

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.

Queues and FIFO Ordering

The Queue<T> ADT

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

1. **Add another element**

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

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

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



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.

Composition Relationships

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



Coding Demo: Queues via Composition



Queue Operation Complexity

	enqueue()	dequeue()	peek()	isEmpty()
SLL				
DAL				

Stacks and LIFO Ordering

The Stack<T> ADT

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

1. **Add another element**

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

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

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



Coding Demo: Implementing Stacks



Stack Operation Complexity

	`push()`	`pop()`	`peek()`	`isEmpty()`
SLL				
DAL				

Stack Application: Expression Parsing

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

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

Next symbol:

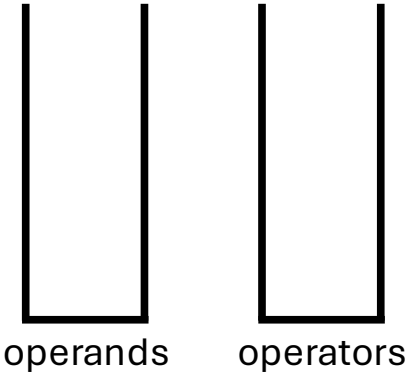
→

* →

\+ →

(→

) →





Coding Demo: `ExpressionEvaluator`

