Java Collections

Software Development 2

Today's lecture

- Recap:
  - Fixed-size collections, esp. **`Array`**
  - **`if/while`** statements
- New material:
  - Flexible-size collections, esp. **`ArrayList`**
  - **`For-each`** loops and **`Iterators`**

Recap: Arrays

- Fixed size collection.
- Object creation (2 options)

1
```
private int[] numbers = { 3, 15, 4, 5 };
```

2
```
numbers = new int[] { 3, 15, 4, 5 };
```

- Length

```
int n = numbers.length;
```

no brackets!

Standard array use

```
private int[] hourCounts;
private String[] names;

...

hourCounts = new int[24];

...

hourcounts[i] = 0;
hourcounts[i]++;
System.out.println(hourcounts[i]);
```

declaration

creation

use

Recap: While vs. For-Loops

- **For-loops** are used when an *index variable* is required.
- **For-loops** offer an alternative to while loops when the number of repetitions is known.
- Use **while-loops** if you need to do something repeatedly *until some condition* is met.

For-loop pseudo-code

General form of the 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

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++;
}
```

Today's lecture:

- Flexible-size Collections
  - especially **`ArrayList`**
  - Generics
  - Builds on the *abstraction* theme from the last lecture.
- **`For-each`** loop and **`Iterators`**.

An organizer for music files

- Track files may be added.
- There is no pre-defined limit to the number of files.

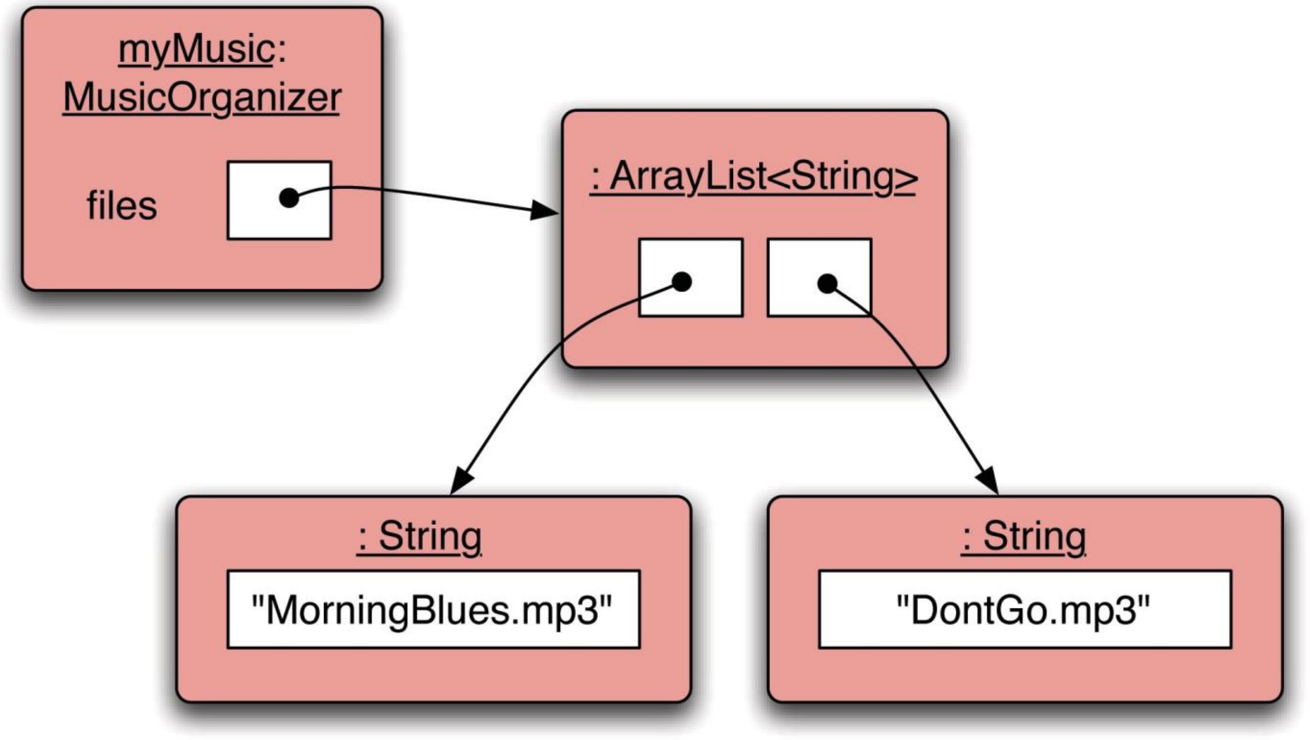
→ we can’t use a fixed size array!

Class libraries

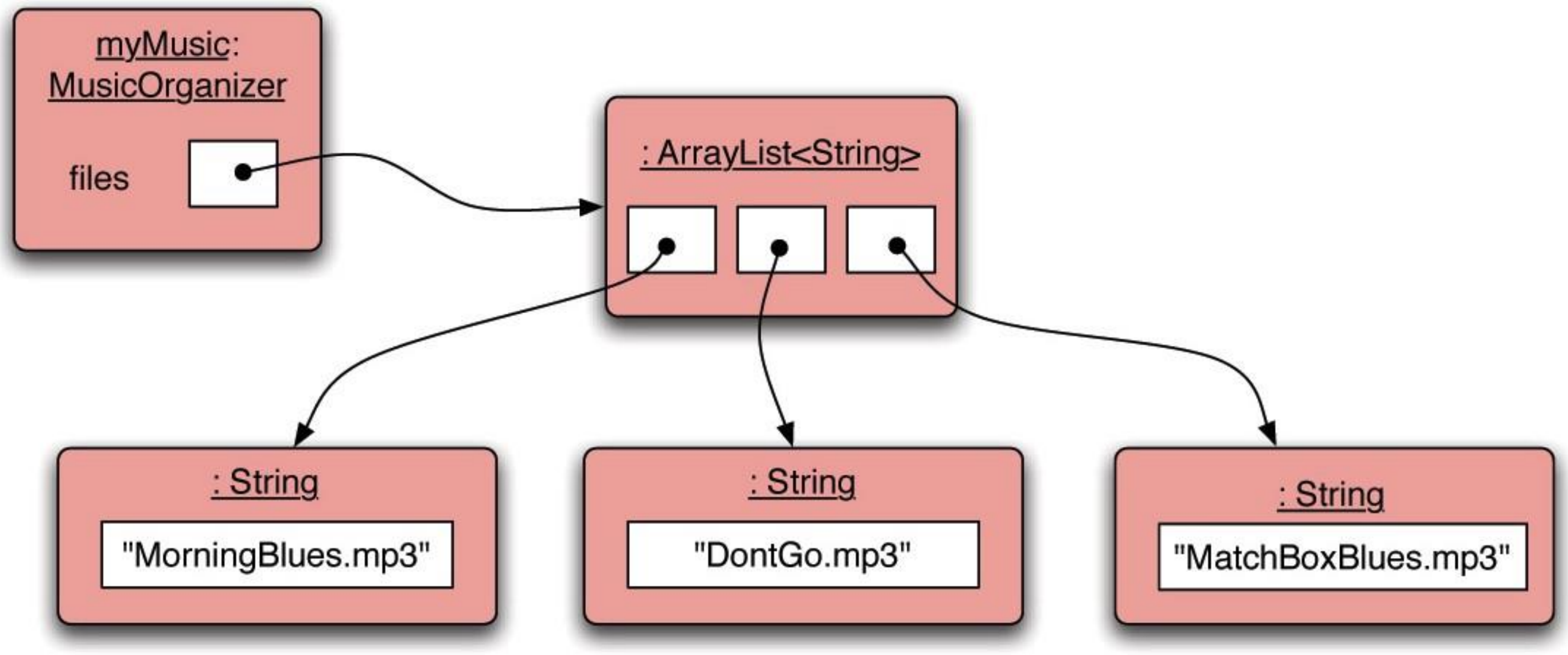
- We don't have to write everything from scratch.
- Java calls its libraries, ***packages***.
- Grouping objects is a recurring requirement.
  - The **`java.util`** package contains classes for doing this.

