Méthodes de Conception Logicielle Design-Patterns

L3 Info – **SINFL5A1**

Année 2024/2025

Yann Mathet yann.mathet@unicaen.fr

Objectifs et chronologie

- Révisions sur le développement objet en Java
- Le cycle de vie du logiciel (de l'étude des besoins jusqu'au délivrable), selon quelques méthodes (Cascade, méthodes agiles, etc.)
- Principes des bonnes pratiques de conception (modularité, maintenabilité, évolutivité, robustesse)
- Etude et mise en pratique de « design patterns » (patrons de conception)
- Tests logiciels

Qu'est-ce que Java ?

- Un langage « orienté objet »
- Multi plate-forme :
  - PC (linux, windows, etc.)
  - Mac.
  - Web (applets)
- Compilé (le compilateur se nomme javac)
- Interprété (l'interprète se nomme java)

Langage « à objets »

- Programmation impérative classique :

  - liste d'instructions pour le traitement de données

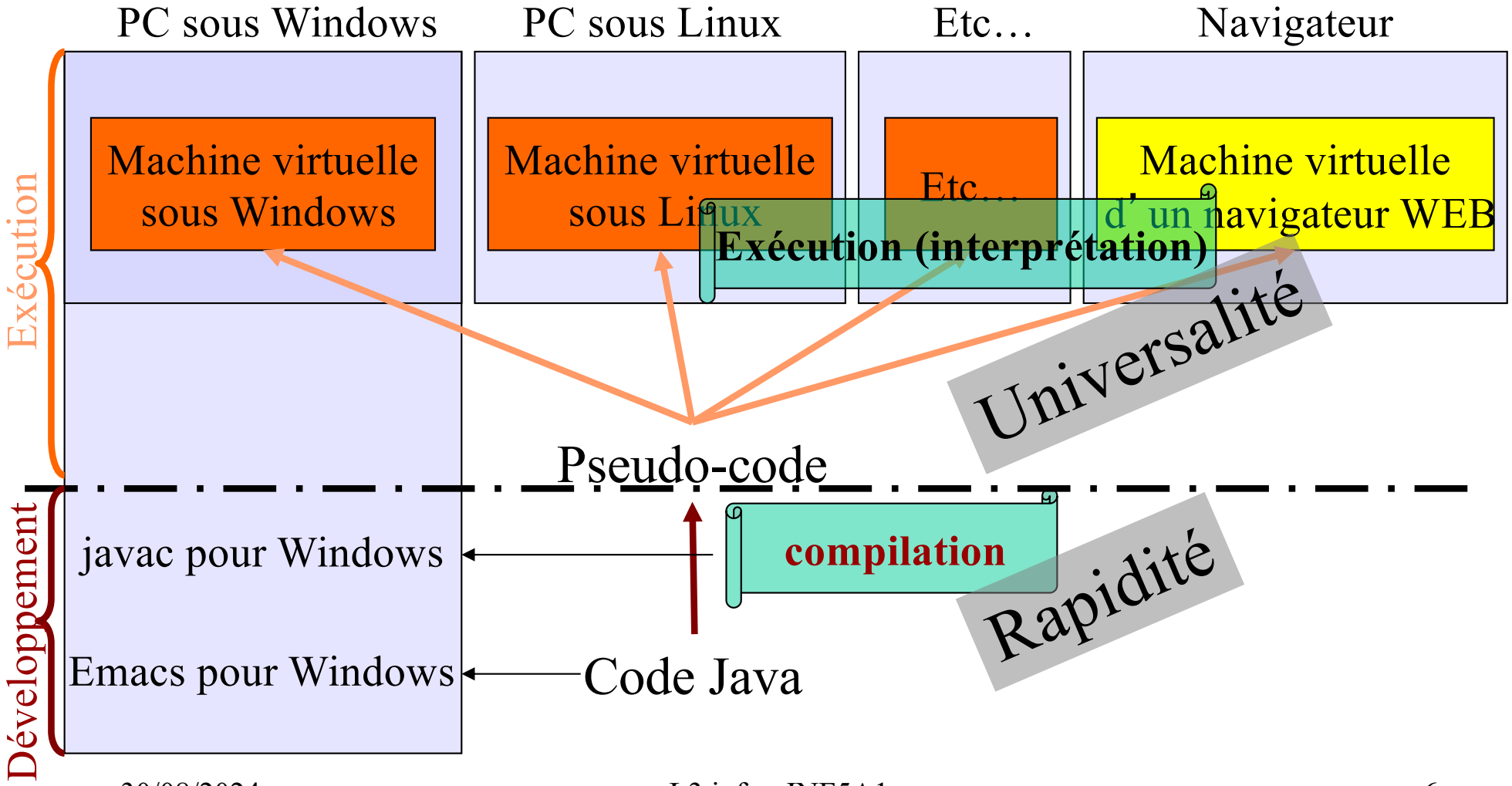
- Programmation « objet » (c++, java) :

