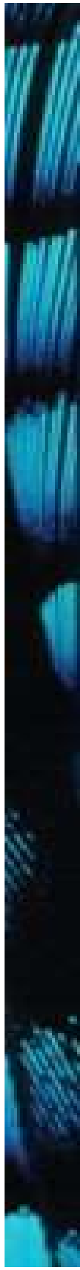
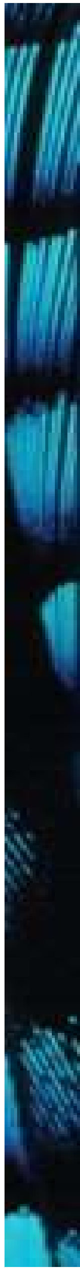
Grouping objects

Iterators



`Iterator` and `iterator()`

- Collections have an **`iterator()`** method.
- This returns an **`Iterator`** object.
- **`Iterator<E>`** has three methods:
  - **`boolean hasNext()`**
  - **`E next()`**
  - **`void remove()`**



Using an Iterator object

java.util.Iterator

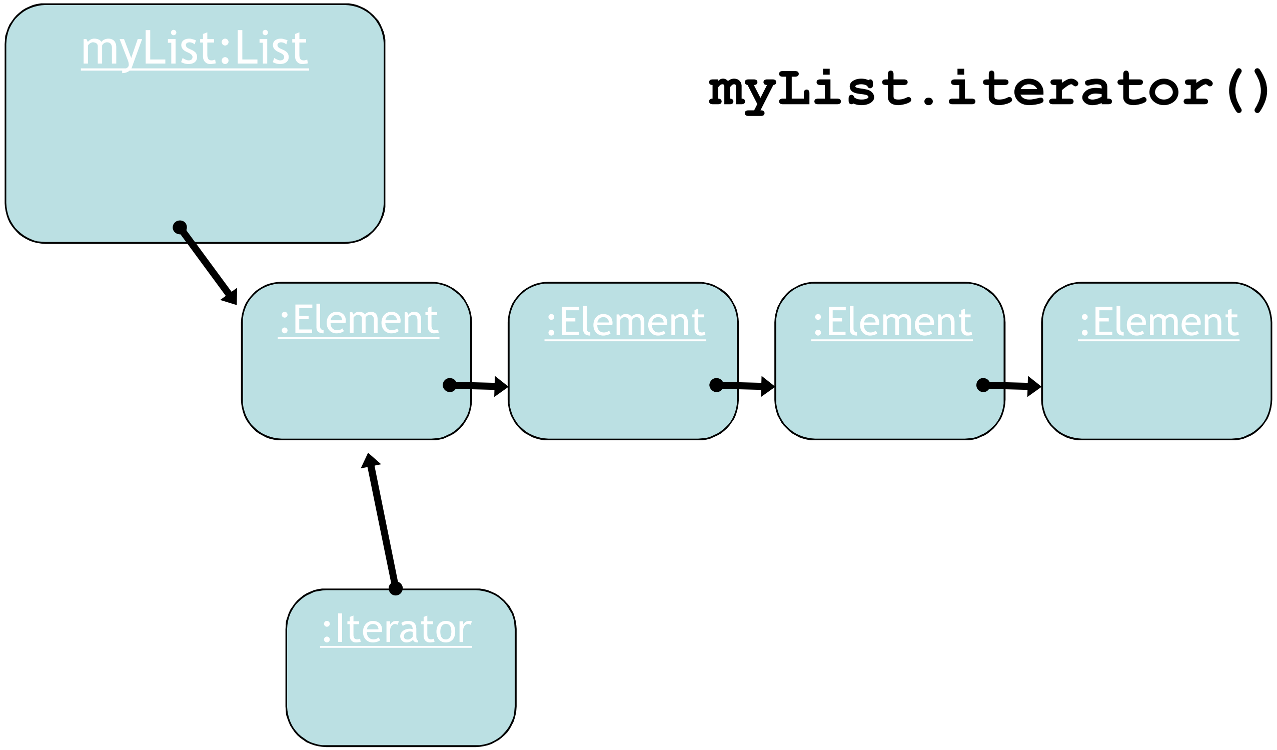
returns an Iterator object

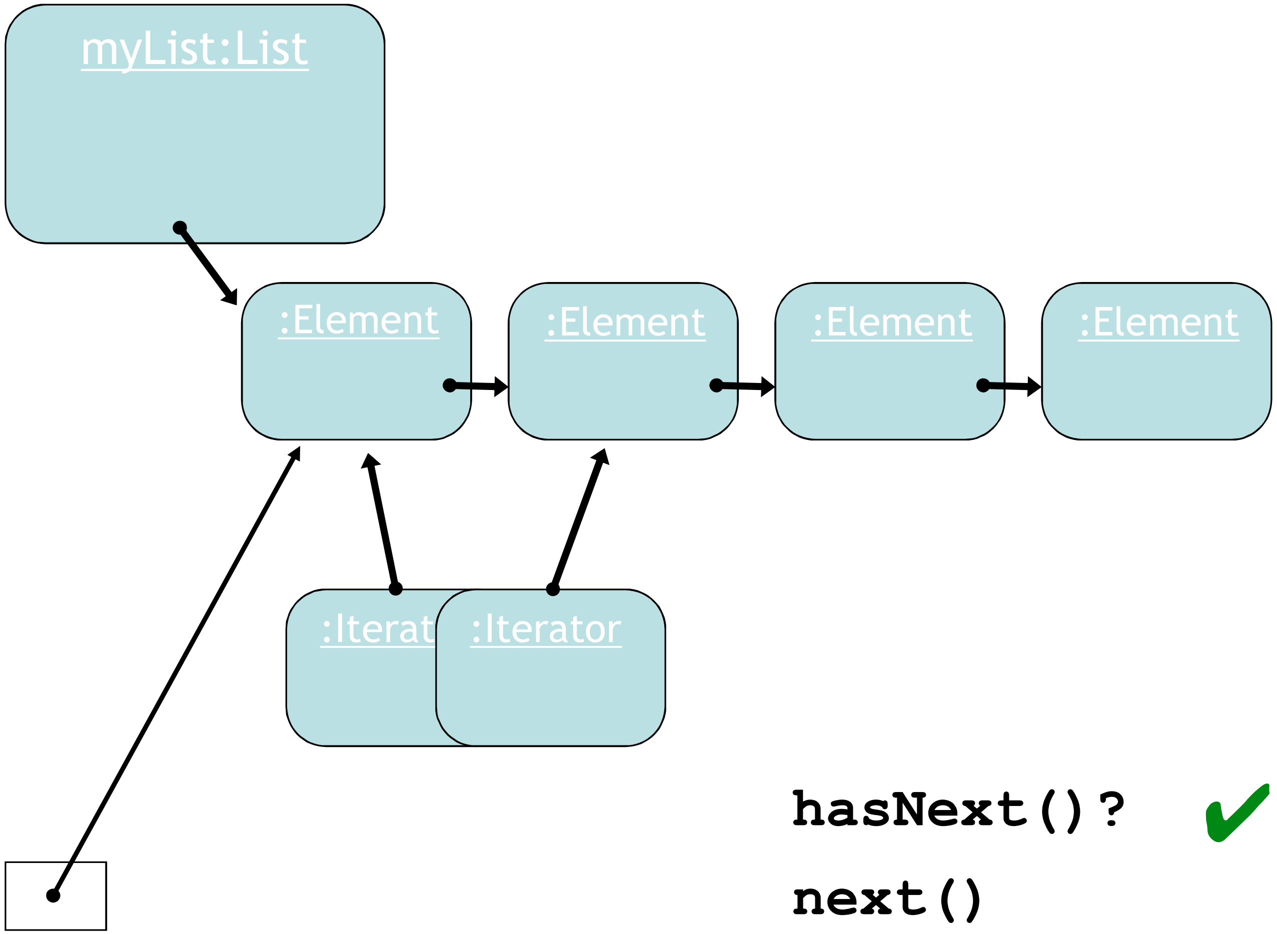
```
Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}

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

Iterator mechanics

`myList.iterator()`

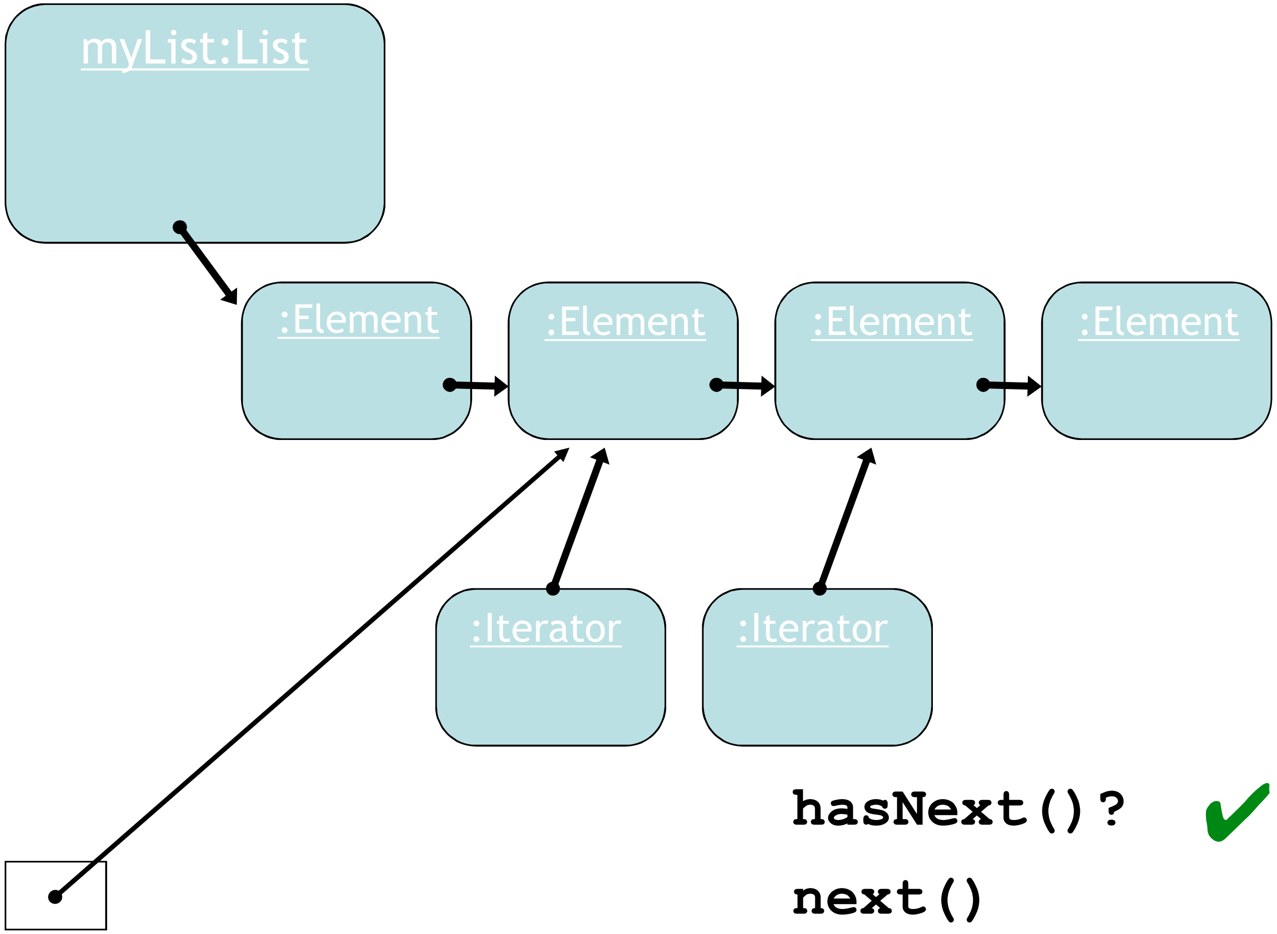




