Lecture 4: Stacks and Queues

06/29/22

A1: LetterInventory

- Due Thursday 6/30 @ 11:59pm

- To be making **satisfactory** progress in the course, your homework should pass all the test cases on Ed.

Abstract Data Type

Abstract Data Type (ADT)

- Composed of:
  - A collection of data
  - The operations that can be performed on that Data
- Describes what a collection does, not how it does it
- Not specific to Java!

Interface

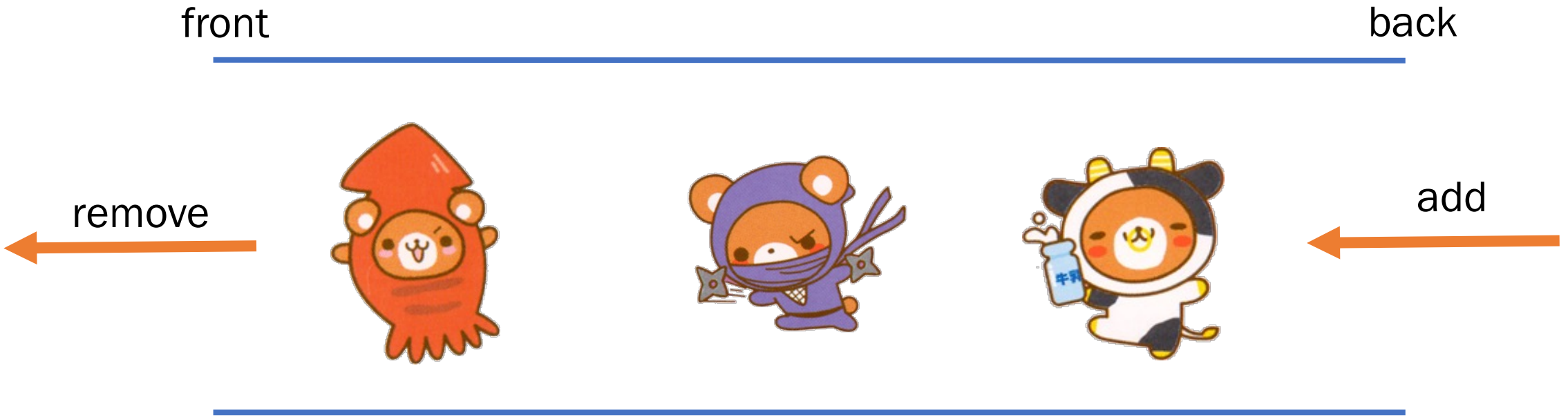
- Java’s way of representing an Abstract Data Type
- Describes all the methods a class must have in order to be that data type
- Doesn’t implement the methods
  - A class with all the guts ripped out

2 New Abstract Data Types!

- queue
  - line at a grocery store

- stack
  - stack of cafeteria trays

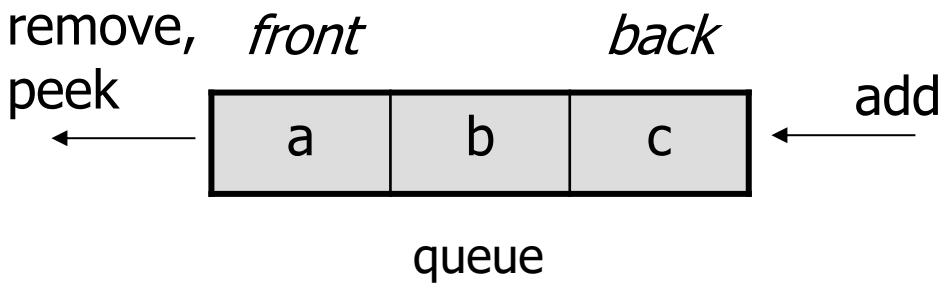
Queue Example



Queue ADT

- **Queue**: First-In, First-Out ("FIFO")
  - No indices

- basic queue operations:
  - **add** (enqueue): Add an element to the back.
  - **remove** (dequeue): Remove the front element.
  - **peek**: Examine the front element.



