
Iterators and Collections

CSE 142, Summer 2003
Computer Programming 1

<http://www.cs.washington.edu/education/courses/142/03su/>

Readings and References

- Reading
 - » The discussion in *Intro to Programming and Object-Oriented Design Using Java*, Niño and Hosch is about their own home-grown Lists and Iterators, *not* the ones in `java.util`
- Other References
 - » The Java tutorial on Collections
 - <http://java.sun.com/docs/books/tutorial/collections/index.html>

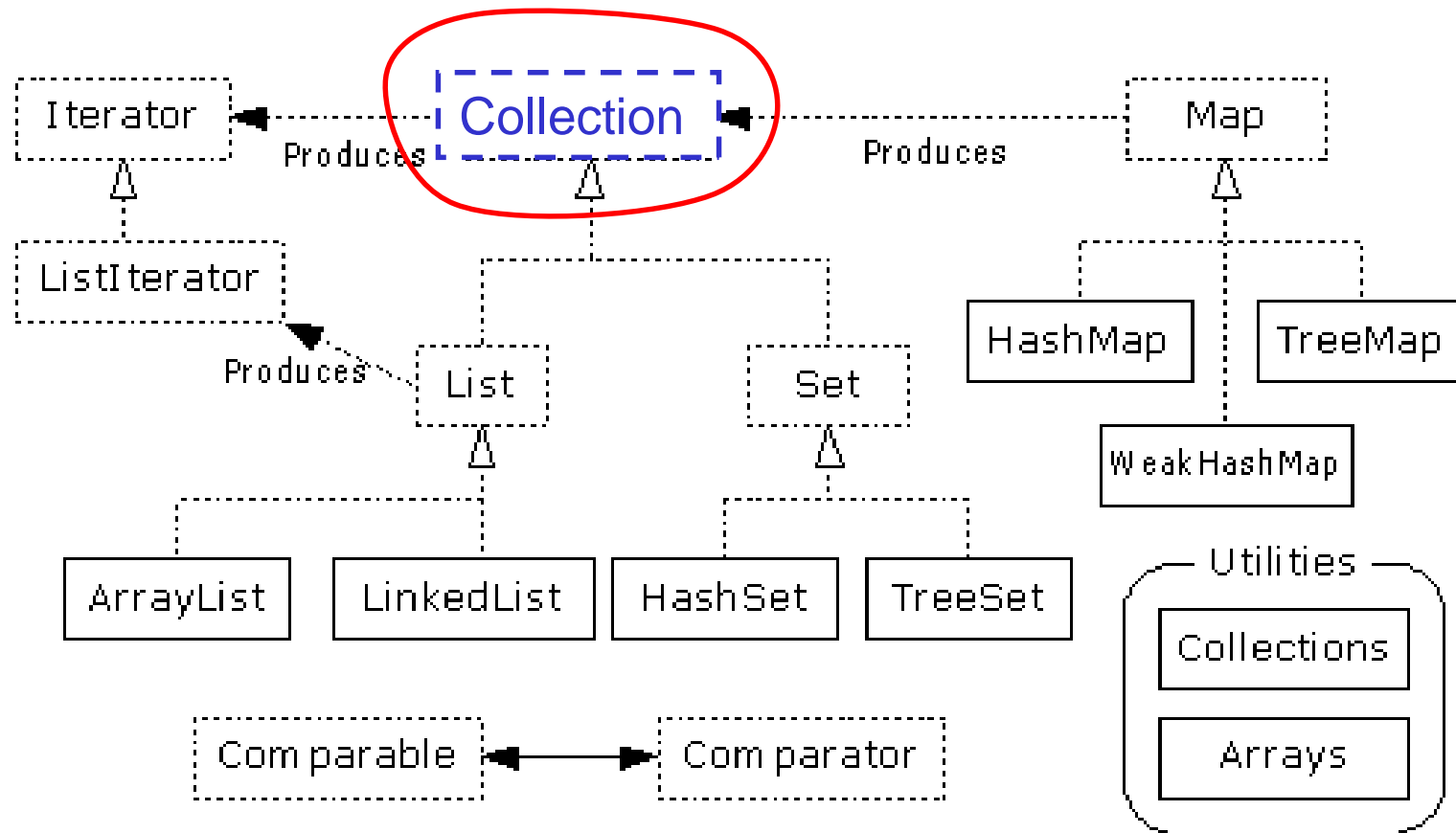
Java fundamentals

- Object oriented programming
 - » classes and objects
 - » interfaces and inheritance
 - » constructors, methods, variables
- The Java language
 - » types, expressions
 - » control flow
 - » exceptions
- Development tools
 - » editors, compiler, Java virtual machine

