**Worksheet 1 for Lecture 02 Stack and Queue**

1. What is Abstract Data Type (ADT)?

A description of what a collection of data can do.

1. Give some examples of ADTs.

List, Map, Queue, Stack

In Java, ADTs are expressed as Implementations.

1. (True/False) Stacks and Queues provide more complex functionality than List.
2. Stacks and Queues are linear ordered structure provides below functionalities.
   1. Add or Push: Insert an element in the data structure
   2. Remove or Pop: Deletes an element in the data structure
   3. Is it Empty?: Checks whether it has no element in the data structure
   4. Size: returns how many elements in the data structure
   5. (Borderline) Peek: Allows you to take a look at the first (in case of Queue) or the last element (in case of Stack) without removing the element
3. Queue vs Stack
   1. First In First Out (FIFO) is for Stack.
   2. Last In First Out (LIFO) is for Queue.
   3. The first element in the Queue is called the front and the last element

in the Queue is called the back.

* 1. The first element in the Stack is called the top and the last element in the Stack is called the bottom.

1. Instantiating an object type Queue<String>.
   1. Can you instantiate Queue type directly?

No.

* 1. Why or why not?

Queue is an interface.

* 1. Write a code to obtain an object of Queue<String> named q.

Queue<String> q = new LinkedList<String>();

1. Instantiating an object type Stack<Integer>
   1. Can you instantiate Stack type directly?

Yes.

* 1. Why or why not?

Stack is a class.

* 1. Write a code to obtain an object of Stack<Integer> named s.

Stack<Integer> s = new Stack<Integer>();

\* Integer is a wrapper object for primitive int type.

1. You are given below code. (SimpleStackQueueExercise.java)

String[] data = {**"four"**, **"score"**, **"and"**, **"seven"**, **"years"**, **"ago"**}; Queue<String> q = new LinkedList<String>();*//need to instantiate*

Stack<String> s = new Stack<String>();*//need to instantiate*

## //Write a code to insert each element in data to q and s

**for** (string str : data) {

# q.add(str);

s.push(str);

}

*//What is the output below for initial state of q and s?* System.***out***.println(**"initial queue = "** + q); System.***out***.println(**"initial stack = "** + s);

## //Below code deletes an element from q until q becomes empty

**while** (!q.isEmpty()) { String str = q.remove();

## //What is the output in each iteration?

System.***out***.println(**"removing "** + str + **", now queue = "** + q);

}

System.***out***.println();

## //Below code deletes an element from s until s becomes empty

**while** (!s.isEmpty()) { String str = s.pop();

## //What is the output in each iteration?

System.***out***.println(**"removing "** + str + **", now stack = "** + s);

}

System.***out***.println();

You are given below code and the output for below code.

[Code]

Queue<Integer> q = *makeQueueOfMultiples*(6, 3); System.***out***.println(**"initial queue = "** + q); System.***out***.println(**"sum = "** + *sum*(q)); System.***out***.println(**"after sum queue = "** + q); System.***out***.println();

[Output]

initial queue = [3, 6, 9, 12, 15, 18]

sum = 63

after sum queue = [3, 6, 9, 12, 15, 18]

[Exercise 1] Finish implementing below method

## // pre : count >= 0

*// post: returns a queue of count multiples of n*

**public static** Queue<Integer> makeQueueOfMultiples(**int** count, **int** n) {

Queue<Integer> q = new LinkedList<Integer>();  
  
for(int i = 1; i <= count; i++){  
 q.add(i \* n);  
}  
return q;

# }

[Exercise 2] Finish implementing below method

## // post: returns the sum of the values in q

**public static** int sum(Queue<Integer> q) {

**int** sum = 0;

for(int i = 0; i < q.size(); i++){

int n = q.remove();

sum += n;

q.add(n);

}

**return** sum;

# }

1. You are given below code and the output. [Code]

s = *makeStackOfMultiples*(6, 5); System.***out***.println(**"initial stack = "** + s); System.***out***.println(**"sum = "** + *sum*(s)); System.***out***.println(**"after sum stack = "** + s); System.***out***.println();

