

R1.01

INITIATION AU DÉVELOPPEMENT







Cours 10 : – passage de paramètres

Hervé Blanchon & Anne Lejeune




Université Grenoble Alpes

IUT 2 – Département Informatique

Plan de la séance


-  Avertissement
 -  Terminologie
 -  Paramètre effectif valeur immédiate ou variable de type primitif
 -  Paramètre effectif classe immuable
 -  Paramètre effectif classe muable ou `ArrayList<E>`
-
-  « classe muable » pourrait être remplacé par « classe altérable »

AVERTISSEMENT

-  Dans ce jeu de planche nous fournissons des schémas représentant l'état de la mémoire
-  ces schémas sont une simplification de ce qui se passe en réalité dans la machine virtuelle Java
-  ces schémas sont suffisants pour comprendre ce qui se passe sans entrer dans des détails qui n'ont pas leur place ici

TERMINOLOGIE

Paramètre formel

 L'entête de définition d'une fonction `static`, d'une procédure `static` ou d'une méthode de classe contient si nécessaire des paramètres formels

 Exemples :


 Procédure avec un **paramètre formel**

```
private/public static void afficheTableDe(int val) {  
    bloc d'instructions  
}
```

 Fonction avec deux **paramètres formels**

```
private/public static int minimum(int x, int y) {  
    bloc d'instructions  
}
```

Paramètre effectif ou argument

 Lors de l'appel une fonction `static`, une procédure `static` ou une méthode de classe qui a des paramètres formels, on utilise des paramètres effectifs

 Exemples :








 Appel de procédure avec un **paramètre effectif**

```
afficheTableDe(uneValeur);
```

 Appel d'une fonction avec deux **paramètres effectifs**

```
int val = minimum(5, 12);
```

Transmission (passage) de paramètre




-  La « transmission de paramètre » décrit ce qui se passe lors de l'appel d'une fonction, procédure ou méthode
 -  en d'autres termes comment un paramètre effectif est associé à son paramètre formel
-  En java les paramètres effectifs sont transmis par valeur
 -  lors d'un appel à une fonction, procédure ou méthode avec paramètre(s)
 -  la fonction, la procédure ou la méthode reçoit la valeur de chacun de ses paramètres effectifs
 -  une valeur pour chaque paramètre
-  *Ici, on va décrire ce qui se passe à très haut niveau sans entrer dans les détails techniques*

PARAMÈTRE EFFECTIF
VALEUR IMMÉDIATE OU VARIABLE DE
TYPE PRIMITIF


Paramètre effectif

- valeur immédiate
- variable de type primitif

Rappels

-  valeur immédiate : valeur écrite dans le code
 -  exemple : 12 (int), 12.5f (float), 12.5 (double)
-  types primitifs : byte, short, int, long, float, double

Lorsque le paramètre effectif est

- une **valeur immédiate**, *ou*
- une **variable initialisée de type primitif**
-  le **paramètre formel** se comporte comme une **variable locale** initialisée avec la **valeur du paramètre effectif**

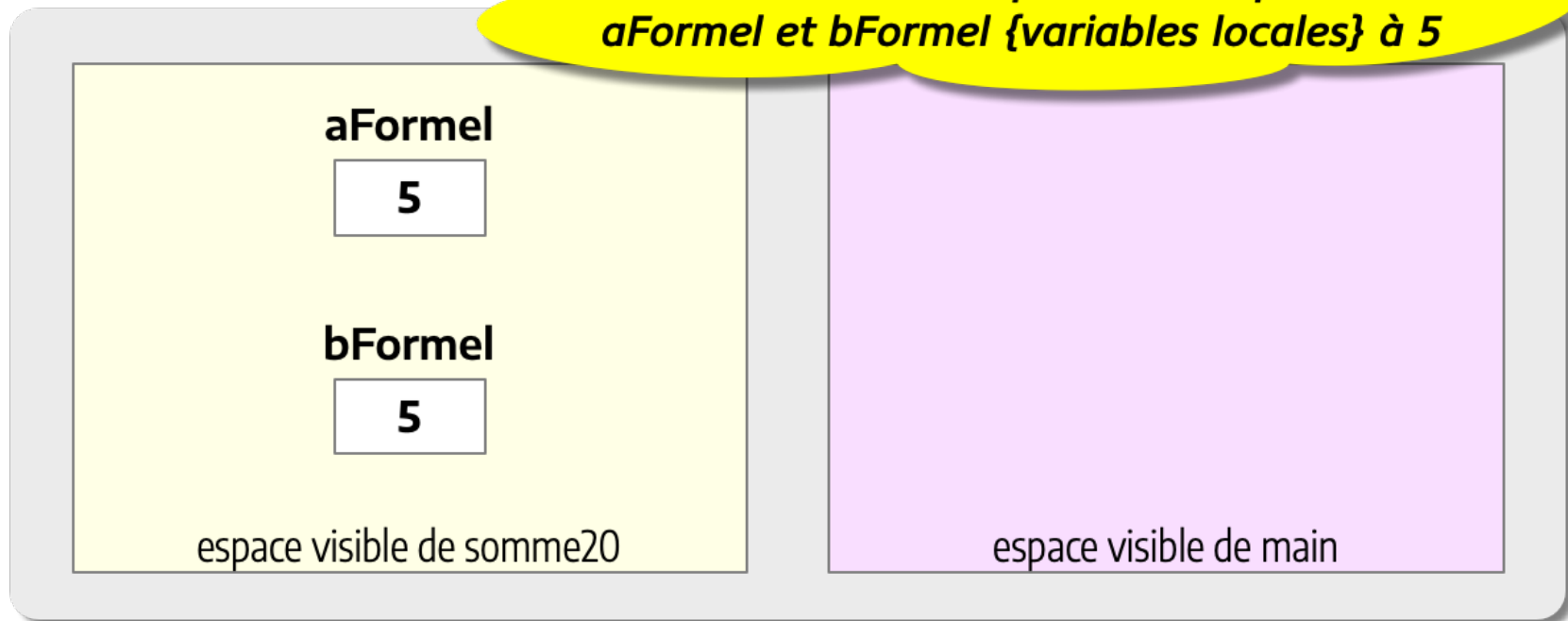
Exemple

```
1 public class TraceParametres {
2     private static int somme20(int aFormel, int bFormel) {
3         // aFormel et bFormel : variables locales à somme
4         // initialisées avec la valeur de leur paramètre effectif
5         aFormel = aFormel + 10;
6         bFormel = bFormel + 10;
7         return aFormel + bFormel;
8     }
9     public static void main(String[] args) {
10        // somme avec paramètres effectifs valeurs immédiates
11        System.out.println("somme20(5, 5) = " + somme20(5, 5));
12        int a = 5, b = 5;
13        // somme avec paramètres effectifs variables initialisées
14        System.out.println("a = " + a + " ; b = " + b);
15        System.out.println("somme20(a, b) = " + somme20(a, b));
16        System.out.println("a = " + a + " ; b = " + b);
17    }
18 }
```

11	somme20(5, 5) = 30	// somme() retourne 5+10 + 5+10
14	a = 5 ; b = 5	// valeur des variables locales à main()
15	somme20(a, b) = 30	// somme() retourne valeur_de a+10 + valeur_de b+10
16	a = 5 ; b = 5	// valeur des variables locales à main() non affectées

```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

appel de la fonction somme20
– *initialisation des paramètres formels*
aFormel et bFormel {variables locales} à 5



```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

exécution de la fonction somme20
– mise à jour des variables locales
aFormel et bFormel

aFormel

15

bFormel

15

espace visible de somme20

espace visible de main

```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

**retour dans la procédure main()
(somme()) a retourné 30)
– toute trace d'exécution de somme20()
est effacée : aFormel et bFormel « disparaissent »**

espace visible de somme20

espace visible de main

