Poll Everywhere

PollEv.com/javabear text **javabear** to 22333



Recall: Our previous `censor()` definition had an $O(N^2)$ runtime.

What lambda expression can we pass into `transformAll()` to satisfy the `censor()` specs and achieve an $O(N)$ runtime?

```
/** Replaces all instances of the given `word` with "****" in these `lyrics`. */
static void censor(CS2110List<String> lyrics, String word) {
  lyrics.transformAll( s -> s.equals(word) ? "****" : s );
}
```

Recall:

```
@FunctionalInterface
public interface Transformation<T> {
  T transform(T input);
}
```



Lecture 15: Stacks and Queues

CS 2110

October 15, 2026

Today's Learning Outcomes

58. Describe the semantics of Stacks and Queues, and utilize them within programs.

59. Describe composition relationships and scenarios where they may be preferable to inheritance.

52. Compare the behaviors, memory utilization, and performance of ordinary Java arrays and Lists backed by dynamic arrays and linked chains. Determine which is preferable for a given use case.

Processing Incoming Data

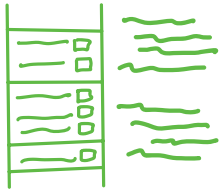
Many applications involve gathering up a lot of data and then systematically processing its elements one at a time.



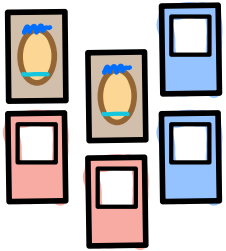
Sensor Readings



Customer Transactions



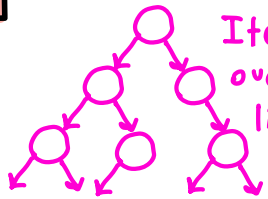
Code Instructions across multiple method calls



Mechanics in deck-building games



Medical Triage (coming soon)

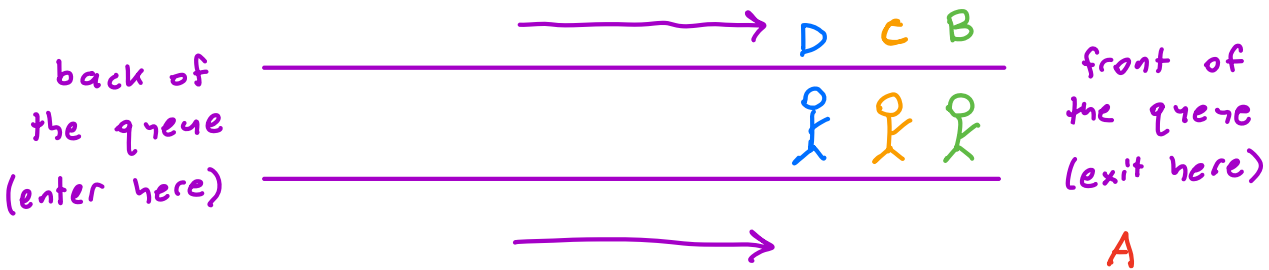


Iterating over non-linear linked data structures

Ordered collections allow retrieval of their elements in one pre-determined order.

Queues and FIFO Ordering

people waiting in line



- anyone can join the queue at any time
- the only person who can exit the queue (be served next) is the person at the front, who has been there the longest

Queues enforce "First in, First out" (FIFO) order condition.

The Queue<T> ADT

Like all "ordered collections", Queues support four operations:

1. **Add another element**

```
/** Adds to back of queue */
void enqueue(T elem);
```

2. **Remove/return "next" element**
   (ADT determines which is "next")

```
/** Removes + returns front element */
T dequeue();
```

3. **Return "next" element**
   (without removing it)

```
/** Returns front element */
T peek();
```

4. **Check if there are more elements**

```
boolean isEmpty();
```

Poll Everywhere

PollEv.com/javabear text `javabear` to 22333



If we model a Queue with a `SinglyLinkedList` which state representation is preferable?

deleting from end of SLL is O(N) ... avoid it

head = back of the queue	(A)
head = front of the queue	(B)
both options work equally well	(C)



