Java Collections

CSE 403, Spring 2004

Software Engineering

http://www.cs.washington.edu/education/courses/403/04sp/

Readings and References

- "Collections", Java tutorial
- http://java.sun.com/docs/books/tutorial/collections/index.html

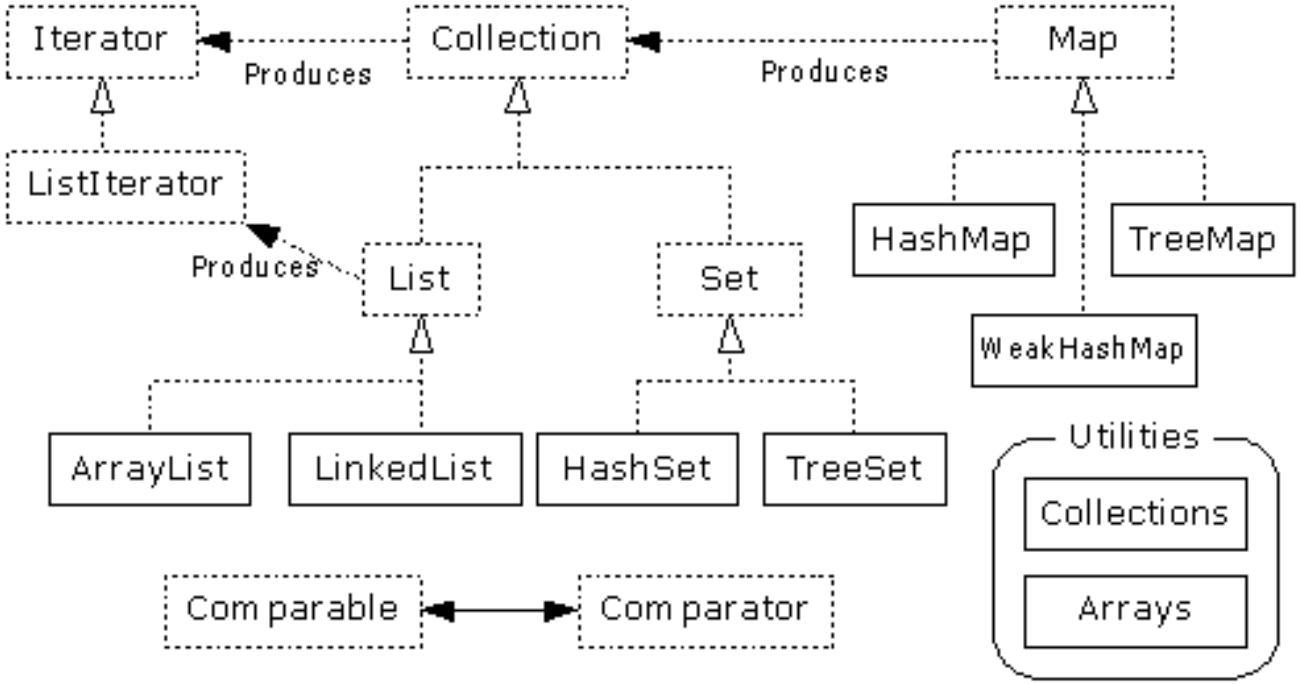
Java 2 Collections

- A collection is an object that groups multiple elements into a single unit
- Very useful
  - » store, retrieve and manipulate data
  - » transmit data from one method to another
  - » data structures and methods written by hotshots in the field
    - Joshua Bloch, who also wrote the Collections tutorial

Collections Framework

- Unified architecture for representing and manipulating collections.
- A collections framework contains three things
  - » Interfaces
  - » Implementations
  - » Algorithms

Collections Framework Diagram



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

Collection Interface

- Defines fundamental methods
  - `int size();`
  - `boolean isEmpty();`
  - `boolean contains(Object element);`
  - `boolean add(Object element);    // Optional`
  - `boolean remove(Object element); // Optional`
  - `Iterator iterator();`
- These methods are enough to define the basic behavior of a collection
- Provides an Iterator to step through the elements in the Collection

