Tableaux

- taille fixe
- type des éléments spécifié -> contrôle au moment de la compilation
- possibilité de stocker des types de base (int, char, …) ou des objets

déclaration

```
String[] tabStrings = new String[10];
boolean[] tabBooleans = {true, true, false, true, false};
```

affectation d'une valeur à une case

```
tab[3] = "salut"
```

taille du tableau

```
tabBooleans.length
```

parcours d'un tableau (et impression)

```
for (int i = 0; i < tabBooleans.length; i++){
   System.out.println(tabBooleans[i]);
}
```

la classe Arrays

Cette classe contient diverses méthodes pour manipuler des tableaux

fill : affecte la même valeur à toutes les cases du tableau

```
public class RandomTab{
   public static void main(String[] args){
      int[] tableau = new int[20];
      Arrays.fill(tableau, (int)(Math.random()*100));
      //remplit toutes les cases avec le MÊME nombre
      for(int i = 0; i < tableau.length; i++){
         System.out.println(tableau[i]);
      }
      for(int i = 0; i < tableau.length; i++){
         tableau[i] = (int)(Math.random()*100);
      }
      //remplit le tableaux avec des nombres différents
      for(int i = 0; i < tableau.length; i++){
         System.out.println(tableau[i]);
      }
   }
}
```

la classe Arrays

equals : égalité profonde

```
Arrays.equals(tableau1, tableau2);
```

equals() retourne vrai si :
- les deux tableaux sont de la même taille
- les deux tableaux contiennent le même type d'éléments
- les paires d'éléments correspondent tableau1[i]=tableau2[i]

sort : pour trier un tableau
(voir page suivante)

```
import java.util.Random;
import java.util.Arrays;

public class RandomTab{

    public static void main(String[] args){
        int[] tableau = new int[20];
        for(int i = 0; i < tableau.length; i++){
            tableau[i] = (int)(Math.random()*100);
        }
        Arrays.sort(tableau);
        for(int i = 0; i < tableau.length; i++){
            System.out.println(tableau[i]);
        }
    }
}
```

binarySearch : recherche d'un élément

```
Arrays.binarySearch(tableau, 17);
```

retourne l'index de l'élément recherché ou une valeur négative sinon
le tableau doit avoir été trié au préalable (avec Arrays.sort())

la classe Arrays

Tri d'un tableau d'objets / recherche dans un tableau d'objets

Par exemple, si on a la classe

```
public class Employe{
    int numeroEmp;
    String nom;
    String prenom;
    int age;
    …
}
```

sur quelle variable faut-il effectuer le tri ?

⇒ Pour pouvoir utiliser les méthodes sort et binarySearch dans un tableau d'objets, il faut s'assurer que le type d'objets stocké dans le tableau implémente l'interface Comparable ou créer une implémentation de l'interface Comparator.

la classe Arrays

Implémentation de l'interface Comparable

```
public class Employe implements Comparable{
    int numeroEmp;
    String nom;
    String prenom;
    int age;
    …

    public int compareTo(Object o){
        int num = ((Employe)o).numeroEmp;
        if (num < numeroEmp) { return(-1); }
        else if (num == numeroEmp) { return(0); }
        else return(1);
    }
}
```

la classe Arrays

Créer une implémentation de l'interface Comparator

```
class EmpComparator implements Comparator{
    public int compare(Object o1, Object o2){
        int i1 = ((Employe)o1).numeroEmp;
        int i2 = ((Employe)o2).numeroEmp;
        return( i1 < i2 ? –1 : (i1 == i2 ? 0 : 1));
    }
}

public class ComparatorTest {
    Employe[] tabEmpl = new Employe[10];
    …  // remplissage du tableau
    Arrays.sort(tabEmpl, new EmpComparator);
    …
}
```

Autre utilitaire pour tableaux

System.arraycopy() : copier un tableau (ou une partie)

```
public class CopieTableau {
    public static void main(String[] args) {
        char[] source = { 'a', 'r', 't', 'i', 'f', 'i', 'c', 'i', 'e', 'l', 'l', 'e' };
        char[] cible = new char[4];
        System.arraycopy(source, 6, cible, 0, 4);
        System.out.println(new String(cible));
    }
}
```

⇒ ciel

Collections

- taille extensible
- stockage d'objets uniquement
- perte d'information sur la classe des éléments

Interfaces

List (liste ou séquence)
Collection ordonnée d'objets. Plusieurs objets ayant la même valeur peuvent figurer dans une liste. Chaque objet a une position à l'intérieur de la liste.

Set (ensemble)
Collection d'objets où chaque objet à une valeur différente. Il n'existe pas d'indice indiquant la position d'un objet à l'intérieur d'un ensemble.

Map (fonction, dictionnaire, tableau associatif)
Collection d'objets dans laquelle la position de chaque objet est indiquée par un autre objet (à la place d'un indice numérique comme dans un tableau ou une liste).

Itérateurs

- Curseur qui se déplace le long d'une collection en visitant successivement tous les éléments.
- Les itérateurs sont nécessaires pour le parcours de collections dont les éléments ne sont pas indicés ou lorsque les valeurs des indices (clés) sont inconnues.

méthodes

- boolean **hasNext**() -> indique s'il y a encore des éléments à parcourir
- Object **next**() -> retourne le prochain élément
- void **remove**() -> supprime le dernier élément retourné par l'itérateur

```
Iterator i = maCollection.iterator();
while (i.hasNext()) {
    objetCourant = i.next();
    <traitement de objetCourant>
    <éventuellement> i.remove();
    // jamais maCollection.remove(…)
}
```

Imprimer une collection

```
System.out.println(maCollection);
```

```
public class RandomColl{
    public static void main(String[] args){
        List l = new ArrayList();
        l.add("moi");
        l.add("toi");
        l.add("elle");
        l.add("lui");
        System.out.println(l);
    }
}
```

⇒ [moi, toi, elle, lui]