Grouping objects

Collections and iterators

The requirement to group objects

- Many applications involve collections of objects:
    - Personal organizers.
    - Library catalogs.
    - Student-record system.
- The number of items to be stored varies.
    - Items added.
    - Items deleted.

Fixed-size collections

- Sometimes the maximum collection size can be pre-determined.

- Programming languages usually offer a special fixed-size collection type: an *array*.

- Java arrays can store objects or primitive-type values.

- Arrays use a special syntax.

The *weblog-analyzer* project

- Web server records details of each access.

- Supports webmaster's tasks.
  - Most popular pages.
  - Busiest periods.
  - How much data is being delivered.
  - Broken references.

- Analyze accesses by hour.

Creating an array object

```java
public class LogAnalyzer {

    private int[] hourCounts;        ← Array variable declaration
    private LogfileReader reader;


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

The `hourCounts` array

hourCounts

: int[ ]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
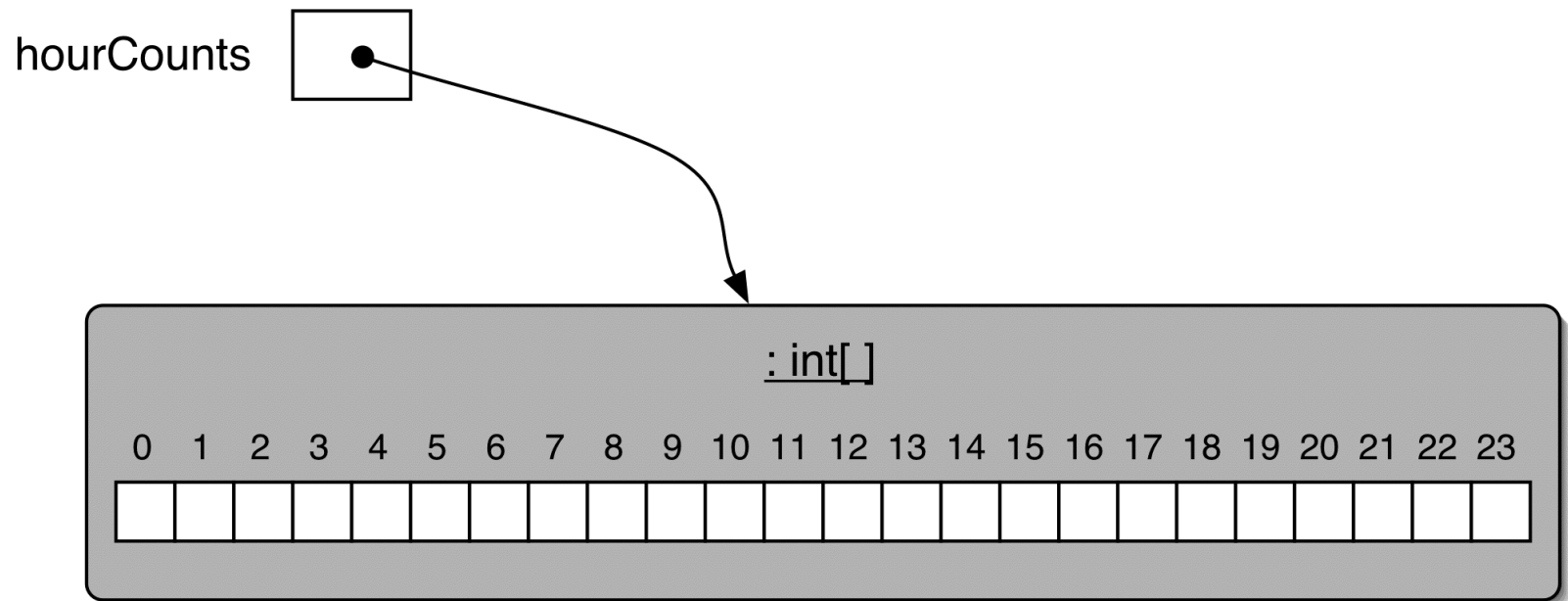
Using an array

- Square-bracket notation is used to access an array element: `hourCounts[...]`

- Elements are used like ordinary variables.
  - On the left of an assignment:
    - `hourCounts[hour] = ...;`
  - In an expression:
    - `adjusted = hourCounts[hour] – 3;`
    - `hourCounts[hour]++;`

Variable-size collections

- But sometimes the maximum collection size cannot be determined!

- Java has us covered here as well ☺

A personal notebook

- Notes may be stored.
- Individual notes can be viewed.
- There is no limit to the number of notes.
- It will tell how many notes are stored.
- Explore the *notebook1* project.

Class libraries

- Collections of useful classes.
- We don't have to write everything from scratch.
- Java calls its libraries, *packages*.
- Grouping objects is a recurring requirement.
  - The `java.util` package contains classes for doing this.

