LEC 07

CSE 122

Stacks & Queues Practice

BEFORE WE START

Talk to your neighbors:

If you were an herb/seasoning, what would you be?

Instructors: Ido Avnon

TAs: Abby Williams
Chloë Mi Cartier
Connor Sun
Cynthia Pan
Katharine Zhang
Marcus Sanches
Rohini Arangam

Questions during Class?

Raise hand or send here

sli.do #cse122

Lecture Outline

- **Announcements**
- Quick Recap
- `copyStack` Review
- Exceptions
- Structured Example: `spliceStack`

Announcements

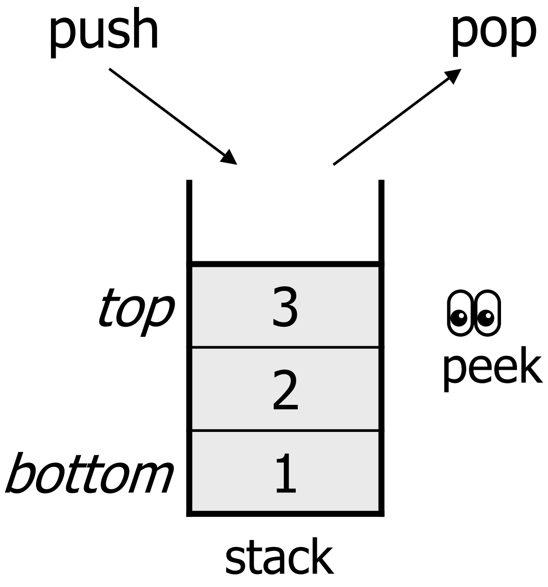
- Creative Project 1 was due yesterday, how’d it go?
- Programming Assignment 1 releasing later tonight
    - Focusing on Stacks and Queues
- Resubmission Cycle 1 form posted
    - Due July 16 by 11:59pm
    - Eligible assignments: P0

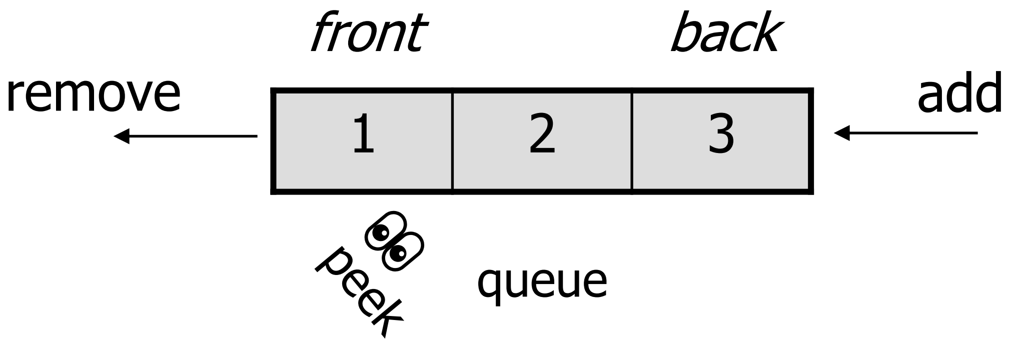
Lecture Outline

- Announcements
- **Quick Recap** ◀
- `copyStack` Review
- Exceptions
- Structured Example: `spliceStack`

Stacks & Queues

- Some collections are constrained, only use optimized operations
  - **Stack:** retrieves elements in reverse order as added
  - **Queue:** retrieves elements in same order as added





Common Stack & Queue Patterns

- 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

Lecture Outline

- Announcements
- Quick Recap
- **copyStack Review**
- Exceptions
- Structured Example: `spliceStack`

`copyStack`

Write a method `copyStack` that takes a stack of integers as a parameter and returns a copy of the original stack (i.e., a new stack with the same values as the original, stored in the same order as the original).

You may use one queue as auxiliary storage.

Ido’s First Try

```
public static Stack<Integer> copyStack(Stack<Integer> s) {
     return s;
}
```

Lecture Outline

- Announcements
- Quick Recap
- `copyStack` Review
- **Exceptions**
- Structured Example: `spliceStack`

Exceptions

- Sometimes we want to limit someone’s input into our method to “valid” options we define
  - Previously printed out “hey don’t do that” messages which isn’t great…

- Allow us to “fail fast” and immediately halt execution
- No longer need to wrap code in conditionals
- Can include custom error messages about what went wrong

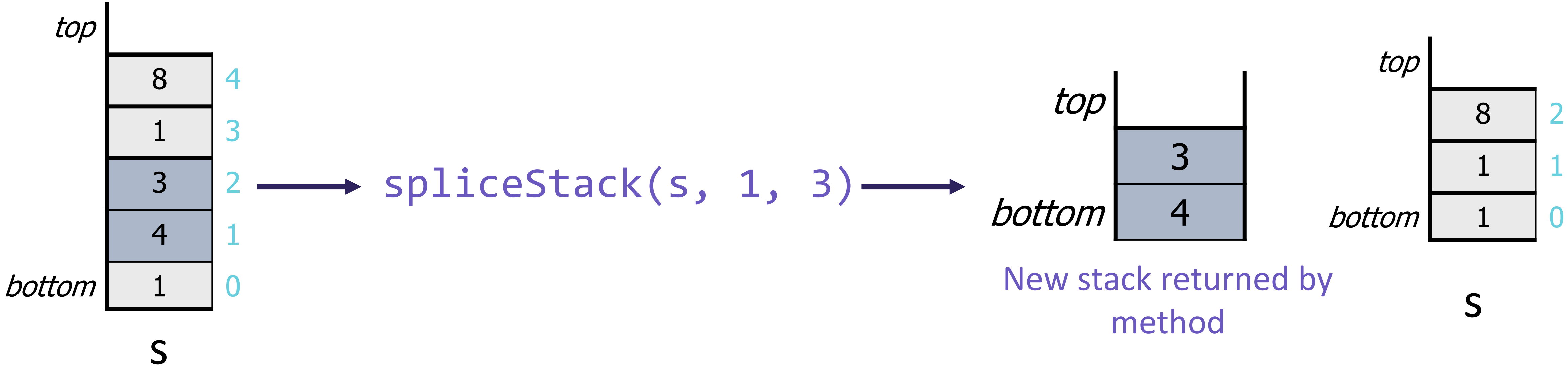
```
if (/* invalid input */) {
    throw new IllegalArgumentException("Error Message");
}
```