  - description des données (propriétés et comportement) par l'intermédiaire de classes

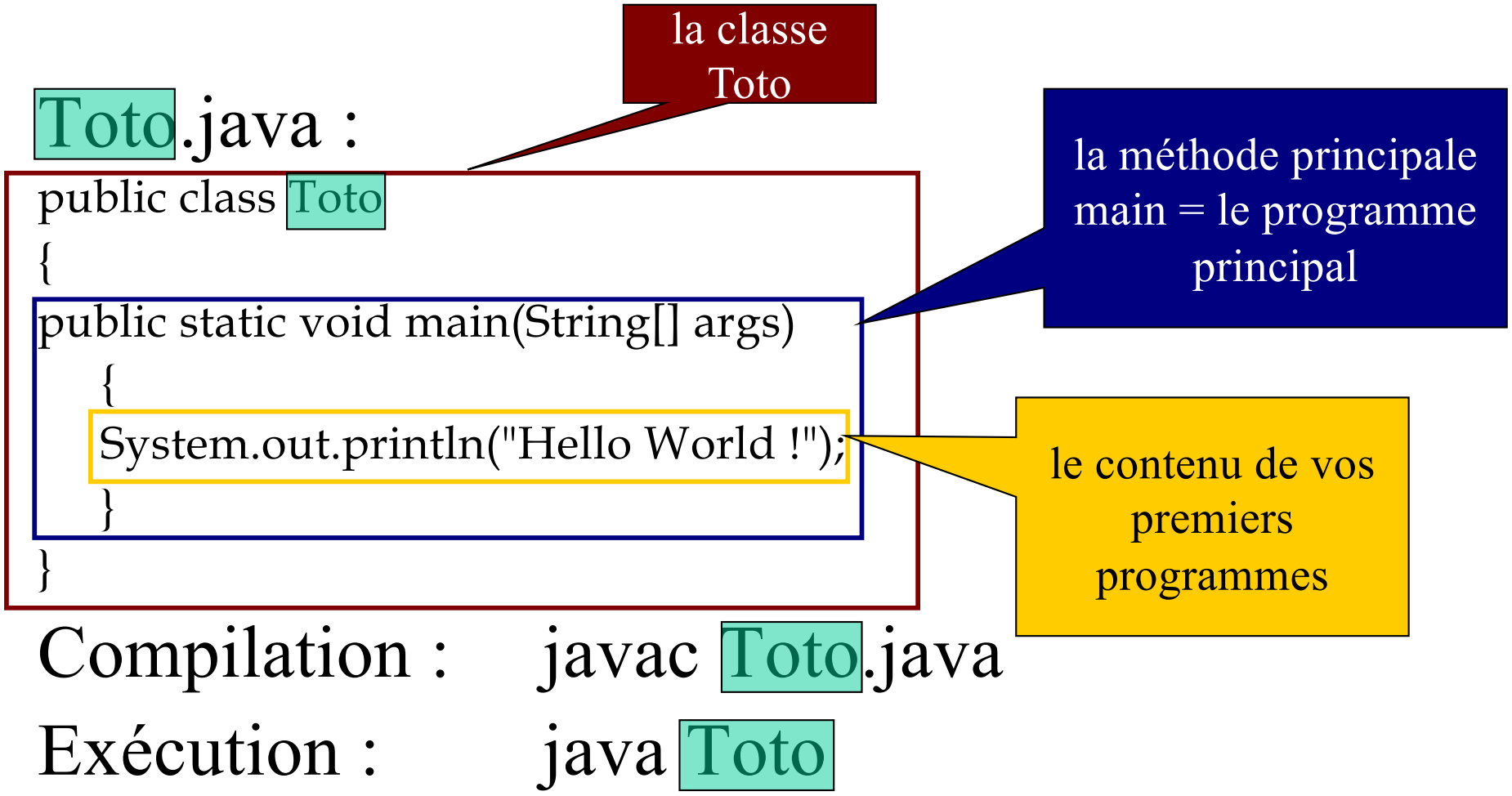
Similitudes et différences

- Similitudes : les éléments de programmation « classique » se retrouvent au sein de la description d'un comportement (~fonctions), donc au sein même des classes. (ex. : boucles « for », variables, etc.)
- Différences : la programmation objet permet d'améliorer notablement la lisibilité, la réutilisabilité (on fournit une classe d'objets qui sait faire telle ou telle chose), la factorisation de code via l'héritage (on part du général que l'on décline en différentes classes plus particulières)

