

Stacks and Queues

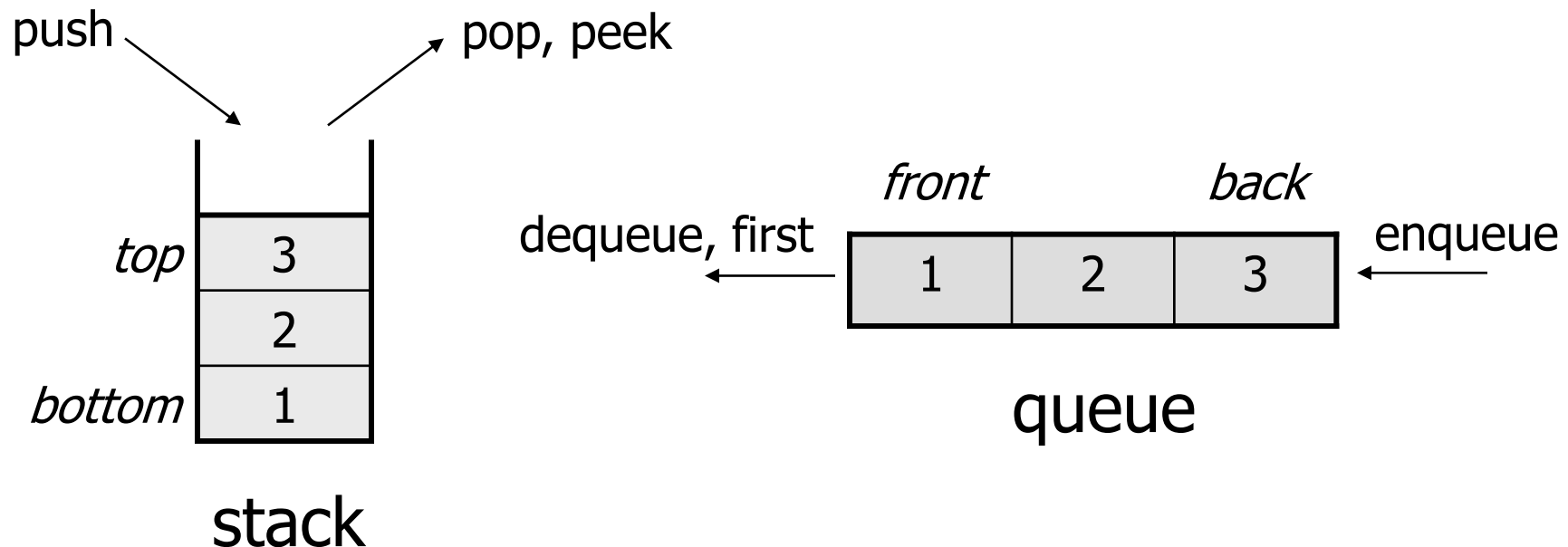


Abstract Data Types (ADTs)

- ▶ **Abstract Data Type (ADT):** A specification of a collection of data and the operations that can be performed on it.
 - Describes *what* a collection does, not *how* it does it
- ▶ We don't know exactly how a stack or queue is implemented, and we don't need to.
 - We just need to understand the idea of the collection and what operations it can perform.
 - Stacks usually implemented with arrays
 - Queues often implemented with a linked list

