Unit 5.2

Collections

UD

Sep 2023

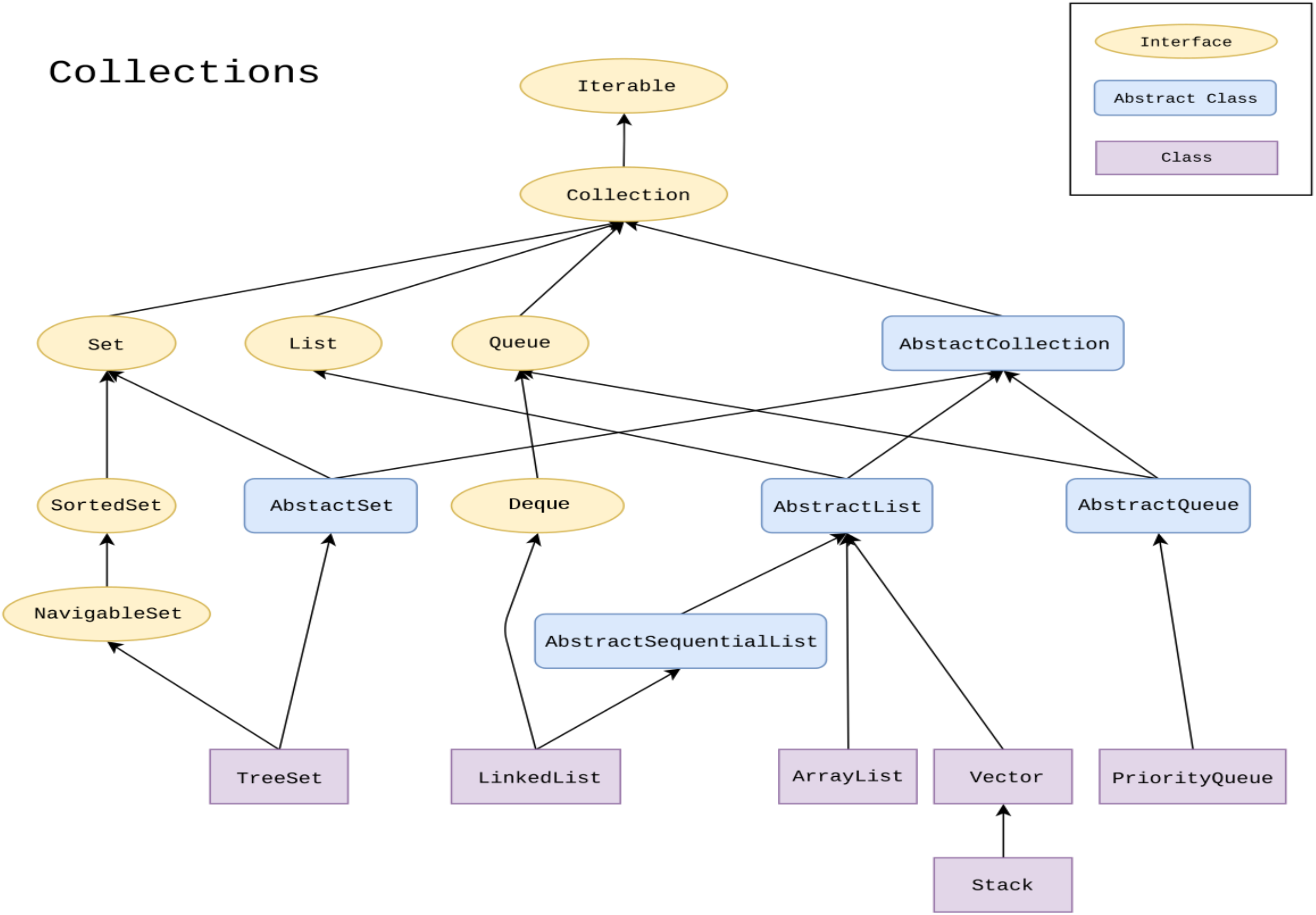
Collections Framework

Unified architecture for representing and manipulating collections.

A collections framework contains three things:

- Interfaces
- Implementations
- Algorithms

Collections Framework Diagram



Collection classes

Collection is a class that fully or partly implements Collection interface. Standard collection classes packaged under java.util:

- AbstractCollection,
  - AbstractList,
    - **AbstractSequentialList,**
      - **LinkedList**,
    - ArrayList,
  - AbstractSet,
    - EnumSet,
    - TreeSet,
    - **HashSet**,
      - **LinkedHashset**,
  - **AbstractQueue**,
    - **PriorityQueue**,
  - ArrayDeque,

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();
```

- Adequate 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()
```