Multi plate-forme, Code, Pseudo-code



Premier programme



Compilation : javac Toto.java

Exécution : java Toto

Eléments de programmation

les deux sortes de variables en Java

- les variables de type simple :

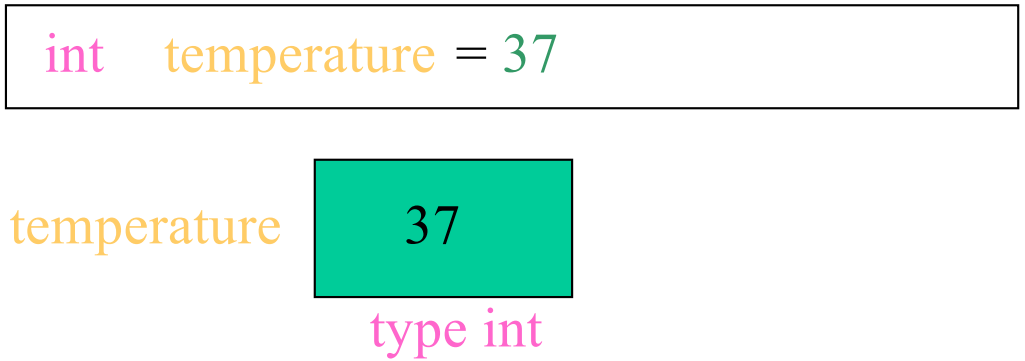
  une telle **variable stocke une valeur** de l'un des types prédéfinis. Elle est donc typée
- les variables **référençant** des objets :

  - elle référence des objets d'une classe particulière (ou de ses classes dérivées). Elle est donc typée

  - elle ne stocke pas le contenu de l'objet, mais **l'adresse mémoire où trouver cet objet**

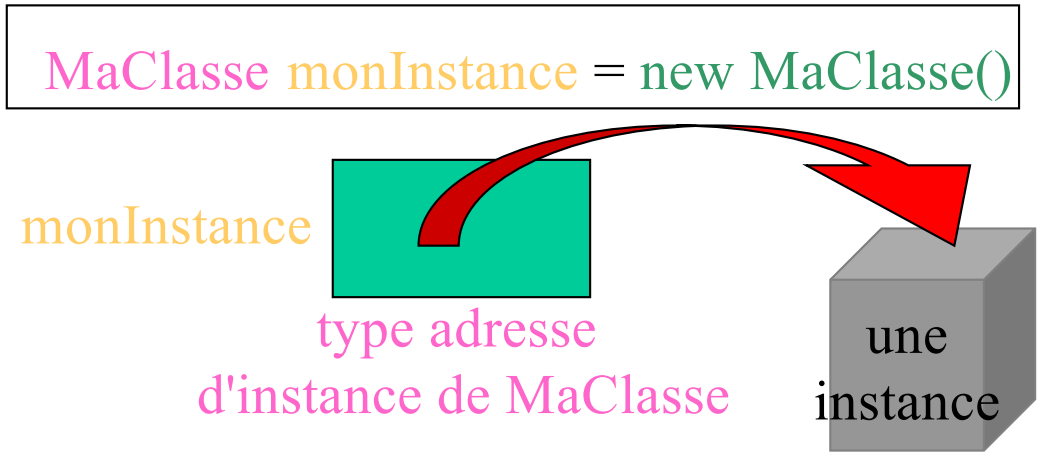
Variables simples et variables d'instance (2)