```java
import java.util.ArrayList;

/**
 * ...
 */
public class 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<String>();
    }


    ...
}
```

Object structures with collections



Adding a third file



Collections and Generics

- We specify:
  - the type of collection: **`ArrayList`**
  - the type of objects it will contain: **`<String>`**
  - **`private ArrayList<String> files;`**
- We say, "ArrayList of String".

Motivation

- Before J2SE 5.0 (no Generics):

```
List v = new ArrayList();
 v.add("test");
 Integer i = (Integer)v.get(0);
      // Run time error
```

- Using Generics:

```
 List<String> v = new ArrayList<String>();
 v.add("test");
 Integer i = v.get(0);
       // (type error)  Compile time error
```

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.

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

```
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*)

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.

Iterating trough Collections

Retrieving an object

```
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?)

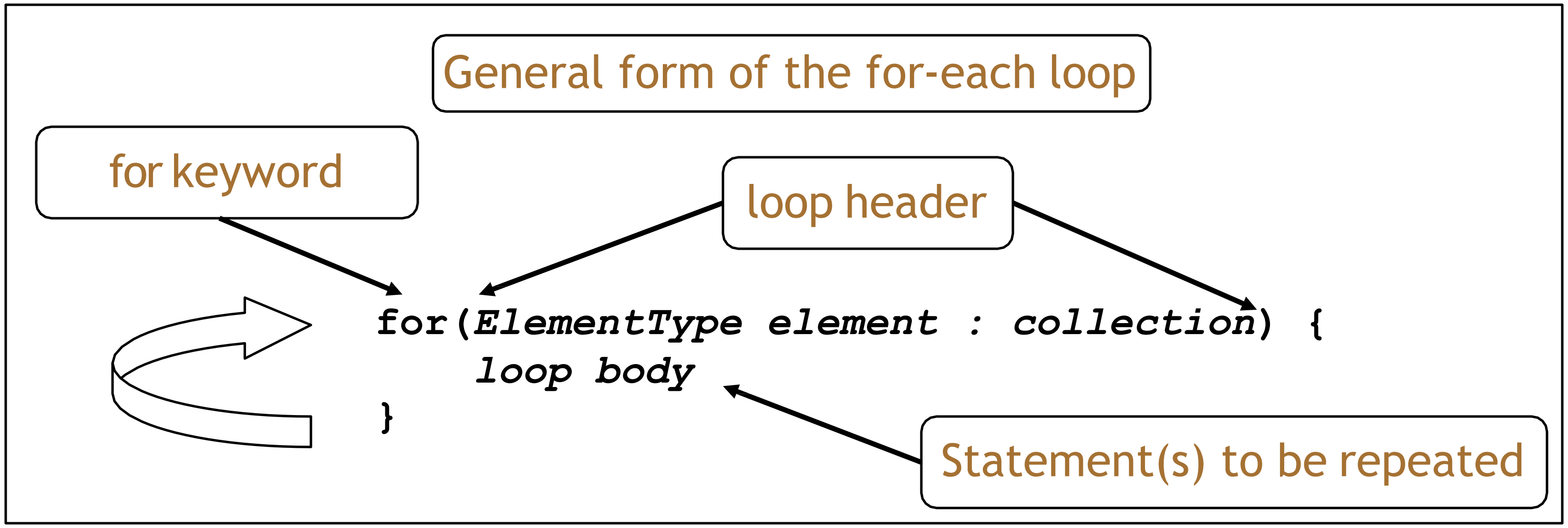
The general utility of indices

- 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`

Iteration fundamentals

- We often want to repeat some actions over and over.
- **Loops** provide us with a way to control how many times we repeat those actions.
- With collections, we often want to repeat things **once for every object** in a particular collection.