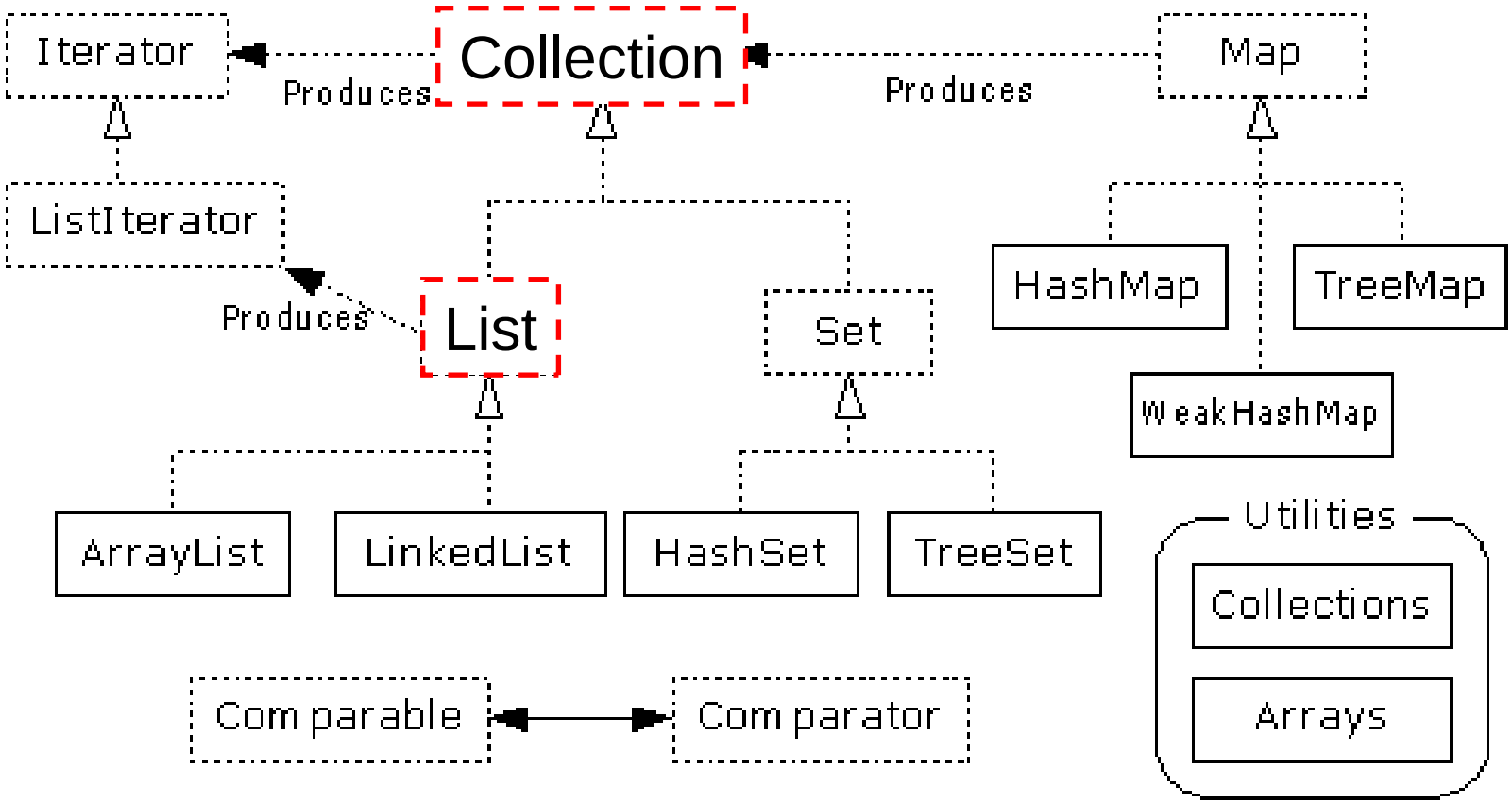
Iterator Interface

- Defines three fundamental methods
  - **`Object next()`**
  - **`boolean hasNext()`**
  - **`void remove()`**
- These three methods provide access to the contents of the collection
- An Iterator knows position within collection
- Each call to next() “reads” an element from the collection
  - Then you can use it or remove it

Example - SimpleCollection

```
public class SimpleCollection  {
   public static void main(String[] args) {
    Collection c;
    c = new ArrayList();
    System.out.println(c.getClass().getName());
    for (int i=1; i <= 10; i++) {
           c.add(i + " * " + i + " = "+i*i);
    }
    Iterator iter = c.iterator();
    while (iter.hasNext())
           System.out.println(iter.next());
   }
}
```