Java data structures

- Arrays
 - » can hold primitive types directly
- ArrayLists
 - » representative of the many Collection types
- but these are only the beginning
 - » Java provides many well designed interfaces, implementations, and algorithms to help you manage your data

Collection interface



- Interfaces, Implementations, and Algorithms
- From Thinking in Java, page 462

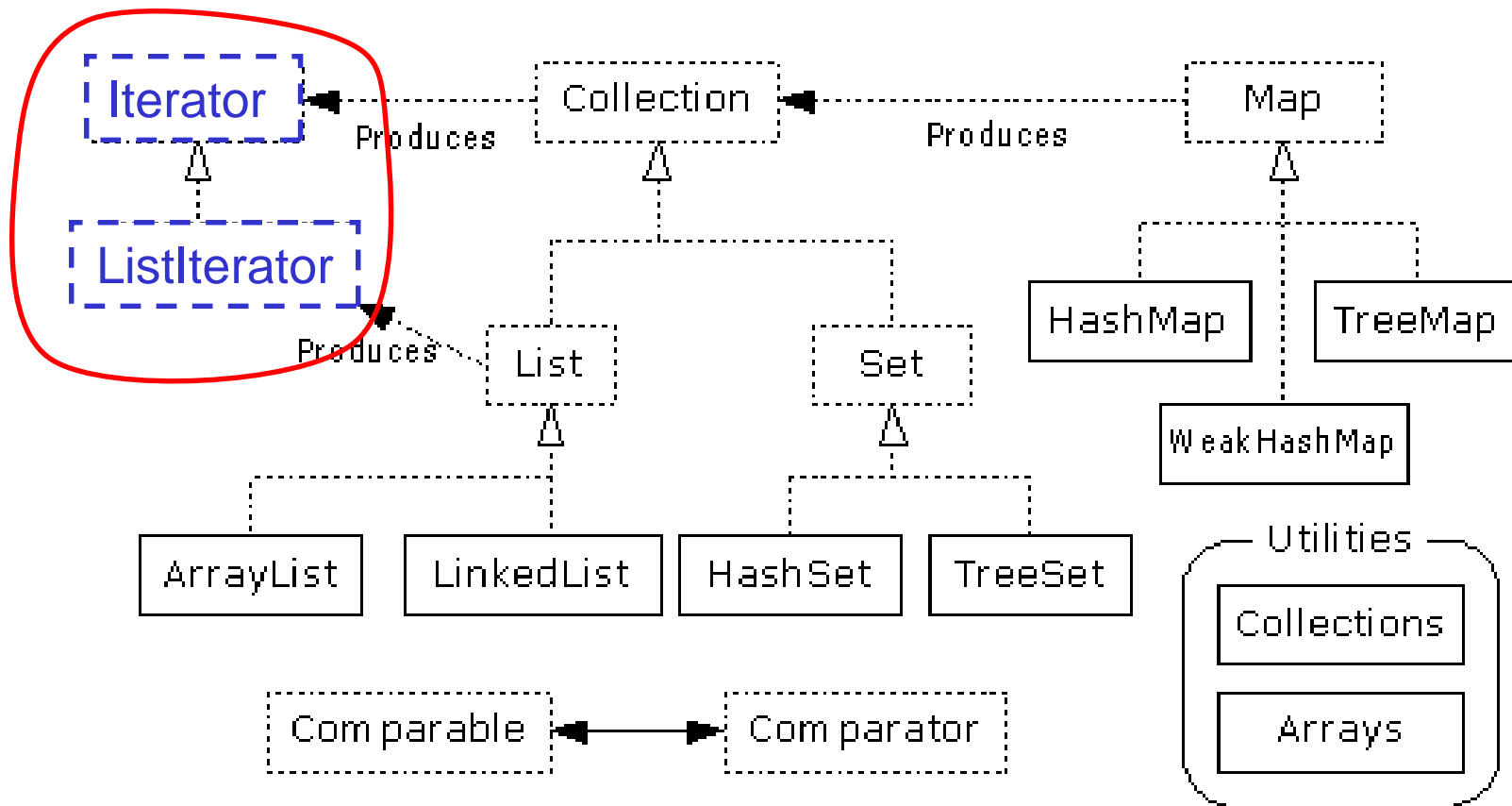
java.util.Collection Interface

- Collection is the root interface in the collection hierarchy
 - » A collection represents a group of objects (the elements of the collection)
 - » Some collections allow duplicate elements and others do not
 - » Some collections are ordered and others are unordered