- Above 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

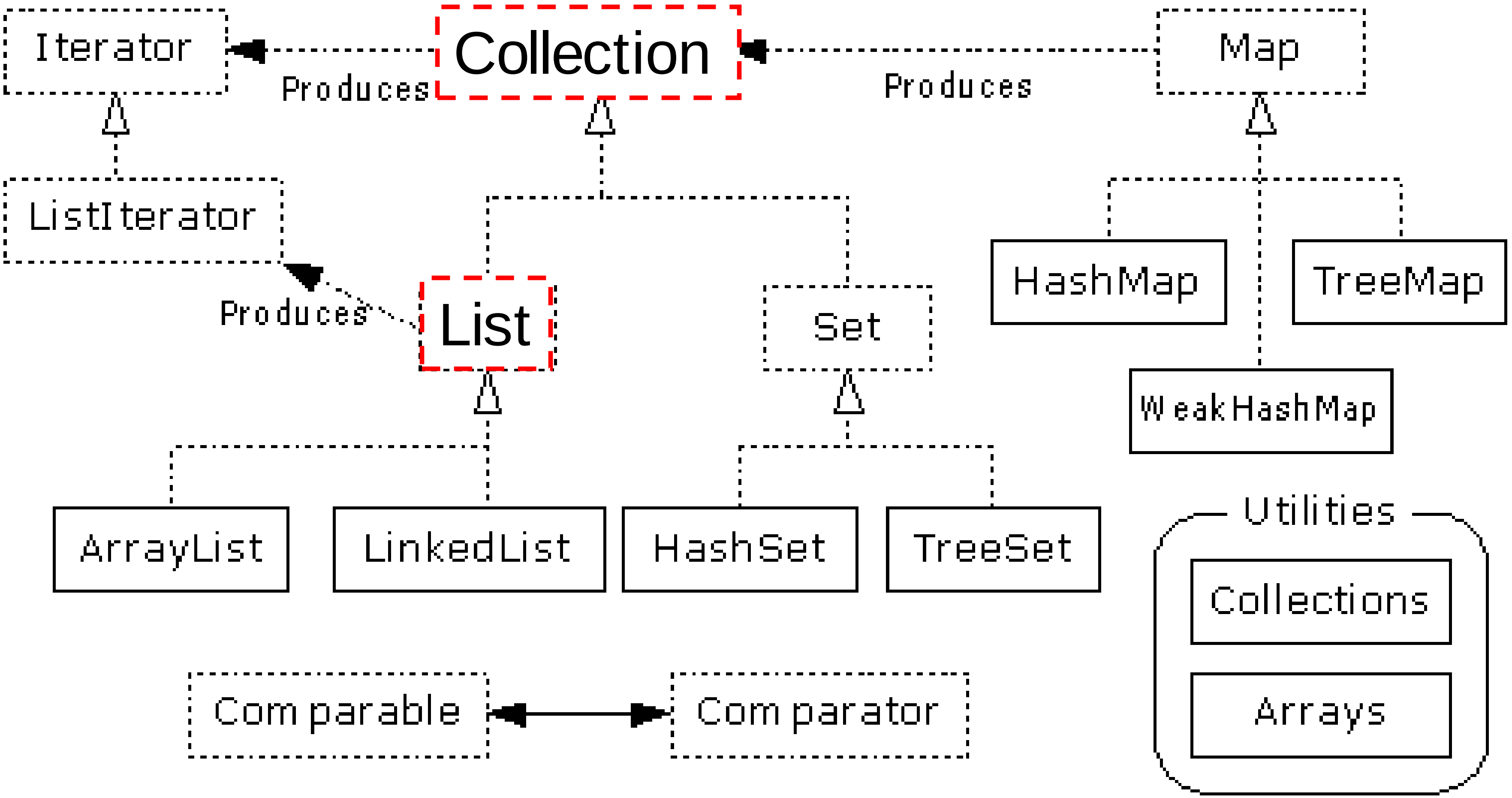
Example – Simple Collection

```
import java.util.*;
public class SimpleCollection {
    public static void main(String[] args) {
        Collection<String> c, l;
        c = new ArrayList<String>();
        l = new LinkedList<String>();
        System.out.println(c.getClass().getName() + " " +
             l.getClass().getName());

        for (int i=1; i <= 10; i++) {
            c.add(i + " * " + i + " = "+i*i);
        }

        Iterator<String> iter = c.iterator();
        while (iter.hasNext())
            System.out.println(iter.next());
    }
}
```

Output:

```
java.util.ArrayList
java.util.LinkedList
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
10 * 10 = 100
```

