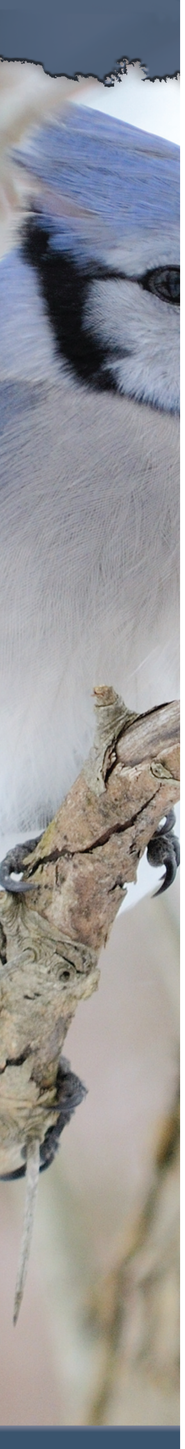
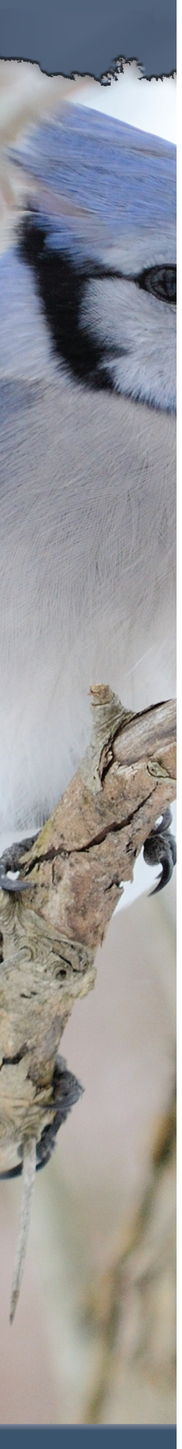


KING'S College LONDON

Programming Practice and Applications

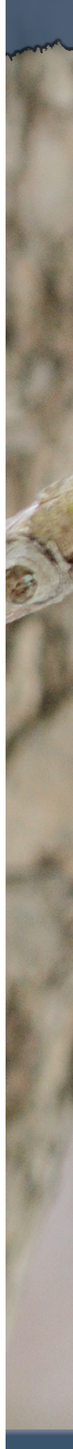
Grouping objects

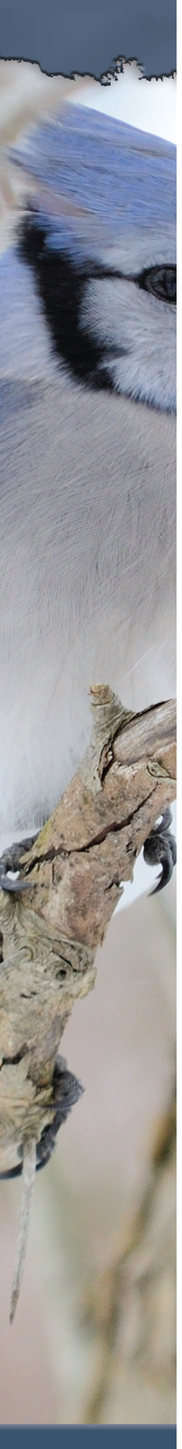
Michael Kölling



Main concepts to be covered

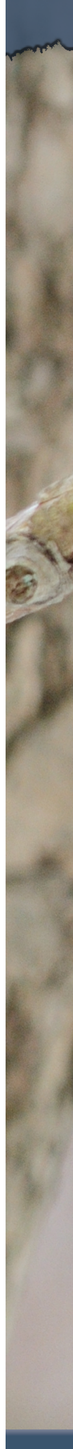
- Collections
  (especially **`ArrayList`**)
- Builds on the *abstraction* theme from the last chapter.

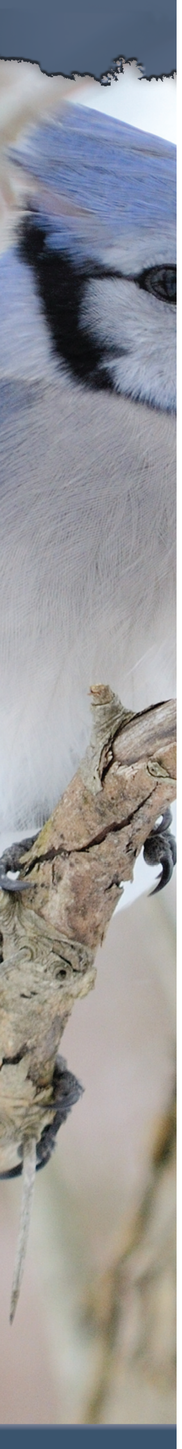




The requirement to group objects

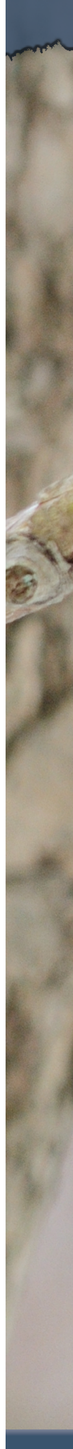
- Many applications involve collections of objects:
  - Personal organisers.
  - Library catalogs.
  - Student-record systems.
- The number of items to be stored varies.
  - Items added.
  - Items deleted.

