

1999 APS105S Lecture 35: Stacks, Queues

Last Day — primitive stacks: push, pop, peek, isEmpty

Today — stacks, queues

News — no classes on Friday (Good Friday)

Projects

- TA assignments have changed -- check web pages!
- PROJECT DEMOS NEXT WEEK (April 5/6)
 - ready for marking immediately
 - SUBMIT your source code right after demo
 - *CHANGE: project report handed in the last class (april 9)*

Stacks

LIFO: Last-In, First-Out

Have you seen stacks in other places?

- web browser “back” function — stores pages in a stack
- card games — solitaire
- office work — urgent “do this now”
- crowded elevators
- Stacks naturally keep the “most recent” on top
- We can use them to reverse a series of numbers...

Using Stacks

- Find a way to use the stack... reverse numbers (again)!
- Imagine we have a Stack of integers...

```
class RevNum
{
    public static void main( String[] args )
    {
        Stack numbers = new Stack();

        // input a series of numbers, and place
        // each of them on the stack. stop
        // when a value < 0 is entered.
        while( true ) {
            int n = Stdin.getInt();
            if( n < 0 ) break;
            numbers.push( n );
        }

        // print all the numbers in the stack,
        // in reverse order! that is, the
        // most recently seen first (LIFO).
        while( !numbers.isEmpty() ) {
            int n = numbers.pop();
            System.out.println( n );
        }
    }
}
```

Using Stacks II

- Use them to shuffle a deck of cards!

```
class Card
{
    public int value; // instance var.
    public int suit; // instance var.

    public Card( int v, int s )
    {
        value = v; // check v < values.length ?
        suit = s; // check s < suits.length ?
    }

    // shared by ALL card objects
    public static String[] values =
    {
        "Ace", "2", "3", "4", "5", "6", "7",
        "8", "9", "10", "Jack", "Queen", "King"
    };

    // shared by ALL card objects
    public static String[] suits =
    {
        "Hearts", "Diamonds", "Clubs", "Spades"
    };

    public String toString()
    {
        return values[value] + " of "
            + suits[suit];
    }
}
```

The program

```
public class Shuffle
{
    public static void shuffle( Stack deck )
    { // create 2 shuffling piles
        Stack pileA = new Stack();
        Stack pileB = new Stack();

        while( !deck.isEmpty() ) {
            if( Math.random() < 0.5 )
                pileA.push( deck.pop() );
            else
                pileB.push( deck.pop() );
        }

        while( !pileA.isEmpty() ) //pileA to deck
            deck.push( pileA.pop() );
        while( !pileB.isEmpty() ) //pileB to deck
            deck.push( pileB.pop() );
    }

    public static void main( String[] args )
    { // create the deck
        Stack deck = new Stack();
        int v,s;
        for( s=0; s<Card.suits.length; s++ )
            for( v=0; v<Card.values.length; v++ )
                deck.push( new Card(v,s) );

        // shuffle the deck
        shuffle( deck );

        while( !deck.isEmpty() ) // print deck
            System.out.println( deck.pop() );
    }
}
```

Queues

Like stacks, queues are another data structure (ADT)
Unlike stacks, queues are processed “in order”

e.g. concert ticket lineup, bank lineup
note: British people rightly call these “queues”, not lineups.
another example — internet routers
uses — time queue in simulators

- Recall
- Stacks are LIFO, “reverse order”
- Queues are FIFO, First-In First-Out, “in-order”

Primitive Operations

- add to queue, “enqueue”
- remove from queue, “dequeue”
- peek front of queue
- is queue empty?

How to build a queue?

Use a linked list

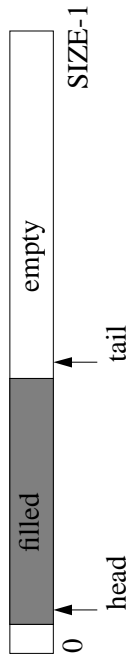
- always add to the tail
- always remove from the head

```
class QueueLinkedList {
    ... as LinkedList before ...
    public void enqueue( QueueElem e )
    { append( e ); }

    public QueueElem dequeue()
    { QueueElem e = head; head = head.next;
      e.next = null; return e; }
}
```

Other Ways to Build a Queue?

we can use an array:



```
class QueueArray
{
    private QueueElem[] array;
    private int head, tail;
    private final int SIZE = 100;

    public QueueArray()
    {
        head=0; tail=0;
        array = new QueueElem[SIZE];
    }

    public void enqueue( QueueElem e )
    {
        array[tail++] = e;
    }

    public QueueElem dequeue()
    {
        if( head == tail )
            return null;
        QueueElem e = array[head++];
        return e;
    }
}
```

- But what about when tail goes past end of array?
- that’s a special case, this program must be fixed!