Chapter 1110: Stacks & Queues

WEEK 4: MINIMAL (BUT USEFUL) DATA STRUCTURES

Stack/Queue Basics

- Like a list, both the stack and the queue store an ordered sequence of values

- Three essential operations:
  - An “add” method – a way to add values to the structure
  - A “remove” method – a way to take values out
  - An “is empty?” method -- a way to test if there are values remaining in the structure

- In their purest form, the stack and queue contain ONLY these operations – in Java, 2 additional methods are included (***size*** and ***peek***, which lets you see the next value in the structure without removing it)

Stack Concepts & Methods

- **LIFO** structure: Last In, First Out – items come off the stack in reverse order compared to how they were added
- All the action occurs at on end of the structure, the “***top***”
- The adding operation is called a “***push***”: ***push(value)***
- The removing operation is called a “***pop***”: ***pop()***
- checking the next value: ***peek()***
  - Does NOT change the stack in any way
- Get # of values: ***size()***
- Check if empty: ***isEmpty()***

Stacks in Java

- Stacks are generic structures: Stack<E>
- The Stack class is one of the oldest classes in the Java Collections Framework
  - Not as well designed as later ADTs in the framework
  - No separate interface specifying the ADT
    - Instead, the class extends a class called Vector, an early version of the ArrayList class
- As a result of its inheritance relationship, a stack can be treated as a list – i.e. values can be added/removed from the middle of the stack – PLEASE DON'T DO THIS!

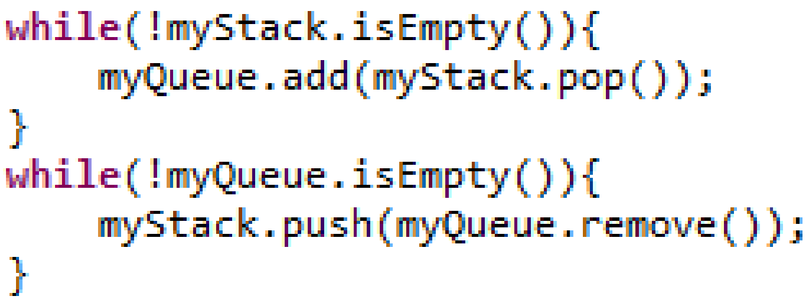
Queue Concepts & Methods

- **FIFO** structure: First In, First Out – items are removed in the same order in which they were added
- Properly designed to just support appropriate queue operations
- There is a separate Queue<E> interface – we will use the LinkedList<E> implementation for the queues we construct

- Methods: add(value), remove(), isEmpty(), peek(), size()

