Amphi 4

Contrôle d'accès, Paquetages, Java API

Plan

- Rappel
- Contrôle d'accès
- Paquetages
- Java API

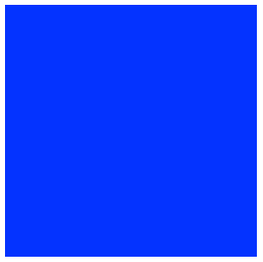
```
public class Compteur{

  int monCompteur = 0;
  static int notreCompteur = 0;

  void incrementer(){
     monCompteur ++;
     notreCompteur ++;
  }

  public static void main (String [] args){
     Compteur c1 = new Compteur();
     Compteur c2 = new Compteur();
     c1.incrementer();
     c1.incrementer();
     c2.incrementer();
     System.out.println(c1.monCompteur + " " + c1.notreCompteur);
     System.out.println(c2.monCompteur + " " + c2.notreCompteur);
  }
}
```

attributs

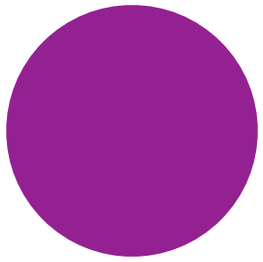
méthode d'instance

```
Compteur c1 = new Compteur();
```

Compteur
de classe



Compteur
de l'objet c1

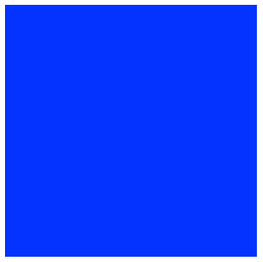


```
notreCompteur = 0;
```

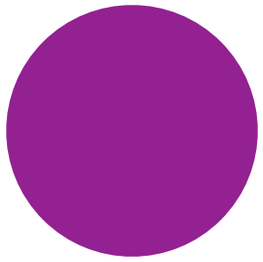
```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();
```

Compteur
de classe



Compteur
de l'objet c1

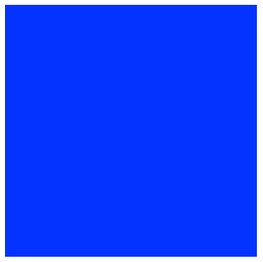


```
notreCompteur = 0;
```

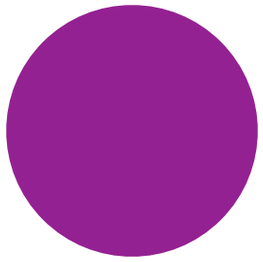
```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();

Compteur c2 = new Compteur();
```

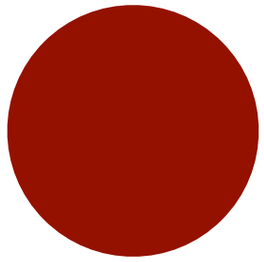
Compteur
de classe



Compteur
de l'objet c1



Compteur
de l'objet c2



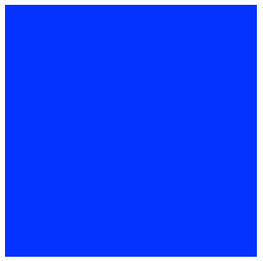
```
notreCompteur = 0;
```

```
monCompteur = 0;
```

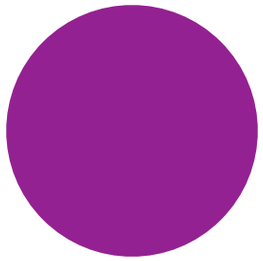
```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();

Compteur c2 = new Compteur();
```

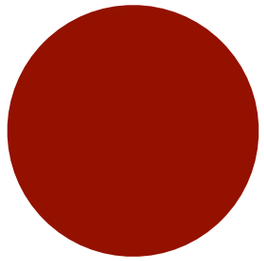
Compteur de classe



Compteur de l'objet c1



Compteur de l'objet c2



```
notreCompteur = 0;
```

```
monCompteur = 0;
```

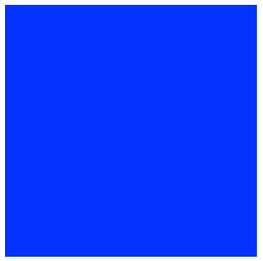
```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();
```

```
Compteur c2 = new Compteur();
```

