CITS1001 week 4
Grouping objects – lecture 2

Arran Stewart

March 29, 2018

Overview

- Last lecture, we looked at how we can group objects together into collections
  - We looked at the `ArrayList` class.
- This lecture, we'll look at doing something *for each* object in a collection
- And next lecture, we'll look at more general ways of doing things repeatedly

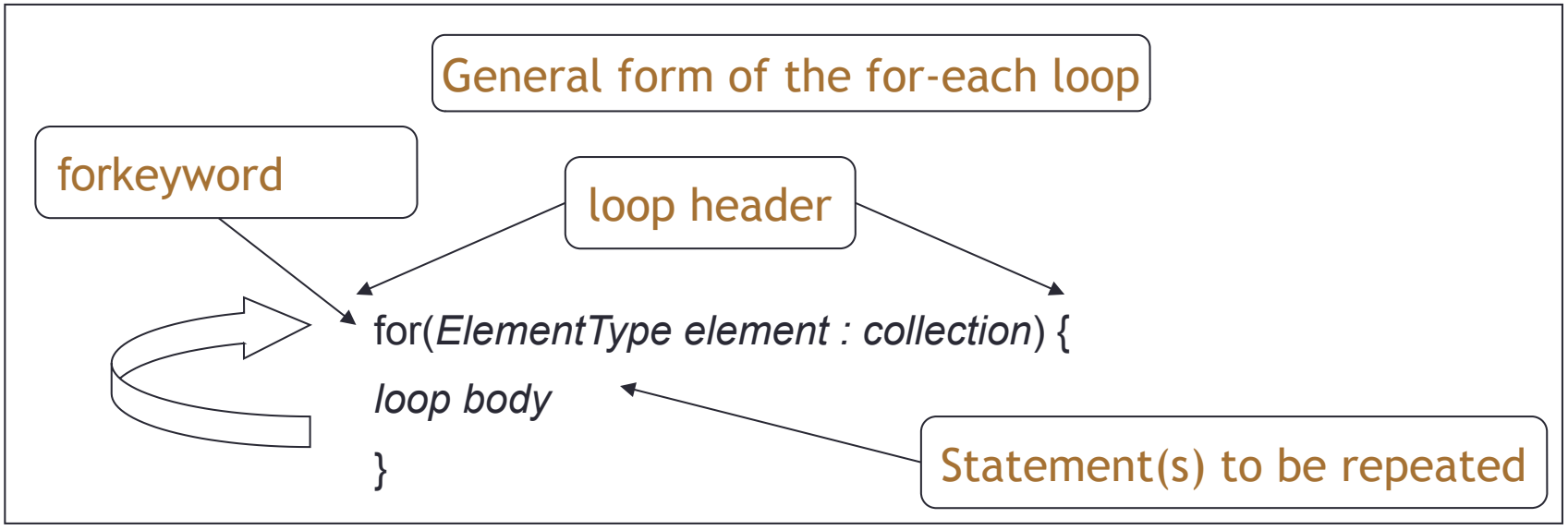
Iteration

- Often in programs, we want to repeat some action over and over (usually with slight variations)
  - e.g. "do this action *for each* student in the university"
  - e.g. "do this action *seventeen times*"
  - e.g. "do this action to the file, *until* it is over 5MB in size"
- Sometimes we know exactly how many times we want to do the action, sometimes we just want to keep going until some condition is met

Iteration over collections

- With collections, we often want to repeat things once, for every object in the collection
- The Java construct for doing this is the **for-each** loop.
- We'll see examples of other kinds of loop besides **for-each** later (e.g. when you want to repeat something until a condition becomes true)
  - Sometimes you could use different kinds of loop to achieve the same result.
  - It's best to choose the kind of loop that most simply and directly expresses what you want.

For-each loop pseudocode



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

For each *element* in *collection*, do the things in the *loop body*.

Example

- The "book journal" class from last lecture let us print the title of a book at a specific index:

```
public class BooksReadJournal {

  private ArrayList<String> bookTitles;

  // ...

  /**
   * Print the details of a book from the collection.
   * @param index The index of the book whose details
   *              are to be printed.
   */
  public void printBookTitle(int index) {

    // ...
```

Example (cont'd)

- We could add the ability to print the title of *all* books in the journal – i.e., "*for each* book, print its title"

Example (cont'd)

- We could add the ability to print the title of *all* books in the journal – i.e., "*for each* book, print its title"
- code:

```
/**
* List all book titles recorded in my book journal
*/
public void printAllTitles() {
  for(String bookTitle : bookTitles) {
    System.out.println(bookTitle);
  }
}
```

Exercise

- Declare an `ArrayList` called `cits1001` of `Student` objects.
- Initialise `cits1001` (i.e., create a new `ArrayList` object).
- Implement a method, `listAllStudentNames`, that prints the names of each student.