List Interface Context



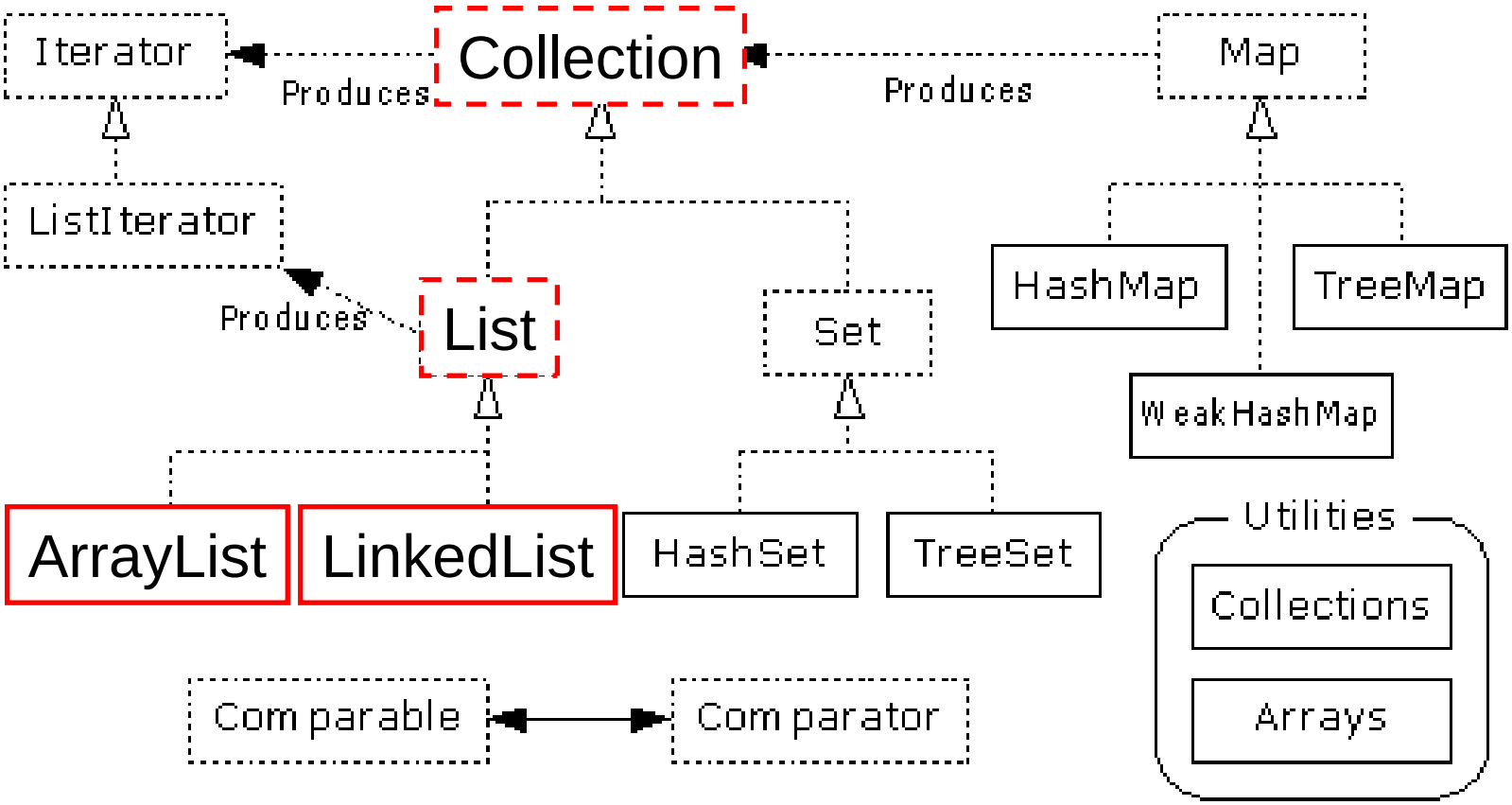
List Interface

- The List interface adds the notion of *order* to a collection
- The user of a list has control over where an element is added in the collection
- Lists typically allow *duplicate* elements
- Provides a ListIterator to step through the elements in the list.

ListIterator Interface

- Extends the Iterator interface
- Defines three fundamental methods
  - » **`void add(Object o)`** - before current position
  - » **`boolean hasPrevious()`**
  - » **`Object previous()`**
- The addition of these three methods defines the basic behavior of an ordered list
- A ListIterator knows position within list

ArrayList and LinkedList Context



List Implementations

- ArrayList
  - » low cost random access
  - » high cost insert and delete
  - » array that resizes if need be
- LinkedList
  - » sequential access
  - » low cost insert and delete
  - » high cost random access

