# CS 2401 – Elementary data structures and algorithms Fall 2024

# Lab 10

Due Date: December 6th, Friday, 11:59PM.

Objective:

The goal of this assignment is to practice stacks and queues. You will implement a PlayWithQueues, PlayWithStacks, Queue, and Stack class. You will test the stacks and queues you made in the classes PlayWithQueues and PlayWithStacks using the methods created in the Queue and Stack class. You have been given boilerplate code.

Extra Info:

As you see in our Node.java and Stack.java class we have a <T> which represents a generic type. Generics are a feature in Java that allows our classes, methods, and interfaces to be used on types specified at runtime. They allow for reusability and type safety without sacrificing performance or flexibility. Type Safety ensures that only a specific type of data can be pushed on the stack e.g. Stack<Integer> will only allow integers on the stack. We see the reusability aspect of <T> as we can reuse the same stack implementation for any type. <T> is a type parameter that acts as a place holder for the actual type and when you make the new stack you replace <T> with the type you actually want to use e.g. Integer or String.

Assignment – Timeline:

Milestone 1 – Nov 27 – Nov 30th

Task 1 – Implement the Stack.java Class

The Stack.java class will have the attributes top and size. The Constructors and getters have already been made for you.

You will complete the following methods push(), pop(), peek(), clear(), and isEmpty()

* Method: push(Object data)

Taks the data T and adds it to the stack

* + You first need to create a new node
  + Set the top as the node
  + Update the size
* Method: pop()

Removes the top element of the stack and returns the data that was just removed from the stack

* + Check if the stack is empty
  + Make a node that has the top information
  + Update the size
  + You want to return the data
* Method: peek()

Returns the data at the top of the stack

* + Check if the top is null
  + Return the data
* Method: clear()

Completely empties the stack, nothing should be in there

* + Change the attributes to show that the stack is empty
* Method: isEmpty()

Checks if the stack is empty and returns true if the stack is empty and false otherwise

* + Hint: If the stack is empty what would the top be?

Milestone 2 – Nov 30 – Dec 3rd

Task 2 – Implement the Queue.java class

The Queue.java class will have the attributes QUEUE\_SIZE, items, front, back, and count. The Constructors and getters have already been made for you.

You will complete the following methods isFull(), isEmpty(), dequeue(), enqueue(), peek(), and size()

* Method: isFull()

Checks if the queue has reached the maximum size allowed and returns true if the queue is full and false otherwise

* + Hint: What would the count be if the queue is full?
* Method: isEmpty()

Checks if the queue is empty and returns true if the queue is empty and false otherwise

* + Hint: What would the count be if the queue is empty?
* Method: dequeue()

Removes and returns the element at the head of the queue. Note that if the queue is empty you cannot dequeue anything

* + Check if the queue is empty
  + If it is not empty, get the item in the front of the queue, change the front of the queue, and update the count.
  + Return the item that was at the head of the queue
  + Else, print an error message and return null
* Method: enqueue()

Adds the newItem into the queue and returns nothing. Note that if the queue is full, nothing can be added

* + Check if the queue is full
  + If it is not full, get the item in the back of the queue, change the back of the queue, and update the count.
  + Else, print an error message
* Method: dequeueAll()

Resets all attributes of the queue

* + Hint: What would the attributes look like if the queue is empty?
* Method: peek()

Returns without removing the element at the head of the queue. Note that if the queue is full, nothing can be peeked at and your method returns null

* + Check if queue is empty
  + If it is not empty return the front item at the queue
  + Else, print an error message and return null
* Method: size()

Gets the size of the queue

* + Hint: What holds the size of the queue?

Final Submission – Dec 3 – Dec 6th

Task 3 – Implement the PlayWithStacks.java and PlayWithQueues.java

* Class: PlayWithStacks

Create your own stack and perform the methods created in the Stacks.java class.

