Iterators

Saikrishna Arcot
(edits by M. Hudachek-Buswell)

March 10, 2017

Iterators

- There are many data structures that exist, such as arrays, array lists and linked lists, various types of trees, hash maps, and more.

Iterators

- There are many data structures that exist, such as arrays, array lists and linked lists, various types of trees, hash maps, and more.
- One common operation that might need to be done is to go through all of the items in the data structure.

Iterators

- There are many data structures that exist, such as arrays, array lists and linked lists, various types of trees, hash maps, and more.
- One common operation that might need to be done is to go through all of the items in the data structure.
- However, with the different types of data structures, you would need to write different code for potentially each data structure.

Iterators

- There are many data structures that exist, such as arrays, array lists and linked lists, various types of trees, hash maps, and more.
- One common operation that might need to be done is to go through all of the items in the data structure.
- However, with the different types of data structures, you would need to write different code for potentially each data structure.
- Instead of doing this, an iterator could be used to go through the data structure.

Iterators

- Iterators are (usually) a special class/object that allows the user to easily traverse a data structure.

Iterators

- Iterators are (usually) a special class/object that allows the user to easily traverse a data structure.
- Iterators abstract away the specific operations that can be done on a specific data structure, and provide a few simple, generic methods.

Iterators

- Iterators are (usually) a special class/object that allows the user to easily traverse a data structure.
- Iterators abstract away the specific operations that can be done on a specific data structure, and provide a few simple, generic methods.
- Iterators usually have methods to get the current item, go to the next item, or go to the previous item.

Iterators

- Iterators are (usually) a special class/object that allows the user to easily traverse a data structure.
- Iterators abstract away the specific operations that can be done on a specific data structure, and provide a few simple, generic methods.
- Iterators usually have methods to get the current item, go to the next item, or go to the previous item.
- In Java, there is an `Iterator` interface in the `java.util` package. This interface has the methods `hasNext()`, `next()`, and `remove()`. (`remove()` does not have to be implemented.)

Using an Iterator

- The data structure itself implements the iterator (usually as an inner class or a private class).

Using an Iterator

- The data structure itself implements the iterator (usually as an inner class or a private class).
- This means that in a linked list implementation in Java, for example, there would be an inner or private class that implements the `Iterator` interface and handles going through each item in the linked list.

Using an Iterator

- The data structure itself implements the iterator (usually as an inner class or a private class).
- This means that in a linked list implementation in Java, for example, there would be an inner or private class that implements the `Iterator` interface and handles going through each item in the linked list.
- In addition, the linked list class itself would implement the `Iterable` interface (in the `java.lang` package), which has just one method: `iterator()`. This method returns an instance of the iterator for that data structure.

Using an Iterator

- The user would then ask for an instance of the iterator, and call the methods in the `Iterator` interface to get each item.

Using an Iterator

- The user would then ask for an instance of the iterator, and call the methods in the `Iterator` interface to get each item.
- Note that changing the data structure in any way (such as adding/removing elements) may break the iterator; some iterators may throw a `ConcurrentModificationException` when the data structure is changed.

Iterators in Java

- The `Iterator` interface in Java has three methods: `hasNext()`, `next()`, and `remove()`.

Iterators in Java

- The `Iterator` interface in Java has three methods: `hasNext()`, `next()`, and `remove()`.
    - `hasNext()` tells you whether calling `next()` will return the next item in the data structure (i.e. if there is at least one more item).

Iterators in Java

- The `Iterator` interface in Java has three methods: `hasNext()`, `next()`, and `remove()`.
  - `hasNext()` tells you whether calling `next()` will return the next item in the data structure (i.e. if there is at least one more item).
  - `next()` returns the next item in the data structure.

Iterators in Java

- The `Iterator` interface in Java has three methods: `hasNext()`, `next()`, and `remove()`.
  - `hasNext()` tells you whether calling `next()` will return the next item in the data structure (i.e. if there is at least one more item).
  - `next()` returns the next item in the data structure.
  - `remove()` removes the most-recently returned item.

Iterators in Java

- The `Iterator` interface in Java has three methods: `hasNext()`, `next()`, and `remove()`.
  - `hasNext()` tells you whether calling `next()` will return the next item in the data structure (i.e. if there is at least one more item).
  - `next()` returns the next item in the data structure.
  - `remove()` removes the most-recently returned item.
- The `Iterable` interface in Java has one method: `iterator()`. This method returns an instance of the `Iterator` interface the data structure implemented.

Iterators in Java

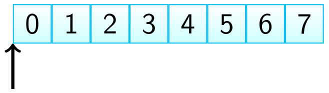
- Not all data structures are required to support `remove()` in the `Iterator` interface; many just throw `UnsupportedOperationException` when implementing that method.

Iterators in Java

- Not all data structures are required to support `remove()` in the `Iterator` interface; many just throw `UnsupportedOperationException` when implementing that method.
- Starting with Java 8, the default implementation of `remove()` throws `UnsupportedOperationException`, so implementing that method in your iterator is no longer required.

Iterators in Java

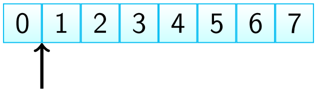
A new iterator starts out in this state, where the iterator is just before the first item in the data structure.



Calling `hasNext()` returns `true`.