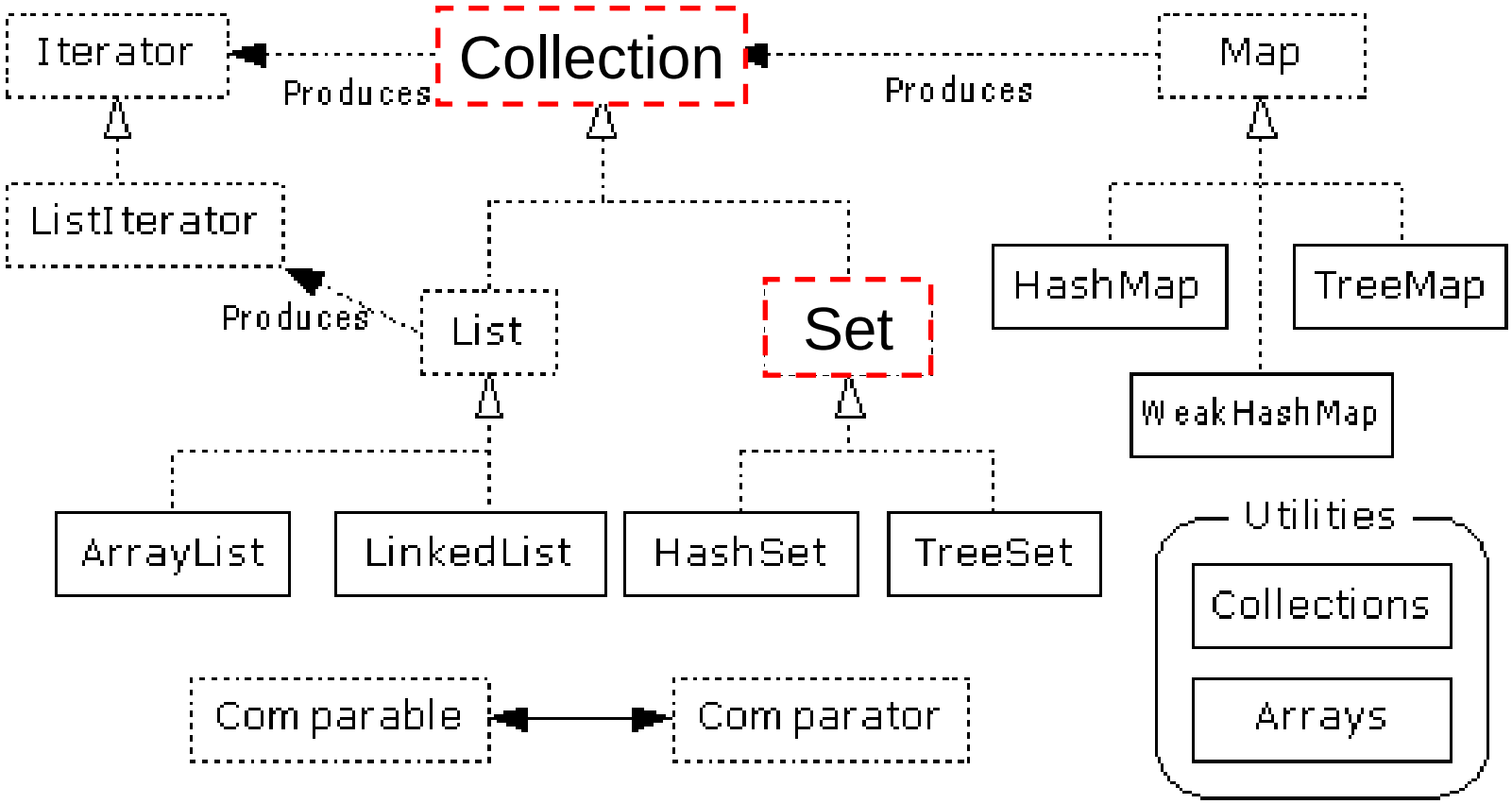
ArrayList methods

- The indexed get and set methods of the List interface are appropriate to use since ArrayLists are backed by an array
  - » **`Object get(int index)`**
  - » **`Object set(int index, Object element)`**
- Indexed add and remove are provided, but can be costly if used frequently
  - » **`void add(int index, Object element)`**
  - » **`Object remove(int index)`**
- May want to resize in one shot if adding many elements
  - » **`void ensureCapacity(int minCapacity)`**

LinkedList methods

- The list is sequential, so access it that way
  - » **`ListIterator listIterator()`**
- ListIterator knows about position
  - » use **`add()`** from ListIterator to add at a position
  - » use **`remove()`** from ListIterator to remove at a position
- LinkedList knows a few things too
  - » **`void addFirst(Object o), void addLast(Object o)`**
  - » **`Object getFirst(), Object getLast()`**
  - » **`Object removeFirst(), Object removeLast()`**

