Iterators

Jonathan Windle

University of East Anglia

J.Windle@uea.ac.uk

May 23, 2017

Overview I

Intro

- Iterate over a collection of objects.
- Uses the `Iterator` interface.

Using iterators - creating instance

- Create an instance like this:

```
Iterator it = list.iterator(); // Raw type iterator
Iterator<Integer> it2 = list.iterator();
Iterator<Double> it3 = arr.iterator();
Iterator<Card> it4 = deck.iterator();
// Can hold arr or list iterator
Iterator<? extends Number> it5 = arr.iterator();
```

Iterating

```
Iterator it = list.iterator(); // Raw type iterator
Iterator<Integer> castEx = list.iterator();
Integer anInt;
// Returns true if any elements are left to iterate over
while (it.hasNext()) {
// Gets the next element and moves the iterator forward.
        anInt = (Integer) it.next(); // Remove cast by using
            generics
        System.out.println("Element = " + anInt);
        it.remove();  // Removes last element returned by
            iterator.
}
```

Iterator Errata

- `List` has a special Iterator. The interface `ListIterator` extends the`Iterator` interface to include extra methods to move backwards in the list:
  - `previous()`
  - `hasPrevious()`
  - `add(E o)`
  - `nextIndex()`
  - `previousIndex()`
  - `set(E o)`
- For-each control structure uses `Iterator` to do so.

```
Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
        System.out.println(it.next());
}
// Is equal to:
for(Integer i : list) // list must be iterable to use in foreach
        System.out.println(i);
```

- Cannot us for-each to structually modify the collection.

Creating own Iterators

- They are defined as private inner classes
- Class should implement the `Iterable` interface, which requires a `iterator()` method.
- When implemented the class can be used in a foreach loop.

```
public class SortedList<T> implements Iterable<T>

        private class SortedIterator implements Iterator<T> {
                int pos = 0;
                @override
                public boolean hasNext() {
                        return pos < size; // Size of array
                }
                @override
                public T next() {
                        return data[pos++];
                }
                @override
                public void remove() {
                        // Optional
                }

        }
}
```

The End