  (You may assume whatever methods you need have been implemented in the Student class.)

More complex logic

- A for-each loop iterates over every item in the collection – so what if we only want to perform an action for some of them?

  e.g. "*for each* student in the university, print their name *if* they are taking French Studies 1 *and* Ancient Greek Language and Literature"

More complex logic

- A for-each loop iterates over every item in the collection – so what if we only want to perform an action for some of them?

  e.g. "*for each* student in the university, print their name *if* they are taking French Studies 1 *and* Ancient Greek Language and Literature"

- We can use an **if** statement, just as we did when validating parameters.

More complex logic

- A for-each loop iterates over every item in the collection – so what if we only want to perform an action for some of them?

  e.g. "*for each* student in the university, print their name *if* they are taking French Studies 1 *and* Ancient Greek Language and Literature"

- We can use an **if** statement, just as we did when validating parameters.

- Pseudocode:

```
for each student in university:
  if (student takes French and student takes Ancient Greek):
    print their name
```

Example of selective processing

- We could print only books whose title contains some search string:

```
public void findBooks(String searchString) {
  for(String bookTitle : bookTitle) {
    if(bookTitle.contains(searchString)) {
      System.out.println(bookTitle);
    }
  }
}
```

Exercise – total marks

- Assume we have a `cits1001` object containing `Students`, and that each student has a `getMark()` accessor method.

  How can we calculate the total marks scored by the class? Write a method for doing this.

Exercise – average

- Now that we have the total marks – how do we calculate the average?

Exercise – average

- Now that we have the total marks – how do we calculate the average?
- Is our code reliable? Are their situations where it may not work? How should we handle those situations?

Search and return pattern

- Often, we'll want to look through a collection, looking for an item that matches particular criteria, and return it *if* we find one.
- In pseudocode:

```
for each object in collection:
  if object meets criteria:
    return object
// if we are here, no object was found -
// do something else
```

Search and return pattern in Java

- Let's create a `findBook` method which returns the *first* book (if any) whose title contains a search string

```
public String findBook(String searchString) {
  for (String bookTitle : bookTitles ) {
    if (bookTitle.contains(searchString)) {
      return bookTitle; // return first match if found
    }
  }
  // if we are here, no book contained the string
  System.out.println("No matching book title found");
  return null; //return null object
}
```

Documenting our method

- Note that we should write a comment for our method, saying what it returns, so that programmers using our method know what to expect:

```
/** Search the journal for a book title containing
 *  searchString.
 *
 *  If some book contains the search string, the
 *  first matching book is returned; otherwise,
 *  null is returned.
 */
public String findBook(String searchString) {
  // ...
```

Documenting our method

- Note that we should write a comment for our method, saying what it returns, so that programmers using our method know what to expect:

```
/** Search the journal for a book title containing
 *  searchString.
 *
 *  If some book contains the search string, the
 *  first matching book is returned; otherwise,
 *  null is returned.
 */
public String findBook(String searchString) {
  // ...
```

- We will see a more formal way of doing this in future lectures.

Exercise - search and return

- Write the signature for a method, `findStudent`, that will search for a particular name in the `cits1001` ArrayList and return the `Student` object which has that name.

Exercise - search and return

- Write the signature for a method, `findStudent`, that will search for a particular name in the `cits1001` ArrayList and return the `Student` object which has that name.
- Now write the implementation of the method.
  (Hint: how will we tell if a `Student` has the name we are looking for?)

Exercise - search and return

- Write the signature for a method, `findStudent`, that will search for a particular name in the `cits1001` ArrayList and return the `Student` object which has that name.
- Now write the implementation of the method.
  (Hint: how will we tell if a `Student` has the name we are looking for?)
- Challenge: If no `Student` objects have that name, we'd like to print an error message – how can we do that?

Pros and cons of for-each

- Pros:
  - Simple to write
  - Don't have to worry about termination conditions
- Cons:
  - We can't add or remove things from the collection

    (What do you think will happen if we try?)
  - No access to the index for an element

    (What if we wanted to print the position of each book, in the journal?)
  - Only way we've seen to stop part-way through is `return`

When to use a for-each loop

- We want to perform some action on every item in a collection:
  - print every one
  - change every one
  - count every one
- We don't need access to the position index
- We don't need to add or remove things from the collection

Other ways of processing a collection

- What if we do want to remove something from the collection?
- One way is to use a type of object called an *iterator*.