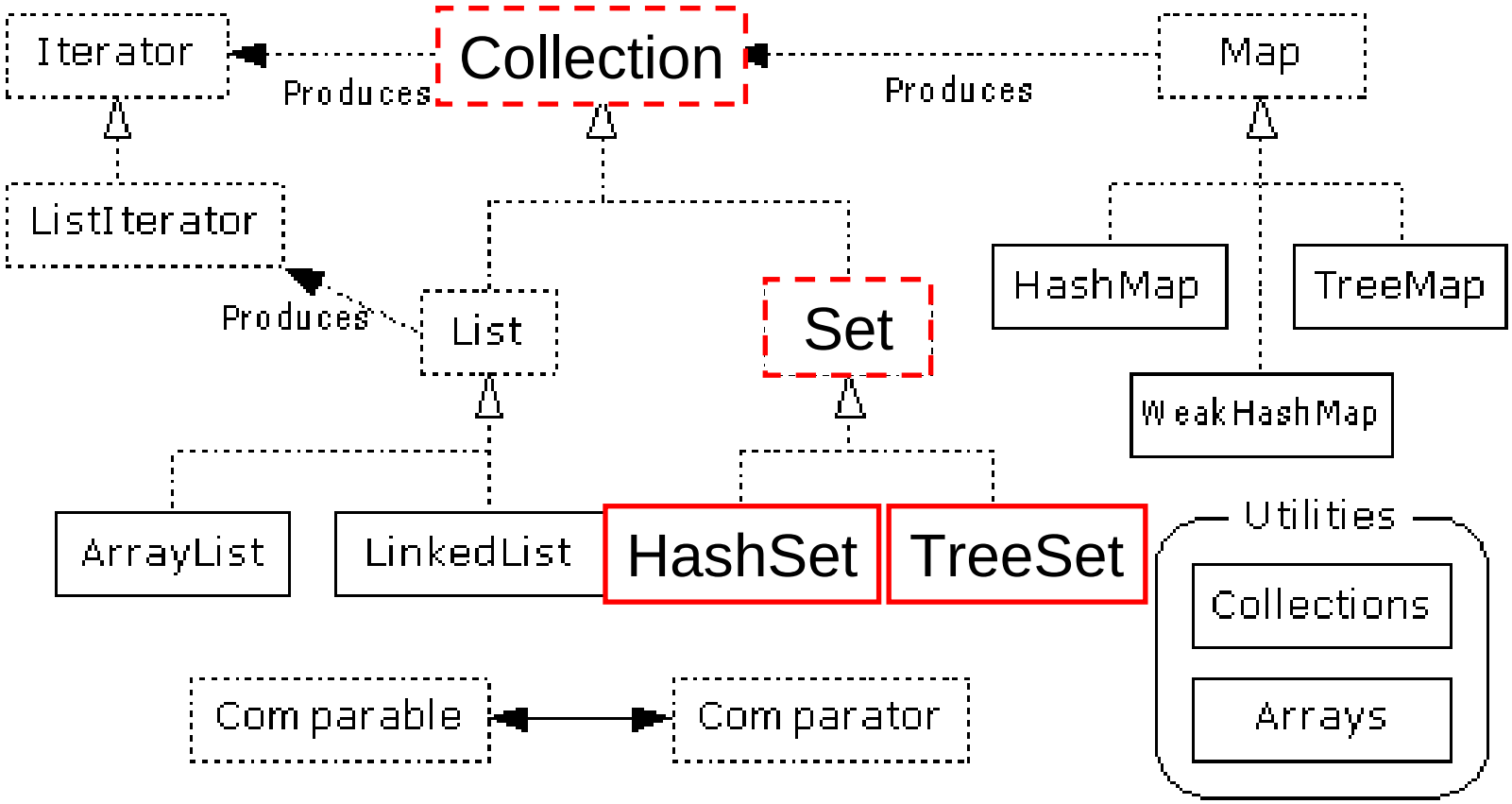
Set Interface Context



Set Interface

- Same methods as Collection
  - » different contract - no duplicate entries
- Defines two fundamental methods
  - » **`boolean add(Object o)`** - reject duplicates
  - » **`Iterator iterator()`**
- Provides an Iterator to step through the elements in the Set
  - » No guaranteed order in the basic Set interface
  - » There is a SortedSet interface that extends Set

