Compléments de POO en Java

L2-MIM4A1

Année 2019/2020

Yann Mathet yann.mathet@unicaen.fr

Objectifs et chronologie

- Révisions sur la POO et sur les spécificités de Java.
- Generics
- Révisions et compléments sur « static »
- Gestion des exceptions
- Entrées/sorties
- Programmation d'applications avec interfaces graphiques en SWING
- Evaluation : Devoir de CC en groupes avec soutenance individuelle

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

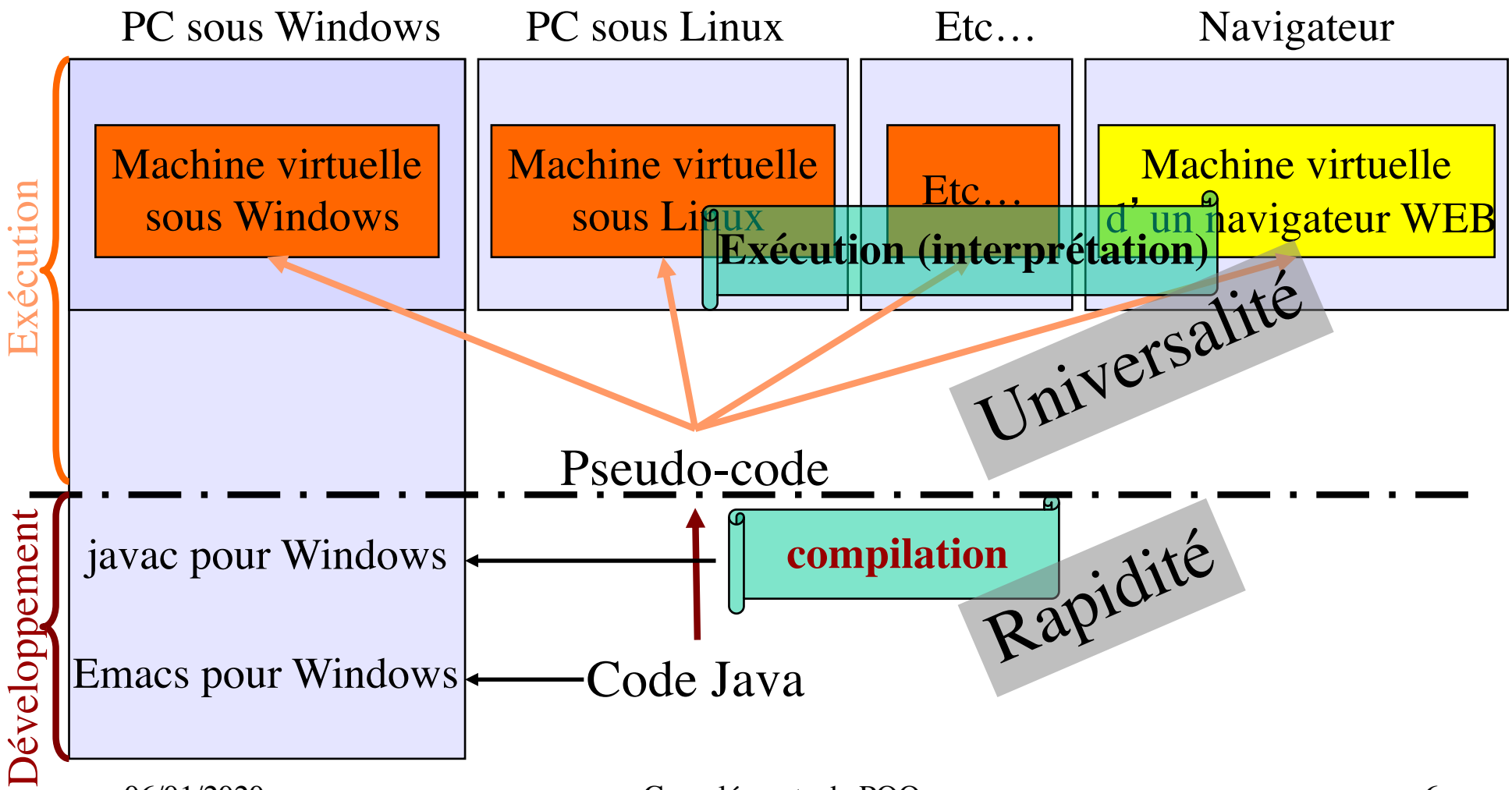
- Programmation « objet » (c++, java) :

