Lists - Ordered set of elements drawn from some type T

- Duplicates, null elements are allowed.

Operations: add, remove, contains, size, isEmpty, clear, iterator

Additional ops: indexing operations:
get(index), add(index, e), set(index, e), remove(index)

Implementations:

ArrayList: Array of elements in list order.

Insert new element into a full arraylist: Reallocate a bigger array,
Copy elements into big array and continue.

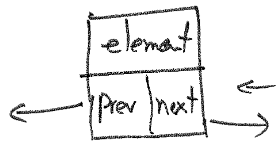
- Resize operation - takes $O(n)$ time when list has n elements
Amortized cost per insert = $O(1)$ if array doubles in size

Efficient ops: get(index), set(index, e), iteration - $O(1)$
$O(1)$ per element.

Inefficient ops: add, remove, contains .. - $O(n)$.

Problem: If contiguous memory is not available, resize will fail even though more than enough memory is available.

Linked Lists:



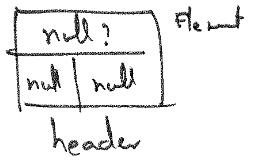
← Entry

Chain of elements (not contiguous) that are linked by prev, next pointers.

Efficient ops: add, iterator, add/remove using iterator - $O(1)$
Inefficient: all indexing operations, contains, ... - $O(n)$

Options: - Singly linked, doubly linked, circular, $\boxed{\text{doubly linked circular}}$
Java LinkedList

- Dummy header element

Empty list:


Standard usage of Lists

```
List<Integer> l = new ArrayList<>();
    :
l.add(x);      ....  for (Integer x : l) {...do something with x}
```

Queues - List of elements accessed in FIFO order (First-in, First Out)

Traditionally: Enqueue, Dequeue, ~~Element~~, Top, isEmpty - operations

Java: Interface with operations

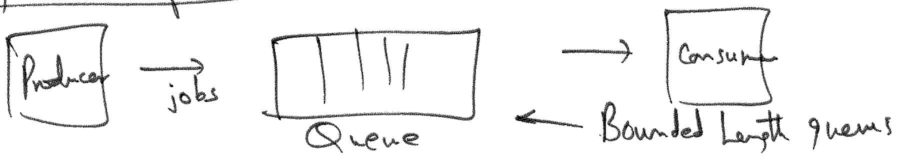
add, remove, Element, isEmpty

Implementations: LinkedList, Priority Queue, ArrayDeque

Common usage:

```
Queue<T> q = new LinkedList<>();
q.add(x);              q.isEmpty() ?...
   :
x = q.remove();
```

Other applications of Queues: Producer/Consumer problems.



~~add~~ offer ~~remove~~ poll ~~Element~~ peek] — never raise exceptions

while (x = q.poll() == null);

Stacks

List of elements accessed in LIFO order (Last in First out)

Operations: Push, Pop, Top, isEmpty.

Java: ~~Stack is implemented as extension of vector~~

Use ArrayDeque instead.

Stacks have many uses. - Next class we will look at some.

Some problems for next class (discuss in forum in between):

1. 2 sorted lists implementing sets — Intersection of two sets

LinkedList<T> intersection(List<T> l_1, List<T> l_2) {..}

- union, set difference

— Write efficient code using only list operations - $O(n)$