```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

déclaration et initialisation de deux
variables locales à la procédure main()
a et b

espace visible de somme

a

5

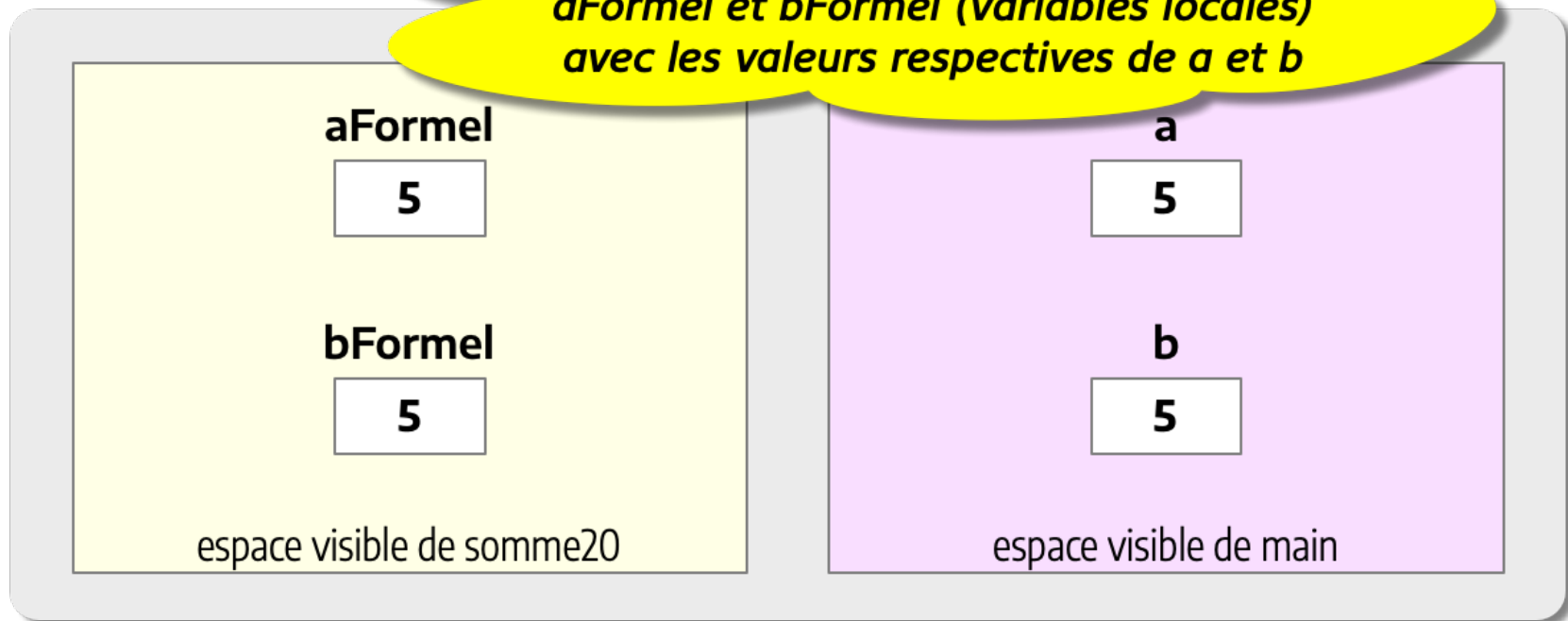
b

5

espace visible de main

```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

appel de la fonction somme20
– *initialisation des paramètres formels
aFormel et bFormel (variables locales)
avec les valeurs respectives de a et b*




```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

exécution de la fonction somme20
– mise à jour des variables locales
aFormel et bFormel

aFormel

15

bFormel

15

espace visible de somme20

a

5

b

5

espace visible de main

```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```

**retour dans la procédure main()
(somme()) a retourné 30)
– toute trace d'exécution de somme20()
est effacée : aFormel et bFormel « disparaissent »**

espace visible de somme20

a

5

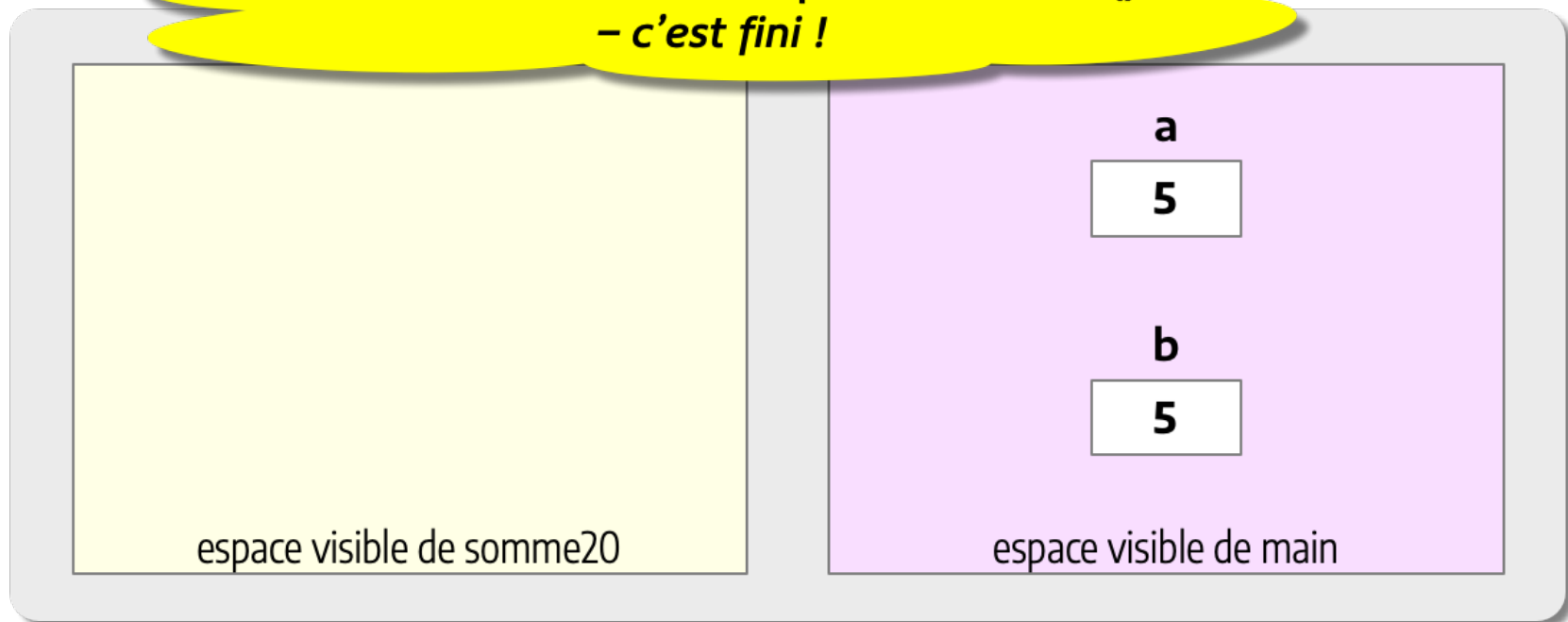
b

5

espace visible de main

```
public class TraceParametres {  
    private static int somme20(int aFormel, int bFormel) {  
        // aFormel et bFormel : variables locales à somme  
        // initialisées avec la valeur de leur paramètre effectif  
        aFormel = aFormel + 10;  
        bFormel = bFormel + 10;  
        return aFormel + bFormel;  
    }  
  
    public static void main(String[] args) {  
        // somme avec paramètres effectifs valeurs immédiates  
        System.out.println("somme20(5, 5) = " + somme20(5, 5));  
        int a = 5, b = 5;  
        // somme avec paramètres effectifs variables initialisées  
        System.out.println("a = " + a + " ; b = " + b);  
        System.out.println("somme20(a, b) = " + somme20(a, b));  
        System.out.println("a = " + a + " ; b = " + b);  
    }  
}
```





**dernière instruction de la procédure main()
– c'est fini !**




PARAMÈTRE EFFECTIF

CLASSE IMMuable

Rappel

-  classe immuable : la valeur des attributs ne peut être modifiée
-  classes immuables proposées par Java
 -  classes enveloppes : Byte, Short, Integer, Float, Double
 -  classe String

Lorsque le paramètre effectif est

- une **valeur immédiate de type classe immuable**
- une **variable initialisée de type classe immuable**
-  le **paramètre formel** se comporte comme une **variable locale initialisée avec la valeur du paramètre effectif**

Exemple

```
1 public class TraceParametresImmuable {
2     private static void afficheCapitalesAjoute10(String chFormel,
3                                                    Integer iFormel) {
4         chFormel = chFormel.toUpperCase();
5         System.out.println("chFormel en majuscules : " + chFormel);
6         iFormel = iFormel + 10;
7         System.out.println("iFormel plus 10 : " + iFormel);
8     }
9     public static void main(String[] args) {
10        String uneChaine = "Ceci Est Une Chaîne";
11        Integer i = 5;
12        System.out.println("uneChaine = " + uneChaine + "; i = " + i);
13        afficheCapitalesAjoute10(uneChaine, i);
14        System.out.println("uneChaine = " + uneChaine + "; i = " + i);
15    }
}
```

12	uneChaine = Ceci Est Une Chaîne; i = 5	// variables locales à main()
5	chFormel en majuscules : CECI EST UNE CHÂÎNE.	// chFormel passée en majuscules
7	iFormel + 10 : 15	// iFormel incrémenté de 10
15	uneChaine = Ceci Est Une Chaîne; i = 5	// variables locales à main() non affectées