Iterators

How do we get an iterator?

- All collections have a method called `iterator()` that will gives us an Iterator object.
- An iterator "points" to a particular spot in the collection

What can it do?

- An iterator lets us do 3 things:
  - see if there's another item still to be processed
  - retrieve that item
  - remove that item from the collection

Iterators are generic

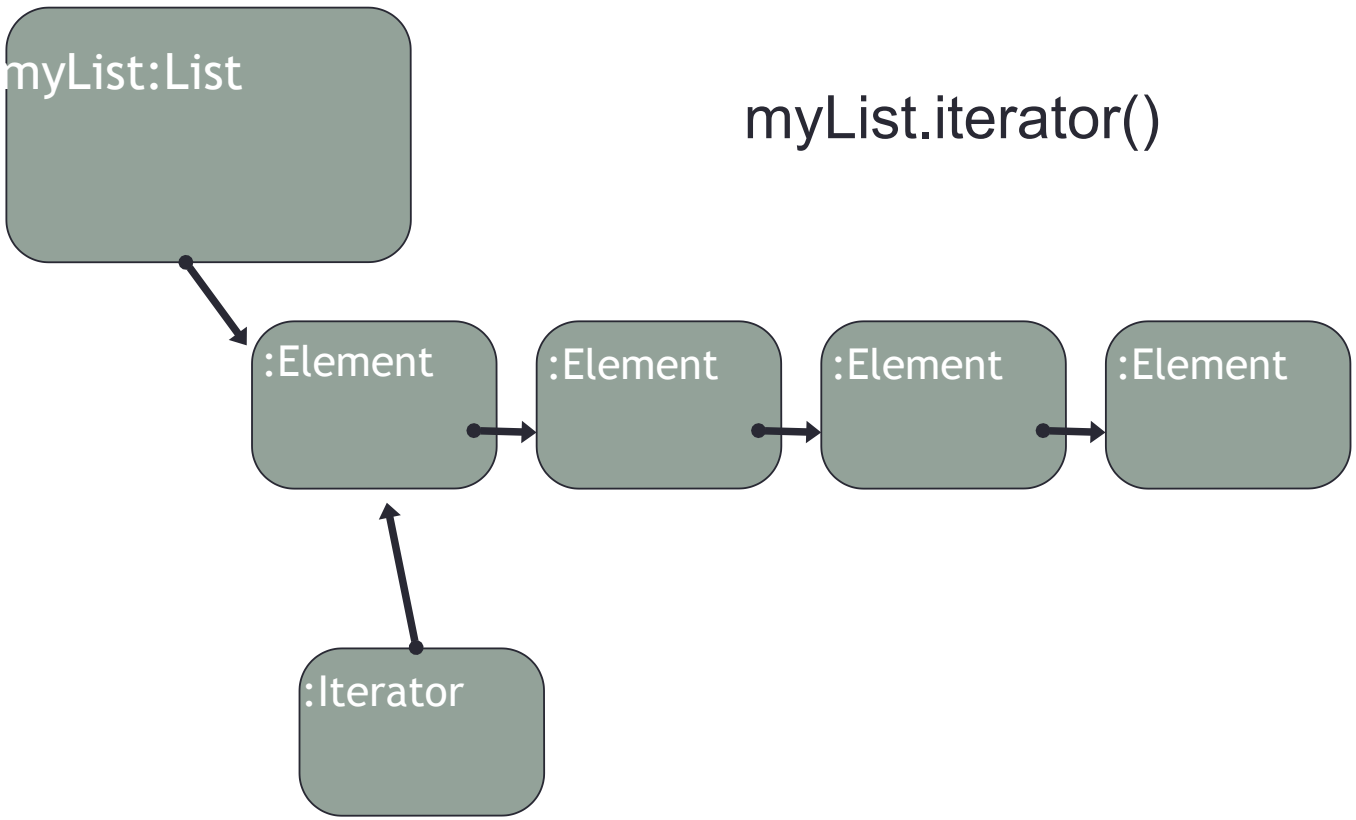
- Like an `ArrayList`, an `Iterator` is a generic or parametric type
  - We can have an `Iterator` that iterates over `Students`, or `Integers`, or any other type of object.
- An `ArrayList` of `Students` would be `ArrayList<Student>` – an `Iterator` over `Students` is `Iterator<Student>`

