Talk to your neighbors:

Dogs or cats?

Instructor: Melissa Lin
TAs:
- Poojitha Arangam
- Darel Gunawan
- Colton Harris
- Atharva Kashyap
- Eesha Kunisetty

Audrey Lin
- Di Mao
- Steven Nguyen
- Ben Wang
- Jaylyn Zhang
Lecture Outline

• Announcements

• Review: ADTs, Stacks & Queues

• Queue Manipulation

• Stack Manipulation

• Problem Solving
Announcements

• Quiz 0 next Monday (July 10th)
• Resub 0 (R0) due tonight
  - P0 grades will be released today, so you technically can resubmit
• Creative Project (C0) due tomorrow
• Programming Assignment 1 (P1) will be released Friday
  - It will be due next Thursday (July 13)
Lecture Outline

• Announcements
• Review
• Queue Manipulation
• Stack Manipulation
• Problem Solving
Abstract Data Types

- **Abstract Data Type (ADT)**: A specification of a collection of data and the operations that can be performed on it.
  - Describes what a collection does, not how it does it

- We don't know exactly how a stack or queue is implemented, and we don't need to!
  - Only need to understand high-level idea of what a collection does and its operations in order to use them

  - **Stack**: retrieves elements in reverse order as added.
    Operations: push, pop, peek, ...
  
  - **Queue**: retrieves elements in same order as added.
    Operations: add, remove, peek, ...
(PCM) Abstract Data Types

**Abstract Data Types**

- **ADT**
  - Examples: queue, stack, list

- **Interface**
  - Examples: Queue<>, List<>  

- **Implementation**
  - Examples: ArrayList, LinkedList, array, Stack

---

**Language specific**

- Language specific

**Language agnostic**

- Language agnostic

---

**More abstract**

- More abstract

---

**More specific**

- More specific
Stack - What is it good for?

What is it?

- A **Last-in-First-out (LIFO)** data structure
  - Elements are removed in the **reverse order** to how they were added
- All elements must be of the same type*
- Dynamically sized

What is Stack particularly good at?

- **push** - add element to top
- **pop** - remove element from top
- Supported operations are few but **very efficient**
(PCM) Stacks

- **push**
- **pop**
- **top**
- **bottom**
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:
  - find out whether a string is a palindrome
  - 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
### Programming with Stacks

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack&lt;(E)&gt;()</td>
<td>constructs a new stack with elements of type (E)</td>
</tr>
<tr>
<td>push(value)</td>
<td>places given value on top of stack</td>
</tr>
<tr>
<td>pop()</td>
<td>removes top value from stack and returns it; throws EmptyStackException if stack is empty</td>
</tr>
<tr>
<td>peek()</td>
<td>returns top value from stack without removing it; throws EmptyStackException if stack is empty</td>
</tr>
<tr>
<td>size()</td>
<td>returns number of elements in stack</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>returns true if stack has no elements</td>
</tr>
</tbody>
</table>

```java
Stack<String> s = new Stack<String>();
s.push("a");
s.push("b");
s.push("c"); // bottom ["a", "b", "c"] top
System.out.println(s.pop()); // "c"
```

- Stack has other methods that we will ask you not to use
Queue - What is it good for?

What is it?

- A **First**-in-First-out (FIFO) data structure
  - Elements are removed in the **same order** to how they were added
- All elements must be of the same type*
- Dynamically sized

What is Queue particularly good at?

- **add** - add element to back
- **remove** - remove element from front
- Supported operations are few but **very efficient**
(PCM) Queue

remove

---

front

add

back
Queues in Computer Science

• Operating systems:
  - queue of print jobs to send to the printer
  - queue of programs / processes to be run
  - queue of network data packets to send

• 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)
Programming with Queues

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(value)</td>
<td>places given value at back of queue</td>
</tr>
<tr>
<td>remove()</td>
<td>removes value from front of queue and returns it; throws a NoSuchElementException if queue is empty</td>
</tr>
<tr>
<td>peek()</td>
<td>returns front value from queue without removing it; returns null if queue is empty</td>
</tr>
<tr>
<td>size()</td>
<td>returns number of elements in queue</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>returns true if queue has no elements</td>
</tr>
</tbody>
</table>

Queue<Integer> q = new LinkedList<Integer>();
q.add(42);
q.add(-3);
q.add(17); // front [42, -3, 17] back
System.out.println(q.remove()); // 42

