Lists

Based on content from:

Java Foundations, 3rd Edition

Lists

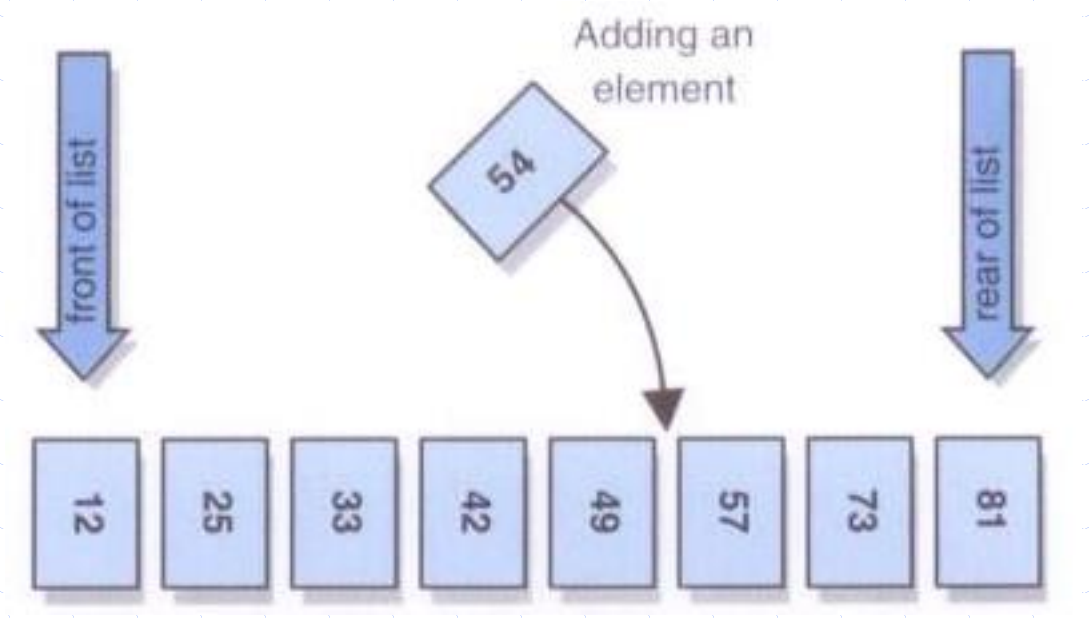
- A list is a linear collection, like stacks and queues, but is more flexible
- Adding and removing elements in lists can occur at either end or anywhere in the middle
- We will examine three types of list collections:
  - ordered lists
  - unordered lists
  - indexed lists

Ordered Lists

- The elements in an *ordered list* are ordered by some inherent characteristic of the elements
  - names in alphabetical order
  - scores in ascending order
- The elements themselves determine where they are stored in the list

Ordered Lists

- An ordered list:

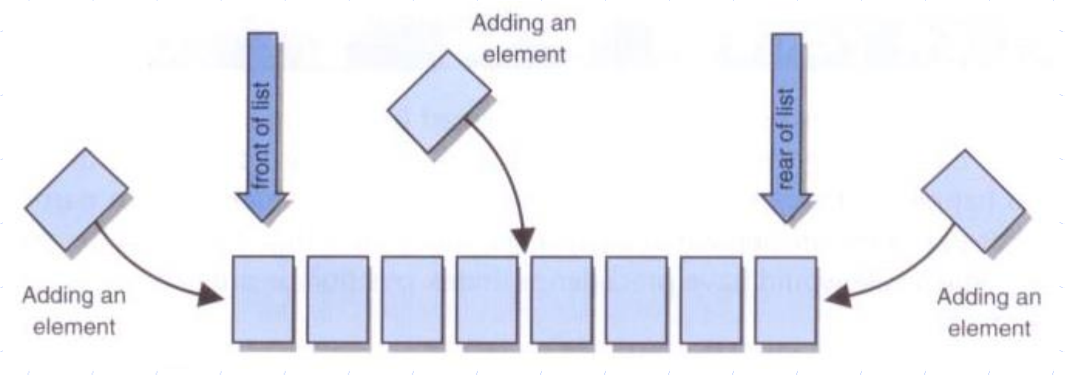


Unordered Lists

- There is an order to the elements in an *unordered list*, but that order is not based on element characteristics
- The user of the list determines the order of the elements
- A new element can be put on the front or the rear of the list, or it can be inserted after a particular element already in the list

Unordered Lists

- An unordered list:

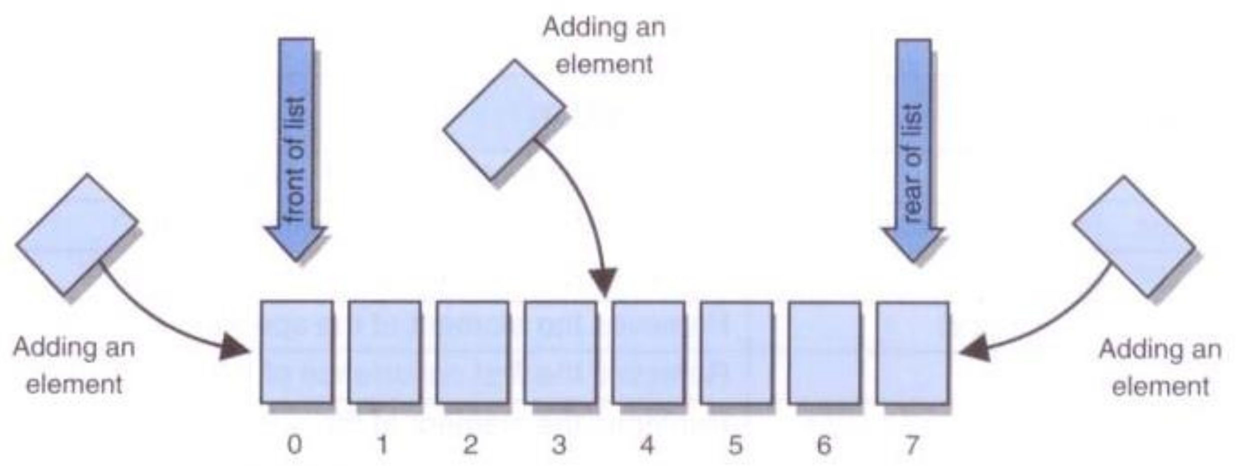


Indexed Lists

- In an *indexed list*, elements are referenced by their numeric position in the list
- Like an unordered list, there is no inherent relationship among the elements
- The user can determine the order
- Every time the list changes, the indexes are updated

Indexed Lists

- An indexed list:

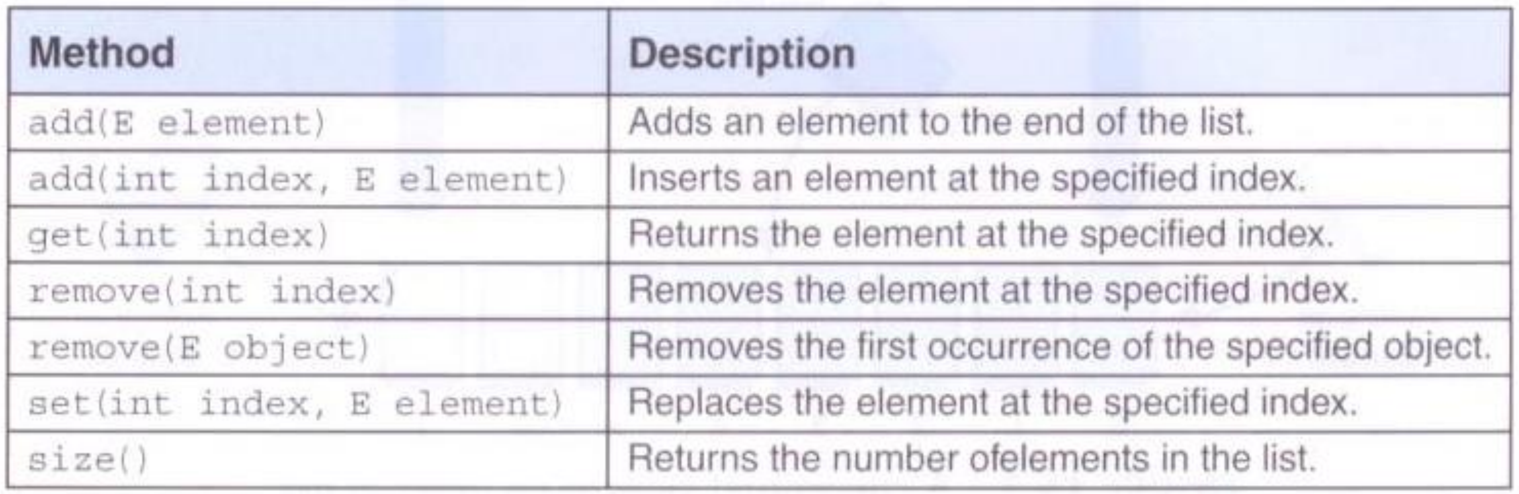


Lists in the Java API

- The list classes in the Java API primarily support the concept of an indexed list (and somewhat an unordered list)
- The API does not have any classes that directly implement an ordered list
- The `ArrayList` and `LinkedList` classes both implement the `List<E>` interface

