Logistics

At Home

- Weiss Ch 15 on ArrayLists
- Weiss Ch 16 Stacks/Queues
- Weiss Ch 17 Linked Lists
- Your PA01 due Sept 24.

Goals Today

- Implementation of Stacks and Queues
- Review iterator
- Work on some example code
Iterators (Review)

Stack

Queue
Iterators (Review)

- Iterators are pointers to the object in the list
- Java Collection creates/provides an iterator

```
5  8  14  6  9
  (a)

5  8  14  6  9
  (b)

5  8  14  6  9
  (c)
```

- operators
  - Use `next()`/`previous()` to move
  - `next()`/`previous()` returns element "moved over"
  - `remove()` removes element that was returned from last `next()`/`previous()`
  - Illegal to remove w/o first calling `next()`/`previous()`
  - `add(x)` before whatever `next()` would return
  - `set(E o)`
What would you do?

```java
// l = [A, B, C, D];
it1 = l.iterator().next().next();
it2 = l.iterator().next();
// l = [ A B C D ]
// 1
// 2
it1.remove();
it2.next(); // ??
```

Where should `it2` be now?

- Determine viable possibilities
- Explore what Java actually does
Java’s premise: **Danger!**

```java
it1 = l.iterator();
it2 = l.iterator();
it1.remove();
it2.next(); // Error
```

Doesn’t try to coordinate multiple iterators changing a collection

- Multiple iterators easy for reading/viewing
- Very difficult to coordinate modifications
- A generally recurring pattern in CS: *multiple simultaneous actors are a pain in the @$@*$*
- Detect multiple concurrent modifications using `modCount` field, see Weiss
Stack: Array Based Implementation

class AStack<T>{
    private T [] stuff;
    int initial_array_size=5;
    int top=0;

    public AStack(){
    public void push(T x){}
    public void pop(){
    public T getTop(){
    public boolean isEmpty(){
    }
}

Work It

▶ Stacks: more or less functionality than ArrayList?
▶ Worst and Amortized Complexity of stack operations?
▶ Can you make the stack “iterable”, i.e., derive from Iterable<T>?
Queue: Create a LinkedQueue with Nodes

class LinkedQueue<T>{
    Node<T> front, back;
    public LinkedQueue();
    // x enters a back
    public void enqueue(T x);
    // front leaves
    public void dequeue();
    // return who’s in front
    public T getFront();
    // true when empty
    public boolean isEmpty();
}

class Node<X>{
    public X data;
    public Node<X> next;
    public Node(X data, Node<X> next)
    {
        this.data=data;
        this.next=next;
    }
}

Consider

- Worst case $O(1)$ for all ops
- Can you make the queue “iterable”, i.e., derive from Iterable<T>?
Queue: Create a `ArrayQueue`

class `ArrayQueue<T>`{
    T [ ] stuff;
    int front=0, back=0;
    int initial_array_size=5;
    public ArrayQueue();
    // x enters a back
    public void enqueue(T x);
    // front leaves
    public void dequeue();
    // return who’s in front
    public T getFront();
    // true when empty
    public boolean isEmpty();
}

class `Node<X>`{
    public X data;
    public Node<X> next;
    public Node(X data, Node<X> next)
    {
        this.data=data;
        this.next=next;
    }
}

Consider

- Worst case time complexity for all ops?
- Can you make the queue “iterable”, i.e., derive from `Iterable<T>`?