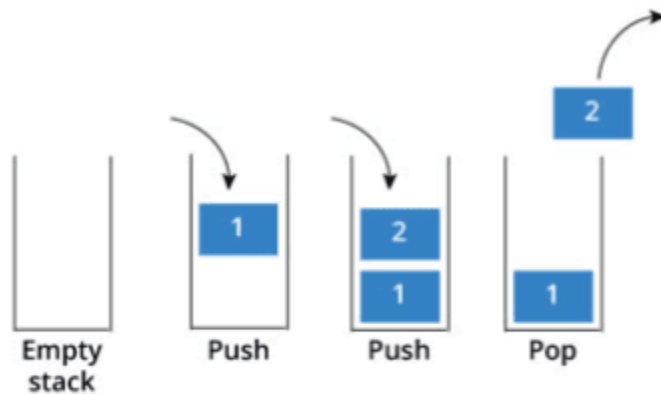


# *Stacks and Queues*

CS 2110



**Stack**



**Queue**

# *Friendly Reminder about Chat Etiquette*

- Chat Etiquette
  - Chat should be used for questions/answers relevant to the course material only
  - No off-topic chatter, spam, or inappropriate posts
  - Please be kind to each other!

# Schedule Changes

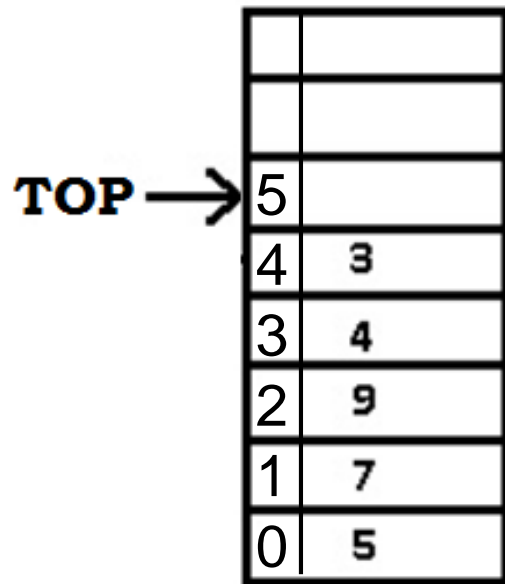
- Exam 2
  - Moved to April 3
- Homework 4 (GUI)
  - Due by 11:30pm on April 1 – *not* an April Fool's joke!
  - (To account for the days of office hours that were lost)
- Office Hours
  - Instructor and TA office hours resume today: Mon., March 23
- Weekly Labs
  - Due by 11:30pm Tuesday Night (seek help: Discord for lab)
  - See Piazza post for Lab Discord link

# *Online Format for CS 2110 (003) – Basit [Sp 2020]*

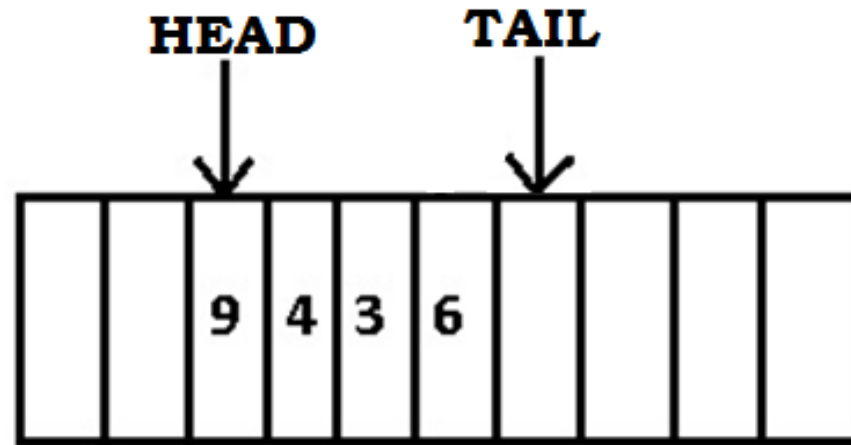
- **LECTURE: Live Streaming**
  - **Twitch** [[https://www.twitch.tv/prof\\_n\\_basit/](https://www.twitch.tv/prof_n_basit/)]
  - Recordings will be made
  - View immediately on Twitch
    - Glitch with Friday's lecture – see Discord in the "Archive" section for links
  - Long-term viewing
    - *Working on a solution – Twitch videos are temporary (stay tuned!)*
- **OFFICE HOURS (Prof. Basit – not TAs)**
  - **Discord** [don't forget to join my Discord channel using the **invite link!**]
    - *Happy to walk you through how to use Discord if you're a first-time user*