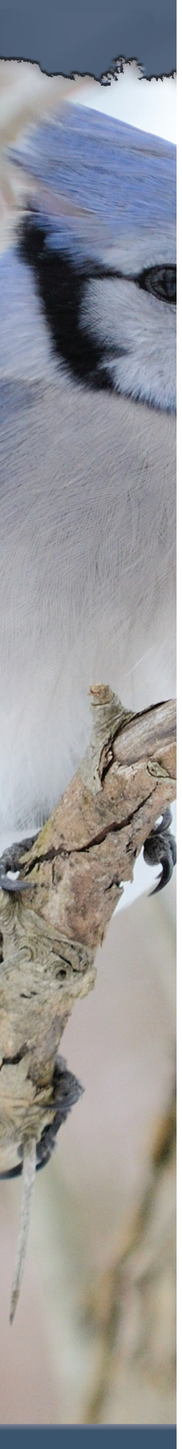




An organiser for music files

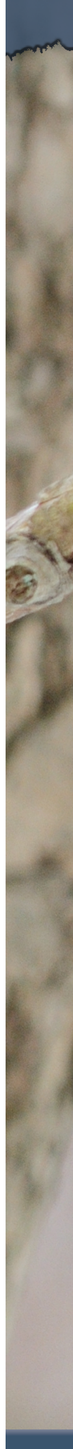
- Single-track files may be added.
- There is no pre-defined limit to the number of files/tracks.
- It will tell how many file names are stored in the collection.
- It will list individual file names.
- Explore the *music-organizer-v1* project.

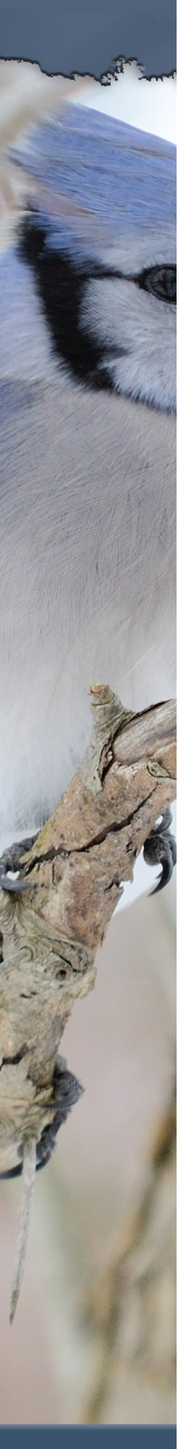




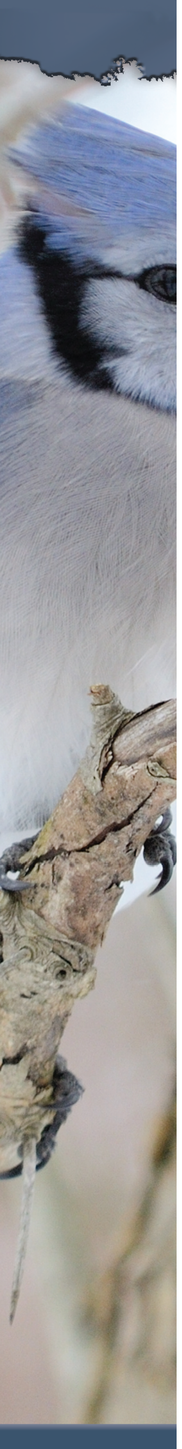
Class libraries

- A library 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 multiple classes for doing this.





```
import java.util.ArrayList;

/**
 * ...
 */
public class MusicOrganizer
{
    // Storage for an arbitrary number of file names.
    private ArrayList<String> files;

    /**
     * Perform any initialization required for the
     * organizer.
     */
    public MusicOrganizer()
    {
        files = new ArrayList<>();
    }

    ...
}
```

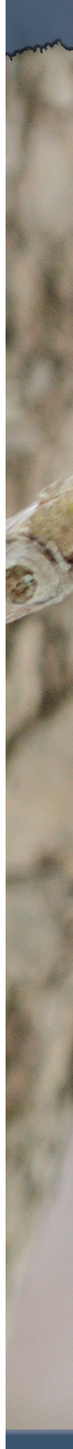


Collections

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

```
private ArrayList<String> files;
```

- We say, “ArrayList of String”.