Lecture Outline

- Announcements
- Quick Recap
- `copyStack` Review
- Exceptions
- **Structured Example: `spliceStack`**

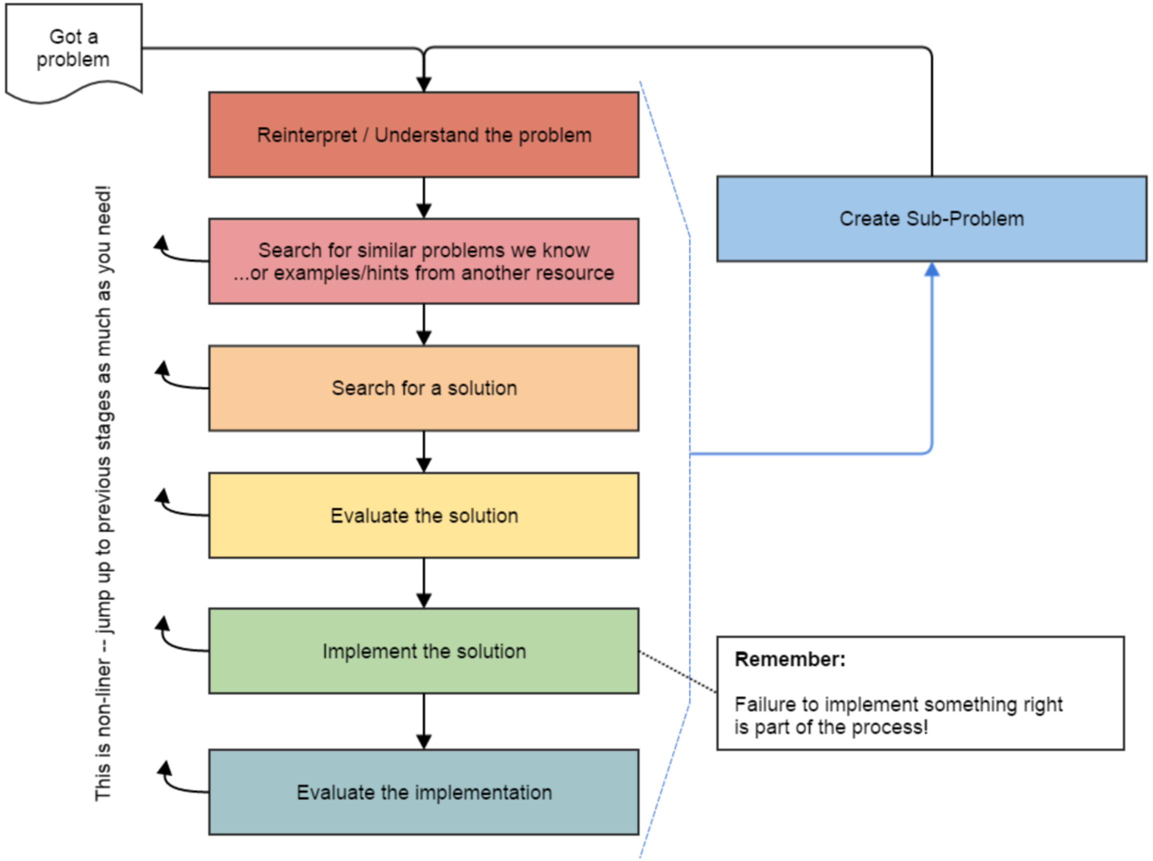
spliceStack

Write a method called `spliceStack` that takes as parameters a stack of integers `s`, a start position `i`, and an ending position `j`, and that removes a sequence of elements from `s` starting at the `i`’th element from the bottom of the stack up to (but not including) the `j`’th element from the bottom of the stack (where position 0 is the bottom of the stack), returning these values in a new stack. The ordering of elements in both stacks should be preserved.



Fundamental Data Structures ➔ 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**


Got a problem
Reinterpret / Understand the problem
Search for similar problems we know ...or examples/hints from another resource
Search for a solution
Evaluate the solution
Implement the solution
Evaluate the implementation
Create Sub-Problem
This is non-liner -- jump up to previous stages as much as you need!
Remember:
Failure to implement something right is part of the process!

Source: Oleson, Ko (2016) - Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance

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

Metacognition

- **Metacognition**: asking questions about your solution process.

- Examples:
  - **While debugging**: explain to yourself why you're making this change to your program.
  - **Before running your program**: make an explicit prediction of what you expect to see.
  - **When coding**: be aware when you're not making progress, so you can take a break or try a different strategy.
  - **When designing**:
    - Explain the tradeoffs with using a different data structure or algorithm.
    - If one or more requirements change, how would the solution change as a result?
    - Reflect on how you ruled out alternative ideas along the way to a solution.
  - **When studying**: what is the relationship of this topic to other ideas in the course?