# Stacks and Queues

- LIFO (stack) vs. FIFO (queue)



**STACK**



**QUEUE**

# *Stack*

Last In – First out (LIFO)

Remember: work is done at ONE end

Pushing and Popping from the top of  
the Stack





# *Applications of Stack*

- **Expression Evaluation**

- Stack is used to evaluate prefix, postfix and infix expressions.

- **Expression Conversion**

- An expression can be represented in prefix, postfix or infix notation.  
Stack can be used to convert one form of expression to another.

- **Syntax Parsing**

- Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.

# *Applications of Stack*

- **Backtracking**

- Suppose we are finding a path for solving maze problem. We choose a path and after following it we realize that it is wrong. Now we need to go back to the beginning of the path to start with new path. This can be done with the help of stack.

- **Parenthesis Checking**

- Stack is used to check the proper opening and closing of parenthesis.

- **String Reversal**

- Stack is used to reverse a string. We push the characters of string one by one into stack and then pop character from stack.



# *Applications of Stack*

- **Function Call**
  - Stack is used to keep information about the active functions or subroutines.
  - The “runtime stack”!
- **Depth First Search**
- ...Many more!

# Stack

```
private String[] theStack;  
private final int STACK_SIZE = 3;  
private int top;    // pointer to the top of the stack (new items added here)  
  
public Stack(){  
    this.theStack = new String[STACK_SIZE];  
    this.top = 0;    // top pointer initialized to index position 0  
}
```



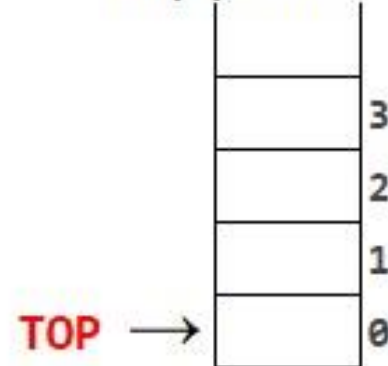
# Stack – push() method

```
public void push(String s){  
    growIfNecessary(); // if running out of room...  
    theStack[top] = s; // new item inserted at position "top"  
    top++; // increment top pointer (ready for new item)  
}
```

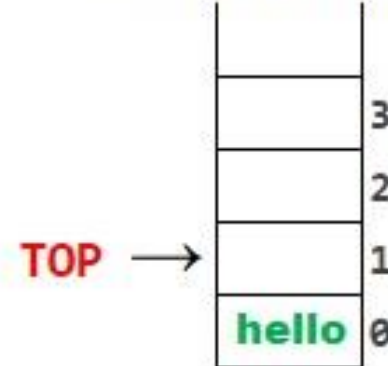
**TOP** →

|   |   |
|---|---|
|   |   |
|   |   |
| 5 |   |
| 4 | 3 |
| 3 | 4 |
| 2 | 9 |
| 1 | 7 |
| 0 | 5 |

Empty Stack:

