Lecture 4.
The Java Collections Framework

Chapters 6.3-6.4

The Java Collections Framework

- We will consider the Java Collections Framework as a good example of how apply the principles of **object-oriented software engineering** (see Lecture 1) to the design of classical data structures.

The Java Collections Framework

- A coupled set of classes and interfaces that implement commonly reusable collection data structures.
- Designed and developed primarily by Joshua Bloch (currently Chief Java Architect at Google).

What is a Collection?

- An object that groups multiple elements into a single unit.
- Sometimes called a **container**.

What is a Collection Framework?

- A unified architecture for representing and manipulating collections.
- Includes:
  - **Interfaces:** A hierarchy of ADTs.
  - **Implementations**
  - **Algorithms:** The methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces.
    - These algorithms are *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface.

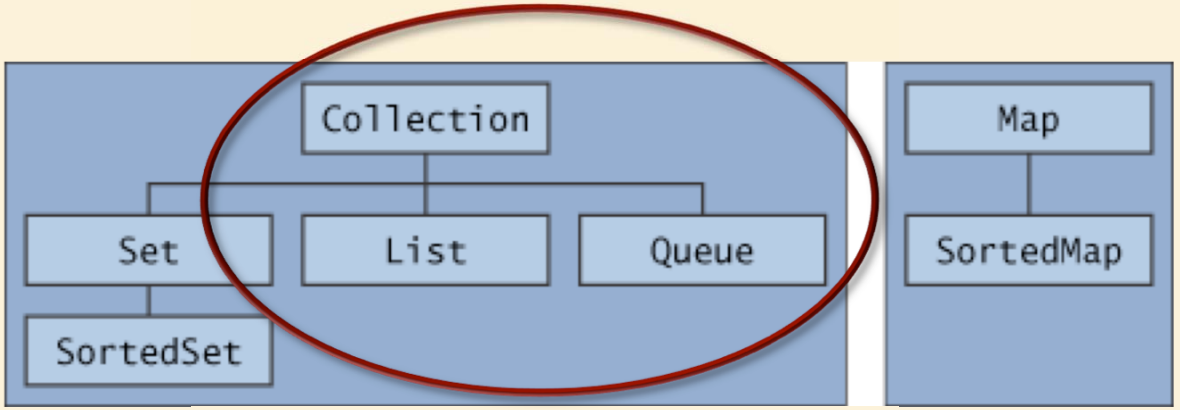
History

- Apart from the Java Collections Framework, the best-known examples of collections frameworks are the C++ Standard Template Library (STL) and Smalltalk's collection hierarchy.

Benefits

- **Reduces programming effort:** By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work.
- **Increases program speed and quality:** Provides high-performance, high-quality implementations of useful data structures and algorithms.
- **Allows interoperability among unrelated APIs:** APIs can interoperate seamlessly, even though they were written independently.
- **Reduces effort to learn and to use new APIs**
- **Reduces effort to design new APIs**
- **Fosters software reuse:** New data structures that conform to the standard collection interfaces are by nature reusable.

Core Collection Interfaces



Traversing Collections in Java

- There are two ways to traverse collections:
  - using **Iterators**.
  - with the (enhanced) **for-each** construct

Iterators

- An Iterator is an object that enables you to traverse through a collection and to remove elements from the collection selectively, if desired.
- You get an Iterator for a collection by calling its iterator method.
- Suppose **collection** is an instance of a **Collection**. Then to print out each element on a separate line:

```
Iterator<E> it = collection.iterator();

while (it.hasNext())

    System.out.println(it.next());
```

Iterators

Iterator interface:

```
public interface Iterator<E> {
        boolean hasNext();
        E next();
        void remove(); //optional
}
```

- **hasNext()** returns true if the iteration has more elements
- **next()** returns the next element in the iteration.
- **remove()** removes the last element that was returned by next.
  - remove may be called only once per call to next
  - otherwise throws an exception.
  - Iterator.**remove** is the *only* safe way to modify a collection during iteration

Implementing Iterators

- Could make a copy of the collection.
  - **Good:** could make copy private – no other objects could change it from under you.
  - **Bad:** **construction** is O(n).
- Could use the collection itself (the typical choice).
  - **Good:** **construction**, **hasNext** and **next** are all O(1).
  - **Bad:** if another object makes a structural change to the collection, the results are unspecified.

The Enhanced For-Each Statement

- Suppose **collection** is an instance of a **Collection**. Then

  for (**Object** o : **collection**)

  System.out.println(o);

  prints each element of the collection on a separate line.

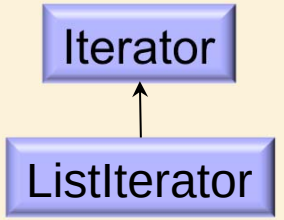
- This code is just shorthand: it compiles to use o.iterator().

The Generality of Iterators

- Note that iterators are general in that they apply to any collection.
  - Could represent a sequence, set or map.
  - Could be implemented using arrays or linked lists.