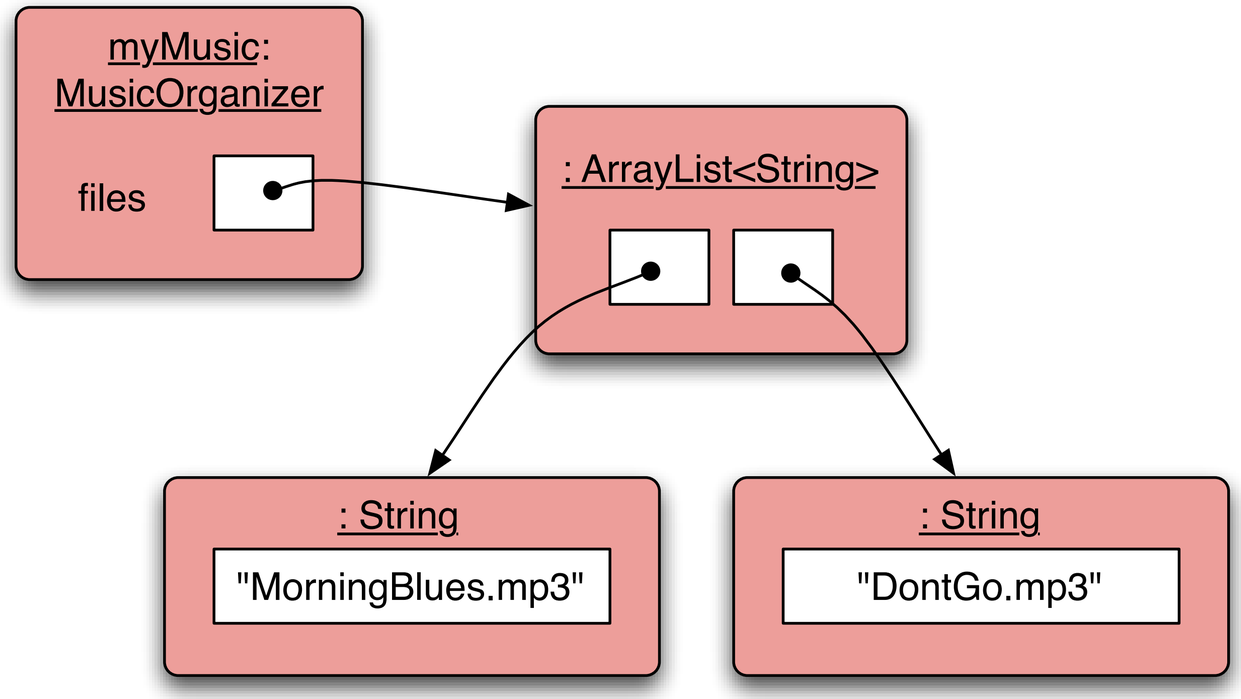
Generic classes

- Collections are known as *parameterised* 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.

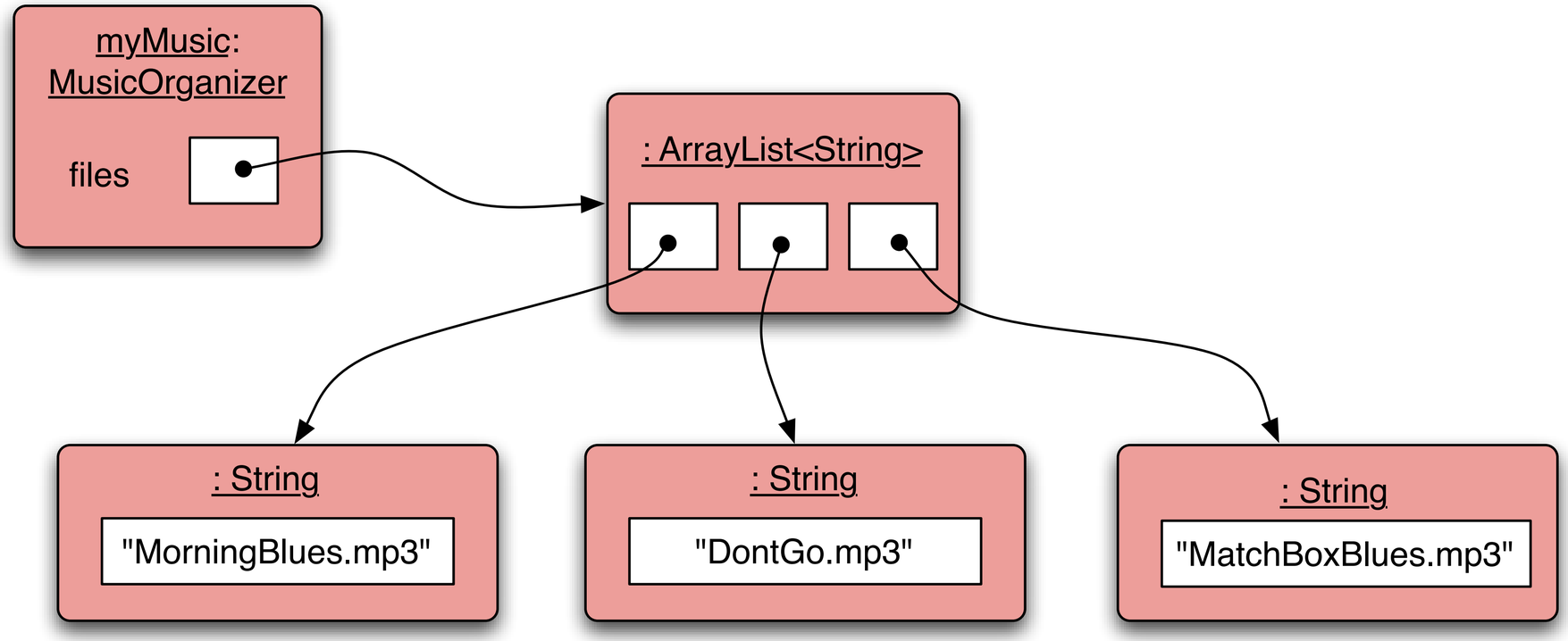
Creating an ArrayList object

- In versions of Java prior to version 7:
  - `files = new ArrayList<String>();`
- Java 7 introduced 'diamond notation'
  - `files = new ArrayList<>();`
- The type parameter can be inferred from the variable being assigned to.
  - A convenience we will use.