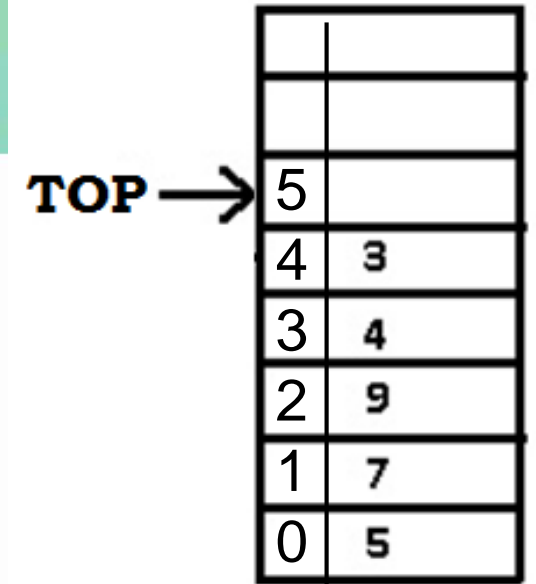


Stack with 1 element:



# Stack – pop() method // peek()

```
public String pop(){  
    if(top == 0){ // if nothing in the Stack (when top is at 0)  
        return null; // return null  
    }  
    top--; // otherwise, decrement top to point to current top item...  
    return theStack[top]; // and return the item that was at the top  
    // when the next push operation happens, item will be added here  
}  
  
// What would we change to peek? (Look but not remove)
```

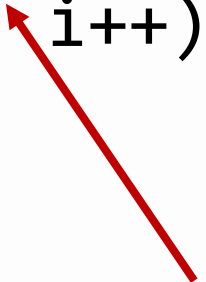


The diagram illustrates a stack as a vertical array of cells. An arrow labeled 'TOP' points to the cell containing the value 5. The stack contains the following elements from bottom to top: 5, 7, 9, 4, 3, and an empty cell at the top.

|   |   |
|---|---|
|   |   |
|   |   |
| 5 |   |
| 4 | 3 |
| 3 | 4 |
| 2 | 9 |
| 1 | 7 |
| 0 | 5 |

# Stack - GrowIfNecessary

```
private void growIfNecessary(){
    if(top == theStack.length){
        String[] newStack = new String[2*theStack.length];
        for(int i = 0; i < theStack.length; i++){
            newStack[i] = theStack[i];
        }
        theStack = newStack;
    }
}
```



Doubling the size of the Stack (Of course, having to create a brand new array to do this, then copy everything over)

Why does this method return **void** ?



# Queue

First In – First out (FIFO)

Remember: work is done at BOTH ends of the queue:  
Adding to the **tail**; Removing from the **head**.





# *Application of Queue*

- **Scheduling**
  - Queue is useful in CPU scheduling, Disk Scheduling. When multiple processes require CPU at the same time, various CPU scheduling algorithms are used which are implemented using Queue data structure.
- **Asynchronous data transfer**
  - When data is transferred asynchronously between two processes. Queue is used for synchronization. Examples : IO Buffers, pipes, file IO, etc.

# *Application of Queue*

- **Print spooling**
  - Documents are loaded into a buffer and then the printer pulls them off the buffer at its own rate. Spooling also lets you place a number of print jobs on a queue instead of waiting for each one to finish before specifying the next one.
- **Handling of interrupts in real-time systems**
  - The interrupts are handled in the same order as they arrive, First come first served.

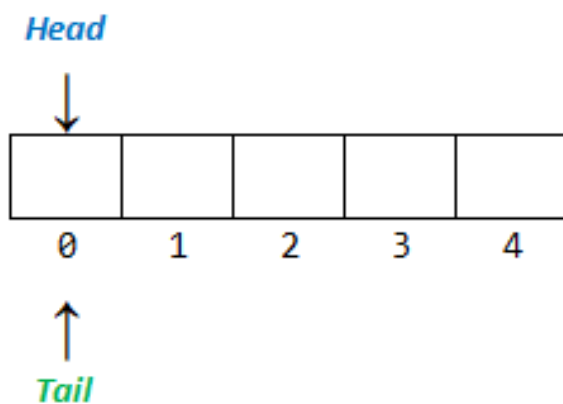
# *Application of Queue*

- **Call Center phone queues**
  - In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
- **Breadth First search**
- ... and many more!