Coding Demo: Queues via Inheritance

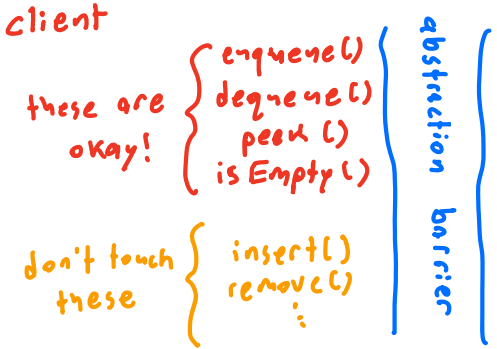


The Problem with Inheritance

When `class B extends class A`, B's client interface automatically (and non-negotiably) includes every method in A's client interface.

Sometimes, this exposes excess functionality to the client that lets them circumvent B's specs.

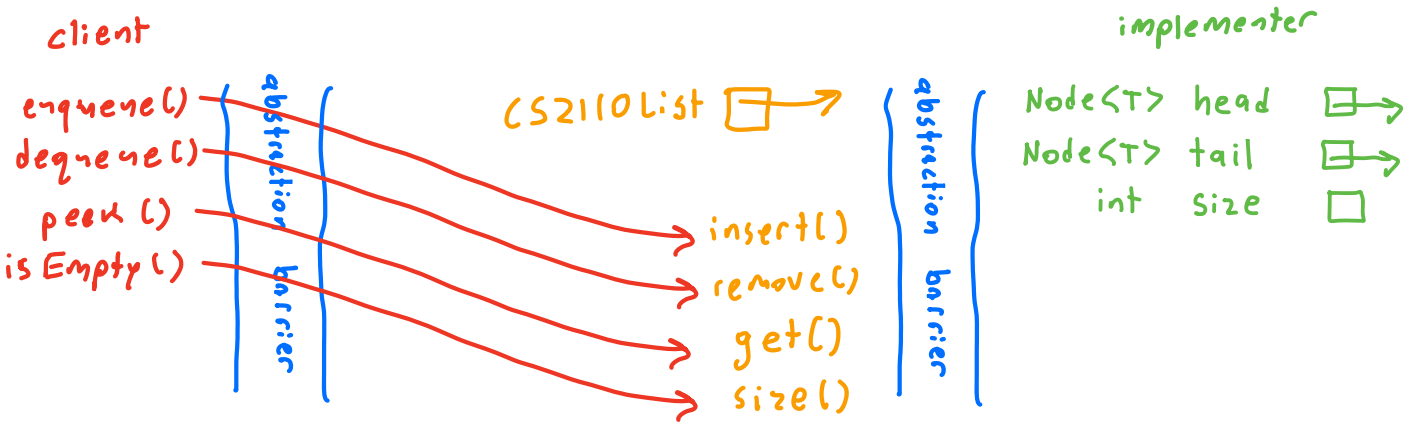
- insert(), remove(), etc. break Queue's FIFO guarantee



Composition Relationships

Idea: "Move" some methods behind the abstraction barrier to get them out of view of the client.

Queue class manages list as a (private) field. Without inheritance, it can decide on its own client interface.





Coding Demo: Queues via Composition



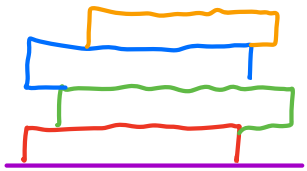
Queue Operation Complexity

	enqueue()	dequeue()	peek()	isEmpty()
SLL	O(1)	O(1)	O(1)	O(1)
DAL	O(N)*	O(1)	O(1)	O(1)

* Can improve push() to amortized O(1) using a dynamic circular buffer (see notes)

Stacks and LIFO Ordering

When we stack up objects (e.g., books) we only ever have direct access to the one on top.



