DATA STRUCTURES

By

Dr. Yasser Abdelhamid

الجامعة المصرية
للتعلم الإلكتروني الأهلية

EELU

THE EGYPTIAN E-LEARNING UNIVERSITY

RESOURCES

- http://javatpoint.com/

OUTLINE

- Stacks

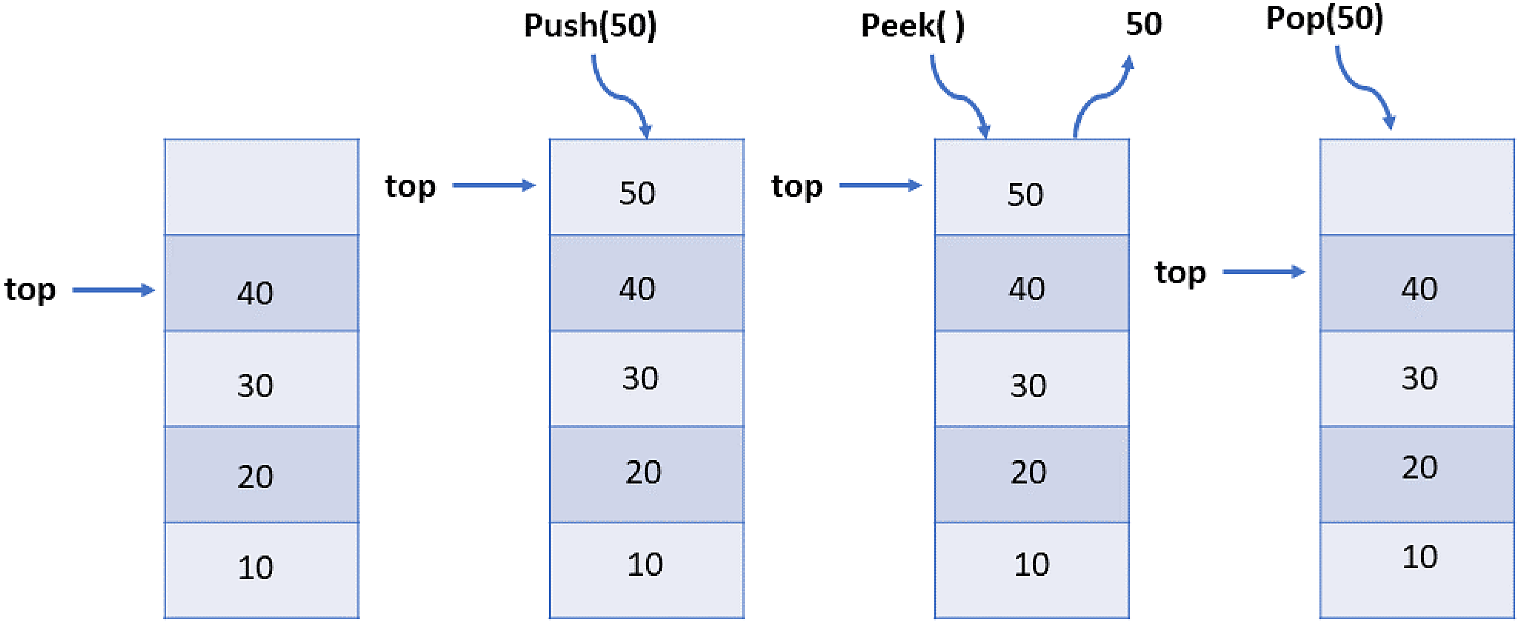
STACK ADT

- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the last-in first-out (LIFO) scheme
- Think of a spring-loaded plate dispenser.
- Main stack operations:
  - push(Object o): inserts element o
  - pop(): removes and returns the last inserted element
- Auxiliary stack operations:
  - top(): returns the last inserted element without removing it
  - size(): returns the number of elements stored
  - isEmpty(): a Boolean value indicating whether no elements are stored

STACK ADT CONT.

- Attempting the execution of an operation of ADT may sometimes cause an error condition, called an exception.
- Exceptions are said to be “thrown” by an operation that cannot be executed.
- In the Stack ADT, operations pop and top cannot be performed if the stack is empty
- Attempting the execution of pop or top on an empty stack throws an EmptyStackException

STACKS



Push(50)
Peek( )
50
Pop(50)
top
40
30
20
10
top
50
40
30
20
10
top
50
40
30
20
10
top
40
30
20
10

EXERCISE

- Describe the output of the following series of stack operations

  Push(8)

  Push(3)

  Pop()

  Push(2)

  Push(5)

  Pop()

  Pop()

  Push(9)

  Push(1)

APPLICATIONS OF STACKS

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Saving local variables when one function calls another, and this one calls another, and so on.
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

CALLING AUXILIARY FUNCTIONS

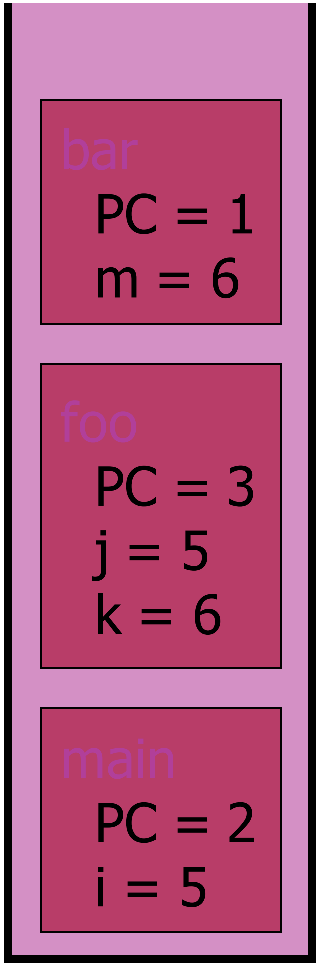
- **When a function is called, the run-time system pushes on the stack a frame containing**
  - Local variables and return value
  - Program counter, keeping track of the statement being executed
- **When a function returns, its frame is popped from the stack and control is passed to the method on top of the stack**

```
main() {
  int i;

  i = 5;
  foo(i);
  }

foo(int j)
{
  int k;
  k = j+1;
  bar(k);
  }

bar(int m)
{
  …
  }
```



