Abstract Data Types (Linear)

Mason Vail
Boise State University Computer Science

Abstract Data Type (ADT)

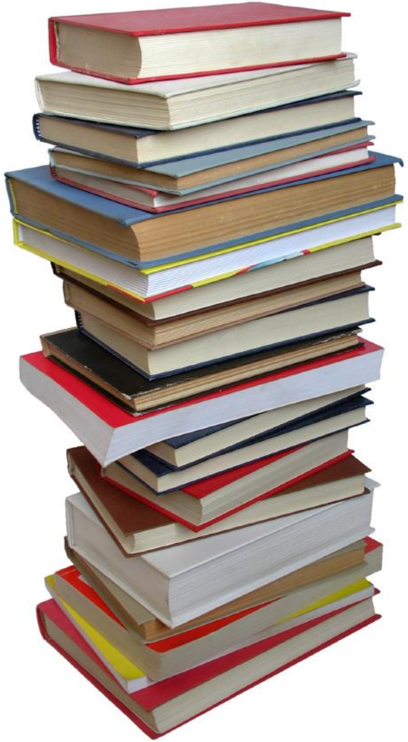
An Abstract Data Type, or ADT, is the abstraction of a class of objects that manages data through a specified set of operations. It is the mental model of a thing from the user's perspective.

Technically, every class and interface fits that definition.

Most often, however, when people talk about ADTs, they're doing so in the context of data structures - objects whose primary purpose is to store and retrieve data according to a specific organization, like a list or a tree.

An ADT, then, describes the mental model for how data will be organized and accessed.

Stack: a simple linear ADT

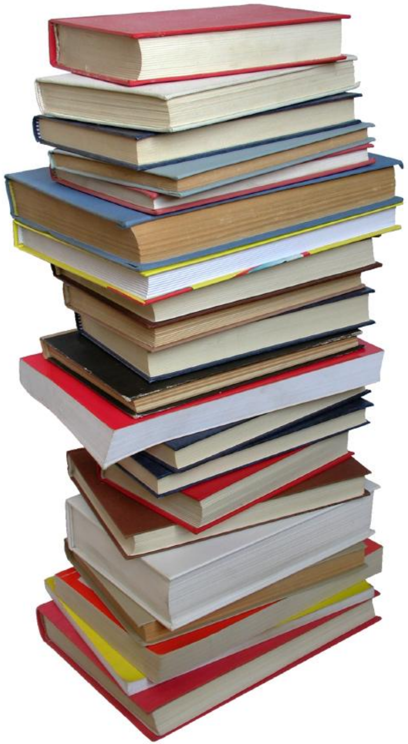


Behold! A stack of books, where the only cover (and title) you can see is for the top book.

You now have a usable mental model of how the Stack ADT works.

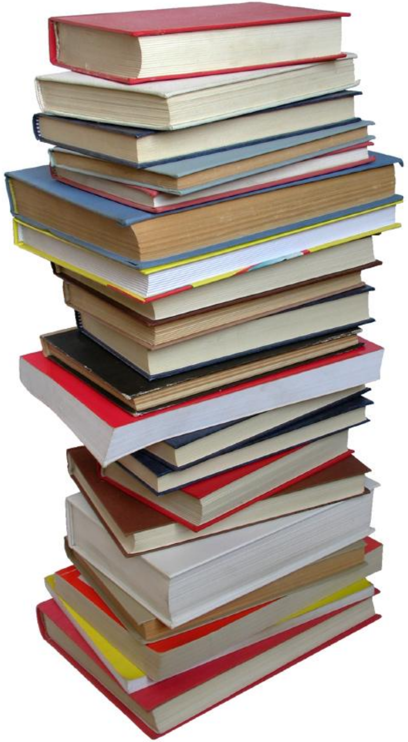
We just need to define methods for its operations in an Interface.

Stack: a simple linear ADT



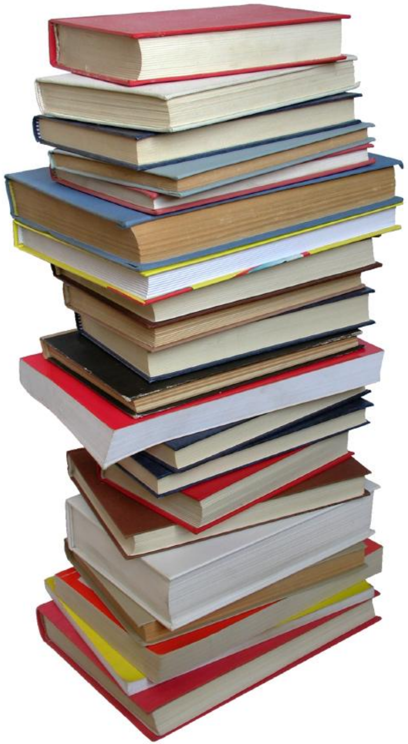
```
public interface StackADT<E> {

    public void push ( E element );

    public E pop ( );

    public E peek ( );

    public int size ( );

    public boolean isEmpty ( );

}
```

