Les structures de données
Problème de l’affectation

Sémantiques de l’affectation

- Une différence subtile (et potentiellement dangereuse) reste détectable entre nos 2 classes de piles (avec tableaux ou dynamiques) ; **l’affectation**
- En effet, la duplication d’objets peut être problématique si elle est mal maîtrisée. Et la sémantique peut dérouter.
- Regardons d’abord pour des tableaux.

Dupliquer un objet (bad example)

```
public static void main(String arg[]) {
    int[] tableau = {1, 2};
    int[] tmp;
    tmp = tableau;
    tmp[0] = 7;
    System.out.println("tableau contient : "+ tableau[0] + ", " + tableau[1]);
}   ===> Affiche, « tableau contient : 7 ,2 » !!!!!
```

2 solutions

- int[] array = {23, 43, 55};

  int[] copiedArray = new int[3];

  System.arraycopy(array, 0, copiedArray, 0, 3);

  ou

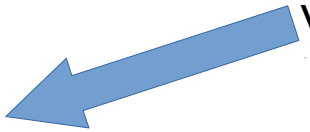
  int[] copiedArray = Arrays.copyOf(array, newLength);
- int[] copiedArray = array.clone();
- (Niveau L2/Ing1) Les 2 solutions font tout de même une «shallow copy», c'est-dire que le contenu est partagé entre les tableaux, c'est-à-dire que les objets contenu dans le tableau initial ne sont pas eux même clonés (danger!) ; int!=objet donc pour nous ça va

Et pour l’égalité ?

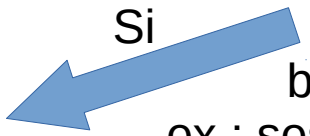
- Le problème va être le même. Est-ce que l’on considère une égalité physique (mémoire ordinateur) ou une égalité logique ?
- Imaginons que des tableaux d’entiers représentent des ensembles.
  - Le tableau {1,2,3} représente l’ensemble 1,2,3
  - Mais le tableau {2,1,3} représente aussi l’ensemble 1,2,3
  - Car 1,2,3 = 2,1,3 (égalité logique)
  - Mais le tableau {1,2,3}!={2,1,3} et même
  - int[] tab1={1,2,3} ; int[] tab2={1,2,3} ; tab1!=tab2 (inégalité physique)

Clone et equals

- La solution proposée par Java (et qui existe aussi dans d'autres langages)



Vrai rôle étudié en L2



Si besoin de bloquer (sécurité, ex : session)

```
public class Toto implements Cloneable {
        …
        public Toto clone() throws CloneNotSupportedException {…}
        …
        public boolean equals(Toto autre)   //on suppose que c'est commutatif s1.equals(s2)==s2.equals(s1)
        }
```

- Exemples :

  int[] t= {1,2,3} » ; ⇒ t!={1,2,3}   mais t.equals({1,2,3}) ;

  int[] tprim=t.clone();

Et pour nos piles alors ?

Il faut varier les codes suivant la structure sous-jacente (tableau ou liste dynamique) :

public boolean equals(Stack s) {return data.equals(s.data)} // Tableaux

ou

idem avec des listes mais il faut aussi avoir l'égalité des listes...

```
public boolean equals(ListeDEntier s) {
    CelluleEntier cAutre = s.cellule ;
    CelluleEntier cMoi = s.cellule ;
    while (cAutre!=null) || (cMoi!=null) {
        if !(cAutre.valeur()).equals(cMoi.valeur()) return false ;
        cAutre=cAutre.celluleSuivante() ;
        cMoi=cMoi.celluleSuivante() ;
    }
    if (cAutre==null) && (cMoi==null) return true ;
```