HashSet and TreeSet Context



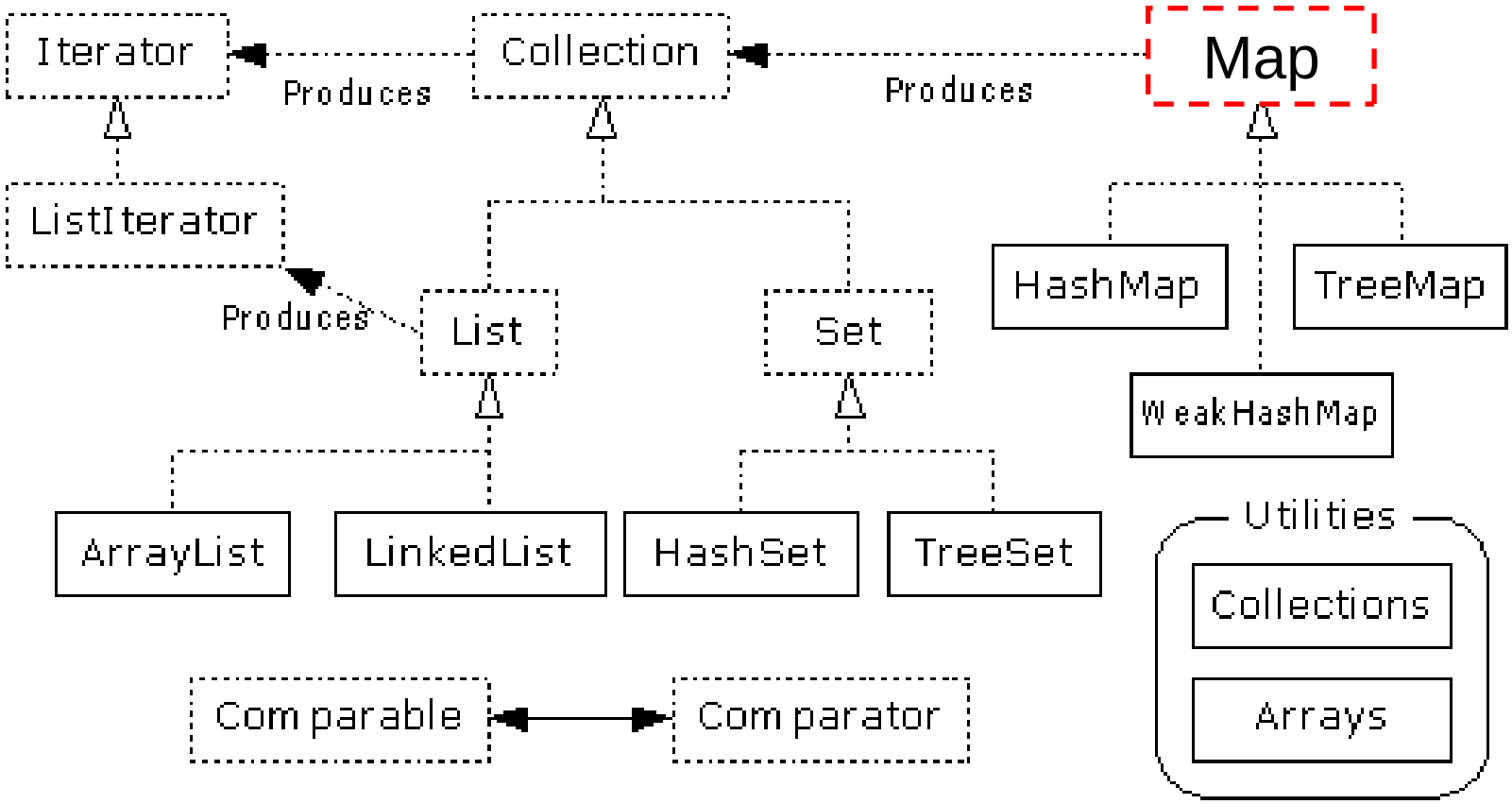
HashSet

- Find and add elements very quickly
  - » uses hashing implementation in HashMap
- Hashing uses an array of linked lists
  - » The **`hashCode()`** is used to index into the array
  - » Then **`equals()`** is used to determine if element is in the (short) list of elements at that index
- No order imposed on elements
- The **`hashCode()`** method and the **`equals()`** method must be compatible
  - » if two objects are equal, they must have the same **`hashCode()`** value

TreeSet

- Elements can be inserted in any order
- The TreeSet stores them in order
  - » Red-Black Trees out of Cormen-Leiserson-Rivest
- An iterator always presents them in order
- Default order is defined by natural order
  - » objects implement the Comparable interface
  - » TreeSet uses **`compareTo(Object o)`** to sort
- Can use a different Comparator
  - » provide Comparator to the TreeSet constructor