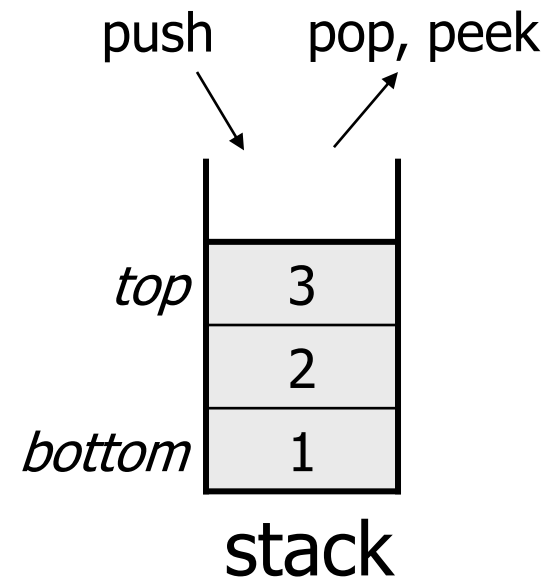
Stacks and Queues

- ▶ Some collections are constrained so clients can only use optimized operations
 - **stack**: retrieves elements in reverse order as added
 - **queue**: retrieves elements in same order as added



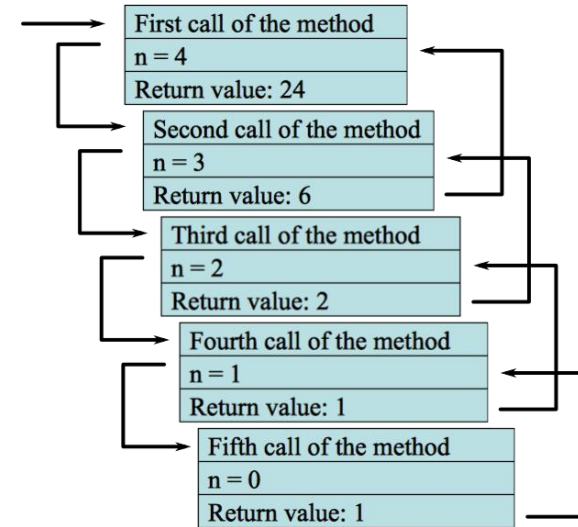
Stacks

- ▶ **Stack:** A collection based on the principle of adding elements and retrieving them in the opposite order.
 - Last-In, First-Out ("LIFO")
 - Elements are stored in order of insertion.
 - We do not think of them as having indexes.
 - Client can only add/remove/examine the last element added (the "top").
- ▶ basic stack operations:
 - **push:** Add an element to the top.
 - **pop:** Remove the top element.
 - **peek:** Examine the top element.



Stacks in Computer Science

- ▶ Programming languages and compilers:
 - method calls are placed onto a stack
 - *call=push*
 - *return=pop*
 - compilers use stacks to evaluate expressions
- ▶ Matching up related pairs of things:
 - find out whether a string is a palindrome
 - examine a file to see if its braces { } match
 - convert "infix" expressions to pre/postfix
- ▶ Sophisticated algorithms:
 - searching through a maze with "backtracking"
 - many programs use an "undo stack" of previous operations



Stack Limitations

- ▶ You cannot loop over a stack in the usual way.

```
Stack<Integer> s = new Stack<Integer>();  
...  
for (int i = 0; i < s.size(); i++) {  
    do something with s.get(i);  
}
```

- ▶ Instead, pop each element until the stack is empty.

```
// process (and destroy) an entire stack  
while (!s.isEmpty()) {  
    do something with s.pop();  
}
```

What happened to my stack?

- ▶ Suppose we're asked to write a method `max` that accepts a Stack of Integers and returns the largest Integer in the stack:

```
// Precondition: !s.isEmpty()
public static void max(Stack<Integer> s) {
    int maxValue = s.pop();
    while (!s.isEmpty()) {
        int next = s.pop();
        maxValue = Math.max(maxValue, next);
    }
    return maxValue;
}
```

- The algorithm is correct, but what is wrong with the code?

What happened to my stack?

- ▶ The code destroys the stack in figuring out its answer.