Iterator methods

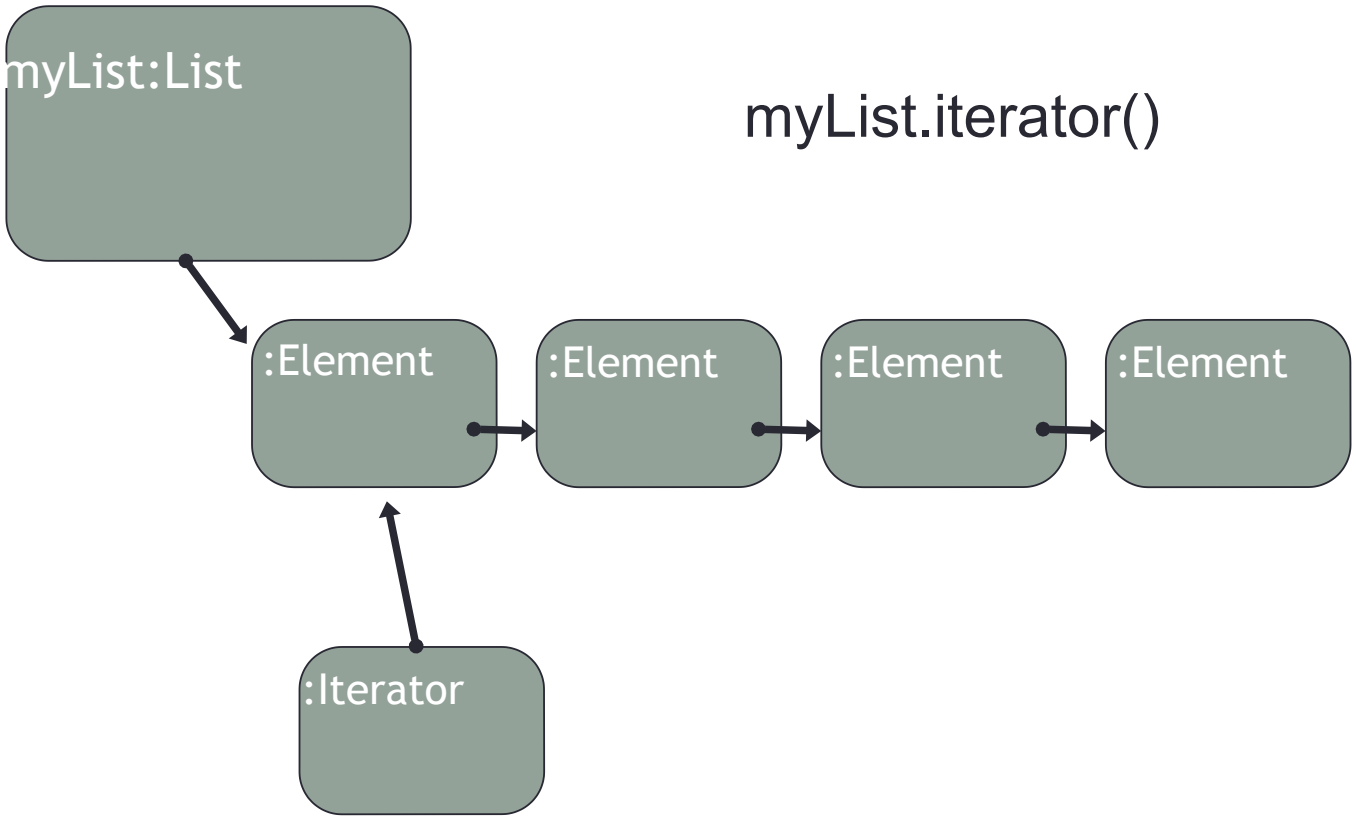
- An `Iterator` has 3 methods:
  - `boolean hasNext()` – is there another item to process?
  - `E next()` – get the next item (where `E` represents the type of item we're getting)
  - `void remove()` – remove the last item we got, from the collection.