  - 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



Canevas d'un programme

Toto.java :

```
public class Toto
{
public static void main(String[] args)
    {
    System.out.println("Hello World !");
    }
}
```

Compilation : javac Toto.java

Exécution : java Toto

Rappels sur des éléments de programmation

Variables : généralités

- Case mémoire dans laquelle on stocke une donnée, la variable est la base de tout traitement informatique (sans variable, un programme ferait invariablement la même chose).
- Les variables sont typées (entier, flottant, etc.) et ne peuvent stocker que des données compatibles avec leur type.
- Chaque variable porte un nom par lequel le programmeur peut accéder à la donnée. Le choix de ce nom est arbitraire, mais doit toujours commencer par une minuscule et ne pas contenir d'espaces.

les deux sortes de variables en Java

- les variables de type simple :

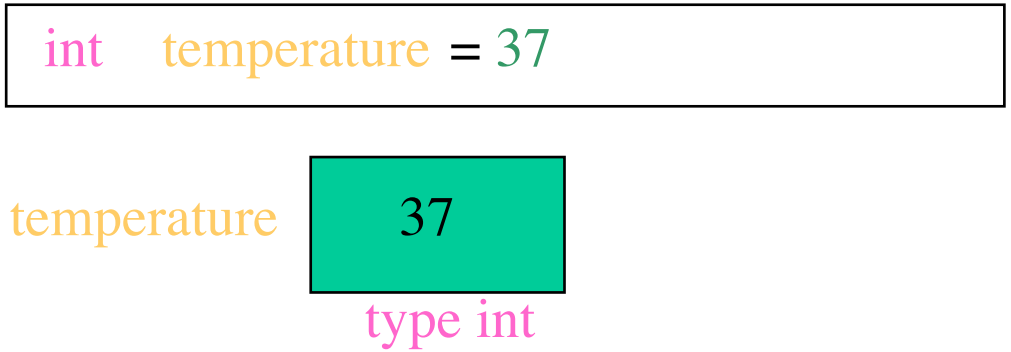
  une telle **variable stocke une valeur** de l'un des types prédéfinis. Elle est donc typée
- les variables référençant des objets :

  - 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**

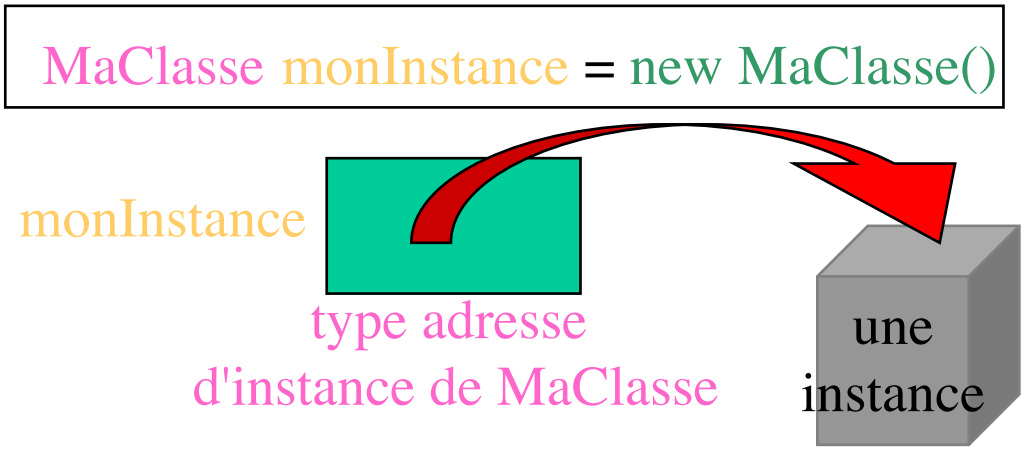
Variables simples et variables d'instance (2)