## Example of a Queue (FIFO)

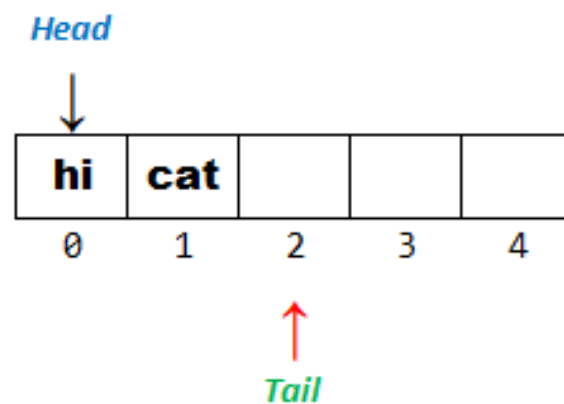
[1]

size = 0



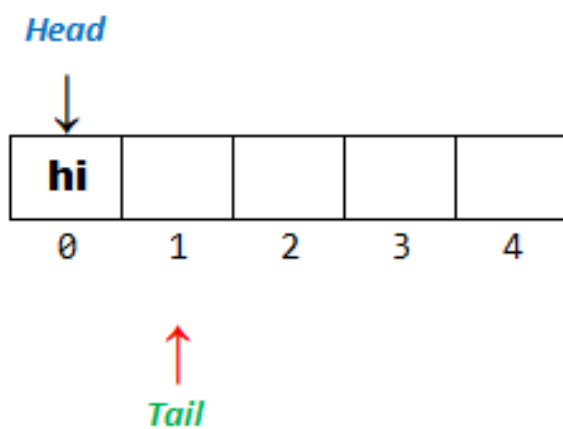
[3] `add("cat")`

size = 2



[2] `add("hi")`

size = 1

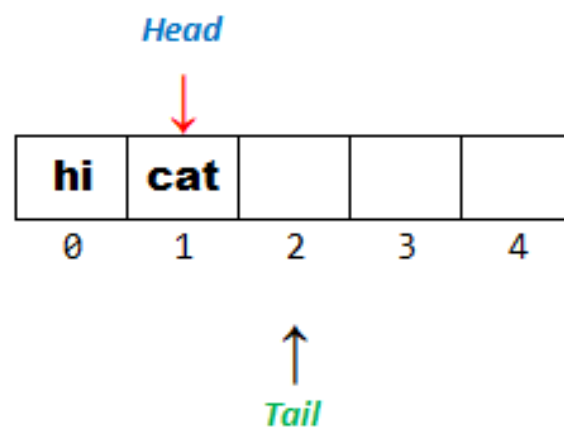


[4] `remove()`

removed = "hi"

size = 1

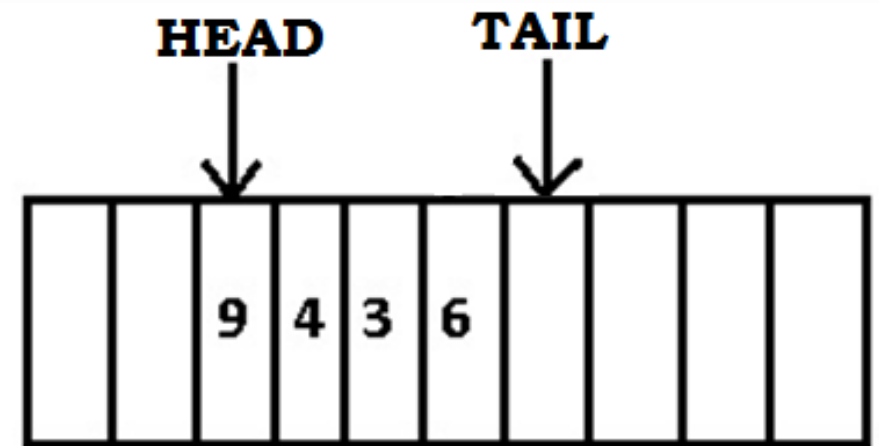
return removed



Although "hi" isn't removed, what is valid  
is between the "Head" and "Tail" pointers