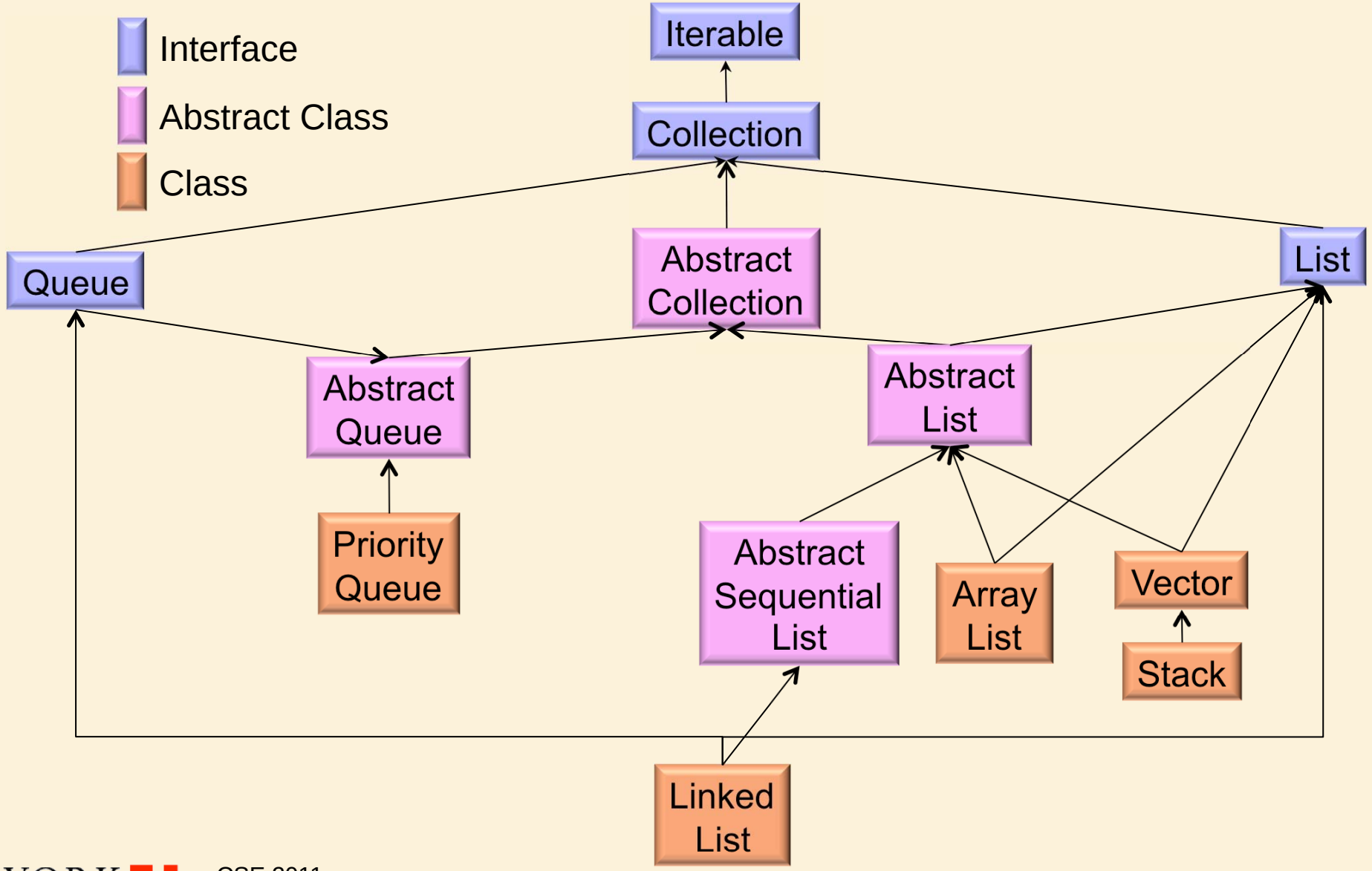
ListIterators

- A **ListIterator** extends Iterator to treat the collection as a list, allowing
  - access to the integer position (index) of elements
  - forward and backward traversal
  - modification and insertion of elements.



- This is achieved through interfaces for the additional methods:
  - **hasPrevious()**
  - **previous()**
  - **nextIndex()**
  - **previousIndex()**
  - **set()**
  - **add(e)**

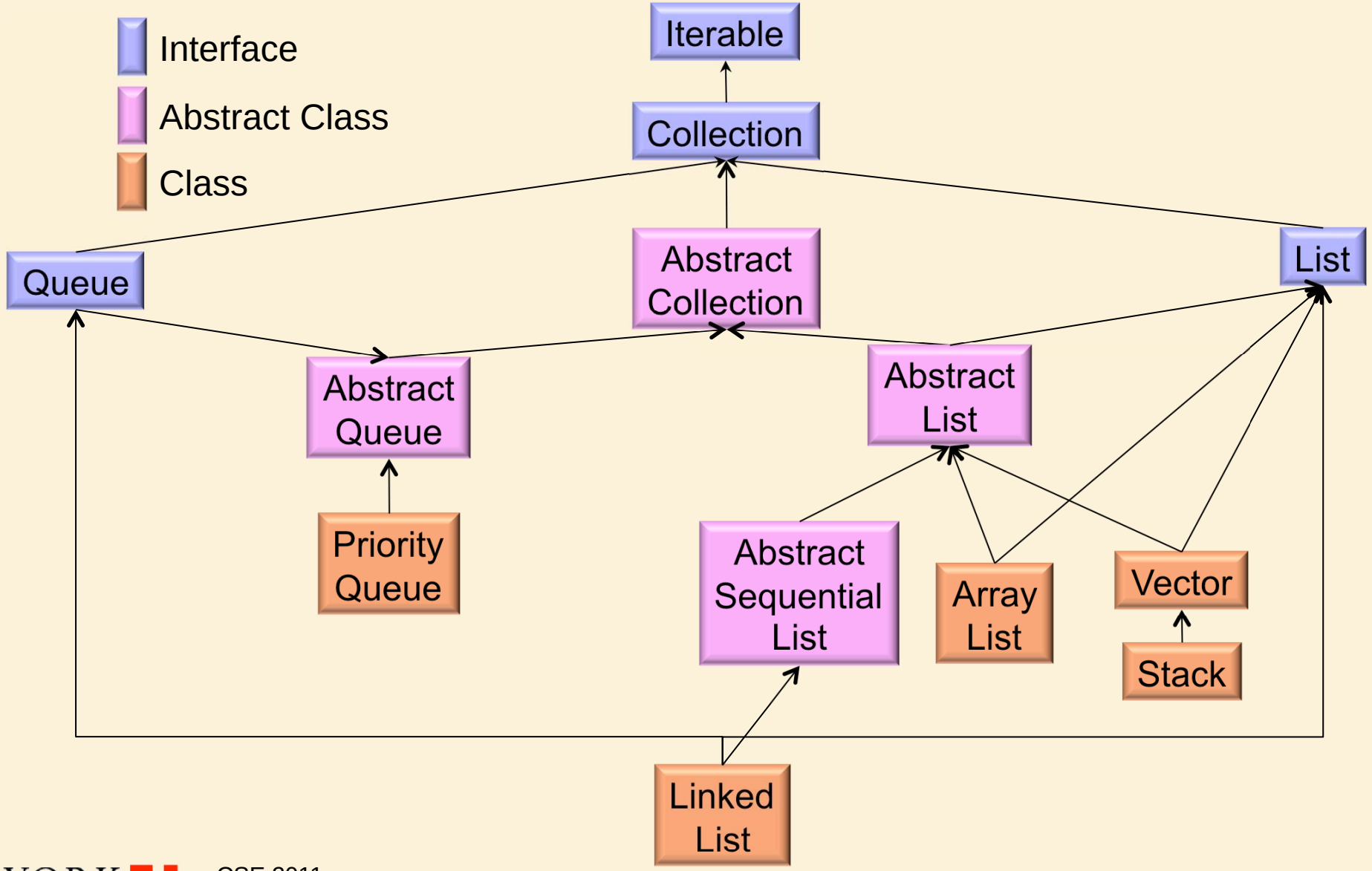
The Java Collections Framework (Ordered Data Types)



The **Iterable** Interface

- Allows an **Iterator** to be associated with an object.
- The iterator allows an existing data structure to be stepped through sequentially, using the following methods:
  - **hasNext**: does the object have any elements after the current position?
  - **next**: get the next element
  - **remove**: removes from the sequence the last element returned by next