bar

PC = 1

m = 6

foo

PC = 3

j = 5

k = 6

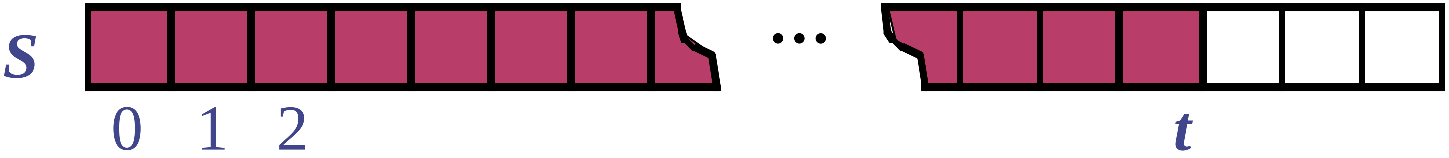
main

PC = 2

i = 5

IMPLEMENTATION OF STACKS USING ARRAYS

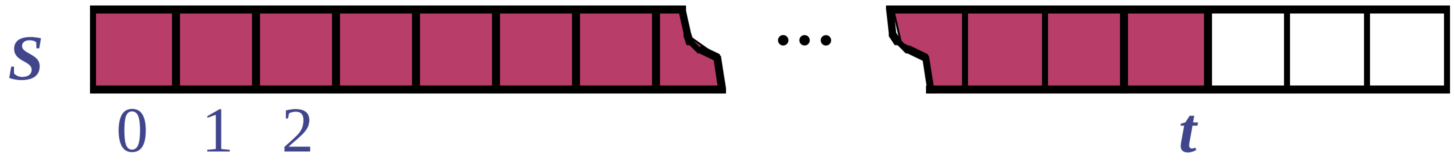
- A simple way of implementing the Stack ADT uses an array.
- We add elements from left to right.
- A variable keeps track of the index of the top element.



GET NUMBER OF ELEMENTS IN STACK

- This method returns the number of elements in the stack.

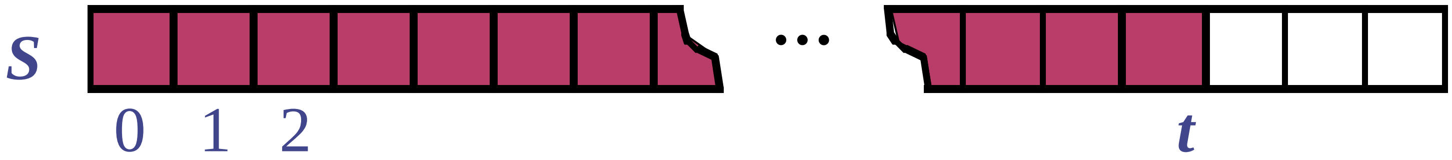
```
Algorithm size()
  return t + 1
```



POP METHOD

- **Returns the element at the top of the stack.**

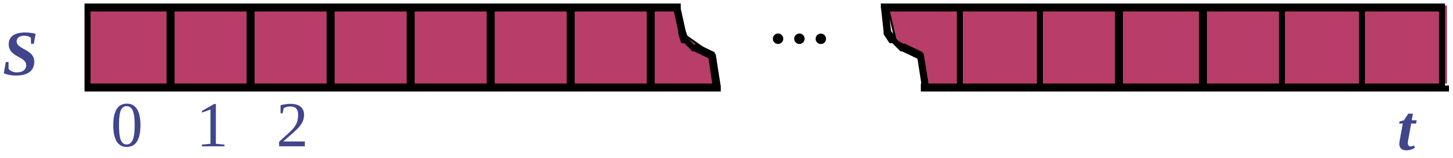
```
Algorithm pop()
  if isEmpty() then
    throw EmptyStackException
   else
    t ← t − 1
    return S[t + 1]
```



PUSH METHOD

- Add a new element at the top of the stack.
- The array storing the stack elements may become full. So, a push operation will then throw a FullStackException
  - Limitation of the array-based implementation
  - Not intrinsic to the Stack ADT

```
Algorithm push(object o)
  if t = S.length − 1 then
    throw FullStackException
   else
    t ← t + 1
    S[t] ← o
```



PERFORMANCE AND LIMITATIONS

Array-based implementation of stack ADT

- **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

IMPLEMENTATION IN JAVA

```
class Stack
{
    int top;
    int maxsize = 10;
    int[] arr = new int[maxsize];

   Stack()   // constructor
     {
        top = -1;
     }
```

top
-1

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]

maxsize
10

ISEMPTY()

```
boolean isEmpty()
   {
       return (top < 0);
   }
```

top
-1

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]

PUSH

```
boolean push (Scanner sc) {
        if(top == maxsize-1) {
            System.out.println("Overflow !!");
            return false;
        }
        else {
            System.out.println("Enter Value");
            int val = sc.nextInt();
            top++;
            arr[top]=val;
            System.out.println("Item pushed");
            return true;
        }
}
```

maxsize
10

top
-1

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]

POP

```
boolean pop () {
      if (top == -1) {
         System.out.println("Underflow !!");
         return false;
      }
      else {
         top --;
         System.out.println("Item popped");
         return true;
      }
}
```

maxsize
10

top
-1

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]

THANK YOU