**Chaper 9 Notes – Iterator Pattern**

**General Solution**

1. **Definition:** “The Iterator Pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.” [GoF, p257].

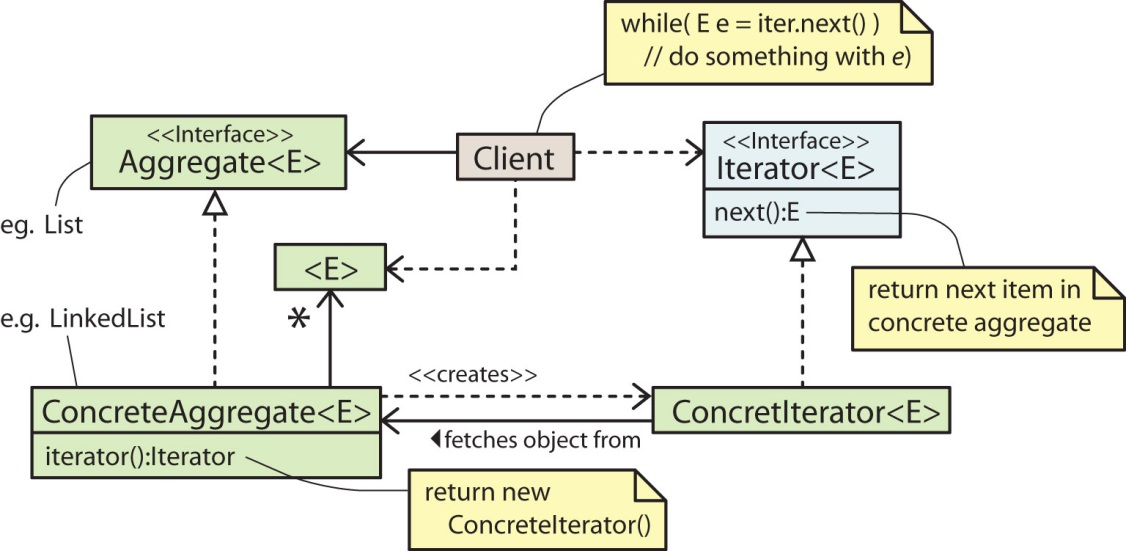
When we use Java’s *enhanced for loop\** (*for-each loop*), the Iterator pattern is behind the scenes. How are elements in an ArrayList stored? A LinkedList? A HashSet? When we use an *Iterator* it doesn’t matter. This allows us to write polymorphic code that can handle different types of collections.

\*The for-each loop in Java is equivalent to using an iterator. The compiler translates:

**for (Employee e : emps)** to:

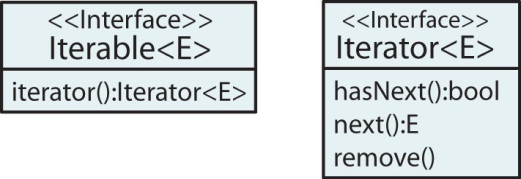
**for (Iterator it = c.iterator(); it.hasNext(); Object e = (Employee)it.next();)**