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



---

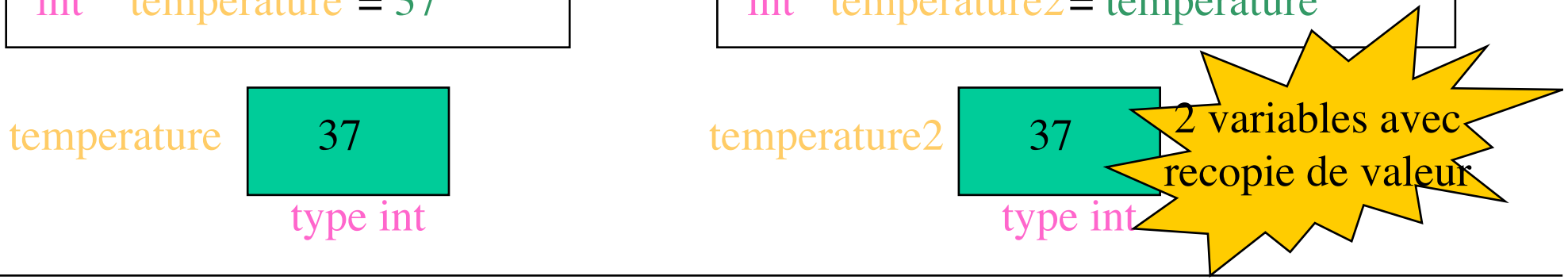
variable d'instance (d'objet) :
-un nom
-un type de classe
-un contenu = une adresse



conséquences de l'affectation

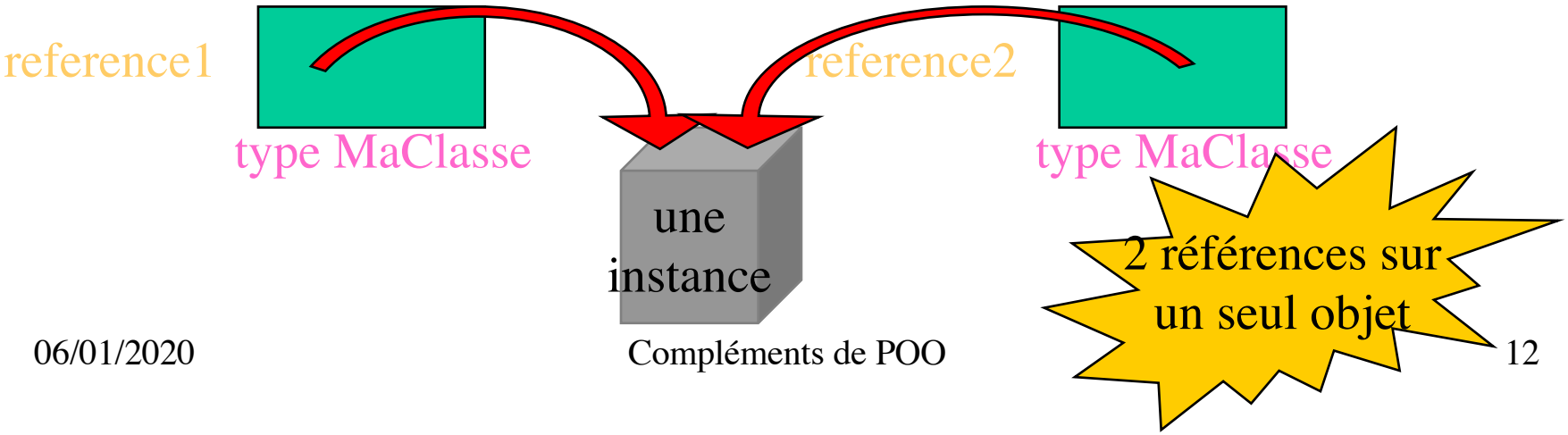
```
int    temperature = 37
```

```
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)

Structures de contrôle : l'alternative

booléen : true ou false

