Lecture 4
Chapter 4: Grouping objects

Collections and iterators

Main concepts to be covered

- Collections
  - Objects which can hold a variable number of other objects e.g. Arrays
  - We can add and remove objects
- Loops:
  - for, while, do/while (like in C)
  - also a new kind of for loop in Java 5
- Iterator objects
  - A way to control looping in Java
- Arrays
  - Mostly like in C

The need 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.

A personal notebook

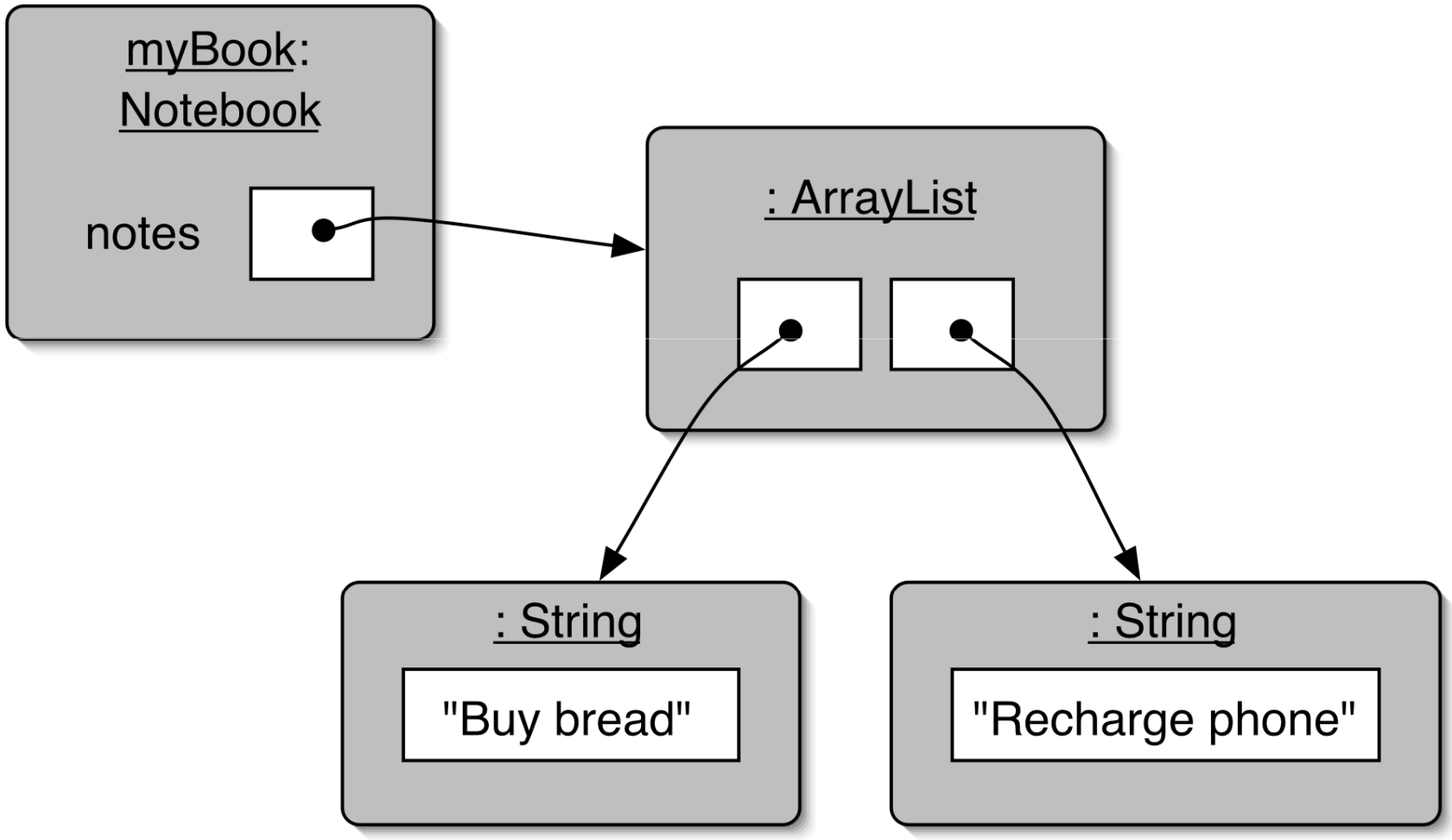
- Notes may be stored.
- Individual notes can be viewed.
- There is no limit to the number of notes.
- It will tell how many notes are stored.
- Explore the *notebook1* project.
- Notebook class is typical Java code:
  - It passes on messages to a library class (ArrayList) that does most of the work
  - We use the notebook class to hide the ArrayList, so we can replace ArrayList later without changing how notebook is used (encapsulation)
  - Notebook also allows us to validate parameters