1. General Class Diagram:

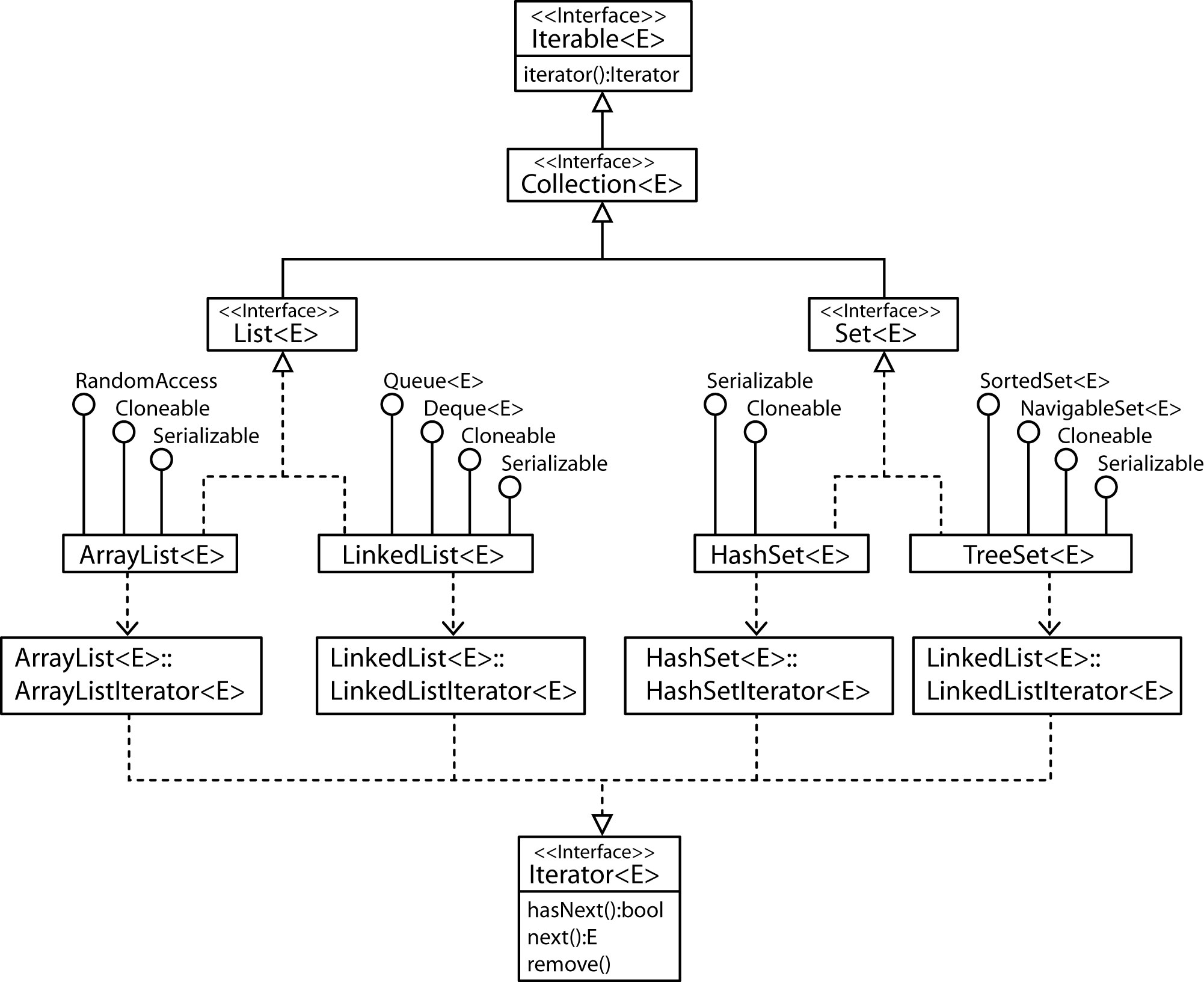


**Iterator Pattern in Java API**

1. Java provides two interfaces to support the Iterator pattern: *Iterable* and *Iterator*.



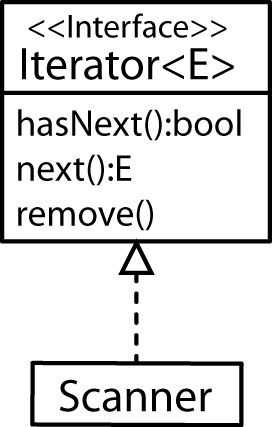
1. All *Collection* classes in the JCF implement the *Iterable* interface. The general design of the *Collection* classes with respect to iteration is shown below. The *iterator* method is a *Factory Method* and the product it produces is an *Iterator*. Thus, in the case of the JCF, the Iterator pattern is implemented with the use of a Factory Method.