Object structures with collections



Adding a third file



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?

Generic classes

- We can use **`ArrayList`** with **any** class type:
  `ArrayList<TicketMachine>`
  `ArrayList<ClockDisplay>`
  `ArrayList<Track>`
  `ArrayList<Person>`
- Each will store multiple objects of the specific type.

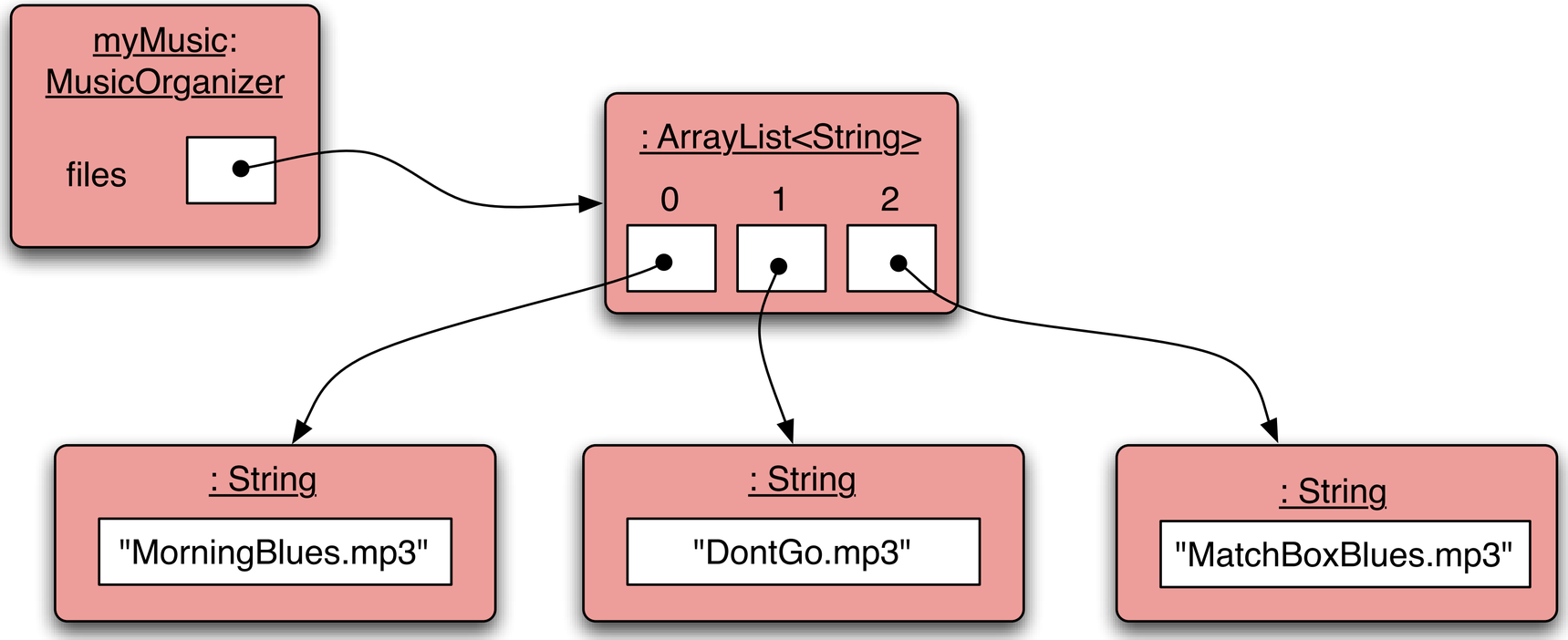
Using the collection

```
public class MusicOrganizer
{
    private ArrayList<String> files;

    ...

    public void addFile(String filename)
    {
        files.add(filename);
    }

    public int getNumberOfFiles()
    {
        return files.size();
    }

    ...
}
```

Adding a new file

Returning the number of files (*delegation*)