```java
import java.util.ArrayList;

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

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

    ...
}
```
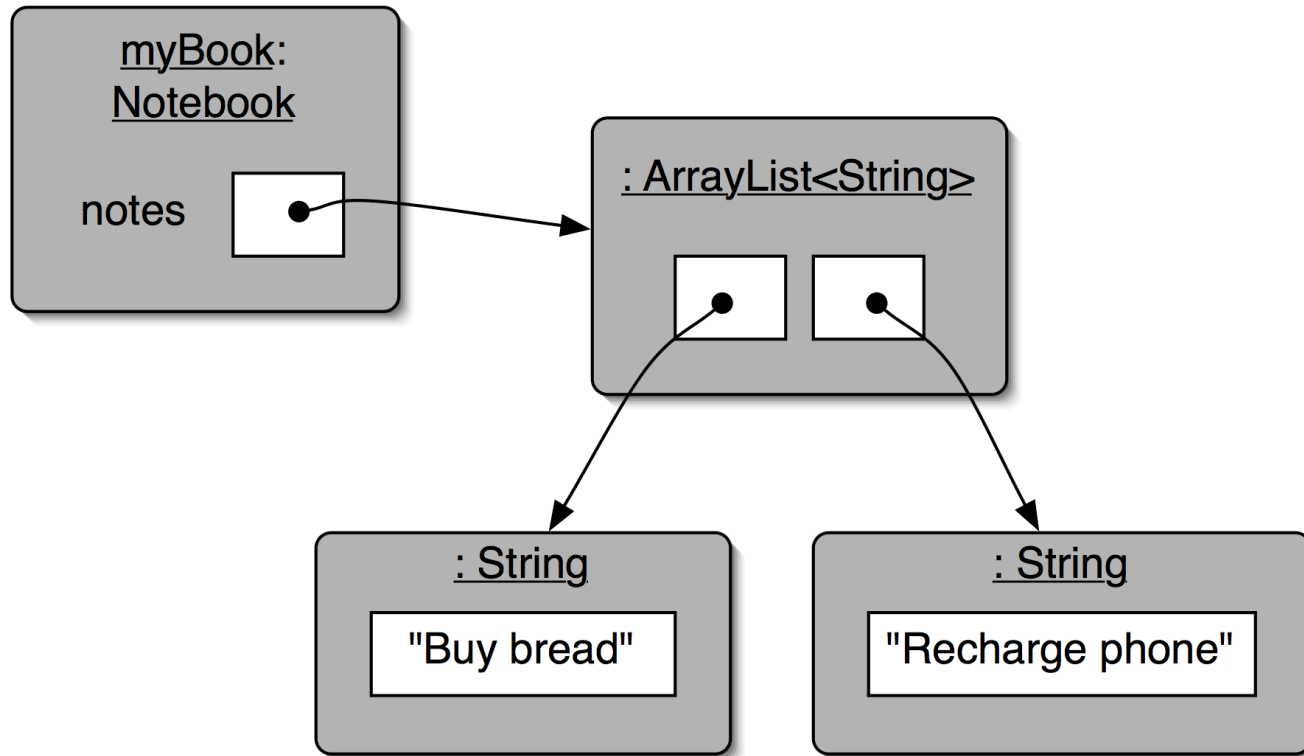
Collections

`ArrayList<String> notes;`

- We specify:
  - the type of collection: `ArrayList`
  - the type of objects it will contain: `<String>`

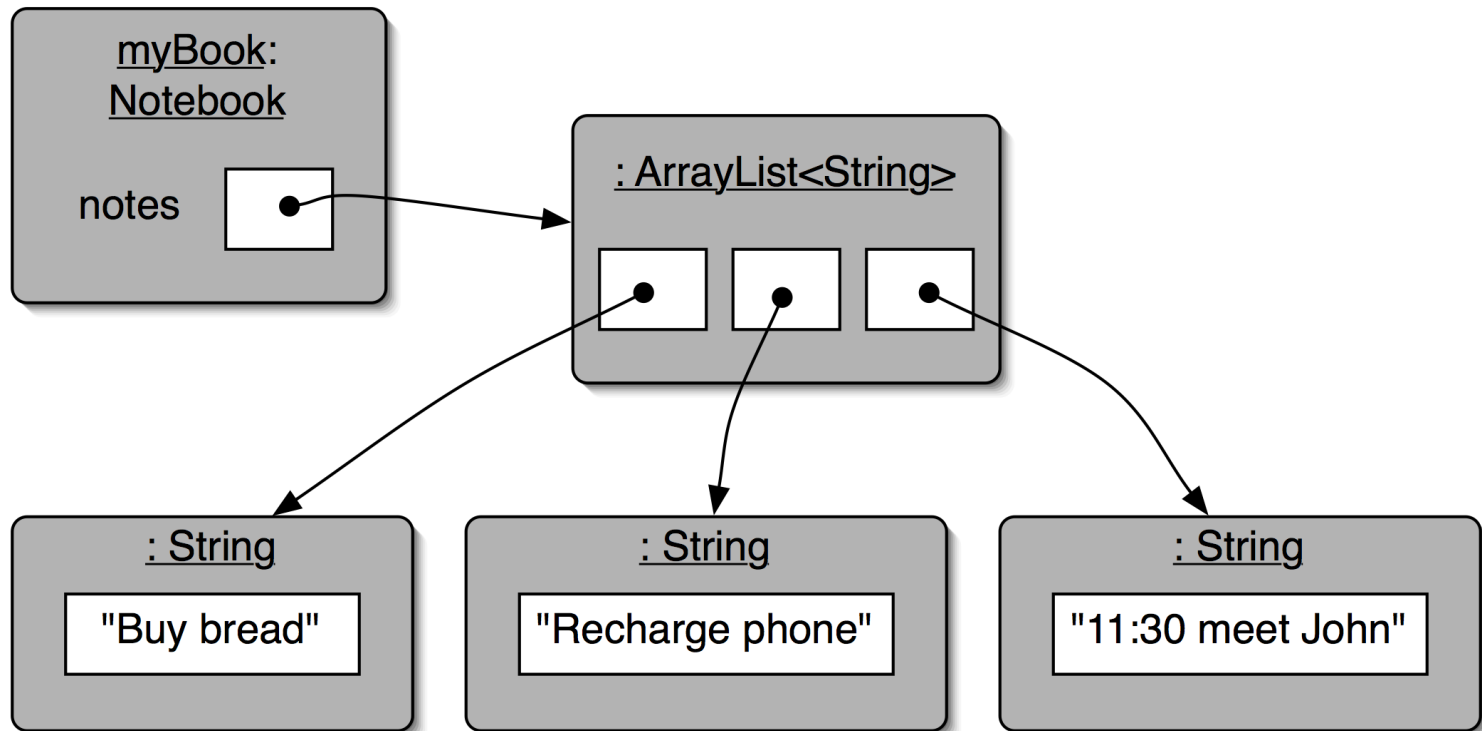- We say, "ArrayList of String".

Generic classes

- Collections are known as *parameterized* or *generic* types.

- `ArrayList` implements list functionality:
  - **add**, **get**, **size**, etc.

- The type parameter says what we want a list of:
  - **ArrayList<Person>**
  - **ArrayList<TicketMachine>**
  - etc.

Object structures with collections