```
if (condition)
        instruction(s)1
else
        instruction(s)2
```

une seule instruction :
instuction;
Plusieurs instructions :
{
instruction1;
instruction2;
etc;
}

```
switch (expression)
        {
        case val1 : instruction11;
                    instruction12 ;
                    …
                    break;
        case val2 :….
        …
        case val_n :…
        default : instructions …;
        }
```

ATTENTION :
on met presque toujours "break" car dans le cas contraire, l'exécution continue (on applique les autres "case" jusqu'à la fin ou jusqu'au premier "break".

Structures itératives (boucles)

boucle "for"

- On connaît à l'avance le nombre d'itérations
- utilise un compteur qu'elle incrémente à chaque itération

boucle "while"

- On ne connaît pas à l'avance le nombre d'itération
- la boucle est répétée tant qu'une certaine condition est vraie (booléen).

boucle "for"

for (initialisation ; condition de continuation ; incrémentation)

instruction(s) à répéter

RAPPEL :
une seule instruction :
instuction;
Plusieurs instructions :
{
instruction1;
instruction2;
etc;
}

Exemple :

```
for (int i=0 ; i<10 ; i++)
        System.out.println("valeur de i = "+i);
```

opérateur de CONCATENATION (mise bout à bout de chaînes de caractères)

boucle "while"

```
while (condition booléenne)
        instruction(s) à répéter
```

Attention :
il est nécessaire de rendre possible l'évolution de la condition booléenne au sein de la boucle, faute de quoi, la boucle est infinie (boucle "folle").

Exemple :

```
while (x!=y)
        {
        System.out.println("Perdu");
        x=Lecture.readInt();
        }
```

CQFD :
x peut changer, donc la condition peut devenir vraie

Similitudes entre "for" et "while"

- En fait, toute boucle "for" peut être remplacée par une boucle "while", et réciproquement.
- Le choix de la boucle se fait donc sur le critère pratique : concision du code, facilité à interpréter
- Exercice : reprendre les deux boucles vues en exemple et les transformer en leur équivalent

Exemple de boucle for :

```
for (int i=0 ; i<10 ; i++)
            System.out.println("valeur de i = "+i);
```

Solution :

```
int i=0;
while (i<10)
    {
    System.out.println("valeur de i = "+i);
    i++;
    }
```

Exemple de boucle while :

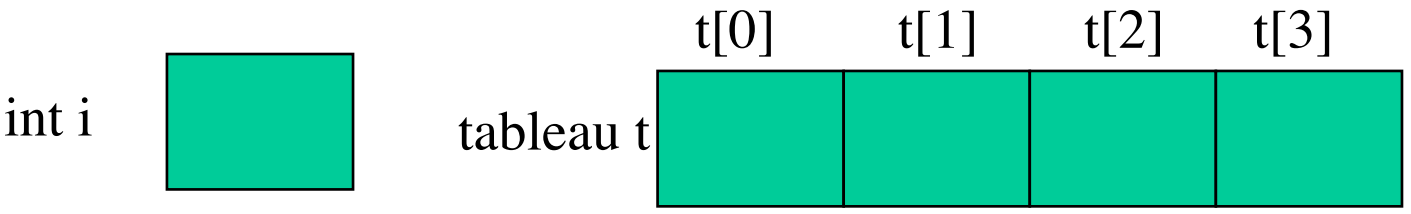
```
while (x!=y)
            {
            System.out.println("perdu");
            Lecture.readInt(x);
            }
```

Solution :

