise** la classe Exception

```
3 public class AgeException extends Exception {  
4  
5     public AgeException() {  
6         super("L'age ne peut pas être négatif");  
7     }  
8  
9     public AgeException(String message) {  
10        super(message);  
11    }  
12}
```

CRÉER SES PROPRES EXCEPTION

■ 2^{ème} étape : tester et générer l'exception

- **throw** de lever une exception manuellement en instanciant un objet de type Exception (ou un objet hérité).
- **throws** permet de signaler à la JVM qu'une méthode est susceptible de générer une exception

```
3 public class Personne {  
4     protected String nom;  
5     protected int age;  
6  
7     public Personne(String n, int a) throws AgeException {  
8         if (a < 0)  
9             throw new AgeException();  
10        this.nom = n;  
11        this.age = a;  
12    }  
13}
```

CRÉER SES PROPRES EXCEPTION

```
AgeException.java
1 package exemple;
2
3 public class AgeException extends Exception {
4
5     public AgeException() {
6         super("L'age ne peut pas être négatif");
7     }
8
9     public AgeException(String message) {
10        super(message);
11    }
12 }
```

```
Personne.java
1 package exemple;
2
3 public class Personne {
4     protected String nom;
5     protected int age;
6
7     public Personne(String n, int a) throws AgeException {
8         if (a < 0) throw new AgeException();
9         this.nom = n;
10        this.age = a;
11    }
12 }
```

```
Etudiant.java
1 package exemple;
2
3 public class Etudiant extends Personne{
4     private int numero;
5     public Etudiant(String n, int a, int numero) throws AgeEx
6         super(n, a);
7         this.numero = numero;
8     }
9     public String toString(){
10        return "nom: "+nom+" | age: "+age+" | numero: "
11        +numero;
12    }
13 }
14 }
```

```
Principale.java
1
2
3 public class Principale {
4     public static void main(String[] args) {
5
6         Etudiant e = null;
7         try {
8             e = new Etudiant("Tarek", -3,1);
9         }catch(AgeException ae){
10            System.out.println(ae.getMessage());
11            System.exit(0);
12        }
13        System.out.println(e);
14    }
15 }
```

Problems Console @ Javadoc Declaration

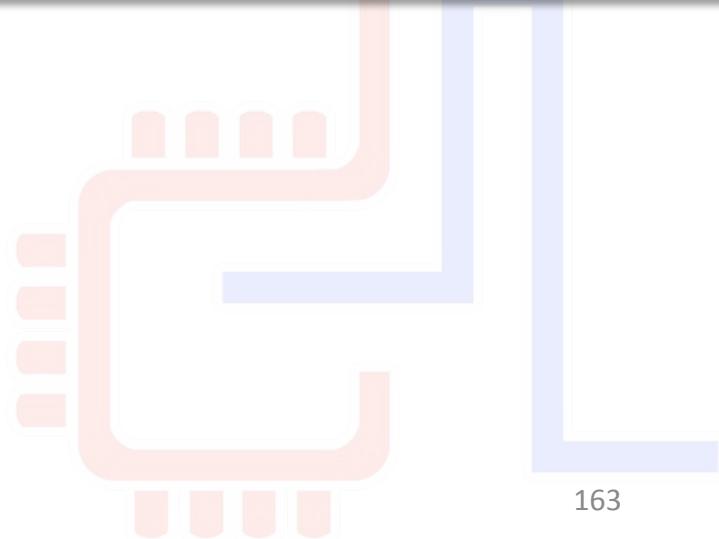
<terminated> Principale (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Home/bin/java (1 déc. 2017 11:15:38)

L'age ne peut pas être négatif

GESTION DE PLUSIEURS EXCEPTIONS (1/3)

```
AgeException.java
1 package exemple;
2
3 public class AgeException extends Exception {
4
5     public AgeException() {
6         super("L'age ne peut pas être négatif");
7     }
8
9     public AgeException(String message) {
10        super(message);
11    }
12}
```

```
NomException.java
1 package exemple;
2
3 public class NomException extends Exception{
4
5     public NomException() {
6         super("Nom trop court");
7     }
8
9 }
```



GESTION DE PLUSIEURS EXCEPTIONS (2/3)

```
Personne.java ✘
3 public class Personne {
4     protected String nom;
5     protected int age;
6
7     public Personne(){
8         nom = "";
9         age = 0;
10    }
11    public Personne(String n, int a) throws AgeException, NomException {
12        if (a < 0)
13            throw new AgeException();
14        if (n.length() < 2)
15            throw new NomException();
16        this.nom = n;
17        this.age = a;
18    }
19 }
20

Etudiant.java ✘
3 public class Etudiant extends Personne{
4     private int numero;
5
6     public Etudiant(){
7         super();
8         numero = -1;
9     }
10    public Etudiant(String n, int a, int numero) throws AgeException , NomException {
11        super(n, a);
12        this.numero = numero;
13    }