- The **for-each loop** allows iteration over a whole collection.

For-each loop pseudo code


General form of the for-each loop
for keyword
loop header
Statement(s) to be repeated

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

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

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

for each *filename* in *files*, print out *filename*

Selective processing

- Statements can be nested, giving greater selectivity:

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

Critique of For-each loop

Pros

- Easy to write.
- Termination happens naturally.
- Out-of-Bounds errors can be avoided.

Cons

- There is **no index** provided.
  - Not all collections are index-based.
  - Alternative: for(int i=1; i<11; i++)
- We can't stop part way through;
  - e.g. find-the-first-that-matches.
  - Alternative: when-loops with termination criteria.

Using an Iterator object

java.util.Iterator

returns an Iterator object

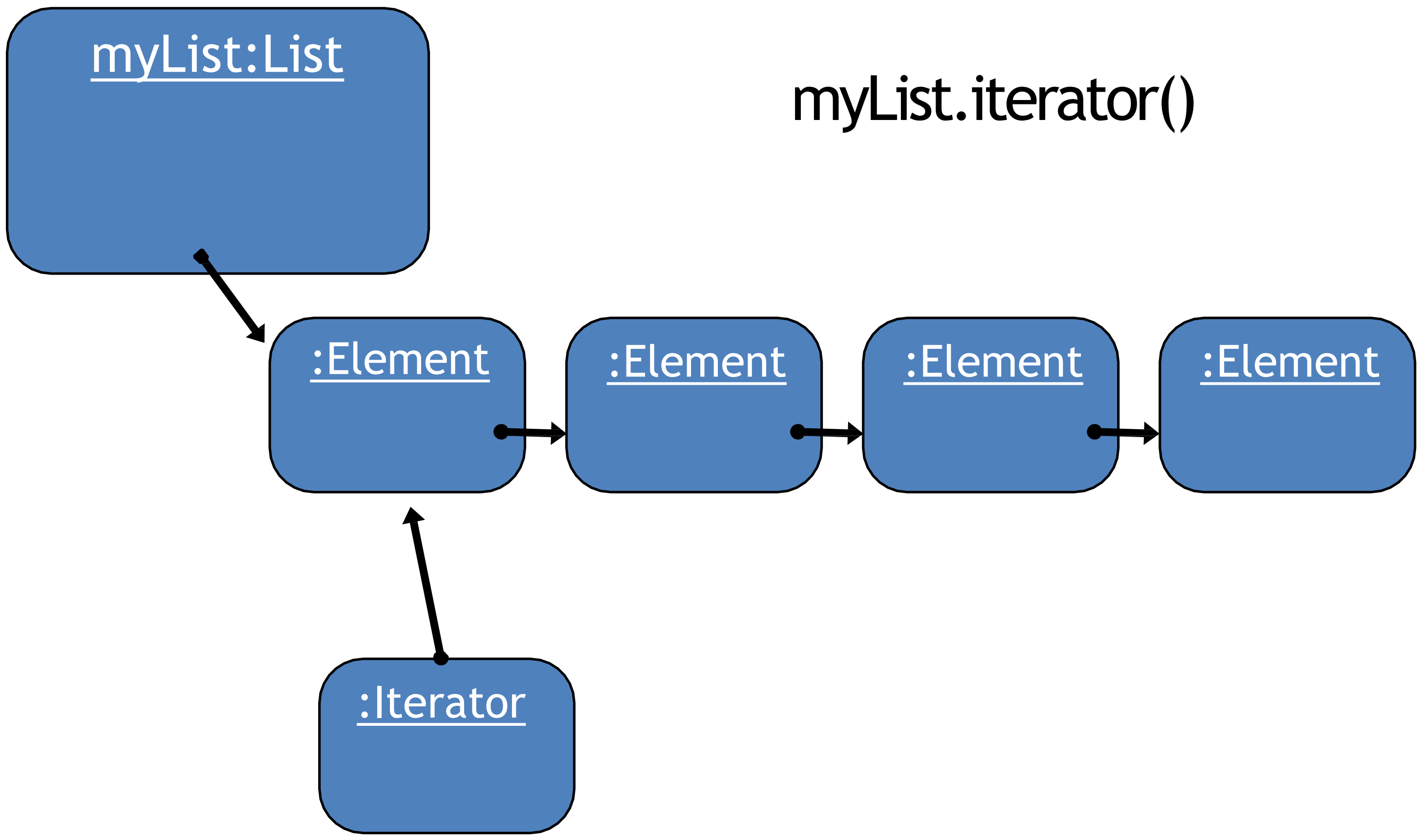
Pseudo-code

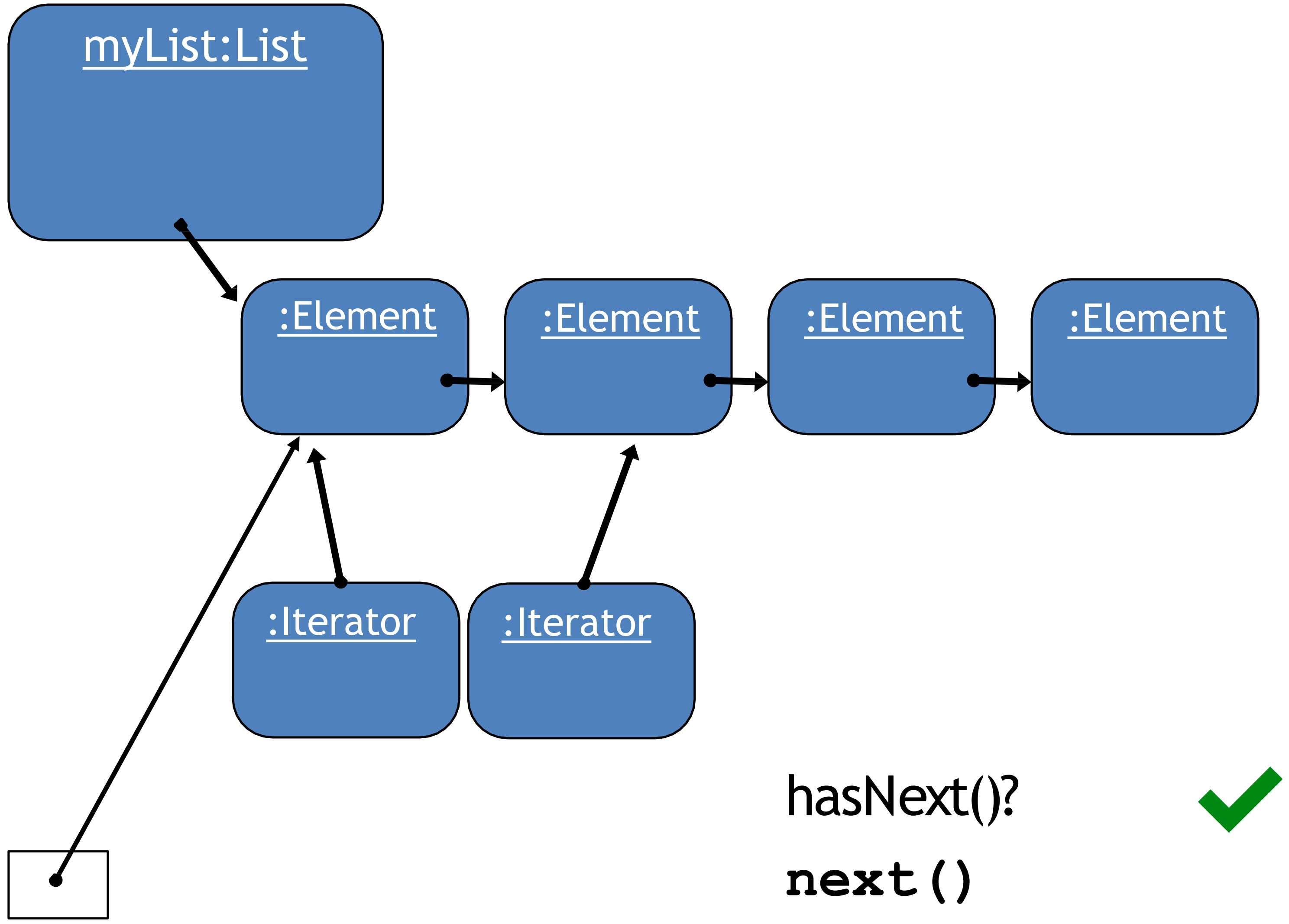
```
Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
    call it.next() //to get the next object
    // do something with that object
}
```

Java example

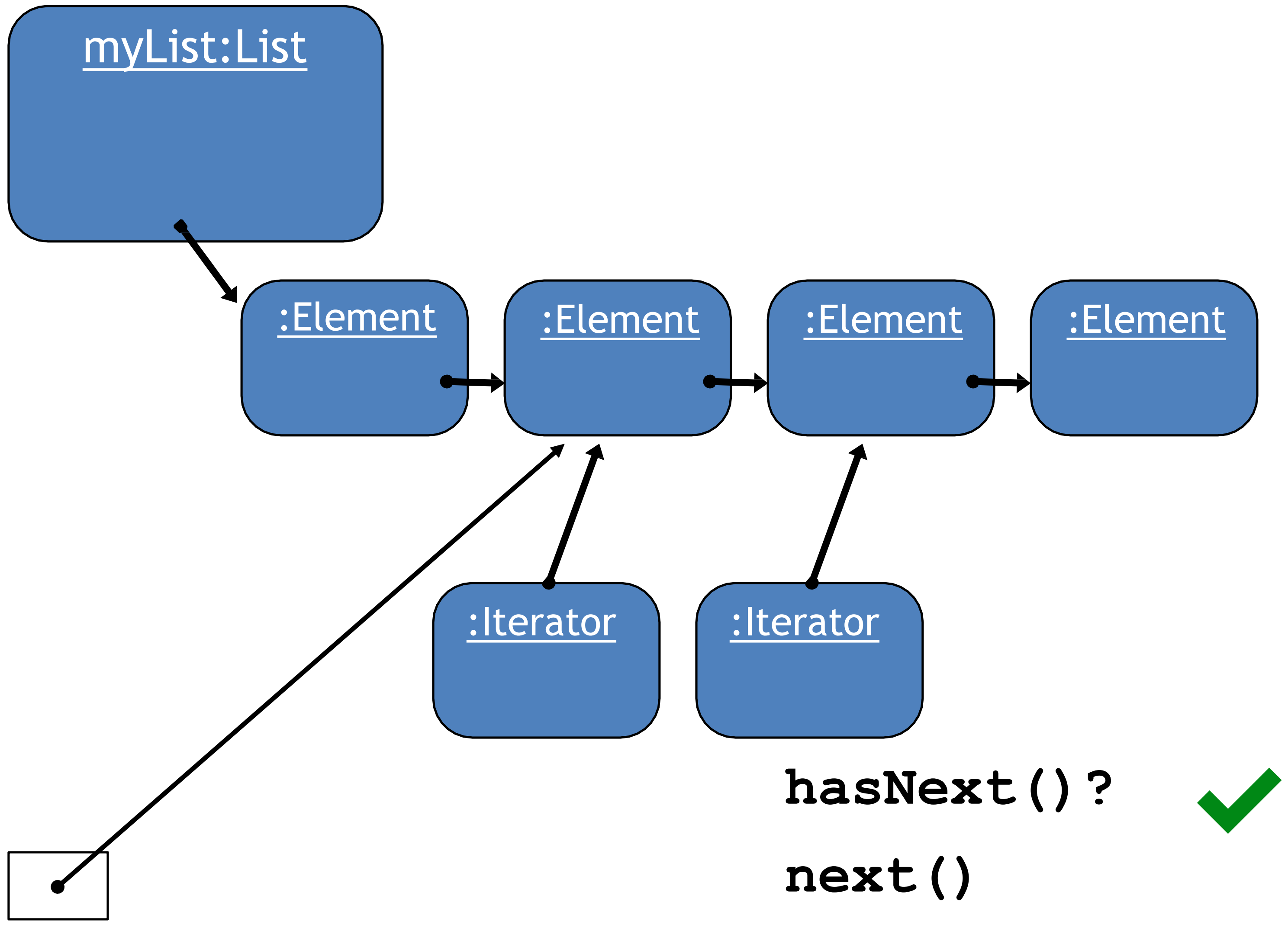
```
public void listAllFiles()
{
    Iterator<Track> it = files.iterator();
    while(it.hasNext()) {
        Track tk = it.next();
        System.out.println(tk.getDetails());
    }
}
```

myList.iterator()



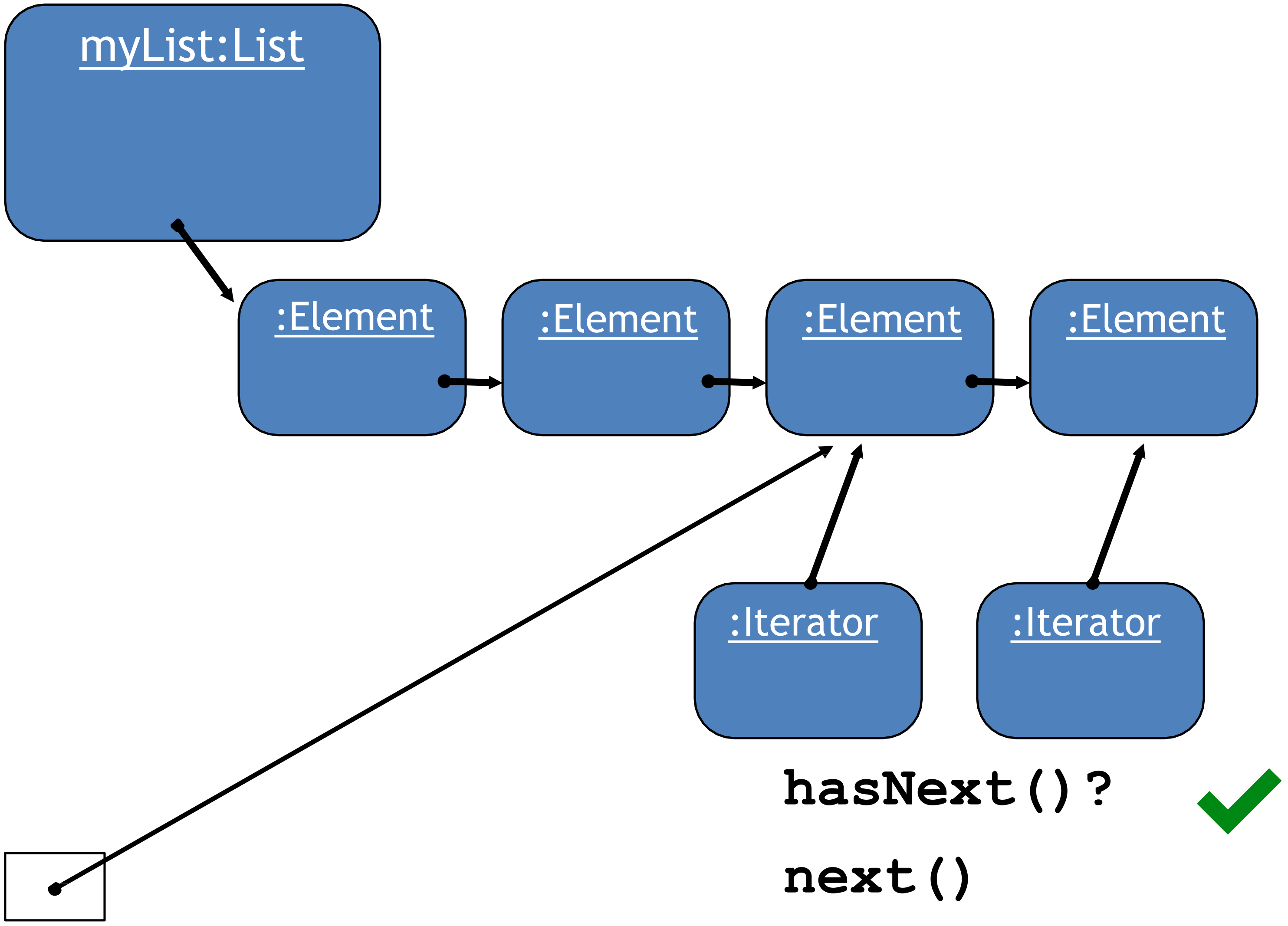