Methods that must be demonstrated: push(), pop(), peek(), clear(), isEmpty()

\*\*You don’t have to do it this way, but you do have to demonstrate how all the methods work, this is just an idea\*\*

Psudocode:

Push 5 and 10 into stack

Print the item that has been popped (10)

Push 25 into the stack

Print the peek of the stack (25)

Clear the stack

Print out if the stack is empty

A screen shot of a computer

Description automatically generated

* Class: PlayWithQueues

Create your own stack and perform the methods created in the Queues.java class. You then will implement the methods inQueue() and printQueue().

Methods that must be demonstrated from the Queue class: isFull(), isEmpty(), dequeue(), enqueue(), dequeueAll(), peek(), size().

You will also test out the methods inQueue() and printQueue() in this class.

\*\*You don’t have to do it this way, but you do have to demonstrate how all the methods work, this is just an idea\*\*

Psudocode:

Create a queue

Enqueue 5 and 6 into queue

Print the current queue

A computer screen shot of a black screen

Description automatically generatedEnqueue 7 and 10 into the queue

Print the peek of the queue (10)

Clear the stack

Print out if the stack is empty



Enqueue 10 into the queue

Print the size of the queue

Print if the queue is full

Dequeue all elements in the queue

Print if the queue is empty

\*\*It would look like the one below \*\*

A black screen with blue text

Description automatically generated

* Method: printQueue()

Prints the whole queue

* + Get the size of the queue
  + Create a loop iterating through the queue
  + Within the loop create a temp where you dequeue the element of the given queue
  + You then enqueue the temp back into the queue and print out the element you enqueued
* Method: inQueue()

Checks to see if an item is in the queue

* + Get the size of the queue
  + Create a loop iterating through the queue
  + Within the loop create a temp where you dequeue the element of the given queue
  + You then enqueue the temp back into the queue
  + Check if the element you enqueued is the element you are searching for, if it is return true, else return false

|  |  |
| --- | --- |
| Deliverables: |  |
|  | |

You are expected to submit one file in Blackboard (in your lab section).

Please use PULSE to develop your solution, then either:

* + Press Submit in PULSE, OR
  + download the .java files (from the Folder), rename the main class as shown below, and submit the implemented files in Blackboard. NOTE: This will also be your class name in java

Lab10\_Lastname.java --- the java file of your program.

Grading Criteria:

* + - [10 points] The program is indented correctly.
    - [10 points] The program is documented properly.
    - [10 points] The program uses correct variable types and meaningful variable names.
    - [10 points] Each milestone submission is met with code that demonstrates effort to complete the tasks within them.
    - [20 points] Program compiles and runs.
  + [20 points] The program has correct logic and generates correct output.
  + [20 points] Demo completed with a member of the instructional team.

\*\* Note that in case of academic dishonesty the grade for this lab will be a 0 and disciplinary actions will be taken \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EXTRA CREDIT [25 points]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[15 points] Complete the method evaluatePostFix(), isOperator(), and isOperand() in the PlayWithStacks class, test the method with 2 examples in the main.

* Method: evaluatePostFix ()

This method, called evaluatePostFix, takes a String str as parameter which represents a post-fix arithmetic expression and returns the numeric value of this expression

* Method: isOperator ()

This method, takes a string as a parameter and checks to see if it is an operator

* Method: isOperand ()

This method, takes a string as a parameter and checks to see if it is an operand

\*\*You must complete all 3 methods to get the complete 15 points\*\*

[10 points] Complete the methods in the countNum() and ascendingOrder() in the PlayingWithQueues class. Test the method in the main.

* [5 points] Method: countNum()

In this method you will count how many times a number appears in the queue

* [5 points] Method: ascendingOrder ()

This method, takes the queue and checks to see if the queue is sorted in ascending order and return true if it is and false otherwise

 Late submission: [-10] points for every 24 hours after the deadline.

If you need any clarification, please ask your TA for further details.