Lec 22: Iterators

Sam McCauley

April 15, 2026

Admin

- Review session today
    - Come with questions
- Apply to be a TA!!
    - Deadline April 24
- Any questions?

Lockdown Drill Today

- We won’t actually lock down today
- Plan for real lockdown:
    - If possible (depending on location of whatever is happening) we should go to a secure room with a door
    - Otherwise we’d barricade in here the best we can

Wrapping Up `List<E>` Interface

Interfaces

- Interfaces help us out when we want a single method to handle different types of objects with the same functionality

- `ArrayList<E>` and `LinkedList<E>` both implement the `List<E>` interface

`List<E>` Interface Methods (partial list)

`add(E item)`	append `item` to the end
`add(int i, E item)`	insert `item` at index `i`
`get(int i)`	return element at index `i`
`set(int i, E item)`	replace element at index `i` with `item`
`remove(int i)`	remove and return element at index `i`
`size()`	return number of elements
`isEmpty()`	return `true` if the list has no elements
`contains(Object o)`	return `true` if `o` is in the list
`indexOf(Object o)`	return index of first occurrence of `o`
`clear()`	remove all elements

Interfaces and Inheritance

- A `Stack<E>` inherits from `ArrayList<E>`. Does that mean it also implements `List<E>`?

- Yes! If class B inherits from class A, then class B implements every method that class A implements

    - Makes sense—class B must have all of these methods implemented, since it has all of A's methods

Interfaces and Inheritance

- Interfaces can `extend` (inherit from) other interfaces
- Remember: an interface is just a list of required methods
- Works like you'd expect: "copies over" the other interface's methods
- To implement a subinterface, a class must implement every method in the subinterface, and all methods in the superinterface

Implementing Multiple Interfaces

- A class may implement multiple interfaces
    - It just means that it meets the requirement (has all required methods) for all of the listed interfaces
- A class may only extend *at most one* other class. You cannot inherit from multiple classes
- Let's look at how the Java library version of `ArrayList` is declared. It inherits from an abstract class, and implements several interfaces

Printing Positive Numbers in a List

The following code prints all of the positive numbers in any `List<Integer>`:

```
1  public void printPositive(List<Integer> list) {
2      for(int i = 0; i < list.size(); i++) {
3          int item = list.get(i);
4          if(item > 0) {
5              System.out.println(item);
6          }
7      }
8  }
```

What is the problem with this code? (It works! It just can be slow...)

Answer: this code takes $O(n^2)$ time for a linked list!

The Issue with Linked Lists

- Any method that access all items of a linked list using `get()` will take $O(n^2)$ time

- Problem: each time we call `get()` we start over at the beginning of the list

Iterators

What We Really Want

- We want a way to iterate through a list
  - `get()` isn't enough—don't want to start over every time!
- We can do this with *iterators*
- We'll see that iterators have other uses and advantages
  - (After midterm) we'll see that we can iterate over other data structures
  - Allow us to use for-each loops (like python!)
  - Can also write iterators that aren't tied to any data structure

What an Iterator Should Do

```
1 for(int i = 0; i < list.size(); i++) {
2     System.out.println(list.get(i));
3 }
```

- We need a way to get the current item
- We need a way to go to the next item
- We need a way to see if we're done (if no items remain)

Java Iterators

```
1 //assume for now list is of type List<Integer>
2 Iterator<Integer> it = list.iterator(); //iterator for list
3 while(it.hasNext()) {
4     System.out.println(it.next());
5 }
```

Iterator Methods:

- `next()`: return the next item in the iteration
- `hasNext()`: check if the next item exists

List Methods:

- The `iterator()` method returns an iterator for the list

Note that a single method, `next()`, is responsible for both going to the next item and returning it. There's no `peek()` here—make sure you store the element if you want to use it again!

What is an Iterator?

- `Iterator<E>` is an *interface*
- The required methods are `hasNext()` and `next()` (that's it)
  - There is an optional `remove()` method that removes the current item from the underlying list
- Different kinds of lists will have different kinds of iterators.
  - That is to say: each kind of list will have its own *class* of iterators. Each one of these classes will implement the `Iterator` interface.

An Iterator for an `ArrayList`

- Let's make an `ArrayListIterator` class together, and extend `ArrayList` to return it

An Iterator for a `LinkedList`

- Let's make a `LinkedListIterator` class together, and extend `LinkedList` to return it

- How long does `next()` take?

Printing Positive Numbers in a List: With an Iterator

The following code *efficiently* prints all of the positive numbers in any `List<Integer>`:

```
1  public void printPositive(List<Integer> list) {
2      Iterator<Integer> it = list.iterator();
3      while(it.hasNext()) {
4          int item = it.next();
5          if(item > 0) {
6              System.out.println(item);
7          }
8      }
9  }
```

for-each loops

for-each loop

- Shorthand to iterate over any data structure that implements the `iterator()` method

- The `:` is read as “in”.

```
1 public void printPositive(List<Integer> list) {
2     for(int item : list) {
3         if(item > 0) {
4             System.out.println(item);
5         }
6     }
7 }
```

When can we use a for-each loop?

- A for-each loop works for any data structure (in fact, any class) that implements the `Iterable<E>` interface
  - That is to say: must have a method `public Iterator<E> iterator()`
  - The `next()` method's return type must match the type of the for-each variable
- Let's use for-each loop with our `ArrayList` and `LinkedList` data structures

Inheritance with Interfaces Example

- In the Java library, the `List<E>` interface extends `SequencedCollection<E>` which extends `Collection<E>` which extends the `Iterable<E>` interface

- So: every `List<E>` must have a `public Iterator<E> iterator()` method

Other Kinds of Iterators

What is an iterator?

- Anything with `next()` and `hasNext()` is an iterator
- Is most often used to iterate through data structures. But doesn't *have* to be used this way
- In these examples, there is no data structure implementing `Iterable<E>`, so we can't use a for-each loop. (We could make one though.)

Iterating through only some entries of an array

- How can we implement an iterator that only goes through the *positive* entries of an array of integers?

Fibonacci Iterator Example

- How can we implement an iterator that iterates through all Fibonacci numbers?