```
Element e = iterator.next();
```



`hasNext()?` ✓

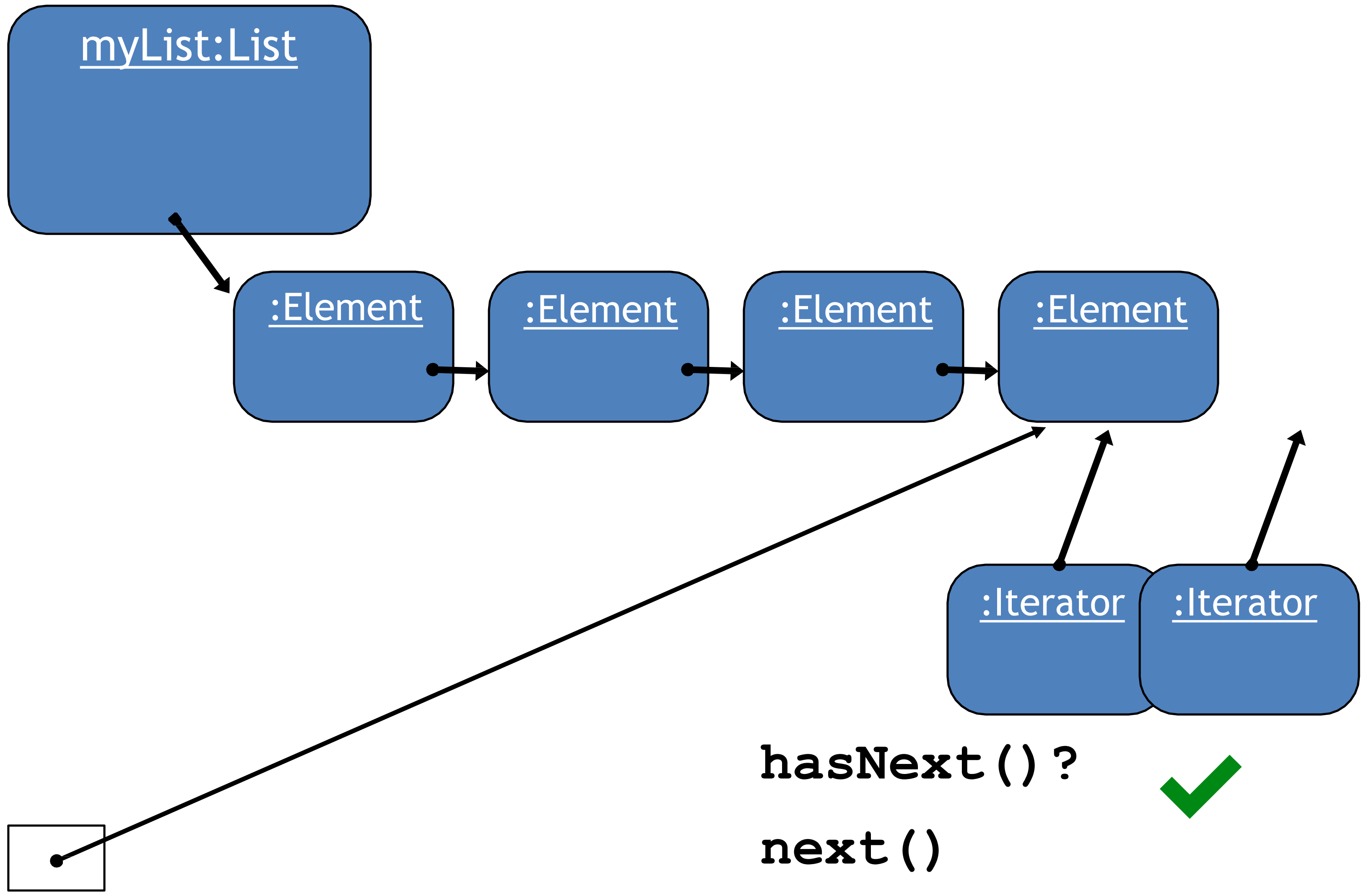
`next()`