Collection interface methods

- Defines two fundamental methods
 - » `boolean add(Object o)`
 - » `Iterator iterator()`
- These two methods are enough to define the basic behavior of a collection
- An Iterator lets you step through the elements in the Collection without knowing the index

Iterator interface



Iterator Interface

- Defines fundamental methods
 - » Object `next()`
 - » boolean `hasNext()`
- These methods provide access to the contents of the collection
- An Iterator knows position within collection
- Each call to **`next()`** gets the next element from the collection

Iterator Position with next()

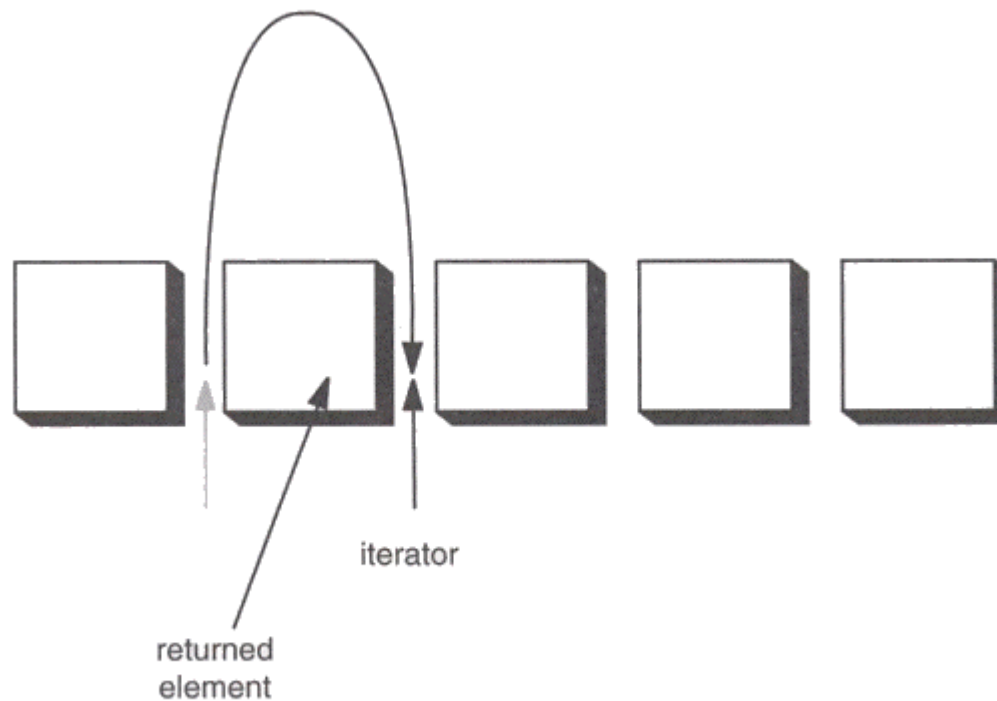
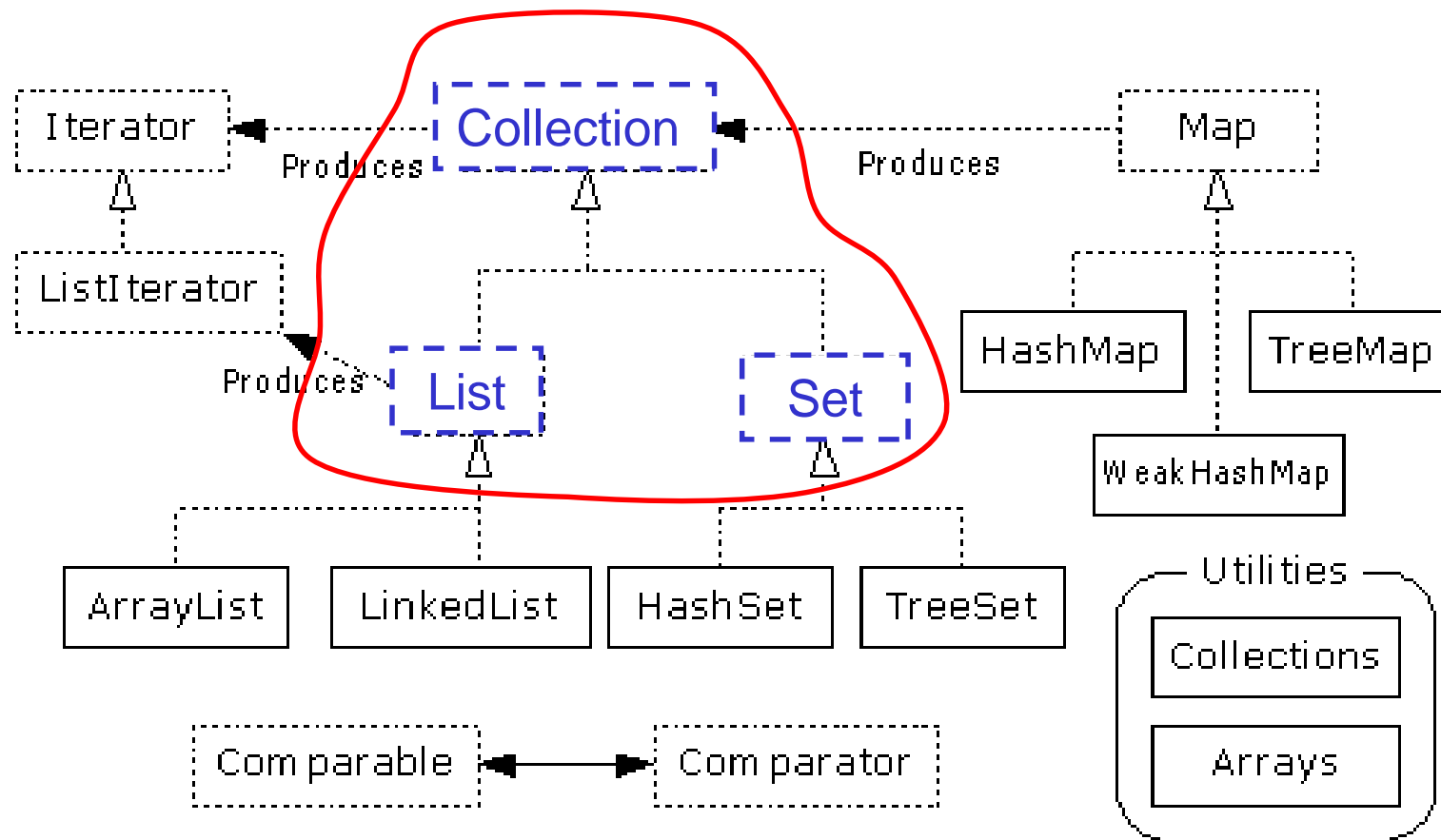