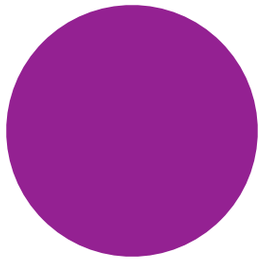
```
c1.incrementer();
```

Compteur
de classe



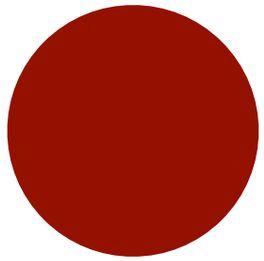
```
notreCompteur = 0;
                1
```

Compteur
de l'objet c1



```
monCompteur = 0;
              1
```

Compteur
de l'objet c2



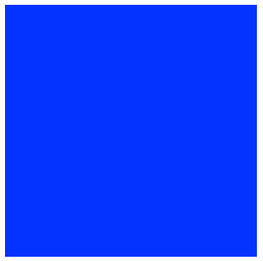
```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();
```

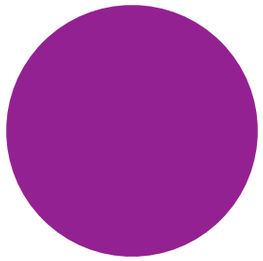
```
c1.incrementer();
```

```
Compteur c2 = new Compteur();
```

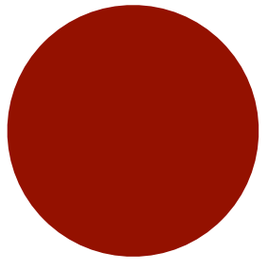
Compteur
de classe



Compteur
de l'objet c1



Compteur
de l'objet c2



```
notreCompteur = 0;
                1
```

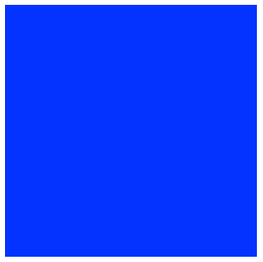
```
monCompteur = 0;
              1
```

```
monCompteur = 0;
```

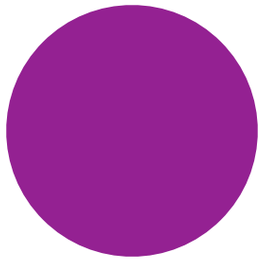
```
Compteur c1 = new Compteur();

Compteur c2 = new Compteur();
```

```
c1.incrementer();

c1.incrementer();
```

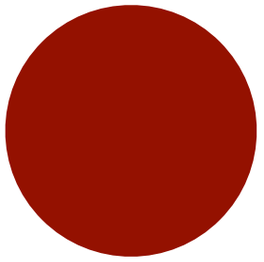
Compteur
de classe



Compteur
de l'objet c1



Compteur
de l'objet c2



```
notreCompteur = 0;
                1
                2
```

```
monCompteur = 0;
              1
              2
```

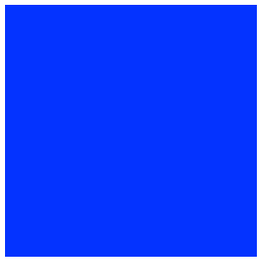
```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();
```

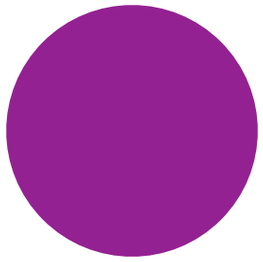
```
c1.incrementer();
c1.incrementer();
```

```
Compteur c2 = new Compteur();
```

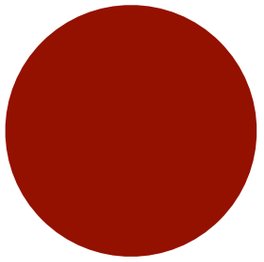
Compteur
de classe



Compteur
de l'objet c1



Compteur
de l'objet c2



```
notreCompteur = 0;
                1
                2
```

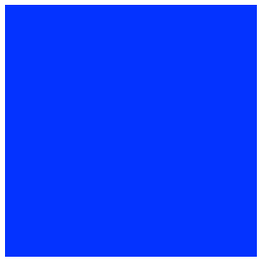
```
monCompteur = 0;
              1
              2
```

```
monCompteur = 0;
```

```
Compteur c1 = new Compteur();

Compteur c2 = new Compteur();
```

```
c1.incrementer();

c1.incrementer();

c2.incrementer();
```