List Interface Context



ListIterator Interface

Extends the Iterator interface

Defines three fundamental methods

`void add(Object o)` - before current position

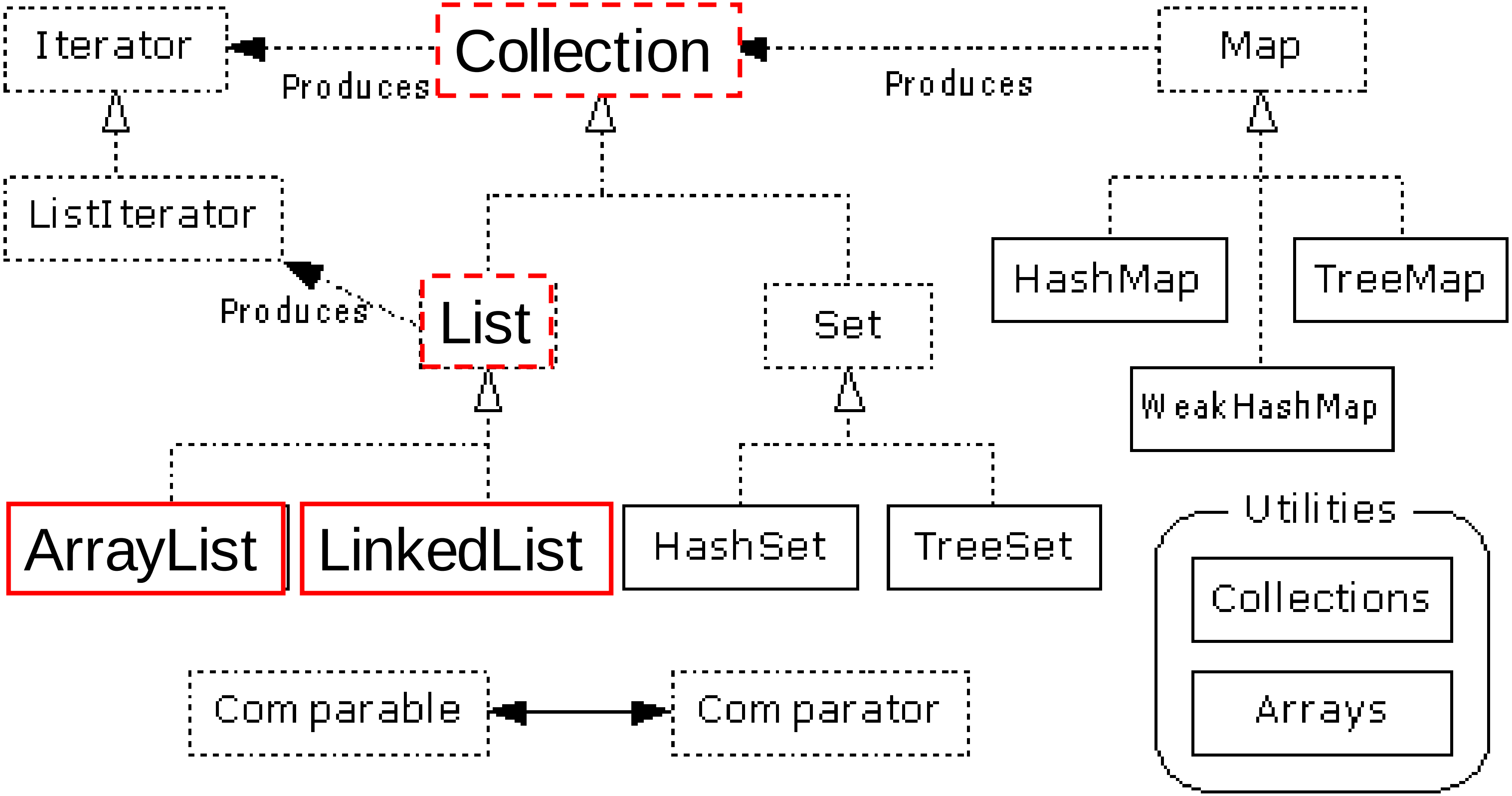
`boolean hasPrevious()`

`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 overview

- Constant time positional access (it’s an array)
- One tuning parameter, the initial capacity

```
public ArrayList(int initialCapacity) {
  super();
  if (initialCapacity < 0)
      throw new IllegalArgumentException(
           "Illegal Capacity: "+initialCapacity);
  this.elementData = new Object[initialCapacity];
}
```

ArrayList methods

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 overview

Stores each element in a node

Each node stores a link to the next and previous nodes

Insertion and removal are inexpensive

>> just update the links in the surrounding nodes

Linear traversal is inexpensive

Random access is expensive

>> Start from beginning or end and traverse each node while counting

LinkedList entries

```
private static class Entry {
       Object element;
       Entry next;
       Entry previous;

       Entry(Object element, Entry next, Entry previous) {
           this.element = element;
           this.next = next;
           this.previous = previous;
       }
}

private Entry header = new Entry(null, null, null);

public LinkedList() {
       header.next = header.previous = header;
}
```

LinkedList methods

- 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()
```