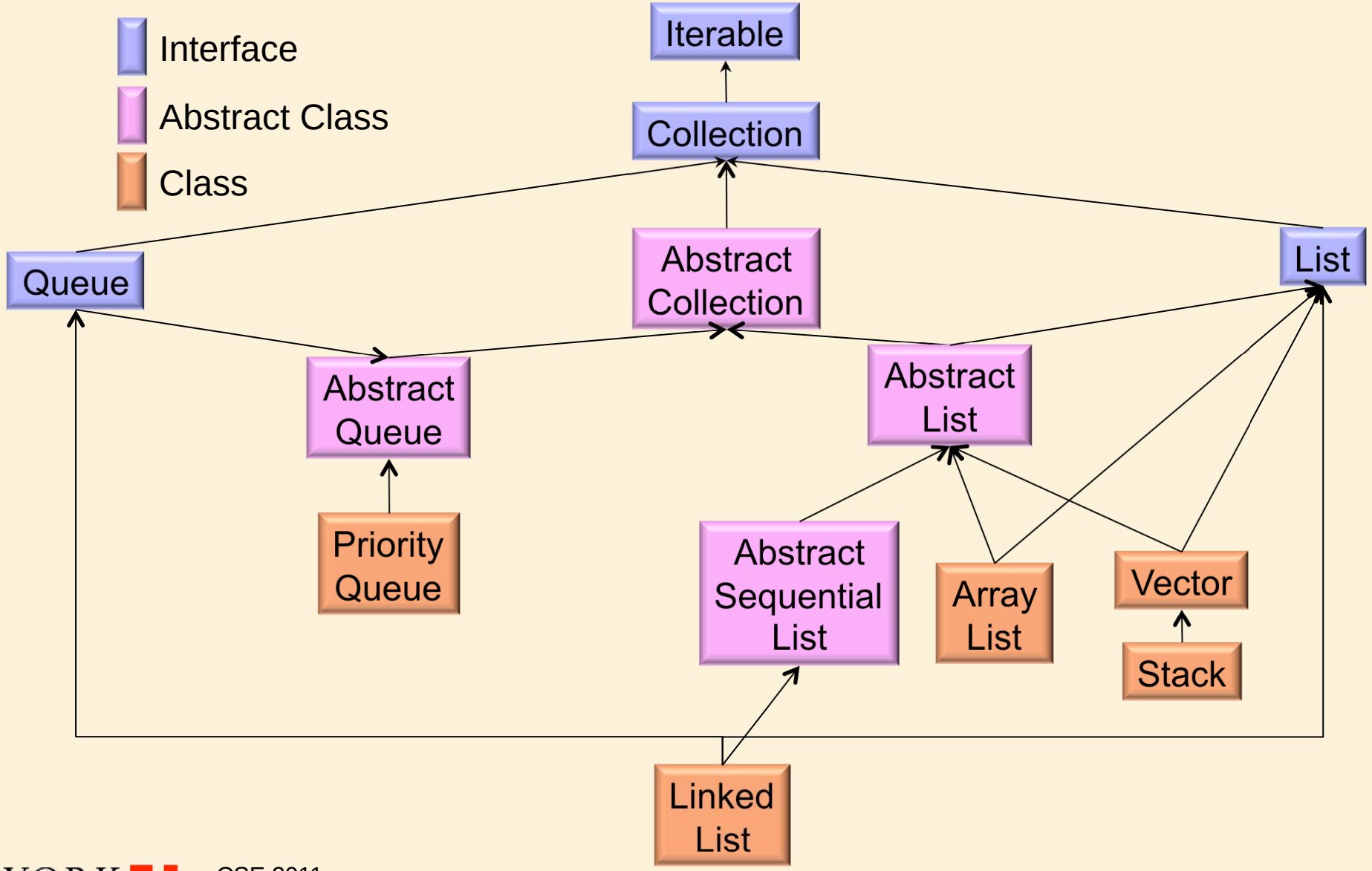
The Java Collections Framework (Ordered Data Types)



The **Collection** Interface

- Allows data to be modeled as a collection of objects. In addition to the **Iterator** interface, provides interfaces for:
  - Creating the data structure
    - **add(e)**
    - **addAll(c)**
  - Querying the data structure
    - **size()**
    - **isEmpty()**
    - **contains(e)**
    - **containsAll(c)**
    - **toArray()**
    - **equals(e)**
  - Modifying the data structure
    - **remove(e)**
    - **removeAll(c)**
    - **retainAll(c)**
    - **clear()**

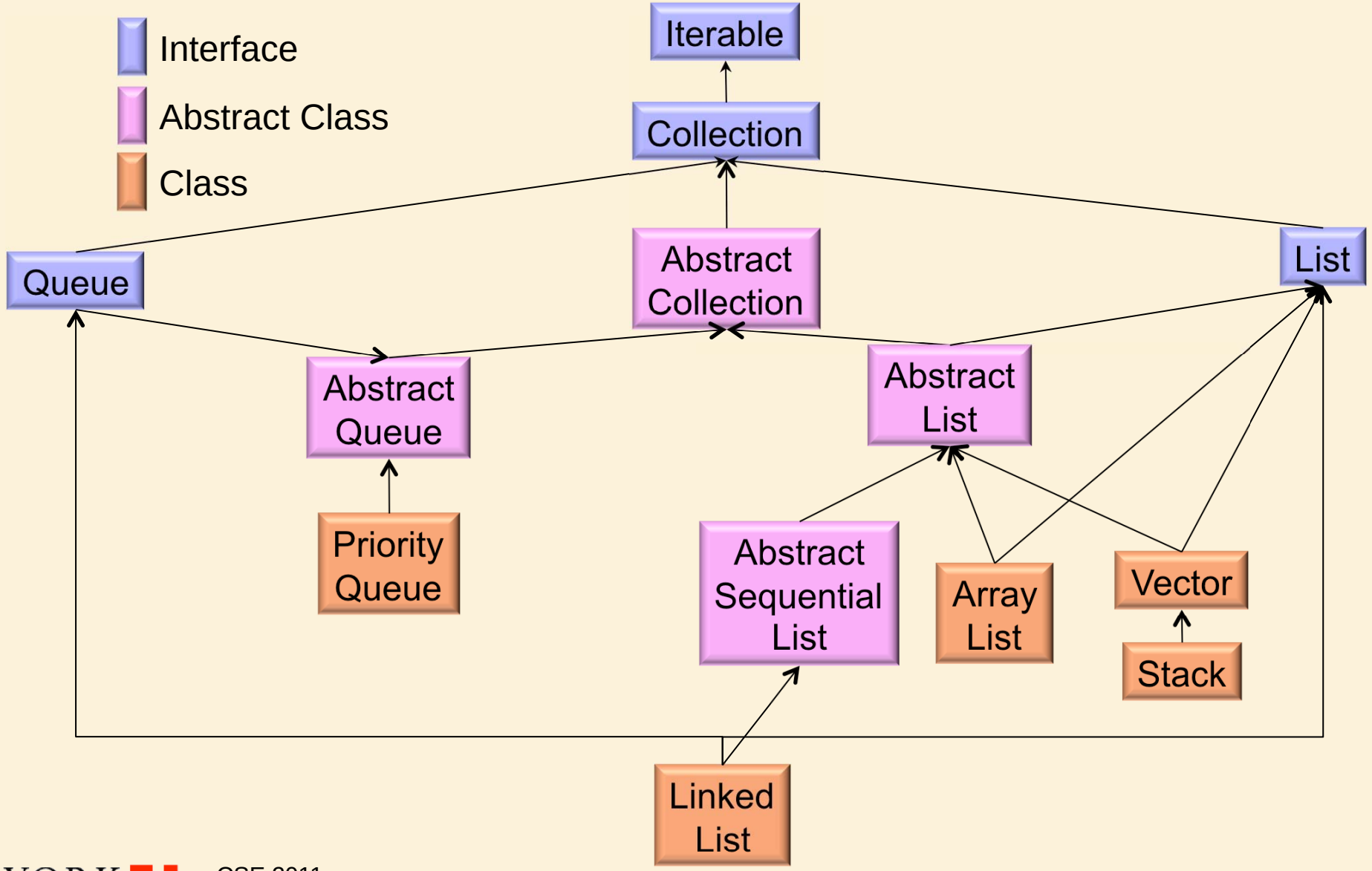
The Java Collections Framework (Ordered Data Types)



The **Abstract Collection** Class

- Skeletal implementation of the **Collection** interface.
- For **unmodifiable** collection, programmer needs to implement:
  - **iterator** (including **hasNext** and **next** methods)
  - **size**
- For **modifiable** collection, need to also implement:
  - **remove** method for **iterator**
  - **add**

