Conception objet en Java avec BlueJ

une approche interactive

4. Groupement d'objets

Collections et itérateurs

David J. Barnes, Michael Kölling

version française: Patrice Moreaux

Principaux concepts étudiés

- Collections
- Boucles
- Itérateurs
- Tableaux

La nécessité de regrouper les objets

- Beaucoup d'applications mettent en jeu des collections d'objets:
  - organiseurs personnels
  - catalogues de bibliothèque
  - système de gestion d'étudiants
- Le nombre d'éléments (item, entrée, enregistrement) à stocker est variable
  - items ajoutés
  - items supprimés

Un agenda électronique

- Stocker des rendez-vous.
- Visualiser les informations d'un rendez-vous.
- Nombre d'entrées illimité.
- Connaître de nombre d'entrées.
- Étudiez le projet *notebook1* ..

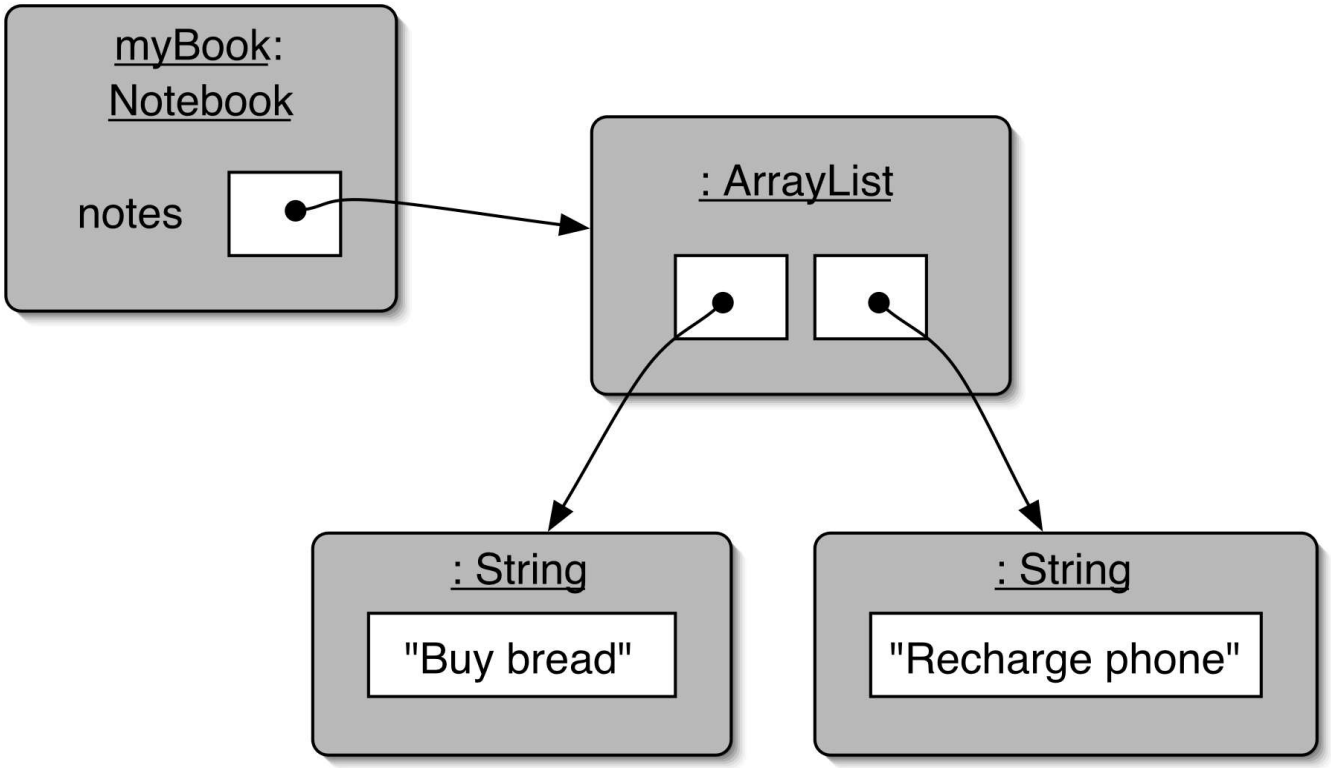
Bibliothèques de classes (1)

- Ensemble de classes utiles.
- Ne pas réécrire tout à partir de zéro.
- Les bibliothèques Java s'appellent des *paquetages (packages)* .
- Il est très fréquent de devoir regrouper des objets.
  - Le paquetage **`java.util`** contient des classes pour cela.