Object diagram of iteration

- Suppose we have a list of some sort, called `myList` ...
- `myList.iterator()` will give us an `Iterator` object



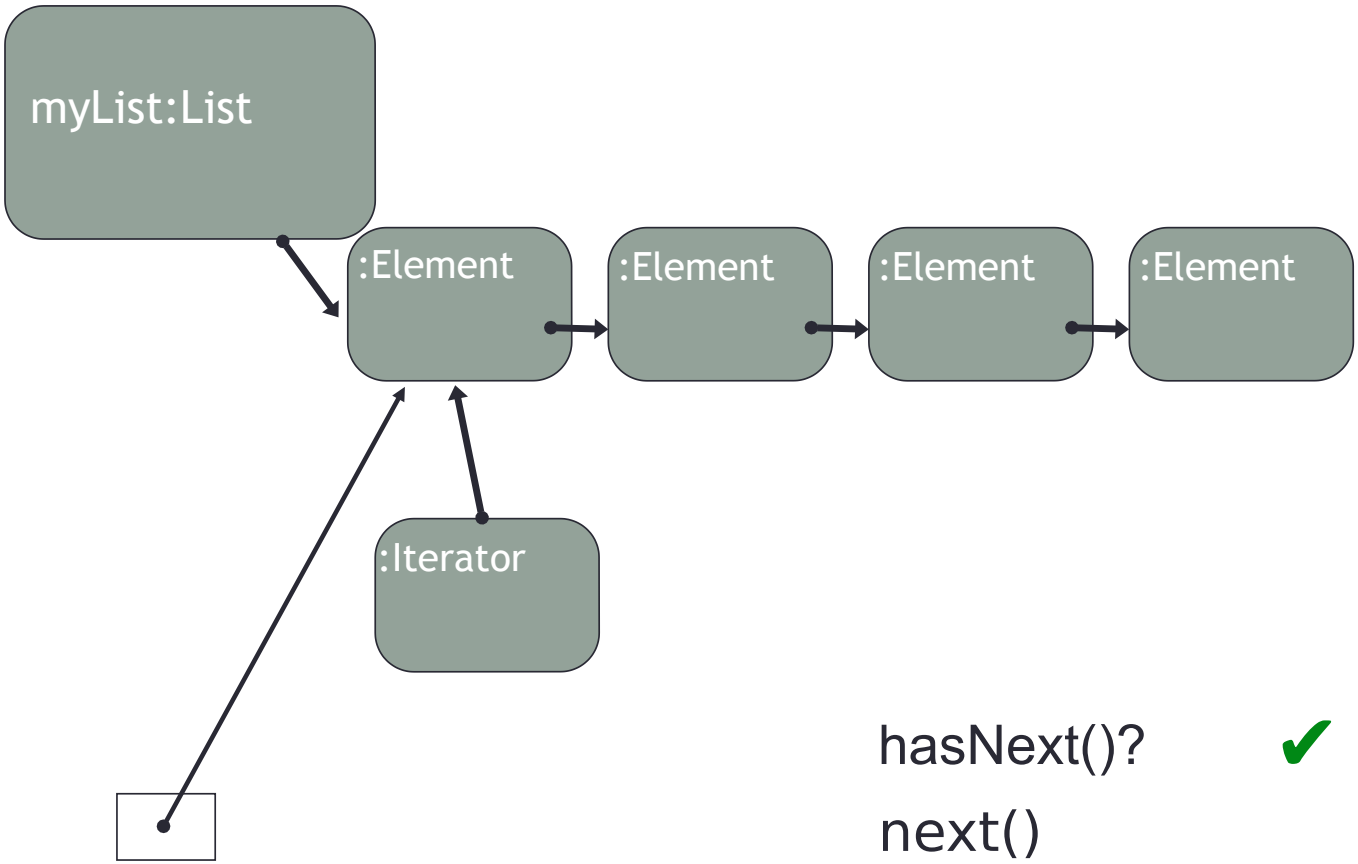
Object diagram cont'd



- If we call `hasNext()`, the `Iterator` will return `true`, confirming there is a first item we can retrieve
- If we call `next()`, we'll get the first item . . .