Figure 2-3: Advancing an iterator

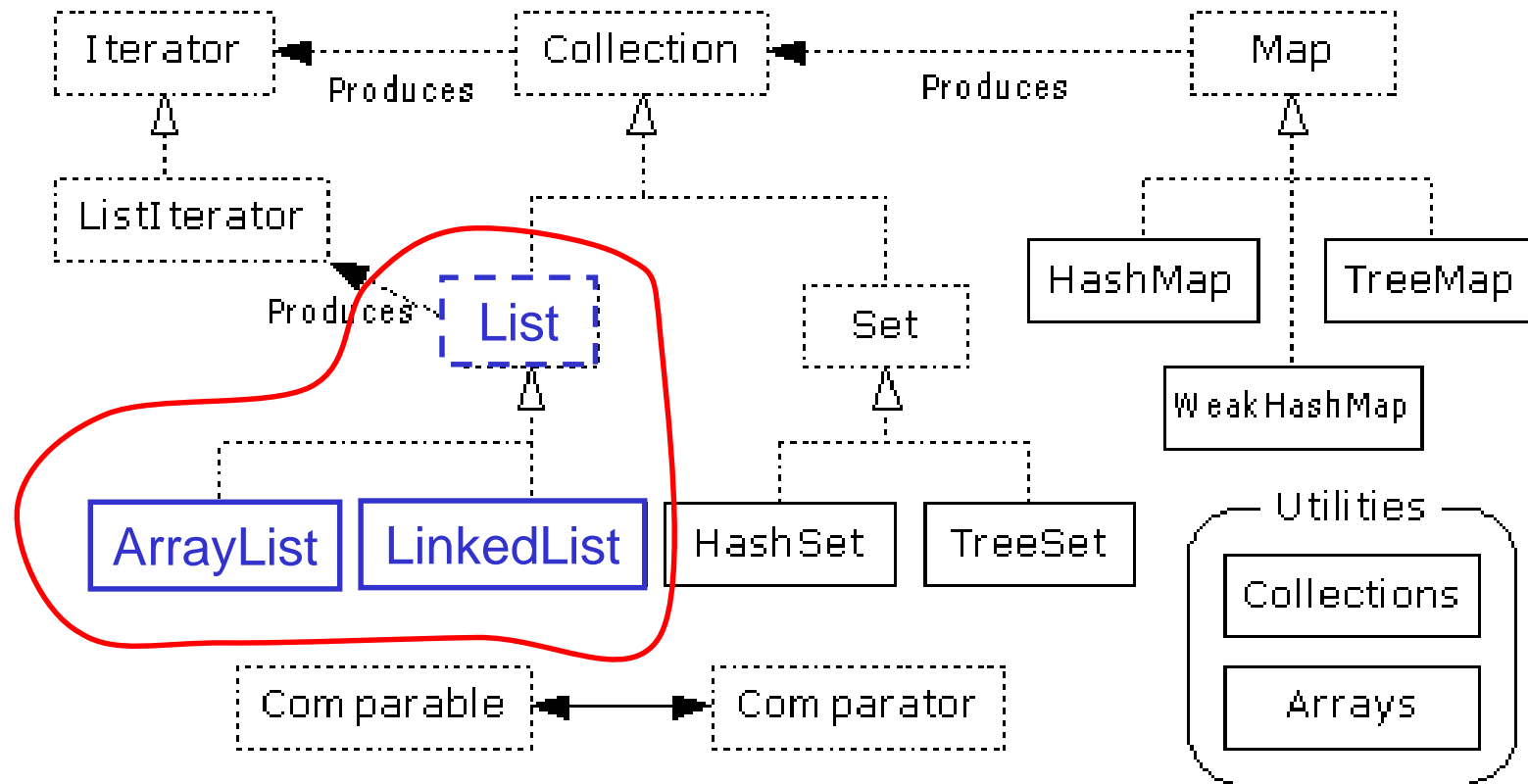
List and Set interfaces



List and Set

- `public interface List extends Collection`
 - » An ordered collection (also known as a *sequence*)
 - » User can store and access elements by their integer index and search for elements in the list
 - » Lists typically allow duplicate elements
- `public interface Set extends Collection`
 - » A collection that contains no duplicate elements and at most one null element
 - » Models the mathematical *set* abstraction