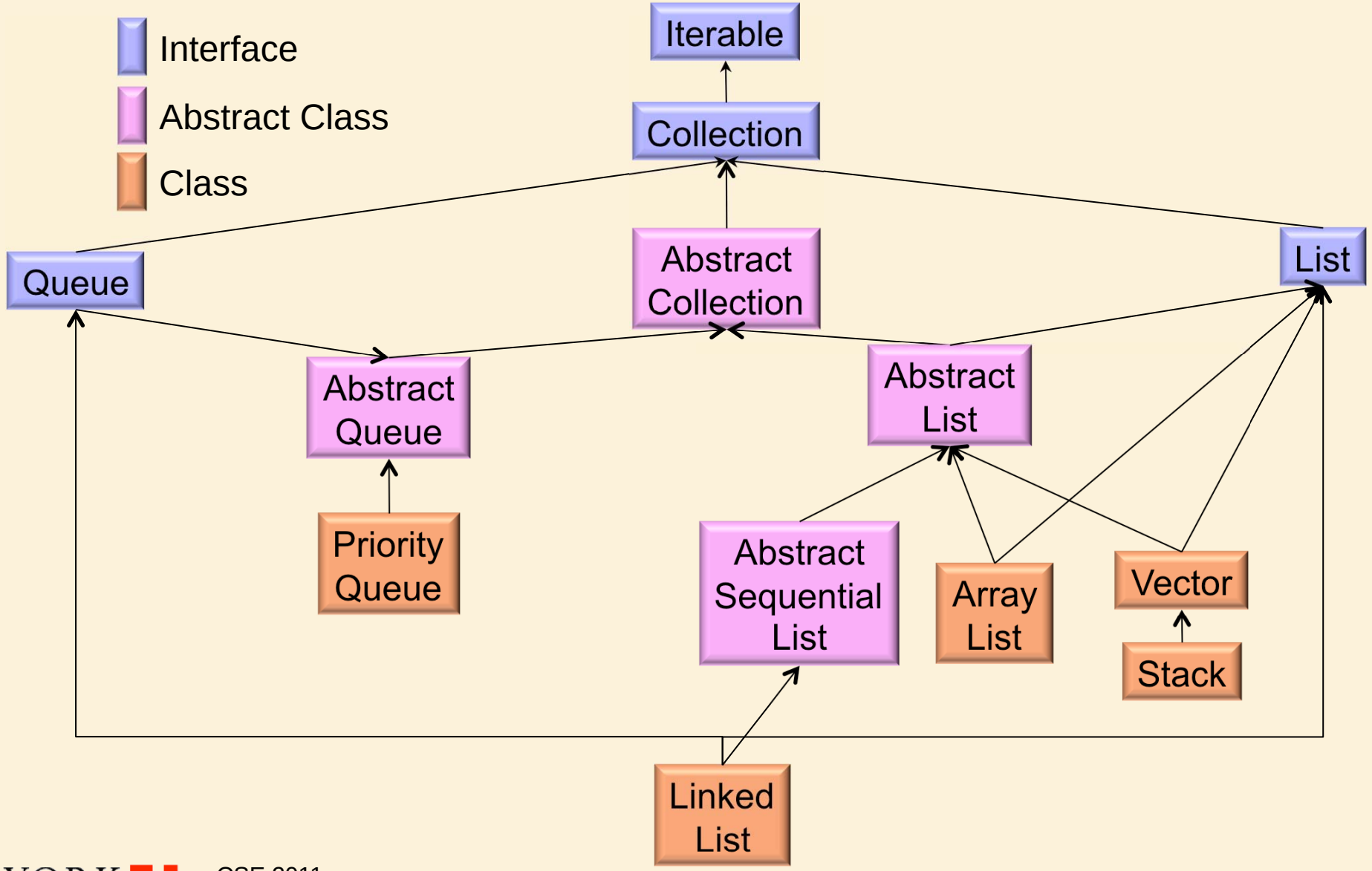
The Java Collections Framework (Ordered Data Types)



The List Interface

- Extends the Collections interface to model the data as an **ordered sequence** of elements, indexed by an integer index (position).
- Provides interface for creation of a **ListIterator**
- Also adds interfaces for:
  - Creating the data structure
    - **add(e)**
    - **add(i, e)**
  - Querying the data structure
    - **get(i)**
    - **indexOf(e)**
    - **lastIndexOf**
    - **subList(i1, i2)**
  - Modifying the data structure
    - **set(i)**
    - **remove(e)**
    - **remove(i)**

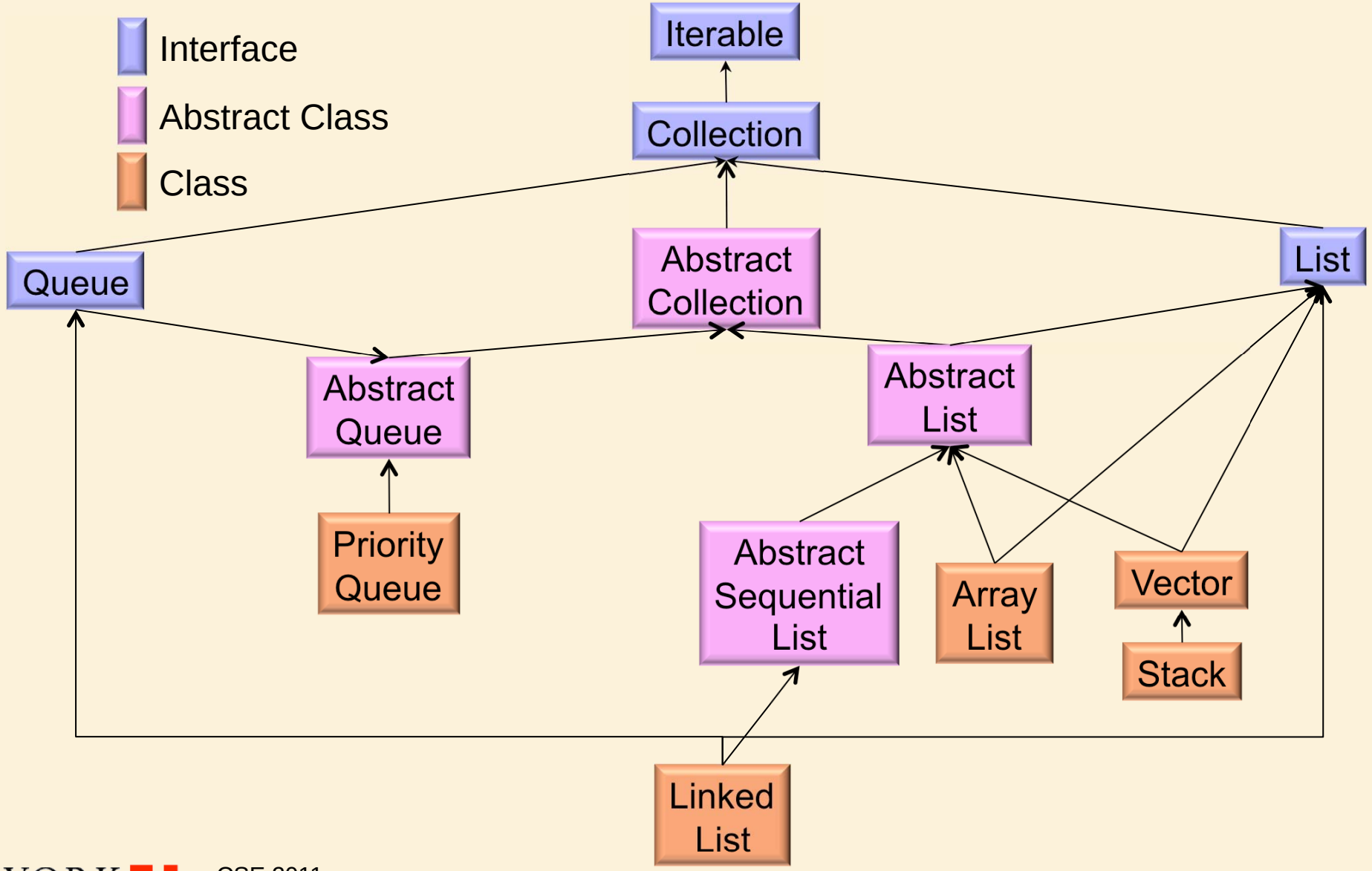
The Java Collections Framework (Ordered Data Types)