[Output]

initial stack = [5, 10, 15, 20, 25, 30]

sum = 105

after sum stack = [5, 10, 15, 20, 25, 30]

[Exercise 1] Finish implementing below method

*// pre : count >= 0*

*// post: returns a stack of count multiples of n*

**public static** Stack<Integer> makeStackOfMultiples(**int** count, **int** n)

{

Stack<Integer> s = new Stack<Integer>();

for(int i = 1; i <= count; i++){  
 s.push(i \* n);  
}

return s;

# }

[Exercise 2] Finish implementing below method

*// post: returns the sum of the values in s*

**public static** int sum(Stack<Integer> s) {

**int** sum = 0;

while (!s.isEmpty()) {

int n = s.pop();

sum = sum + n;

q.add(n);

}

queueToStack(q, s); //OR use an auxillary stack

stackToQueue(s, q); //OR use an auxillary stack

queueToStack(q, s); //OR use an auxillary stack

**return** sum;

}

1. You are given below code and the output for below code.

[Code]

System.***out***.println(**"current queue = "** + q); Stack<Integer> s = **new** Stack<Integer>(); *queueToStack*(q, s);

System.***out***.println(**"after queueToStack:"**); System.***out***.println(**" queue = "** + q); System.***out***.println(**" stack = "** + s); System.***out***.println();

[Output]

current queue = [3, 6, 9, 12, 15, 18] after queueToStack:

queue = []

stack = [3, 6, 9, 12, 15, 18]

[Exercise] Finish implementing below method

## // post: Values from q moved to s

*// (added in queue order, front to back);*

*// q is empty*

**public static** void queueToStack(Queue<Integer> q, Stack<Integer> s) {

**while** (!q.isEmpty()) {

int n = q.remove();

s.push(n);

}

}

1. You are given below code and the output. [Code]

System.***out***.println(**"current queue = "** + q);

System.***out***.println(**"current stack = "** + s); *stackToQueue*(s, q); System.***out***.println(**"after stackToQueue:"**); System.***out***.println(**" stack = "** + s); System.***out***.println(**" queue = "** + q);

[Output]

current queue = []

current stack = [5, 10, 15, 20, 25, 30] after stackToQueue:

stack = []

queue = [30, 25, 20, 15, 10, 5]

[Exercise]

*// post: Values from s moved to q*

*// (added in stack order, top to bottom);*

*// s is empty*

**public static void** stackToQueue(Stack<Integer> s, Queue<Integer> q) {

while(!s.isEmpty()){

int n = s.pop();

q.add(n);

