KEY POINTS

Abstract Data Type:

- WHAT NOT HOW for each data type.
- Uses an interface in Java → Exceptions

NAME/DATE/SUBJECT

ADT

NOTES

Abstract Data Types:

- An ADT is an abstraction of a data type. It specifies
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- Focuses on the WHAT of each implementation-not the how.
- Expressed by an interface in Java.

EXAMPLE: Stock trading

- Stores buy / sell orders
- Several operations
  - buy
  - sell
  - cancel
- Error conditions
  - Buy /sell non existent stock
  - Cancel an order that does not exist.

SUMMARY

An ADT is a way to blueprint a data type, and can be defined in Java using an Interface.

NAME/DATE/SUBJECT

Stack ADT

KEY POINTS

- LIFO
- push, pop

Auxiliary operations:

- top
- size
- isEmpty

Java

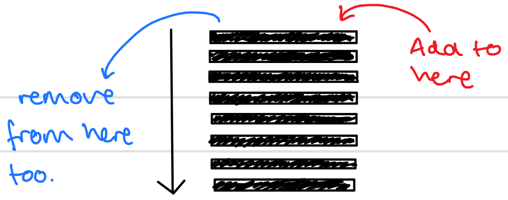
- java.util.Stack
- Exceptions → null
- JVM method stack
- FullStackException

Array based:

- $O(n)$ space
- operations have $O(1)$ time
- fixed space + impl. specific exception.

NOTES

What is a stack?



A collection of objects that obey the Last in First out (LIFO) principle.

If we try to top() or pop an empty stack, then we typically return null.

e.g. reversing an array by placing items on a stack

Typical applications: (Direct) web history, undo sequence
(Indirect) Aux. DS for algorithms.

The JVM method stack uses frames to hold local variables and return values, and a program counter. When a method ends, its frame will be popped from the stack; control passes to the top frame which allows for recursion.

For an array-based stack, pushing to a full stack will cause a FullStackException.

SUMMARY

- What is an ADT?
- What is the stack ADT?
  - ↳ fundamental DS
  - ↳ array implementation
- Algorithms can use stacks
  - ↳ Reversing an array
  - ↳ Parenthesis matching