The **Abstract List** Class

- Skeletal implementation of the **List** interface.
- For **unmodifiable** list, programmer needs to implement methods:
  - **get**
  - **size**
- For **modifiable** list, need to implement
  - **set**
- For **variable-size** modifiable list, need to implement
  - **add**
  - **remove**

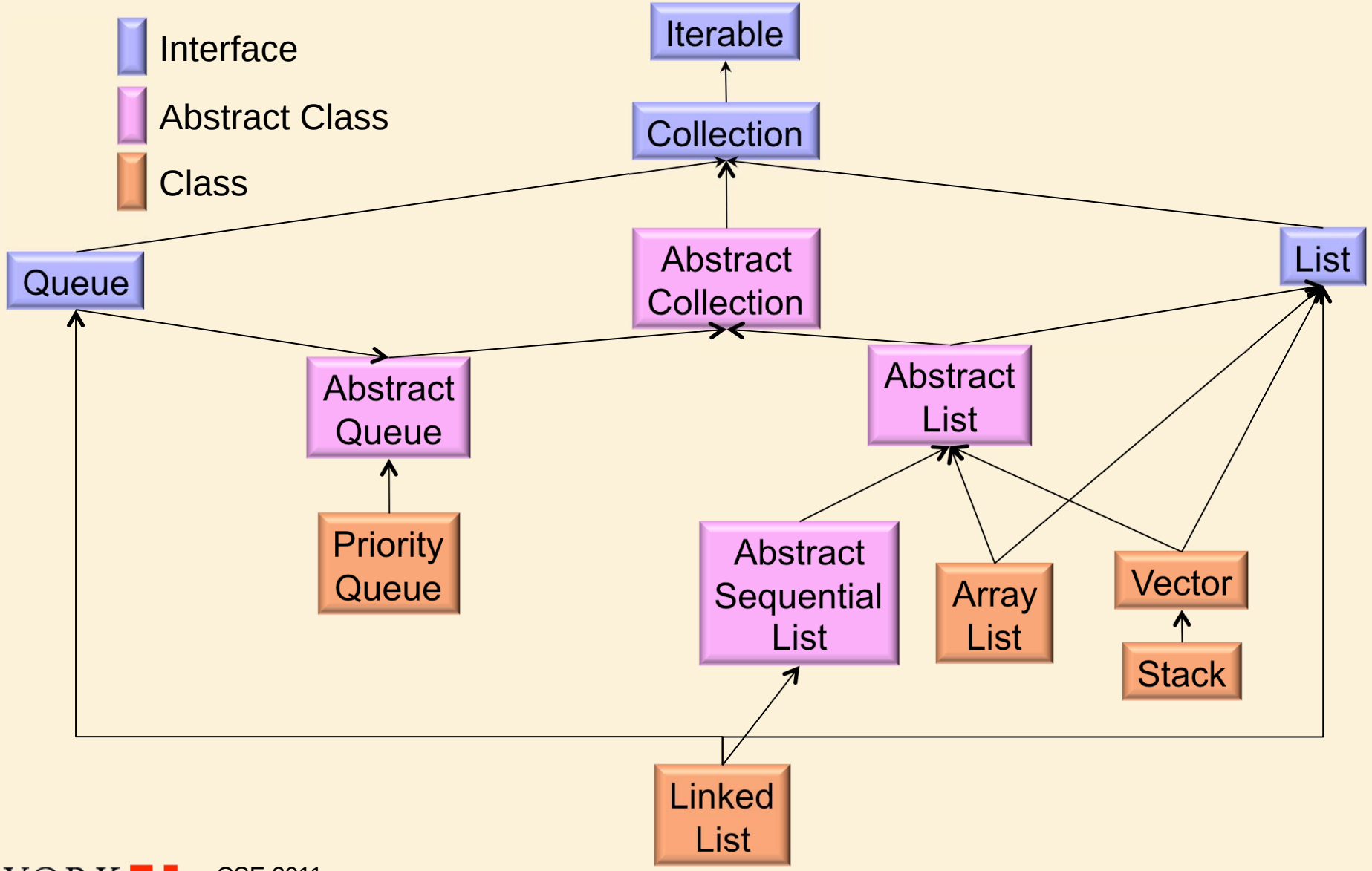
The Java Collections Framework (Ordered Data Types)



The **ArrayList** Class

- **Random access** data store implementation of the **List** interface
- Uses an **array** for storage.
- Supports automatic array-resizing
- Adds methods
  - **trimToSize()**
  - **ensureCapacity(n)**
  - **clone()**
  - **removeRange(i1, i2)**
  - **RangeCheck(i)**
  - **writeObject(s)**
  - **readObject(s)**

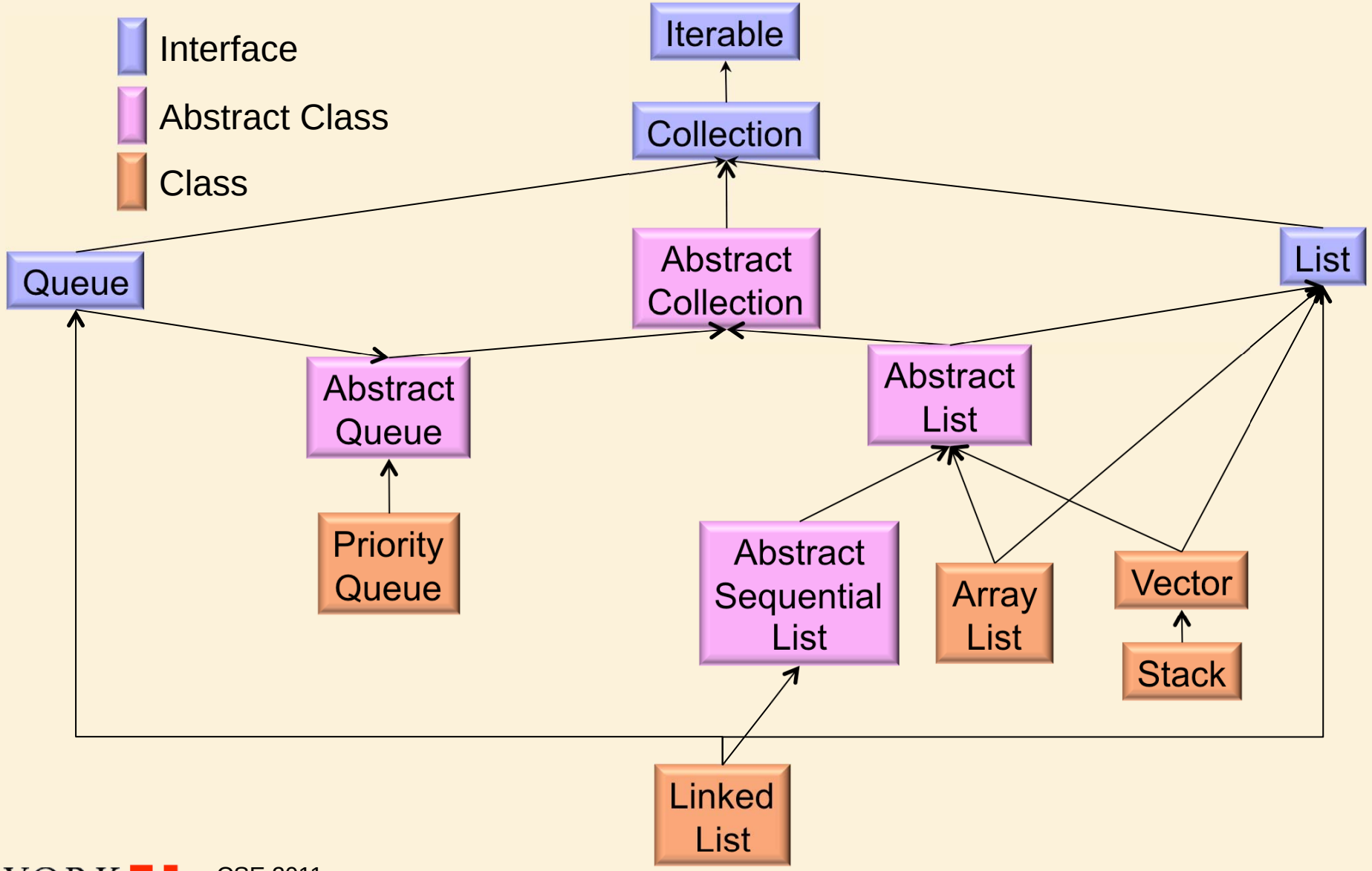
The Java Collections Framework (Ordered Data Types)



The Vector Class

- Similar to Array List.
- But all methods of Vector are synchronized.
  - Guarantees that at most one thread can execute the method at a time.
  - Other threads are blocked, and must wait until the current thread completes.