variable simple :
-un nom
-un type simple
-un contenu = une valeur



---

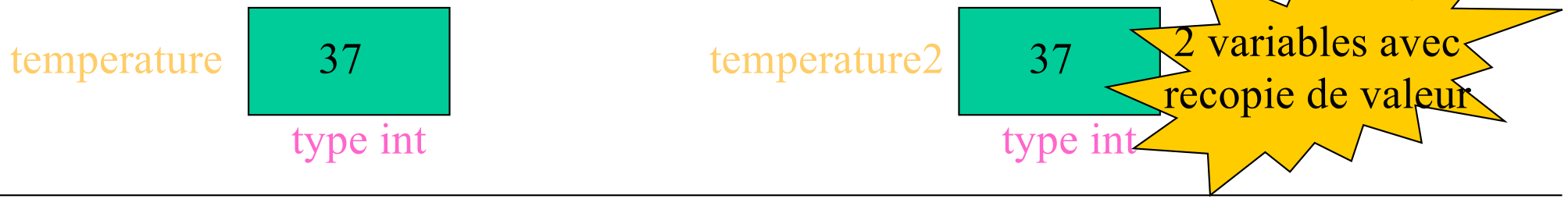
variable d'instance (d'objet) :
-un nom
-un type de classe
-un contenu = une adresse



conséquences de l'affectation

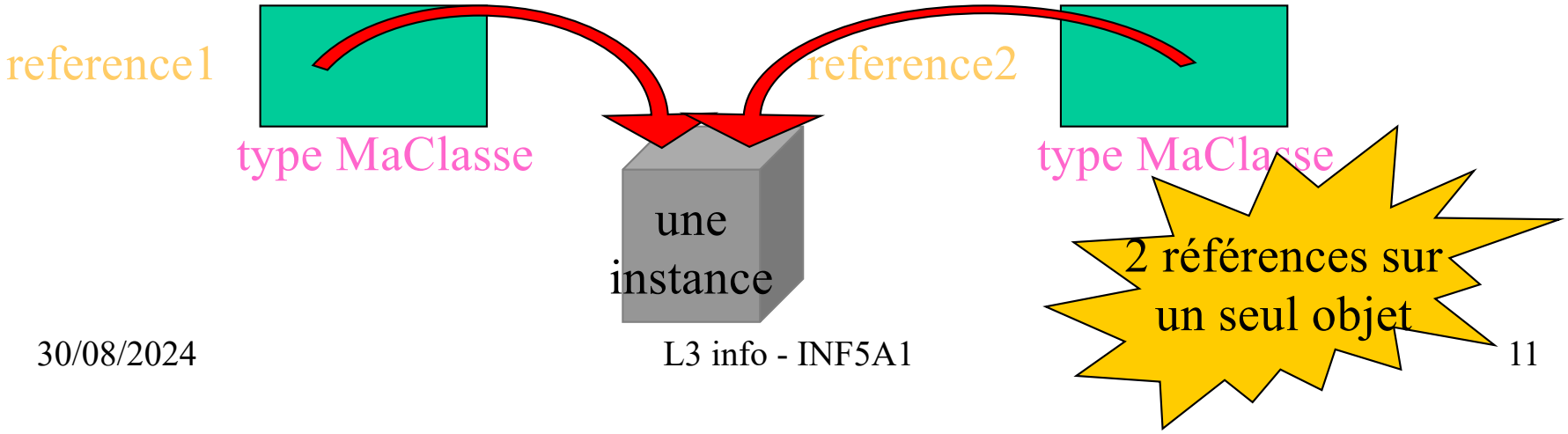
```
int    temperature = 37
```

```
int    temperature2= temperature
```




```
MaClasse  reference1 = new MaClasse()
```

```
MaClasse  reference2   = reference1
```



Types simples

Les types entiers

- `byte` `(1 octet)`
- `short` `(2 octets)`
- `int` `(4 octets)`
- `long` `(8 octets)`