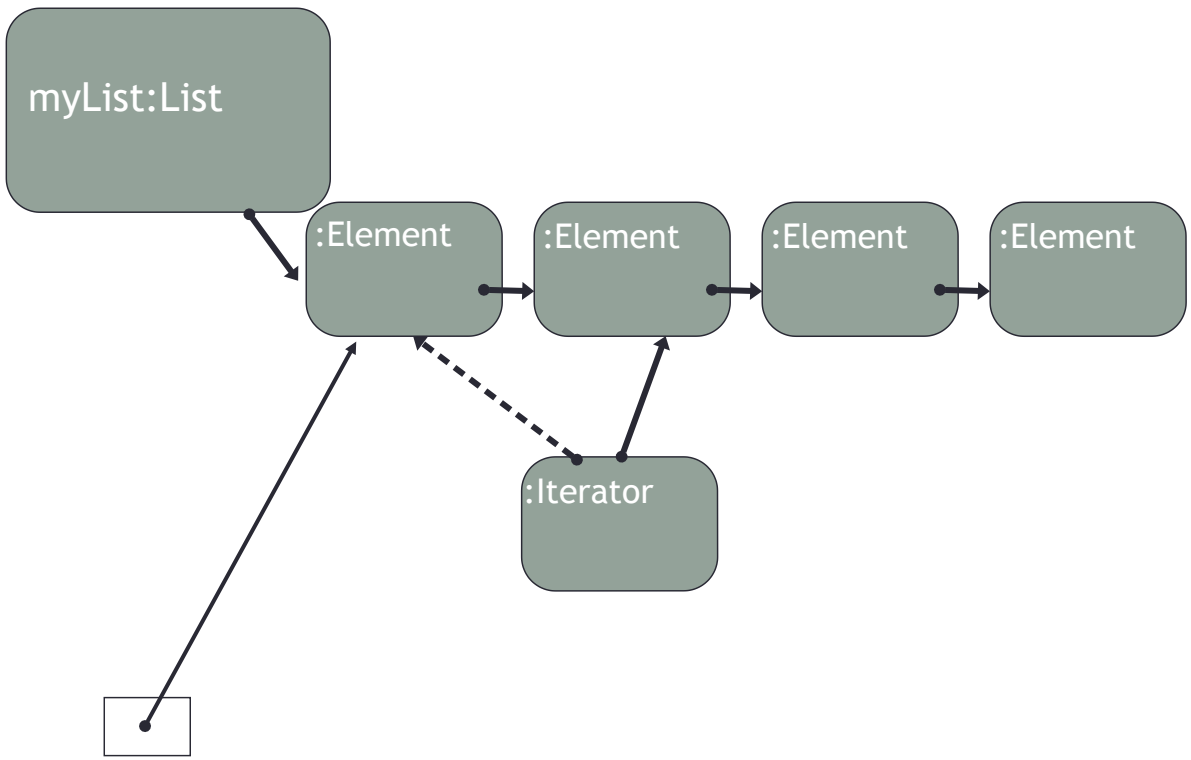
Object diagram cont'd

- Calling `next()`:



Object diagram cont'd

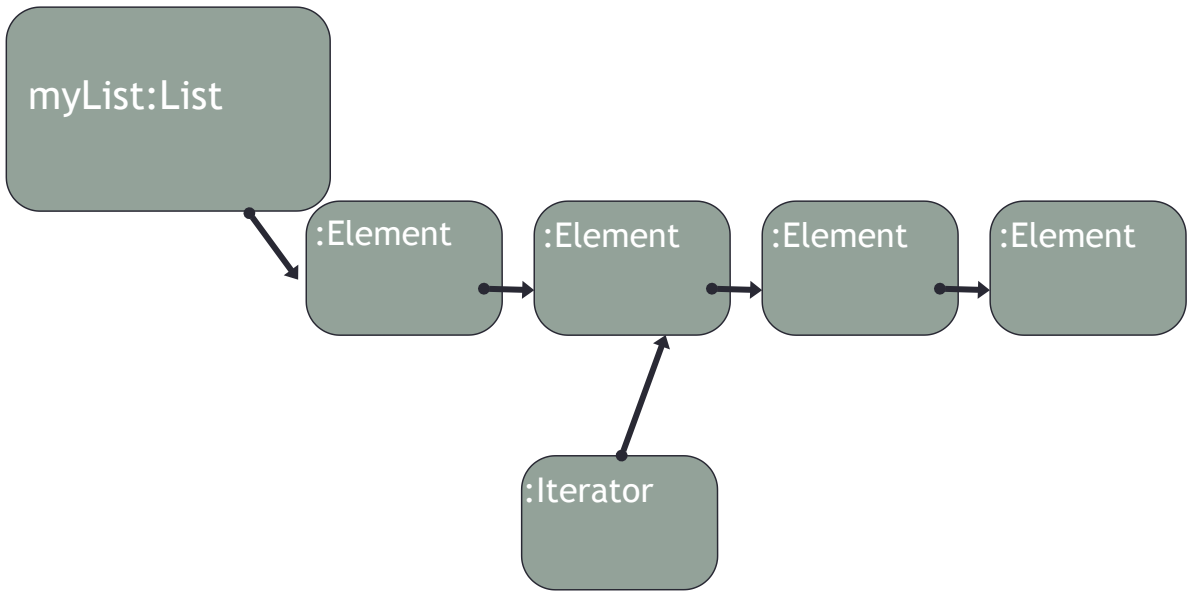
- And as soon as we've *called* `next()`, the `Iterator` will change to point at the next object (if there is one).




```
iterator.next();
```