Index numbering



Retrieving from the collection

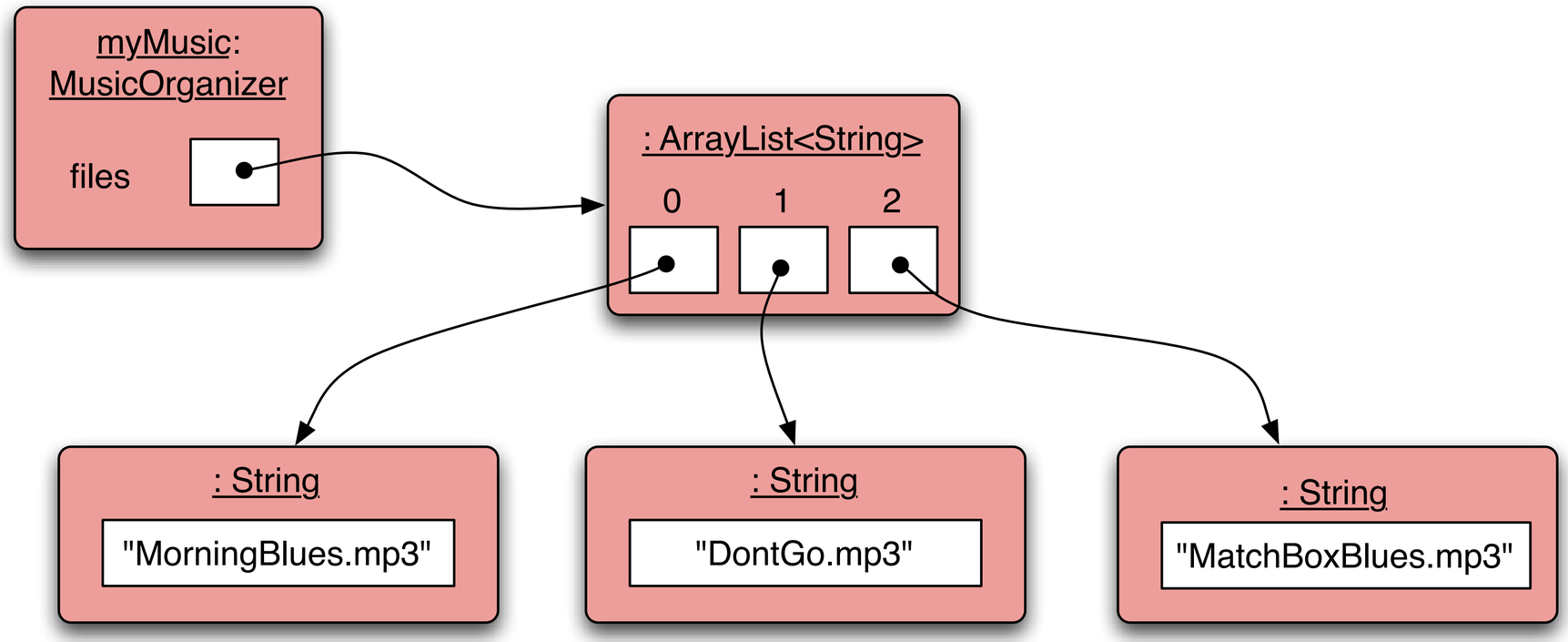
```
public void listFile(int index)
{
    if(index >= 0 && index < files.size()) {
        String filename = files.get(index);
        System.out.println(filename);
    }
    else {
        // This is not a valid index.
    }
}
```