Les types réels

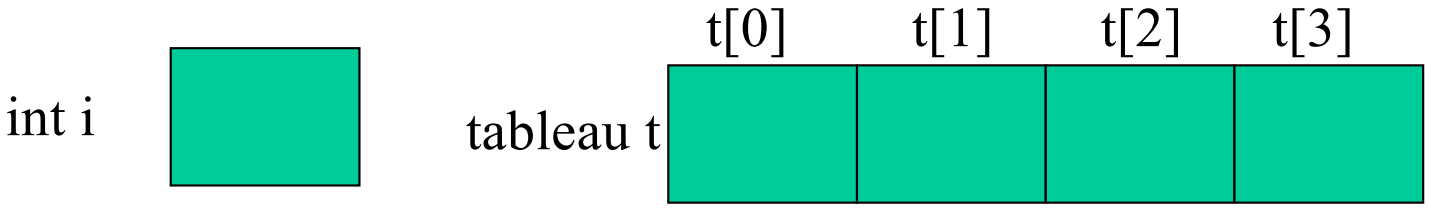
- `float` `(4 octets)`
- `double` `(8 octets)`

Booléen : `boolean (1 bit)`

Caractère : `char (2 octets)`

Tableaux

- Une variable classique (simple ou d'instance) permet de mémoriser une seule valeur ou référence.
- Lorsque l'on veut grouper sous un même nom un ensemble de variables, on utilise un tableau.
- Chaque élément du tableau est accessible par son indice, compris entre 0 et n-1, n étant la taille du tableau



Tableaux (2)

Pour créer un tableau, on utilise l'opérateur [].

2 Syntaxes équivalentes : typeDuTableau nomDuTableau[]
typeDuTableau[] nomDuTableau

Exemple :

int tabEntier[]; ou int[] tabEntier ;

Ceci définit une référence sur un tableau d'entiers

➔ pas de place réservée pour mettre les éléments du tableau.

Allocation mémoire : l'opérateur new

int[] tabEntier = new int[7] ; // 7 éléments dans le tableau

int tabEntier[ ]= new int[7];

Les tableaux ont un indice qui commence à 0.

Initialisation d'un tableau d'un type primitif : int[] table_entier = {1,5,9};

(Ou évidemment avec une boucle for)

Programmation Objet

- Description des données (et de leur comportement) par le biais de "classes"
- Une classe = un moule à objets
- Un objet = une instance = une entité créée à partir d'une certaine classe.

Instance (ou objet)

- une instance possède :

  - un état (attributs) cf. variables

  - un comportement (méthodes) cf. fonctions

- Attention : deux instances d'une même classe sont distinctes ; leurs états sont indépendants

Classe

- Une classe est une "usine à objets"
- C'est dans la classe que l'on définit les objets qu'elle va produire (ses données et son comportement).
- Définir une classe = définir un nouveau type de variable, plus complexe et plus intéressant que les types simples.

Définition d'une classe en Java

La définition d'une classe se compose :

```
class NomDeLaClasse
        {
        attributs
        constructeurs
        méthodes
        }
```

Les attributs et méthodes sont utilisables avant ou après leur définition

Plusieurs méthodes de même nom possibles (distinction par leurs paramètres différents)

➔c'est l'opération de **surcharge**

Remarques (convention) :

