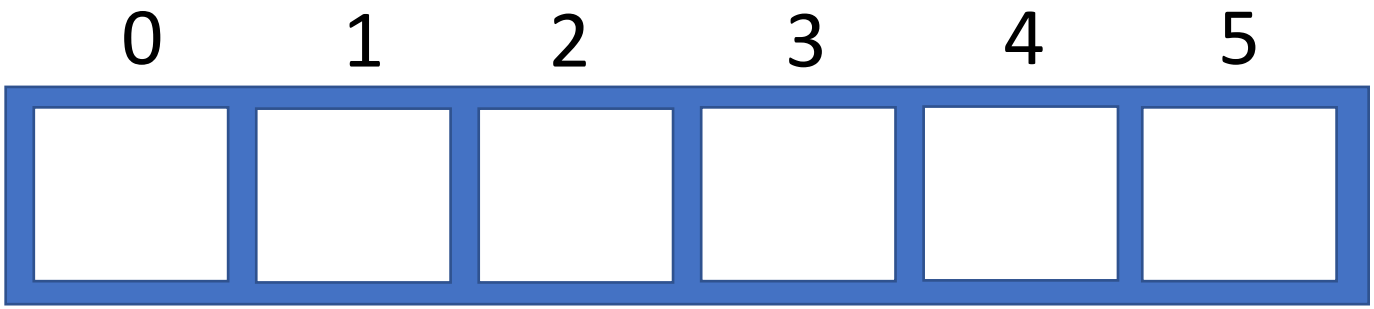
Comprendre les tableaux

Base de l'implémentation d'une ArrayList

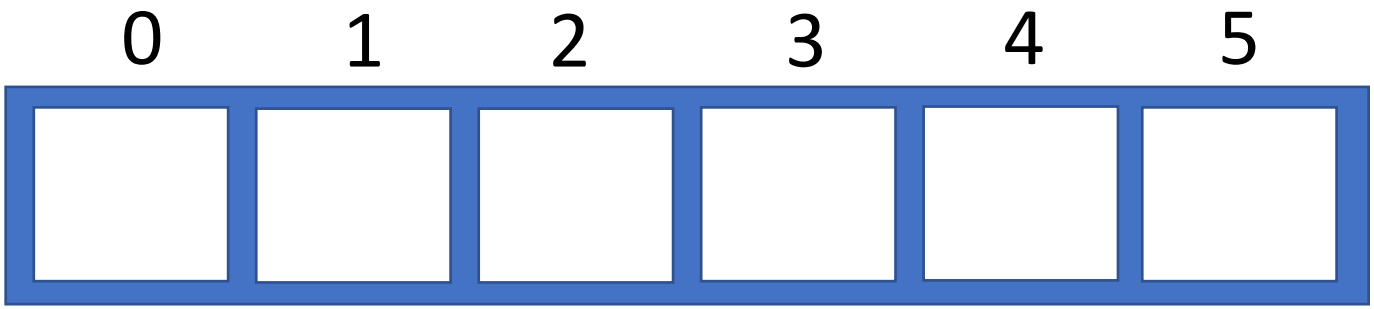
Université de Montpellier
Faculté des sciences
Mars 2020