Lists in the Java API

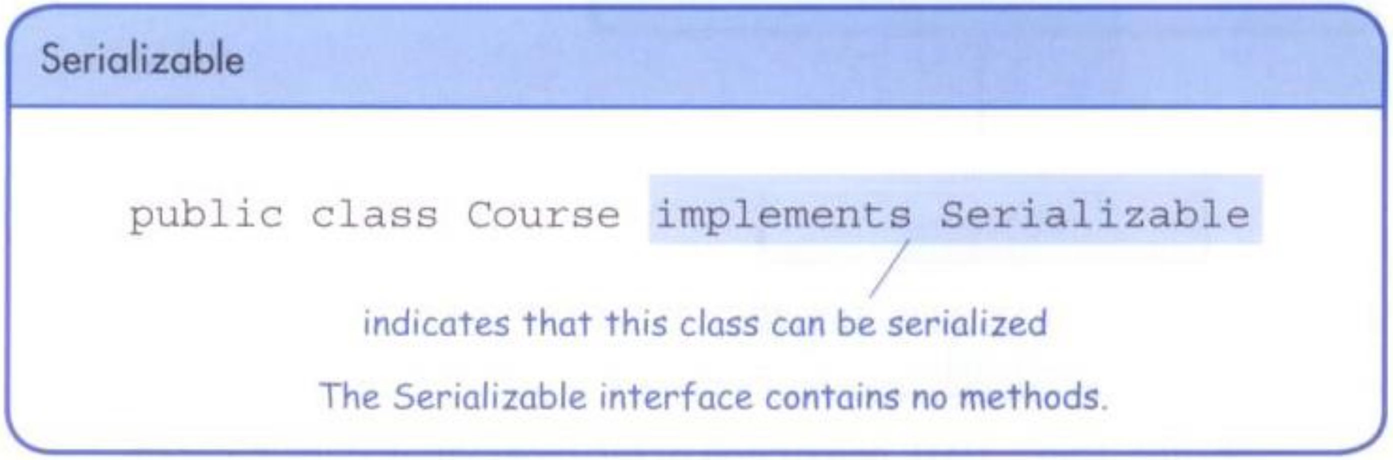
- Some of the operations from the `List<E>` interface:



Method	Description
`add(E element)`	Adds an element to the end of the list.
`add(int index, E element)`	Inserts an element at the specified index.
`get(int index)`	Returns the element at the specified index.
`remove(int index)`	Removes the element at the specified index.
`remove(E object)`	Removes the first occurrence of the specified object.
`set(int index, E element)`	Replaces the element at the specified index.
`size()`	Returns the number ofelements in the list.

Serialization

- Any class whose objects will be saved are tagged with the `Serializable` interface