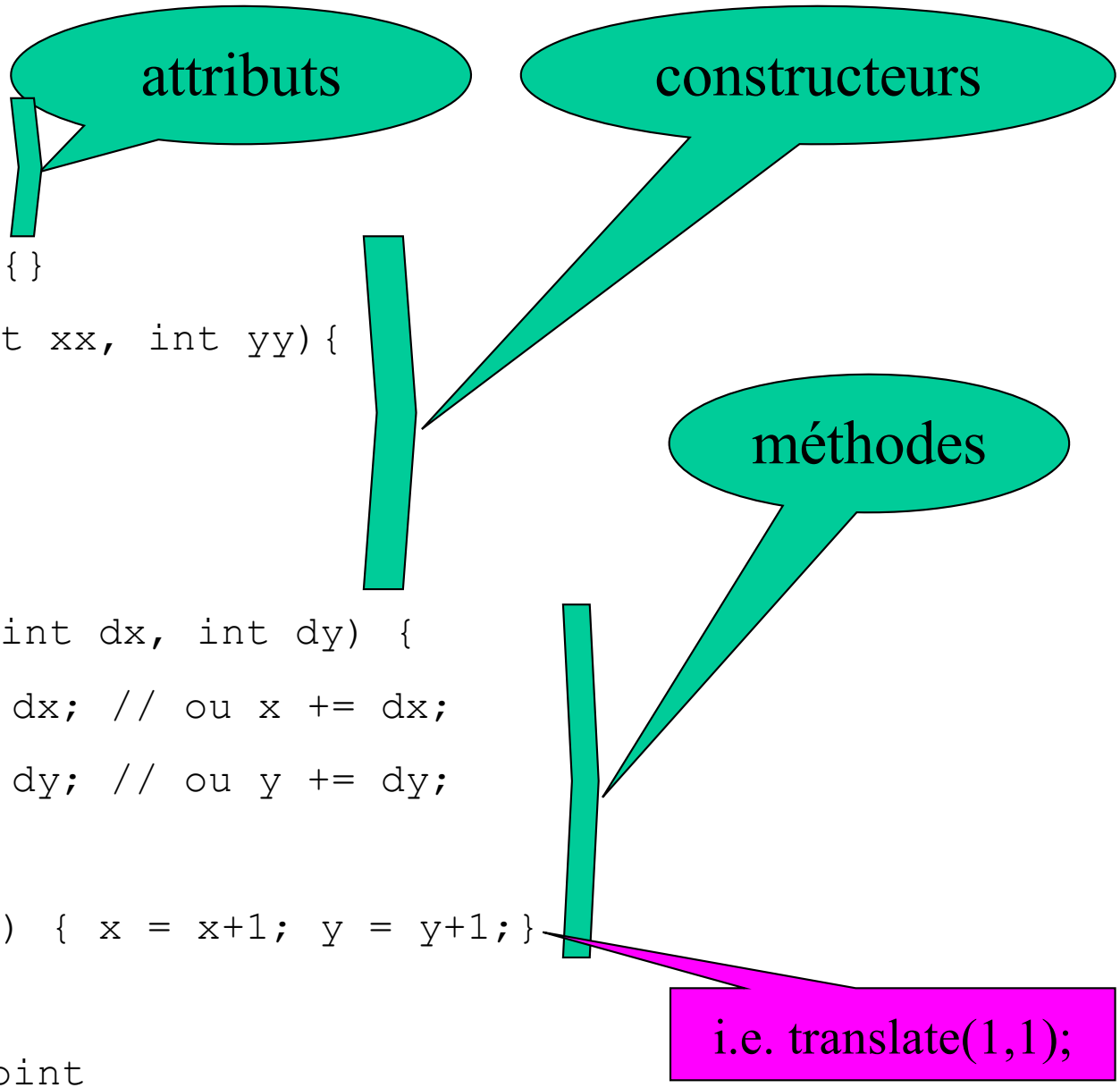
-nom d'une classe commence par une majuscule : `Point`

- nom d'une méthode par une minuscule : `translate()`
  autres mots composant le nom commencent par majuscule : `printPointsRouges()`
- constantes en majuscules : `PI`