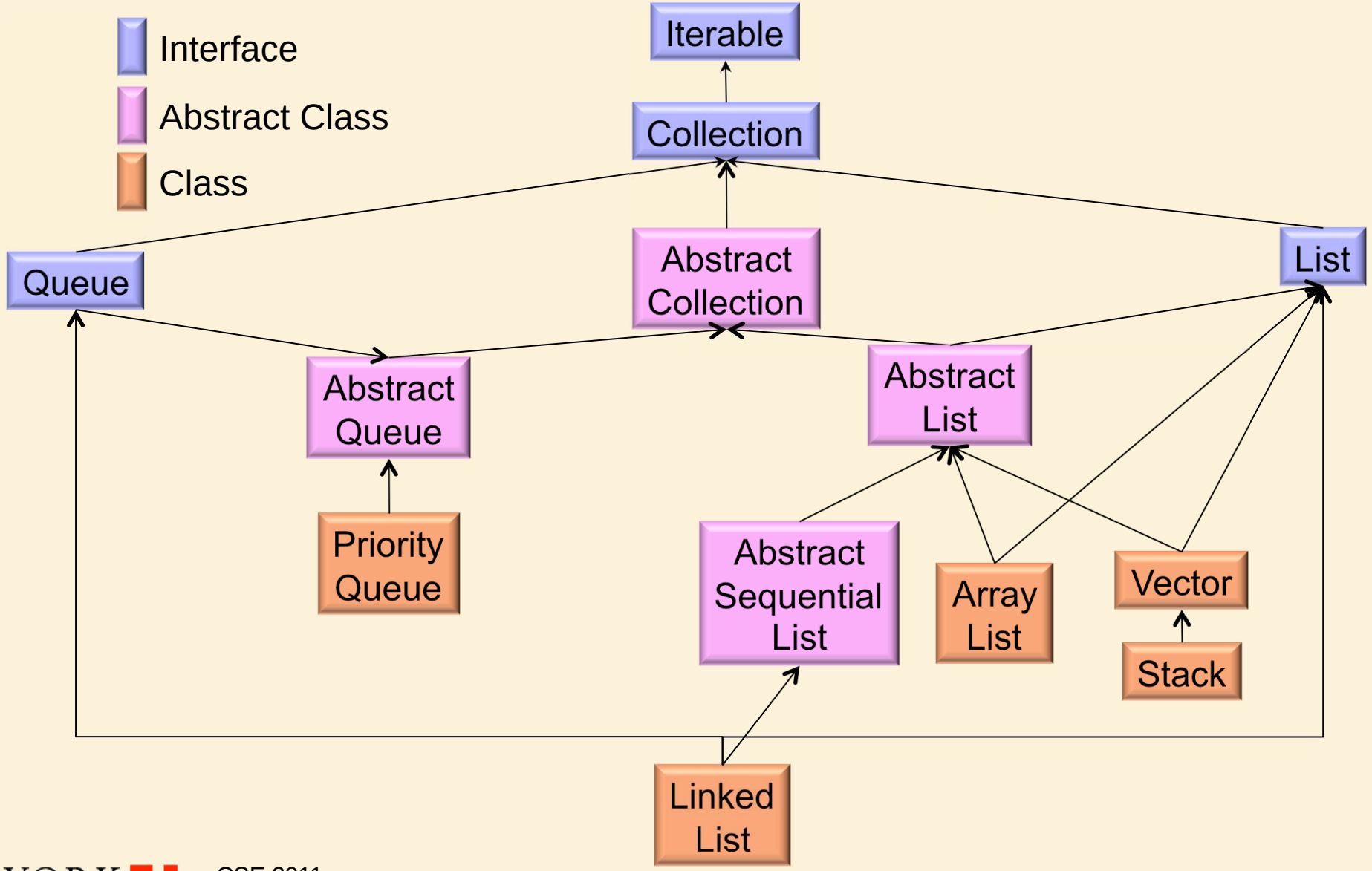
The Java Collections Framework (Ordered Data Types)



The **Stack** Class

- Represents a last-in, first-out (LIFO) stack of objects.
- Adds 5 methods:
  - **push()**
  - **pop()**
  - **peek()**
  - **empty()**
  - **search(e)**

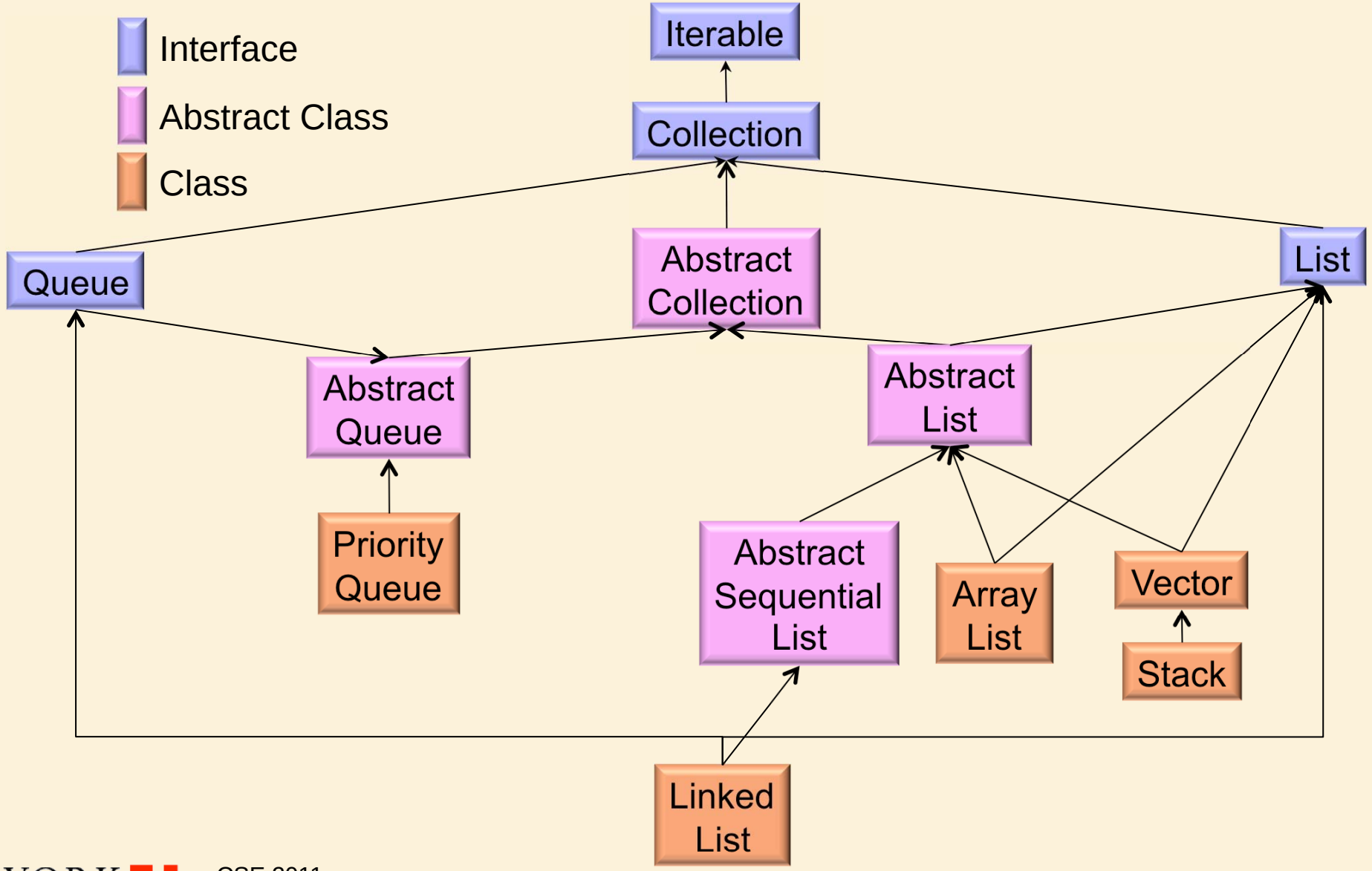
The Java Collections Framework (Ordered Data Types)



