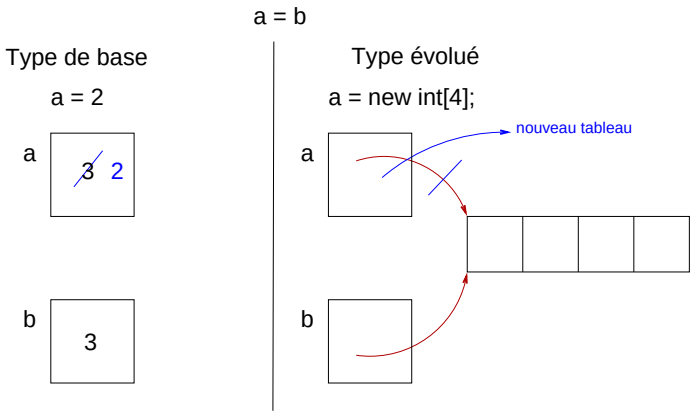
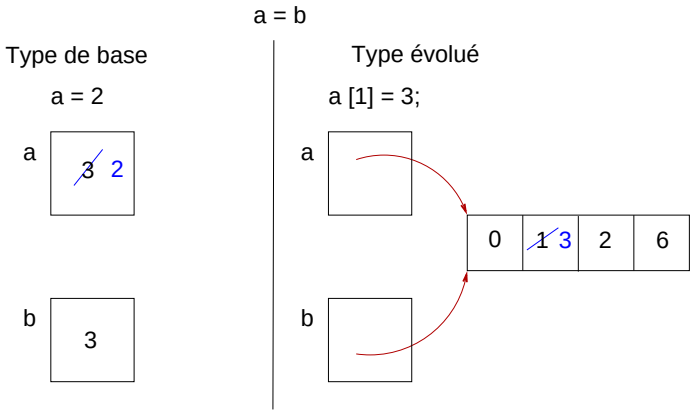
Types de base / Types évolué (rappel)



☞ Type de base : modifier `a` ne modifie pas `b`

☞ Type évolué : modifier la référence `a` ne modifie pas la référence `b`

Types de base / Types évolué (rappel)



☞ Type évolué : modifier (l'objet référencé par) `a` modifie (l'objet référencé par) `b`

Tableaux : sémantique de l'opérateur =

```
// Les tableaux a et b pointent vers deux emplacements
// différents en memoire
int[] a = new int[10]; // tableau de 10 entiers
int[] b = new int[10]; // tableau de 10 entiers

for (int i = 0; i < a.length; ++i) {
   a[i] = i; // remplissage du tableau pointé par a
}
b = a; // opérateur =  (affectation)
System.out.println("a[2] vaut " + a[2] + " et b[2] vaut " + b[2]);
a[2] = 42;
System.out.println("a[2] vaut " + a[2] + " et b[2] vaut " + b[2]);
```

ce qui affiche :

a[2] vaut 2 et b[2] vaut 2
a[2] vaut 42 et b[2] vaut 42

Les deux tableaux `a` et `b`, après l'affectation `b=a;` pointent vers le même emplacement mémoire ⇒ En changeant `a[2]` on change alors implicitement `b[2]` et inversément !

Le tableau créé pour `b` : `int[] b = new int[10];` n'est donc jamais rempli ni utilisé !

Tableaux : utilisation (rare) de l'opérateur =

A moins de vouloir deux noms de variables pour le même tableau, il n'y a pas d'intérêt à vouloir assigner un tableau un à autre

☞ l'utilisation de l'opérateur `=` pour les tableaux est donc rare !

Pour avoir deux tableaux distincts `a` et `b` qui ont les mêmes valeurs (c'est-à-dire faire une copie de `a` dans `b`) il aurait fallu utiliser :

```
for(int i = 0; i < a.length; ++i) {
 b[i] = a[i];
}
```

Attention : il faut que `b.length` $\geq$ `a.length` !

Tableaux : sémantique de l'opérateur == (1)

L'opérateur `a == b` teste si les variables `a` et `b` référencent le même emplacement mémoire.

⇒ ce qui est donc le cas lors de l'affectation `b = a;`

L'opérateur `a == b` **ne teste pas** l'égalité des valeurs contenues dans les tableaux pointés par `a` et `b` !

Tableaux : sémantique de l'opérateur == (2)

Pour vérifier l'égalité de contenu des tableaux, il faut écrire explicitement les tests :

```
if (a == null || b == null || a.length != b.length) {
    System.out.println("contenus différents ou nuls");
}
else {
    int i = 0;
    while(i < a.length && (a[i] == b[i])) {
        ++i;
    }
    if (i >= a.length) {
        System.out.println("contenus identiques");
 }
    else {
        System.out.println("contenus différents")
    }
}
```

Quelques exemples de manipulation de tableaux

Soit un tableau déclaré par :

```
double[] tab = new double[10];
```

Affichage du tableau :

- si l'on n'a pas besoin d'expliciter les indices :

```
System.out.print("Le tableau contient : ");
for(double element : tab) {
   System.out.print(element + " ");
}
System.out.println();
```

- si l'on veut expliciter les indices :

```
for(int i = 0; i < tab.length; ++i) {
   System.out.println("L'élément " + i + " vaut " + tab[i]);
}
```

Quelques exemples de manipulation de tableaux

Saisie au clavier des éléments du tableau :

- Il est toujours nécessaire d'expliciter les indices :

```
for(int i = 0; i < tab.length; ++i) {
   System.out.prinln("Entrez l'élément " + i +  ":");
   tab[i] = scanner.nextDouble();
}
```