Exemple Point.java

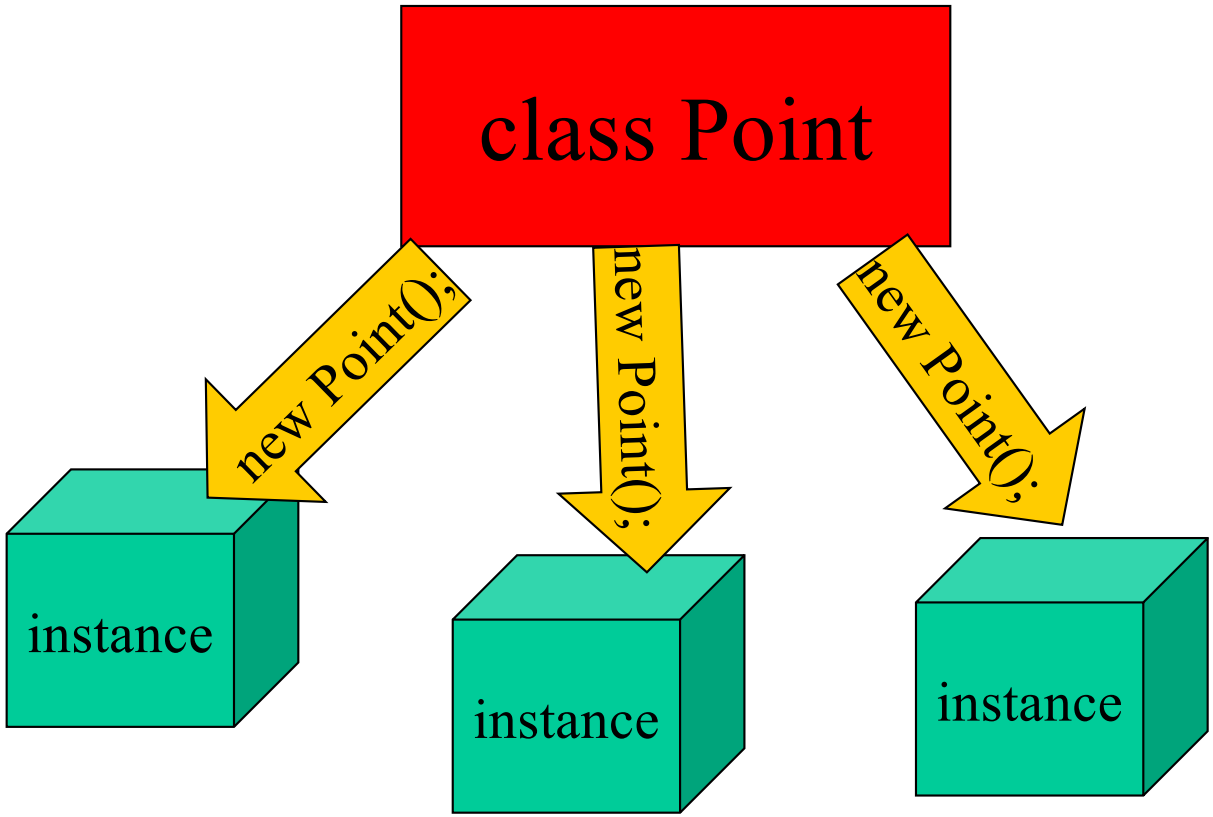
```
class Point {
        private int x;
        private int y;
        public Point() {}
        public Point(int xx, int yy){
         x=xx;
         y=yy;
       }
        void translate(int dx, int dy) {
                x = x + dx; // ou x += dx;
                y = y + dy; // ou y += dy;
        }
        void translate() { x = x+1; y = y+1;}

}// fin de la classe Point
```