The **Abstract Sequential List** Class

- Skeletal implementation of the **List** interface.
- Assumes a **sequential** access data store (e.g., **linked list**)
- Programmer needs to implement methods
  - **listIterator()**
  - **size()**
- For **unmodifiable** list, programmer needs to implement list iterator's methods:
  - **hasNext()**
  - **next()**
  - **hasPrevious()**
  - **previous()**
  - **nextIndex()**
  - **previousIndex()**
- For **modifiable** list, need to also implement list iterator's
  - **set**
- For **variable-size** modifiable list, need to implement list iterator's
  - **add**
  - **remove**

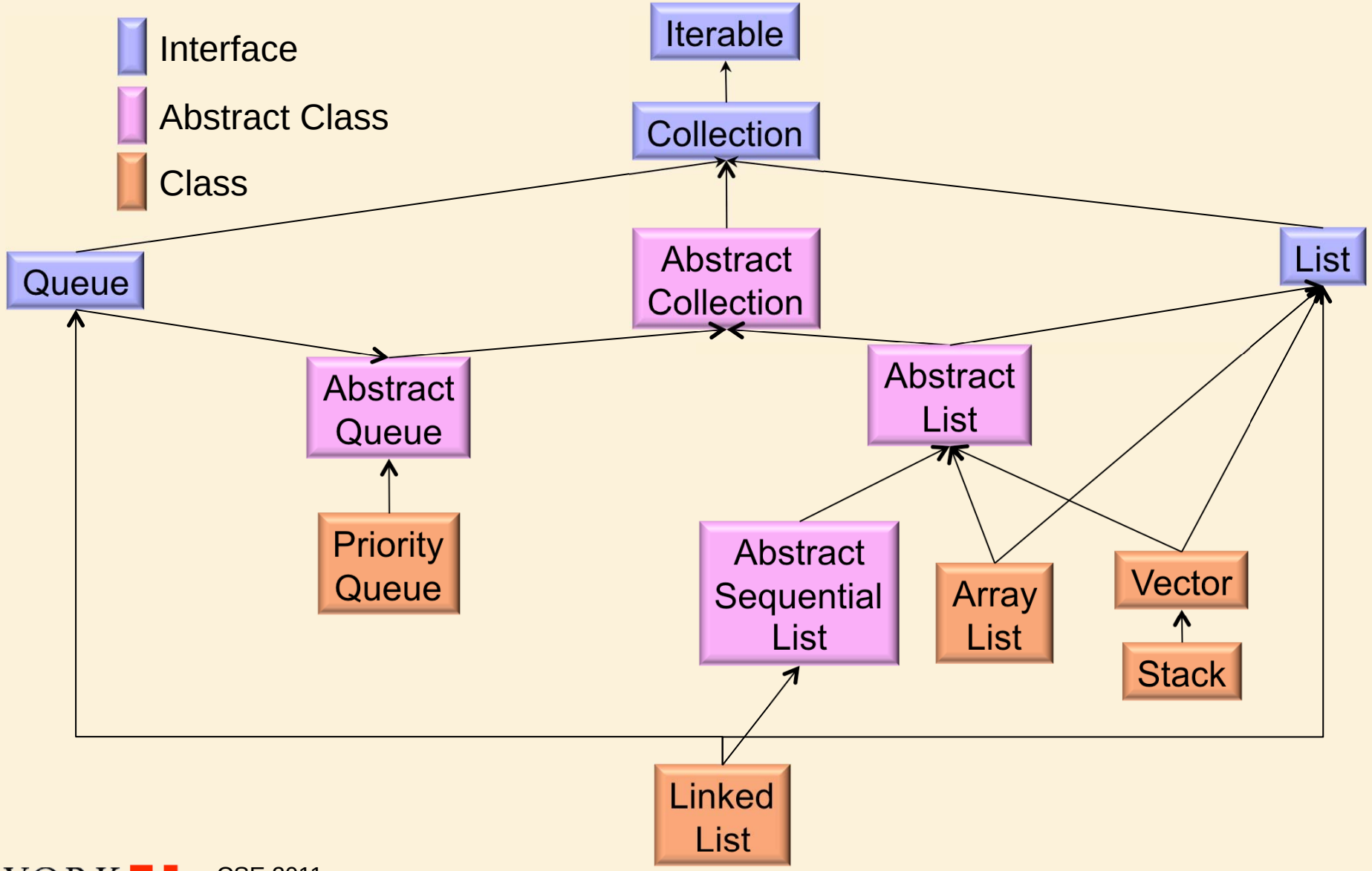
The Java Collections Framework (Ordered Data Types)



The **Queue** Interface

- Designed for holding elements prior to processing
- Typically first-in first-out (FIFO)
- Provides additional insertion, extraction and inspection operations.
- Extends the **Collection** interface to provide interfaces for:
  - **offer**
  - **poll**
  - **remove**
  - **peek**
  - **element**

The Java Collections Framework (Ordered Data Types)