Compteur
de classe

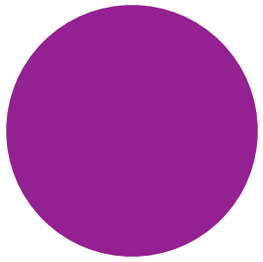


notreCompteur = ~~0~~;
~~1~~
~~2~~
3

Compteur c1 = new Compteur();

Compteur c2 = new Compteur();

Compteur
de l'objet c1

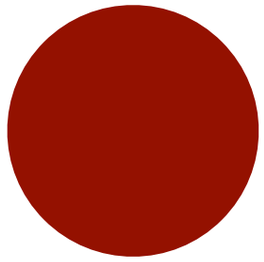


monCompteur = ~~0~~;
~~1~~
2

c1.incrementer();

c1.incrementer();

Compteur
de l'objet c2



monCompteur = ~~0~~;
1

c2.incrementer();

Contrôle d'accès

```
public class Carte{
  String numeroCarte;
  double depense;

  void facturer (double somme){
     depense = depense + somme;
  }

  String getNumeroCarte
              (String motDePasse){
     if (motDePasse.equals
              ("CodeSecrete")){
        return this.numeroCarte;
     }
     else {
        return "Attention!";
     }
  }
}
```

Contrôle d'accès

```
public class Carte{
  String numeroCarte;
  double depense;

  void facturer (double somme){
     depense = depense + somme;
  }

  String getNumeroCarte
              (String motDePasse){
     if (motDePasse.equals
              ("CodeSecrete")){
        return this.numeroCarte;
     }
     else {
        return "Attention!";
     }
  }
}
```

Problème

```
public class Malhonnete{
     static void malMethode
                    (Carte c){
        c.depense = 0;
        System.out.println
                (c.numeroCarte);
     }
}
```

Contrôle d'accès

```
public class Carte{
  String numeroCarte;
  double depense;

  void facturer (double somme){
     depense = depense + somme;
  }

  String getNumeroCarte
              (String motDePasse){
     if (motDePasse.equals
              ("CodeSecrete")){
        return this.numeroCarte;
     }
     else {
        return "Attention!";
     }
  }
}
```

Problème

```
public class Malhonnete{
     static void malMethode
                    (Carte c){
        c.depense = 0;
        System.out.println
                (c.numeroCarte);
     }
}
```

Comment se protéger?

Public & Private

- Public: les autres peuvent utiliser
- Private: seulement les méthodes dans la classe peuvent accéder et utiliser.

Public/Private s'appliquent aux attributs ainsi qu'aux méthodes.

Contrôle d'accès

```
public class Carte{
  String numeroCarte;
  double depense;

  void facturer (double somme){
     depense = depense + somme;
  }

  String getNumeroCarte
              (String motDePasse){
     if (motDePasse.equals
              ("CodeSecrete")){
        return this.numeroCarte;
     }
     else {
        return "Attention!";
     }
  }
}
```

```
public class Carte{
  private String numeroCarte;
  private double depense;

  public void facturer
              (double somme){
     depense = depense + somme;
  }

  public String getNumeroCarte
              (String motDePasse){
     if (motDePasse.equals
("CodeSecrete")){
        return this.numeroCarte;
     }
     else {
        return "Attention!";
     }
  }
}
```

Pourquoi contrôle d'accès

- Protéger l'information privée
- Clarifier comment une classe utilise d'autres classes
- Séparer l'implémentation et l'interface

Paquetages

- Chaque classe appartient à un paquetage.
- Les classes dans un même paquetage servent aux mêmes buts

- Les paquetages sont les répertoires.
- Les classes dans un autre paquetage doivent être importées (s'il y a besoin d'utilisation)

Paquetages

- Définition d'un paquetage

```
package chemin.paquetage.foo

class Foo {
  ...
}
```

- Utilisation d'un paquetage

```
import chemin.paquetage.foo
import chemin.paquetage.*

class Goo {
  ...
}
```

Pourquoi paquetages?

