

LEC 05

**CSE 122****Stacks**

Questions during Class?  
Raise hand or send here

sli.do #cse122

**BEFORE WE START*****Talk to your neighbors:***

*How do you relax after a stressful  
day/week?*

**Respond on sli.do!****Instructor:** Brett Wortzman

| <b>TAs:</b> | Archit   | Daniel   | Naomi    | Sreshta |
|-------------|----------|----------|----------|---------|
|             | Blake P. | David    | Neha     | Suyash  |
|             | Blake R. | Hanna    | Nicole   | Trisha  |
|             | Cady     | Isis     | Rio      | Wesley  |
|             | Carson   | Jun      | Rohan    | Yash    |
|             | Cole     | Landon   | Shreyank |         |
|             | Connor   | Michelle | Somya    |         |

Music: 🎸 [CSE 122 26au Lecture Tunes](#) 🎵

# Lecture Outline

- **Announcements** 
- Review: Stacks
- Review: Exceptions
- Stack Programming Example
- Data Structure Choice
- Commenting/Implementation Details

# Announcements

- P0 now past due
  - Congratulations!!
  - Start early, ask questions, test your code often
- CQ0 now running
  - Be sure to schedule if you haven't— spots are filling fast!
- P1 will release next Friday, October 23
- PQ0 Tuesday, October 27 (in section)

# 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?
  - **After finishing**: consider what went well and didn't, and what you should change or continue next time

# Lecture Outline

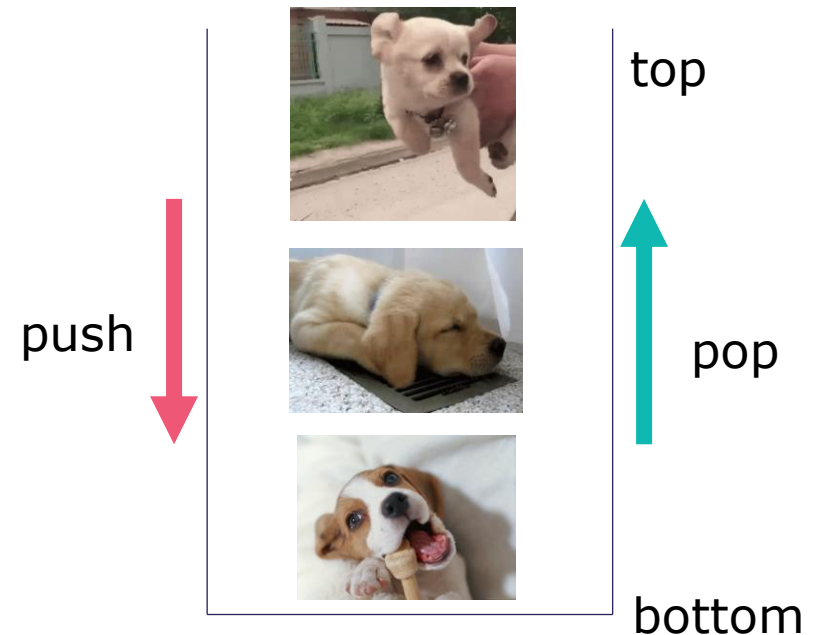
- Announcements
- **Review: Stacks** 
- Review: Exceptions
- Stack Programming Example
- Data Structure Choice
- Commenting/Implementation Details

# (PCM) Stacks

- **Stack:** A collection based on the principle of adding elements and retrieving them in the **opposite** order.
  - Last-In, First-Out ("LIFO")
  - Elements are stored in order of insertion.
    - We do not think of them as having indexes.
  - Client can only add/remove/examine the last element added (the "top")

Basic **Stack** operations:

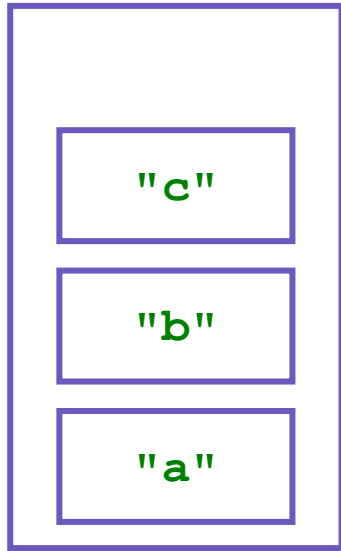
- **push:** Add an element to the top
- **pop:** Remove the top element
- **peek:** Examine the top element



# Stacks in Computer Science

- Programming languages and compilers:
  - method calls are placed onto a stack (*call*  $\leftrightarrow$  *push*, *return*  $\leftrightarrow$  *pop*)
  - compilers use stacks to evaluate expressions
- Operating Systems:
  - Call stacks  $\rightarrow$  memory stack for processes' data
- 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

# (PCM) Programming with Stacks



|                                      |                                                                                                                |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <code>Stack&lt;<b>E</b>&gt;()</code> | constructs a new stack with elements of type <b>E</b>                                                          |
| <code>push(<b>value</b>)</code>      | places given value on top of stack                                                                             |
| <code>pop()</code>                   | removes top value from stack and returns it;<br>throws <code>EmptyStackException</code> if stack is empty      |
| <code>peek()</code>                  | returns top value from stack without removing it;<br>throws <code>EmptyStackException</code> if stack is empty |
| <code>size()</code>                  | returns number of elements in stack                                                                            |
| <code>isEmpty()</code>               | returns <code>true</code> if stack has no elements                                                             |

```
➔ Stack<String> s = new Stack<String>();  
➔ s.push("a");  
➔ s.push("b");  
➔ s.push("c");  
➔ System.out.println(s.pop());
```