```
public class TraceParametresImmuable {  
    private static void afficheCapitalesAjoute10(String chFormel,  
                                                Integer iFormel) {  
  
        dFormel = dFormel.toUpperCase();  
        System.out.println("dFormel en majuscules : " + dFormel);  
        iFormel = iFormel + 10;  
        System.out.println("iFormel plus 10 : " + iFormel);  
    }  
  
    public static void main(String[] args) {  
        String uneChaine = "Ceci Est Une Chaîne";  
        Integer i = 5;  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
        afficheCapitalesAjoute10(uneChaine, i);  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
    }  
}
```

déclaration et initialisation de deux
variables locales à la procédure main()
uneChaine et i

espace visible de afficheCapitalesAjoute10

uneChaine



"Ceci Est Une Chaîne"

i

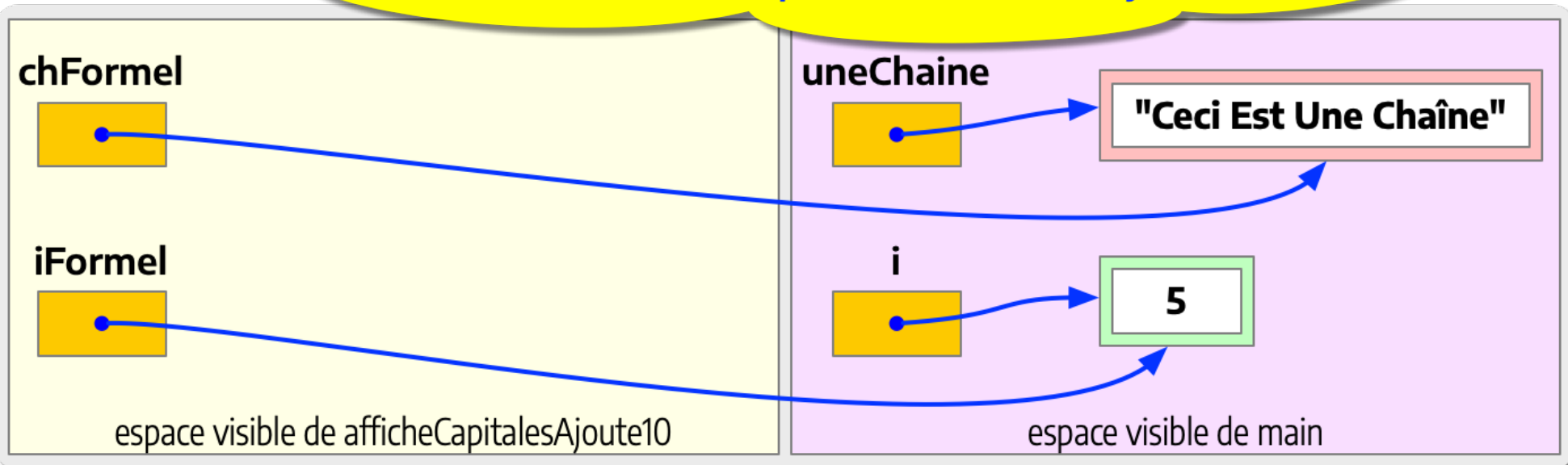


5

espace visible de main

```
public class TraceParametresImmuable {  
    private static void afficheCapitalesAjoute10(String chFormel,  
                                                Integer iFormel) {  
  
        dFormel = dFormel.toUpperCase();  
        System.out.println("dFormel en majuscules : " + dFormel);  
        iFormel = iFormel + 10;  
        System.out.println("iFormel plus 10 : " + iFormel);  
    }  
  
    public static void main(String[] args) {  
        String uneChaine = "Ceci Est Une Chaîne";  
        Integer i = 5;  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
        afficheCapitalesAjoute10(uneChaine, i);  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
    }  
}
```

appel de la procédure afficheCapitalesAjoute10()
– initialisation des paramètres formels chFormel et iFormel
(variables locales) avec les valeurs respectives de uneChaine et i
(ces valeurs sont des références {pointeurs})
– uneChaine et chFormel pointent sur le même objet !
– i et iFormel pointent sur le même objet !




```
public class TraceParametres {  
    private static void afficheCapitalesAjoute10(String chFormel,  
                                                Integer iFormel) {  
        chFormel = chFormel.toUpperCase();  
        System.out.println("chFormel en majuscules : " + chFormel);  
        iFormel = iFormel + 10;  
        System.out.println("iFormel plus 10 : " + iFormel);  
    }  
    public static void main(String[] args) {  
        String uneChaine = "Ceci Est Une Chaîne";  
        Integer i = 5;  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
        afficheCapitalesAjoute10(uneChaine, i);  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
    }  
}
```

exécution de la procédure `afficheCapitalesAjoute10()`
– étape 1 : mise à jour de `chFormel` qui pointe sur un objet
local à `afficheCapitalesAjoute10()`

chFormel



"CECI EST UNE CHAÎNE"

iFormel



espace visible de `afficheCapitalesAjoute10`

uneChaine



"Ceci Est Une Chaîne"

i



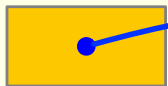
5

espace visible de `main`

```
public class TraceParametresImmuable {  
    private static void afficheCapitalesAjoute10(String chFormel,  
                                                Integer iFormel) {  
  
        dFormel = dFormel.toUpperCase();  
        System.out.println("dFormel en majuscules : " + dFormel);  
        iFormel = iFormel + 10;  
        System.out.println("iFormel plus 10 : " + iFormel);  
    }  
  
    public static void main(String[] args) {  
  
        String uneChaine = "Ceci Est Une Chaîne";  
        Integer i = 5;  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
        afficheCapitalesAjoute10(uneChaine, i);  
  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
    }  
}
```

exécution de la procédure afficheCapitalesAjoute10()
– étape 2 : mise à jour de iFormel qui pointe sur un objet local à afficheCapitalesAjoute10()

chFormel



"CECI EST UNE CHAÎNE"

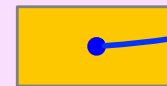
iFormel



15

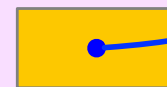
espace visible de afficheCapitalesAjoute10

uneChaine



"Ceci Est Une Chaîne"

i

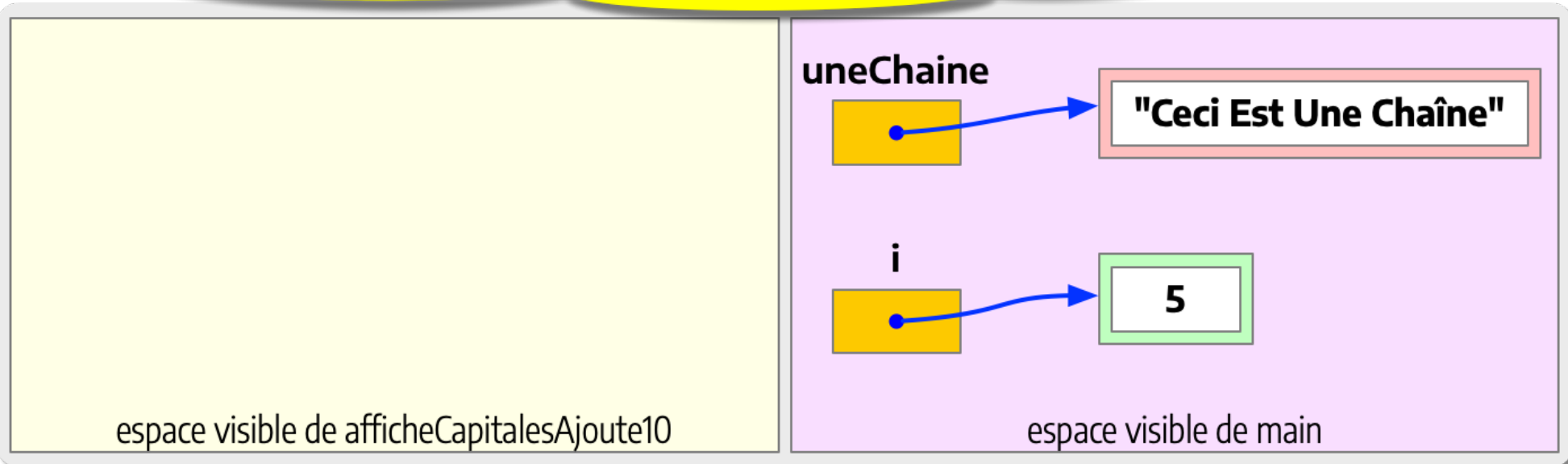


5

espace visible de main

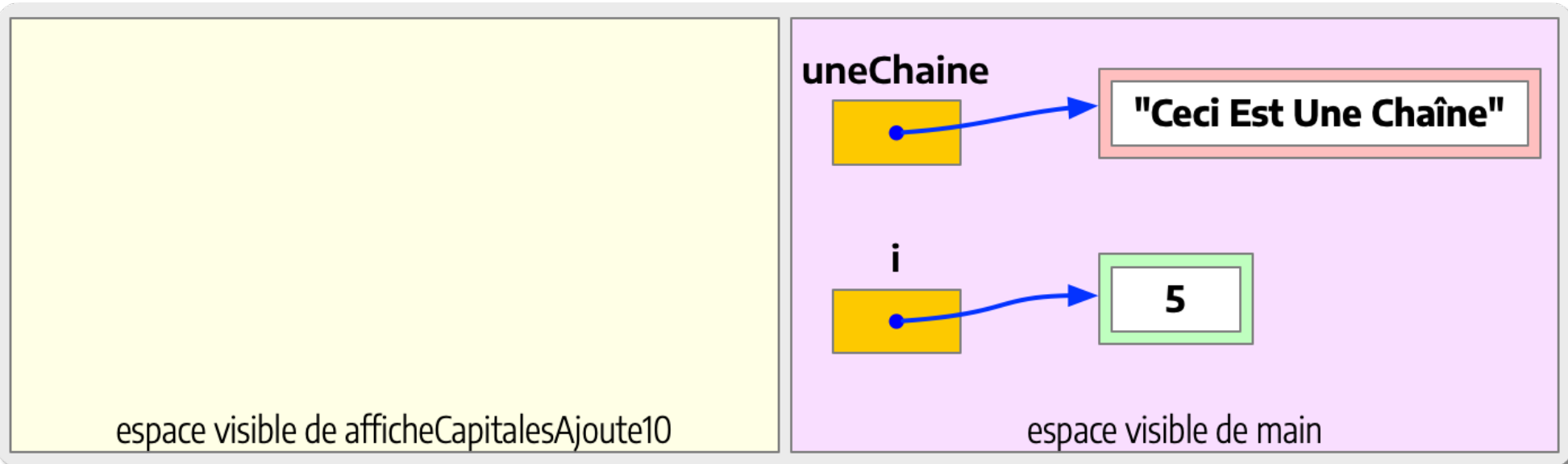
```
public class TraceParametresImmuable {  
    private static void afficheCapitalesAjoute10(String chFormel,  
                                                Integer iFormel) {  
  
        dFormel = dFormel.toUpperCase();  
        System.out.println("dFormel en majuscules : " + dFormel);  
        iFormel = iFormel + 10;  
        System.out.println("iFormel plus 10 : " + iFormel);  
    }  
  
    public static void main(String[] args) {  
        String uneChaine = "Ceci Est Une Chaîne";  
        Integer i = 5;  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
        afficheCapitalesAjoute10(uneChaine, i);  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
    }  
}
```

retour dans la procédure main()
– toute trace d'exécution de afficheCapitalesAjoute10()
est effacée : chFormel et iFormel « disparaissent »



```
public class TraceParametresImmuable {  
    private static void afficheCapitalesAjoute10(String chFormel,  
                                                Integer iFormel) {  
  
        dFormel = dFormel.toUpperCase();  
        System.out.println("dFormel en majuscules : " + dFormel);  
        iFormel = iFormel + 10;  
        System.out.println("iFormel plus 10 : " + iFormel);  
    }  
  
    public static void main(String[] args) {  
        String uneChaine = "Ceci Est Une Chaîne";  
        Integer i = 5;  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
        afficheCapitalesAjoute10(uneChaine, i);  
  
        System.out.println("uneChaine = " + uneChaine + "; i = " + i);  
    }  
}
```

dernière instruction de la procédure main()
– *c'est fini !*



PARAMÈTRE EFFECTIF


CLASSE MUABLE OU `ArrayList<E>`

Paramètre effectif

- classe muable
- ArrayList


Rappel

 classe muable : au moins un attribut avec setter

 Lorsque le paramètre effectif est

- une **variable initialisée de type classe muable**
- une **variable initialisée de type ArrayList**

 le **paramètre formel** se comporte comme une **variable partagée avec l'appelant**

 le paramètre formel a le même comportement que le paramètre effectif

PARAMÈTRE EFFECTIF

CLASSE MUABLE

Classe Point (Rappel)

```
public class Point {
    // attributs
    private int x;
    private int y;
    // constructeurs
    public Point() {x = 0; y = 0;}
    public Point(int x, int y) {this.x = x; this.y = y;}

    // getters et setters des attributs
    public int getX() {return x;}
    public void setX(int x) {this.x = x;}
    public int getY() {return y;}
    public void setY(int y) {this.y = y;}

    // déplacement
    public void deplace(int dx, int dy) {
        x = x + dx; y = y + dy;
    }

    // affichage mathématique avec printXX
    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```


Exemple avec une classe muable

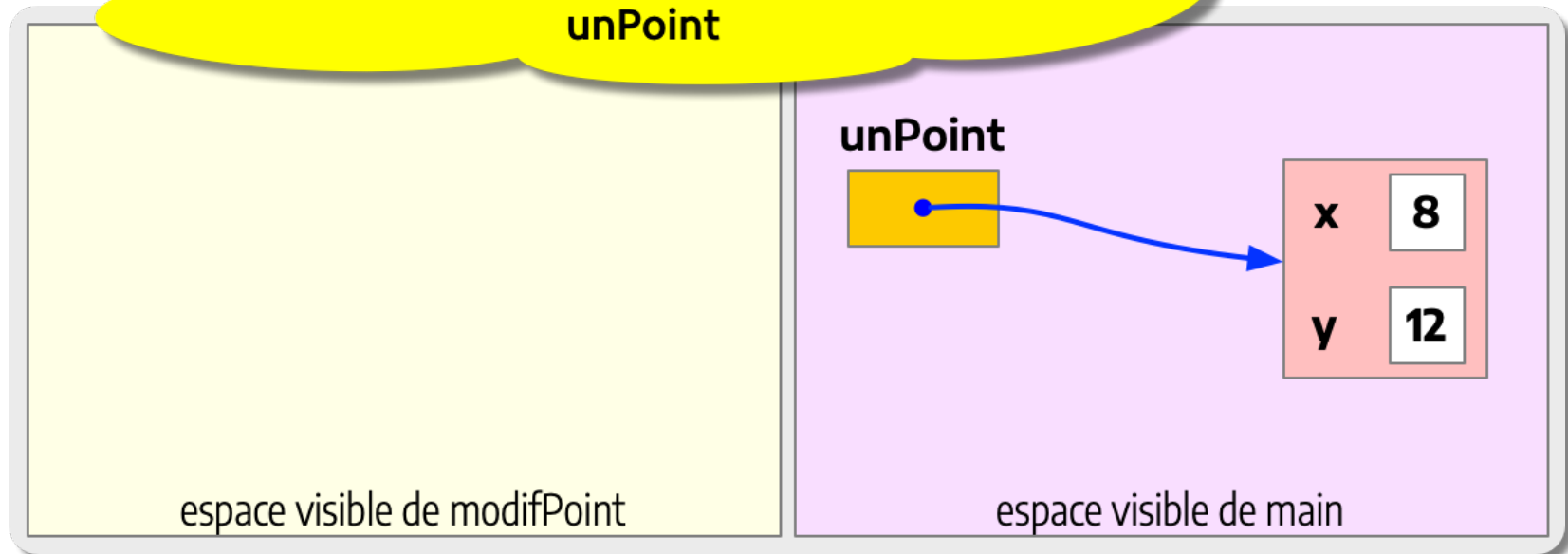
```
1 public class TraceParametresMuable {
2
3     private static void modifPoint(Point pFormel) {
4         // ajouter 10 en abscisse et en ordonnée
5         pFormel.deplace(10, 10);
6     }
7
8     public static void main(String[] args) {
9         Point unPoint = new Point(8, 12);
10        System.out.println("unPoint avant modifPoint = " + unPoint);
11        modifPoint(unPoint);
12        System.out.println("unPoint après modifPoint = " + unPoint);
13    }
14 }
```

10 unPoint avant modifPoint = (8, 12)

12 unPoint après modifPoint = (18, 22) // la variable unPoint de main a été modifiée
// par la procédure modifPoint

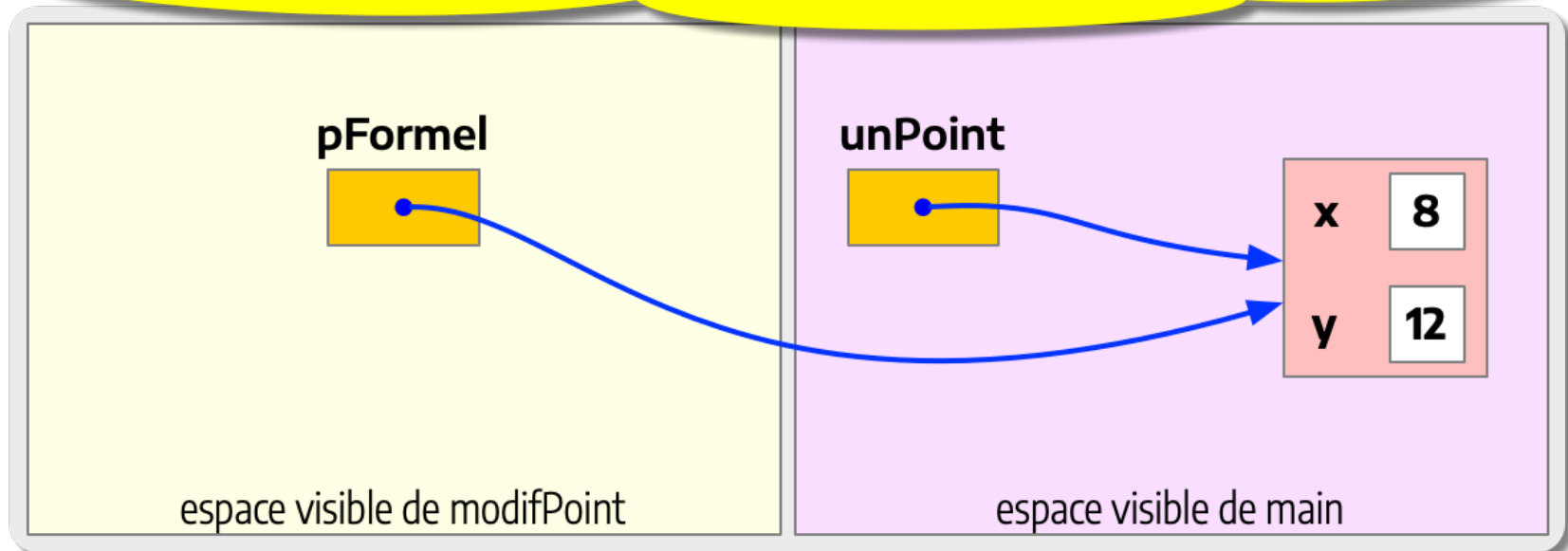
```
public class TraceParametresMuable {  
  
    private static void modifPoint(Point pFormel) {  
  
        // ajouter 10 en abscisse et en ordonnée  
        pFormel.deplace(10, 10);  
    }  
  
    public static void main(String[] args) {  
        Point unPoint = new Point(8, 12);  
        System.out.println("unPoint avant modifPoint = " + unPoint);  
  
        modifPoint(unPoint);  
        System.out.println("unPoint après modifPoint = " + unPoint);  
    }  
}
```

