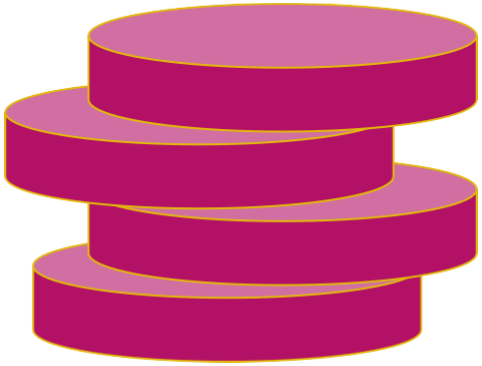
Stack

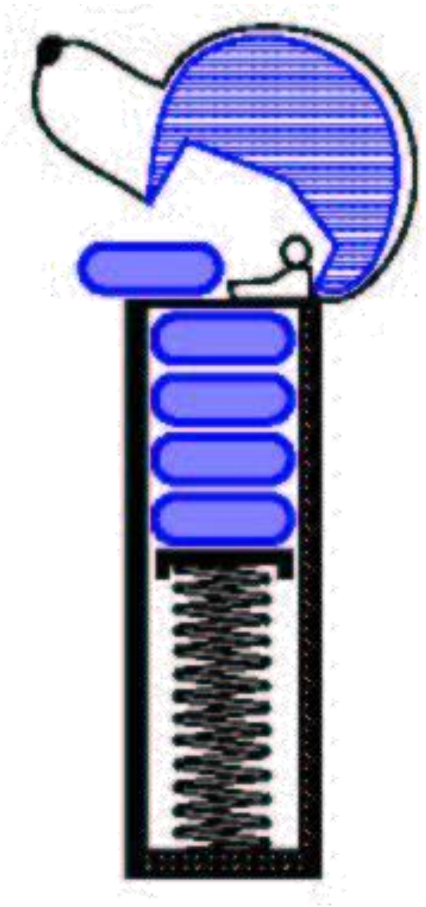
Part 1 : Stack Array

Stacks



Stacks

- A ***stack*** is a collection of objects that are inserted and removed according to the ***last-in, first-out*** (***LIFO***) principle
- Think of a spring-loaded plate dispenser
- insert , access or remove the most recently inserted object that remains at the “top” of the stack.



Data Structures and Algorithms

BY : DR. JAWAD ALZAMILY

Applications of Stacks

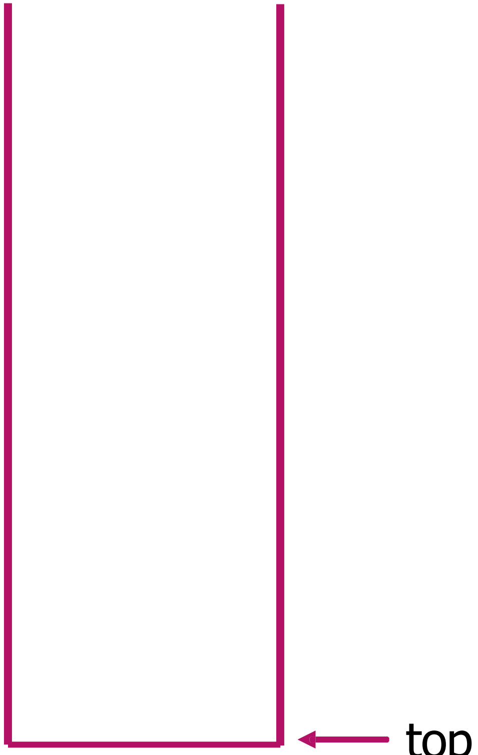
- **Applications**
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Chain of method calls in the Java Virtual Machine

The Stack ADT

- **a stack is an abstract data type (ADT) that supports**
- **Two update methods:**
  - push(object): inserts an element
  - object pop(): removes and returns the last inserted element
- **Auxiliary accessor methods:**
  - object top(): returns the last inserted element without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored

Example:

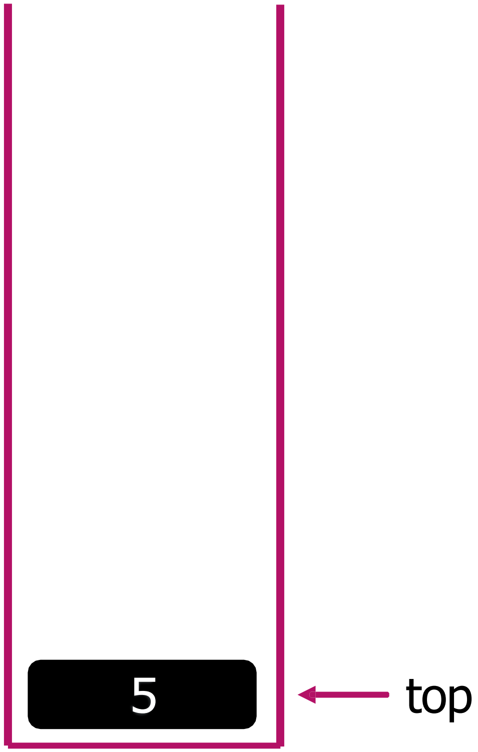
- Empty Stack



Returned value

Example:

- Empty Stack
- Push((5

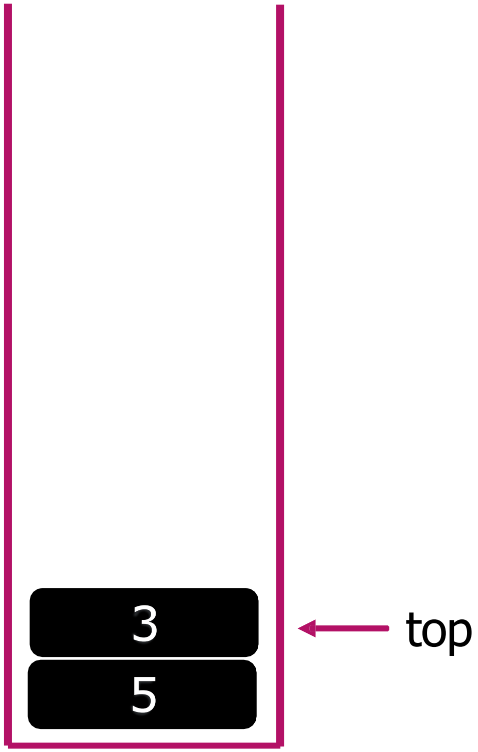


Returned value

Example:

