

CSE 12 – Basic Data Structures and Object-Oriented Design

Lecture 7

Greg Miranda & Paul Cao, Winter 2021

Announcements

- Quiz 7 due Friday @ 8am
- Survey 3 due Friday @ 11:59pm
- PA2 due tonight @ 11:59pm

Topics

- Queue
- Stacks
- Other Topics
 - Adapter Pattern
 - Composition
 - Stacks & Queues with LinkedList

Queue – insertion and deletion operations are performed at two different ends



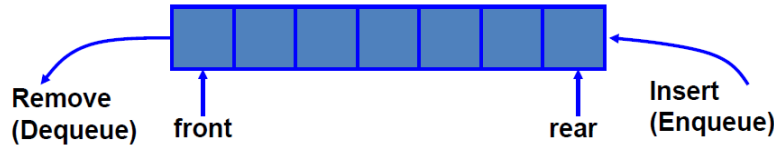
Is a Queue an ADT or a data structure?

- A. ADT
- B. Data structure
- C. I have no idea, what's the difference again??

Queue

- A *queue* is an ADT, in which insertion is done at one end, while deletion is performed at the other end.
- Accessing the elements of queues follows a First In, First Out (FIFO) order.
 - Like customers standing in a line in a store, the first customer in is the first customer served.
- We can only add to the end of the queue, and can only examine/remove the front of the queue.

Enqueue and Dequeue



- Like lines in a store, a queue has a front and a rear.
- Enqueue – insert an element at the rear of the queue
- Dequeue – remove an element from the front of the queue

Example of Enqueue and Dequeue

- Suppose we have an empty static integer queue that is capable of holding a maximum of three values. With that queue we execute the following enqueue operations.

Enqueue(3);

Enqueue(6);

Enqueue(9);


```
Enqueue(3);
```

```
Enqueue(6);
```

```
Enqueue(9);
```

```
Dequeue();
```

```
Dequeue();
```

```
Dequeue();
```

Queues

Consider doing the following operations on an initially empty queue, q:

`s.enqueue(4)`

`s.enqueue(10)`

`s.enqueue(13)`

`s.dequeue()`

`s.enqueue(5)`

What are the contents of the stack, from front (left) to rear (right):

A. 4, 10, 13, 5

B. 10, 13, 5

C. 4, 10, 5

D. 5, 10, 4

E. other

Searching with a Queue

	co 0	co 1	co 2	co 3
ro 0				
ro 1				
ro 2				S
ro 3	Exit			

SearchForTheExit

- Initialize a **Queue** to hold Squares as we search
- Mark starting square as visited
- Put starting square on task list
- While **Queue** is not empty
 - Remove square sq from task list
 - Mark sq as visited
 - If sq is the Exit, we're done!
 - For each of square's unseen neighbors (S, W, N, E):
 - Set neighbor's previous to sq
 - Add neighbor to **Queue**

Label the cells in the order in which they are visited by the algorithm.

What is the final path to the goal?

	co <u>0</u>	co <u>1</u>	co <u>2</u>	co <u>3</u>
ro <u>0</u>				
ro <u>1</u>				
ro <u>2</u>				S
ro <u>3</u>	Exit			

Queue - Implementation

One option: Implement the methods in the ADT from scratch.

```
public class MyQueue<E>{
    Object[] data;
    int size;
    int front;
    int back;
    public boolean enqueue(E elem){
        //if full resize
        //change front
        //add in the new element
    }
}
```

Adapter Pattern

Adapter Pattern

- Lazy Paul needs to implement the Queue Interface with the following methods:
 - void enqueue(E element) – add elements on to the queue.
 - E dequeue() – remove element from front of the queue.
 - int size() – return the size of the queue.
- Let's see what he can do to be as lazy as possible.

Adapter Pattern

- Paul has access to a data structure implementation that supports the following methods. Let's call this data structureArrayList
 - void add(int index, E value)
 - E remove(int index)
 - int size()

Inheritance?

Paul realizes that he can just make Queue extend the ArrayList and write the additional methods by using other existing methods.

Ex:

```
public Queue<E> extends ArrayList<E>
{
    ...

    public E dequeue() {
        E toReturn = this.contents.get(0);
        this.contents.remove(0);
        return toReturn;
    }

    ...
}
```

Pros? Cons?