- Stack has other methods that we will ask you not to use 🙄

# Lecture Outline

- Announcements
- Review: Stacks
- **Review: Exceptions** ◀
- Stack Programming Example
- Data Structure Choice
- Commenting/Implementation Details

# 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

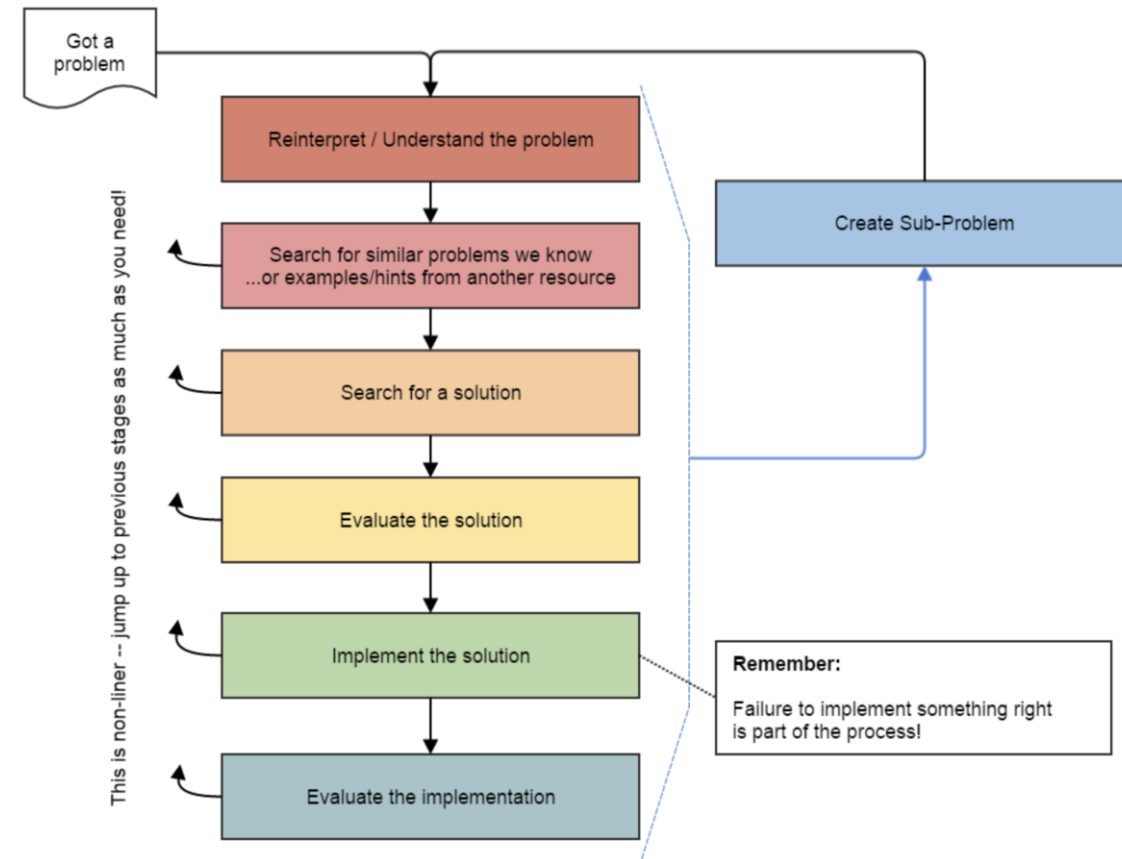
- Announcements
- Review: Stacks
- Review: Exceptions
- **Stack Programming Example** ◀
- Data Structure Choice
- Commenting/Implementation Details

# Lecture Outline

- Announcements
- Review: Stacks
- Review: Exceptions
- Stack Programming Example
- **Data Structure Choice** ◀
- Commenting/Implementation Details

# Data Structure Choice

- Many problems are made easier or harder based on the data structures used
- When given the choice, think carefully
  - Don't always choose a list
- Sometimes you *won't* be given the choice
  - Learn to work within constraints



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

# Lecture Outline

- Announcements
- Review: Stacks
- Review: Exceptions
- Stack Programming Example
- Data Structure Choice
- **Commenting/Implementation Details** ◀

# Commenting/Implementation Details

- In CSE 122, we use header comments as general *documentation*
  - Describing what the *client* needs to know about the program/method
- See the [Commenting Guide](#) for example and guidelines
- Should avoid providing *implementation details*
  - Things that are not relevant to the *client's* understanding
  - The “how” instead of the “what”
  - Note: this is not universal, but it is a 122 expectation

# Implementation Details

```
// B: Uses a for-loop to go from 0 up to but not including numCount and
//     puts the running sums in an ArrayList.
//     For example, if numCount is 4, the list will contain [0, 1, 3, 6].
// R: sums - the list of sums being returned
// P: numCount - the number of terms to sum
public static List<Integer> sumNumbers(int numCount) {
    List<Integer> sums = new ArrayList<>();
    int total = 0;
    for (int i = 0; i < numCount; i++) {
        total += i;
        sums.add(total);
    }

    return sums;
}
```

# Implementation Details

```
// B: Puts the running sums from 0 up to but not including numCount in a list.  
//     For example, if numCount is 4, the list will contain [0, 1, 3, 6].  
// R: the list of running sums  
// P: numCount - the number of terms to sum  
public static List<Integer> sumNumbers(int numCount) {  
    List<Integer> sums = new ArrayList<>();  
    int total = 0;  
    for (int i = 0; i < numCount; i++) {  
        total += i;  
        sums.add(total);  
    }  
  
    return sums;  
}
```