



Data Structures and algorithms

STACKS AND QUEUES

WEEK 1

LECTURER: DR ANWARUL PATWARY

Outline

- ▶ What are stacks?
- ▶ Basic operations of stack in Java
- ▶ What are Queues?
- ▶ Basic operations of stack in Java.

What is a Stack

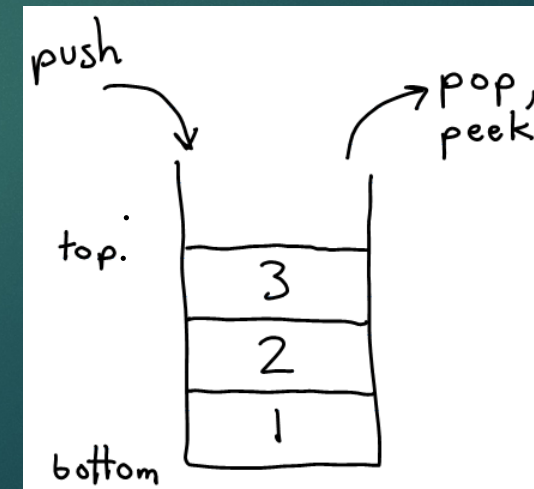
- ▶ Stack of Books



Stack ADT

4

- ▶ A stack is data structure of ordered items such that items can be inserted and removed only at one end.
- ▶ **stack**: a list with the restriction that insertions/deletions can only be performed at the top/end of the list
 - ▶ Last-In, First-Out ("LIFO")
 - ▶ The elements are stored in order of insertion but we do not think of them as having indexes.
 - ▶ The 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.



Applications of Stacks

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

Class Stack

<code>Stack<E>()</code>	constructs a new stack with elements of type E
<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<Integer> s = new Stack<Integer>();  
s.push(42);  
s.push(-3);  
s.push(17);      // bottom [42, -3, 17] top
```

```
System.out.println(s.pop()); // 17
```


Stack limitations/idioms

7

- ▶ Remember: You can't loop over a stack like you do a list.

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

- ▶ Instead, you pull contents out of the stack to view them.
 - ▶ Idiom: Remove each element until the stack is empty.

```
while (!s.isEmpty()) {  
    do something with s.pop();  
}
```


What is a Queue?



Queues

- ▶ What is a queue?

- ▶ A data structure of ordered items such that items can be inserted only at one end and removed at the other end.

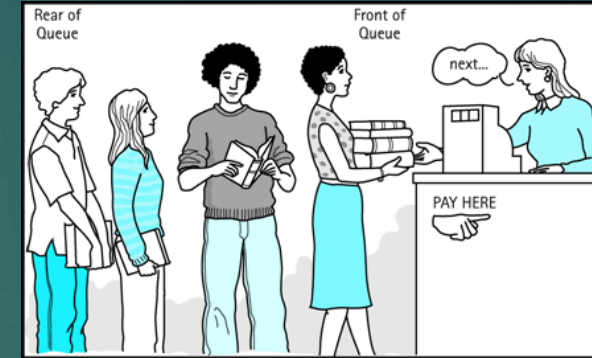
- ▶ Example

- ▶ A line at the supermarket

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

Queue ADT interface

- ▶ Let's write our own implementation of a queue.
 - ▶ As is done in the Java Collection Framework, we will define queues as an ADT by creating a queue interface.

```
public interface Queue<E> {  
    void clear();  
    boolean isEmpty();  
    E peek();  
    E remove();           // remove from back  
    void add(E value);     // add to front  
    int size();  
}
```



```
public class ArrayQueue<E> implements Queue<E> {
    private E[] elements;
    private int size;
    ...
}
```

- ▶ A queue is tough to implement efficiently with an unfilled array.
 - ▶ The array is fast to add/remove at the end, but slow at the front.

```
queue.add(26);    // client code
queue.add(-9);
queue.add(14);
queue.remove();
```

[illegible]