

Objects First With Java
A Practical Introduction Using BlueJ

Grouping objects

Collections and iterators

Concepts covered so far

- Abstraction
- Modularization
- Classes define types
- Class and object diagrams
- Object references
- Primitive types
- Object types
- Object creation
- Overloading
- Internal/external method call

Main concepts to be covered

- Collections
- Loops
- Iterators
- Arrays

The requirement to group objects

- Many applications involve objects grouped into collections:
 - Personal organizers (objects: notes).
 - Library catalogs (objects: files).
 - Student-record system (objects: records).
- The number of items to be stored varies.
 - Items added (new appointments or books).
 - Items deleted (students leave university or appointments are cancelled).

Possibilities so far?

- Define one class:
 - A field for each note / book / student.
 - Drawback: Possible number of entries is fixed, not variable!
- Again, define one class:
 - Create objects (instances of the class) for new notes / books / students.
 - Drawback: Objects are not grouped at all, just living individually!
- Hence: need for more flexibility!

Example: A personal notebook

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

Class libraries, library classes

- Libraries are collections of a large number of predefined useful classes.
- Java has many of such libraries.
- We don't have to write everything from scratch 😊.
- Java calls its libraries *packages*.
- No differences in using such library classes.
- Grouping objects is a recurring requirement.
 - The `java.util` package contains classes for doing this.
 - One example is the `ArrayList` class used in the following.
 - `ArrayList` is a *collection class*.
 - Collections can store an arbitrary number of elements, each element being another object.

```
import java.util.ArrayList;
```

Import: from a library!
(to be placed before class defs.)

```
/**
```

```
 * ...
```

```
 */
```

```
public class Notebook
```

```
{
```

```
    // Storage for an arbitrary number of notes.
```

```
    private ArrayList notes; Once imported, ArrayList can be  
used as usual.
```

```
    /**
```

```
     * Perform any initialization required for the  
     * notebook.
```

```
     */
```

```
    public Notebook()
```

```
    {
```

```
        notes = new ArrayList(); Constructor: initialize!
```

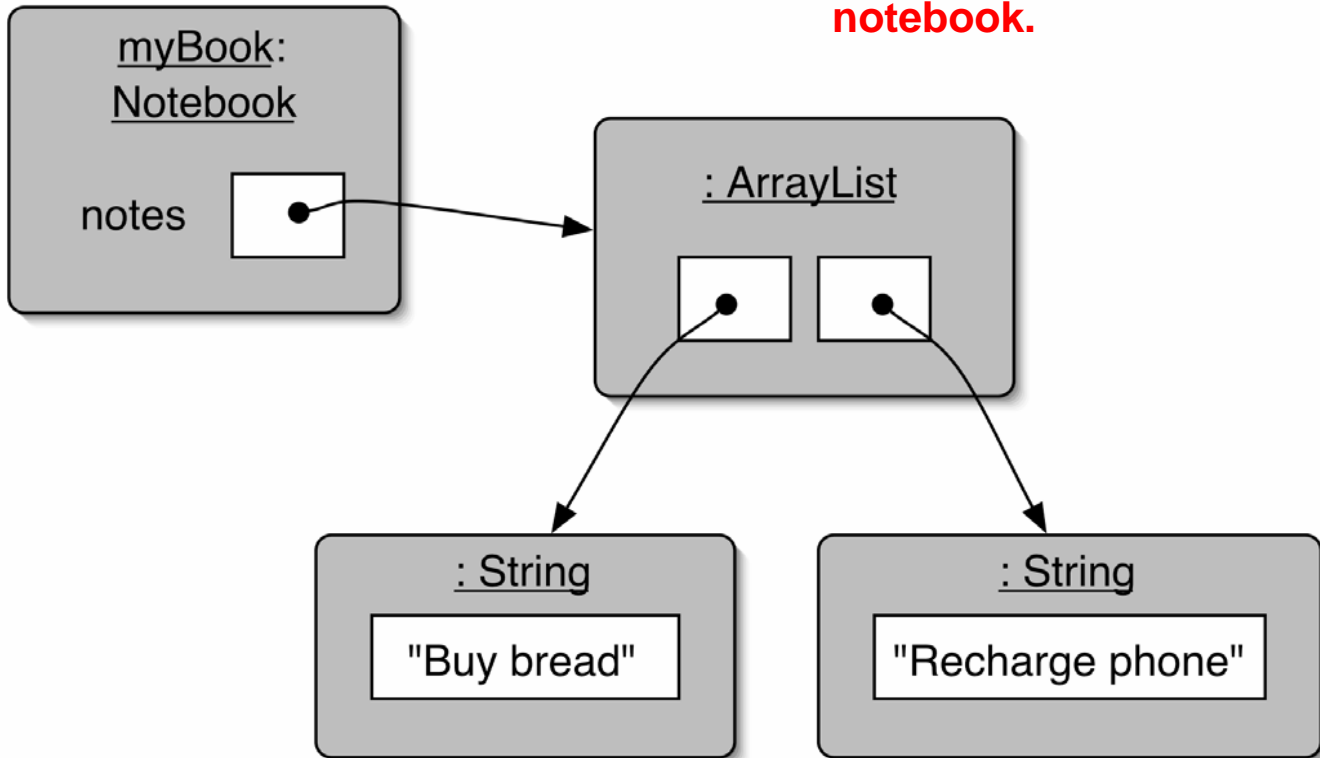
```
    }
```

```
    ...
```

```
}
```