Iterators in Java

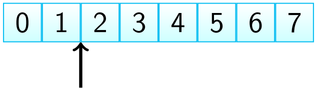
Calling `next()` returns 0, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

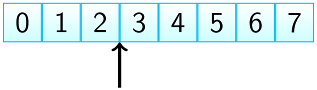
Calling `next()` returns `1`, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

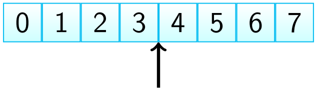
Calling `next()` returns 2, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

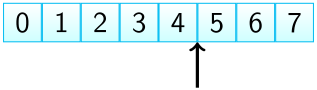
Calling `next()` returns 3, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

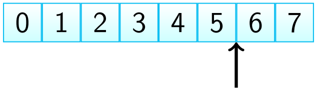
Calling `next()` returns 4, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

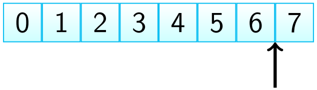
Calling `next()` returns 5, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

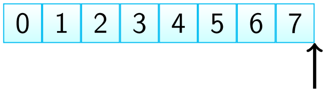
Calling `next()` returns 6, and the iterator moves one item forward.



Calling `hasNext()` returns `true`.

Iterators in Java

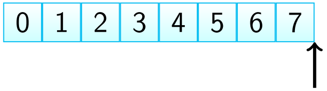
Calling `next()` returns 7, and the iterator moves one item forward.



Calling `hasNext()` returns `false`.

Iterators in Java

Calling `next()` results in `NoSuchElementException` being thrown, and the iterator stays as-is.



Calling `hasNext()` returns `false`.

Iterators in Java

Here's a possible implementation of an iterator for a singly-linked list:

```java
public class LinkedList<T> implements Iterable<T> {

    // Other methods/variables not shown for brevity

    @Override
    public Iterator<T> iterator() {
        return new LinkedListIterator<T>(head);
    }

    private static class LinkedListIterator<T> implements Iterator<T> {
        private Node<T> currentNode;

        public LinkedListIterator(Node<T> startingNode) {
            currentNode = startingNode;
        }

```

Iterators in Java

```
        @Override
        public boolean hasNext() {
            return currentNode != null;
        }

        @Override
        public T next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            T currentData = currentNode.data;
            currentNode = currentNode.next;
            return currentData;
        }
    }
}

```

Note how, in the `next()` method, the current node's data is being returned, and *then* the node is moved forward.

Iterators in Java

Here's how an iterator might be used by the user:

```java
Iterator<Integer> llIterator = linkedList.iterator();
while (llIterator.hasNext()) {
    Integer data = llIterator.next();
    System.out.println("Data: " + data);
}

```

Note how `next()` is called if and only if `hasNext()` is true, to avoid any exceptions from being thrown.

For-Each Loop in Java

- Instead of directly dealing with the iterator object, Java lets you use a for-each loop (or enhanced for-loop) to go through the data structure.

For-Each Loop in Java

- Instead of directly dealing with the iterator object, Java lets you use a for-each loop (or enhanced for-loop) to go through the data structure.
- Given the data structure, Java gets the iterator object and gives back to you each item.

For-Each Loop in Java

- Instead of directly dealing with the iterator object, Java lets you use a for-each loop (or enhanced for-loop) to go through the data structure.
- Given the data structure, Java gets the iterator object and gives back to you each item.
- This may be less error-prone than directly using the iterator object, as there is less code to write.

For-Each Loop in Java

- Instead of directly dealing with the iterator object, Java lets you use a for-each loop (or enhanced for-loop) to go through the data structure.
- Given the data structure, Java gets the iterator object and gives back to you each item.
- This may be less error-prone than directly using the iterator object, as there is less code to write.
- The main requirement for this is that the class implements the `Iterable` interface and returns a proper `Iterator`

For-Each Loop in Java

The same code on the previous slide can be re-written using a for-each loop as follows:

```java
for (Integer data : linkedList) {
    System.out.println("Data: " + data);
}

```

Fancy Iterators

- It is possible for data structures to implement different types of iterators.

Fancy Iterators

- It is possible for data structures to implement different types of iterators.
- For example, in the case of lists, a data structure may implement one iterator that goes from the first to last element of the list and another iterator that goes from the last to first element of the list.

Fancy Iterators

- It is possible for data structures to implement different types of iterators.
- For example, in the case of lists, a data structure may implement one iterator that goes from the first to last element of the list and another iterator that goes from the last to first element of the list.
- In the case of trees (which are covered later), there may be pre-order, in-order, post-order, and level order iterators.

Fancy Iterators

- It is possible for data structures to implement different types of iterators.
- For example, in the case of lists, a data structure may implement one iterator that goes from the first to last element of the list and another iterator that goes from the last to first element of the list.
- In the case of trees (which are covered later), there may be pre-order, in-order, post-order, and level order iterators.
- Note that only one iterator can be used with the for-each loop (as only one iterator can be returned in the `iterator()` method).

Performance

- In most cases, using an iterator will have the same or better time complexity than using other methods.

Performance

- In most cases, using an iterator will have the same or better time complexity than using other methods.
- For example, with an array list, using an iterator will have the same time complexity as calling `get()` for each item.

Performance

- In most cases, using an iterator will have the same or better time complexity than using other methods.
- For example, with an array list, using an iterator will have the same time complexity as calling `get()` for each item.
- However, for a linked list, using an iterator will have better time complexity than calling `get()` for each item, as an iterator will be $O(n)$ ($O(1)$ for getting the next item, repeated n times), whereas calling `get()` will be $O(n^2)$ ($O(n)$ for getting each item, repeated n times).