CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 7

Greg Miranda, Spring 2021

Announcements

- Quiz 7 due Wednesday @ 12pm
- PA2 due Wednesday @ 11:59pm
- Survey 3 due ~~tonight~~ @ 11:59pm

Friday

PA2 Code → autograde only resubmission → upto 50%

questions → No resubmission

Topics

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

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();
```

i = 13

i2 = 5

What number is stored in i?

4

A: 4   B: 10   C: 13 *(circled) 29*   D: 5

          E: Something else
          6

What number is stored in i2?

0   *3*   *0*

A: 4   B: 10   C: 13   D: 5 *(circled) 30*

          E: Something else
          1

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

3 A. 5, 13, 10, 4
33 B. 10, 4 *(circled)*
C. 5, 13
D. 13, 10, 4
E. other

10, 4

top

10
4

bottom

```java
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

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 Greg needs to implement the Queue Interface with the following methods:
  - void enqueue(E element) – add elements to the back of 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

- Greg has access to a data structure implementation that supports the following methods.
  - void add(int index, E value)
  - E remove(int index)
  - int size()

- Let's call this data structure ....ArrayList

Inheritance?

Greg 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

- Greg has access to a data structure implementation that supports the following methods.
  - void add(int index, E value)
  - E remove(int index)
  - int size()

- Let's call this data structure ....ArrayList

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!

composition

Adapter Pattern – Example

```java
public class Queue<E> implements QueueInterface<E> {
    private ArrayList<E> container;

    ...

    public void enqueue(E element) {
        this.contents.add(this.contents.size(), element);
    }
}
```
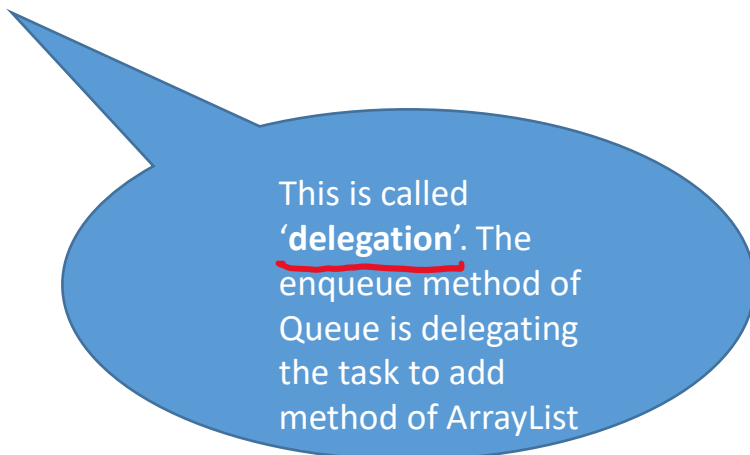
composition

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

And no one needs to know..

Everyone thinks I implemented Queue from scratch

Stack - Implementation

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?)

efficiency
↳ adding / removing from the back

Top

Mapping methods

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

composition

Topics

- Stacks & Queues with LinkedList