```
for (y=36 ; x!=y ; Lecture.readInt(x) )
            System.out.println("perdu");
```

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) :

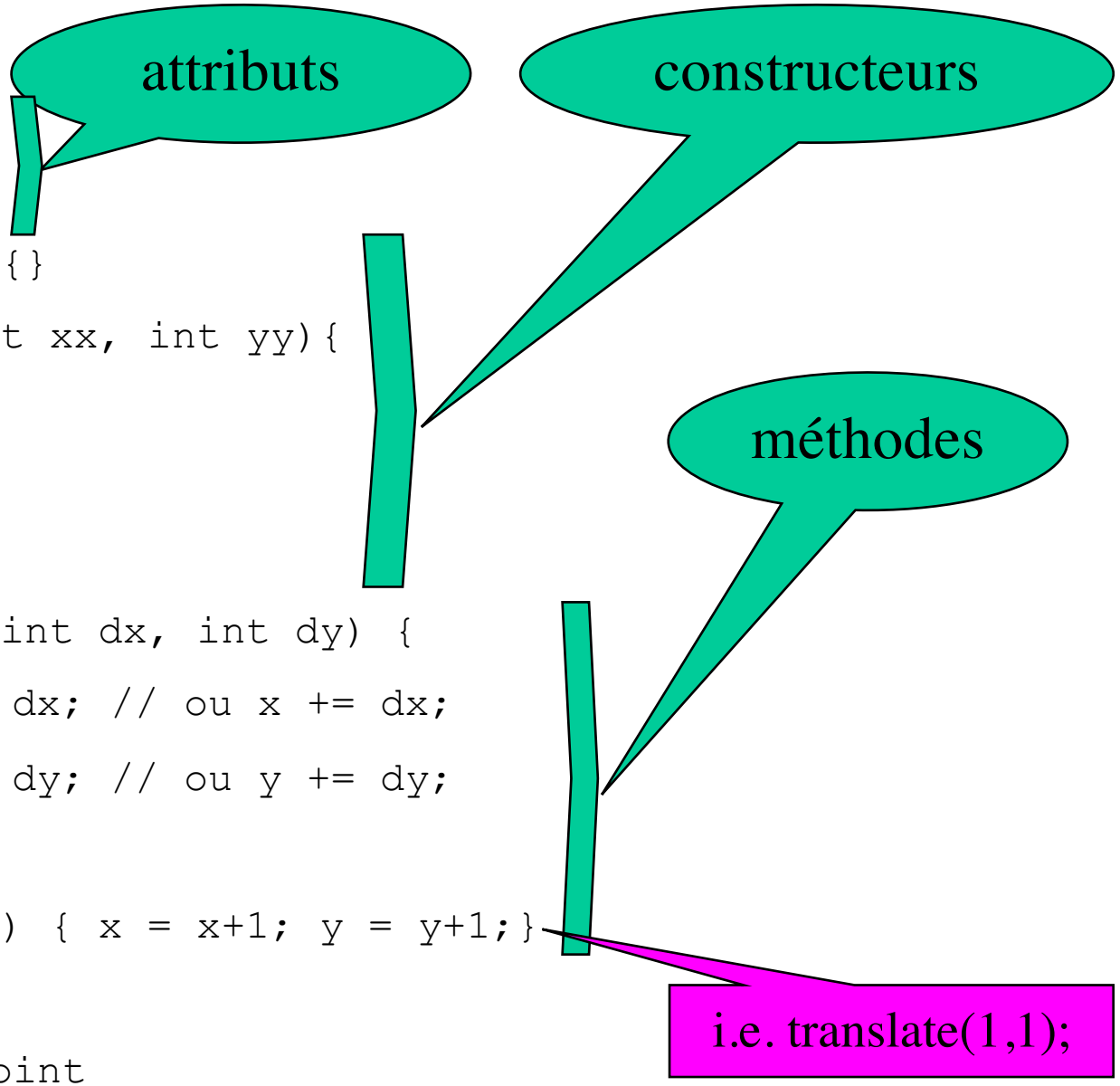
-nom d'une classe commence par une majuscule : `Point`

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

Exemple Point.java

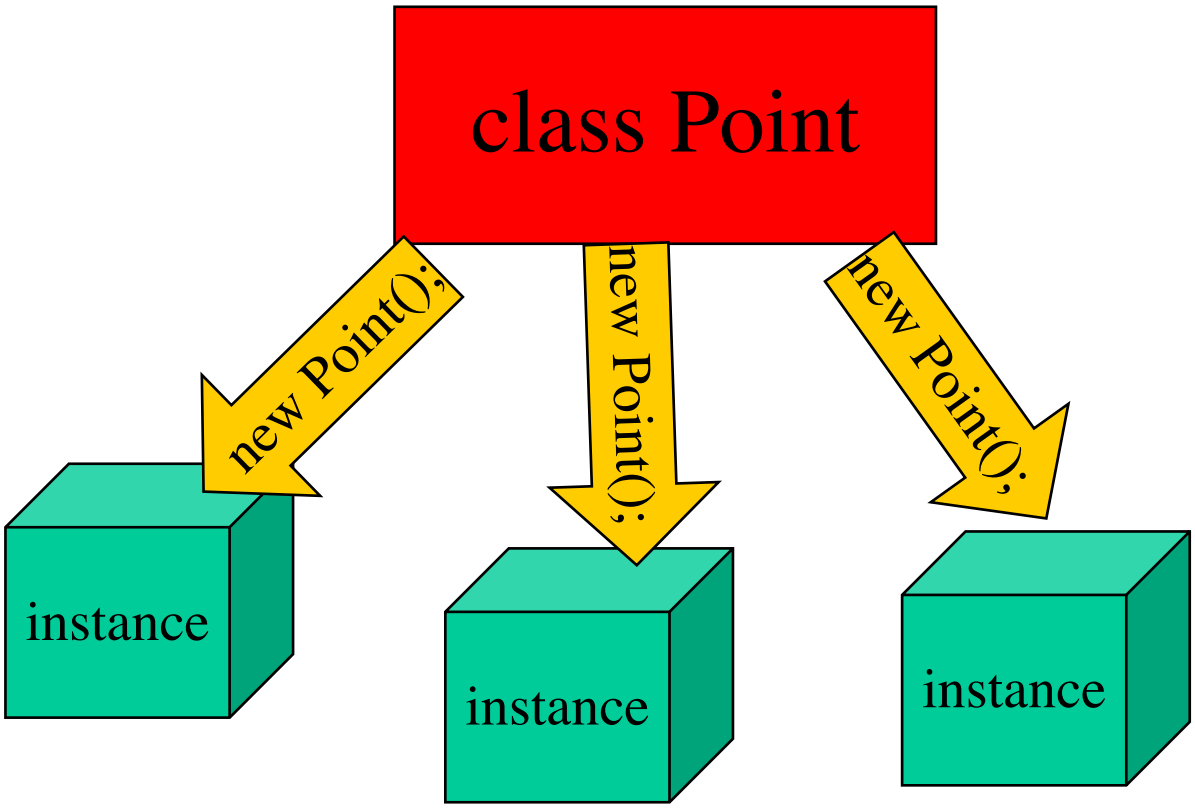
```
class Point {
        private int x;
        private int y;
        public Point() {}
        public Point(int xx, int yy){
         x=xx;
         y=yy;
       }
        void translate(int dx, int dy) {
                x = x + dx; // ou x += dx;
                y = y + dy; // ou y += dy;
      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