déclaration et initialisation d'une variable
locale à la procédure main()
unPoint



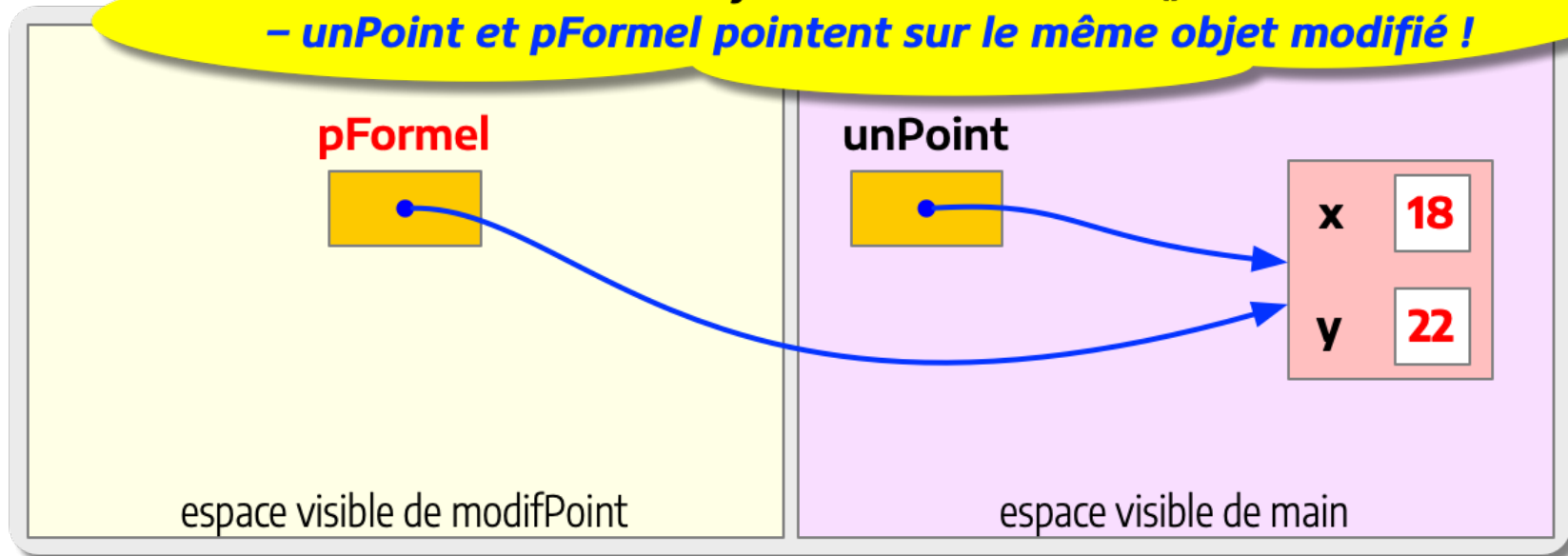
```
public class TraceParametresMuable {  
  
    private static void modifPoint(Point pFormel) {  
        // ajouter 10 en abscisse et en ordonnée  
        pFormel.deplace(10, 10);  
    }  
  
    public static void main(String[] args) {  
        Point unPoint = new Point(8, 12);  
        System.out.println("unPoint avant modifPoint = " + unPoint);  
        modifPoint(unPoint);  
        System.out.println("unPoint après modifPoint = " + unPoint);  
    }  
}
```

- appel de la procédure modifPoint()
- initialisation du paramètre formel *pFormel* (variable locale) avec la valeur de *unPoint* (cette valeur est une référence {pointeur})
 - *unPoint* et *pFormel* pointent sur le même objet !



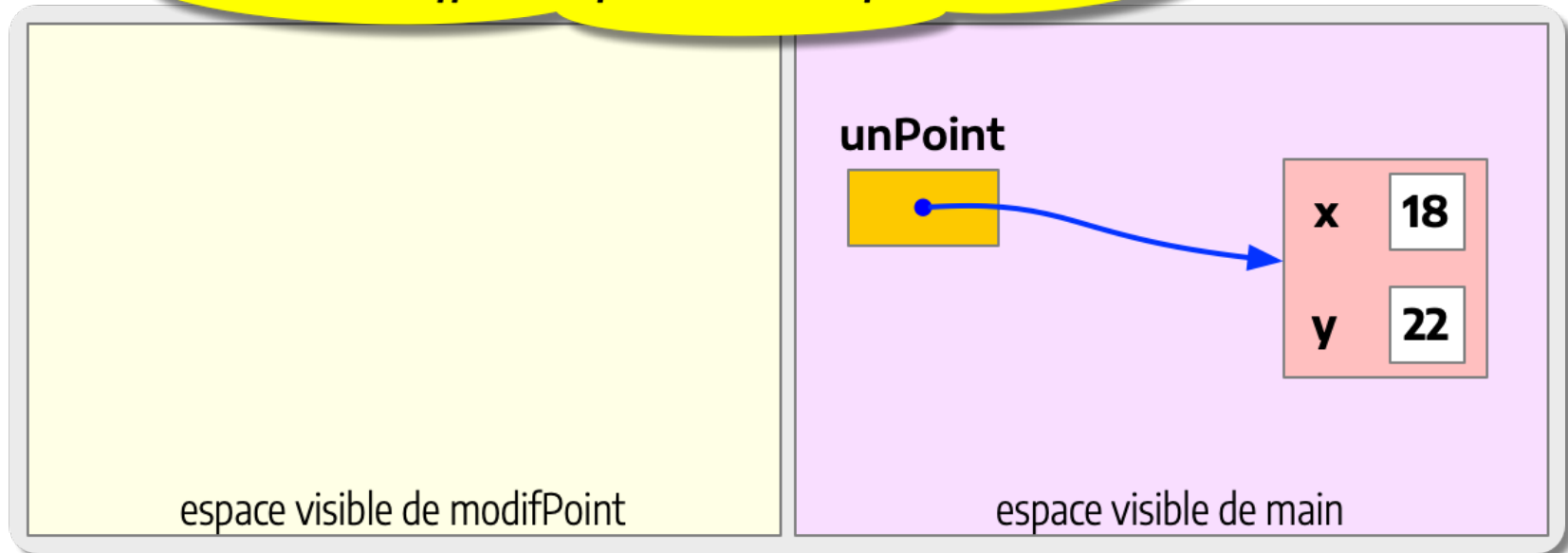
```
public class TraceParametresMuable {  
  
    private static void modifPoint(Point pFormel) {  
        // ajouter 10 en abscisse et en ordonnée  
        pFormel.deplace(10, 10);  
    }  
  
    public static void main(String[] args) {  
        Point uPoint = new Point(8, 12);  
        System.out.println("uPoint avant modifPoint=" + uPoint);  
        modifPoint(unPoint);  
        System.out.println("uPoint après modifPoint=" + uPoint);  
    }  
}
```

exécution de la procédure modifPoint()
– mise à jour du Point pointé par pFormel qui à pour effet de mettre à jour unPoint de main()
– unPoint et pFormel pointent sur le même objet modifié !