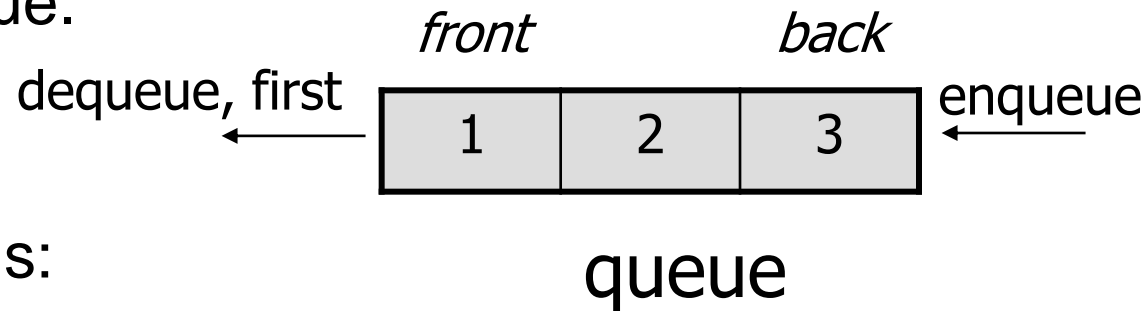
- To fix this, you must save and restore the stack's contents:

```
public static void max(Stack<Integer> s) {  
    Stack<Integer> backup = new Stack<Integer>();  
    int maxValue = s.pop();  
    backup.push(maxValue);  
    while (!s.isEmpty()) {  
        int next = s.pop();  
        backup.push(next);  
        maxValue = Math.max(maxValue, next);  
    }  
    while (!backup.isEmpty()) {    // restore  
        s.push(backup.pop());  
    }  
    return maxValue;  
}
```


Queues

‣ **Queue:** Retrieves elements in the order they were added.

- First-In, First-Out ("FIFO")
- Elements are stored in order of insertion but don't have indexes.
- Client can only add to the end of the queue, and can only examine/remove the front of the queue.



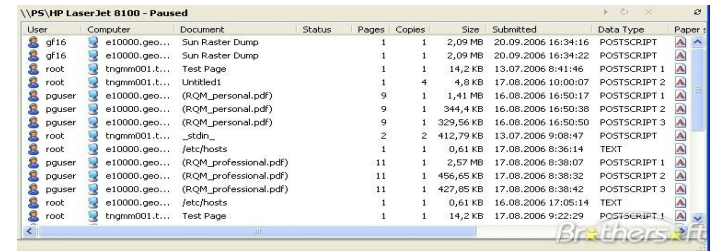
‣ Basic queue operations:

- **enqueue** (add): Add an element to the back.
- **dequeue** (remove): Remove the front element.
- **first**: Examine the front element.

Queues in Computer Science

▶ Operating systems:

- queue of print jobs to send to the printer
- queue of programs / processes to be run
- queue of network data packets to send



User	Computer	Document	Status	Pages	Copies	Size	Submitted	Data Type	Paper
gf16	e10000.geo...	Sun Raster Dump		1	1	2,09 MB	20.09.2006 16:34:16	POSTSCRIPT	
gf16	e10000.geo...	Sun Raster Dump		1	1	2,09 MB	20.09.2006 16:34:22	POSTSCRIPT	
root	tngrm001.t...	Test Page		1	1	14,2 KB	13.07.2006 8:41:46	POSTSCRIPT 1	
root	tngrm001.t...	Untitled1		1	4	4,8 KB	17.08.2006 10:00:07	POSTSCRIPT 2	
pguser	e10000.geo...	(RQM_personal.pdf)		9	1	1,41 MB	16.08.2006 16:50:17	POSTSCRIPT 1	
pguser	e10000.geo...	(RQM_personal.pdf)		9	1	344,4 KB	16.08.2006 16:50:38	POSTSCRIPT 2	
pguser	e10000.geo...	(RQM_personal.pdf)		9	1	329,56 KB	16.08.2006 16:50:50	POSTSCRIPT 3	
root	tngrm001.t...	_stdin_		2	2	412,79 KB	13.07.2006 9:08:47	POSTSCRIPT	
root	e10000.geo...	/etc/hosts		1	1	0,61 KB	17.08.2006 8:36:14	TEXT	
pguser	e10000.geo...	(RQM_professional.pdf)		11	1	2,57 MB	17.08.2006 8:38:07	POSTSCRIPT 1	
pguser	e10000.geo...	(RQM_professional.pdf)		11	1	456,65 KB	17.08.2006 8:38:32	POSTSCRIPT 2	
pguser	e10000.geo...	(RQM_professional.pdf)		11	1	427,85 KB	17.08.2006 8:38:42	POSTSCRIPT 3	
root	e10000.geo...	/etc/hosts		1	1	0,61 KB	16.08.2006 17:05:14	TEXT	
root	tngrm001.t...	Test Page		1	1	14,2 KB	17.08.2006 9:22:29	POSTSCRIPT 1	