Index validity checks

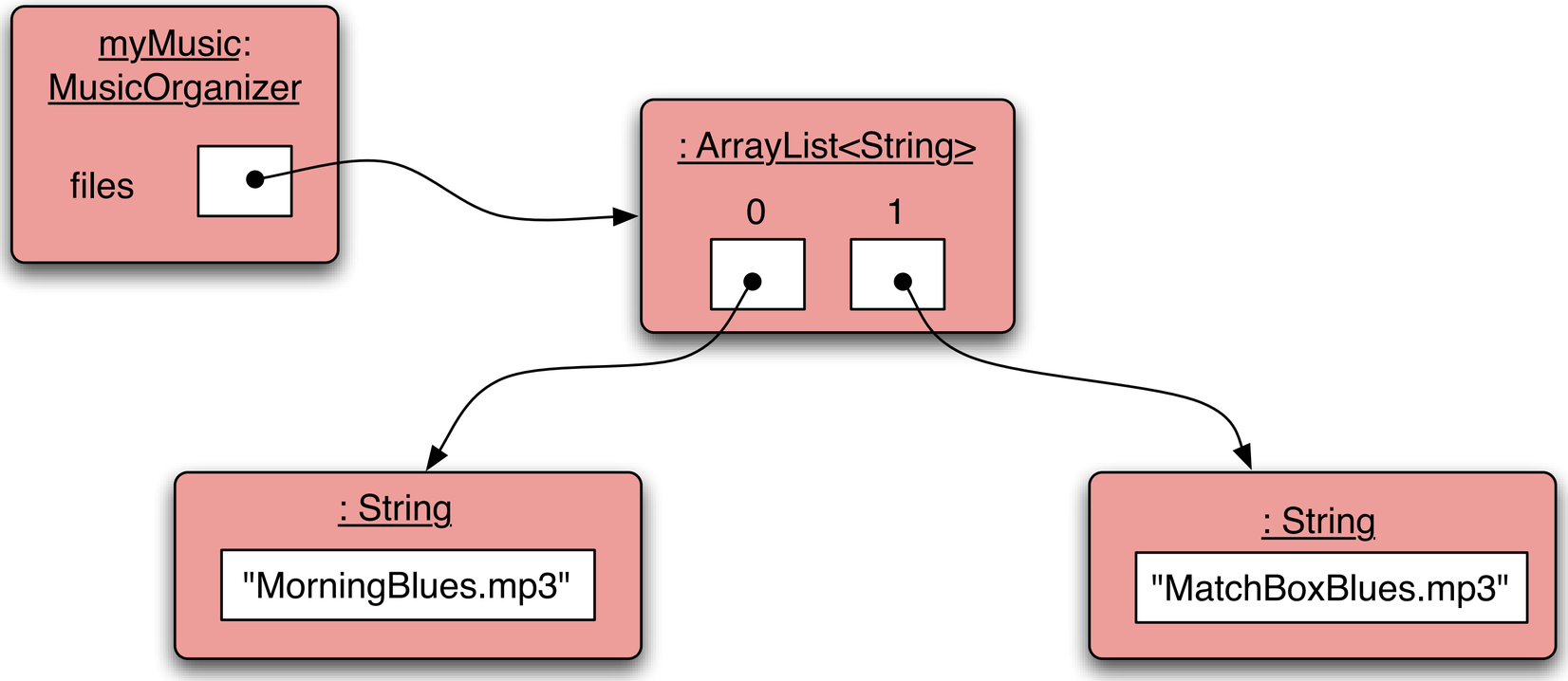
Retrieve and print the file name

Needed? (Error message?)

Removal may affect numbering

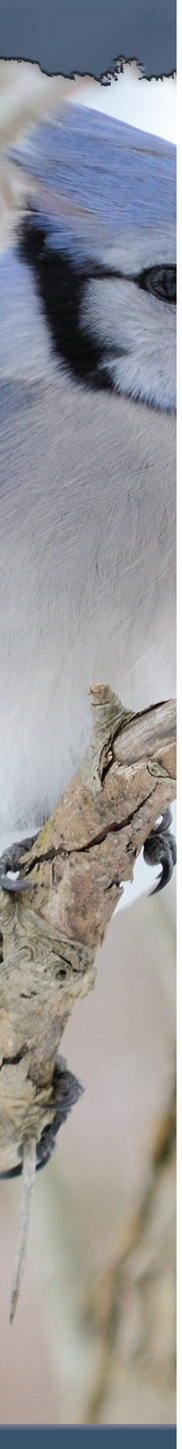


Removal may affect numbering



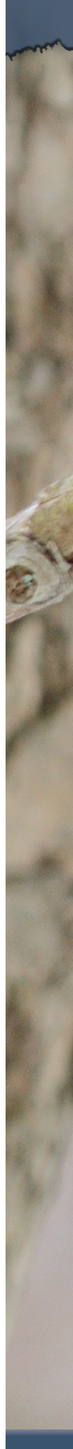
The general utility of indices

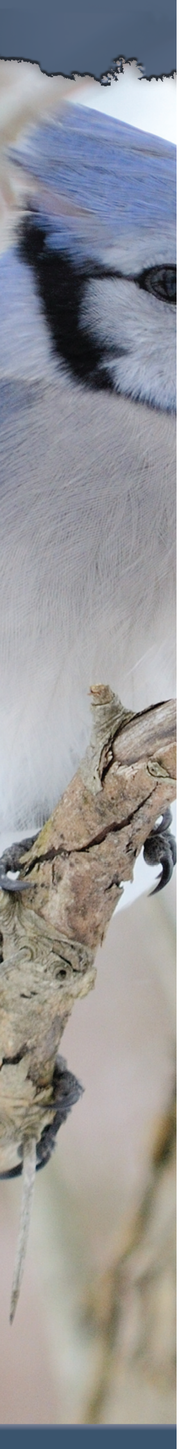
- Using integers to index collections has a general utility:
  - 'next' is: `index + 1`
  - 'previous' is: `index - 1`
  - 'last' is: `list.size() - 1`
  - 'the first three' is: the items at indices `0`, `1`, `2`
- We could also think about accessing items in sequence: `0, 1, 2, …`



Review

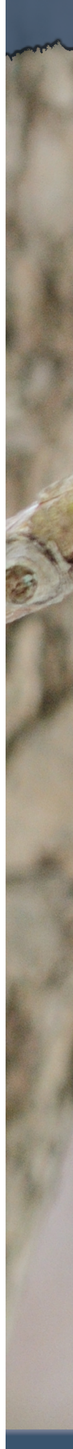
- Collections allow an arbitrary number of objects to be stored.
- Class libraries usually contain tried-and-tested collection classes.
- Java's class libraries are called *packages*.
- We have used the **`ArrayList`** class from the **`java.util`** package.