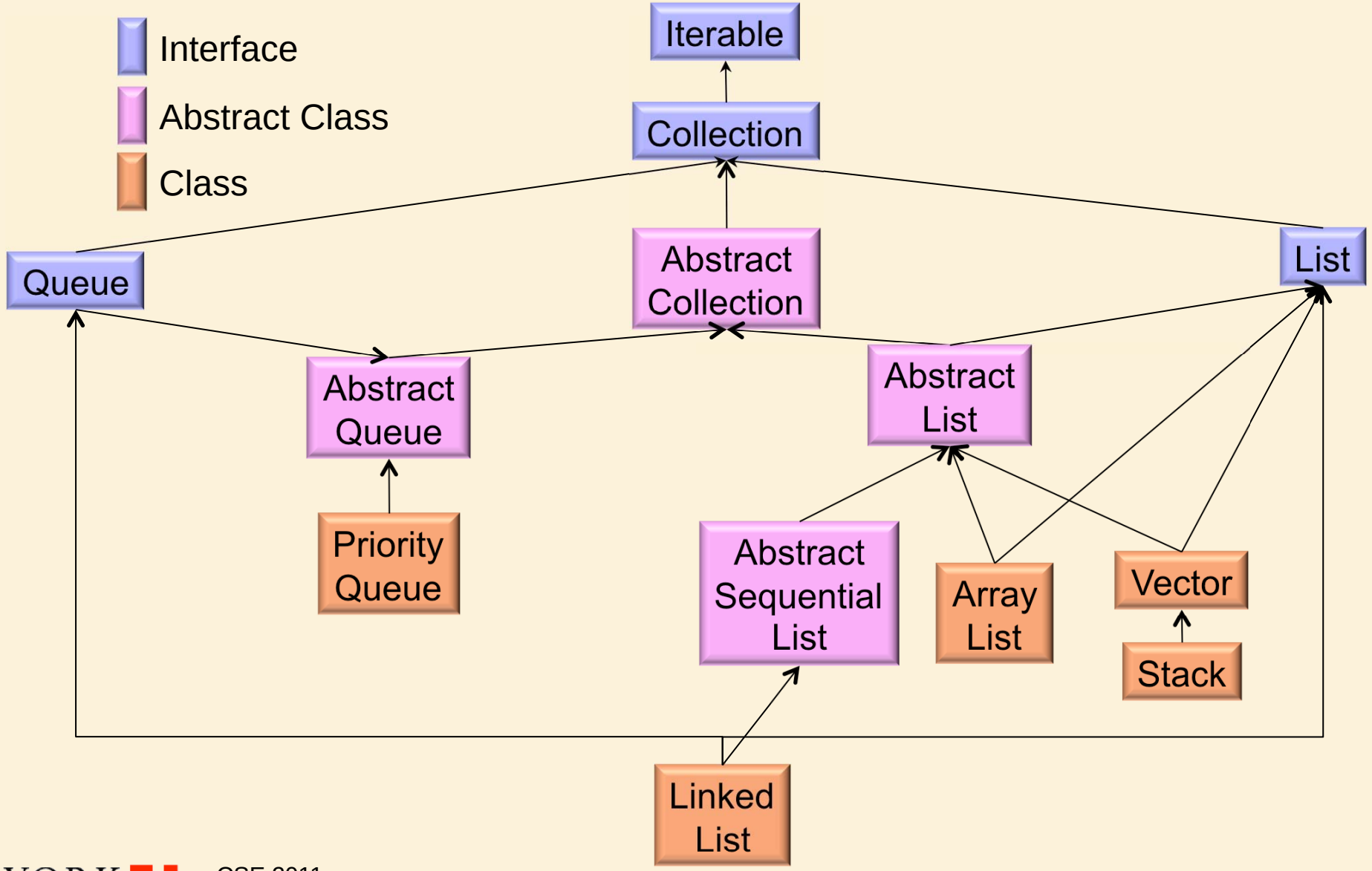
The **LinkedList** Class

- Implements the **List** and **Queue** interfaces.
- Uses a **doubly-linked list** data structure.
- Extends the **List** interface with additional methods:
  - **getFirst**
  - **getLast**
  - **removeFirst**
  - **removeLast**
  - **addFirst(e)**
  - **addLast(e)**
- These make it easier to use the LinkedList class to create stacks, queues and dequeues (dequeues).
- **Not** synchronized.

The LinkedList Class

- LinkedList objects are **not** synchronized.
- However, the LinkedList iterator is **fail-fast**: if the list is structurally modified at any time after the iterator is created, in any way except through the Iterator's own remove or add methods, the iterator will throw a **ConcurrentModificationException**.
- Thus the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

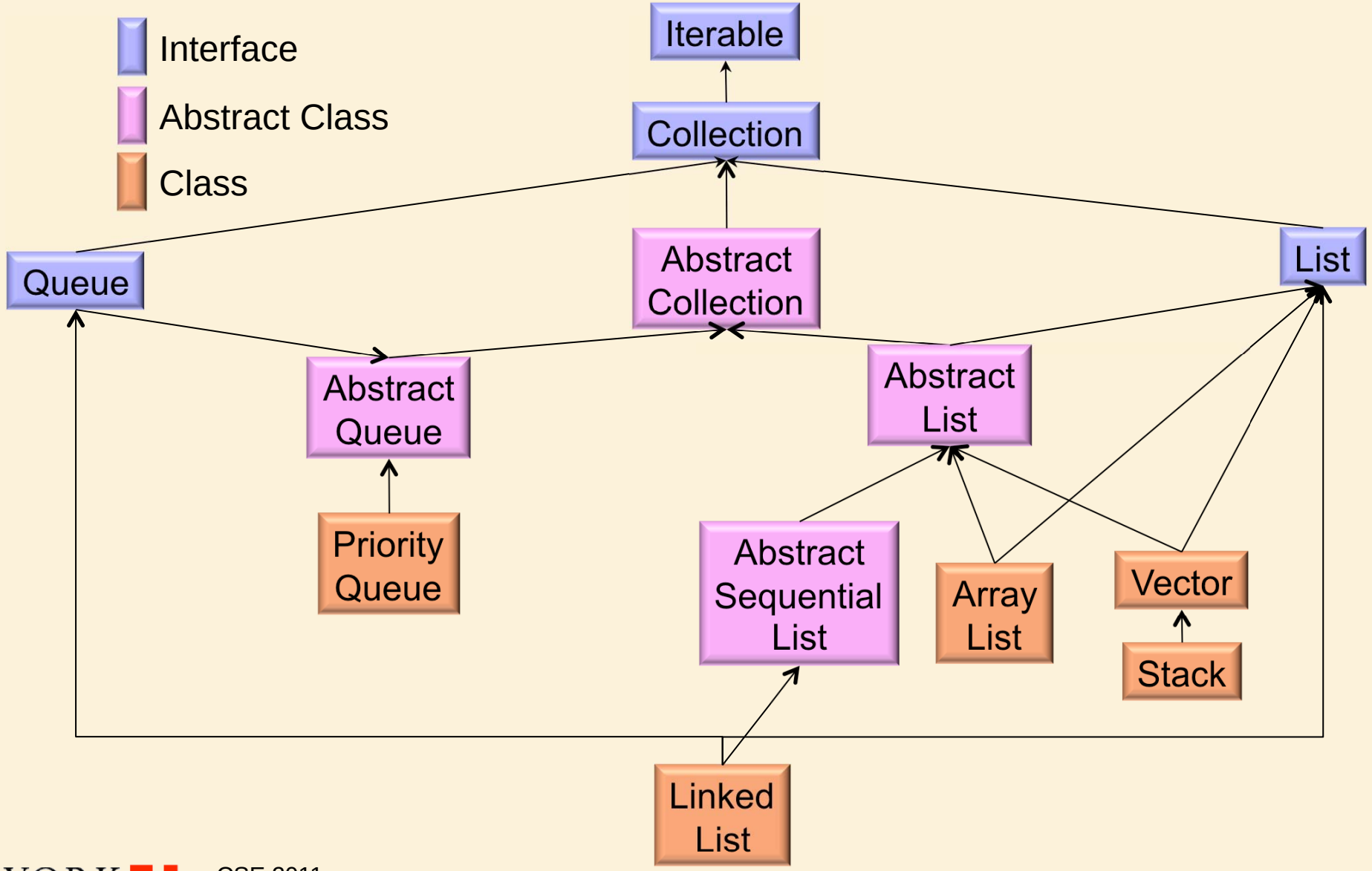
The Java Collections Framework (Ordered Data Types)



The **Abstract Queue** Class

- Skeletal implementation of the **Queue** interface.
- Provides implementations for
  - **add(e)**
  - **remove()**
  - **element**
  - **clear**
  - **addAll(c)**

The Java Collections Framework (Ordered Data Types)



The **Priority Queue** Class

- Based on priority heap
- **Provides an iterator**

- **This will be our next topic!**