- Combiner les fonctionnalités similaires

```
org.evry.bibliotheque.livres

org.evry.bibliotheque.disques
```

- Séparer les noms similaires

```
org.evry.shopping.List

org.evry.packing.List
```

Propriétés

- Les classes dans un même paquetages se trouve automatiquement (pas besoin d'importer)

- Toutes les classes, par défaut, importent (implicitement) les classes dans java.lang, exemple: java.lang.String, java.lang.System

Java API

- Java API: Java Application Programming Interface
- Java incluit beaucoup de paquetages/classes (par exemple, ArrayList, ...)
- Utiliser les classes dans Java API pour économiser/optimiser le travail
- http://docs.oracle.com/javase/6/docs/api (le lien se trouve sur le site du cours)

Classe génériques à spécialiser

La classe générique ArrayList<E>

ArrayList<E> doit être spécialisée à l'aide d'un nom de classe d'objets

- Example de spécialisation de classe générique

  `ArrayList <Domino>` représente la classe des listes de dominos

  `ArrayList <Livre>` représente la classe des listes de livres

  `ArrayList <int>` interdit

- Instanciation (utilisation d'un constructeur par défaut)

```
ArrayList <Domino> listDominos = new ArrayList<Domino>();

ArrayList <Livre>  listLivres  = new ArrayList<Livre>();
```

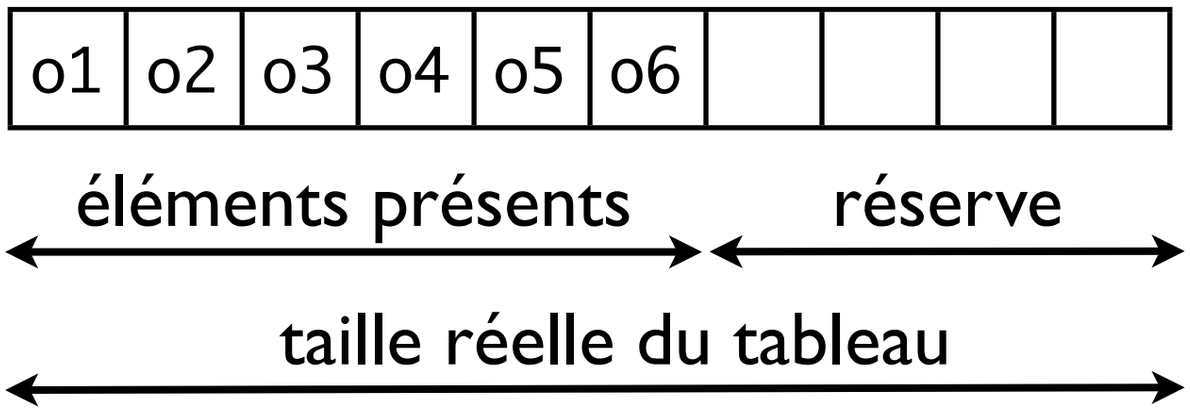
Classe génériques à spécialiser

Une fois la classe génériques spécialisée, on dispose d'une véritable classe qui:

- encapsule un ensemble d'objets
  - la classe ArrayList<E> encapsule un tableau d'objets et en gère les aspect dynamique
- dispose de nombreuses méthodes d'instance publiques
  - par rapport à l'ensemble `size(), isEmpty()`
  - par rapport à la lecture `get(), contains()`
  - par rapport à l'écriture `add(), remove()`
- considère un ordre naturel sur l'ensemble
- les méthodes d'instance sont optimisées

ArrayList<E> & tableaux d'objets

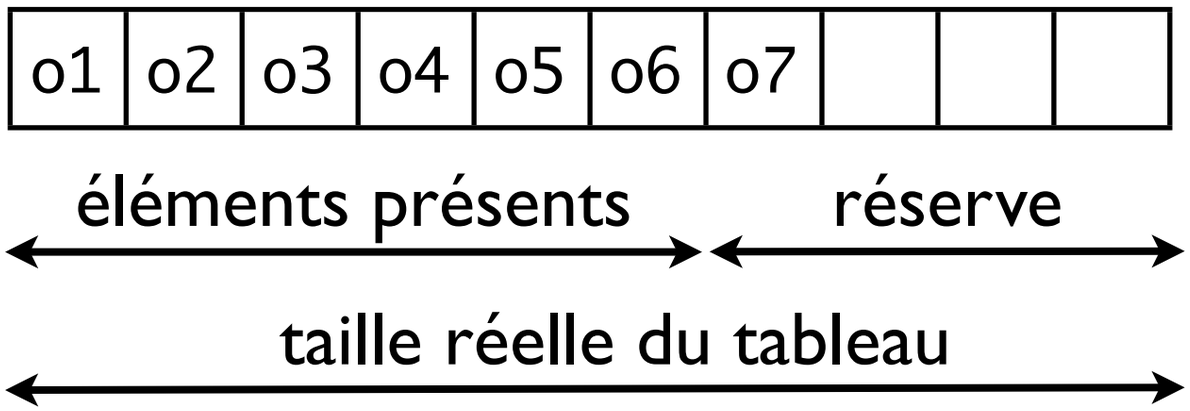
Comment ArrayList<E> gère-t-elle son tableau d'objet?



- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ArrayList<E> & tableaux d'objets

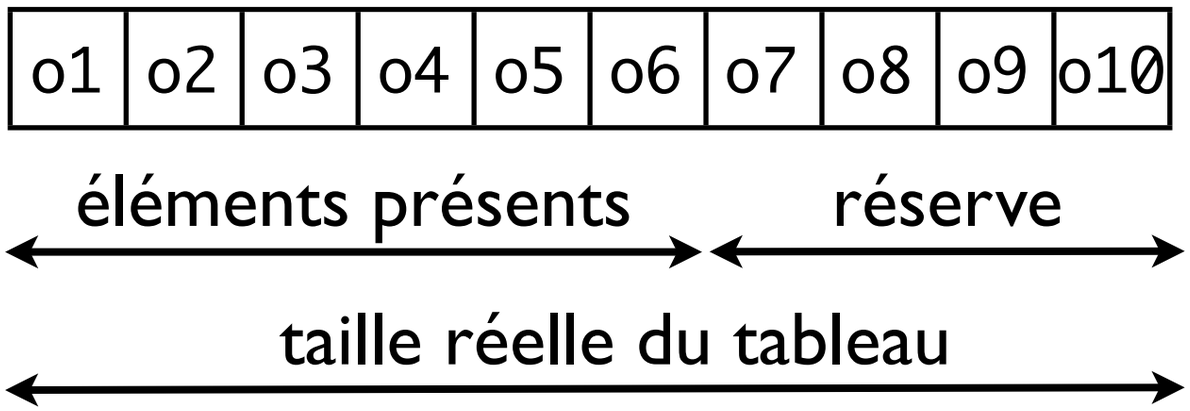
Comment ArrayList<E> gère-t-elle son tableau d'objet?



- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ArrayList<E> & tableaux d'objets

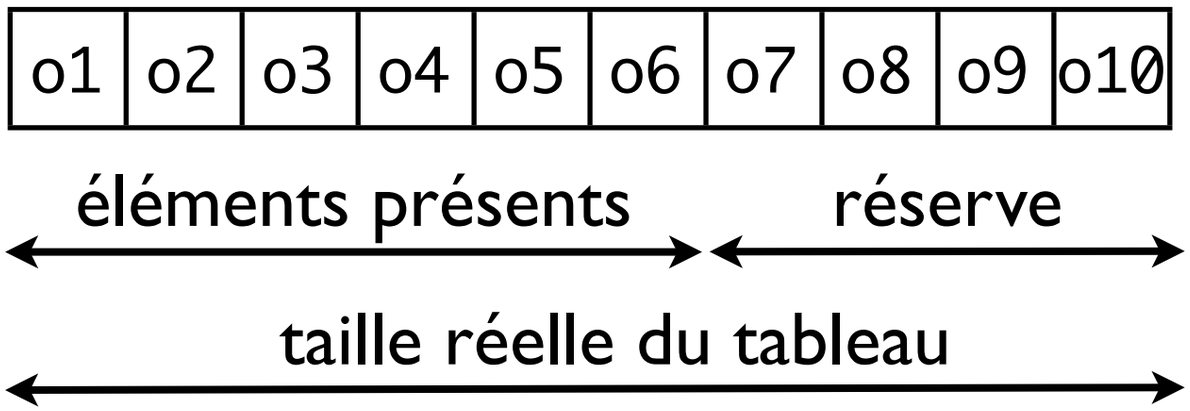
Comment ArrayList<E> gère-t-elle son tableau d'objet?