Tableaux : une structure rigide et fixe



- Tableau de 6 cases = comme une boîte avec 6 emplacements

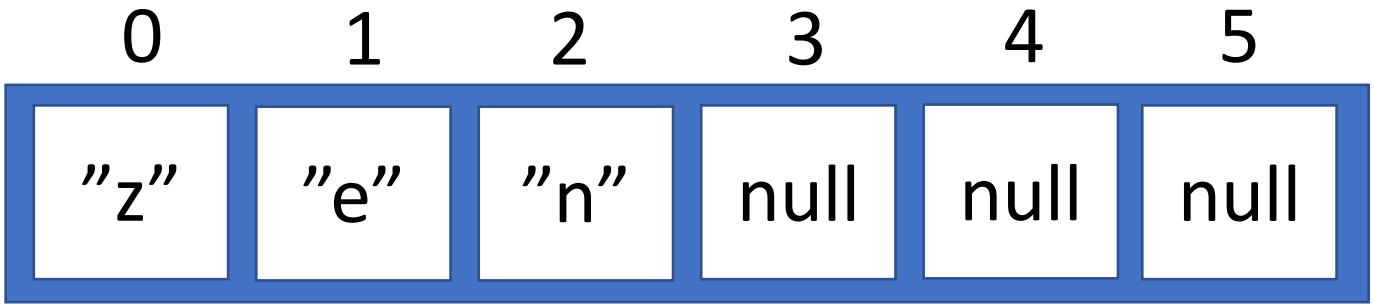
Tableaux : une structure rigide et fixe



- Tableau de 6 cases = comme une boîte avec 6 emplacements

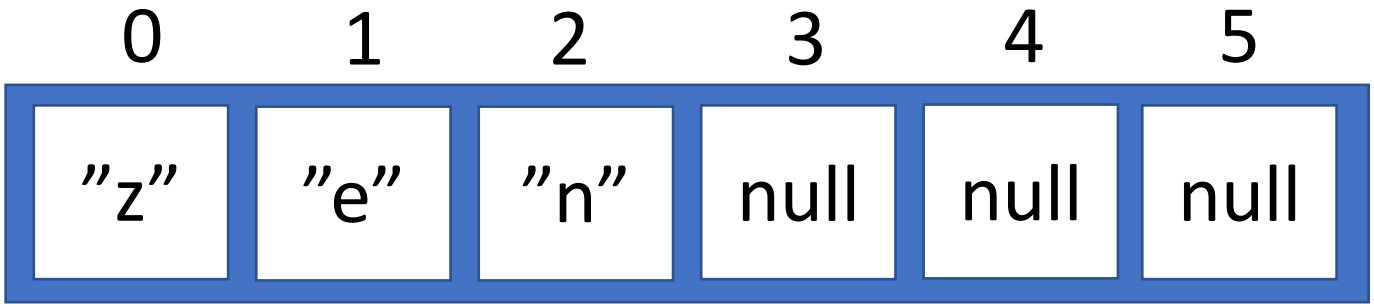
```
// création du tableau
String[] tableau = new String[6];
// par défaut il contient null dans toutes les cases
System.out.println(Arrays.toString(tableau));
```

Tableaux : une structure rigide et fixe



- Tableau de String
- On veut stocker les chaînes “z”, “e” et “n”
- Que laisse-t-on dans les autres cases ? **null**

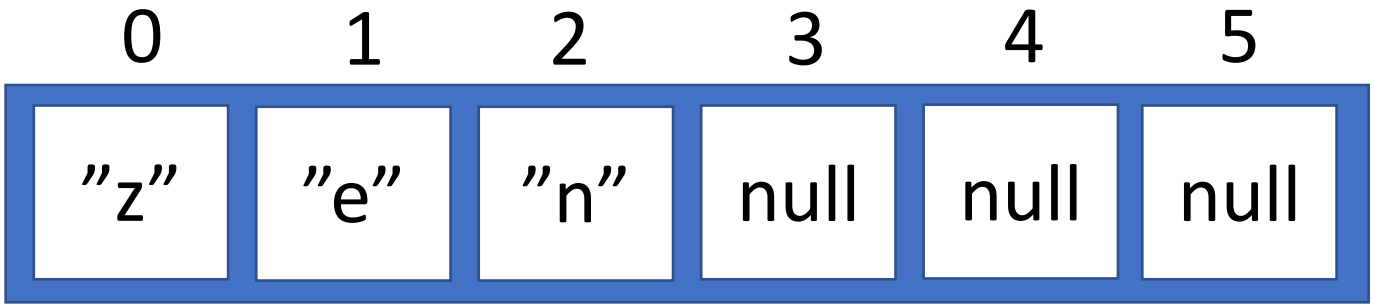
Tableaux : une structure rigide et fixe



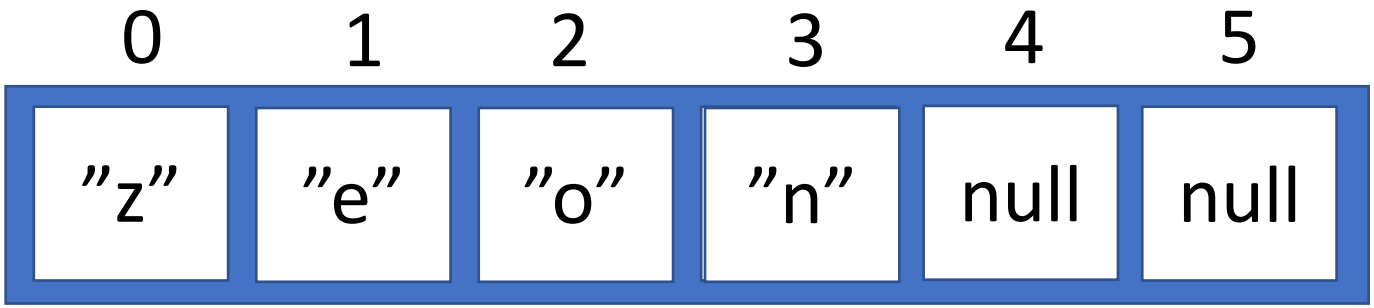
- On veut stocker les chaînes “z”, “e” et “n”

```
tableau[0]="z";
tableau[1]="e";
tableau[2]="n";
```