Class libraries

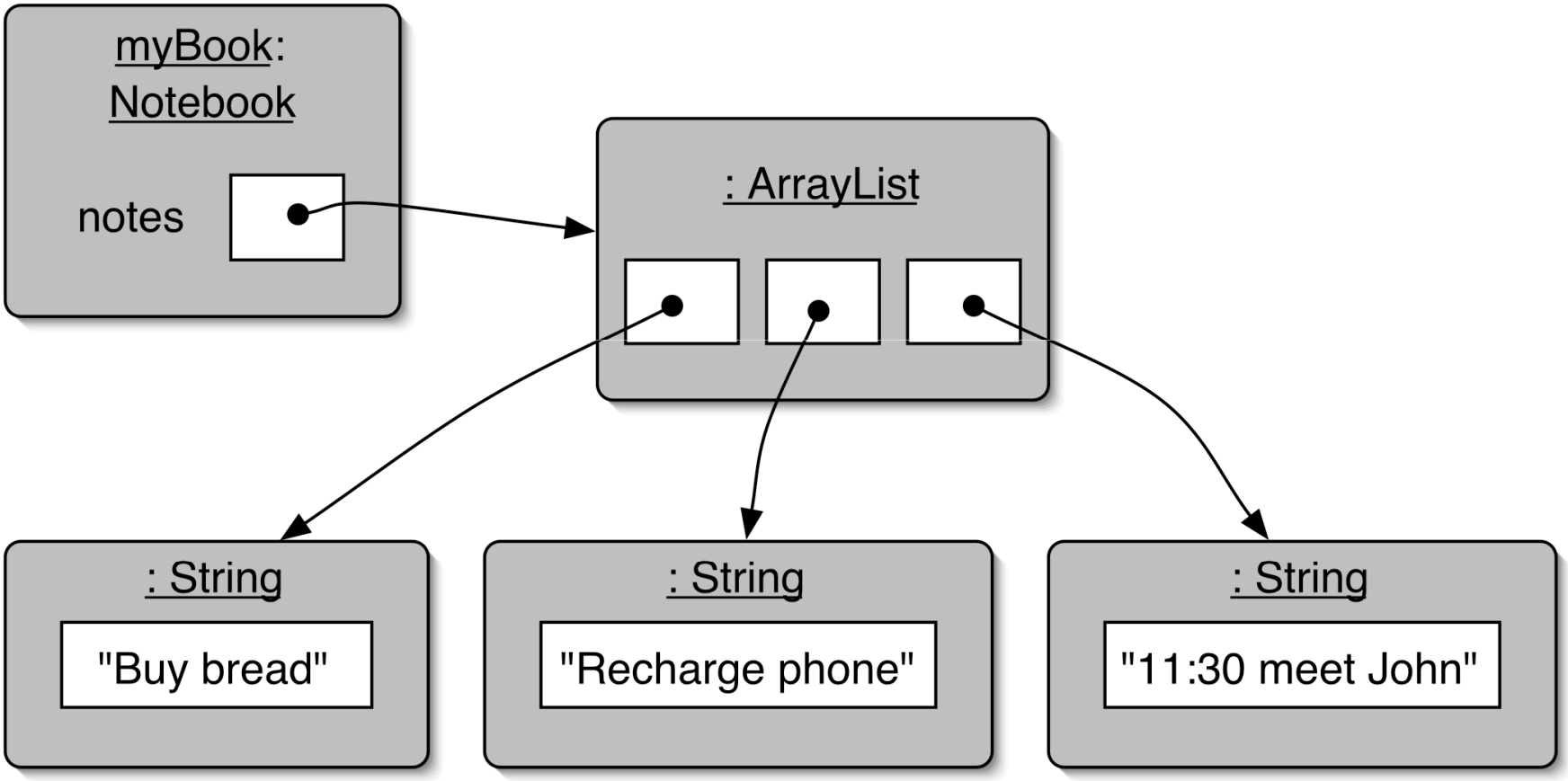
- Collections of useful classes.
- We don’t have to write everything from scratch.
- Java calls its libraries *packages*.
- Grouping objects is a recurring requirement.
  - The `java.util` package contains classes for doing this.
  - Need to import this package to use it.
  - Import must occur before class definition.

```
import java.util.ArrayList;

/** ... */
public class Notebook
{
    // Stores arbitrary number of notes.
    private ArrayList notes;

    /**
     * Perform any initialization
     */
    public Notebook()
    {
        notes = new ArrayList();
    }
    ...
}
```