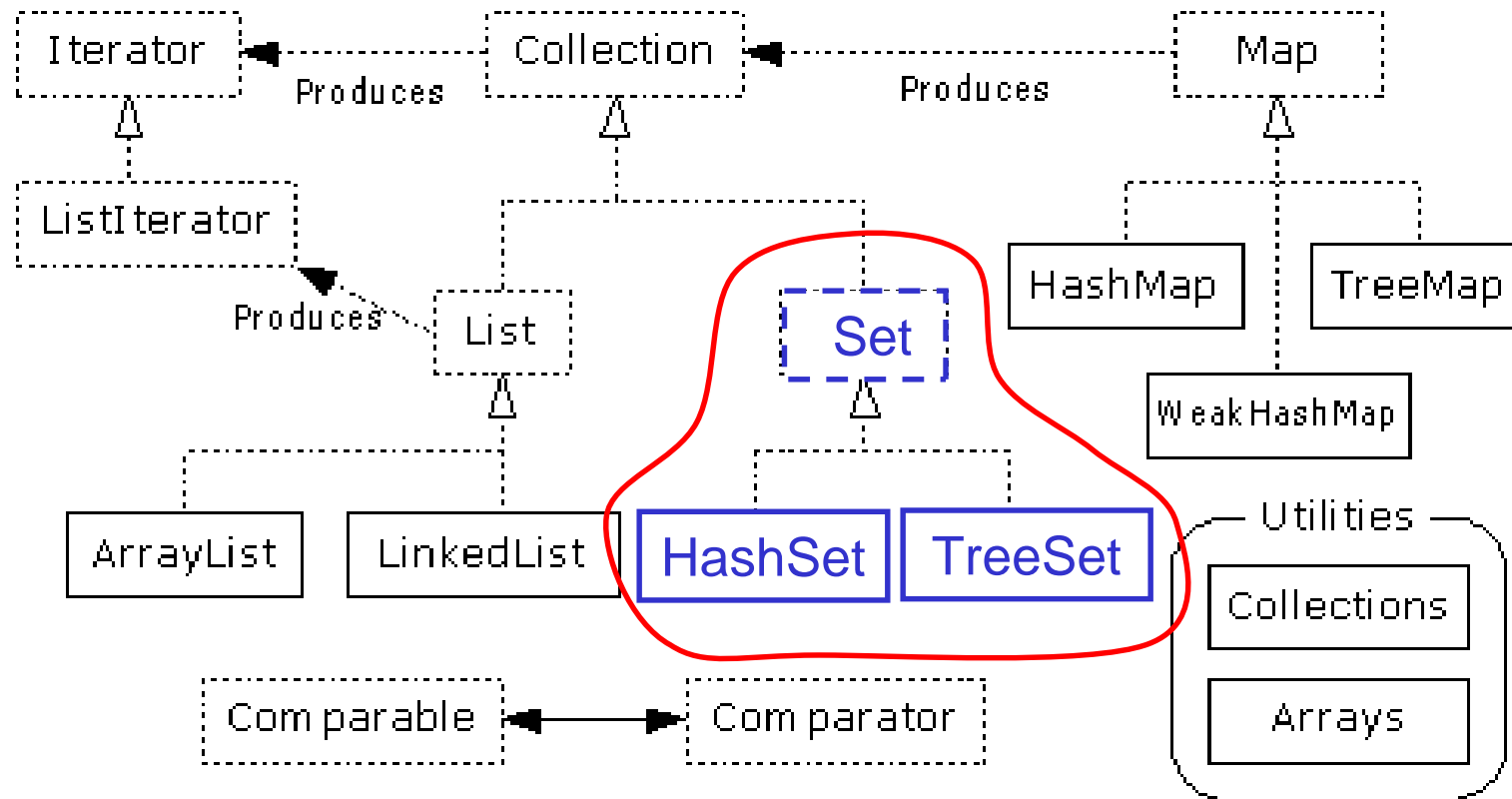
Concrete classes that implement List



ArrayList and LinkedList

- ArrayList
 - » fast access to any element in the List by index
 - » implemented with an array of Objects, ie, Object[]
 - » automatically increases array size when needed
 - » add at the end is fast, but add in the front requires copying the entire array to make room
- LinkedList
 - » fast insert and delete at any point in a list
 - » slow if you want to access elements by index

Concrete classes that implement Set



HashSet and TreeSet

- HashSet
 - » Like all Collections, a HashSet stores objects
 - » This class offers constant time performance for the basic operations (add, remove, contains and size)
 - » No guarantee as to the order of the elements
- TreeSet
 - » Guarantees that the set will be sorted in ascending element order

Example - CollectionManager

```
import java.util.*;
public class CollectionManager {
    public void fillCollection(Collection c) {
        System.out.println("\nFilling "+c.getClass().getName());
        for (int i=4; i >= 0; i--) {
            c.add(i + " * " + i + " = "+i*i); // first entry
            c.add(i + " * " + i + " = "+i*i); // duplicate entry
        }
    }
    public void printCollection(Collection c) {
        Iterator iter = c.iterator();
        while (iter.hasNext()) {
            System.out.println(iter.next());
        }
    }
}
```

Iterators vs Indexed Access

- We can process an ArrayList using get(index)

```
for (int k = 0; k < names.size( ); k++) {  
    process names.get(k);  
}
```

- Tradeoffs
 - » Iterators are more general – work on all collections, even if the collection doesn't support indexed access
 - » Iterators only support traversal of a collection from one element to the next (or previous) – if we want to go in some other arbitrary order, we need indexed access