Tableaux : une structure rigide et fixe



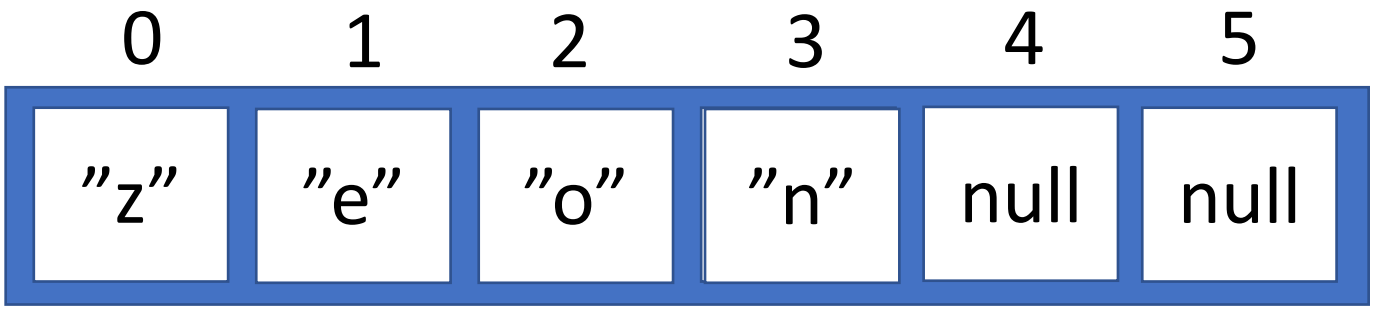
- On veut ajouter “o” **entre** “e” et “n”
- On doit **décaler** “n” dans la case de droite, case 2 vers case 3



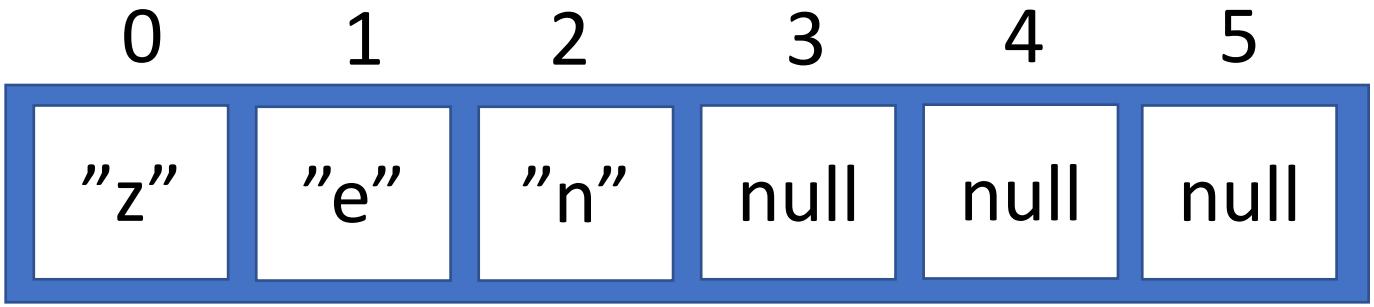
Tableaux : une structure rigide et fixe

- On veut ajouter “o” **entre** “e” et “n”
- On doit **décaler** “n” dans la case de droite (case 3)

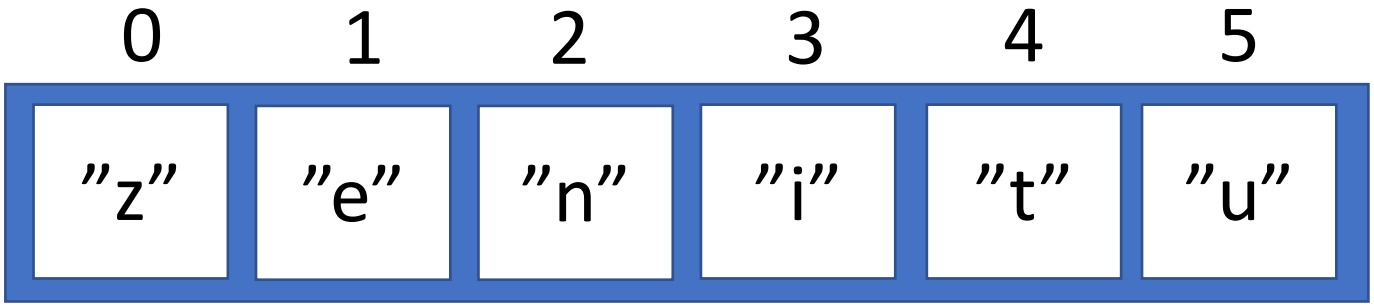
```
tableau[3]=tableau[2]; // décalage
tableau[2]="o";
```



