LEC 05

CSE 122

Stacks & Queues Practice

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What is your favorite spot on campus and why?

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
- P1 walkthrough
- Quick Recap
- copyStack Review
- Structured Example: spliceStack

Announcements

- Quiz 0 next Monday (July 10)
  - see EdStem announcement for more details
- P0 feedback was released Wednesday
  - Grade Checker
- Resub 0 grades will be released next Monday (July 10)
- Resub 1 form posted today and due next Tuesday (July 11)
- P1 released today
  - due next Thursday (July 13)

Lecture Outline

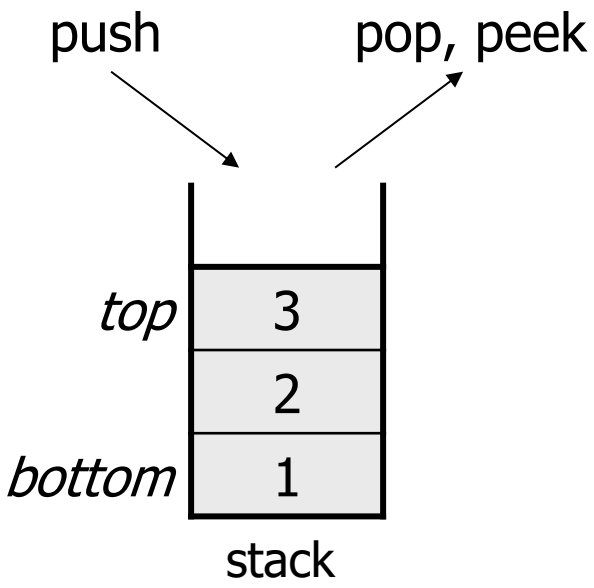
- Announcements
- **P1 walkthrough**
- Quick Recap
- copyStack Review
- Structured Example: spliceStack

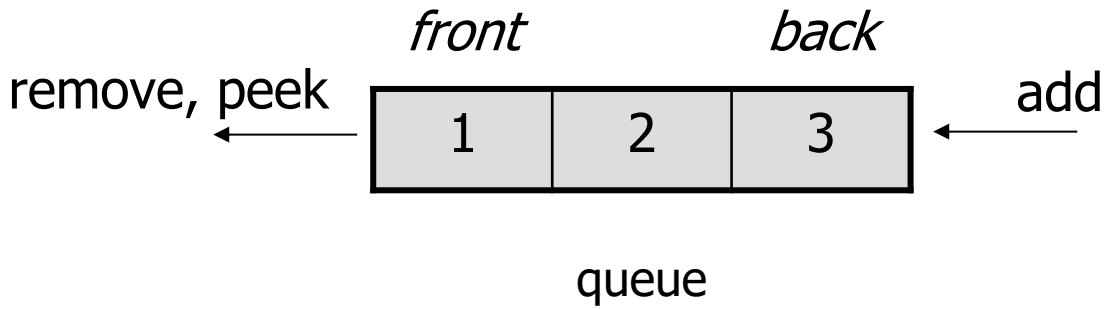
Lecture Outline

- Announcements
- P1 walkthrough
- **Quick Recap**
- copyStack Review
- Structured Example: spliceStack

(Recap) 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





(Recap) 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?

Practice : Think

sli.do #cse122

```
// s: bottom [0, 1, 2, 3, 4] top
public static void mystery(Stack<Integer> s) {
        Stack<Integer> s2 = new Stack<Integer>();
        Queue<Integer> q = new LinkedList<Integer>();

        // s -> s2
        while (!s.isEmpty()) {
                s2.push(s.pop());
        }
        // s2 -> q
        while(!s2.isEmpty()) {
                q.add(s2.pop());
        }
        // q -> s
        while (!q.isEmpty()) {
                s.add(q.remove());
        }
}
```

What does s contain after mystery finishes?

A) bottom [0, 1, 2, 3, 4] *top*

B) bottom [4, 3, 2, 1, 0] *top*

Practice : Pair

sli.do #cse122

```
// s: bottom [0, 1, 2, 3, 4] top
public static void mystery(Stack<Integer> s) {
        Stack<Integer> s2 = new Stack<Integer>();
        Queue<Integer> q = new LinkedList<Integer>();

        // s -> s2
        while (!s.isEmpty()) {
                s2.push(s.pop());
        }
        // s2 -> q
        while(!s2.isEmpty()) {
                q.add(s2.pop());
        }
        // q -> s
        while (!q.isEmpty()) {
                s.add(q.remove());
        }
}
```

What does s contain after mystery finishes?

A) bottom [0, 1, 2, 3, 4] *top*

B) bottom [4, 3, 2, 1, 0] *top*

(PCM) Common Exceptions

`IllegalArgumentException` - When a client passes in an invalid parameter. For example, if a parameter is supposed to represent a month, you might throw this exception if a client passes a negative number.

`IllegalStateException` - When the state of the program should not be possible. For example, if you had a program that a user logged into, you might throw this exception if they are ever suddenly logged out.