Top book = most recent one added to stack

Stacks enforce a "Last in, First out" (LIFO) order condition

The runtime stack is a stack of call frames

The currently executing method is at top of the stack and will be the first to be removed when it finishes executing

The Stack<T> ADT

Like all "ordered collections", Stacks support four operations:

1. **Add another element**

```
/** Adds to top of stack */
void push(T elem);
```

2. **Remove/return "next" element**
   (ADT determines which is "next")

```
/** Removes + returns top element */
T pop();
```

3. **Return "next" element**
   (without removing it)

```
/** Returns top element */
T peek();
```

4. **Check if there are more elements**

```
boolean isEmpty();
```

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333



If we implement a `Stack` via composition with a `DynamicArrayList` which state representation is preferable?

adding / removing from beginning of DAL is O(N)

- index 0 = top of stack **(A)**
- index 0 = bottom of stack **(B)**
- both options work equally well **(C)**

Poll Everywhere

PollEv.com/javabear text `javabear` to 22333



If we implement a `Stack` via composition with a `SinglyLinkedList` which state representation is preferable?

deleting from end of SLL is O(N) ... avoid it

- head = top of the stack (A)
- head = bottom of the stack (B)
- both options work equally well (C)



Coding Demo: Implementing Stacks



Stack Operation Complexity

	push()	pop()	peek()	isEmpty()
SLL	O(1)	O(1)	O(1)	O(1)
DAL	Amortized O(1)	O(1)	O(1)	O(1)

Stack Application: Expression Parsing

Given a `String` representing a mathematical expression, can we determine its value (in $O(N)$ time)?

"3 * (6+2)"

Idea: Read characters one at a time L→R and keep track of "state" of parsing as we go

Potential Issue: May need to keep track of arbitrarily large state

"2 * (3 + 4 * (5 + 6 * (7+8)))

order of these operations: 6 5 4 3 2 1

simplification happens R→L, so we can use stacks to keep track of state

Operator and Operand Stacks

Idea: Maintain two stacks, one for `operands` and one for `operators`, then use the order of operations to figure out when to simplify.

3 * 4 + 6 * (2 + 7) * 5

*see lecture notes for an animation

Next symbol:

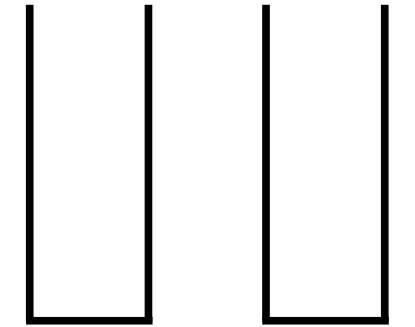
→ push onto operands

* → simplify any * atop the operators stack
then push '*' onto operators

+ → simplify any +, * atop the operators stack
then push '+' onto operators

(→ push onto operators

) → simplify operators until we find matching '('
then pop '('



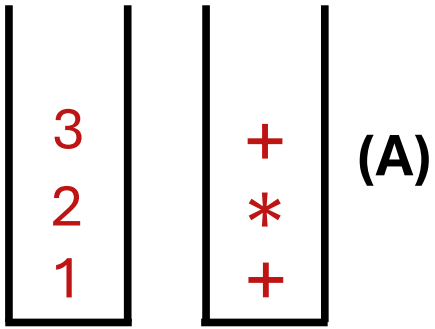
Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

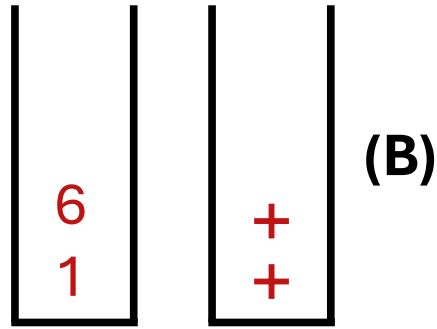
Which of the following depicts our parser state just after parsing the second "+" in "1+2*3+4" ?

left associativity

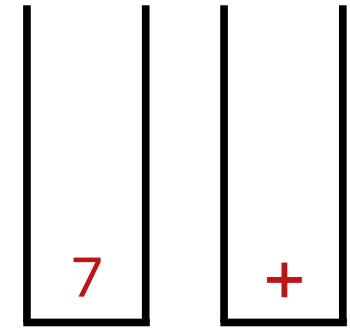
higher precedence



(A)



(B)



(C)



Coding Demo: `ExpressionEvaluator`