Tableaux : une structure rigide et fixe



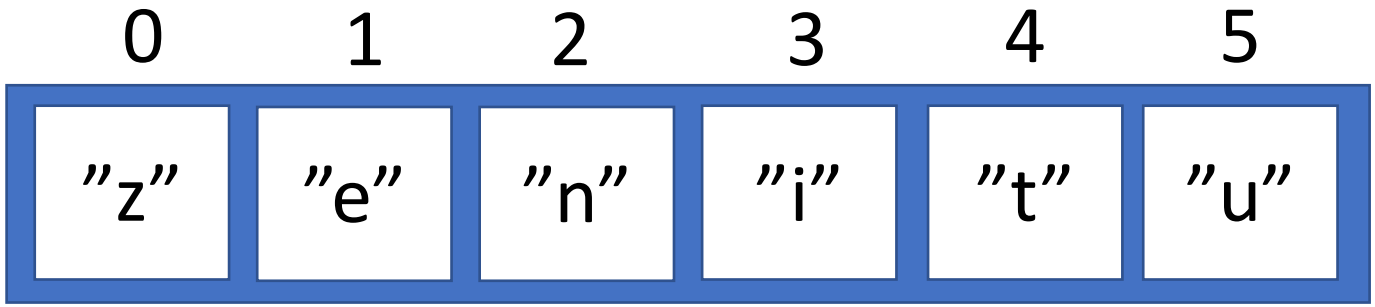
- En repartant du tableau initial, on veut ajouter “i”, “t”, ”u”, ”d”, “e”
- On commence à remplir mais on manque de place pour “d” et “e”



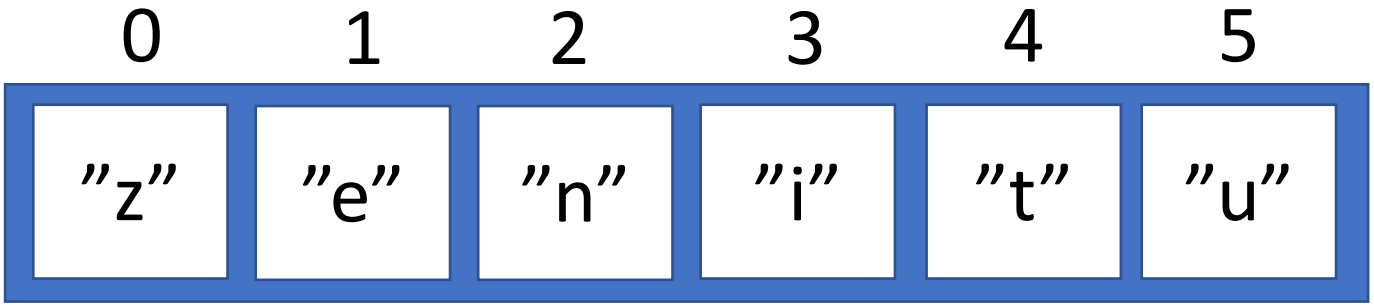
Tableaux : une structure rigide et fixe

- On veut ajouter “i”, “t”, ”u”, ”d”, “e”
- On commence à remplir

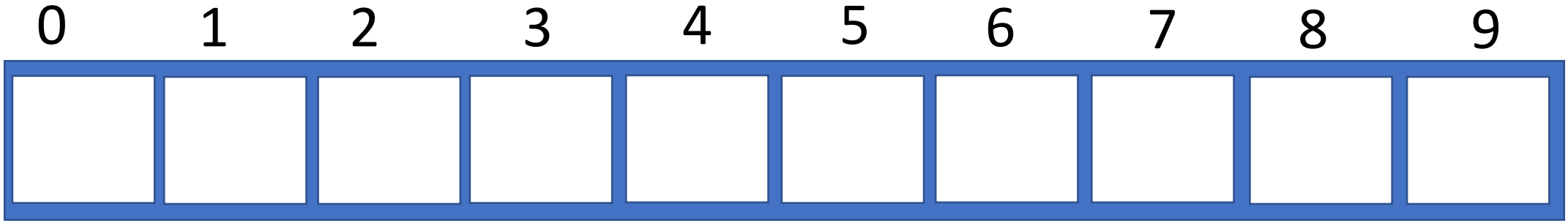
```
tableau[3]="i";
tableau[4]="t";
tableau[5]="u";
```