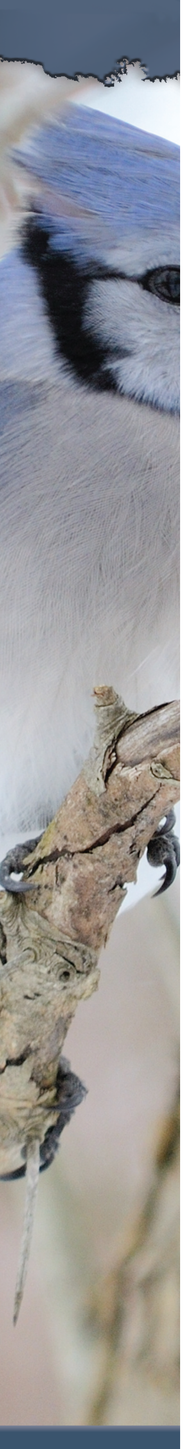




Review

- 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.

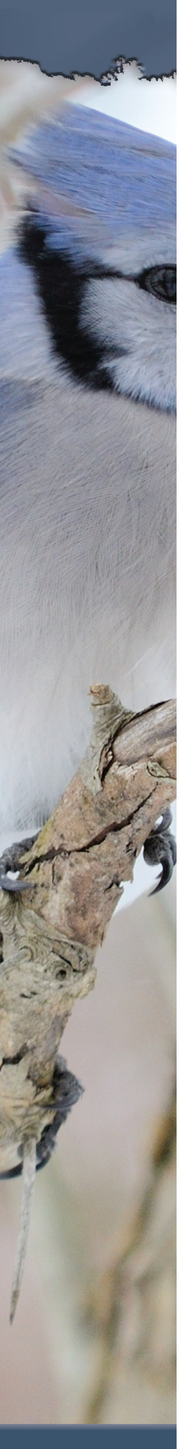




KING'S College LONDON

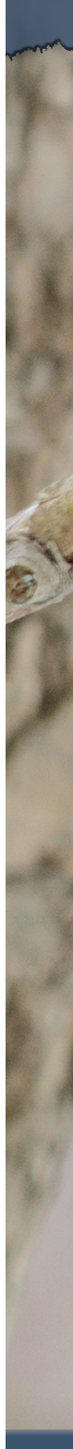
Grouping objects

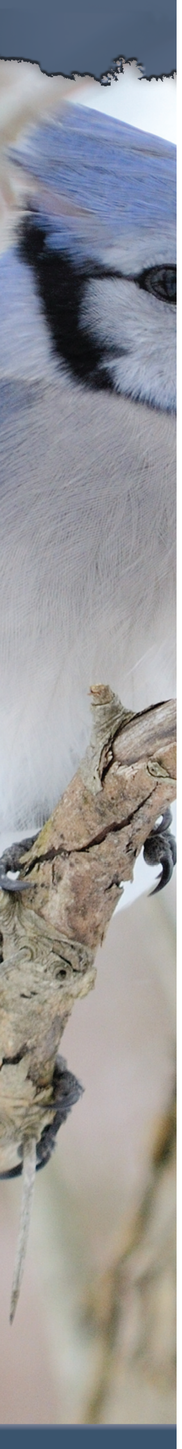
Collections and the for-each loop



Main concepts to be covered

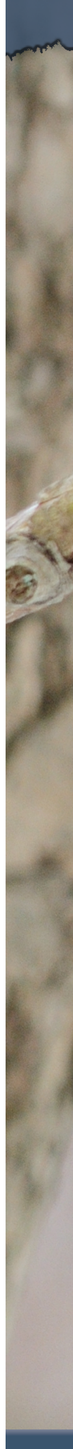
- Collections
- Iteration
- Loops: the for-each loop

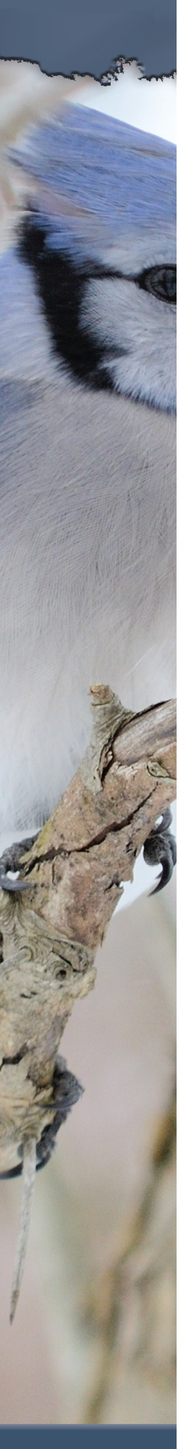




Iteration

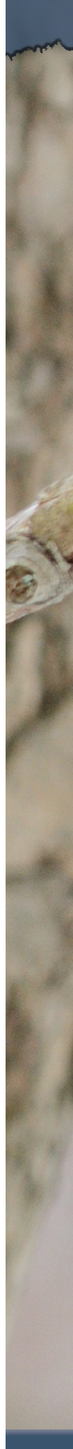
- We often want to perform some actions an arbitrary number of times.
  - E.g., print all the file names in the organiser. How many are there?
- Most programming languages include *loop statements* to make this possible.
- Java has several sorts of loop statement.
  - We will start with its *for-each loop*.