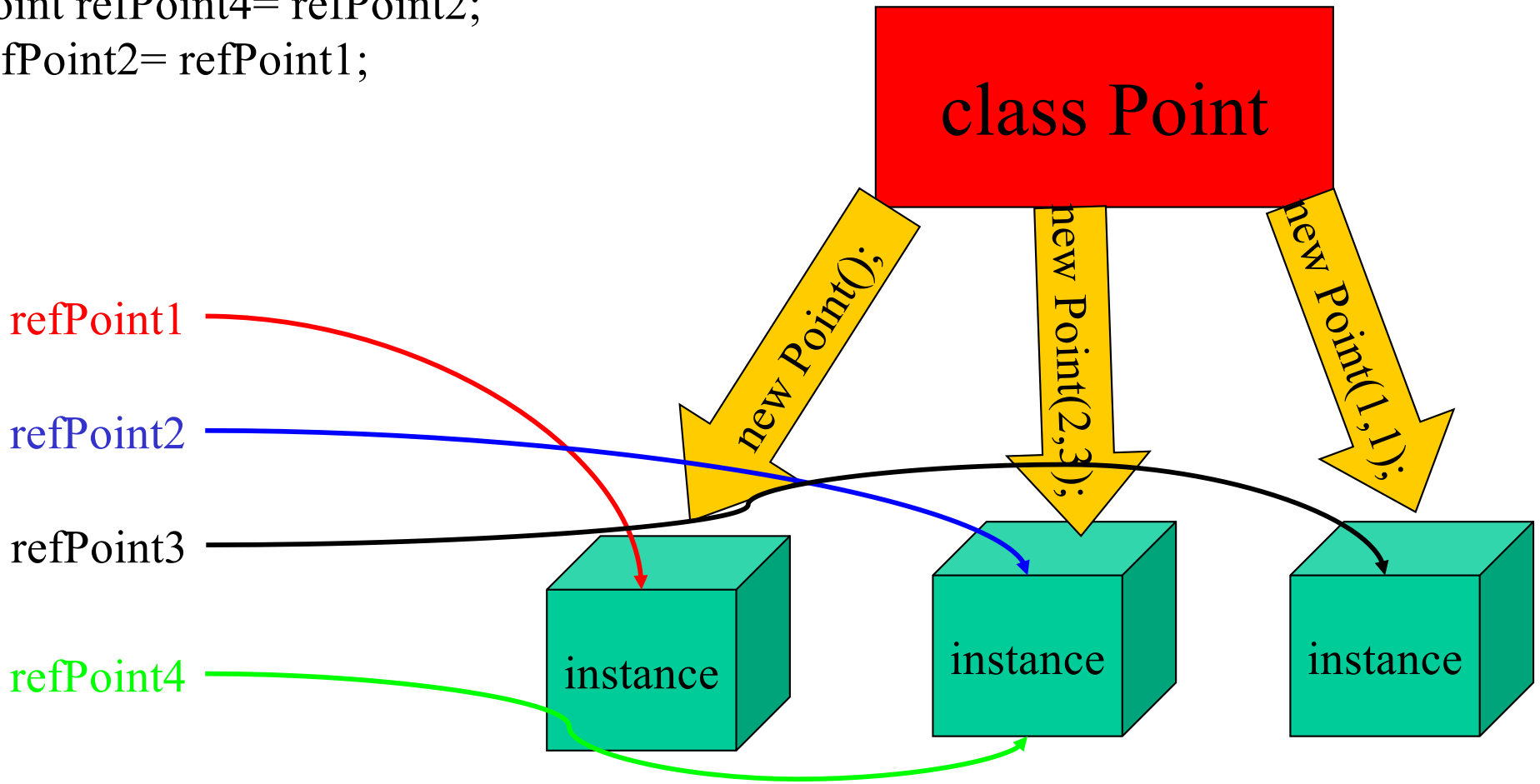


Création d'instances

- Opérateur : new
- Sauf exception, il y a autant d'instances créées que de fois où l'opérateur new est exécuté.
- Syntaxe = new NomDeLaClasse(…);
- NomDeLaClasse(…) est un constructeur de la classe, c'est-à-dire une méthode particulière portant le nom de la classe et dédiée à la création d'instances.


class Point
new Point();
new Point();
new Point();
instance
instance
instance

```
Point refPoint1 = new Point();
Point refPoint2= new Point(2,3);
Point refPoint3= new Point(1,1);
Point refPoint4= refPoint2;
refPoint2= refPoint1;
```



Classes versus Instances (objets)

- Une classe est chargée en mémoire (via un classLoader) dès que son nom apparaît dans le code. Elle existe préalablement à toutes ses éventuelles instances
- Une instance (un objet) n'existe qu'après qu'un appel à « new » est fait sur le constructeur d'une classe donnée

Exemple

- Point p; // Si non encore fait, classe chargée
- p = new Point(1,3); // une instance créée
- Point p2 = new Point(2, 2); // 2ème instance

static versus non-static

- Toute classe peut posséder ses propres membres (attributs, méthodes)
- Ces derniers sont déclarés « static ». Ils existent et sont accessibles dès le chargement de la classe, avant même que des instances ne soient créées.
- Ils sont utilisables directement au niveau de la classe, sans passer par les instances, et même en l'absence de toute instance
- Exemples : Math.PI, Math.sin(...), System.out, etc.
- Le bloc « static » permet d'initialiser les attributs static (voir exemple)

```
public class TableMultiplication
{
public static final int[][] TABLE;
public static final int TAILLE=11;
static
        {
        TABLE=new int[TAILLE][TAILLE];
        for (int i=0; i<TAILLE; i++)
                for (int int j=0; j<TAILLE; j++)
                        TABLE[i][j]=i*j;
        }
(...)
}
```

utilisation extérieure :

System.out.println(TableMultiplication.TABLE[2][3]); // → valeur 6

problème :

TableMultiplication.TABLE[2][3]= -2; // les valeurs ne sont pas « final »

```
public class TableMultiplication
{
private static final int[][] TABLE;
public static final int TAILLE=11;
static
        {
        TABLE=new int[TAILLE][TAILLE];
        for (int i=0; i<TAILLE; i++)
                for (int int j=0; j<TAILLE; j++)
                        TABLE[i][j]=i*j;
        }
public static int getValue(int i, int j)
        {
        return TABLE[i][j];
        }
}
```

utilisation extérieure :

System.out.println(TableMultiplication.getValue(i,j)); // → valeur 6

plus de problème :

TableMultiplication.TABLE[2][3]=5; // ne compile pas