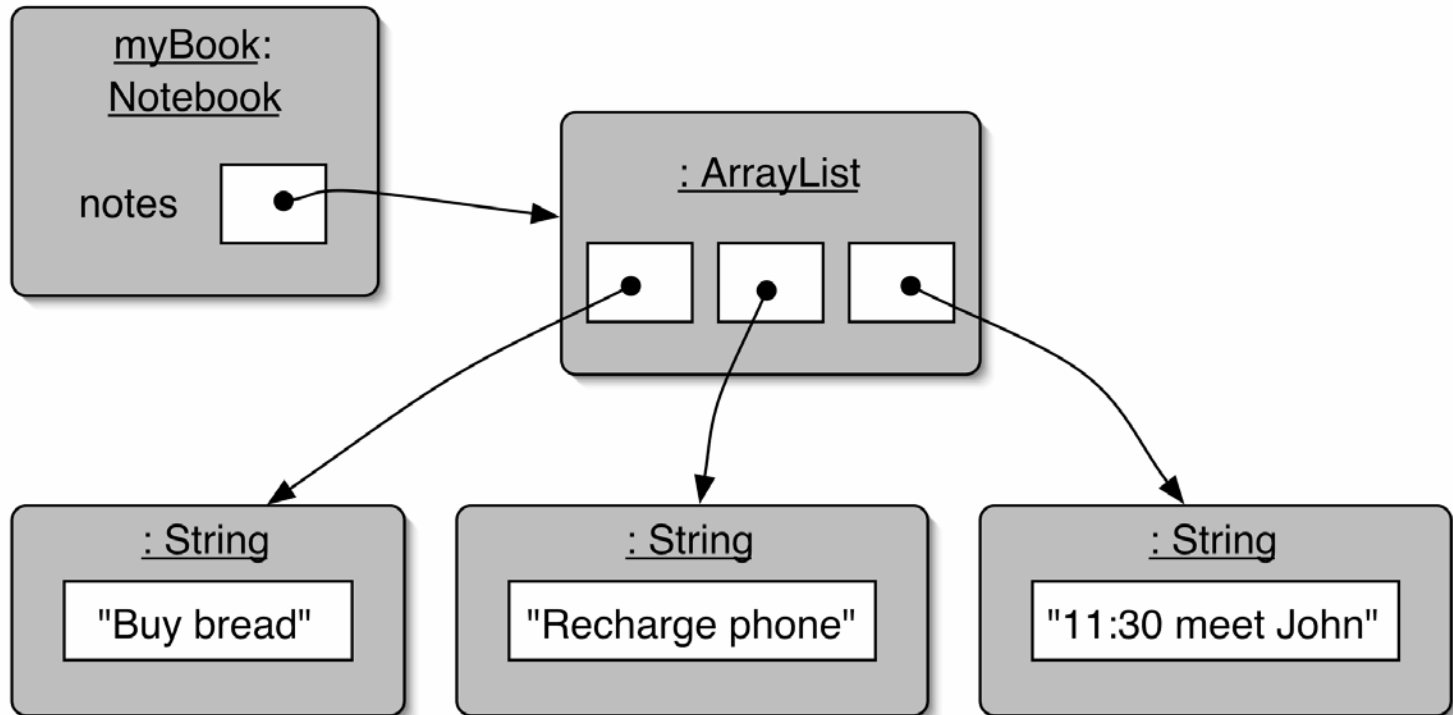

Object structures with collections

At present, two notes in the notebook.