# Queue – add() and remove() methods

- Think about how you would keep track of where to **insert** into the array and where you would **remove** from the array [Hint: pointers]
- Think about how you would handle the fact that when you remove from an array, you have an empty slot  
[Hint: either shift all the elements inside the array, or just keep track, via int pointers, of the location of the head and tail]
- *Are there other ways you can think of to do this?*



# Queue

```
final int INITIAL_SIZE = 4; // a constant
String[] elements;
int currentSize, head, tail; // head and tail are pointers

public Queue(){
    this.elements = new String[this.INITIAL_SIZE];
    this.currentSize = this.head = this.tail = 0;
    // all initialized to 0
}
```



# ***In-Class Activity***

## Stacks and Queues

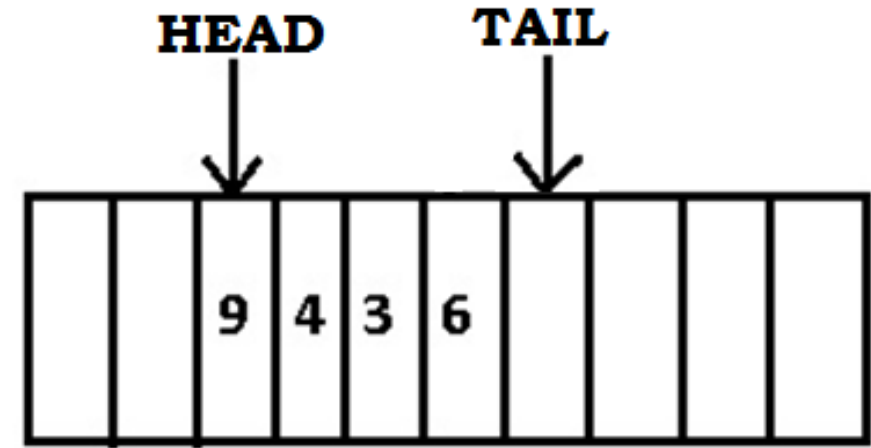
We've shown you Stacks... can you write `add()` and `remove()` for queues?

# *In-Class Activity: Stacks and Queues*

1. After reviewing the **Stack.java** code together as a class, with a partner review this code one more time and make sure you understand what is going on.
2. Now that you've seen the code to implement a **stack**, it is your turn to write `add( . . . )` and `remove( )` methods to implement a **Queue**. (Type your code into **Queue.java**)
  - Think about how you would keep track of where to insert into the array and where you would remove from the array
  - Think about how you would handle the fact that when you remove from an array, you have an empty slot

[Hint: either shift all the elements inside the array, or just keep track, via **int pointers**, of the location of the **head** and **tail**]
3. Carry out some *main-method testing* to ensure your code seems to be working as you hope
4. Submit your **Queue.java** file on Collab by the end of class today; or by 11:30pm tonight! EVERYBODY must make a submission individually (even if you were working with a partner).

# Queue – adjusting pointers \*hints\*



- **Add()** [ADD AT “TAIL” (END) OF QUEUE]
  - Increment size counter
  - Add at the tail: `elements[tail] = s;` // *s is the String value*
  - Adjust tail to be: `tail = (tail + 1) % elements.length;` // *can loop*
- **Remove()** [REMOVE FROM “HEAD” (FRONT) OF QUEUE]
  - Check if queue is empty, if so return null
  - Remove at the head: `String removed = elements[head];` // *to return*
  - Adjust head to be: `head = (head + 1) % elements.length;` // *can loop*
  - Decrement size counter
  - return `removed`