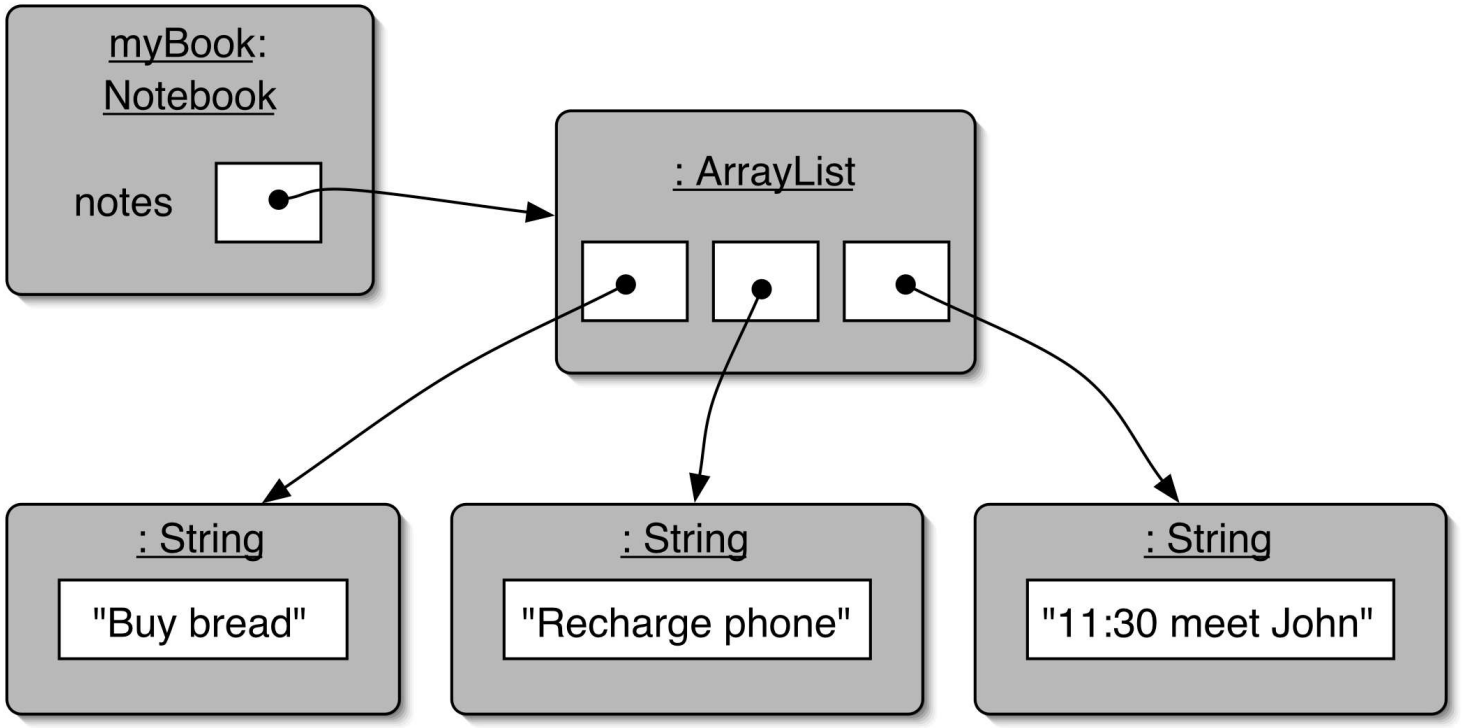
Bibliothèques de classes (2)

```
import java.util.ArrayList;

/**
 * ...
 */
public class Notebook
{
    // Storage for an arbitrary number of notes.
    private ArrayList notes;

    /**
     * Perform any initialization required for the
     * notebook.
     */
    public Notebook()
    {
        notes = new ArrayList();
    }

    ...
}
```

Structures objets avec les collections



Ajouter une troisième note



Propriétés d'une collection

- Augmente sa capacité si nécessaire.
- Maintient un compteur privé (accesseur **`size()`**).
- Conserve les objets dans l'ordre.
- Les détails de réalisation sont masqués.
  - est-ce important? Cela nous empêche-t- il de l'utiliser?