Adding a third note

Then, a third one is added.



Features of the collection

- It increases its capacity as necessary (i.e. no memory problems).
- It keeps a private count (`size()` accessor method).
- It keeps the objects in the order they have been inserted.
- Details of how all this is done are hidden.
 - Does that matter? Must we know details? Does not knowing how prevent us from using it?
 - Advantage: Someone else did work, we just need to know how to use `ArrayList`.

Using the collection

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

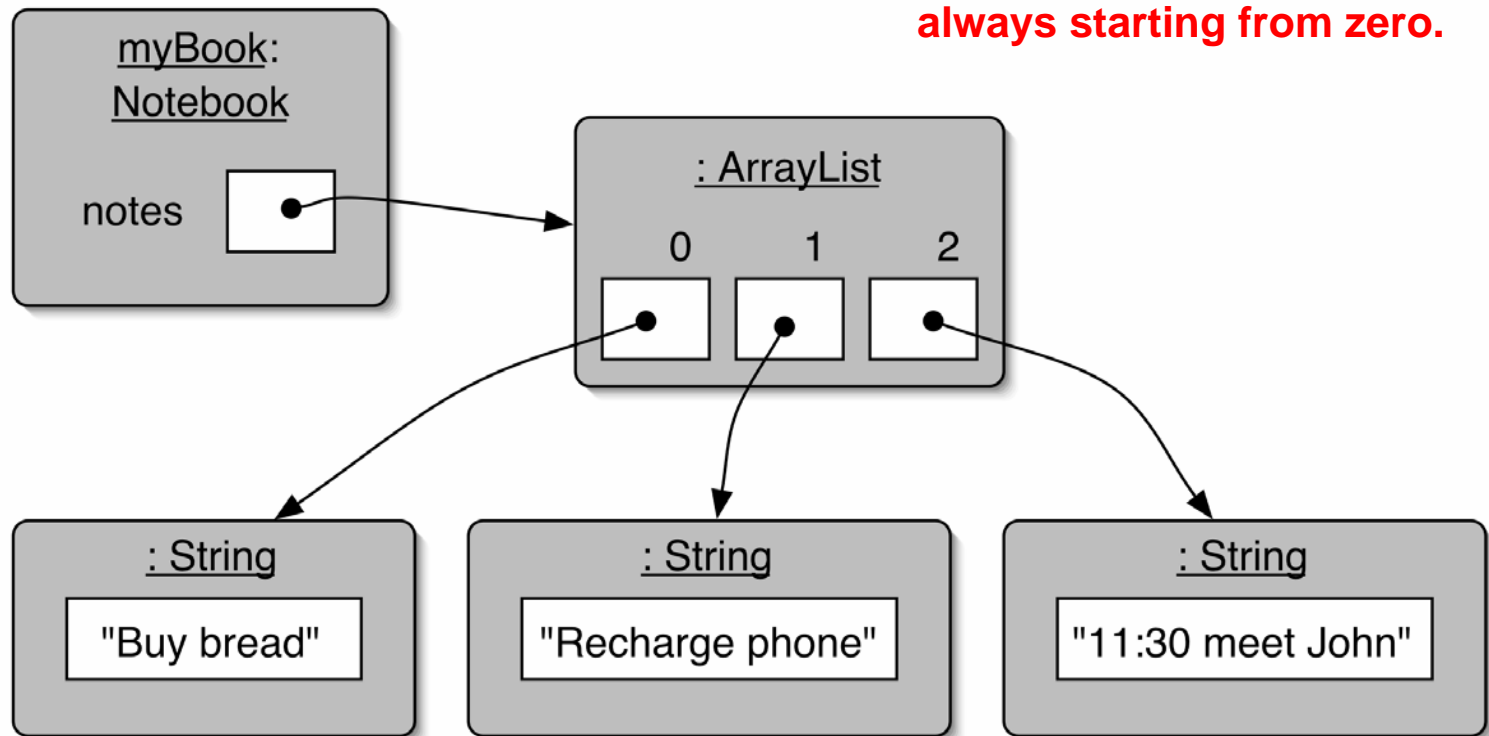
    public void storeNote(String note)
    {
        notes.add(note); ← Adding a new note
    }

    public int numberOfNotes()
    {
        return notes.size(); ← Returning the number of notes
                               (delegation to ArrayList).
    }