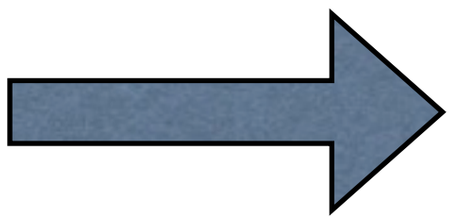


- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ArrayList<E> & tableaux d'objets

Comment ArrayList<E> gère-t-elle son tableau d'objet?



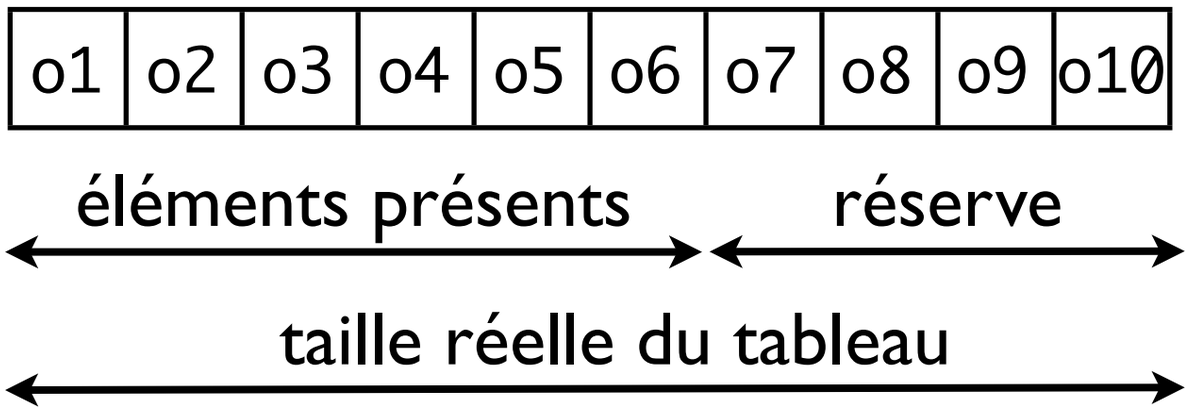


ajouter d'un
nouveau élément

- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

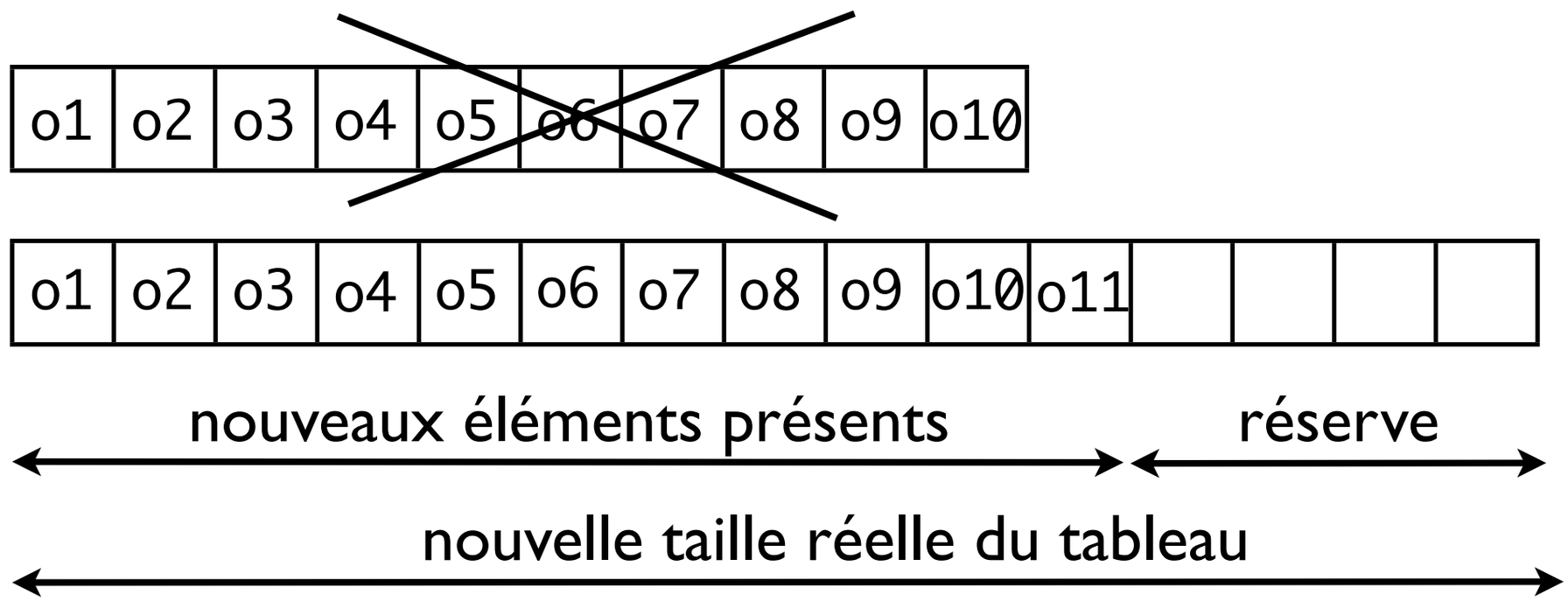
ArrayList<E> & tableaux d'objets

Comment ArrayList<E> gère-t-elle son tableau d'objet?



- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ajouter d'un nouveau élément



ArrayList<E> & méthodes utiles

- par rapport à l'ensemble:

"E" : Livre, Domino, etc

```
int size()              retourne la taille de l'ensemble
boolean isEmpty()       teste si l'ensemble est vide
Object clone()          retourne une copie de l'objet ArrayList<E>
```

- par rapport à la lecture dans l'ensemble:

```
E get(int index)        retourne l'élément d'indice index
boolean contains (E o)          teste si l'ensemble contient o
int indexOf (E o)               retourne l'indice de l'élément o ou -1 s'il n'existe pas
```

- par rapport à l'écriture dans l'ensemble

```
boolean add(E e)        ajouter un élément e à la fin du tableau (retourner true)
void add(int index, E e)    insérer l'élément e à l'indice index
E set(int index, E e)   remplace l'élément d'indice index par e et retourner l'ancien
E remove(int index)     supprimer l'élément d'indice index et le retourne
E remove(E e)           supprimer e et retourn true s'il existait, false sinon
```

ArrayList<E> & Démo

```
import java.util.ArrayList;

class ArrayListExemple{
  public static void main  (String [] args){
     ArrayList<String> strings = new ArrayList<String>();

     strings.add("Even");
     strings.add("Eugene");
     strings.add("Adam");

     System.out.println(strings.size());
     System.out.println(strings.get(0));
     System.out.println(strings.get(1));

     strings.set(0,"Eva");