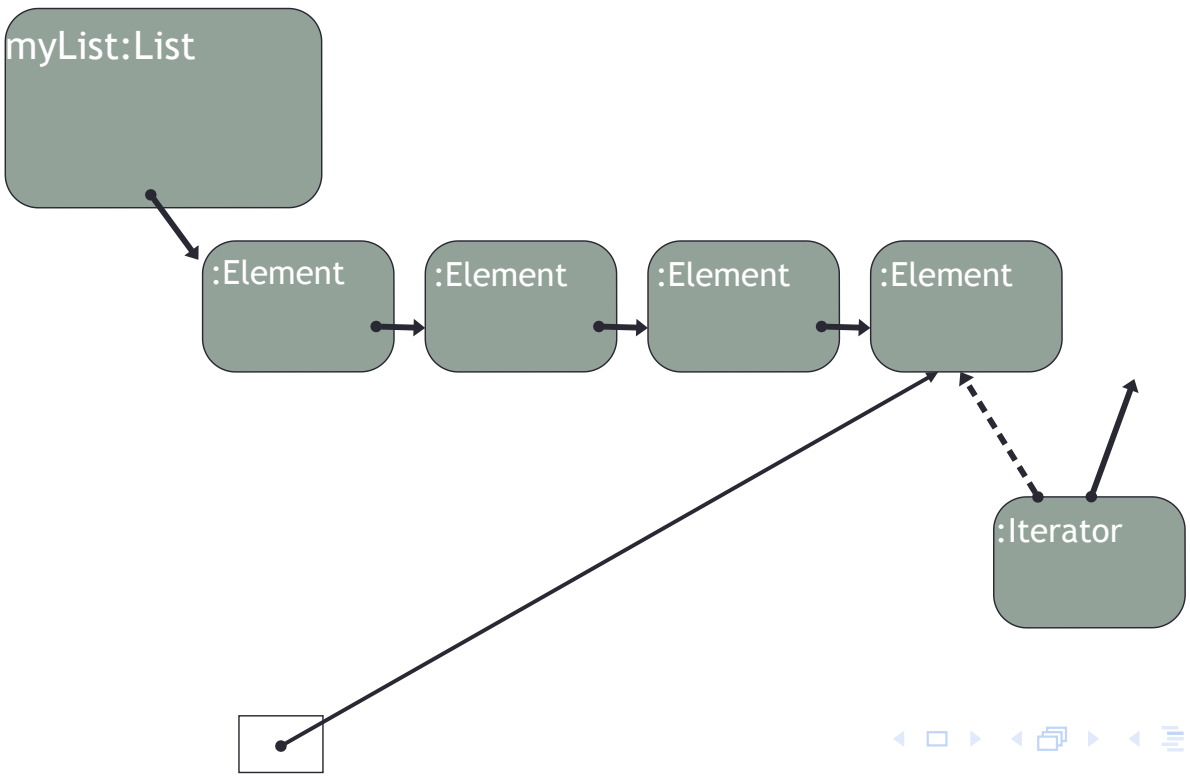
Object diagram cont'd

- And so on – we can call `hasNext()` again to see that there is a next object, and `next()` to retrieve it.