Serializable

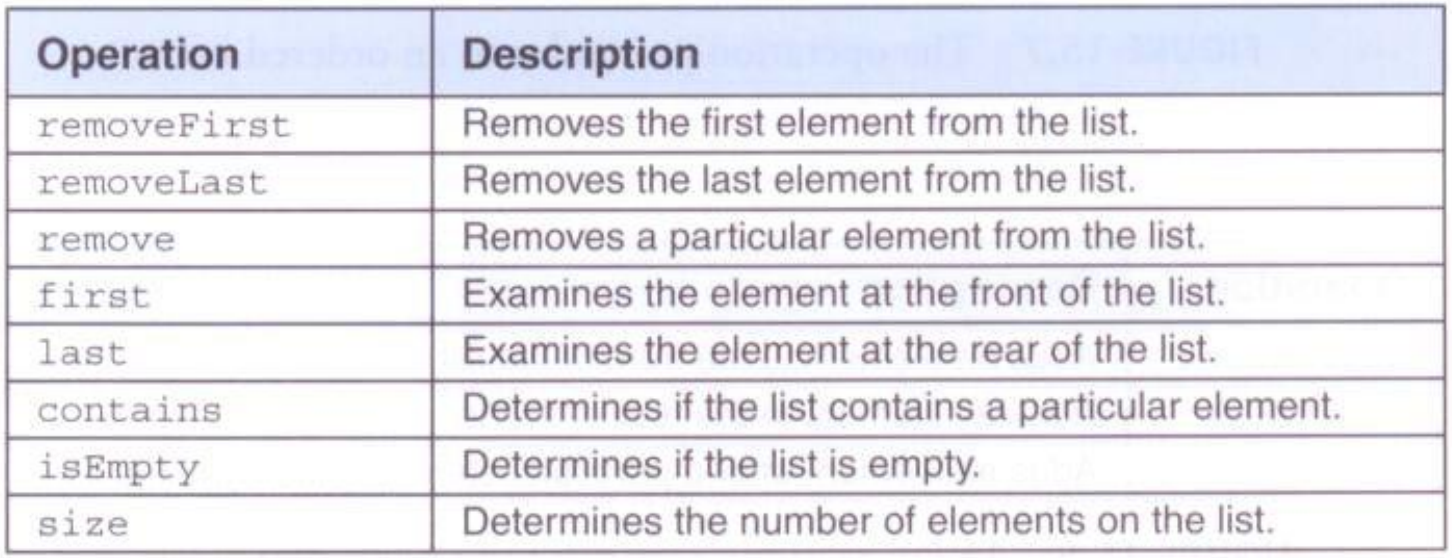
```
public class Course implements Serializable
```

indicates that this class can be serialized

The Serializable interface contains no methods.

Implementing Lists

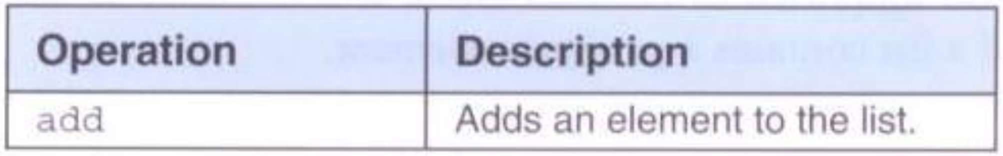
- The following operations are common to most types of lists:



Operation	Description
`removeFirst`	Removes the first element from the list.
`removeLast`	Removes the last element from the list.
`remove`	Removes a particular element from the list.
`first`	Examines the element at the front of the list.
`last`	Examines the element at the rear of the list.
`contains`	Determines if the list contains a particular element.
`isEmpty`	Determines if the list is empty.
`size`	Determines the number of elements on the list.

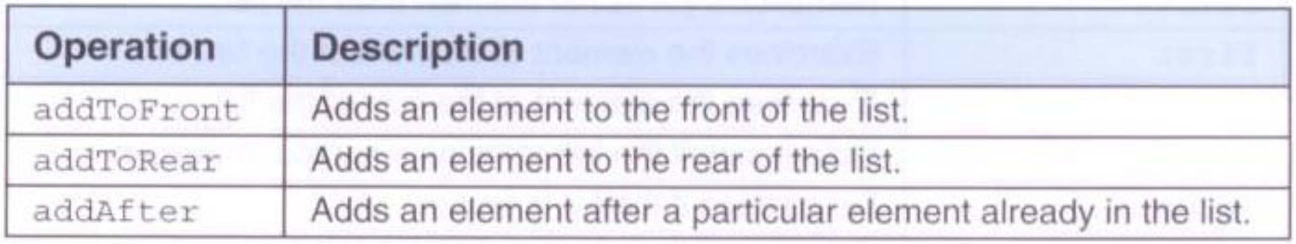
Implementing Lists

- Operation particular to an ordered list:



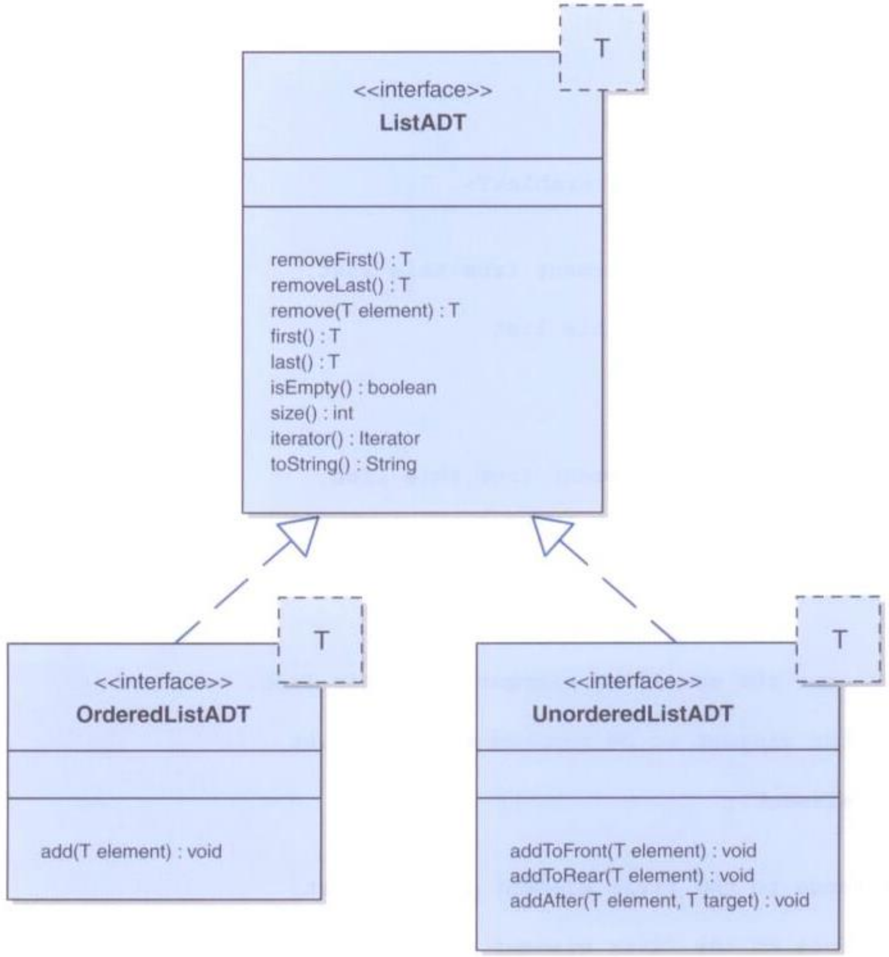
Operation	Description
`add`	Adds an element to the list.

- Operations particular to an unordered lists:



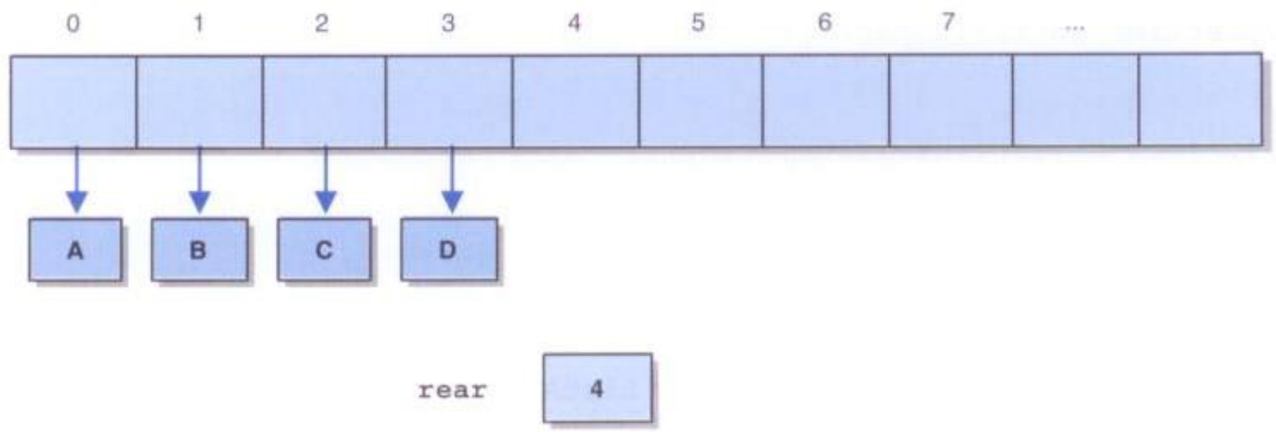
Operation	Description
`addToFront`	Adds an element to the front of the list.
`addToRear`	Adds an element to the rear of the list.
`addAfter`	Adds an element after a particular element already in the list.

Class Diagram of List Classes



Implementing a List with an Array

- Since elements can be added anywhere in the list, shifting elements cannot be avoided
- So a straightforward implementation can be adopted:



Implementing a List with Links

- A classic linked list is an obvious choice for implementing a list collection
- Will need to implement `Node` class
- Both `head` and `tail` references are maintained, as well as an integer `count`