hasNext()? ✓

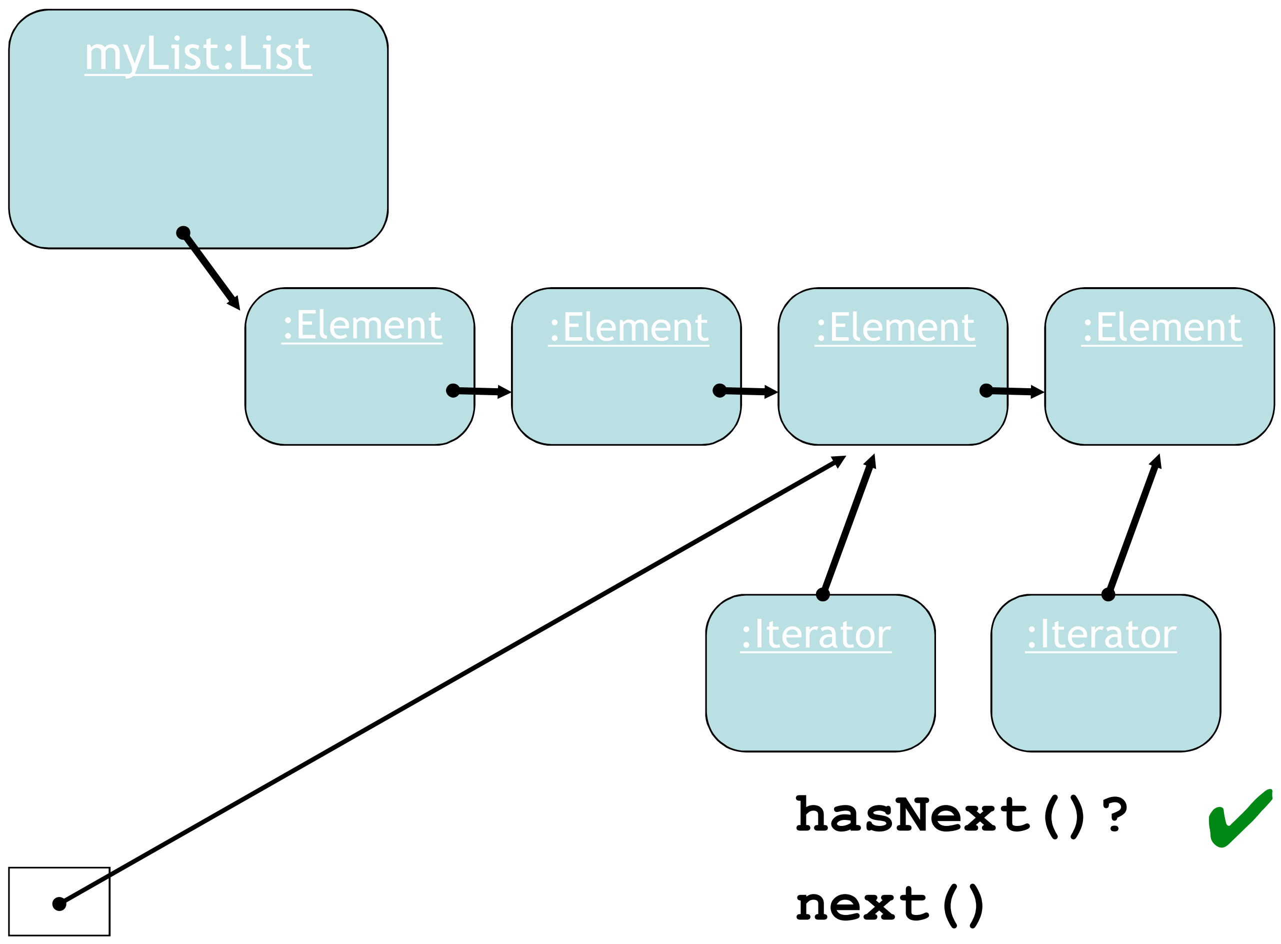
next()

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



hasNext()? ✓

next()



myList:List

:Element

:Element

:Element

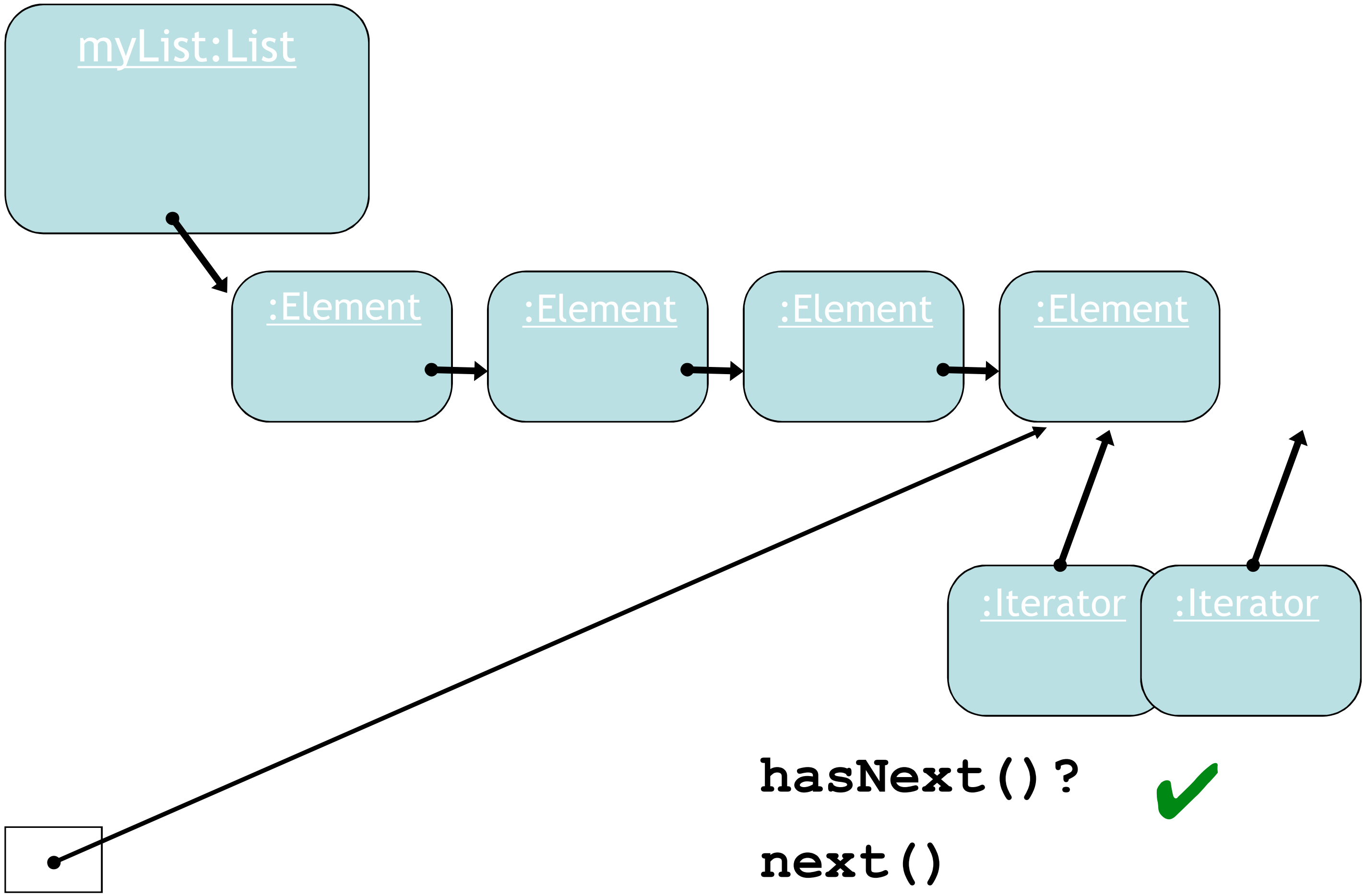
:Element

:Iterator

:Iterator

hasNext()? ✓

next()



myList:List

:Element

:Element

:Element

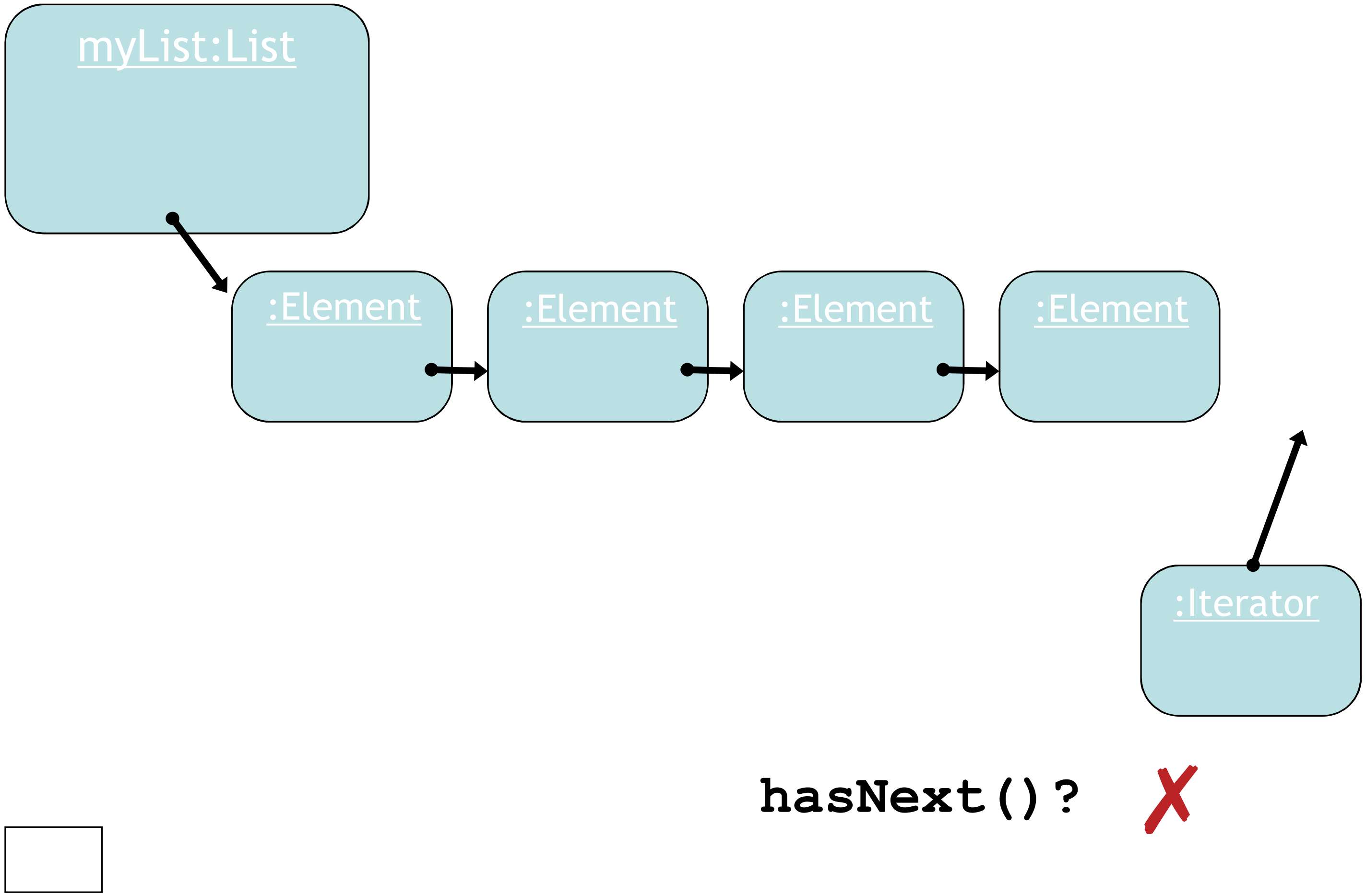