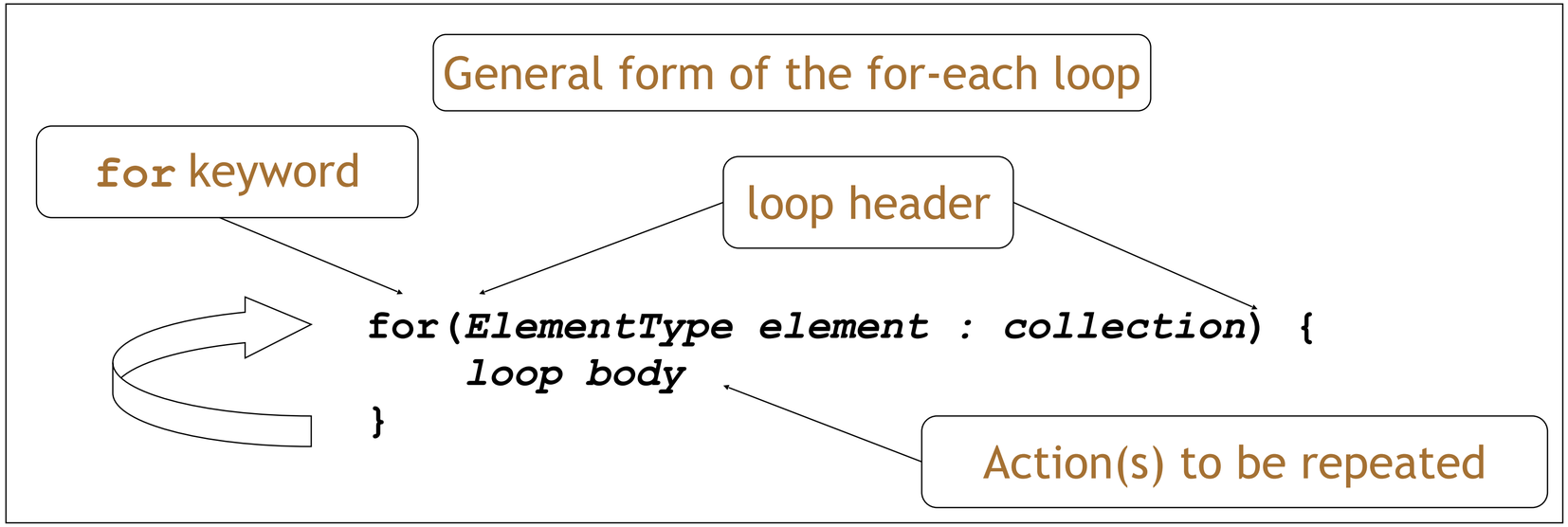


Iteration fundamentals

- The process of repeating some actions over and over.
- Loops provide us with a way to control how many times we repeat those actions.
- With a collection, we often want to repeat the actions: *exactly once for every object in the collection*.



For-each loop pseudo code



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

Using each *element* in *collection* in order, do the things in the *loop body* with that *element*.

A Java example

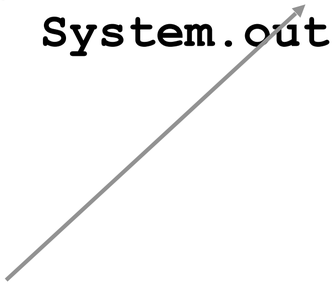
```
/**
 * List all file names in the organizer.
 */
public void listAllFiles()
{
    for(String filename : files) {
        System.out.println(filename);
    }
}
```

Using each *filename* in *files* in order, print *filename*

Selective processing

- Statements can be nested, giving greater selectivity to the actions:

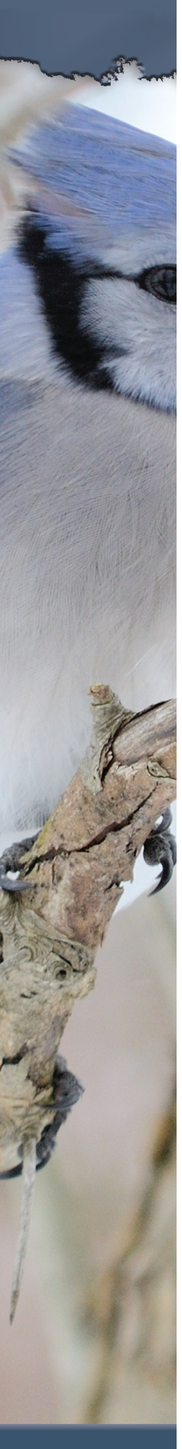
```
public void findFiles(String searchString)
{
    for(String filename : files) {
        if(filename.contains(searchString)) {
            System.out.println(filename);
        }
    }
}
```



`contains` gives a partial match of the filename;
use `equals` for an exact match

Critique of for-each

- Easy to write.
- Termination happens naturally.
- *The collection cannot be changed by the actions.*
- There is no index provided.
  - Not all collections are index-based.
- *We can't stop part way through*;
  - e.g., if we only want to find the first match.
- It provides 'definite iteration' – aka 'bounded iteration'.



Review

- Loop statements allow a block of statements to be repeated.
- The for-each loop allows iteration over a whole collection.
- With a for-each loop *every* object in the collection is made available *exactly once* to the loop's body.