Stack: a simple linear ADT



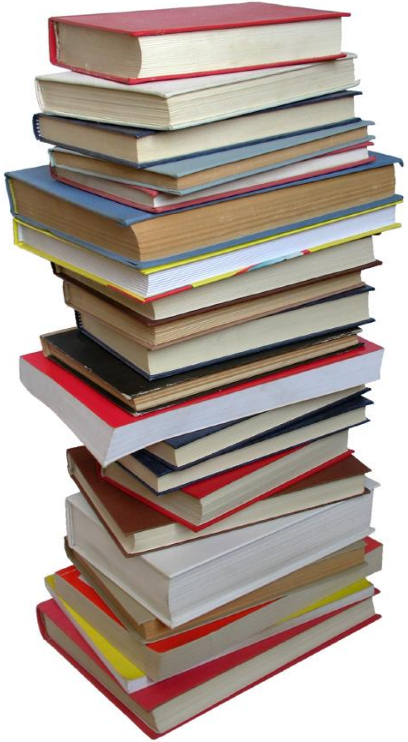
```
/** Abstract Data Type for a Stack -
 * a vertically-oriented linear data
 * structure in which elements can
 * only be added or removed from the
 * top. Also known as a "last in,
 * first out", or LIFO, structure.
*/
public interface StackADT<E> {

    ...

}
```

Stack: a simple linear ADT



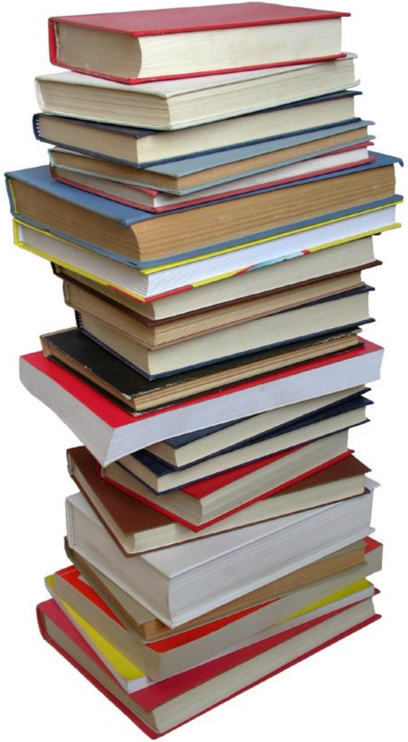
```
public interface StackADT<E> {

    /** Adds a new element to the top
     * of the Stack */
    public void push ( E element );

    /** Removes and returns the top
     * element from the Stack */
    public E pop ( );

    ...

}
```

Stack: a simple linear ADT



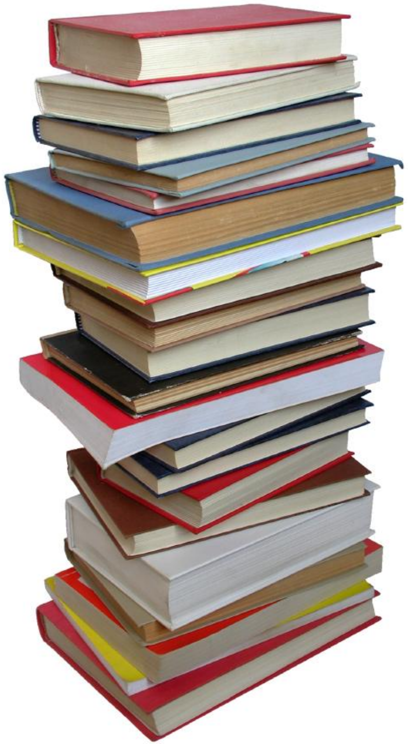
```
public interface StackADT<E> {

    ...

    /** Returns the top element of the
     * Stack, but does not remove it */
     public E peek ( );

    ...

}
```

Stack: a simple linear ADT



```
public interface StackADT<E> {

    ...

    /** Returns the number of
     * elements in the Stack */
    public int size ( );

    /** Returns true if the Stack is
     * empty, else false */
    public boolean isEmpty ( );

}
```

Stack: a simple linear ADT



```
public interface StackADT<E> {

    public void push ( E element );

    public E pop ( );

    public E peek ( );

    public int size ( );

    public boolean isEmpty ( );

}
```

Using the Stack ADT

```
StackADT<Integer> stack = new
StackImplementation<Integer>();
```

Mental Model

(empty)

Console

Using the Stack ADT

```
System.out.println ( stack.size() );
```

Mental Model	Console
(empty)	`0`

Using the Stack ADT

```
System.out.println ( stack.isEmpty() );
```

Mental Model	Console
(empty)	`true`

Using the Stack ADT

```
stack.push ( 1 );
```

Mental Model	Console
1	

Using the Stack ADT

```
stack.push ( 2 );
```

Mental Model

2
1

Console

Using the Stack ADT

```
stack.push ( 3 );
```

Mental Model

3

2

1

Console

Using the Stack ADT

```
System.out.println ( stack.size() );
```

Mental Model	**Console**
3 2 1	3

Using the Stack ADT

```
System.out.println ( stack.isEmpty() );
```

Mental Model

3
2
1

Console

```
false
```

Using the Stack ADT

```
System.out.println ( stack.peek() );
```

Mental Model	**Console**
3 2 1	3

Using the Stack ADT

```
System.out.println ( stack.pop() );
```

Mental Model

2
1

Console

3

Using the Stack ADT

```
System.out.println ( stack.pop() );
```

Mental Model	**Console**
1	2

Using the Stack ADT

```
System.out.println ( stack.pop() );
```

Mental Model	Console
(empty)	1

Queue: another linear ADT

Have you ever stood in line?

Then you have a good mental model for the Queue ADT.

Queue: another linear ADT

Adding to the Queue:

- void add (E element)
- void offer (E element)
- void enqueue (E element)

Removing from the Queue:

- E remove ( )
- E poll ( )
- E dequeue ( )

Examining the first element:

- E element ( )
- E peek ( )
- E first ( )

Current size:

- int size ( )

Is it empty?

- boolean isEmpty ( )

Queue: another linear ADT

```
public interface QueueADT<E> {

    public void enqueue ( E element );

    public E dequeue ( );

    public E first ( );

    public int size ( );

    public boolean isEmpty ( );

}
```

Using the Queue ADT

```
QueueADT<Integer> queue = new
QueueImplementation<Integer>();
```

Mental Model	**Console**
(empty)	

Using the Queue ADT

```
queue.enqueue ( 1 );
```

Mental Model

(front) 1 (rear)

Console

Using the Queue ADT

```
queue.enqueue ( 2 );
```

Mental Model

(front) 1 2 (rear)

Console

Using the Queue ADT

```
queue.enqueue ( 3 );
```

Mental Model

(front) 1 2 3 (rear)

Console

Using the Queue ADT

```
System.out.println ( queue.first ( ) );
```

Mental Model

(front) 1 2 3 (rear)

Console

1

Using the Queue ADT

```
System.out.println ( queue.dequeue ( ) );
```

Mental Model	Console
(front) 2 3 (rear)	1

Using the Queue ADT

```
System.out.println ( queue.dequeue ( ) );
```

Mental Model	Console
(front) 3 (rear)	2

Using the Queue ADT

```
System.out.println ( queue.dequeue ( ) );
```

Mental Model	Console
(empty)	3

List: flexible, general-purpose linear ADT

We’re all familiar with the idea of lists, but there are different kinds of lists, for different purposes. Three common types:

- Ordered - always in some inherent order - automatic insertion in the right location
- Unordered - no inherent order - can work from either end, insert after a known element, or remove by identity
- Indexed - directly access elements by their location in the list - e.g. “what is the 5th element?”

List: Common Operations

```
public interface ListADT<E> {
   public E remove ( E element );
   public E removeFirst ( );
   public E removeLast ( );
   public E first ( );
   public E last ( );
   public boolean contains ( E element );
   public int size ( );
   public boolean isEmpty ( );
}
```

List: Ordered List

```
/** A List ADT that maintains its own
 * inherent order. */
public interface OrderedListADT<E>
   extends ListADT<E> {

   /** Inserts element into its correct
    * position in the ordered list */
   public void add ( E element );

}
```

List: Unordered List

```java
/** A List ADT where elements can be added
 * or removed from either end, and inserted
 * after an element already in the list.
*/
public interface UnorderedListADT<E>
   extends ListADT<E> {

   ...

}
```

List: Unordered List

```
public interface UnorderedListADT<E> extends ListADT<E> {

    /** Adds element to front of the list */
    public void addToFront ( E element );

    /** Adds element to rear of the list */
    public void addToRear ( E element );

    /** Inserts element after target */
    public void addAfter ( E element, E target );

}
```

List: Indexed List

```java
/** A List ADT where elements are accessible by index
 * position, beginning with 0 and ending with size() - 1
*/
public interface IndexedListADT<E> extends ListADT<E> {

    /** Adds element at index */
    public void add ( int index, E element );

    /** Removes and returns element at index */
    public E remove ( int index );

    ...

}
```

List: Indexed List

```
public interface IndexedListADT<E> extends ListADT<E> {

    ...

    /** Replace element at index */
    public void set ( int index, E element );

    /** Get element at index. Does not remove element. */
    public E get ( int index );

    ...

}
```

List: Indexed List

```
public interface IndexedListADT<E> extends ListADT<E> {

    ...

    /** Returns index where matching element is found
     * or -1 if the element is not found in the list */
    public int indexOf ( E element );

}
```

List: blended

Ordered Lists are usually custom-created for a specific application, so general-purpose Ordered Lists aren't in common use.

Most often, people want a blend of Unordered List and Indexed List functionality. Using the Interfaces previously defined, we could get that combination with:

```
public interface IndexedUnorderedListADT<E>
    extends IndexedListADT<E>, UnorderedListADT<E> { }
```

Abstract Data Types (Linear)

Mason Vail
Boise State University Computer Science