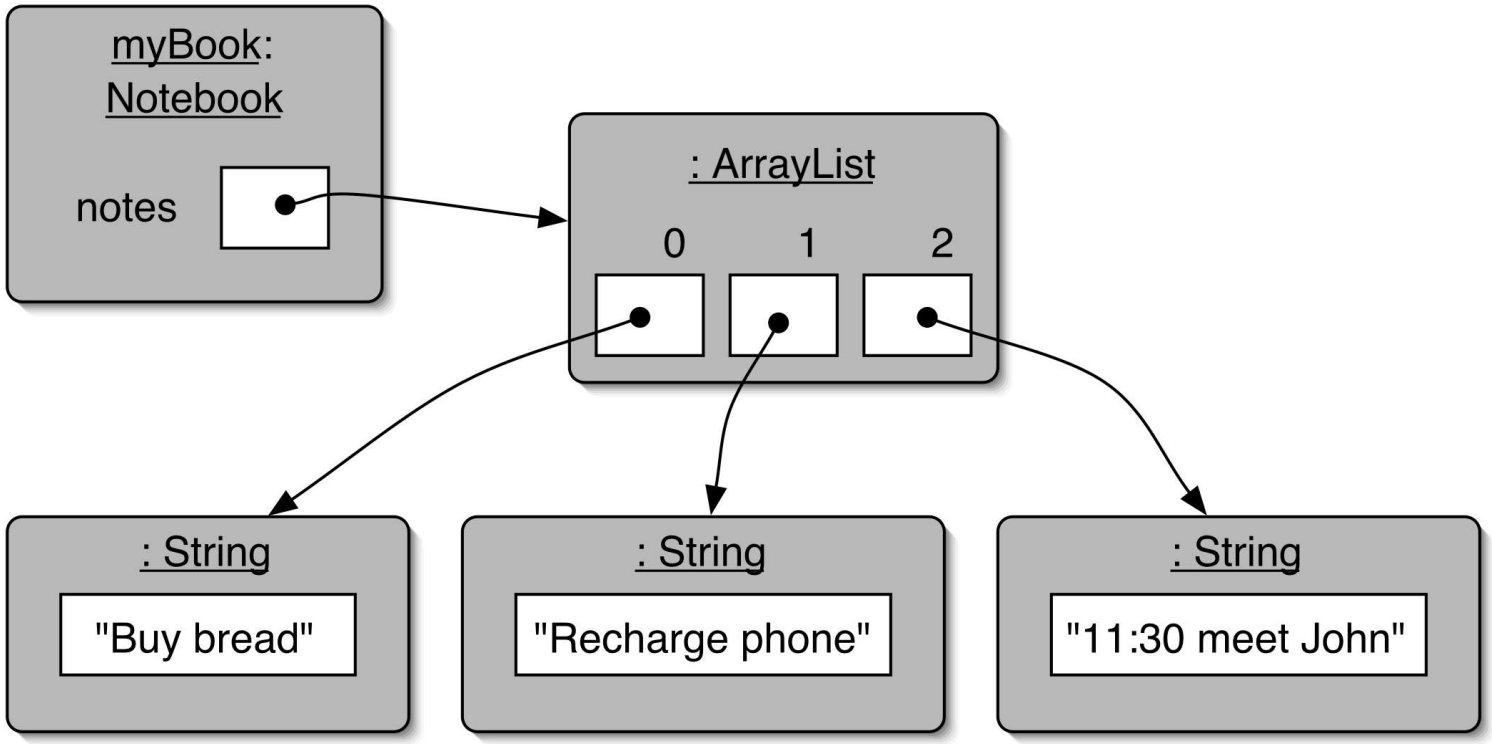
Utiliser une collection

```
public class Notebook
{
    private ArrayList notes;
    ...

    public void storeNote(String note)
    {
        notes.add(note);
    }

    public int numberOfNotes()
    {
        return notes.size();
    }

    ...
}
```

Ajouter une nouvelle entrée

Renvoyer le nombre d'entrées (*délégation*)

