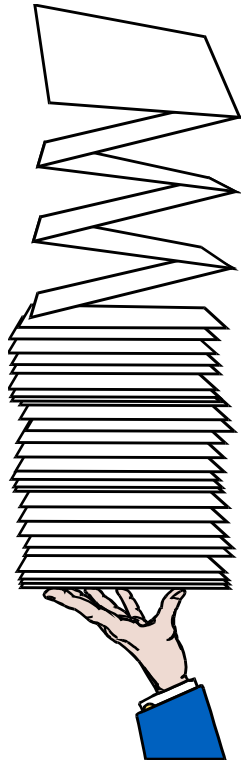
Stacks



- Lists with "only one end."
- Objects are added and removed from the list at the same end.
- Stacks are also called Last-In-First-Out (LIFO) data structures.

Operations on a stack… (informal):

add, remove, test if stack is empty/full.

Array Implementation

```
class LongStack {
  protected long[] theArray;
  protected int size; //maxsize of stack array
  protected int tos; //top of stack -1 if
                     // empty

  public LongStack(int s) {
    size = s;
    theArray = new long[s];
    tos = -1;
  }
```

Array Implementation

```
  public void push(long num) {
     // increment top, insert number
     theArray[++tos] = num;
  }

  public long pop() {
     // take number off the top
     return theArray[tos--];
  }
   public boolean isEmpty() {
     // true if stack is empty
     return (tos == -1);
    }

  public boolean isFull() {
     // true if stack is full
     return (tos==theArray.length-1);
   }
}
```

LongStack -- Issues

- What happens when pop an empty stack?
- What happens when push a full stack?
- This stack can only hold longs
  - If I want a stack to hold doubles I need to do it all over again
  - Can’t I have a generic stack that can hold anything?

Empty?? Throw an Exception

- May "throw" a predefined exception
  - Or write your own, e.g.,

```
public class StackException extends Exception {
  public StackException() {super("Stack is empty.");}
  public StackException(String s) {super(s);}
}
```

- To throw an exception:
  - `if (tos==-1) throw new StackException();`
  - **OR**

```
try { return theArray[tos--]; }
catch (Exception e) {
   throw new StackException("poor Judd is
dead ...");
}
```

Full??

increase the array size

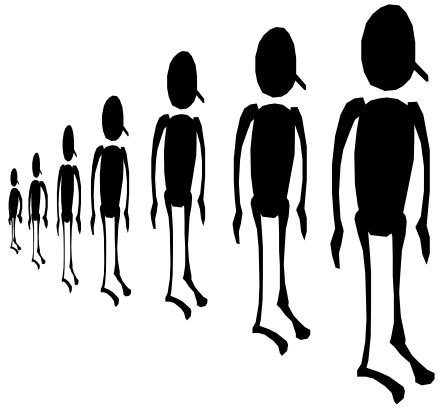
```
protected void double()
{
   long[] old = theArray;
   size = size * 2;
   theArray = new long[size];
   for(int i=0; i<tos; i++)
      theArray[i]=old[i];
}

public void push(long num) {
     // increment top, insert number
     if (tos+1 == size) doubleSize();
     theArray[++tos] = num;
   }
```

Generic Stack??
Use Objects

- Putting data into an object stack
  - All classes inherit from Object so that is easy
  - long, int, double do not inherit from Object
    - But Double, Integer, Long, ... do
      - Double n = new Double(5.3);
- Getting data back out
  - need to cast to the correct class
    - (Color) obStack.pop()
  - For numbers:
    - double d = ((Number)obStack.pop()).doubleValue();