Tableaux : une structure rigide et fixe

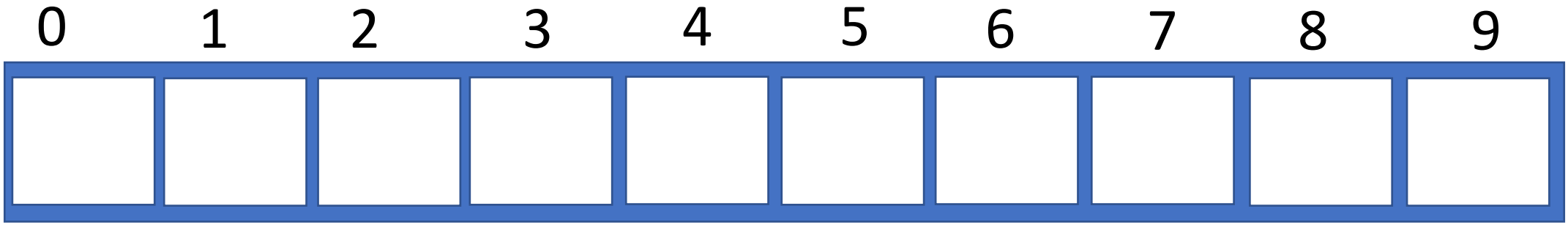


- On crée un tableau plus grand (assez grand mais pas trop)



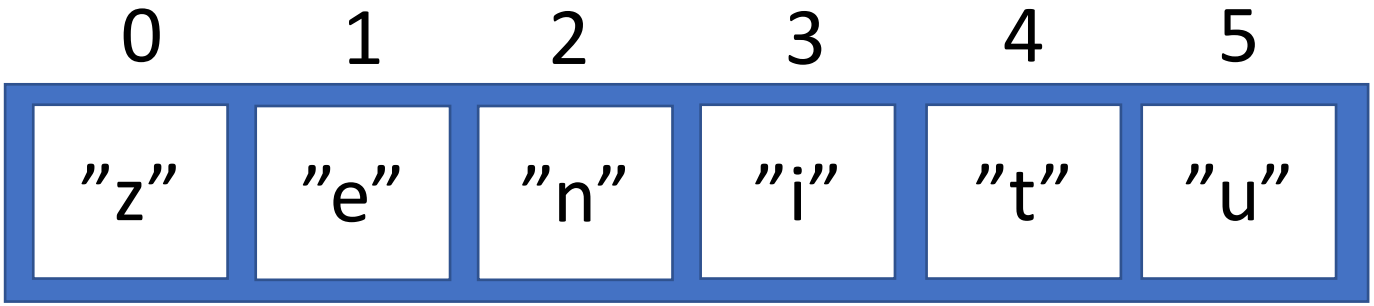
Tableaux : une structure rigide et fixe

- On crée un tableau plus grand (assez grand mais pas trop)

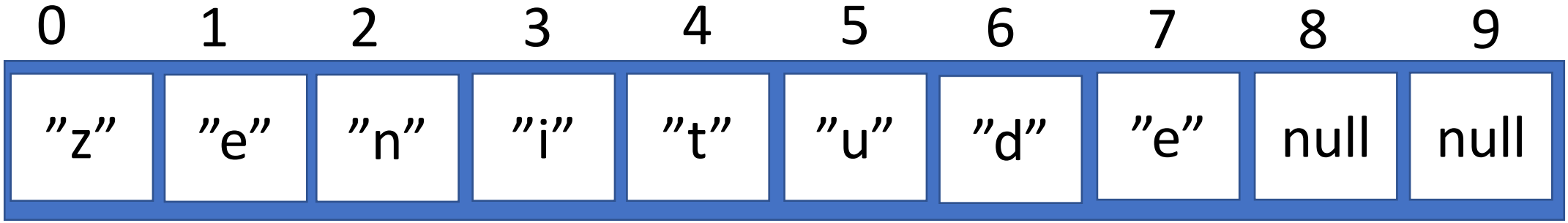


String[] tableauPlusGrand = **new** String[10];