```

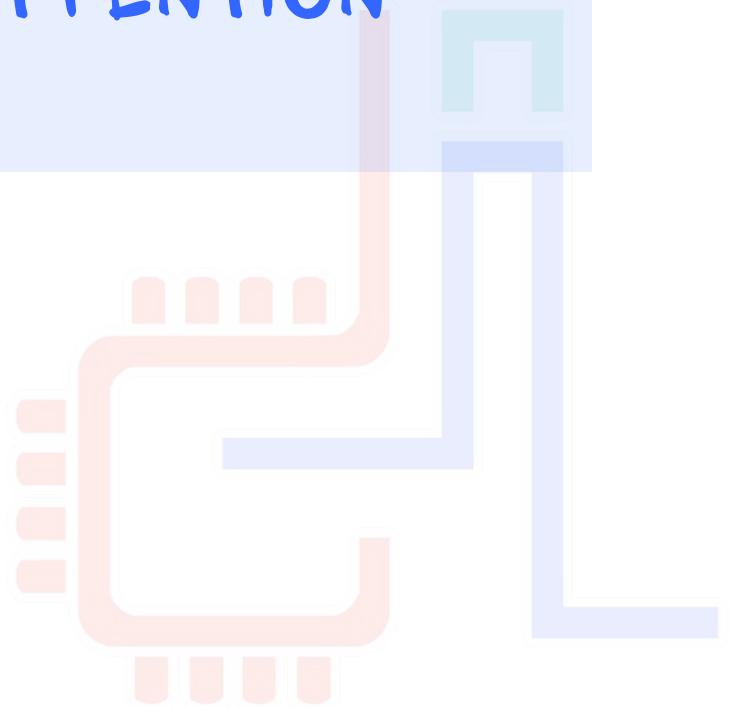
GESTION DE PLUSIEURS EXCEPTIONS (3/3)

```
1 package exemple;
2
3 public class Principale {
4     public static void main(String[] args) {
5
6         Etudiant e = null;
7         try {
8             e = new Etudiant("Tarek", -3,1);
9         }
10        catch(AgeException ae){
11            System.out.println(ae.getMessage());
12        }
13        catch(NomException ne){
14            System.out.println(ne.getMessage());
15        }
16        finally{
17            if (e == null)
18                e = new Etudiant();
19        }
20        System.out.println(e);
21    }
22 }
```

BLOCK FINALLY

- Est utilisé pour effectuer des nettoyages (fermer des fichiers, libérer des ressources...).
- Un bloc finally suit:
 - soit un bloc try
 - soit un bloc try suivi d'un ou plusieurs bloc catch
- Il contient les instructions qui sont exécutées après avoir quitté la clause /try, indépendamment du type de sortie
 - normalement, exception gérée par un catch, exception non saisie, return, break, continue.
- Seul l'appel de System.exit() empêchera l'exécution du bloc finally

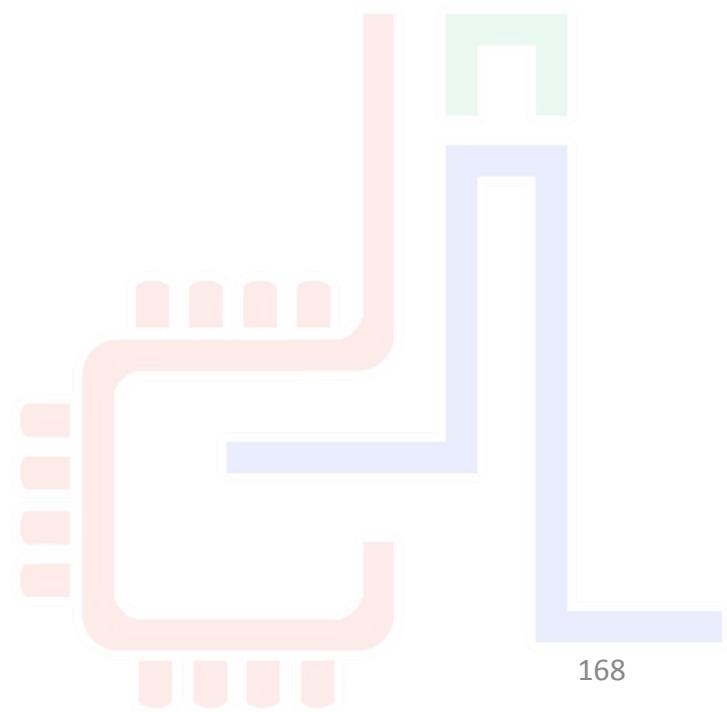
MERCI POUR VOTRE ATTENTION



REFLEXIVITÉ

2017/18

TBM



168