Q&A

1a.

It would be useful to not just have an instance variable tracking the front of our list, but also
have an instance variable tracking the back of our list. Why is this helpful? For which methods
does it speed up the implementation?

- The reason why it is useful is that it is more efficient to add a value to the end of the stack.
- Keeping track of the end of the linked list speeds up the following:
  1. ListStack.push()
  2. ListStack.peek()
  3. ListQueue.enqueue()
  4. ListQueue.peek()

1b

Method	ArrayList	LinkedList
add(int value)	O(1)	O(1)
add(int value, int value)	O(n)	O(n)
Clear()	O(1)	O(1)
contains(int value)	O(n)	O(n)
get(int index)	O(1)	O(n)
isEmpty()	O(1)	O(1)
remove(int index)	O(n)	O(n)
toString()	O(n)	O(n)
Equals(Object o)	O(n)	O(n)

1c

I would rather use ArrayList instead of LinkedList when I want to look for the number of cars produced in a particular year casue I know that their number is fixed and can access any of them in constant time.

1d

I would use a LinkedList to keep track of how much money I am spending each month because it is not fixed that how many months I live, it is likely to change.

2a

Method	ArrayStack	ListStack
push(int value)	O(n)	O(1)
pop()	O(1)	O(n)
peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
size()	O(1)	O(1)
clear()	O(1)	O(1)
toString()	O(n)	O(n)
equals(Object o)	O(n)	O(n)

2b

I would use LinkedList to Implement my Stack class because what is most important in a stack is the front and the end. Therefore, linked lists are more efficient in this case.

3a

Method	ArrayQueue	ListQueue
enqueue(int value)	O(n)	O(1)
dequeue ()	O(n)	O(1)
peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
size()	O(1)	O(1)
clear()	O(1)	O(1)

toString()	O(n)	O(n)
equals(Object o)	O(n)	O(n)

3b

I would still be using Linked List to implement Queue, why, because it is similar to Stacks, what we care most in Stacks and Queues are what is at the front and at the end.