Object diagram of notebook



Adding a third note



Partial ArrayList API

boolean add(Object) // add to end of collection

boolean add(int index, Object)

Object get(int index)

Object remove(int index)

boolean remove(Object) // remove one instance

int size()

See http://java.sun.com/j2se/1.5.0/docs/api/ for complete Java 1.5 API

Features of ArrayList

- 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?

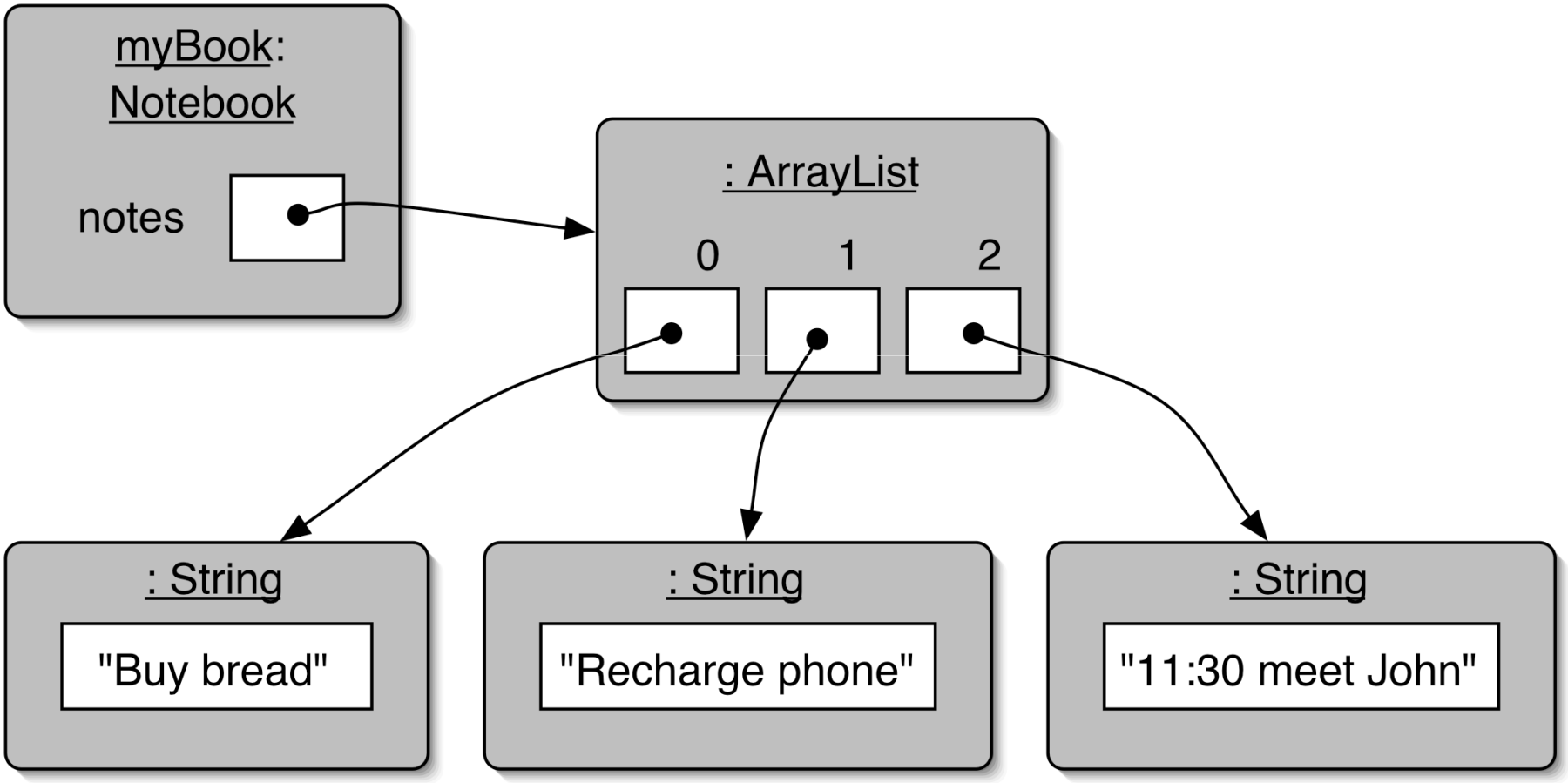
Using the collection

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

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

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

    ...
}
```

Adding a new note

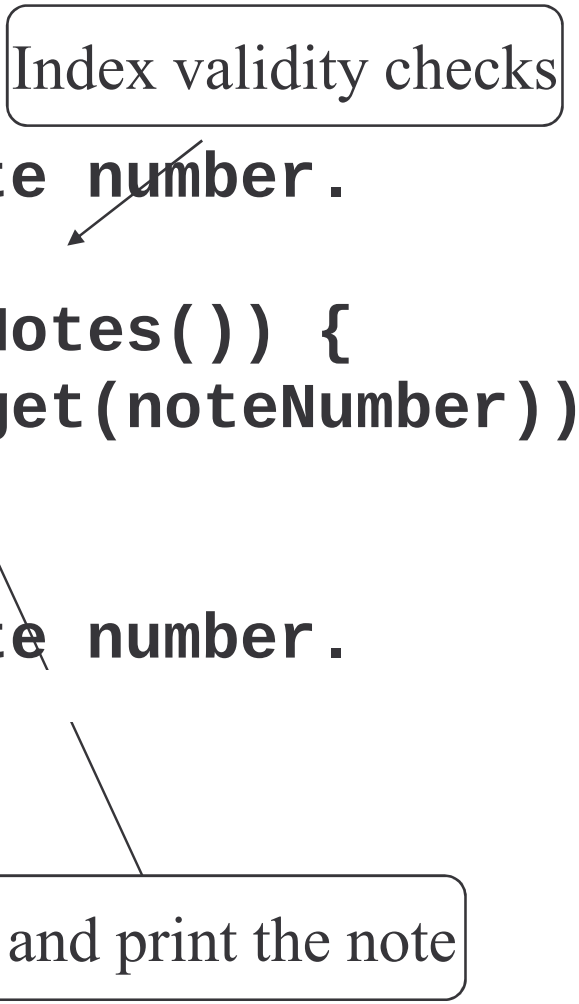
Returning the number of notes (*delegation*).

Index numbering

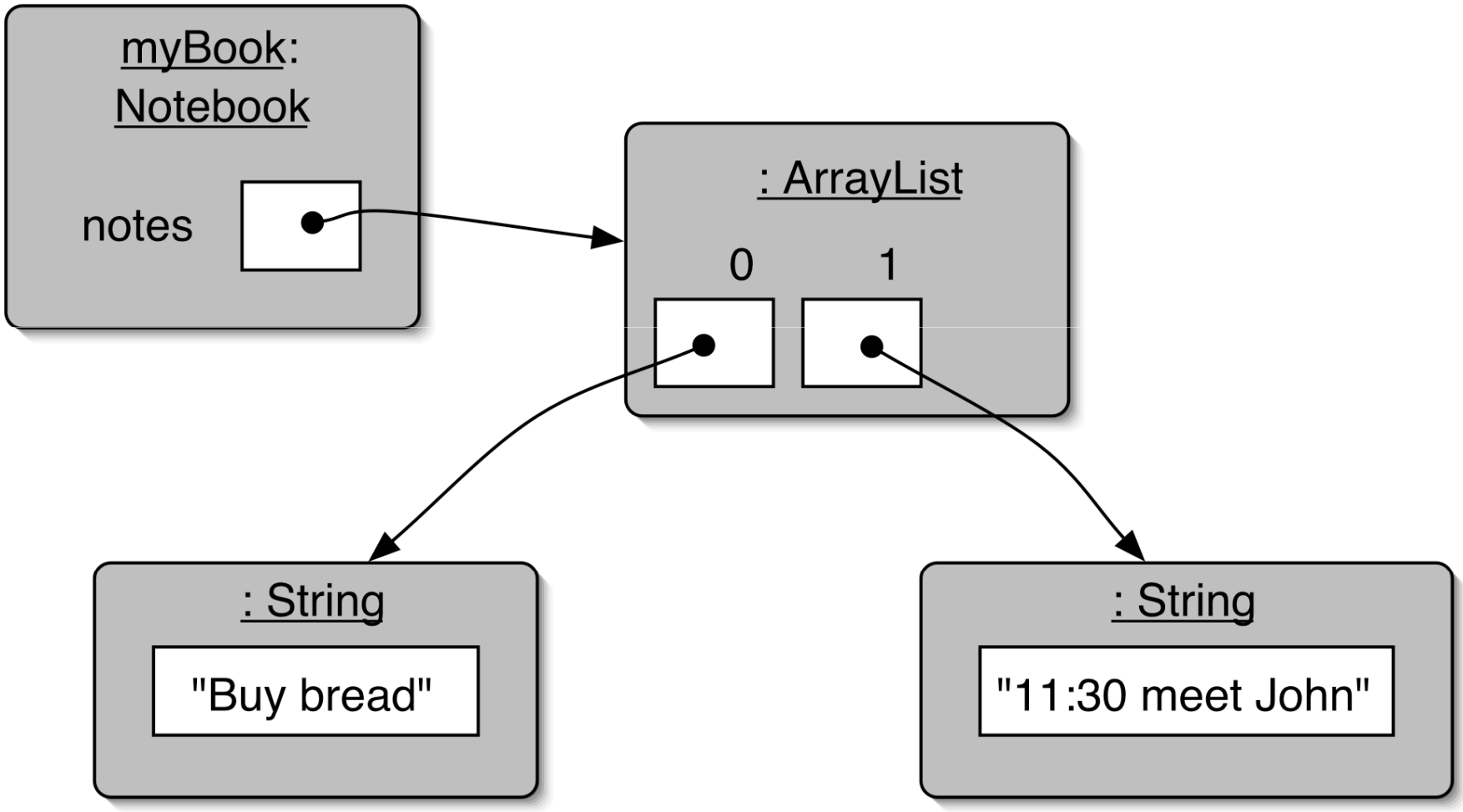


Retrieving an object