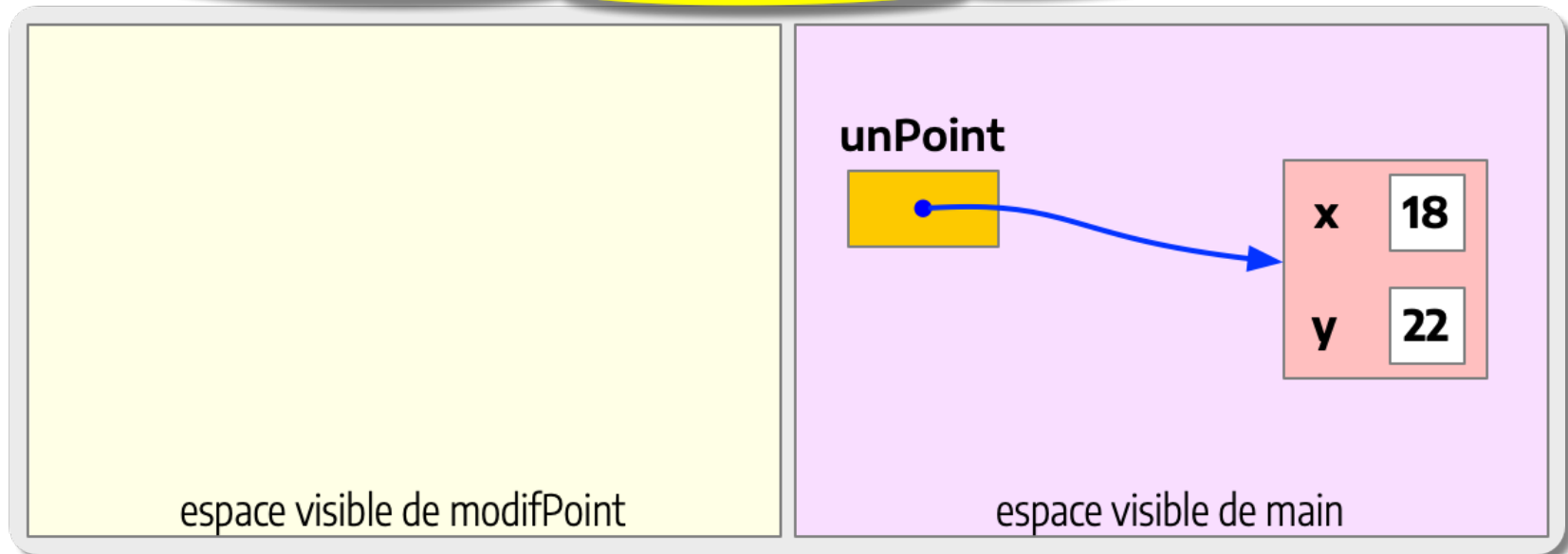
```
public class TraceParametresMuable {  
  
    private static void modifPoint(Point pFormel) {  
        // ajouter 10 en abscisse et en ordonnée  
        pFormel.x += 10;  
    }  
  
    public static void main(String[] args) {  
        Point unPoint = new Point(8, 12);  
        System.out.println("unPoint avant modifPoint = " + unPoint);  
        modifPoint(unPoint);  
        System.out.println("unPoint après modifPoint = " + unPoint);  
    }  
}
```

retour dans la procédure main()
– toute trace d'exécution de modifPoint()
est effacée : pFormel « disparaît »



```
public class TraceParametresMuable {  
  
    private static void modifPoint(Point pFormel) {  
        // ajouter 10 en abscisse et en ordonnée  
        pFormel.x += 10;  
        pFormel.y += 10;  
    }  
  
    public static void main(String[] args) {  
        Point uPoint = new Point(8, 12);  
        System.out.println("uPoint avant modifPoint = " + uPoint);  
        modifPoint(uPoint);  
        System.out.println("unPoint après modifPoint = " + unPoint);  
    }  
}
```

dernière instruction de la procédure main()
– *c'est fini !*



PARAMÈTRE EFFECTIF

ArrayList<E>

Exemple avec un ArrayList<E>

```
1 public class TraceParametresMuable {
2
3     private static void modifVecteur(ArrayList<Integer> vFormel) {
4         vFormel.set(0, 500);
5         vFormel.add(100);
6         vFormel.add(200);
7     }
8
9     public static void main(String[] args) {
10         ArrayList<Integer> v = new ArrayList<>(Arrays.asList(0, 1, 2,
11                                                                3, 4, 5));
12         System.out.println("v avant modifVecteur : " + v);
13         modifVecteur(v);
14         System.out.println("v après modifVecteur : " + v);
15     }
16 }
```

12 v avant modifVecteur : [0, 1, 2, 3, 4, 5]

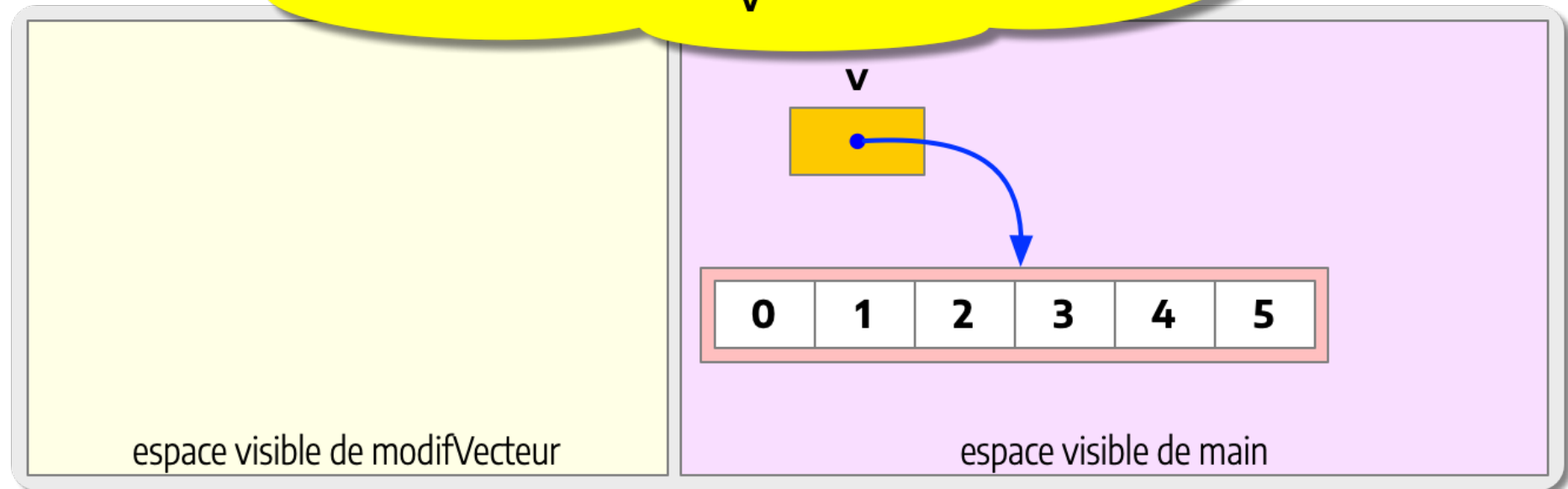
14 v après modifVecteur : [500, 1, 2, 3, 4, 5, 100, 200]

// la variable v de main a été modifiée par la procédure modifVecteur


```
public class TraceParametresMuable {  
  
    private static void modifVecteur(ArrayList<Integer> vFormel) {  
        vFormel.set(0, 500);  
vFormel.set(100);  
vFormel.set(200);  
    }  
  
    public static void main(String[] args) {  
        ArrayList<Integer> v = new ArrayList<>(Arrays.asList(0, 1, 2, 3, 4, 5));  
        System.out.println("v avant modifVecteur : " + v);  
        modifVecteur(v);  
System.out.println("v après modifVecteur : " + v);  
    }  
}
```

déclaration et initialisation d'une variable
locale à la procédure main()

v



```
public class TraceParametresMuable {
```

```
    private static void modifVecteur(ArrayList<Integer> vFormel) {
```

```
        vFormel.set(0, 500);  
        vFormel.set(100);  
        vFormel.set(200);
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        ArrayList<Integer> v = new ArrayList<>();  
        System.out.println("Avant modifVecteur : " + v);
```

```
        modifVecteur(v);
```

```
        System.out.println("Après modifVecteur : " + v);
```