:Element

:Iterator

:Iterator

hasNext()? ✓

next()



myList:List

:Element

:Element

:Element

:Element

:Iterator

`hasNext()?` ✗

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.
    - Use to remove from a collection.
- Iteration is an important programming *pattern*.

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.

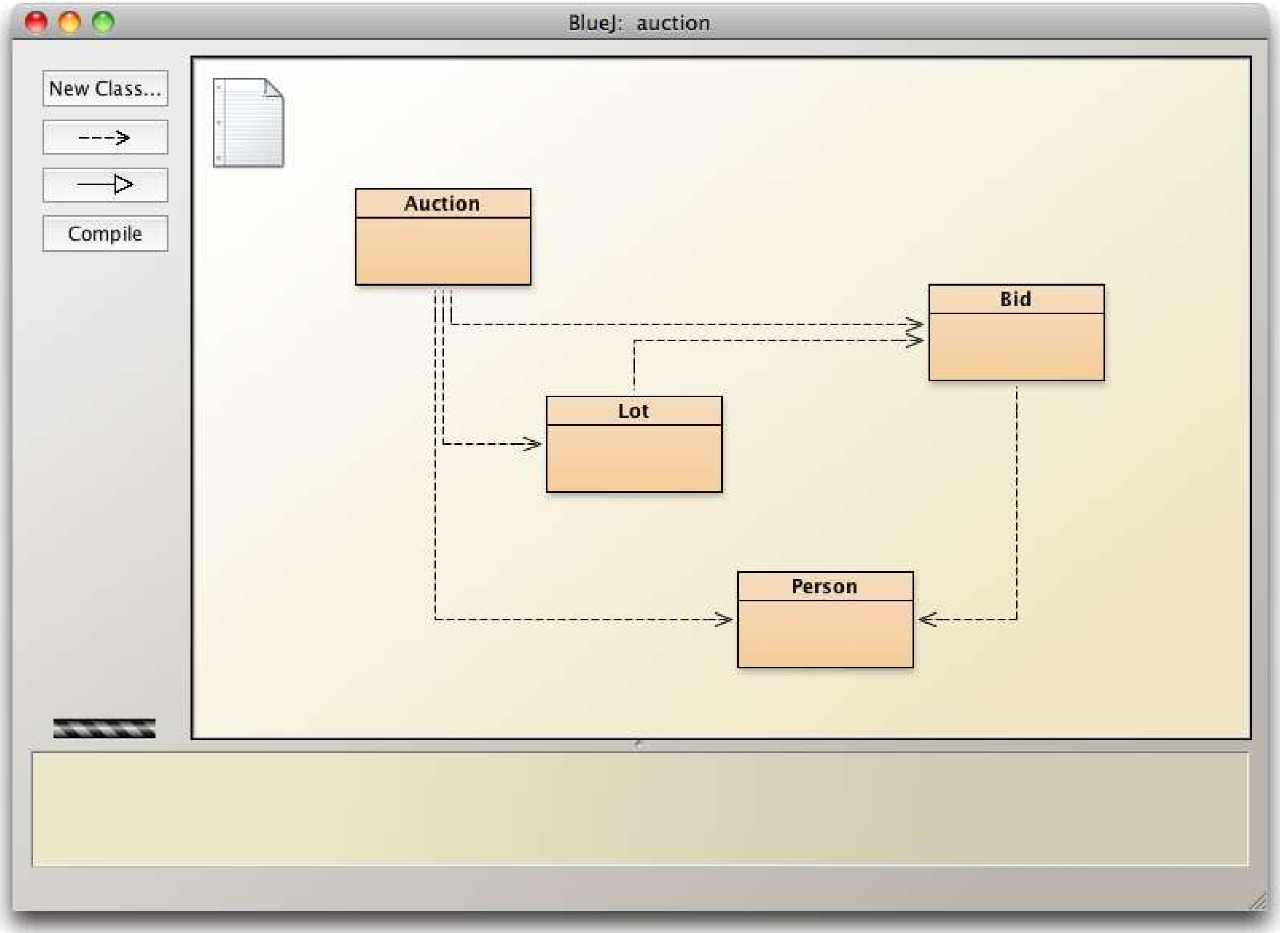
Review

- Loop statements allow a block of statements to be repeated.
- The for-each loop allows iteration over a whole collection.
- The while loop allows the repetition to be controlled by a boolean expression.
- All collection classes provide special `Iterator` objects that provide sequential access to a whole collection.

The *auction* project

- The *auction* project provides further illustration of collections and iteration.
- Examples of using **`null`**.
- Anonymous objects.
- Chaining method calls.

The auction project



null

- Used with object types.
- Used to indicate, 'no object'.
- We can test if an object variable holds the **null** value:

```
if(highestBid == null) …
```

- Used to indicate ‘no bid yet’.

Anonymous objects

- Objects are often created and handed on elsewhere immediately:

```
Lot furtherLot = new Lot(…);
lots.add(furtherLot);
```

- We don’t really need **`furtherLot`**:

```
lots.add(new Lot(…));
```

Chaining method calls

- Methods often return objects.
- We often immediately call a method on the returned object.

```
Bid bid = lot.getHighestBid();
Person bidder = bid.getBidder();
```

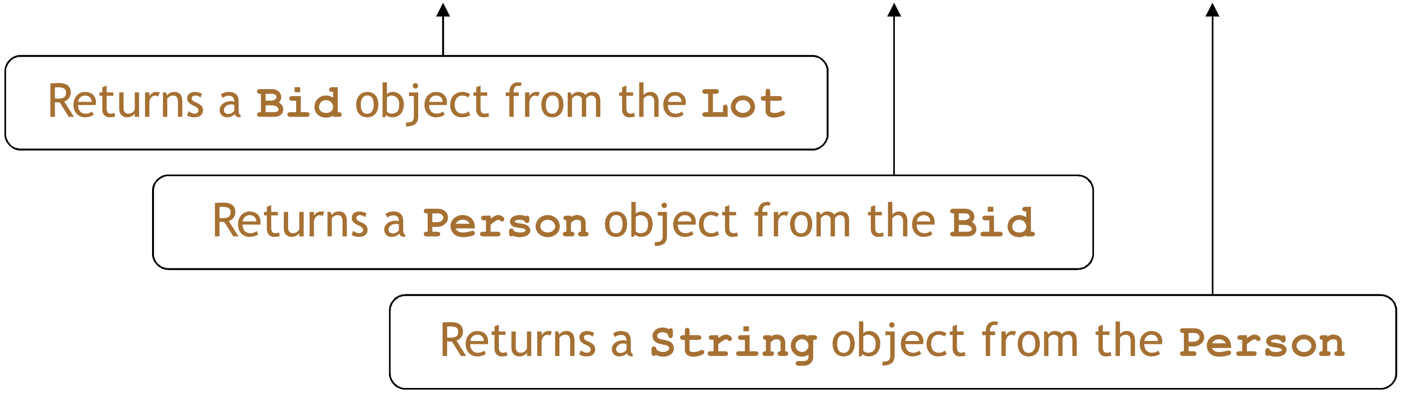
- We can use the anonymous object concept and *chain* method calls:

```
lot.getHighestBid().getBidder()
```

Chaining method calls

- Each method in the chain is called on the object returned from the previous method call in the chain.