Numérotation par index



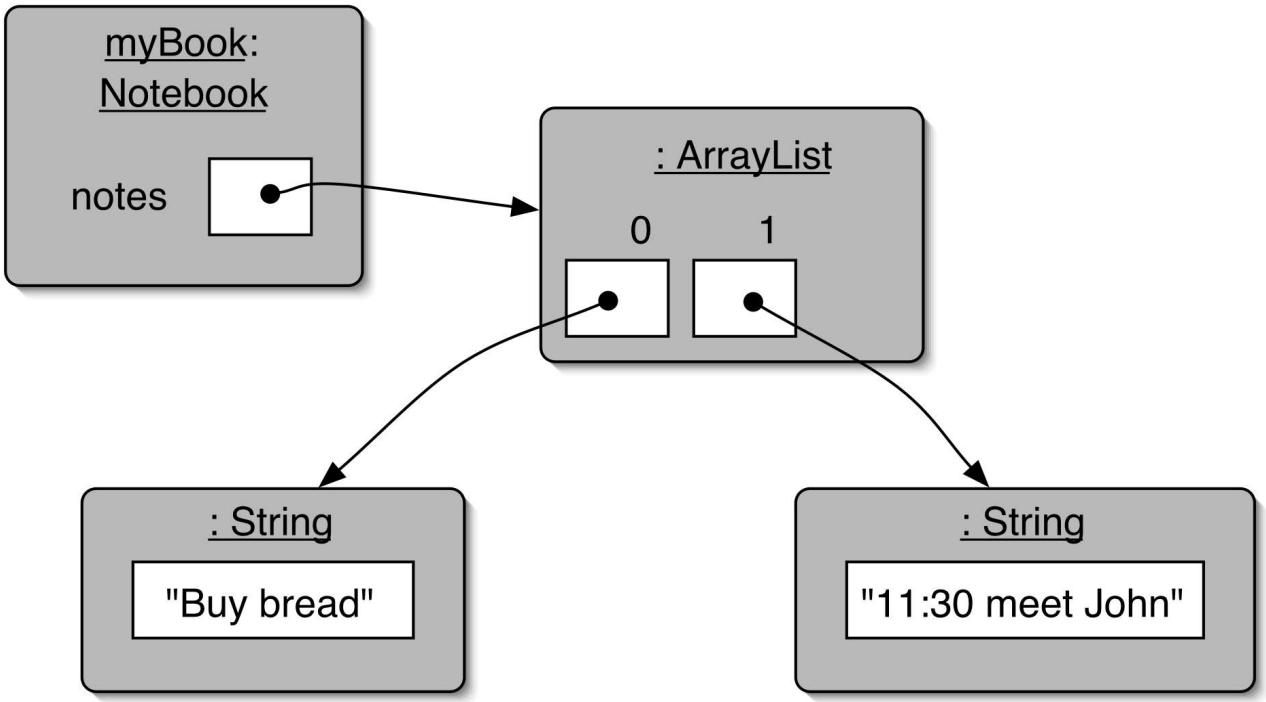
Obtenir un objet

```
public void showNote(int noteNumber)
{
    if(noteNumber < 0) {
        // This is not a valid note number.
    }
    else if(noteNumber < numberOfNotes) {
        System.out.println(notes.get(noteNumber));
    }
    else {
        // This is not a valid note number.
    }
}
```

Contrôle de validité des indices

Obtenir et afficher la note

La suppression peut modifier la numérotation



Résumé (1)

- Les collections permettent le stockage d'un nombre quelconque d'objets.
- Les bibliothèques (*libraries*) de classes contiennent en général des classes collection déjà vérifiées et testées.
- Les bibliothèques de classes Java s'appellent des *paquetages (packages)*.
- Nous avons utilisé la classe **`ArrayList`** du paquetage **`java.util`**.

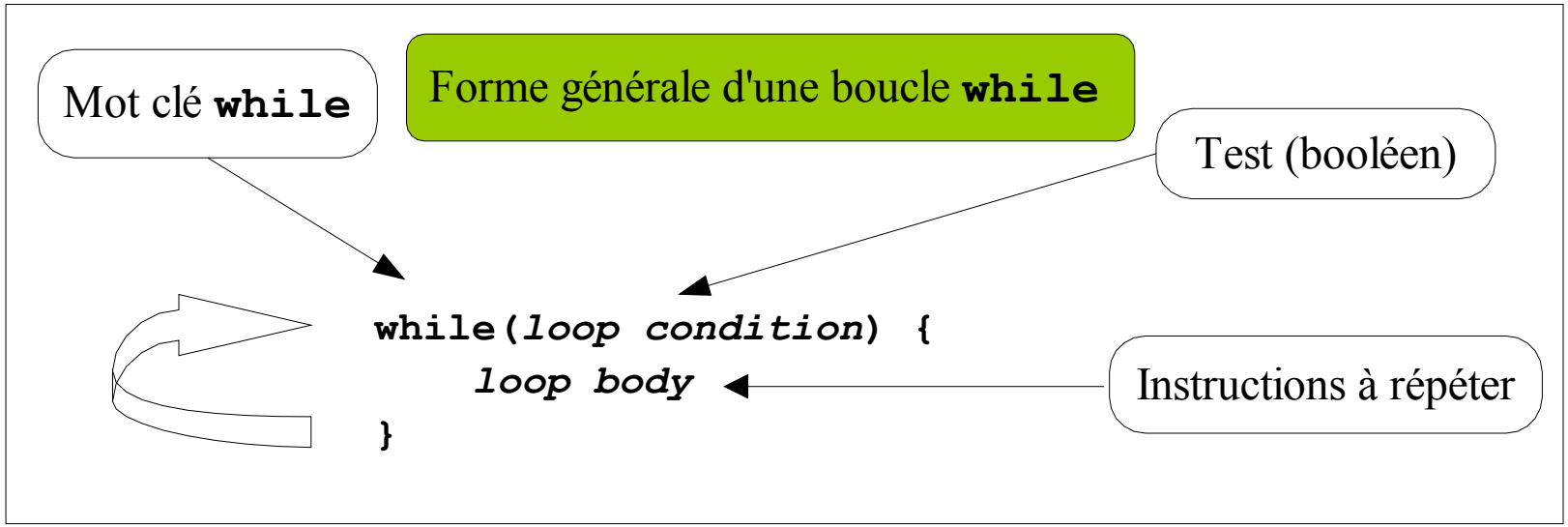
Résumé (2)