Adding a third note

Features of the collection

- It increases its capacity as necessary.
- It keeps a private count (`size()` accessor).
- It keeps the objects in order.
- Details of how all this is done are hidden.
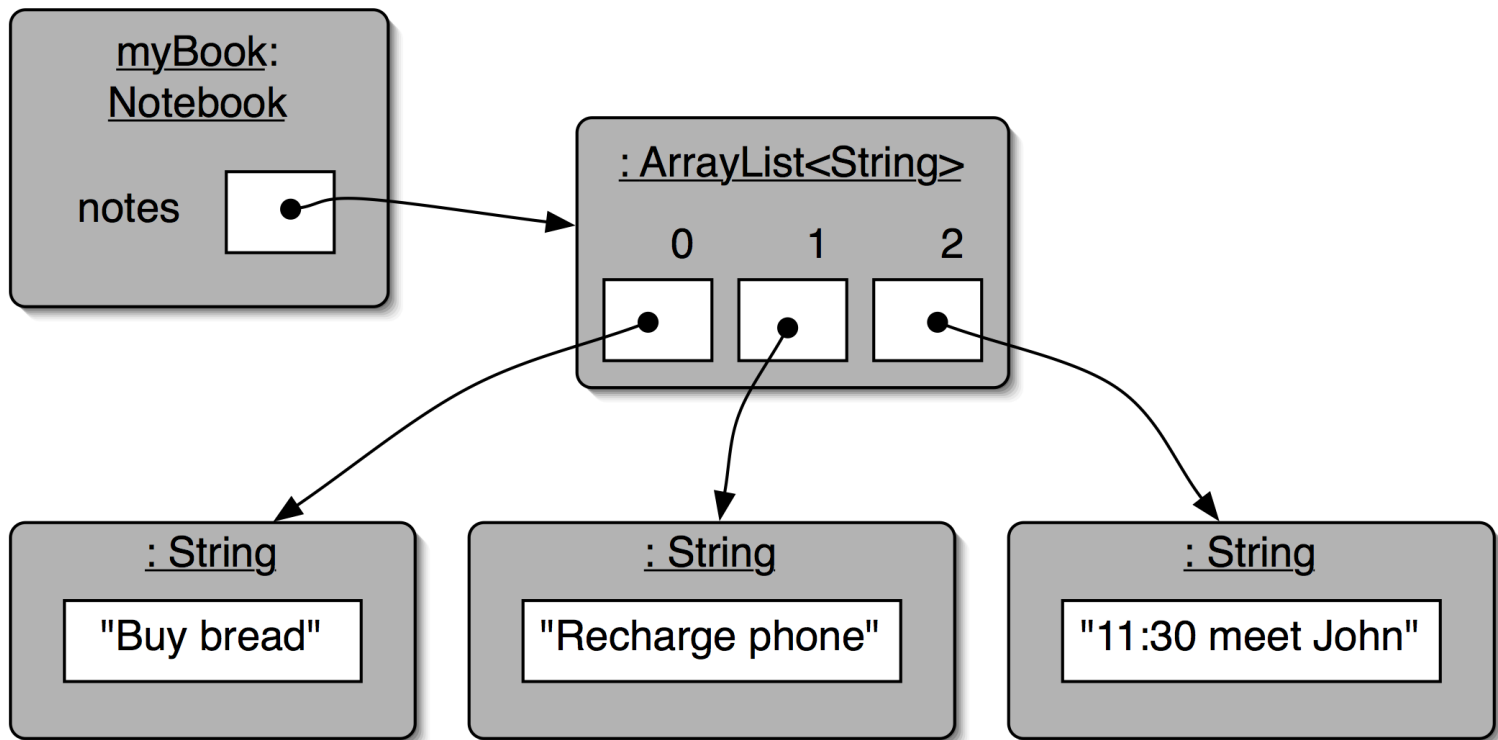   - Does that matter? Does not knowing how prevent us from using it?

Using the collection