`FileNotFoundException` - If you try to read from/write to a file that does not exist

`IndexOutOfBoundsException` - If an invalid index is accessed

✨ Crash early ✨

Practice : Think

sli.do #cse122

Which is the most appropriate way to throw an exception for this method? The method should return the n'th element from the top of a Stack.

A

```
public static int getN(Stack<Integer> s, int n) {
    if (n >= s.size()) {
        throw new IllegalArgumentException();
    } else {
        Stack<Integer> aux = new Stack<>();
        // return the n'th element
    }
}
```

B

```
public static int getN(Stack<Integer> s, int n) {
    if (n >= s.size() || n < 0) {
        throw new IllegalArgumentException();
    }
    Stack<Integer> aux = new Stack<>();
    // return the n'th element
}
```

C

```
public static int getN(Stack<Integer> s, int n) {
    Stack<Integer> aux = new Stack<>();
    if (n >= s.size()) {
        throw new IllegalArgumentException();
    } else if (n < 0) {
        throw new IllegalArgumentException();
    } else {
        // return the n'th element
    }
}
```

Practice : Pair

sli.do #cse122

Which is the most appropriate way to throw an exception for this method? The method should return the n'th element from the top of a Stack.

A

```
public static int getN(Stack<Integer> s, int n) {
    if (n >= s.size()) {
        throw new IllegalArgumentException();
    } else {
        Stack<Integer> aux = new Stack<>();
        // return the n'th element
    }
}
```

B

```
public static int getN(Stack<Integer> s, int n) {
    if (n >= s.size() || n < 0) {
        throw new IllegalArgumentException();
    }
    Stack<Integer> aux = new Stack<>();
    // return the n'th element
}
```

C

```
public static int getN(Stack<Integer> s, int n) {
    Stack<Integer> aux = new Stack<>();
    if (n >= s.size()) {
        throw new IllegalArgumentException();
    } else if (n < 0) {
        throw new IllegalArgumentException();
    } else {
        // return the n'th element
    }
}
```

Reference Semantics

- References are like phone numbers
  - The number itself is not the person, just a way to reach them

- What happens if I give a friend's number to someone else?

```
public static void main(String[] args) {
    Stack<String> stack = new Stack<>();
    int number = 0;
    method(stack, number);
}

public static void method(Stack<String> s, int n) {
    ...
}
```

All Objects are passed by reference!

Lecture Outline

- Announcements
- P1 walkthrough
- Quick Recap
- **copyStack Review** 
- Structured Example: spliceStack

(PCM) 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).

Your method should create the new stack and fill it up with the same values that are stored in the original stack. It is not acceptable to return the same stack passed to the method; you must create, fill, and return a new stack.

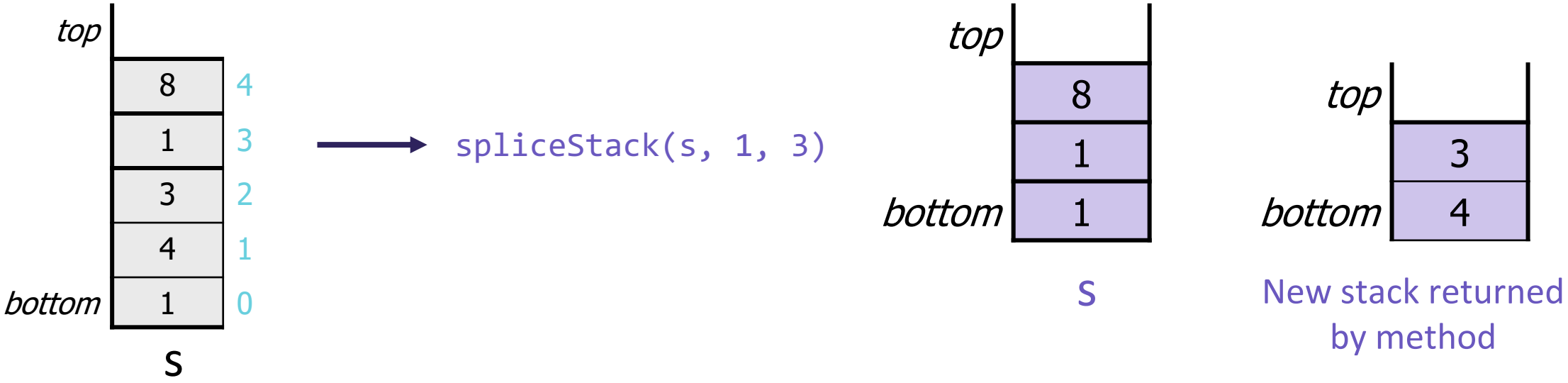
You may use one queue as auxiliary storage.

Lecture Outline

- Announcements
- P1 walkthrough
- Quick Recap
- copyStack Review
- **Larger 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.



Reminders

- Remember to complete Quiz 0 on Monday!
- P1 + Resub 1 will be released after lecture today
- Complete the PCM for Wednesday's lecture