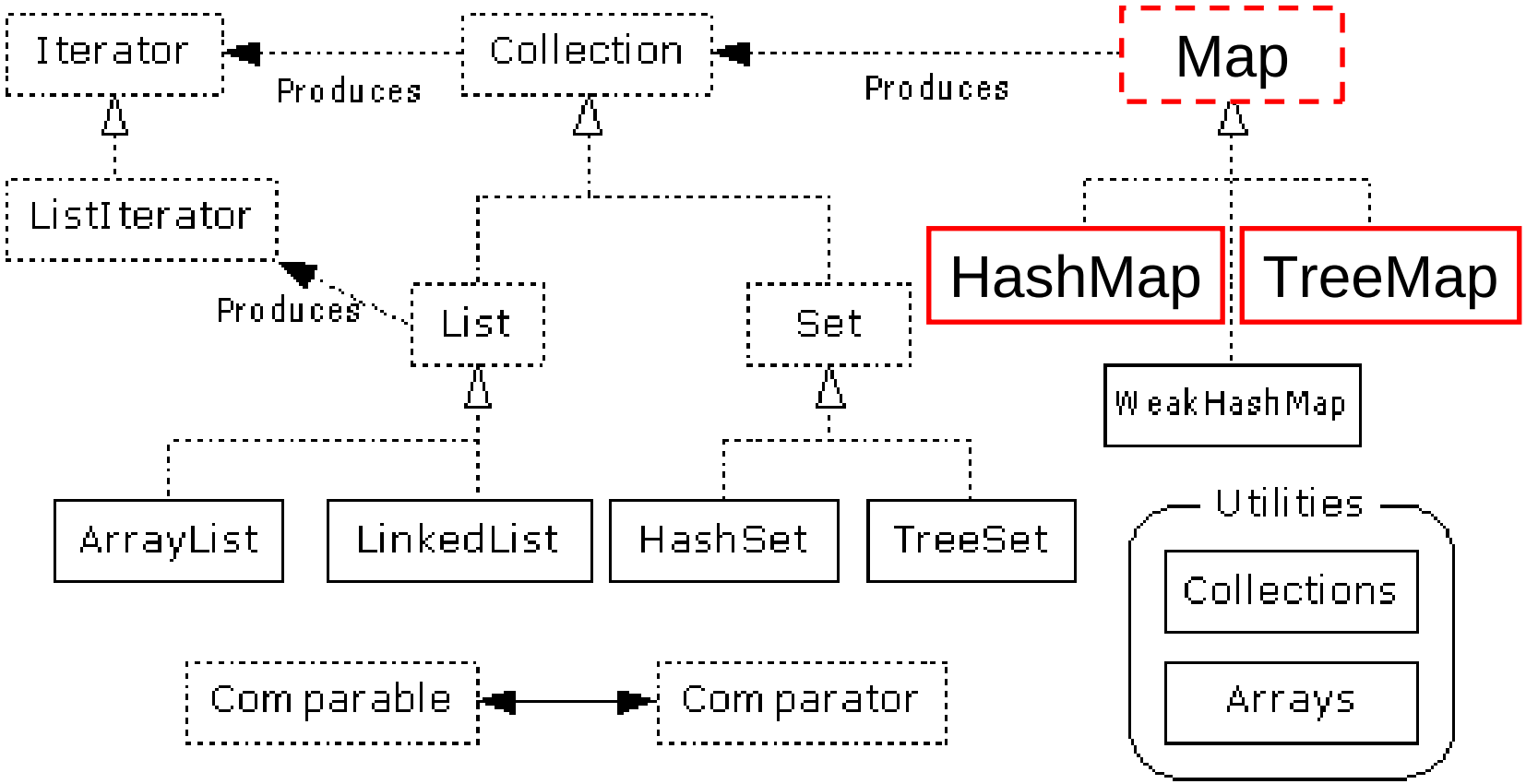
Map Interface Context



Map Interface

- Stores key/value pairs
- Maps from the key to the value
- Keys are unique
  - » a single key only appears once in the Map
  - » a key can map to only one value
- Values do not have to be unique