Queues in Computer Science

- Operating systems:
  - queue of print jobs to send to the printer
  - queue of programs / processes to be run
- Programming:
  - modeling a line of customers or clients
  - storing a queue of computations to be performed in order
- Real world examples:
  - people on an escalator or waiting in a line
  - cars at a gas station (or on an assembly line)

`Queue` Interface in Java

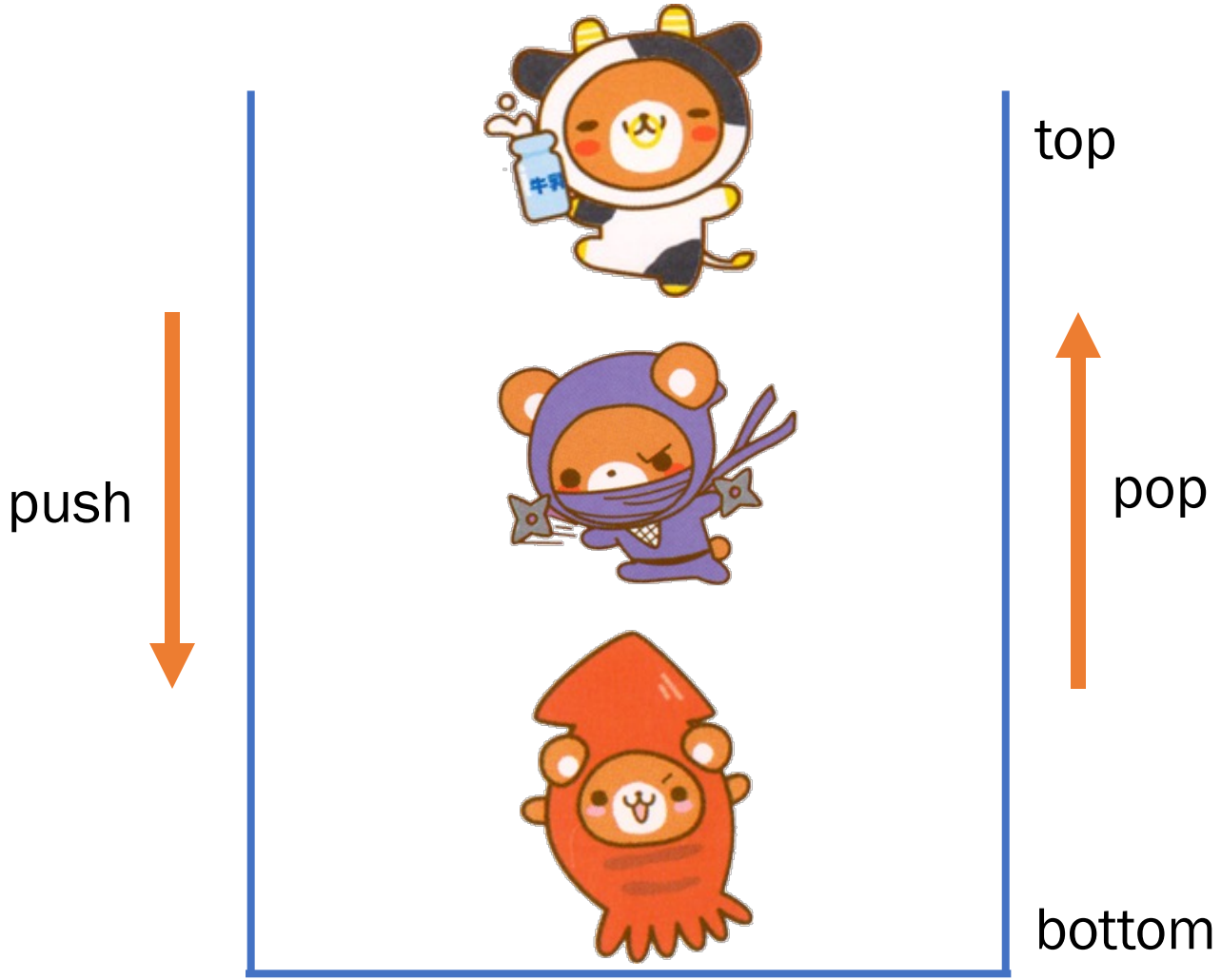
`add(`**value**`)`	places given value at back of queue
`remove()`	removes value from front of queue and returns it; throws a `NoSuchElementException` if queue is empty
`peek()`	returns front value from queue without removing it; returns `null` if queue is empty
`size()`	returns number of elements in queue
`isEmpty()`	returns `true` if queue has no elements

`Queue` has other methods that are off-limits (not efficient)

```
Queue<String> q = new LinkedList<>();
```

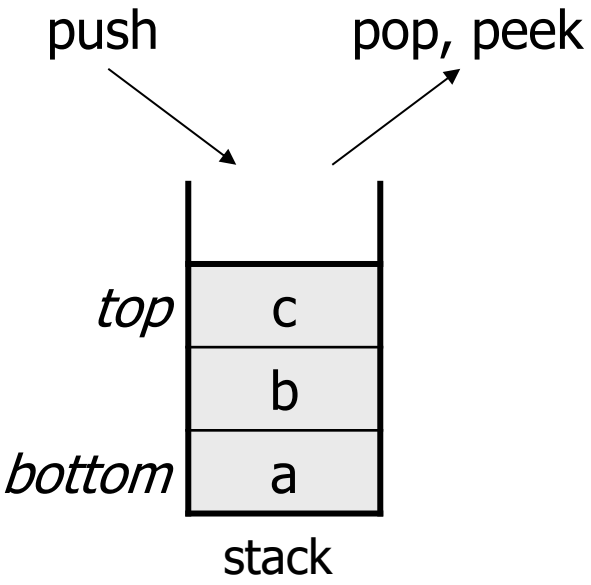
`LinkedList` implements the `Queue` interface!

Stack Example



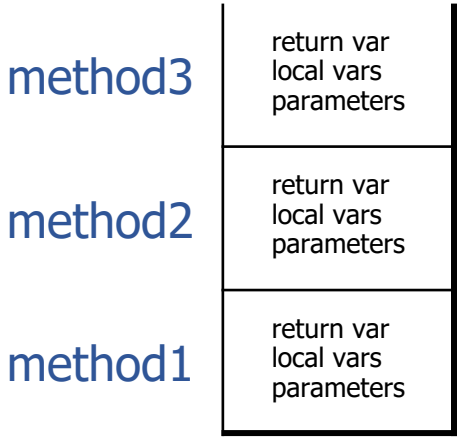
Stack ADT

- **Stack**: Last-In, First-Out ("LIFO")
  - No indices
- basic stack operations:
  - **push**: Add an element to the top.
  - **pop**: Remove the top element.
  - **peek**: Examine the top element.



Stacks in Computer Science

- Programming languages and compilers:
  - method calls are placed onto a stack *(call=push, return=pop)*
  - compilers use stacks to evaluate expressions
- Matching up related pairs of things:
  - examine a file to see if its braces { } match
  - convert "infix" expressions to pre/postfix
- Sophisticated algorithms:
  - searching through a maze with "backtracking"
  - many programs use an "undo stack" of previous operations



`Stack` Class in Java

`Stack<`**E**`>()`	constructs a new stack with elements of type **E**
`push(`**value**`)`	places given value on top of stack
`pop()`	removes top value from stack and returns it; throws `EmptyStackException` if stack is empty
`peek()`	returns top value from stack without removing it; throws `EmptyStackException` if stack is empty
`size()`	returns number of elements in stack
`isEmpty()`	returns `true` if stack has no elements

`Stack` has other methods that are off-limits (not efficient)