```
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.
    }
}
```



Removal may affect numbering



Notebook is typical OO code

- It uses a library class to do most of the work
- It passes on messages (method calls) from the objects which call it to the ArrayList it contains
  - OOP involves a lot of message passing!
  - In contrast, imperative programming is more about step-by-step instructions
  - It does some validity checking on parameters

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.

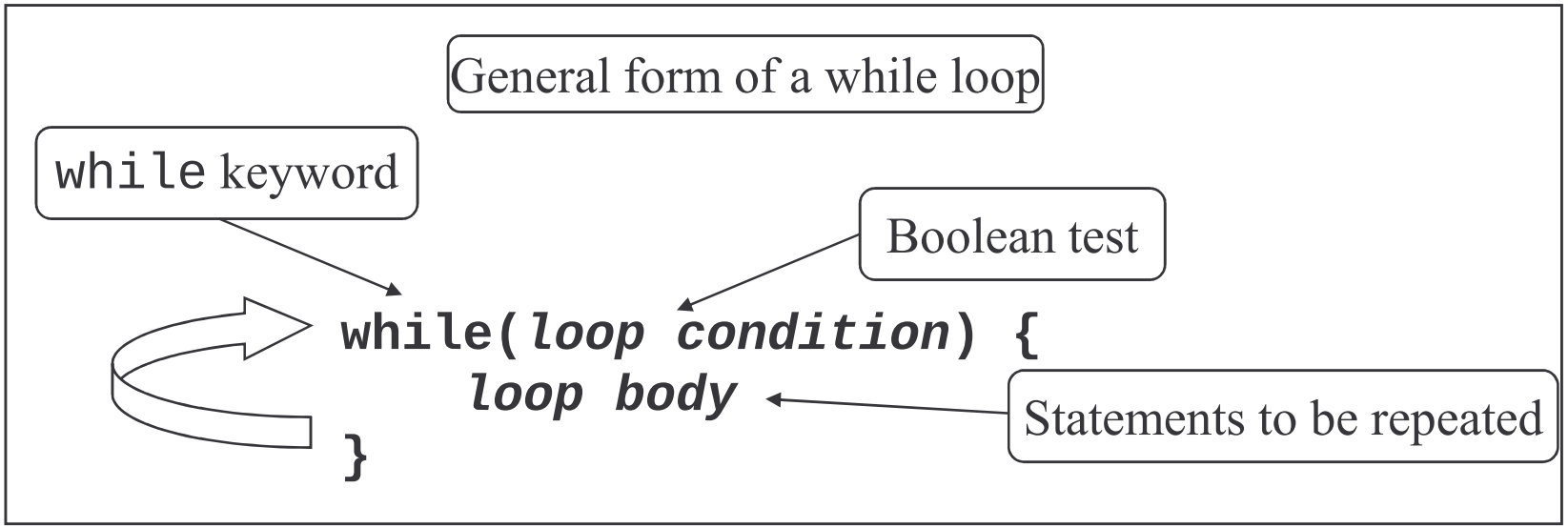
Review

- 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`.

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 three sorts of loop statement:
  - While loop, for loop, do/while loops
  - Same as in C.
  - Java also has iterator objects (later).

While loop pseudo code [c]



Pseudo-code example to print every note

```
while(there is at least one more note to be printed)
{
    show the next note
}
```

A Java example [c]

```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        System.out.println(
            notes.get(index));
        index++;
    }
}
```

Iterator objects

For iterating through collections, the alternative to a loop is an iterator object

```
import java.util.Iterator;
```

Partial Iterator API:

```
Boolean hasNext()
Object next()
void remove() // remove last object
              // returned by next
```

If you call remove() when next() has never been called, you get an error

Iterating over a collection

java.util.Iterator

Returns an `Iterator` object

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

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

Creating iterators

Notice the iterator object was created by calling ArrayList's iterator() method:

```
Iterator it = notes.iterator();
```

Not by using the normal object creation syntax:

```
Iterator it = new Iterator();
```

We get the collection to make the iterator so it can initialise it:

- Iterator knows the indexes of ArrayList contents
- Each collection class can make their own kind of iterator, but they all have the same interface

Iterators and removal

- Asking an ArrayList to insert and remove objects reorders it
  - This confuses any existing iterators
  - Using an existing iterator causes a ConcurrentModificationException – an error
  - Making a new iterator is ok
- Better to use Iterator.remove() instead of ArrayList.remove() when using iterators
  - This way all iterators are informed of change
- This is messy, but it's the price of having 2 ways of iterating

The *auction* project

- The *auction* project provides further illustration of collections and iteration.
- Two further points to follow up:
  - The `null` value.
  - Casting from one type to another

Null

- An object variable which has not been initialised has the value *null*

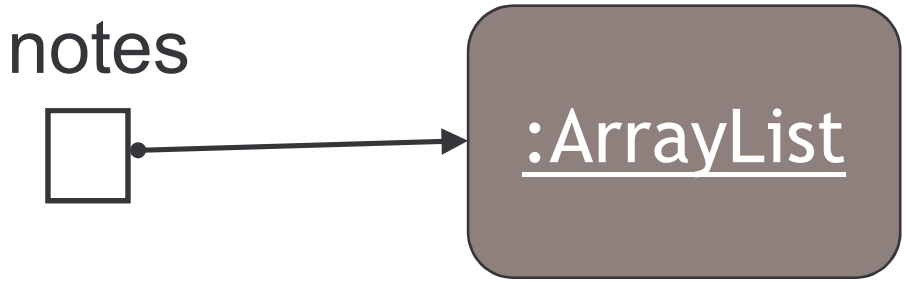
```
myType myObject;
System.out.println(myObject);
myObject.myMethod(); // error
```

- We can check to see whether it has been initialised with:

```
if (myObject == null) …
```

Null

```
public class Notebook {
    private ArrayList notes;

    public Notebook() {
      System.out.println(notes); // null
      notes = new ArrayList();
      System.out.println(notes); // not null
    }
    ...
```



Retrieving objects from collections

- *Untyped* collections don't remember what type an object has (their methods return type Object)
- When we get an object from an untyped collection, we usually have to *cast* it back to its type before we can use it, e.g.:

```
ArrayList notes = new ArrayList();
String myString = "hello";
Notes.add(myString);
String newString = (String) notes.get(0);
```

- This line in showNote() is ok:

```
System.out.println(notes.get(noteNumber));
```

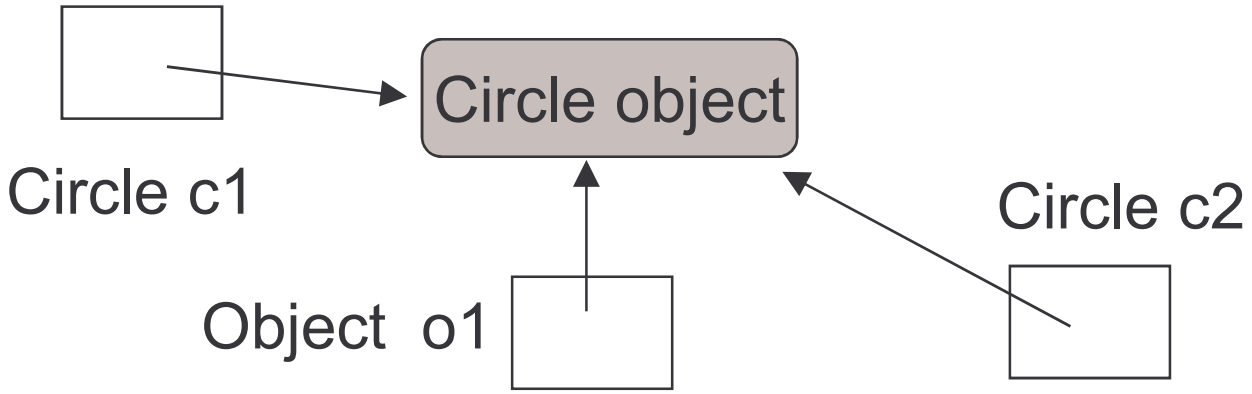
  Because println accepts type Object:

```
void println(Object);
```

Casting objects

- A parameter, variable or field of type Object can be assigned an object of any class
  - This is how untyped Collections work
  - We can later cast the object back to its real type
  - We **cannot** cast it to something it is not e.g. Triangle

```
Circle c1 = new Circle();
Object o1 = c1;              //Java casts c1 to type Object
Circle c2 = (Circle) o1; // force object back to type Circle
```



Casting primitives [C]

- This works because there is a hierarchy of types (more in a later lecture)
- Java will implicitly cast primitive types when this does not result in a loss of information

```
int i = 45;
double d = i; // Java automatically casts i to
              // double
```

- We can explicitly force a cast even when information may be lost

```
double d = 45.5;
int i = (int) d; // i becomes 45
```

Typed collections

- A rule of thumb is to have only 1 type of object in a collection
  - If you must mix types, you can give them all the same type using an interface (Ch. 10)
- New in Java 5
  - We can make typed collections
  - In fact Java 5 or later issues a warning if we don't

```
Note:...\projects\chapter04\notebook1\Not
  ebook.java uses unchecked or unsafe
  operations.
```

Typed collections

Instead of:

```
Private ArrayList notes;
```

Write:

```
Private ArrayList<String> notes;
```

Read it as “Private ArrayList of Strings”

Now notes:

- only accepts Strings
- returns Strings (no need to cast)

Typed Iterators

We can type iterators too:

Instead of:

```
Iterator it = myList.iterator();
```

Write:

```
Iterator<String> it = myList.iterator();
```

Now we don’t need to cast what it returns:

```
String newString = it.next();
```

Review

2 ways to iterate over collections:

- A Java while loop allows the repetition to be controlled by a boolean expression.
- Collection classes have special `Iterator` objects for iteration over the whole collection.
  - If using iterators, use them to remove objects

For ArrayList the two are comparable.

For other collections, iterators are preferable (more in another lecture).

Review

- Preferable to only have 1 type/class in a collection
- Use typed collections and typed iterators with Java 5 or later

Fixed-size collections

- Sometimes the maximum collection size can be pre-determined.
- Programming languages usually offer a special fixed-size collection type: an *array*.
- Less flexible than ArrayList, but more efficient.
- Java arrays can store objects or primitive-type values.
- Arrays use a special syntax.
- Java arrays of primitive types are just like C
- Java arrays of objects initially contain null

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.

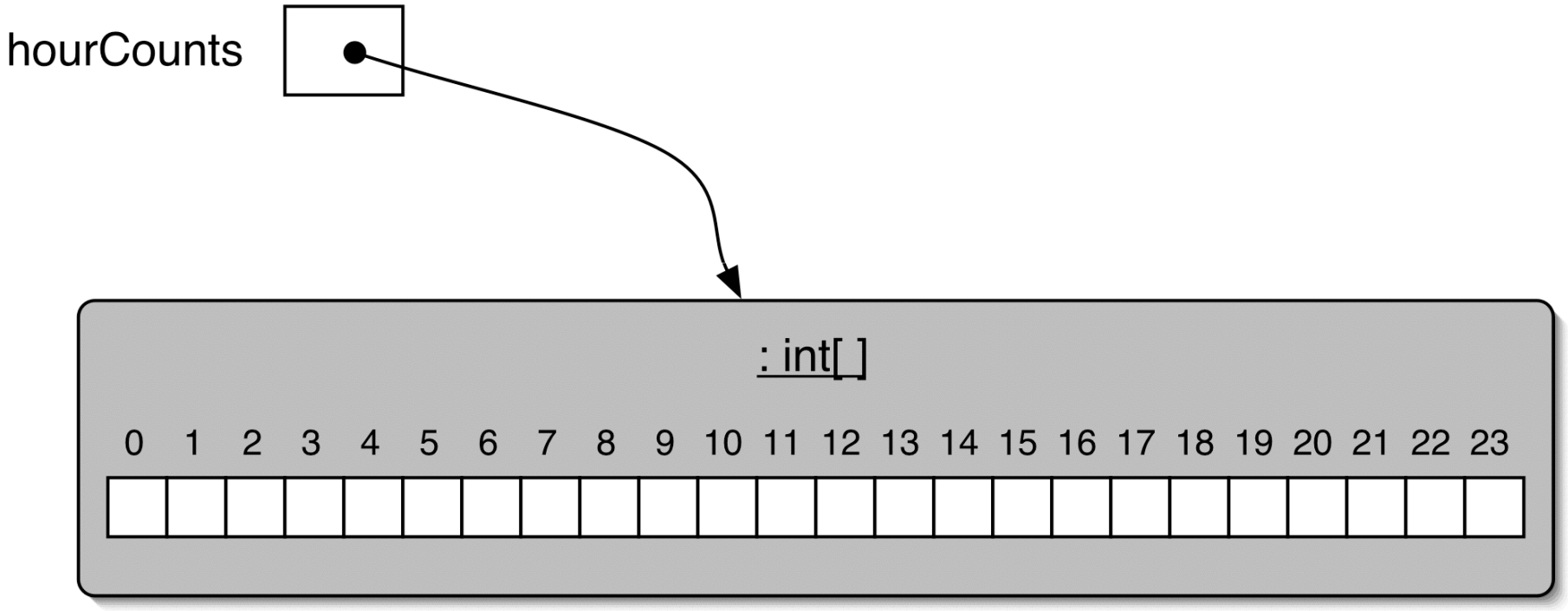
Creating an array object

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

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

Array variable declaration

Array object creation

Object diagram: The `hourCounts` array



Using an array [c]

- 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]++;`**

The for loop [c]

- Similar to a while loop.
- Often used to iterate a fixed number of times.
- Often used to iterate over an array.
- Just like C's for loop.

For loop pseudo-code [c]

General form of a for loop

```
for(initialization; condition; post-body action){
    statements to be repeated
}
```

Equivalent in while-loop form

```
initialization;
while(condition) {
    statements to be repeated
    post-body action
}
```

A Java example [c]

for loop version

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

while loop version

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

Enhanced for loops

- New in Java 5
- No initialising, testing or incrementing
  - Helps avoid some common errors
- No loop variable
- Can be used with all Collections (including Arrays)
- Syntax:

```
for (Type element : collection)
   // do something with element
```

Enhanced for loops

```
public int totalQuantity(Product[] products)
{
    int total = 0;
    for(Product p : products) {
        total += p.getQuantity();
    }
    return total;
}
```

- Read “for” as “for each” and “:” as “in”
  - The code above says “For each Product p in products”
- Sometimes you need a loop variable
  - In this case, use the old C style for loop

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 often used to iterate over arrays.

Next Lecture

More classes from the Java class library:

- Random numbers
- Hash maps
- Sets
- String tokenization