HashMap and TreeMap Context

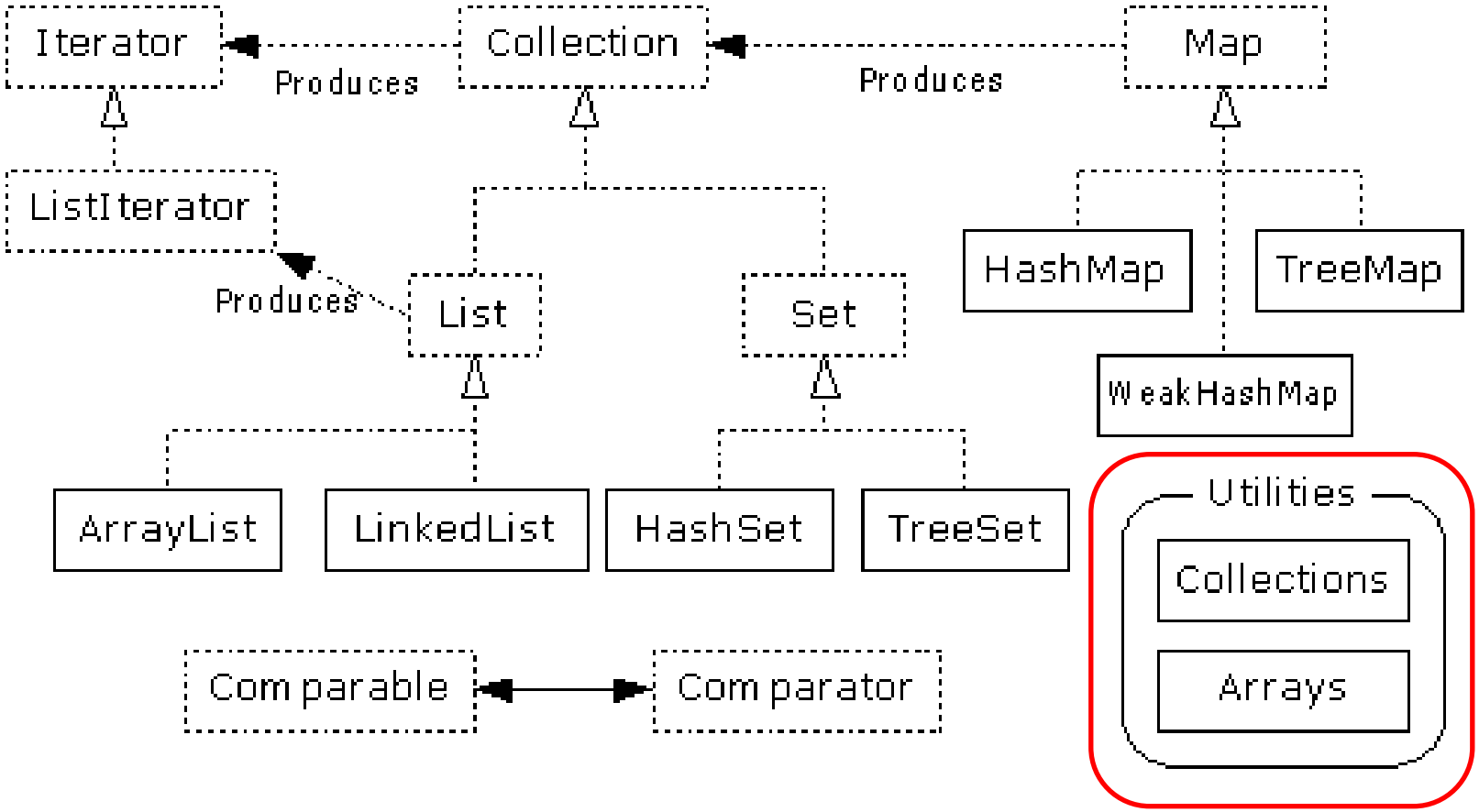


HashMap and TreeMap

- HashMap
  - » The keys are a set - unique, unordered
  - » Fast

- TreeMap
  - » The keys are a set - unique, ordered
  - » Same options for ordering as a TreeSet
    - *Natural order (Comparable, compareTo(Object))*
    - *Special order (Comparator, compare(Object, Object))*

Utilities Context



Utilities

- The Collections class provides a number of static methods for fundamental algorithms
- Most operate on Lists, some on all Collections
  - » Sort, Search, Shuffle
  - » Reverse, fill, copy
  - » Min, max
- Wrappers
  - » synchronized Collections, Lists, Sets, etc
  - » unmodifiable Collections, Lists, Sets, etc

Appendix

Legacy classes

- Still available
- Don’t use for new development
  - » unless you have to, eg, J2ME, J2EE in some cases
- Retrofitted into Collections framework
- Hashtable
  - » use HashMap
- Enumeration
  - » use Collections and Iterators
  - » if needed, can get an Enumeration with Collections.enumeration(Collection c)

More Legacy classes

- Vector
  - » use ArrayList
- Stack
  - » use LinkedList
- BitSet
  - » use ArrayList of boolean, unless you can't stand the thought of the wasted space
- Properties
  - » legacies are sometimes hard to walk away from …
  - » see next few pages

Properties class

- Located in java.util package
- Special case of Hashtable
  - » Keys and values are Strings
  - » Tables can be saved to/loaded from file

System properties

- Java VM maintains set of properties that define system environment
  - » Set when VM is initialized
  - » Includes information about current user, VM version, Java environment, and OS configuration

```
Properties prop = System.getProperties();
Enumeration e = prop.propertyNames();
while (e.hasMoreElements()) {
   String key = (String) e.nextElement();
   System.out.println(key + " value is " +
       prop.getProperty(key));
}
```