Transferring between Stacks and Queues

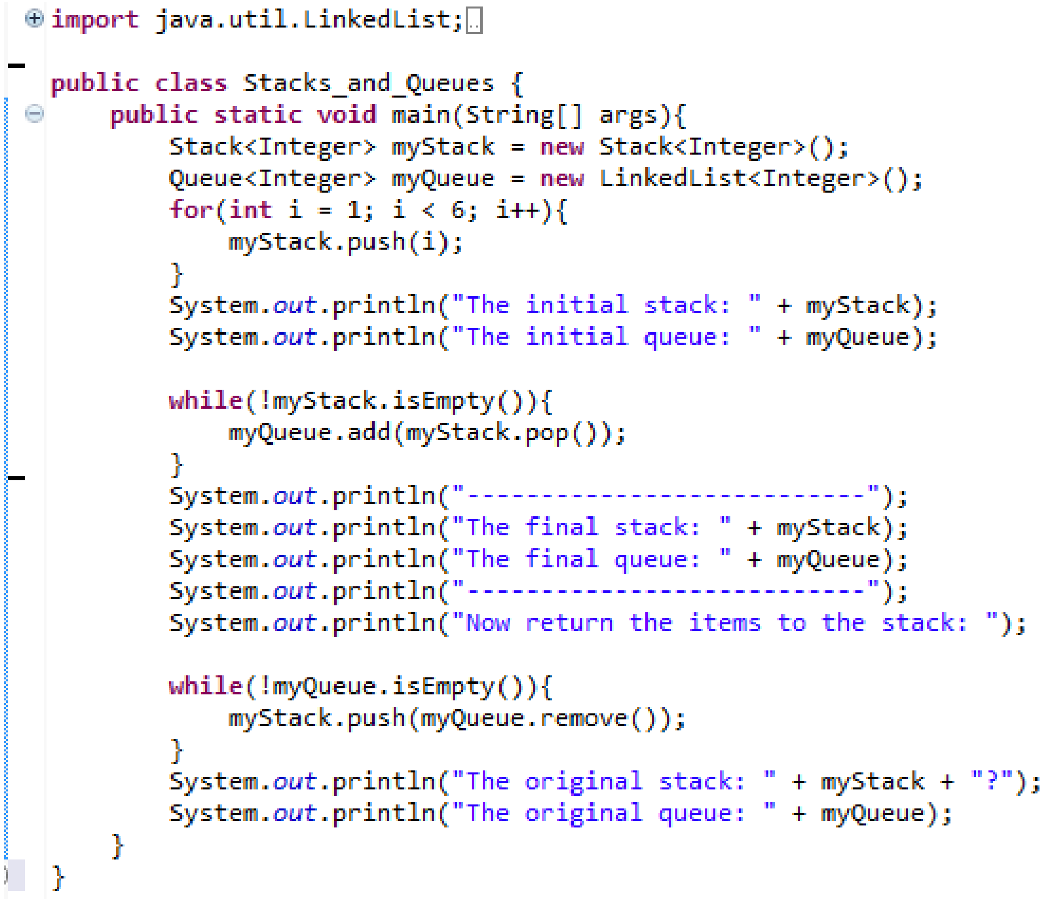
- What if you were asked to transfer all the values from a queue to a stack and back again?
- The following code would accomplish this:



```
while(!myStack.isEmpty()){
    myQueue.add(myStack.pop());
}
while(!myQueue.isEmpty()){
    myStack.push(myQueue.remove());
}
```

But will you end up with the same stack after completing these two while loops?

Let’s try it out:



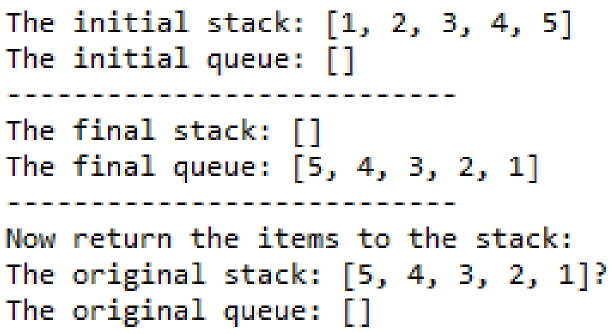
```java
import java.util.LinkedList;

public class Stacks_and_Queues {
    public static void main(String[] args){
        Stack<Integer> myStack = new Stack<Integer>();
        Queue<Integer> myQueue = new LinkedList<Integer>();
        for(int i = 1; i < 6; i++){
            myStack.push(i);
        }
        System.out.println("The initial stack: " + myStack);
        System.out.println("The initial queue: " + myQueue);

        while(!myStack.isEmpty()){
            myQueue.add(myStack.pop());
        }
        System.out.println("---------------------------");
        System.out.println("The final stack: " + myStack);
        System.out.println("The final queue: " + myQueue);
        System.out.println("---------------------------");
        System.out.println("Now return the items to the stack: ");

        while(!myQueue.isEmpty()){
            myStack.push(myQueue.remove());
        }
        System.out.println("The original stack: " + myStack + "?");
        System.out.println("The original queue: " + myQueue);
    }
}
```

Our output:

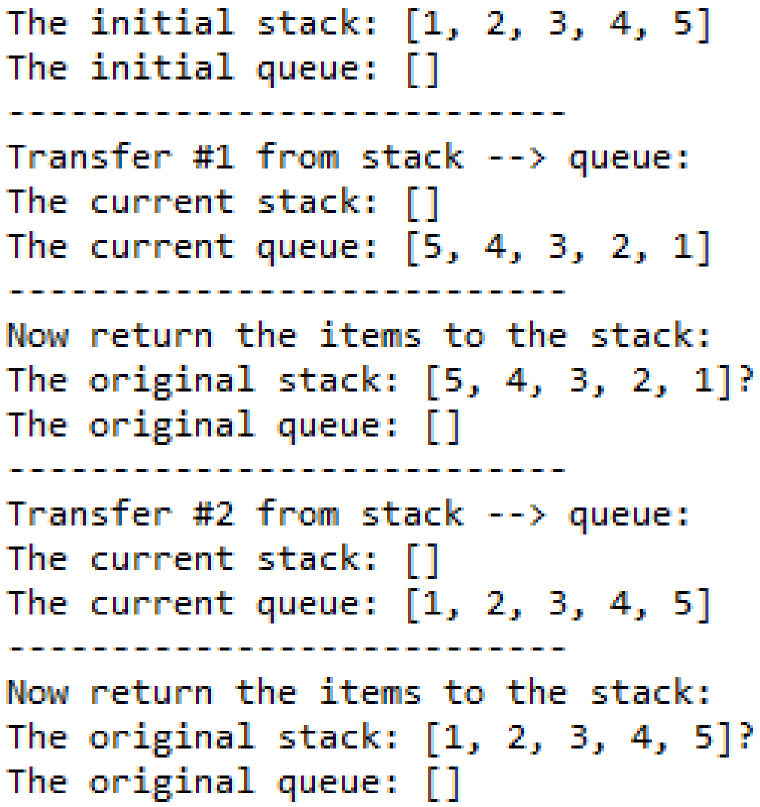


```
The initial stack: [1, 2, 3, 4, 5]
The initial queue: []
---------------------------
The final stack: []
The final queue: [5, 4, 3, 2, 1]
---------------------------
Now return the items to the stack:
The original stack: [5, 4, 3, 2, 1]?
The original queue: []
```

MAYBE NOT QUITE WHAT WE WERE EXPECTING… BUT POTENTIALLY USEFUL

Transferring between stacks and queues, continued…

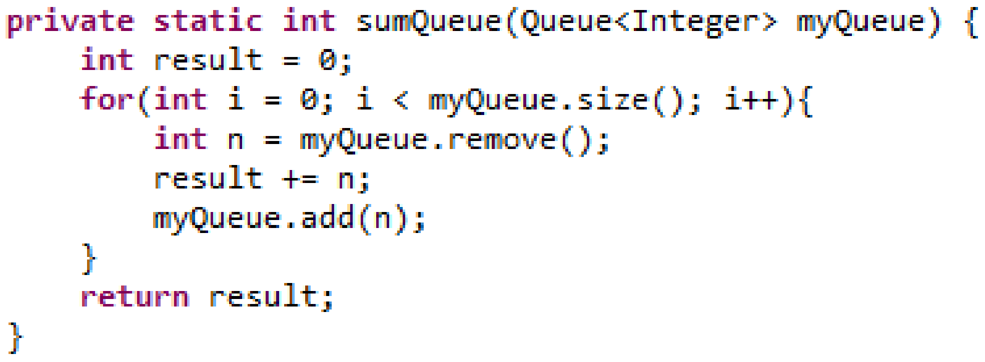
- We seem to have come up with a handy way to reverse a collection
- In order to regain our original stack, we will have to do the transferring a second time:



```
The initial stack: [1, 2, 3, 4, 5]
The initial queue: []
---------------------------
Transfer #1 from stack --> queue:
The current stack: []
The current queue: [5, 4, 3, 2, 1]
---------------------------
Now return the items to the stack:
The original stack: [5, 4, 3, 2, 1]?
The original queue: []
---------------------------
Transfer #2 from stack --> queue:
The current stack: []
The current queue: [1, 2, 3, 4, 5]
---------------------------
Now return the items to the stack:
The original stack: [1, 2, 3, 4, 5]?
The original queue: []
```

Removing values from a queue

- A for loop can be used to cycle through the elements of a queue, removing each element in turn and adding it back – e.g. to find the sum of the values in the queue:

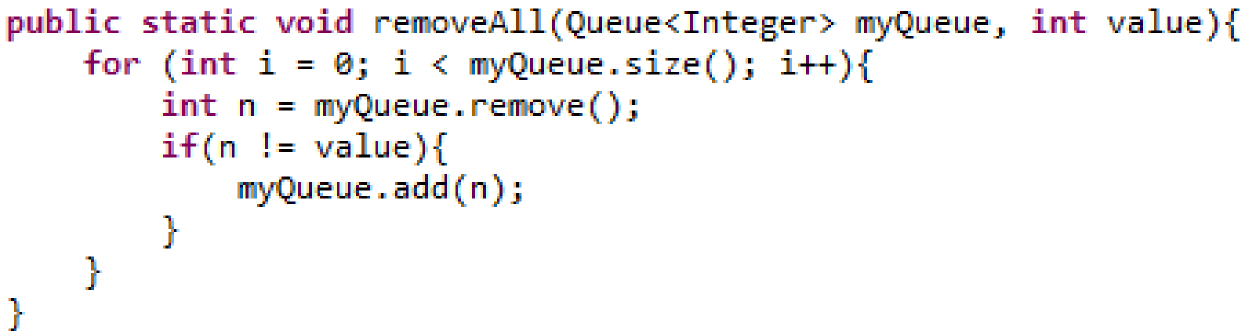


```
private static int sumQueue(Queue<Integer> myQueue) {
    int result = 0;
    for(int i = 0; i < myQueue.size(); i++){
        int n = myQueue.remove();
        result += n;
        myQueue.add(n);
    }
    return result;
}
```

- In this method, we are not actually changing the size of the queue – if we try to permanently remove values, we may run into some problems…

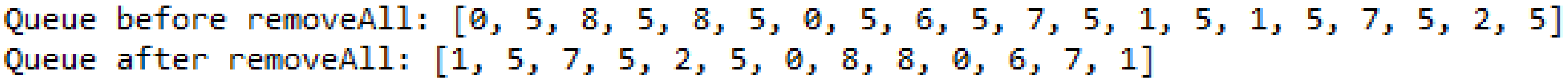
Removing queue values, cont…

This method ***LOOKS*** reasonable (but won’t work):



```
public static void removeAll(Queue<Integer> myQueue, int value){
    for (int i = 0; i < myQueue.size(); i++){
        int n = myQueue.remove();
        if(n != value){
            myQueue.add(n);
        }
    }
}
```

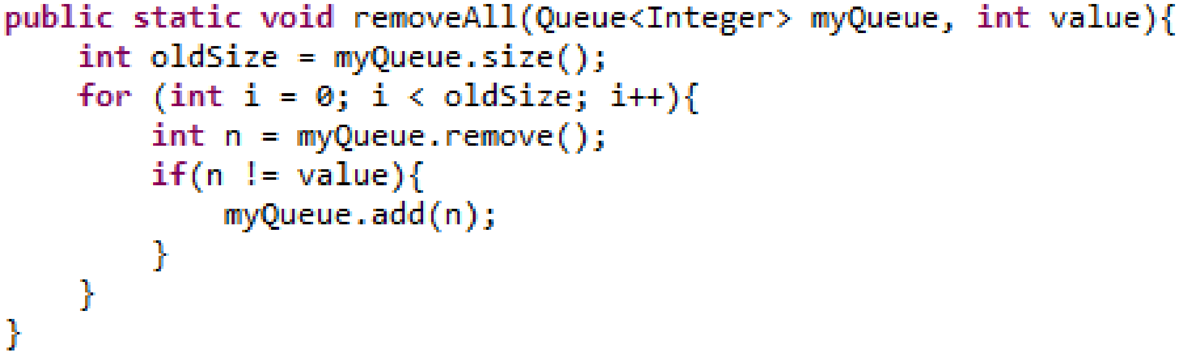
Output when trying to remove all 5s from a queue:



```
Queue before removeAll: [0, 5, 8, 5, 8, 5, 0, 5, 6, 5, 7, 5, 1, 5, 1, 5, 7, 5, 2, 5]
Queue after removeAll: [1, 5, 7, 5, 2, 5, 0, 8, 8, 0, 6, 7, 1]
```

Removing queue values, cont…

- Not only has our method failed to remove all the 5s, it has also destroyed the order that the queue had originally
- The problem is due to the fact that removing items changes the size of the queue, a property we are using as a loop control variable
- We need to store the original size of the list in a variable, and use this to control how many loop iterations are performed:
- This version WILL work:



```
public static void removeAll(Queue<Integer> myQueue, int value){
    int oldSize = myQueue.size();
    for (int i = 0; i < oldSize; i++){
        int n = myQueue.remove();
        if(n != value){
            myQueue.add(n);
        }
    }
}
```