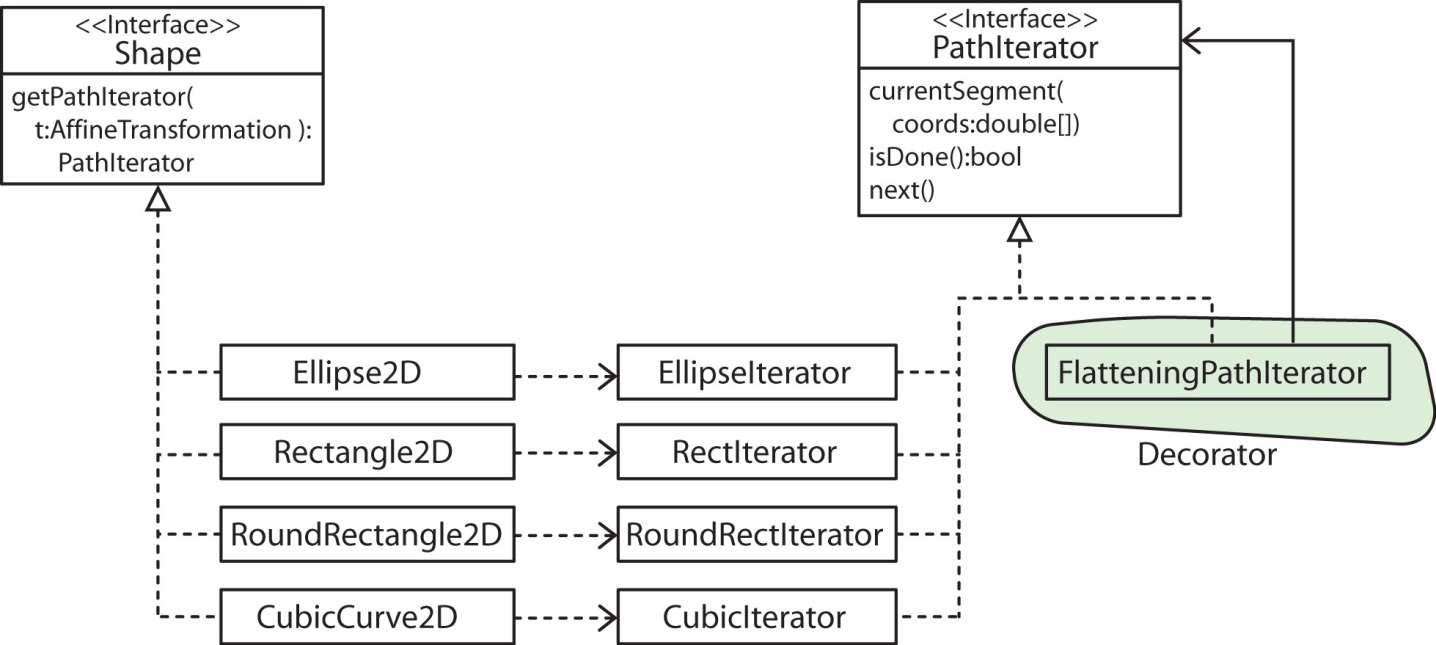
See the code in ArrayList class, lines 773, 780:

http://www.docjar.com/html/api/java/util/ArrayList.java.html

1. The iterator shown above is an *external iterator* which means that a client drives the iteration. Between calls to the *next* method, the external iterator must keep track of the where in the iteration the client is, in other words, the location of the next item to be returned. With simple collections, such as an array backed structure, we simply need to keep an integer that represents the location in the array of the next item. In other data structures, such as a binary tree, external iterators require a good bit more thought.
2. An internal iterator is one where a method must process all the elements in a collection. Thus, the internal iterator controls the iteration and the outside client is unaware of how it does this.
3. A Scanner views an InputStream as a collection of tokens. As a convenience, it implements the Iterator interface.



1. The *java.awt.geom* package contains the *Shape* and *PathIterator* interfaces. The iterators provide a way to iterate over the *Shape* one segment at a time where segments are 1st through 3rd order Bezier curves. The *FlatteningPathIterator* is a Decorator and returns an iterator over the flattened path (straight lines)



1. Example: Suppose you wanted to create an animation along a curved path, say a Car following a curved path. You could create a CubicCurve using Bezier control points. Then, create a FlattenedPathIterator from the CubicCurve and have the car be moved along the flattened path.

**Shape shape = new CubicCurve2D.Float(30, 400, 150, 400, 200, 500, 350, 450);**

**PathIterator pi = shape.getPathIterator(null);**

**PathIterator flatShapeIter = new FlatteningPathIterator( pi, 1 );**

Now, iterate over each iterator (code on website)

**Shape 1 Cubic Curve**

**move to 30.0, 400.0**

**cubic to 150.0, 400.0, 200.0, 500.0, 350.0, 450.0**

**Flattened Shape 1**

**move to 30.0, 400.0**

**line to 51.72119140625, 401.11083984375**

**line to 72.05078125, 404.19921875**

**line to 91.23779296875, 408.89892578125**

**line to 109.53125, 414.84375**

**line to 144.43359375, 429.00390625**

**line to 178.75, 443.75**

**line to 214.47265625, 456.15234375**

**line to 233.48388671875, 460.55908203125**

**line to 253.59375, 463.28125**

**line to 275.05126953125, 463.95263671875**

**line to 298.10546875, 462.20703125**

**line to 323.00537109375, 457.67822265625**

**line to 350.0, 450.0**

1. Others: CharArrayIterator (java.awt.font), ElementIterator, LeafIterator, ColumnIterator, RowIterator (javax.swing)