```
String name =
    lot.getHighestBid().getBidder().getName();
```



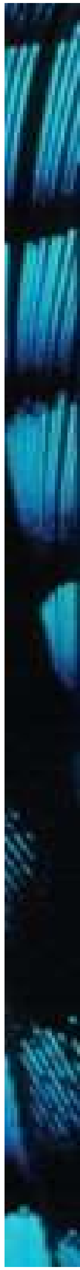
Returns a **Bid** object from the **Lot**

Returns a **Person** object from the **Bid**

Returns a **String** object from the **Person**

Grouping objects

Arrays



Fixed-size collections

- Sometimes the maximum collection size can be pre-determined.
- A special fixed-size collection type is available: an *array*.
- Unlike the flexible `List` collections, arrays can store object references or primitive-type values.
- Arrays use a special syntax.

The *weblog-analyzer* project

- Web server records details of each access.
- Supports analysis tasks:
  - Most popular pages.
  - Busiest periods.
  - How much data is being delivered.
  - Broken references.
- Analyze accesses by hour.

Creating an array object

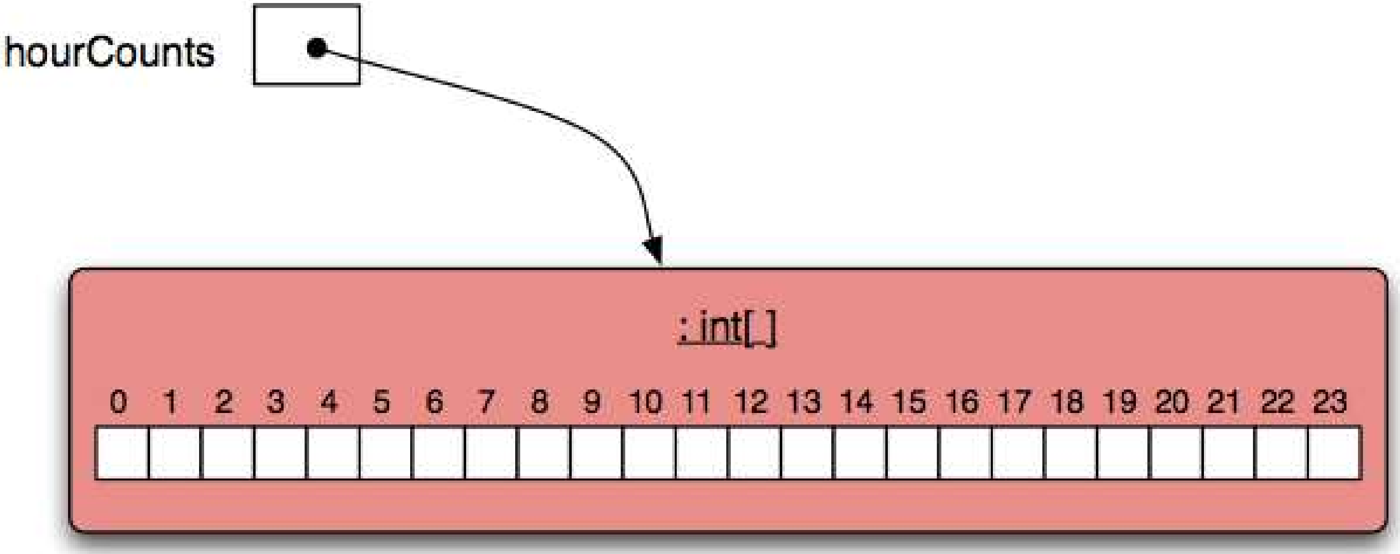
```
public class LogAnalyzer
{
    private int[] hourCounts;
    private LogfileReader reader;

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

Array *variable declaration*
— does *not* contain size

Array *object creation*
— specifies size

The `hourCounts` array



Using an array

- Square-bracket notation is used to access an array element: `hourCounts[...]`
- Elements are used like ordinary variables.
- The target of an assignment:

```
hourCounts[hour] = ...;
```

- In an expression:

```
hourCounts[hour]++;
adjusted = hourCounts[hour] - 3;
```

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

Array literals

- The size is inferred from the data.

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

declaration, creation and initialization

- Array literals in this form can only be used in declarations.
- Related uses require **`new`**:

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

Array length

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

int n = numbers.length;
```

no brackets!

- NB: **`length`** is a field rather than a method!
- It cannot be changed – ‘fixed size’.

The for loop

- There are two variations of the for loop, *for-each* and *for*.
- The for loop is often used to iterate a fixed number of times.
- Often used with a variable that changes a fixed amount on each iteration.

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++;
}
```

Practice

- Given an array of numbers, print out all the numbers in the array, using a for loop.

```
int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

for ...
```

Practice

- Fill an array with the Fibonacci sequence.

0 1 1 2 3 5 8 13 21 34 ...

```
int[] fib = new int[100];

fib[0] = 0;
fib[1] = 1;

for ...
```

for loop with bigger step

```
// Print multiples of 3 that are below 40.
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```

Review

- Arrays are appropriate where a fixed-size collection is required.
- Arrays use a special syntax.
- 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.
- Used with a regular step size.