Lecture 13: Stacks & Queues

CS 62
Spring 2015
Kim Bruce & America Chambers

Stack

- Interface Stack<E> {
  - void push(E value)
  - E pop()
  - E peek()
- Example: Trays in cafeteria

- Last In - First Out (LIFO)

Stack Implementations

- ArrayList:
  - Which end should be head?
  - How complex for push, pop, peek?
- SinglyLinkedList: *Why not doubly-linked?*
  - Which end should be head?
  - How complex for push, pop, peek?
- Space differences?
  - What if there are several stacks?
- java.util.Stack based on Vector - don't use!

ArrayList Implementation

```
public class StackArrayList<E> extends AbstractStack<E>
                               implements Stack<E> {
    // The ArrayList containing the stack data.
    protected ArrayList <E> data;

    public StackArrayList()
    {
        data = new ArrayList<E>();
    }

    public void push(E item)
    {
        data.add(item);
    }

    public E pop() {
        return data.remove(size()-1);
    }

    public E peek() {
        // raise an exception if stack is already empty
        return data.get(size()-1);
    }
}
```

Queue

- FIFO: Waiting in line
- Operations:
  - enqueue (at end)
  - dequeue (from beginning)
- Examples:
  - Simulations
  - Event queue
  - Keeping track when searching

Queue Implementations

- SinglyLinkedList:
  - Which end should be front, rear?
  - How complex for enqueue, dequeue?
- ArrayList:
  - Which end should be front, rear?
  - What happens when run off end? ... when full?
  - How complex for enqueue, dequeue?
- Space differences?

```
public class QueueArray<E> extends AbstractQueue<E>
                        implements Stack<E> {
    // The array containing the queue data.
    protected E[] data;
    protected int head;
    protected int count;

    public QueueArray(int size) {
        data = (E[])new Object[size];
        head = 0;
        count = 0;
    }

    public void enqueue(E value) {
        if(count == data.length) throw new RunTimeException("Queue full");
        int tail = (head + count) % data.length;
        data[tail] = value;
        count++;
    }

    public E dequeue() {
        if (count == 0) throw new RunTimeException("Queue empty");
        E value = data[head];
        head = (head + 1) % data.length;
        count--;
        return value;
    }
}
```

What else could you do?

Dequeue

- Steque:
  - Add and remove from one end. Only add from other.
- java.util.Deque: Double-Ended Queue
  - Can add or remove from either end.
  - Resizable array implementation
  - Faster than Stack when used as stack, faster than LinkedList (doubly-linled) when used as queue.