```java
public class Notebook {

    private ArrayList<String> notes;
    ...

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

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

    ...
}
```

Adding a new note

Returning the number of notes
(*delegation*)

Index numbering

Retrieving an object

Index validity checks

```java
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.
    }
}
```
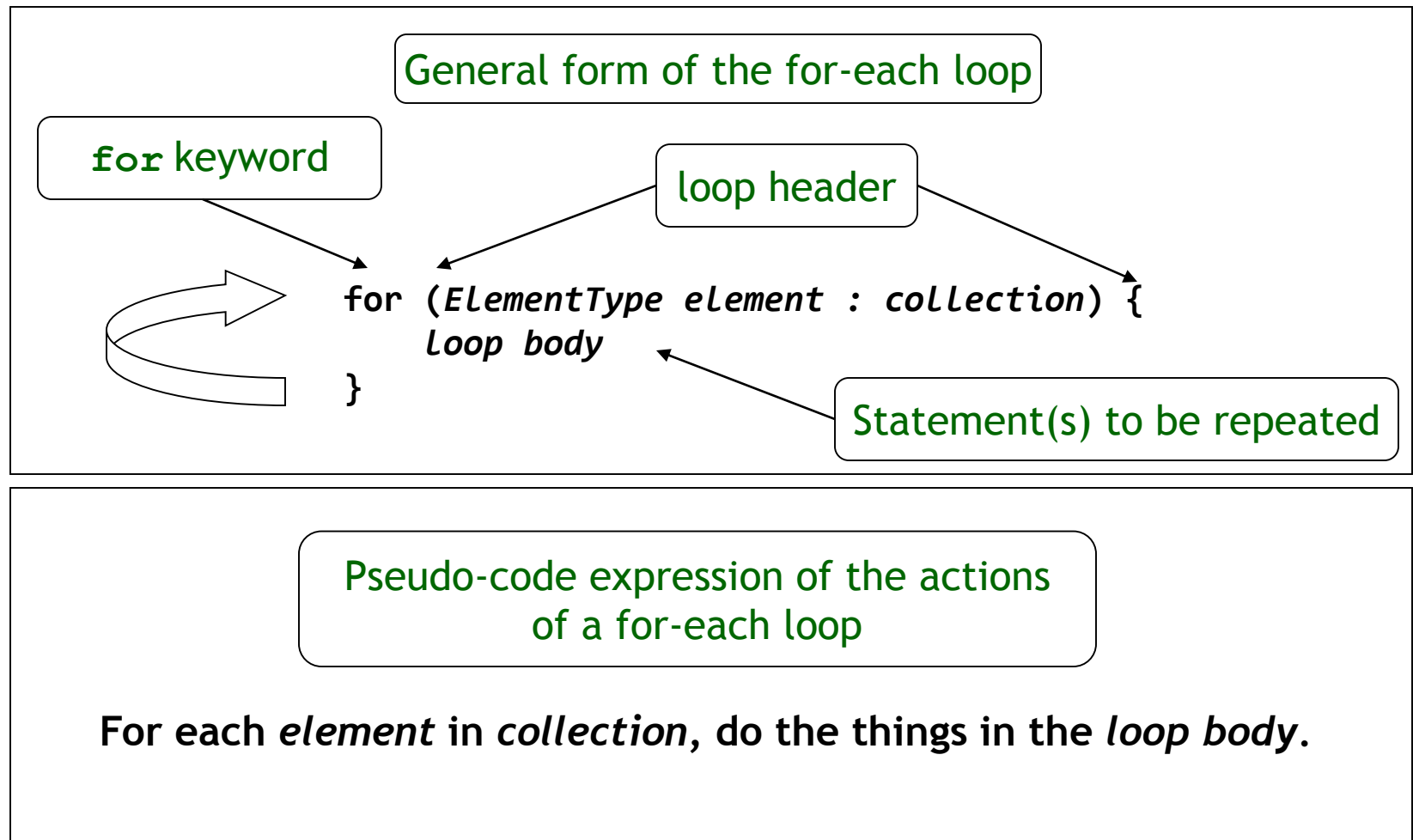
Retrieve and print the note

Iteration

- We often want to perform some actions an arbitrary number of times.
    - E.g., print all the notes in the notebook. How many are there?

- Most programming languages include *loop statements* to make this possible.

- Java has several sorts of loop statement.
    - We will start with its *for-each loop*.

For-each loop pseudo code

General form of the for-each loop

for keyword

loop header

```
for (ElementType element : collection) {
    loop body
}
```

Statement(s) to be repeated

Pseudo-code expression of the actions
of a for-each loop

For each *element* in *collection*, do the things in the *loop body*.

A Java example

```java
/**
 * List all notes in the notebook.
 */
public void listNotes() {
    for (String note : notes) {
        System.out.println(note);
    }
}
```

for each *note* in *notes*, print out *note*

for-each versus while

- for-each:
  - easier to write.
  - safer: it is guaranteed to stop.
- while:
  - we don't *have* to process the whole collection.
  - doesn't even have to be used with a collection.
  - take care: could be an *infinite loop.*

Searching a collection