Queues

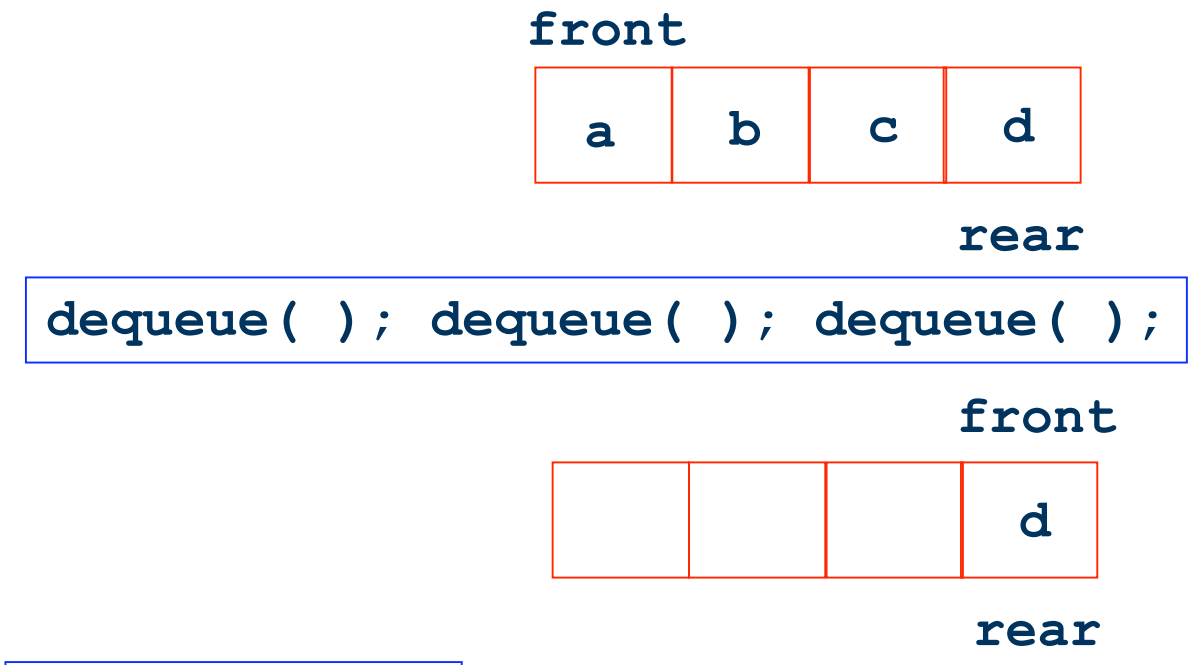


Like queues in real life --- elements enter at the rear of queue, advance through the queue and come out at the front (FIFO).

```
public interface Queue {
  public void enqueue(Object x);
                                        // Put x at the back
  public Object front() throws QueueEmptyException;
                                       // Return front element
  public Object dequeue() throws QueueEmptyException;
                          // Return and remove front element
  public boolean isEmpty();
}
```

Array Implementation of Queues

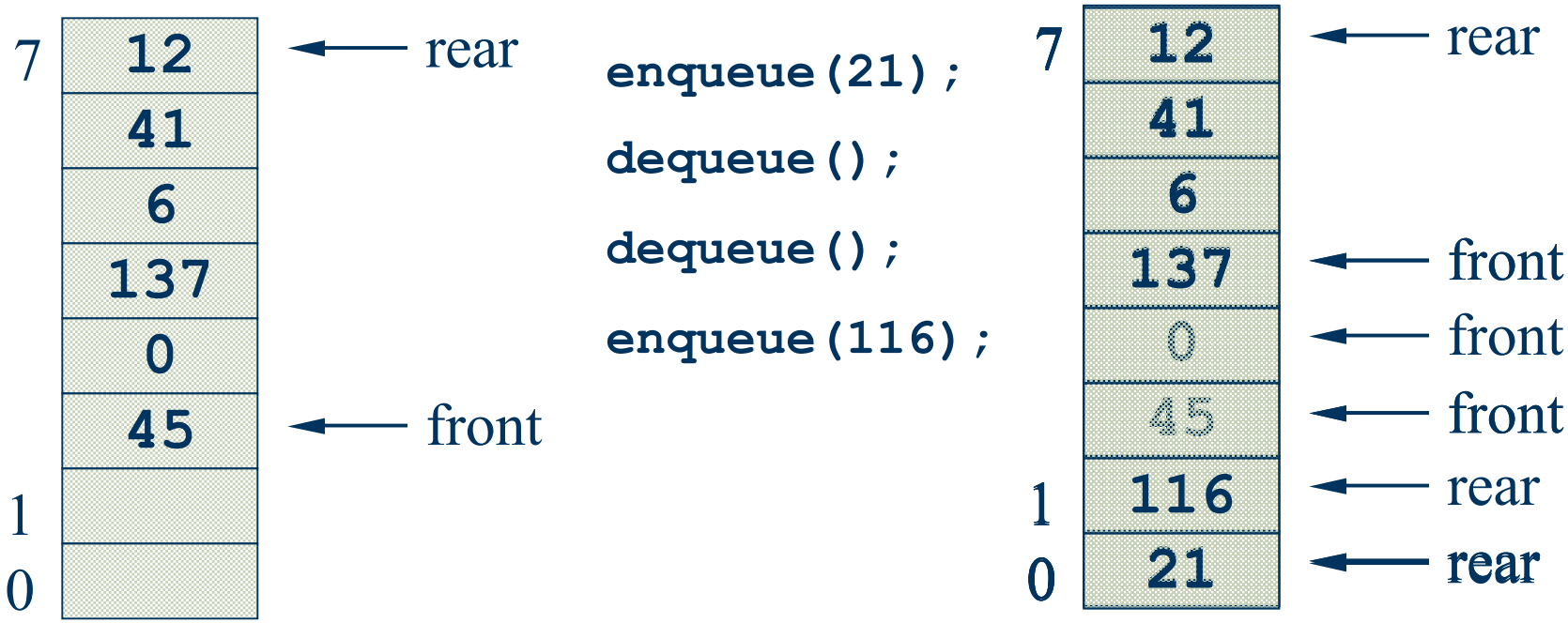
The problem of drift:



No place to insert x even though array has room!

Solution: Circular Queues

- Make array wrap around. When front/rear reaches end of array allow it to go to the beginning.



Circular Queues

- What is the relation between **`front`** and **`rear`** for an empty queue?
- When queue has one element they point to the same place. For an **`enqueue()`** operation we increment **`rear`**.
- So, when queue is empty **`rear`** is (**`front - 1`**) (in a wrapped-around sense).
- What is the relation between front and back when queue is full? Can we tell a full queue from an empty queue?

Circular Array Implementation

```
class QueueAr {
  private Object[] theArray;
  private int currentSize, front, rear;

  static final int DEFAULT_CAPACITY = 128;

  public QueueAr( ) {
    theArray = new Object[DEFAULT_CAPACITY];
    currentSize = 0; // Number of elements in the queue
    front = 0;
    rear = -1;
  }
  public boolean isEmpty() {
    return ( currentSize == 0 );
  }
```

```
public void enqueue( Object x ) {
  //if (currentSize == theArray.length) doubleArray();
  rear = increment(rear);
  theArray[rear] = x;
  currentSize++;
}
public Object dequeue( ) throws QueueEmptyException {
  if ( isEmpty() ) throw new QueueEmptyException();
  currentSize--;
  Object returnValue = theArray[front];
  front = increment(front);
  return returnValue;
}
public int increment(int r) {
    r++;
    if (r==theArray.length)
        return 0;
    return r;
  }
```