- IMPORTANT: When constructing a queue you must use a new LinkedList object instead of a new Queue object.
Lecture Outline

• Announcements

• Review

• **Queue Manipulation**

• Stack Manipulation

• Problem Solving
Lecture Outline

• Announcements
• Review
• Queue Manipulation
• Stack Manipulation
• Problem Solving
What is the return of this method?

```java
// numbers: bottom [1, 2, 3, 4, 5] top
public static int sum(Stack<Integer> numbers) {
    int total = 0;
    for (int i = 0; i < numbers.size(); i++) {
        int number = numbers.pop();
        total += number;
        numbers.push(number);
    }
    return total;
}
```

A) 0  
B) 1  
C) 5  
D) 15  
E) 25  
F) Throws an error
What is the return of this method?

```java
// numbers: bottom [1, 2, 3, 4, 5] top
public static int sum(Stack<Integer> numbers) {
    int total = 0;
    for (int i = 0; i < numbers.size(); i++) {
        int number = numbers.pop();
        total += number;
        numbers.push(number);
    }
    return total;
}
```

A) 0  
B) 1  
C) 5  
D) 15  
E) 25  
F) Throws an error
What is the return of this method?

```java
// numbers: bottom [1, 2, 3, 4, 5] top
public static int sum(Stack<Integer> numbers) {
    Queue<Integer> q = new LinkedList<>();

    int total = 0;
    for (int i = 0; i < numbers.size(); i++) {
        int number = numbers.pop();
        total += number;
        q.add(number);
    }

    return total;
}
```

A) 0  
B) 1  
C) 5  
D) 12  
E) 15  
F) Throws an error
What is the return of this method?

```java
// numbers: bottom [1, 2, 3, 4, 5] top
public static int sum(Stack<Integer> numbers) {
    Queue<Integer> q = new LinkedList<>();

    int total = 0;
    for (int i = 0; i < numbers.size(); i++) {
        int number = numbers.pop();
        total += number;
        q.add(number);
    }

    return total;
}
```

A) 0  
B) 1  
C) 5  
D) 12 
E) 15 
F) Throws an error
Stack Sum bug

// numbers: bottom [1, 2, 3, 4, 5] top
public static int sum(Stack<Integer> numbers) {
    Queue<Integer> q = new LinkedList<>();

    int total = 0;
    for (int i = 0; i < numbers.size(); i++) {
        int number = numbers.pop();
        total += number;
        q.add(number);
    }

    // Still need to move back to the stack!
    return total;
}

// Loop Table

<table>
<thead>
<tr>
<th>i</th>
<th>total</th>
<th>numbers</th>
<th>numbers.size()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>[4, 3, 2, 1]</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>[3, 2, 1]</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>[2, 1]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>[1]</td>
<td>2</td>
</tr>
</tbody>
</table>

Exit the loop!!
Lecture Outline

• Announcements
• Review
• Queue Manipulation
• Stack Manipulation
• Problem Solving
Problem Solving

• On their own, Stacks & Queues are quite simple with practice (few methods, simple model)

• Some of the problems we ask are complex because the tools you have to solve them are restrictive
  - sum(Stack) is hard with a Queue as the auxiliary structure

• We challenge you on purpose here to practice problem solving

Common Problem-Solving Strategies

• **Analogy** – Is this similar to a problem you’ve seen?
  - `sum(Stack)` is probably a lot like `sum(Queue)`, start there!

• **Brainstorming** – Consider steps to solve problem before writing code
  - Try to do an example “by hand” → outline steps

• **Solve Sub-Problems** – Is there a smaller part of the problem to solve?
  - Move to queue first

• **Debugging** – Does your solution behave correctly on the example input.
  - Test on input from specification
  - Test edge cases ("What if the Stack is empty?")

• **Iterative Development** – Can we start by solving a different problem that is easier?
  - Just looping over a queue and printing elements
Common Stack & Queue Patterns

• Stack → Queue and Queue → Stack
  - We give you helper methods for this on problems

• Reverse a Stack with a S→Q + Q→S

• “Cycling” a queue: Inspect each element by repeatedly removing and adding to back size times
  - Careful: Watch your loop bounds when queue’s size changes

• A ”splitting” loop that moves some values to the Stack and others to the Queue
See you Friday!

- Practice with Stacks & Queues in Section
- Quiz on Monday (July 10th)
- Challenge problem in lecture on Friday
- P1, released Friday, will use Stacks & Queues
- Remember to do the PCM for Friday!