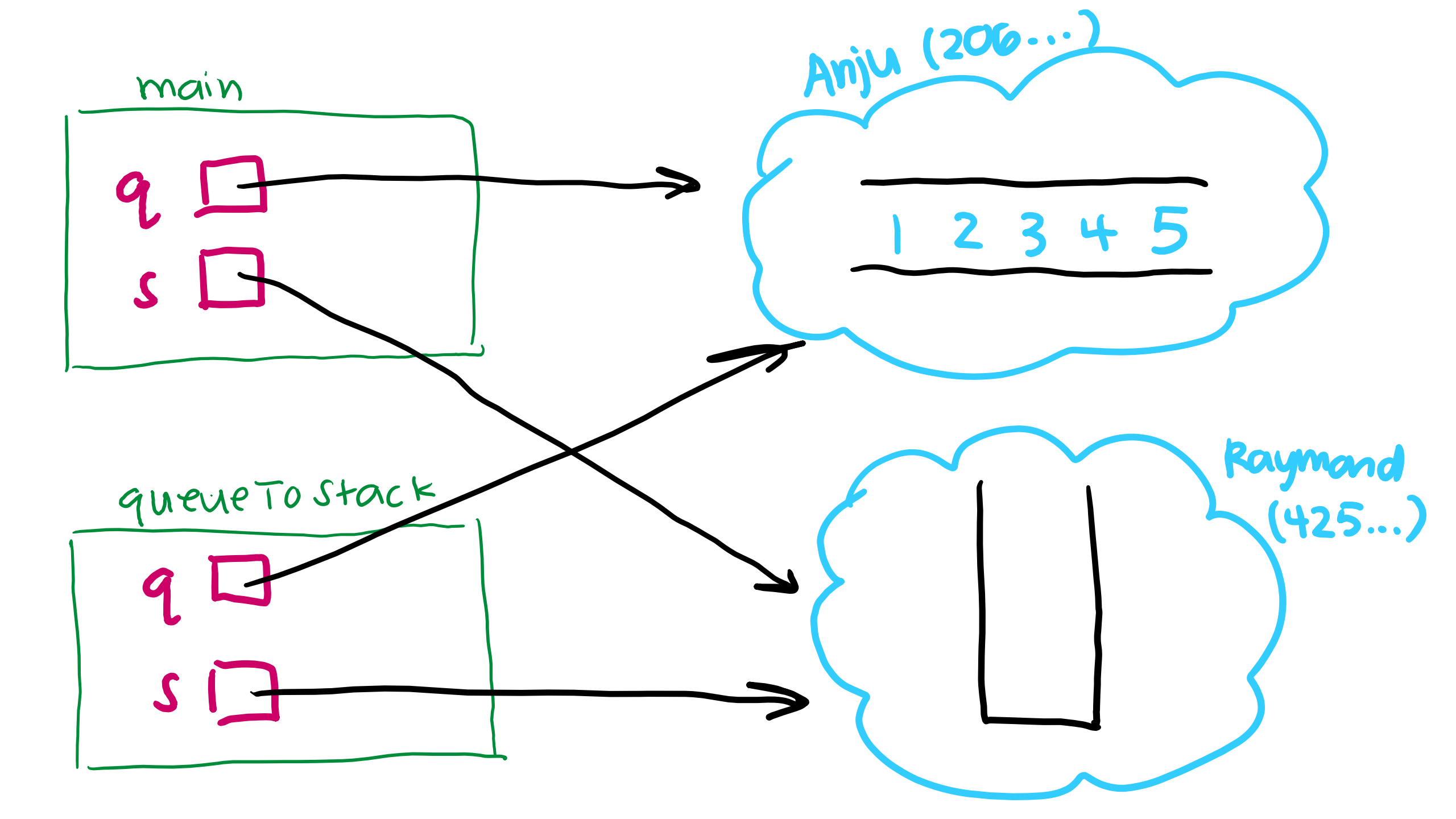
```
Stack<String> s = new Stack<>();
```

Java messed up, there is no `Stack` interface ☹

Misc. Notes

- Lecture and section problems are brainteasers, not great applications of stacks and queues
  - Practice problem solving!

- (Reminder: Exam problems are exactly like section problems! We're not trying to surprise you)
  - peek() method isn't allowed on exam/section questions


main
q
s
Anju (206...)
1 2 3 4 5
queueToStack
q
s
Raymond
(425...)