Noam Zeise, 276631Z
ADS Assesed Exercise 1

AE2

Running the code

The code lives in a single java file, *ae2.java*. The program can be run using the java app from the terminal using the command java ae1.java .

The class has only public static methods and classes, so it could also be used from another java file if needed.

Part 1

First I made a small cons cell class to build the linked list with. The head pointer is the most recent element added to the queue, and stores a pointer to the last element for $O(1)$ access to the first and last element. This gives constant time for enqueue and dequeue operations.

Enqueue works by adding a new cell that points to the last element (the same that head points to), then setting the head's pointer to the new cell and making the new cell the current head.

Dequeue works by getting the value from the last element (the cell that head points to), then updating the head to point to the cell that the dequeued node used to point to. The value is then returned.

Part 2

First I created an *Elem* type, which is a cell of a double linked list with a count that holds how many of that value is in the stack. When a new item is added, the stack is searched for the item. If found, the count of the item is incremented. If it isn't already in the stack, it is added to the head.

Pop works by scanning through the list for the max count of an entry. The count of that entry is decremented by 1. If the count is now zero then the element is removed from the stack. As the stack is searched from head to tail, in the event of a tie, the element closer to the top of the stack is returned.

push and *pop* are $O(n)$, as we need to find if an element already exists, and to find the largest count value.

Part 3

My function $genSameHashStrings(n)$ will return an arraylist containing n strings that hash to the same value when using $hashCodes(s)$. This works by exploiting the fact that the hash will divide the length of the string by 3 to get the step value for the for loop. This means that any string of length $3i$, where $i \in \mathbb{Z}_{\geq 1}$, will always check 3 indices from the string for the hash function. So I just need to generate n strings with length $3i$ for $i \in \{1,\ 2,\ ...\ ,\ n\}$, all with the same letter, then the hash function will generate the same key for all of those strings. My function happens to use the dash character.