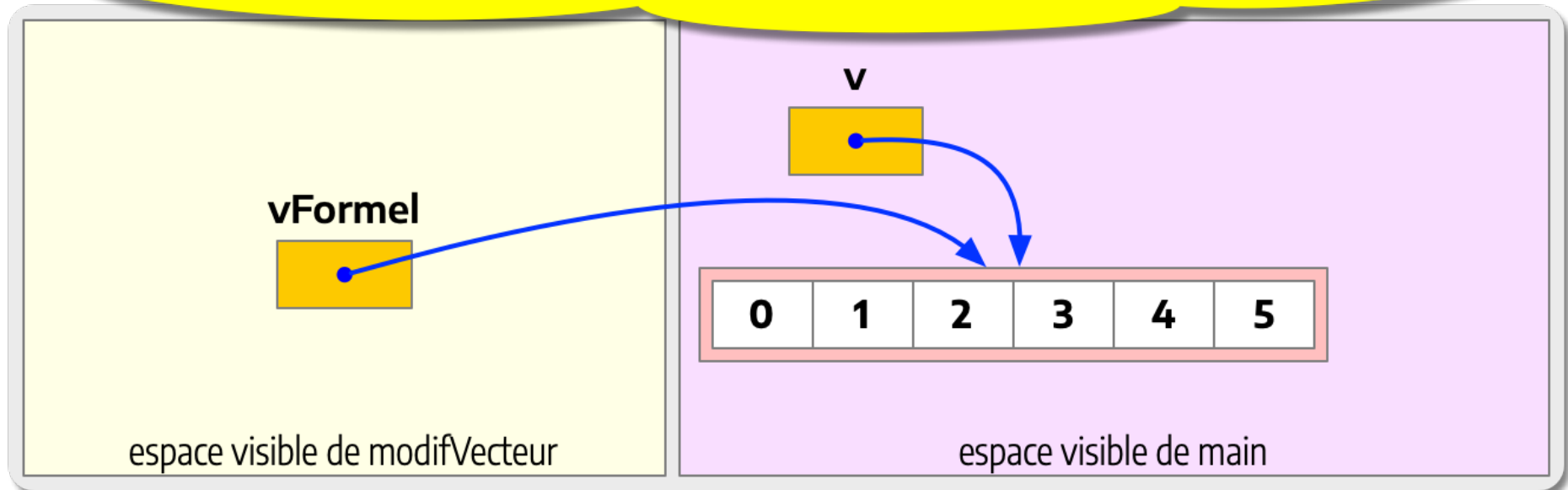
```
    }
```

```
}
```

appel de la procédure modifVecteur()

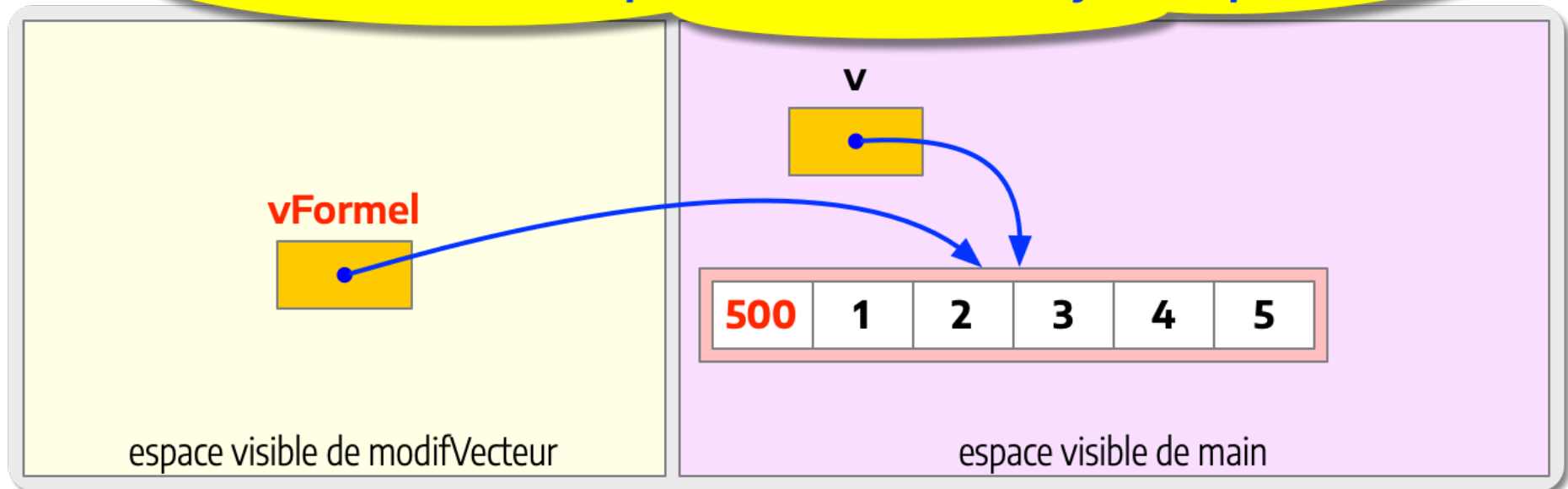
– initialisation du paramètre formel vFormel (variable locale) avec la valeur de v (cette valeur est une référence {pointeur})

– v et vFormel pointent sur le même objet !



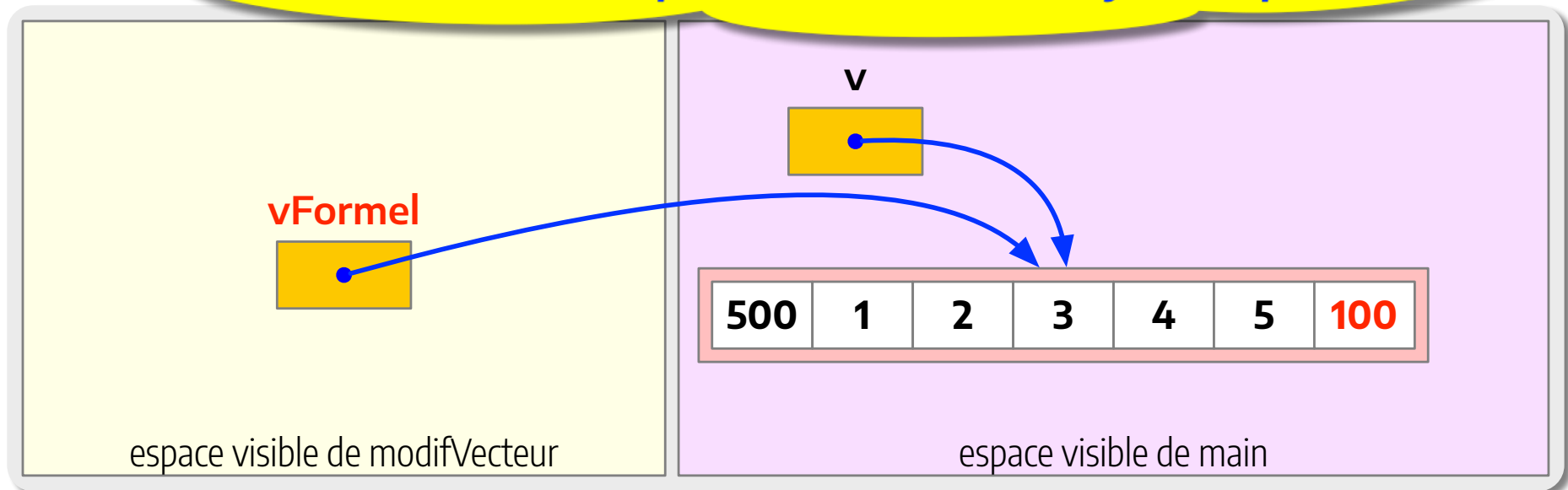
```
public class TraceParametresMuable {  
  
    private static void modifVecteur(ArrayList<Integer> vFormel) {  
        vFormel.set(0, 500);  
        vFormel.add(100);  
        vFormel.add(200);  
    }  
  
    public static void main(String[] args) {  
        ArrayList<Integer> v = new ArrayList<>();  
        System.out.println("V avant modifVecteur : " + v);  
        modifVecteur(v);  
        System.out.println("V après modifVecteur : " + v);  
    }  
}
```

exécution de la procédure modifVecteur()
– étape 1 : mise à jour de la valeur à l'indice 0 de vFormel
qui à pour effet de mettre à jour v de main()
– v et vFormel pointent sur le même objet modifié !



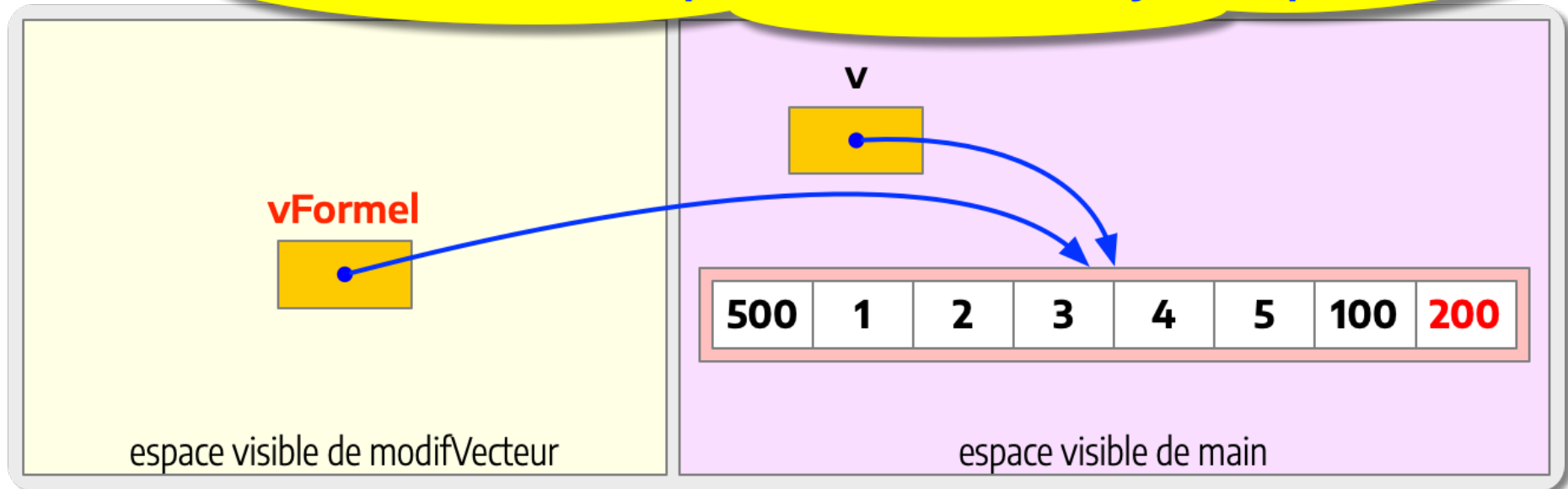
```
public class TraceParametresMuable {  
  
    private static void modifVecteur(ArrayList<Integer> vFormel) {  
        vFormel.set(0, 500);  
        vFormel.add(100);  
        vFormel.add(200);  
    }  
  
    public static void main(String[] args) {  
        ArrayList<Integer> v = new ArrayList<>();  
        System.out.println("Avant modifVecteur : " + v);  
        modifVecteur(v);  
        System.out.println("Après modifVecteur : " + v);  
    }  
}
```

