LEC 05

CSE 122

Autumn 2022

Stacks & Queues Practice

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

Debate: Are Pop-Tarts ravioli?

Music: [Hunter/Miya's Playlist]()

Instructor	Hunter Schafer / Miya Natsuhara		
TAs	Ajay	Gaurav	Melissa
	Andrew	Hilal	Noa
	Anson	Hitesh	Parker
	Anthony	Jake	Poojitha
	Audrey	Jin	Samuel
	Chloe	Joe	Sara
	Colton	Joe	Simon
	Connor	Karen	Sravani
	Elizabeth	Kyler	Tan
	Evelyn	Leon	Vivek

After Quiz 0

- We’ve heard your feedback about the first quiz. Thanks for sharing!
- As we’ve said many times, this is the first time offering the class and we are still trying to figure out the best structures for everything! Your feedback is very helpful in this!
- Have some planned changes to respond to feedback, but are going to wait until we finish grading Quiz 0 before making any concrete updates
  - We will look carefully at the time expectation for future quizzes
  - More clearly outline practice resources
  - Remember that syllabus grade promises are *minimum* guarantees, we can always be more lenient of certain assessments ended up being more difficult

Metacognition

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

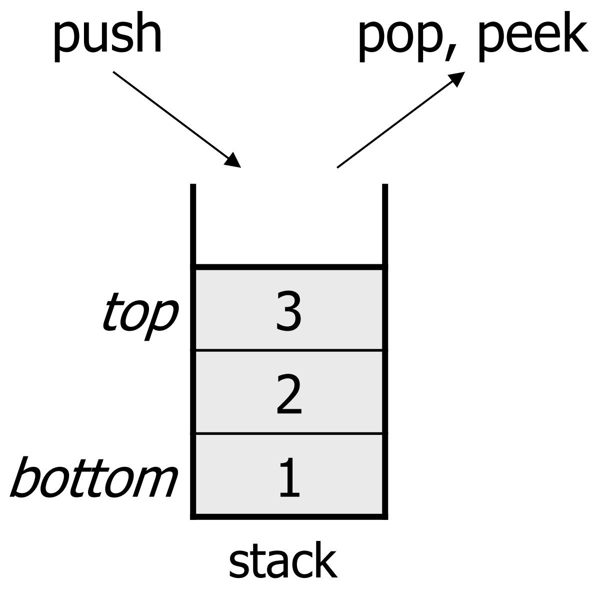
- Examples:
  - **While debugging**: explain to yourself why you're making this change to your program.
  - **Before running your program**: make an explicit prediction of what you expect to see.
  - **When coding**: be aware when you're not making progress, so you can take a break or try a different 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?

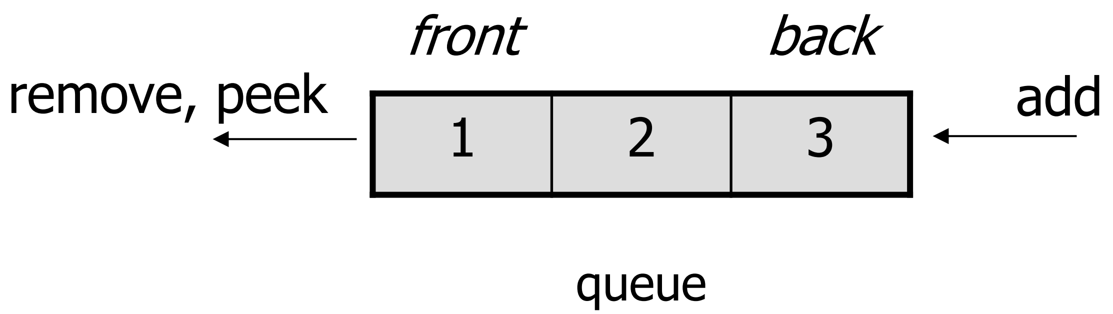
Learning is an Iterative Process!