     strings.remove(1);
     for (int i=0; i<strings.size(); i++) {
        System.out.println(strings.get(i));
     }
     for (String s: strings) {
        System.out.println(s);
     }
  }
}
```

Sets

- Ressemble à ArrayList, mais
  - seulement un copie pour chaque objet
  - pas d'indice de tableau
- Propriétés:
  - ajouter un objet dans un ensemble (méthode add)
  - supprimer un objet dans un ensemble (méthode remove)
  - vérifier si un objet donné appartient à un ensemble

TreeSet: ordonnés (petit à grand, alphabet)
HashSet: non-ordonnés (pseudo-random)

TreeSet<E> & Démo

```
import java.util.TreeSet;

class TreeSetExemple{
  public static void main   (String [] args){
     TreeSet<String> strings = new
TreeSet<String>();

     strings.add("Even");
     strings.add("Eugene");
     strings.add("Adam");

     System.out.println(strings.size());
     System.out.println(strings.first);
     System.out.println(strings.last);

     strings.remove("Eugene");

     for (String s: strings) {
        System.out.println(s);
     }
  }
}
```

Maps

- Stocker un couple (clé, valeur) d'objets

- Donné un clé, chercher la valeur.

  Exemple: (nom, mail); (étudiant, numéro d'étudiant)

TreeMap: ordonnés (petit à grand, alphabet)
HashMap: non-ordonnés (pseudo-random)

HashMap<E> & Démo

```
import java.util.*; // import some packages for
                    // HashMap

class HashMapExemple{
  public static void main  (String [] args){
     HashMap<String,String> strings = new
          HashMap<String,String>();

     strings.put("Even", "even@bio.evry");
     strings.add("Eugene", "eugene@math.evry");
     strings.add("Adam", "adam@info.evry");

     System.out.println
          (strings.size());
     strings.remove("Even");
     System.out.println
       (strings.get("Eugene"));
}
```

```
for (String s: strings.keySet()) {
   System.out.println(s);
}

for (String s: strings.values()) {
   System.out.println(s);
}

}
```