exécution de la procédure modifVecteur()
– étape 2 : insertion de 100 à la fin de vFormel
qui à pour effet de mettre à jour unPoint de main()
– v et vFormel pointent sur le même objet modifié !



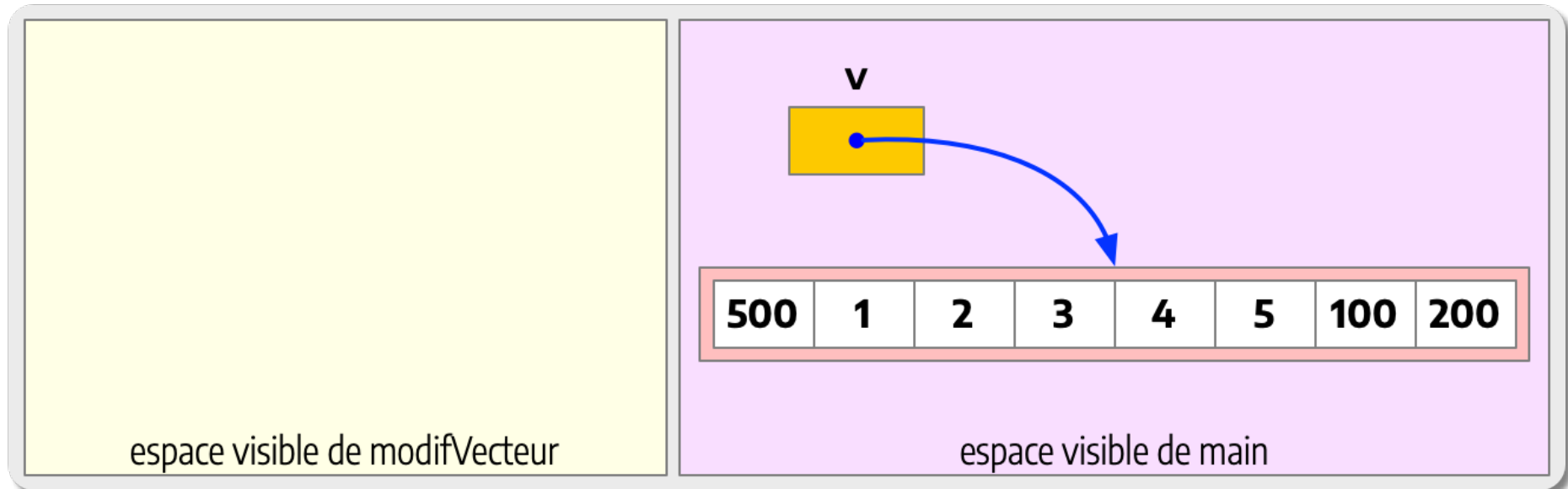
```
public class TraceParametresMuable {  
  
    private static void modifVecteur(ArrayList<Integer> vFormel) {  
        vFormel.add(500);  
vFormel.add(100);  
        vFormel.add(200);  
    }  
  
    public static void main(String[] args) {  
        ArrayList<Integer> v = new ArrayList<>();  
System.out.println("avant modifVecteur : " + v);  
        modifVecteur(v);  
        System.out.println("après modifVecteur : " + v);  
    }  
}
```

exécution de la procédure modifVecteur()
– étape 3 : insertion de 200 à la fin de vFormel
qui à pour effet de mettre à jour unPoint de main()
– v et vFormel pointent sur le même objet modifié !



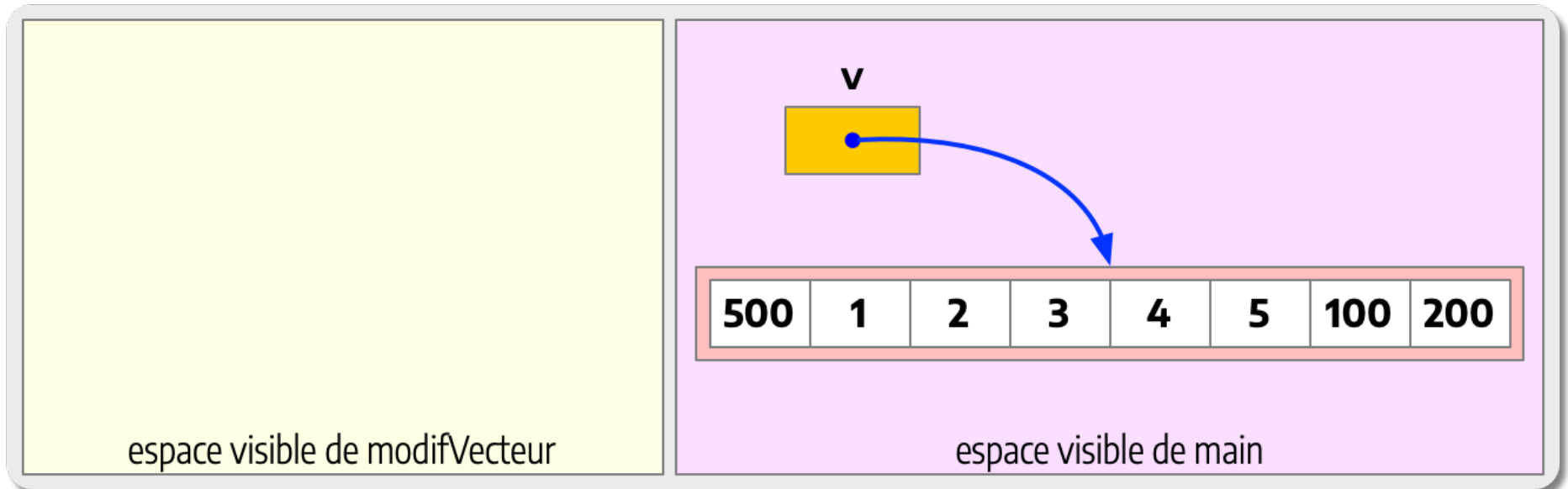
```
public class TraceParametresMuable {  
  
    private static void modifVecteur(ArrayList<Integer> vFormel) {  
        vFormel.set(0, 500);  
        vFormel.set(100);  
        vFormel.set(200);  
    }  
  
    public static void main(String[] args) {  
        ArrayList<Integer> v = new ArrayList<>();  
        System.out.println("Avant modifVecteur : " + v);  
        modifVecteur(v);  
        System.out.println("Après modifVecteur : " + v);  
    }  
}
```

retour dans la procédure main()
– toute trace d'exécution de modifVecteur()
est effacée : vFormel « disparaît »




```
public class TraceParametresMuable {  
  
    private static void modifVecteur(ArrayList<Integer> vFormel) {  
        vFormel.set(0, 500);  
        vFormel.set(100);  
        vFormel.set(200);  
    }  
  
    public static void main(String[] args) {  
        ArrayList<Integer> v = new ArrayList<>();  
        System.out.println("v avant modifVecteur : " + v);  
        modifVecteur(v);  
        System.out.println("v après modifVecteur : " + v);  
    }  
}
```


dernière instruction de la procédure main()
– c'est fini !



Exemple de mise en œuvre

-  Nous avons profité du comportement des paramètres effectifs de type `ArrayList<E>` dans les algo. de tri

```
private static void triSelection(ArrayList<Integer> v) {...}  
private static void triBulles(ArrayList<Integer> v) {...}  
private static void triBullesAmeliorer(ArrayList<Integer> v) {...}  
private static void triInsertion(ArrayList<Integer> v) {...}  
private static void triInsertion(ArrayList<Integer> v) {...}  
// vu en TP
```

-  C'est bien le vecteur passé en paramètre effectif (lors de l'appel de la procédure de tri) qui est trié