Inheritance is not always the right answer

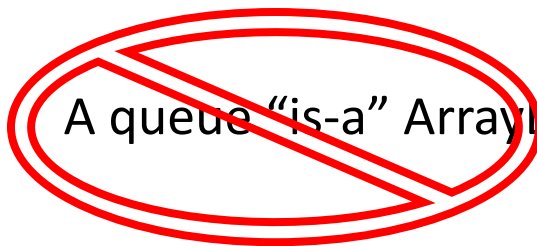
- Paul has access to a data structure implementation that supports the following methods. Let's call this data structureArrayList
 - void add(int index, E value)
 - E remove(int index)
 - int size()

Adapter Pattern

Making the ArrayList variable private makes sure that users of the Queue cannot access the ArrayList or its methods.

Only the Queue methods are public and therefore usable by clients.

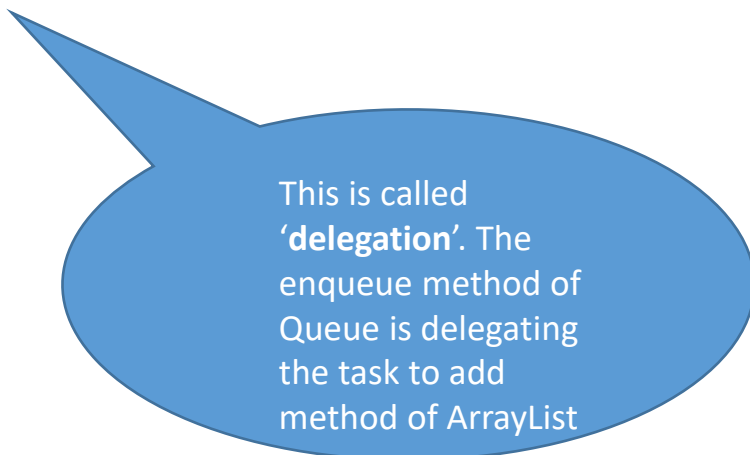
You can happily use ArrayList within Queue and pass on operations to it.

 A queue “is-a” ArrayList

A queue has-a ArrayList!

Adapter Pattern – Example

```
public class Queue<E> implements QueueInterface<E> {  
    private ArrayList<E> container;  
    ...  
    public void enqueue(E element) {  
        this.contents.add(this.contents.size(), element);  
    }  
}
```



This is called
'**delegation**'. The
enqueue method of
Queue is delegating
the task to add
method of ArrayList

And no one needs to know..



Every one thinks I implemented Queue from scratch

Mapping Attributes

- Before deciding on what methods to use, one needs to map the corresponding attributes.
- For example: To use the ArrayList as a Stack, we need to map the Top of the stack to some position in the list (front or back—our choice, but how to choose?)

Mapping methods

- Once this is done, we can map the methods on top of the stack to methods operating on the head of the List.

If we choose the front....

- push -> add
- pop -> remove
- peek -> get

Adapter Pattern Summary

You would like to implement an Interface A.

You have an implementation B that implements another interface C which defines methods very much similar to the methods in A but differ slightly (like name).

You use an instance of B inside your class that implements A and delegate tasks to it.

Your class A “has a” class B.

Adapt Queue from Deque

- enqueue -> addLast
- dequeue -> removeFirst
- peek -> peekFirst



A **stack** has two operations, **push** and **pop**. Pushing adds an element to the **top** of the stack, and **pop** removes the **top** element and returns it.

```
Stack<Integer> s = new ALStack<>();
s.push(4);
s.push(10);
s.push(13);
Integer i = s.pop();
s.push(5);
Integer i2 = s.pop();
```

What number is stored in i?

A: 4 B: 10 C: 13 D: 5
E: Something else

What number is stored in i2?

A: 4 B: 10 C: 13 D: 5
E: Something else

What is the contents of the stack? (starting at the **top**)

A. 5, 13, 10, 4
B. 10, 4
C. 5, 13
D. 13, 10, 4
E. other

```
import java.util.ArrayList;

public interface Stack<E> {
    void push(E element);
    E pop();
    int size();
}

// IDEA: Use array lists to implement both
class ALStack<E> implements Stack<E> {

```

Other Topics

- Adapter Pattern
- Composition
- Stacks & Queues with LinkedList