}// fin de la classe Point
```

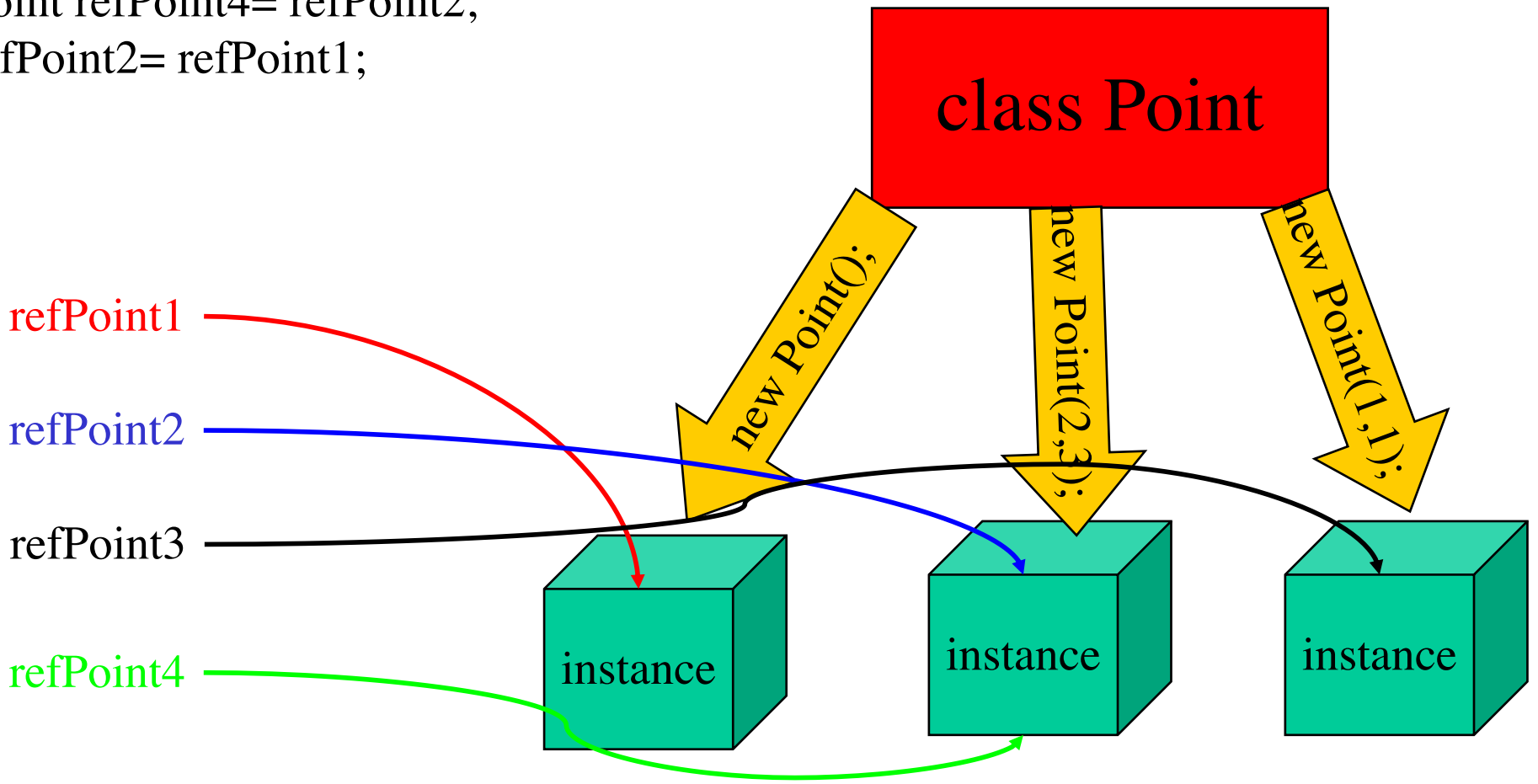


Création d'instances

- 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;
```



« static »

- Une classe est en fait elle-même un objet particulier qui est créé au chargement de la classe.
- La classe existe donc indépendamment des instances qu'elle peut créer, avant même que ces dernières soient créées.
- Elle possède ses propres membres (attributs, méthodes). On dit que ces membres sont « static »

« static » : quelques exemples classiques

- Math.PI
- Math.abs(-3)
- public static void main(String[] args)

Le bloc static{…}

- Lors du chargement de la classe, le bloc static{…}, s'il est présent dans la classe, est exécuté (une seule fois, jusqu'à la mort de la machine virtuelle).
- Il permet d'exécuter des instructions relatives aux membres statiques (et seulement à ces derniers)
- C'est un peu l'équivalent d'un constructeur
- Exemple : intialisation d'un tableau static

Héritage et Polymorphisme

- Quelques rappels fondamentaux…