    ...
}
```

Index numbering

Implicit numbering (index),
always starting from zero.



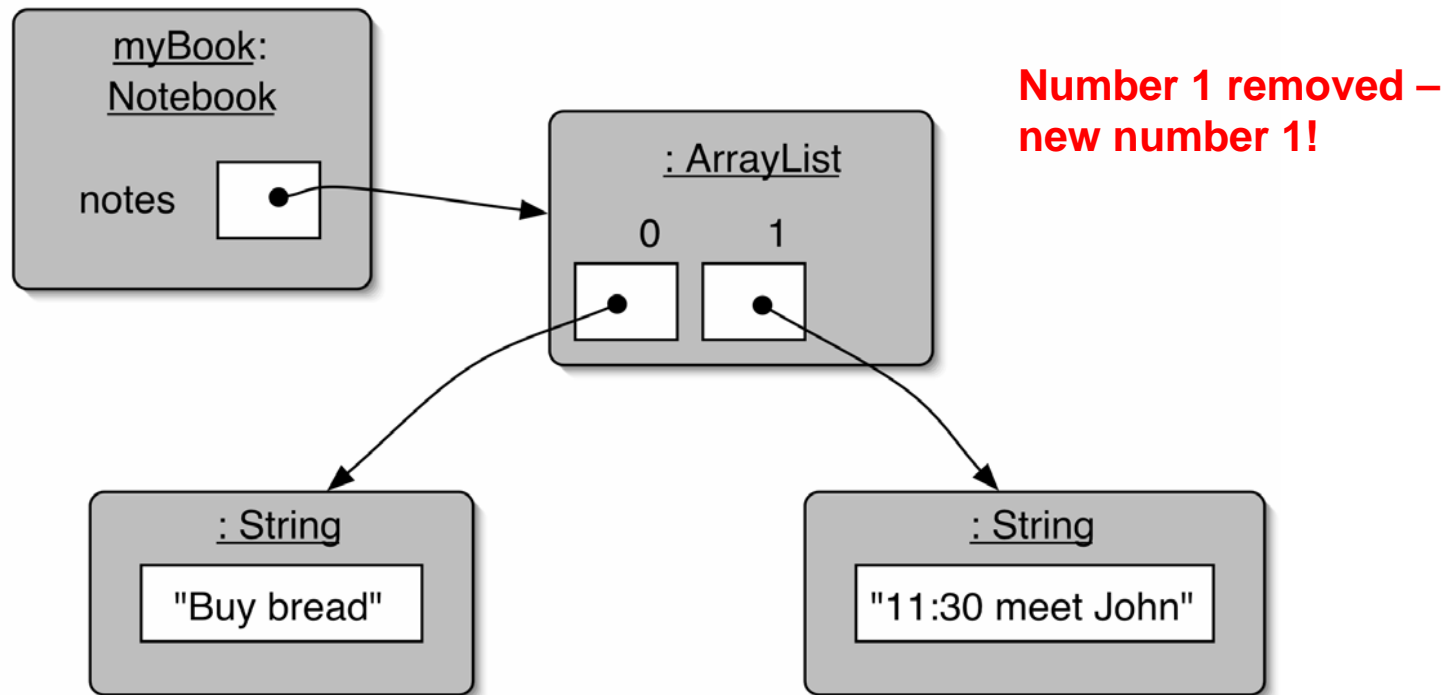
Retrieving an object

```
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.
    }
}
```

Index validity checks

Retrieve and print the note

Removal may affect numbering



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`.

Documentation of Java packages

- Java API Documentation
 - <http://download-llnw.oracle.com/javase/6/docs/api/index.html>
- List of packages
 - List of interfaces and classes
 - Description of each class
 - Field, Constructors, and Methods summary
- Example: `java.util` -> `ArrayList`

Iteration

- *Iteration* means doing something several times, where the number of times may vary due to circumstances.
- 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:
 - We will focus on its *while loop*.

A Java example

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

**Increment by one: equivalent
to
index += 1;
index = index+1;**

Iterating over a collection: an alternative way