Tableaux : une structure rigide et fixe

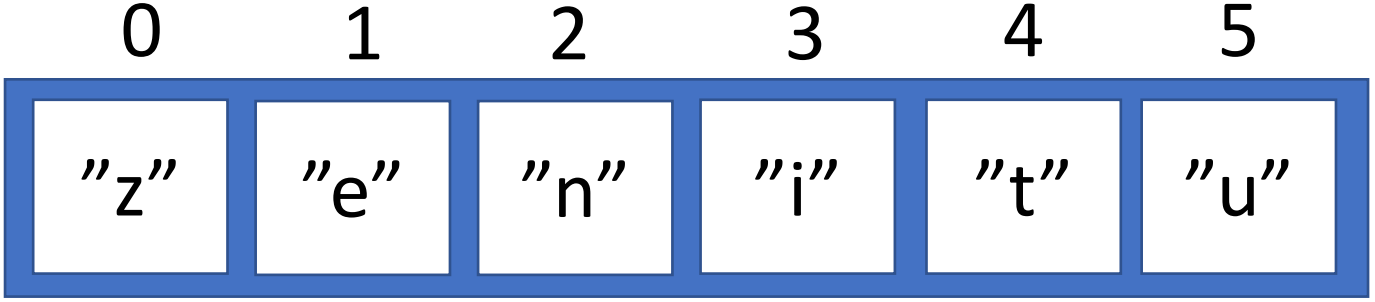


- On recopie le petit dans le grand et on complète



- Puis le premier peut disparaître, ce qui libère la mémoire

Tableaux : une structure rigide et fixe

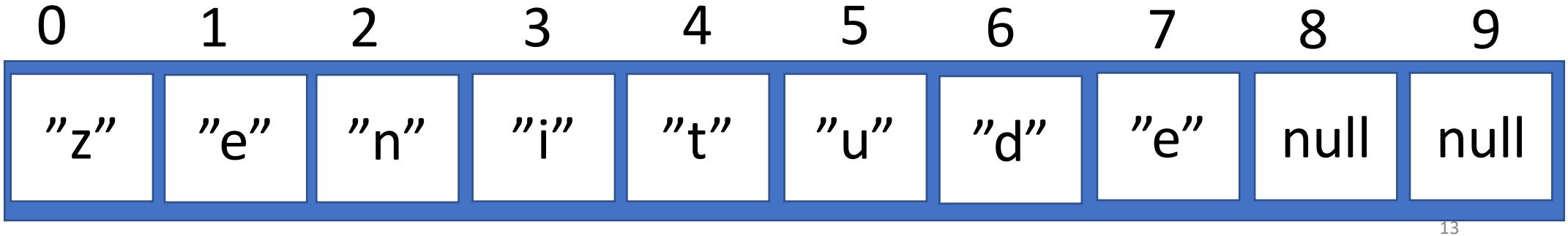


On recopie le petit dans le grand

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

On complète

```
tableauPlusGrand[6]="d";
tableauPlusGrand[7]="e";
```



Puis on refait la référence : tableau = tableauPlusGrand; Et l’ancienne zone disparaîtra

Tableaux : une structure rigide et fixe

- En synthèse :
  - C'est facile de remplir à partir du début et de compléter par la fin
  - La place occupée est seulement celle correspondant aux valeurs
  - L'accès à un élément est facile à partir d'un indice entier, ex. Tableau[4]
- Mais :
  - Quand on veut ajouter au milieu : on doit décaler vers la droite
  - Quand on ajoute et que c'est plein : on doit créer un tableau plus grand, il n'est pas évident de savoir de quelle quantité agrandir, et ensuite on doit tout recopier avant d'ajouter les nouveaux éléments
  - Si on faisait un retrait : il faut "reboucher" le trou en décalant ou choisir de laisser des trous mais les verifier après
  - On doit mémoriser dans une variable à part le nombre d'éléments insérés (car tableau.length donne le nombre total de cases mais pas le nombre de cases occupées)

Tableaux : une structure rigide et fixe

- Ce procédé est celui utilisé dans les ArrayList

- On ne s'aperçoit pas de toutes les complications quand on utilise une ArrayList, car les méthodes réalisent les traitements nécessaires :
  - décalage, agrandissement, etc.

- Et si vous voulez aller plus loin, on peut lire le code source de la ArrayList par exemple ici :

http://www.docjar.com/html/api/java/util/ArrayList.java.html