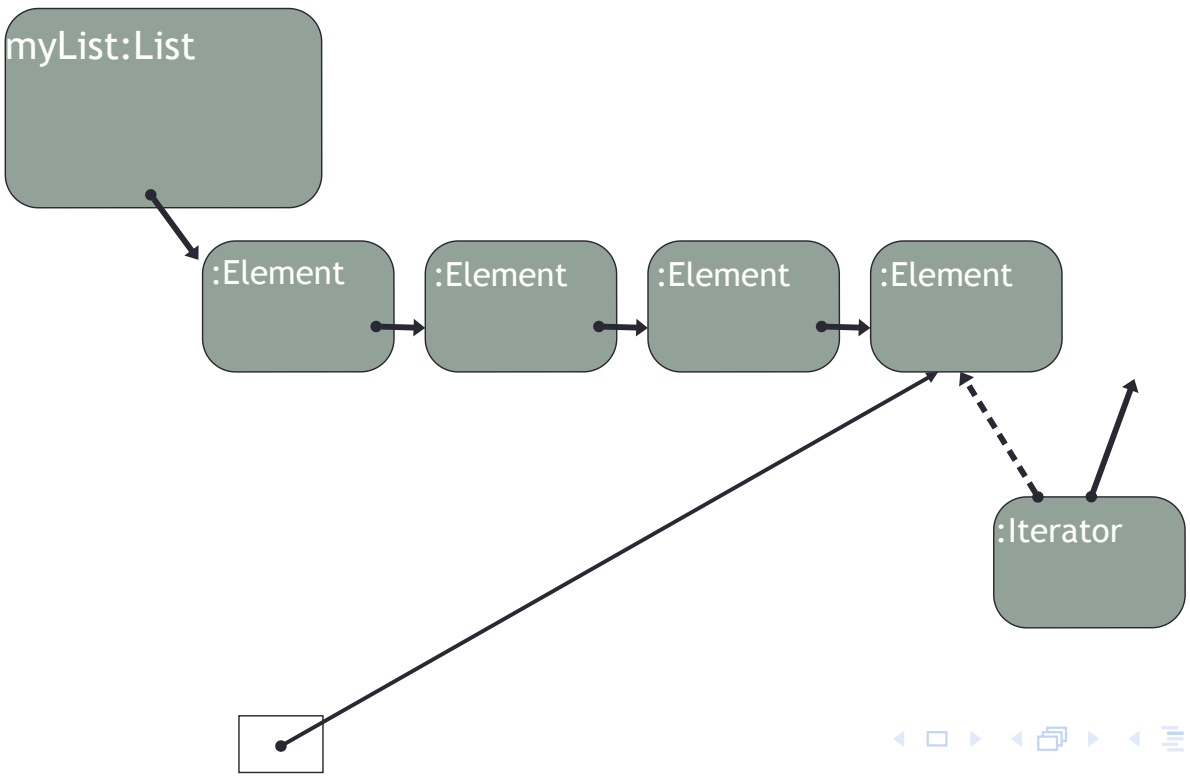
Object diagram cont'd

- And eventually, we'll call `next()` and get a reference to the last object, and the `Iterator` will point . . . *beyond* the last object.



Object diagram cont'd

- And eventually, we'll call `next()` and get a reference to the last object, and the `Iterator` will point . . . *beyond* the last object.
- At that point, if we call `hasNext()`, the `Iterator` will return `false`.



Iterator code

- What does the code for this look like?
- Code for just accessing the first element:

```
Iterator<Element> myIter = myList.iterator();
if ( myIter.hasNext() ) {
  Element elem = myIter.next();
  // ... do something with the Element
}
```

Iteration in a loop

- If we want to use an iterator to access all the items in a collection, we'll need a `while` loop – more on this next week.
- Code to loop over the list:

```
Iterator<Element> myIter = myList.iterator();
while ( myIter.hasNext() ) {
  Element elem = myIter.next();
  // ... do something with the Element
}
```

Iterating over the book journal

- We can use iterators to loop over titles in our book journal.
- The following code prints all titles (which we have done before, using a for-each loop):

```
public void printBookTitles() {
  Iterator<String> iter = bookTitles.iterator();
  while ( iter.hasNext() ) {
    String title = iter.next();
    System.out.println(title);
  }
}
```

Iterating over the book journal (2)

- But we can also safely *remove* items from our collection:

```
/** delete titles that match a search string */
public void deleteTitles(String searchString) {
  Iterator<String> iter = bookTitles.iterator();
  while ( iter.hasNext() ) {
    String title = iter.next();
    if (title.contains(searchString)) {
      iter.remove();
    }
  }
}
```

- If the journal contains multiple books with titles containing the search string – how many would be removed? Just the first one? Or all of them?

Iterating over other sorts of collections

- The only sort of collection we have dealt with so far is the `ArrayList`.
- Our book journal class stores book titles in an `ArrayList`, which means that the collection of books is *ordered* – each book has a position in the list.
- We will see other sorts of collection soon which are *not* ordered – iterators work perfectly well with those, too

Index vs 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.

Exercises

- Suppose we have an `ArrayList` of `Student` objects, called `cits1001`. Each student has a `getMark()` method.
- Identify what sort of loops would be best for performing the following tasks, and write them:
  - print the names and marks of all students
  - print the names and marks of every second student
  - print the names and marks of all students with a mark above 50
  - delete a student with the name "Adam Smith"
- Suppose we want to process our `ArrayList`, and return a *list* of students with marks above 50 – how would we do that?