declaration:

    public boolean myMethod(int a,
                            int b)

definition:

    {
        
        ...

        
    }
abstract data type (ADT):
a set of objects +
the set of operations

Ex: set of any type +
    add, remove, contains

Ex: disjoint sets +
    union two sets

find: tells you which
    set an object belongs
to

Java: interfaces

C++: abstract classes

Swift: protocols
Exs: Java Collections API (application programming interface) bunch of standard ADTs also basic implementations of these ADTs
Lists

A list is a collection of elements $A_0, A_1, \ldots, A_{n-1}$.

If $n = 0$, it is the empty list.

$A_i$ follows or succeeds $A_{i-1}$ (for $i > 0$)
$A_i$ proceeds $X_{i+1}$

(for $i = n-1$)

position of $A_i$ is $i$.

operations:
- print $List$
- find $(x)$: return least $i$ s.t. $A_i = x$ or
- 1 if no such $i$
- insert((x, i)): add x at position i and move
  the old A[i] up one position.

- remove(i): removes element at position i and
  move later elements down one position.

- find_kth(i): returns element at position i.
Implementation 1: An array!

Print list in \( \Theta(n) \) time

Find kth: \( \Theta(1) \) time

Find \( (x) \): \( O(u) \) time if \( x \in \mathbb{A} \)

Insert \& remove: \( \Theta(n) \) time in worst case

(faster if \( u = n-1 \) ...)
Java Collections API has an ArrayList for faster operations at end of list.
Linked Lists

A sequence of nodes, not necessarily adjacent in memory.

Each node stores one element and a link to its successor's node. Last node links to null.

Maintain a link to
first element.

\[
\begin{array}{c}
A_0 \quad \text{next} \\
A_1 \quad \text{next} \\
A_2 \quad \text{next} \\
A_3 \quad \text{next} \\
A_4 \quad \text{next}
\end{array}
\]

↑

first

printList & find!

iteratively follow next links & deal with elements as you go

\[\Theta(n) + \Theta(i)\] as before
Find $k$th $(i)$: have to follow next links so $\Theta(i)$ time

remove: just update the next link of the previous node
insert: two links updates

\[ \Theta(1) \text{ time if you have a link to predecessor. } \]
\[ \Theta(n) \text{ time if not. } \]
For quick insert/remove at end...

maintain a link to last member of list & add prev links to all nodes

\[ \Theta(1) \text{ time insert/remove at beginning or end} \]
- or - if you have a link to node you're removing ↑

doubly linked list

first element often called the head of the list
textbook tail: last element but usually tail is
A.

A guy

-1

A
public class OurLinkedList implements OurList {
   // ...

   private static class Node {
      public Node(int element, Node prev, Node next) {
         this.element = element;
         this.prev = prev;
         this.next = next;
      }

      public int element;
      public Node prev;
      public Node next;
   }
   // ...

   public OurLinkedList() {
      this.first = null;
      this.last = null;
      this.size = 0;
   }
   // ...

   private Node first;
   private Node last;
   private int size;
}
public class OurLinkedList implements OurList {
    static final int NOT_FOUND = -1;

    // ...

    public void printList() {
        Node current = this.first;
        while (current != null) {
            System.out.println(current.element);
            current = current.next;
        }
    }

    public int find(int x) {
        Node current = this.first;
        int position = 0;
        while (current != null) {
            if (current.element == x) {
                return position;
            }
            current = current.next;
            position++;
        }
        return NOT_FOUND;
    }

    // ...
}
public class OurLinkedList implements OurList {
    // ...
    
    // Either returns the ith node or throws an exception.
    private Node getNode(int i) {
        if (i < 0 || i >= this.size) {
            throw new IndexOutOfBoundsException();
        }
        Node current = this.first;
        int position = 0;
        while (current != null) {
            if (position == i) {
                return current;
            }
            current = current.next;
            position++;
        }
        return null; // should never reach this line
    }
    // ...
}
public class OurLinkedList implements OurList {
    // ...

    public int findKth(int i) {
        Node kth = this.getNode(i);
        return kth.element;
    }

    public void insert(int x, int i) {
        Node newNode = new Node(x, null, null);

        if (i > 0 && i < this.size) {
            Node prev = this.getNode(i - 1);
            Node next = prev.next;
            newNode.prev = prev;
            prev.next = newNode;
            newNode.next = next;
            next.prev = newNode;
        } else if (i > 0) {
            Node prev = this.getNode(i - 1);
            newNode.prev = prev;
            prev.next = newNode;
            this.last = newNode;
        } else if (i < this.size) {
            Node next = this.getNode(i);
            newNode.next = next;
            next.prev = newNode;
            this.first = newNode;
        } else {
            this.first = newNode;
            this.last = newNode;
        }
    }

    public void remove(int i) {
        Node oldNode = this.getNode(i);

        if (oldNode == this.first) {
            this.first = oldNode.next;
        } else {
            oldNode.prev.next = oldNode.next;
        }

        if (oldNode == this.last) {
            this.last = oldNode.prev;
        } else {
            oldNode.next.prev = oldNode.prev;
        }
    }

    // ...
}
Stack

All insertions & removals often just reading data happen at end of list

end of list called the top of stack.

The top of stack:

1) `top()`: returns top element
2) `push()`: insert new top
3) `pop()`: remove top of stack
Also called LIFO (last in, first out)

Just need to update a few list links so all operations O(1) time.
Could also use array. That's larger than $(n)$. Just move top of stack as you push and pop.
Exs: Check for balanced parentheses:

Loop through characters.
At open `, push onto stack.
At close `)`, check if it matches top of stack then pop.
At end, check that stack is empty.
Ex: Method variables go in a stack frame.

On method call, push a new frame.
Pop on return.

Push down to lower addresses in practise.
Watch out for stack overflow!
Queues:

all inserts at end
all removals at start

insert called enqueue
removal called dequeue

dqueue ← enqueue
enqueue ←

FIFO (first in first out)
$\Theta(1) \text{ time operations}$