```
import 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  
}
```

Attention: Iterator, iterator!

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

Iterator offers 2 methods: next, hasNext

The *auction* project

- The *auction* project provides further illustration of collections and iteration.
- Two further points to follow up:
 - The `null` value – reference to “no object”.
 - **Casting**. Used to store the result of `get` into a variable (of a specific type):
 - `String message = (String) notes.get(0);`

Review

- Loop statements allow a block of statements to be repeated.
- A Java while loop allows the repetition to be controlled by a boolean expression.
- Collection classes have special `Iterator` objects that simplify iteration over the whole collection.

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 (collections can store objects only).
- Arrays use a special syntax (different from method concept for historical reasons).
- Access to arrays is typically more efficient.

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.

The *weblog-analyzer* project

- Log files contain line records:
 - year month day hour minute
 - 2006 11 09 08 10
- Classes:
 - `LogAnalyzer`, `LogReader`, `LogEntry` and `LoglineTokenizer`

Creating an array object

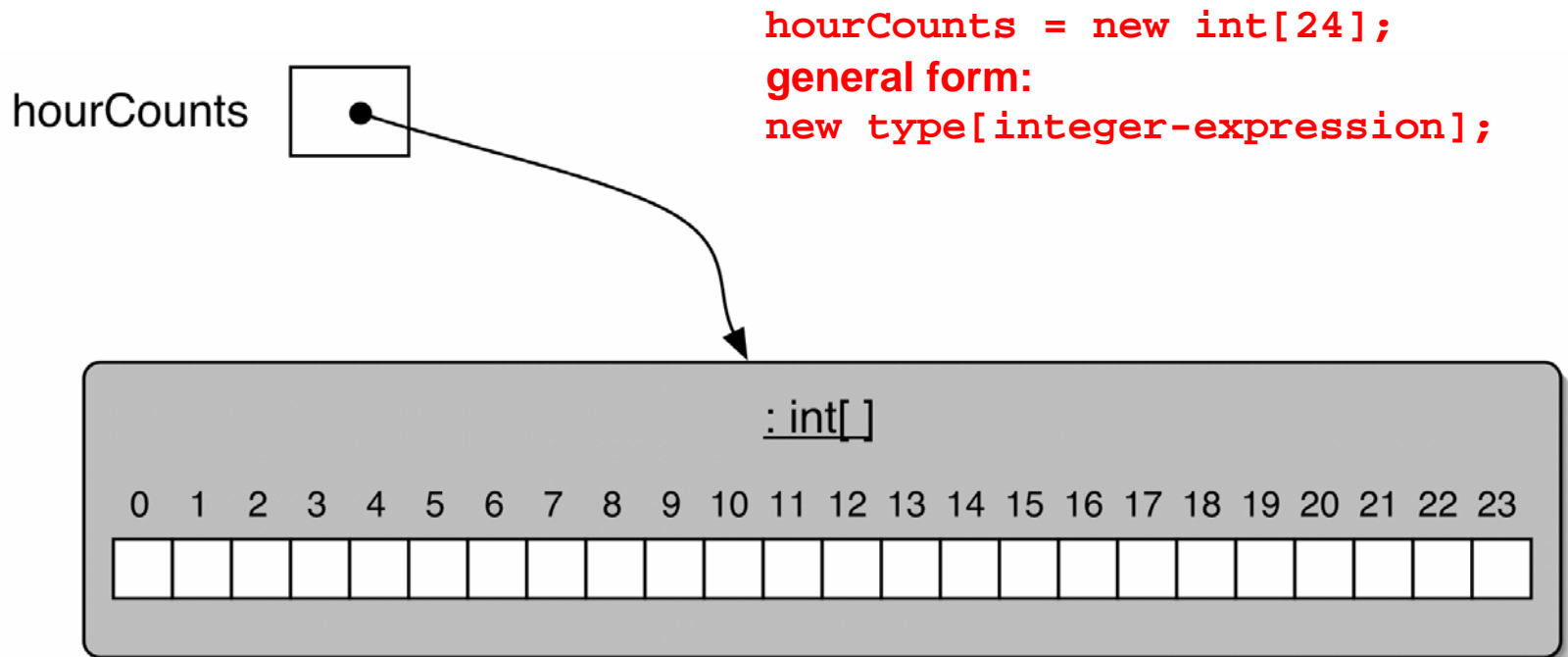
```
public class LogAnalyzer
{
    private int[] hourCounts; ←
    private LogfileReader reader;

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

**Array variable
declaration – no
creation of an array
yet!**

Array object creation

The hourCounts array



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]++;`
- Array indices always start at zero.

A Java for loop example using an array

for loop version

for loop MATLAB?

for loop C?

```
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++;  
}
```

Review

- Arrays are appropriate where a fixed-size collection is required.
- Arrays use special syntax.
- Classical for loop usage to iterate over arrays.

Concepts covered today

- collection
- array
- index
- iterator
- while loop
- for loop
- import statement
- library
- package
- cast