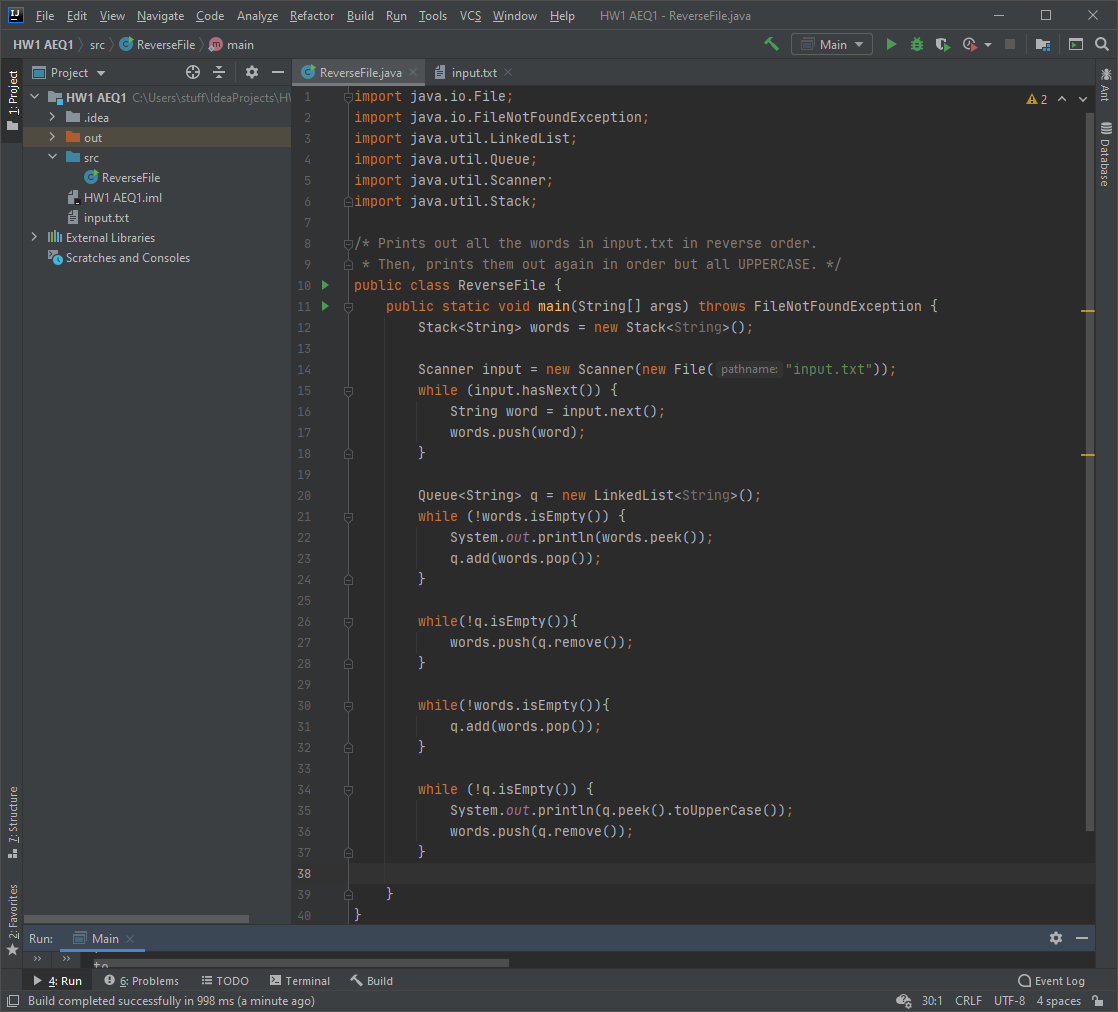
}

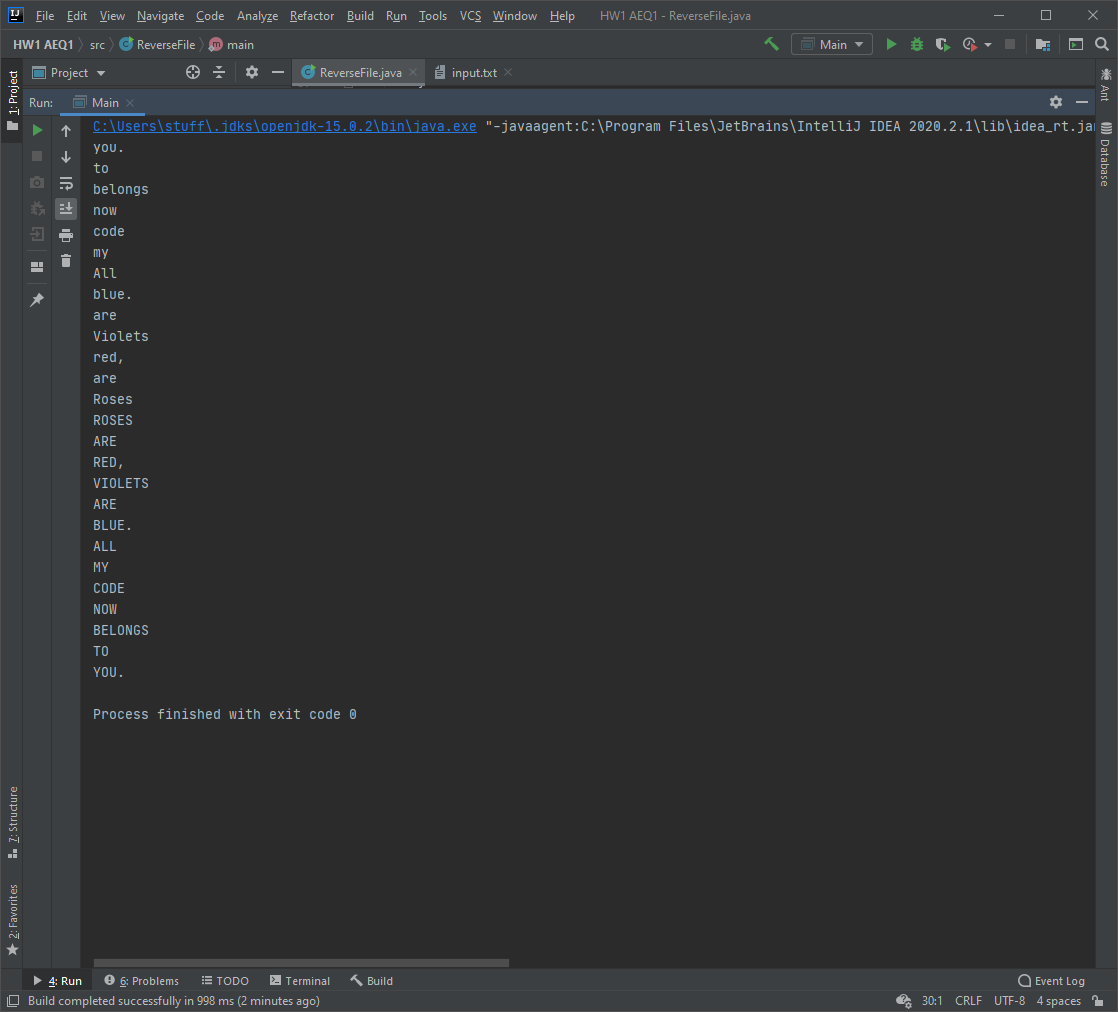
}

**Additional Exercise Questions**

\* Be prepare for the hard Stack/Queue questions in our first midterm

1. Using your IntelliJ, create a new project. Download code from BB called ReverseFile.java and place it under /src. Also download input.txt and place it under root (the parent of /src). Implement the main method, where it should print all words in input.txt in reverse order. Then prints them out again in order but all UPPERCASE. (You may use stackToQueue or QueueToStack method as the helper methods). Take screenshots of 1) your code and 2) the output and include them in a Word doc and submit to BB. The output should also show that words data structure containing the words in order after you print out them in UPPERCASE.





1. Write a method splitStack that takes a stack of integers as a parameter and splits it into negatives and non-negatives. The numbers in the stack should be rearranged so that all the negatives appear on the bottom of the stack and all the non-negatives appear on the top. In other words, if after this method is called you were to pop numbers off the stack, you would first get all the nonnegative numbers and then get all the negative numbers. It does not matter what order the numbers appear in as long as all the negatives appear lower in the stack than all the non-negatives. You may use a single queue as auxiliary storage. Write your code in the back and submit this page.

public static Stack<Integer> splitStack(Stack<Integer> s){  
  
 Queue<Integer> aux = new LinkedList<>();  
 int positiveCount = 0;  
  
 while(!s.isEmpty()) {  
 if (s.peek() > 0) {  
 positiveCount++;  
 }  
 aux.add(s.pop());  
 }  
  
 while(positiveCount > 0){  
 if(aux.peek() > 0){  
 aux.add(aux.remove());  
 positiveCount--;  
 } else {  
 s.push(aux.remove());  
 }  
 }  
  
 while(!aux.isEmpty()) {  
 s.push(aux.remove());  
 }  
  
 return s;  
  
}