▶ Programming:

- modeling a line of customers or clients
- storing a queue of computations to be performed in order

▶ Real world examples:

- people on an escalator or waiting in a line
- cars at a gas station (or on an assembly line)

Using Queues

- ▶ As with stacks, must pull contents out of queue to view them.

```
// process (and destroy) an entire queue
while (!q.isEmpty()) {
    do something with q.remove();
}
```

- To examine each element exactly once.

```
int size = q.size();
for (int i = 0; i < size; i++) {
    do something with q.remove();
    (including possibly re-adding it to the queue)
}
```

- Why do we need the `size` variable?

Mixing Stacks and Queues

- ▶ We often mix stacks and queues to achieve certain effects.
 - Example: Reverse the order of the elements of a queue.

```
Queue<Integer> q = new LinkedList<Integer>();  
q.add(1);  
q.add(2);  
q.add(3);                                // [1, 2, 3]
```

```
Stack<Integer> s = new Stack<Integer>();  
while (!q.isEmpty()) {                    // Q -> S  
    s.push(q.remove());  
}  
while (!s.isEmpty()) {                    // S -> Q  
    q.add(s.pop());  
}  
System.out.println(q);                   // [3, 2, 1]
```

Java Stack Class

<code>Stack<T>()</code>	constructs a new stack with elements of type T
<code>push(value)</code>	places given value on top of stack
<code>pop()</code>	removes top value from stack and returns it; throws <code>EmptyStackException</code> if stack is empty
<code>peek()</code>	returns top value from stack without removing it; throws <code>EmptyStackException</code> if stack is empty
<code>size()</code>	returns number of elements in stack
<code>isEmpty()</code>	returns <code>true</code> if stack has no elements


```
Stack<String> s = new Stack<String>();  
s.push("a");  
s.push("b");  
s.push("c");           // bottom ["a", "b", "c"] top  
System.out.println(s.pop()); // "c"
```

- Stack has other methods that are off-limits (not efficient)

Java Queue Interface

<code>add (value)</code>	places given value at back of queue
<code>remove ()</code>	removes value from front of queue and returns it; throws a <code>NoSuchElementException</code> if queue is empty
<code>peek ()</code>	returns front value from queue without removing it; returns <code>null</code> if queue is empty
<code>size ()</code>	returns number of elements in queue
<code>isEmpty ()</code>	returns <code>true</code> if queue has no elements

```
Queue<Integer> q = new LinkedList<Integer>();  
q.add(42);  
q.add(-3);  
q.add(17);           // front [42, -3, 17] back  
System.out.println(q.remove()); // 42
```



- **IMPORTANT:** When constructing a queue, you must use a new `LinkedList` object instead of a new `Queue` object.
 - Because `Queue` is an *interface*.

Exercises

- ▶ Write a method `stutter` that accepts a queue of Integers as a parameter and replaces every element of the queue with two copies of that element.
 - `[1, 2, 3]` becomes `[1, 1, 2, 2, 3, 3]`
- ▶ Write a method `mirror` that accepts a queue of Strings as a parameter and appends the queue's contents to itself in reverse order.
 - `[a, b, c]` becomes `[a, b, c, c, b, a]`