```
int index = 0;
boolean found = false;
while (index < notes.size() && !found) {
    String note = notes.get(index);
    if (note.contains(searchString)) {
        found = true;
        // We don't need to keep looking.
        break;
    }
    else {
        index++;
    }
}
// Either we found it, or we searched the whole
// collection.
```

Removal may affect numbering
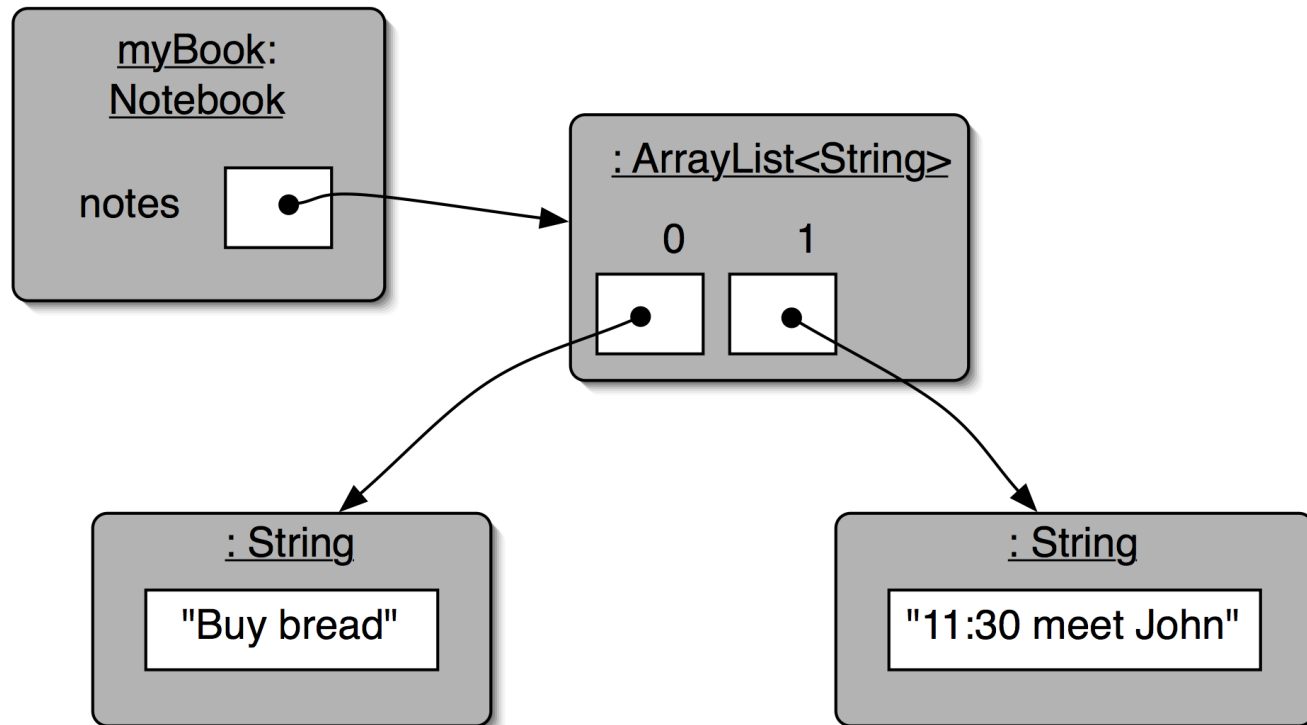
Using an Iterator object

java.util.Iterator

returns an **Iterator** object

```
Iterator<ElementType> it = myCollection.iterator();
while (it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

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

Index versus Iterator

- Ways to iterate over a collection:
  - for-each loop.
    - Use if we want to process every element.
  - while loop.
    - Use if we might want to stop part way through.
    - Use for repetition that doesn't involve a collection.
  - Iterator object.
    - Use if we might want to stop part way through.
    - Often used with collections where indexed access is not very efficient, or impossible.
- Iteration is an important programming *pattern.*

Review

- Arrays are appropriate where a fixed-size collection is required.

- Arrays use special syntax.

- For loops offer an alternative to while loops when the number of repetitions is known.

- For loops are used when an index variable is required.

Review

- Collections allow an arbitrary number of objects to be stored.

- Class libraries usually contain tried-and-tested collection classes.

- Java's class libraries are called *packages*.

- We have used the `ArrayList` class from the `java.util` package.

Review

- Items may be added and removed.
- Each item has an index.
- Index values may change if items are removed (or further items added).
- The main `ArrayList` methods are `add`, `get`, `remove` and `size`.
- `ArrayList` is a parameterized or generic type.