```
hasNext()?
```

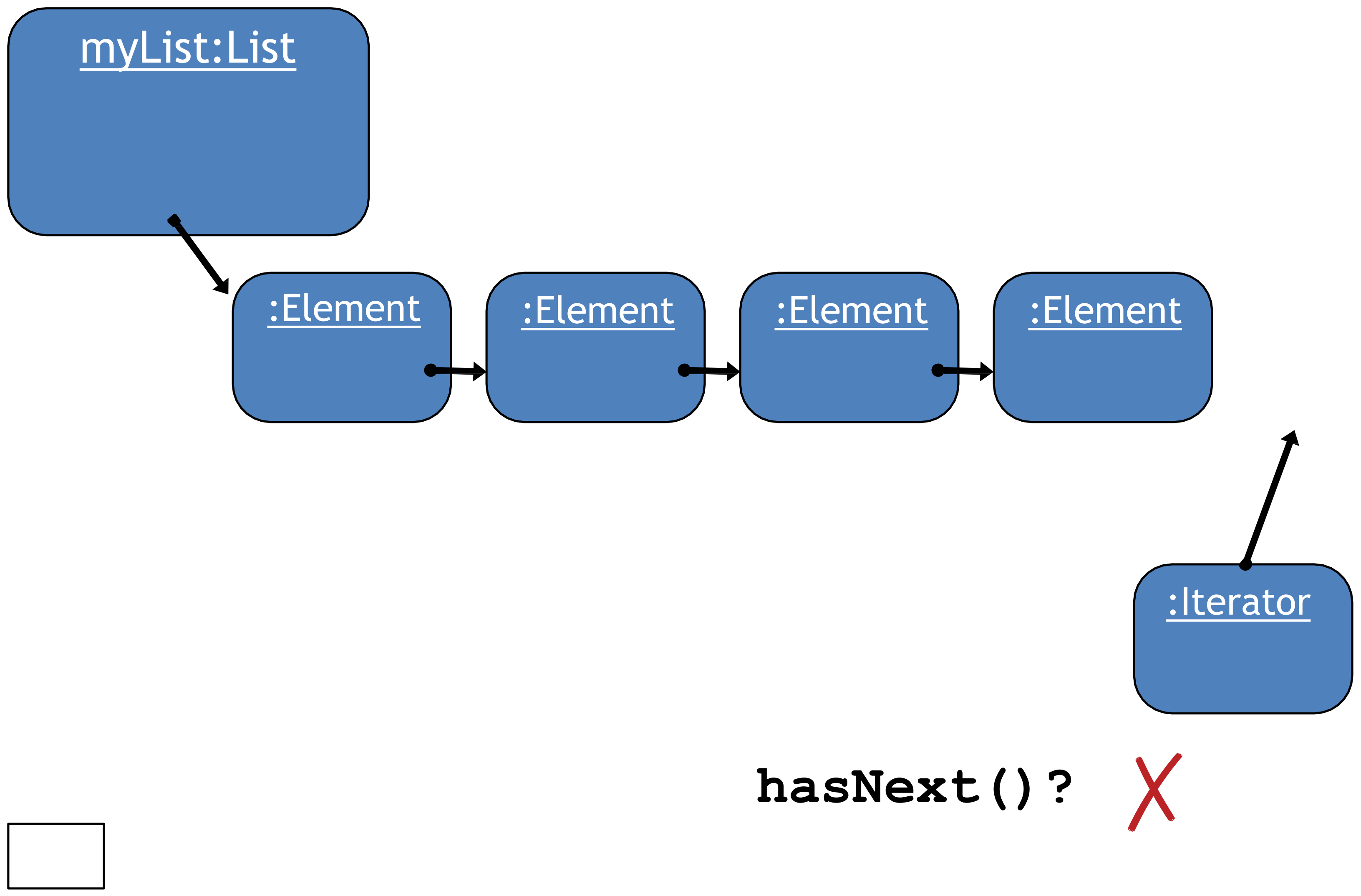
✓

```
next()
```



`hasNext()?` ✓

`next()`



hasNext()? ✗

Removing from a collection

```
Iterator<Track> it = tracks.iterator();
while(it.hasNext()) {
    Track t = it.next();
    String artist = t.getArtist();
    if(artist.equals(artistToRemove)) {
        it.remove();
    }
}
```

Use the `Iterator`'s `remove` method.

Index versus Iterator

- Ways to iterate over a collection:
  - **for-each loop**.
    - Use if we want to process <u>every</u> 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.
    - Use to remove from a collection.
- Iteration is an important programming *pattern*.

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.

Review

- **Loop statements** allow a block of statements to be repeated.
- The **for-each loop** allows iteration over a whole collection.
- For **loops with indices** allow direct access.
- The **while loop** allows the repetition to be controlled by a boolean expression (termination criteria).
- All collection classes provide special **`Iterator`** objects that provide sequential access to a whole collection.

Homework

Object's First Book (5th edition):

- Read chapter 4 (all!).