- Des entrées peuvent être ajoutées ou supprimées.
- Chaque entrée a un indice.
- Les valeurs des indices peuvent changer si des entrées sont supprimées (ou d'autres ajoutées).
- Les principales méthodes de **`ArrayList`** sont **`add`**, **`get`**, **`remove`** et **`size`**.

Itérations

- Il est souvent nécessaire de répéter certaines actions un nombre arbitraire de fois. Exemple:
  - afficher toutes les informations de l'agenda. Combien y a-t-il d'entrées?
- La plupart des langages de programmation disposent *d'instructions de boucle* pour cela.
- Java possède trois instructions de boucle.
  - Nous étudions sa boucle **`while`** (*tant que*).

Boucle **while** (tant que) pseudo-code



Exemple de pseudo-code pour afficher toutes les notes

```
while(il y a au moins une note à afficher) {
    afficher la note suivante
}
```

Un exemple en Java

```
/**
 * Liste toutes les notes de l'agenda.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        System.out.println(notes.get(index));
        index++;
    }
}
```

Incrément de 1

Itérer sur une collection

`java.util.Iterator`

retourne un objet `Iterator`

```
Iterator it = myCollection.iterator();
while(it.hasNext()) {
    appeler it.next() pour obtenir l'objet suivant
    faire quelque chose avec cet objet
}
```

```
public void listNotes()
{
    Iterator it = notes.iterator();
    while(it.hasNext()) {
        System.out.println(it.next());
    }
}
```

Le projet *auction*

- Le projet *auction* fournit une nouvelle illustration des collections et de l'itération.
- Deux points à souligner:
  - la valeur **`null`** .
  - le transtypage (*casting*). Utilisé pour stocker le résultat de **`get`** dans une variable:
  -
```
String message =
                (String)notes.get(0);
```

Résumé

- Les instructions de boucle permettent l'exécution répétée d'un bloc d'instructions.
- Une boucle **`while`** Java contrôle la répétition avec une expression booléenne.
- Les classes collection possèdent des objets **`Iterator`** spéciaux qui simplifient l'itération sur l'ensemble de la collection.

Collections de taille fixe

- La taille maximale d'une collection peut parfois être déterminée à l'avance.
- Les langages de programmation disposent en général d'un type spécial de collection de taille fixe: le *tableau* (*array*).
- En Java, les tableaux peuvent stocker des objets ou des valeurs de types primitifs.
- Les tableaux utilisent une syntaxe particulière.

Le projet *weblog-analyzer*

- Un serveur Web enregistre les détails de chaque accès.
- Aide au travail du webmaster
  - pages les plus lues.
  - périodes d'activité maximale.
  - quantité de données transférées.
  - références obsolètes.
- Analyse heure par heure des accès.

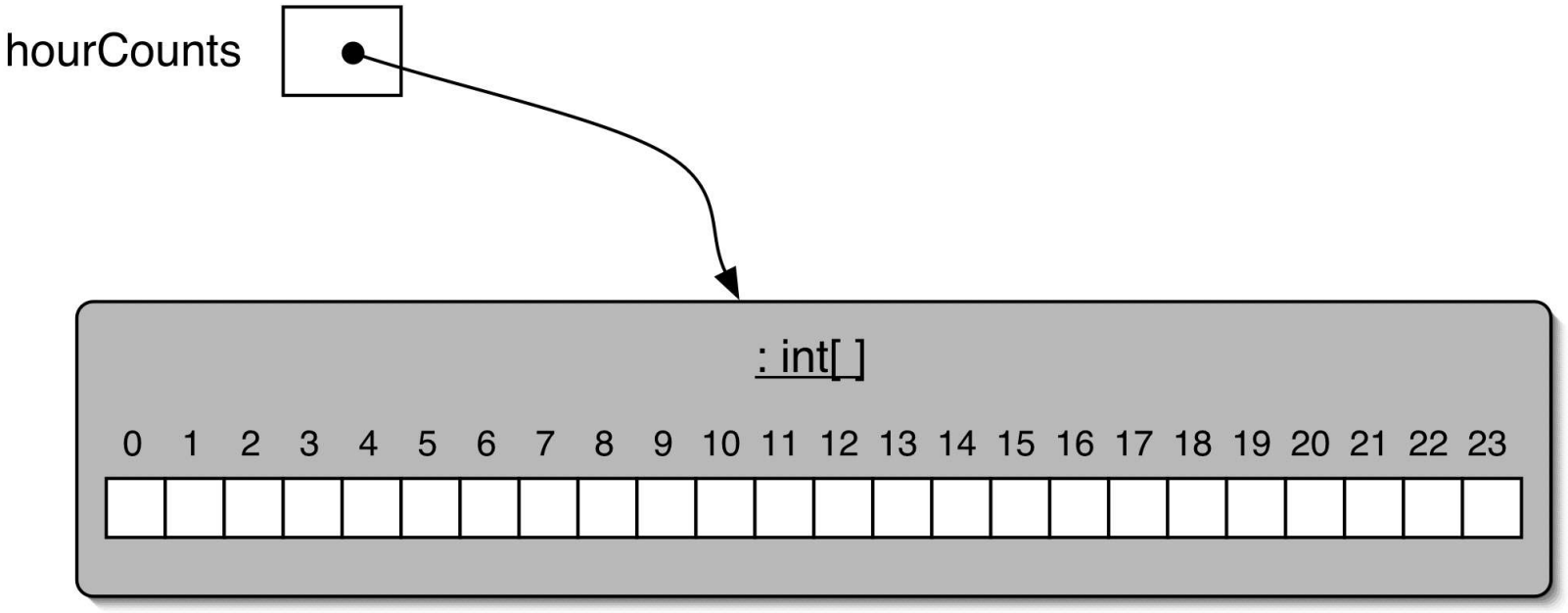
Créer un objet **`array`** (tableau)

```
public class LogAnalyzer
{
    private int[] hourCounts;
    private LogfileReader reader;

    public LogAnalyzer()
    {
        hourCounts = new int[24];
        reader = new LogfileReader();
    }
    ...
}
```

Déclaration de variable **`Array`**

Création d'un objet **`Array`**

Le tableau `hourCounts`



Utiliser un tableau

- On emploie la notation entre crochets pour accéder à un élément de tableau: **`hourCounts[...]`**
- Les éléments sont utilisés comme des variables ordinaires.
  - à gauche d'une affectation:
    - **`hourCounts[hour] = ...;`**
  - dans une expression:
    - **`adjusted = hourCounts[hour] - 3;`**
    - **`hourCounts[hour]++;`**

Boucle **for** (pour)

- Même principe que pour la boucle **`while`**.
- Souvent utilisée pour itérer un nombre connu de fois.
- Souvent employée pour itérer sur un tableau.

Boucle **for** – pseudo-code

Forme générale d'une boucle **`for`**

```
for(initialisation; condition; action après le corps) {
    instructions à répéter
}
```

Équivalent avec une boucle **`while`**

```
initialisation;
while(condition) {
    instructions à répéter
    action après le corps
}
```

Un exemple en Java

Version boucle **for**

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

Version boucle **while**

```
int hour = 0;
while(hour < hourCounts.length) {
    System.out.println(hour + ": " + hourCounts[hour]);
    hour++;
}
```

Résumé

- Utiliser les tableaux (**`array`**) lorsqu'une collection de taille fixe est nécessaire.
- Les tableaux emploient une syntaxe spéciale.
- On peut employer une boucle **`for`** au lieu d'une boucle **`while`** lorsque le nombre de répétitions est connu à l'avance.
- Les boucles **`for`** sont souvent utilisées pour itérer sur les tableaux.

Sommaire général