- Retake logistics coming soon after Quiz 0 Feedback is released
  - Hopefully, Tuesday night / Wednesday morning
- Now that you know what to expect, you now have better ways you can respond if you felt your performance on the quiz wasn't what you wanted
- Here are some reflection questions
  - How did your studying/practice help you on the quiz? What aspects can you improve?
  - How did your development process on coding problems help/hinder your ability to complete the problem?
  - How did you budget time while taking the quiz?
- Come work with us on any of these skills!
  - The Ed Discussion board
  - The IPL and Hunter/Miya's Office Hours
  - Academic Support Programs

(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) Programming with Stacks

`Stack<`**`E`**`>()`	constructs a new stack with elements of type **E**
`push(`**`value`**`)`	places given value on top of stack
`pop()`	removes top value from stack and returns it; throws `EmptyStackException` if stack is empty
`peek()`	returns top value from stack without removing it; throws `EmptyStackException` if stack is empty
`size()`	returns number of elements in stack
`isEmpty()`	returns `true` if stack has no elements

```
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

(Recap) Programming with Queues

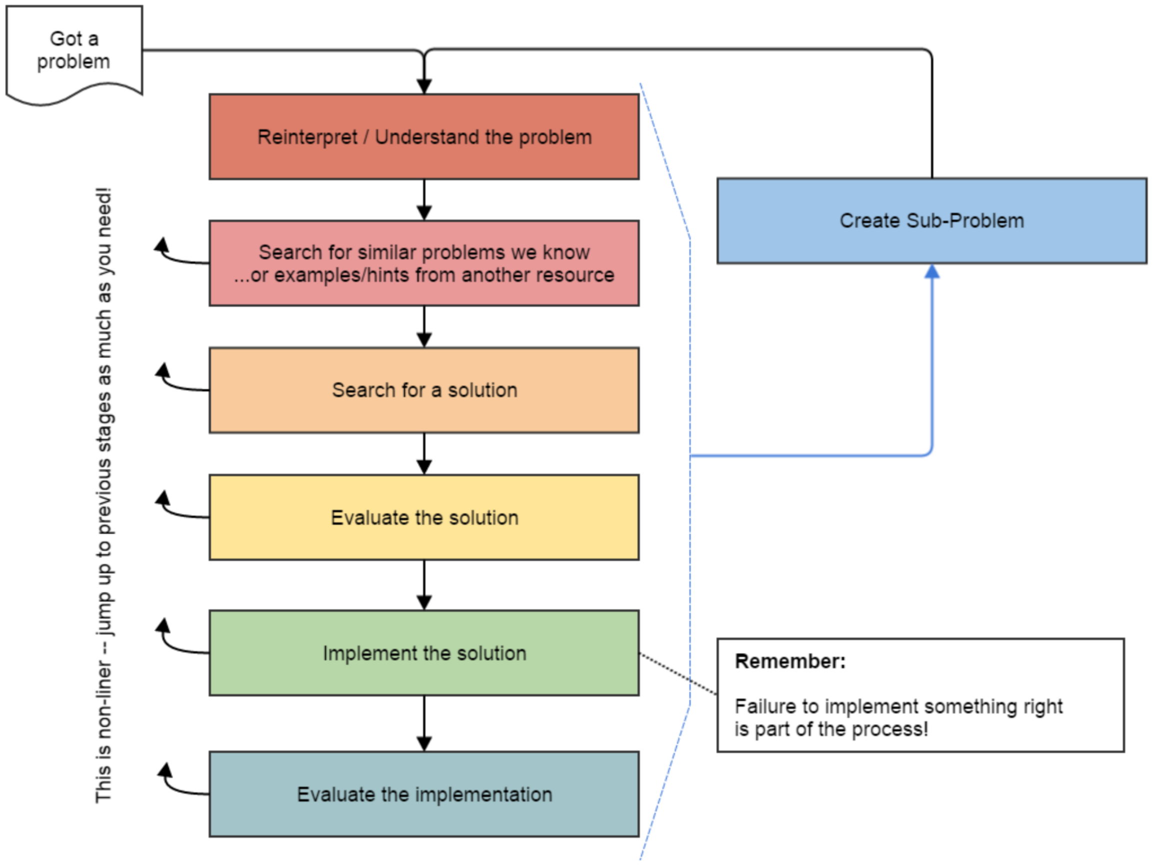
`add(`**`value`**`)`	places given value at back of queue
`remove()`	removes value from front of queue and returns it; throws a `NoSuchElementException` if queue is empty
`peek()`	returns front value from queue without removing it; returns `null` if queue is empty
`size()`	returns number of elements in queue
`isEmpty()`	returns `true` if queue has no elements

```
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.
  - This has to do with a topic we'll discuss later called *interfaces*.

(Recap) 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

(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

(Recap) 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