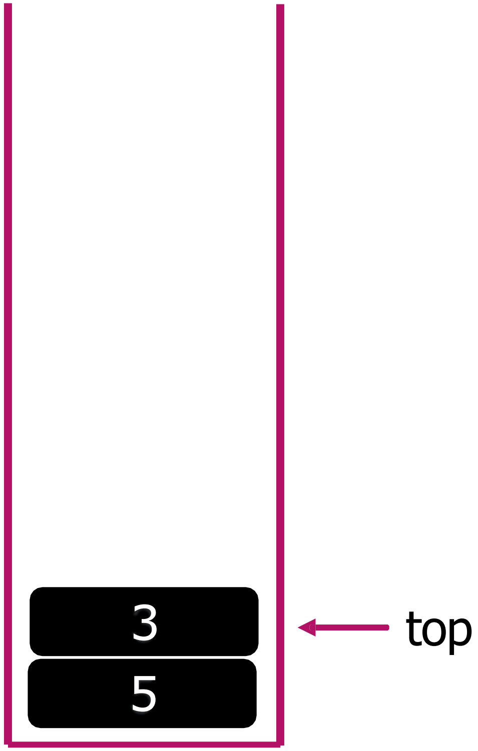
- Empty Stack
- Push((5
- Push((3

Returned value



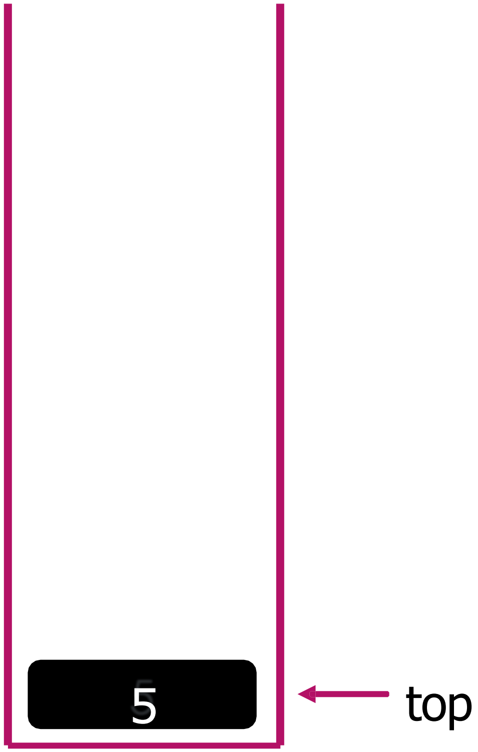
Example:

▸ Empty Stack	Returned value
▸ Push((5	-----
▸ Push((3	-----
▸ Size()	2



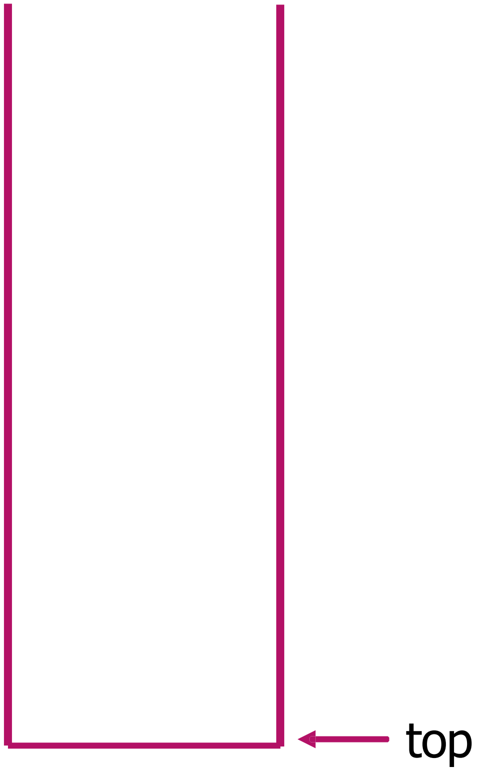
Example:

	Returned value
Empty Stack	
Push((5	-----
Push((3	-----
Size()	2
Pop()	3
isEmpty()	false



Example:

Empty Stack	Returned value
Push((5	-----
Push((3	-----
Size()	2
Pop()	3
isEmpty()	false
Pop()	5
Pop()	null
isEmpty()	true



Stack Interface in Java

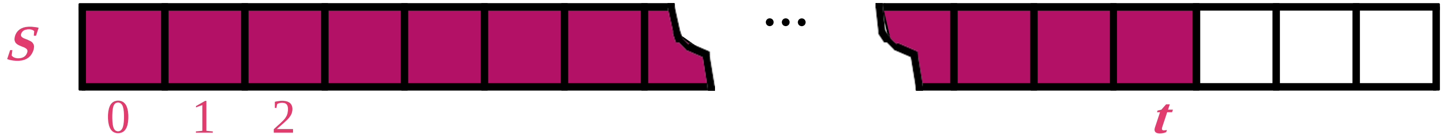
- Java interface corresponding to our Stack ADT
- Assumes null is returned from top() and pop() when stack is empty
- Different from the built-in Java class java.util.Stack

Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top (t) element

```
Algorithm size()
    return $t + 1$

Algorithm pop()
    if isEmpty() then
        return null
    else
        $t \leftarrow t - 1$
        return $S[t + 1]$
```



Array-based Stack

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException
  - Limitation of the array-based implementation

```
Algorithm push (0)
if $t == S.length - 1$ then
    throw IllegalStateException
  else
    $t \leftarrow t + 1$
    $S[t] \leftarrow o$
```

Array-based Stack in Java

```
 //Stack implementation in Java
class Stack {
  int arr [ ];
  int top;
  int capacity;
  //Creating a stack Stack(int
 size} (
   arr = new int[size ;[
   capacity = size ;
   top = -1;
 }
```

```
 //Check if the stack is empty
 public Boolean isEmpty() {
   return (top == -1);
 }

  //Check if the stack is full
 public Boolean isFull() {
 return (top == capacity - 1) ;
 }

 //Utility function to return the size of the stack
 public int size () {
return (top+1) ;
   }
```

Array-based Stack in Java

```
// Add elements into stack
Public void push ( int x)
{
   If (isFull)
     System.out.println("OverFlow\nProgram rminated”);
   else
     {
      System.out.println("Inserting " + x);
      arr[++top] = x;
      }
}
```

Array-based Stack in Java

```
 //Remove element from stack
public int pop()
{
If (isEmptry())
System.out.println("STACK EMPTY");
else
  return arr[top--] ;
}
```

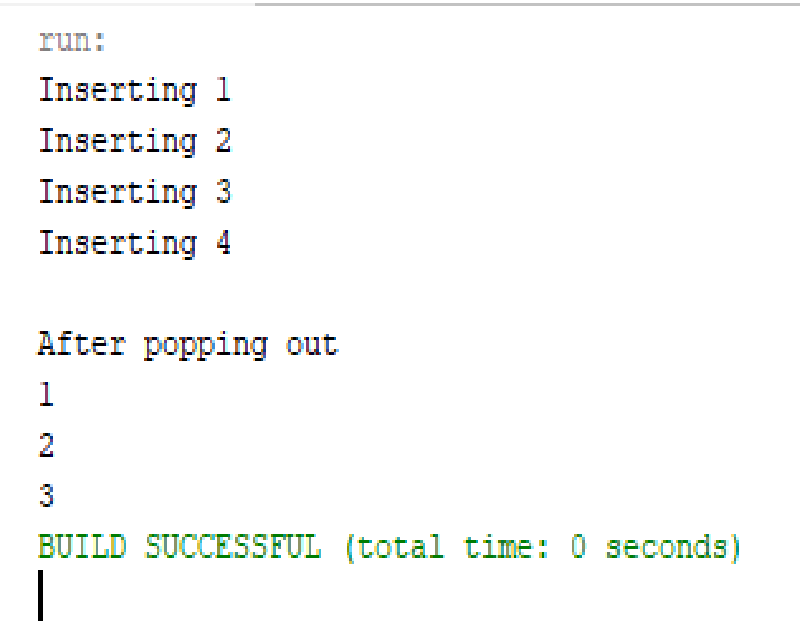
Array-based Stack in Java

```
public void printStack() {
  for (int i = 0; i <= top; i++){
  System.out.println(arr[ i ]); }
 }
```

Array-based Stack in Java

```
public static void main(String[ ] args)  {
  Stack stack = new Stack(5);
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack .push(4);

    stack.pop;()
    System.out.println("\nAfter popping out  ;("
    stack.printStack;()
{
```



```
run:
Inserting 1
Inserting 2
Inserting 3
Inserting 4

After popping out
1
2
3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Performance and Limitations

- **Performance**
  - Let ***n*** be the number of elements in the stack
  - The space used is ***O***(***n***)
  - Each operation runs in time ***O***(***1***)
- **Limitations**
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception