ITI 1121. Introduction to Computing II

Stack: concept

by

Marcel Turcotte

Version February 3, 2020

Preamble

Preamble

Overview

Overview

Stack: concept

We're interested in all aspects of **stacks** in programming. A stack is an **abstract data type** similar to physical stacks. It's a linear data structure such that the **access to the data is only from one end**, called the **top** of the stack, and textbfone element at a time.

General objective :

- This week you'll be able to describe a stack.

Preamble

Learning objectives

Learning objectives

- **Describe** the stack concept in computer science.
- **Justify** the role of a stack in solving a computer problem.
- **Implement** a stack using an array.

Readings:

- Pages 147-150 of E. Koffman and P. Wolfgang.

Preamble

Plan

Plan

1 Pr[Pleaseinsertintopreamble]ambule

2 Théorie

3 Implémentation à l'aide d'un tableau

4 Prologue

Theory

Theory

Discussion

Discussion

- Discuss the implementation of mechanisms to **undo** and **redo** operations in a text editor?
  - What information do you need to save?
  - In what order is this information saved and accessed?
- What do these mechanisms have in common with those of a browser software allowing the return to a previous or already visited URL address?

Theory

Definition

Theory

A **stack** is an abstract data type similar to physical stacks.

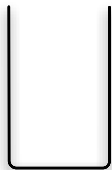
- Books
- Plates
- Trays
- PEZ

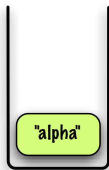
The analogy with the plate dispenser found in a cafeteria is particularly interesting, because **access is limited to the top item** and **you have to remove the top plates one by one in order to access the bottom ones.**

Definition

A **stack** is a **linear** data structure such that the data is accessed from only one end, called the **top** of the stack, and one element at a time.

This structure is often called **LIFO**, from “ *last-in first-out* ”.





s = new StackImpl() **s.push("alpha")** **s.push("bravo")** **s.pop()**

Theory

Applications

Applications

Stacks are the data structures behind the following applications :

- the development of **compilers**, and the analysis of formal languages in general;
- the implementation of **backtracking algorithms** used by automatic theorem proving systems, game algorithms and artificial intelligence;
- memory management during **program execution**;
- support “ **undo** ” operations in word processing programs or the “ **back** ” button on your Internet browser.

Theory

Operations

Operations

The **basic operations** are :

push: **add** an item on the stack

pop: **remove** and **return** the top element

isEmpty: check that the stack is empty.

Abstract data type (ADT)

```java
public interface Stack<E> {
    E push(E elem);
    E pop();
    E peek();
    boolean isEmpty();
}
```

As we saw in the last lesson, the interface can receive a parameter of type!

```java
class Mystery {
    public static void main(String[] args) {

        Stack<String> stack;
        stack = new StackImplementation<String>();

        for (int i=0; i<args.length(); i++) {
            stack.push(args[ i ]);
        }

        while (! stack.empty()) {
            System.out.print(stack.pop());
        }
    }
}
```

What does it produce "`java Mystery a b c d e`"?

Remarks

- Removed items appear in reverse order.
- The following construction is common when using stacks:

```
while (! stack.empty()) {
    element = stack.pop();
    ...
}
```

- You have to be careful not to loop *ad infinitum*, for example by omitting the **pop()**..

Operations (continued)

peek: **returns** the value of the element on top **without removing it**;

Implementation using an array

Implementation of a stack using an array

Implementation using an array

Learning objectives :

- **Implementing** a stack using an array.

Readings:

- Pages 70–75 of E. Koffman and P. Wolfgang.

Implementations

Think about a possible implementation?

There are **two** major families of implementations :

- based on **arrays**
- based on **linked elements**

```
Stack<Integer> s;

s = new ArrayStack<Integer>();
s = new DynamicArrayStack<Integer>();
s = new LinkedStack<Integer>();
```

Discussion

- One of the proposed implementations uses an array, **why don't we just use an array** to design algorithms? **What are the advantages?**

Implementation using an array

Instance variables

Implementation using an array : ArrayStack

- What are the instance variables?

- What are the types of elements in this array?

Implementation using an array

Strategy

Implementation using an array : strategy

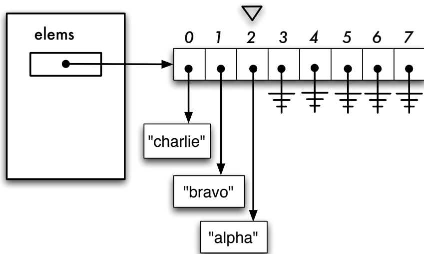
What will be the strategy for saving items in the stack?

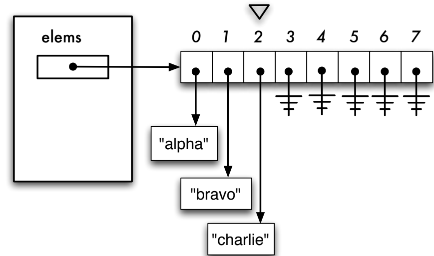
- In what order will you save the items?
- Where to add a new element (**push**) and which element to remove (**pop**)?

Take a few moments to strategize.

Implementation using an array : order of elements

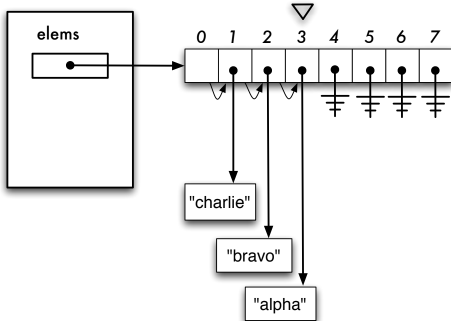
- In what **order** should the elements be added?
- Is that **important**?

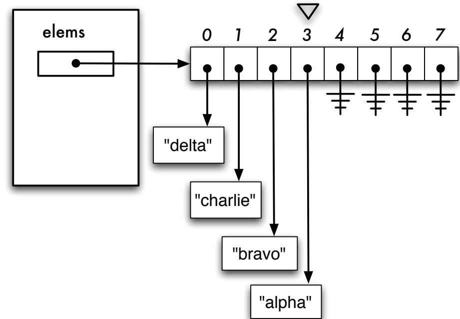




Implementation using an array : bottom on the right

What do you think about it?





Implementation using an array : finding an empty cell

Proposal. How do you feel about that?

- The method **push** visits each cell of the array, from left to right, and inserts the new element in the first cell whose value is **null**.
- The method **pop** visits each cell of the table, from right to left, and removes the element in the first cell whose value **is not null**.

Implementation using an array : ArrayStack

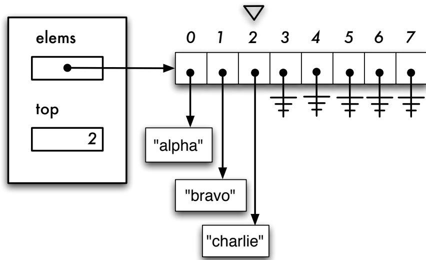
- Ideally, the stack should **memorize** the index of the rightmost occupied cell.
- In object-oriented programming, how does an object, in this case a stack, remember a value?

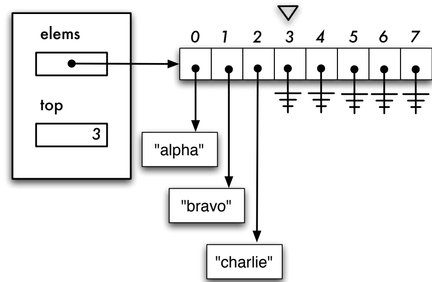
Summary

- We use an **array** to save the stack elements.
- The **order** of the elements is important.
  - The **top element** will be the one on the **more right**.
- We need two instance variables.
  - One of them is a **reference** to an array
  - The other variable allows us to determine the location of the **top element**.

Implementation using an array: top

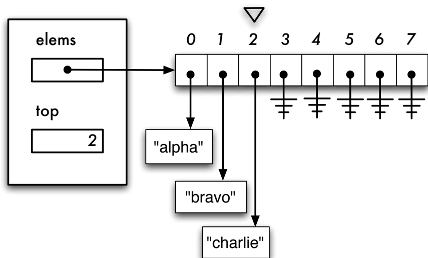
- Does the variable **top** designate the top element or the empty cell following the top element?
- What's the **type** for this variable?





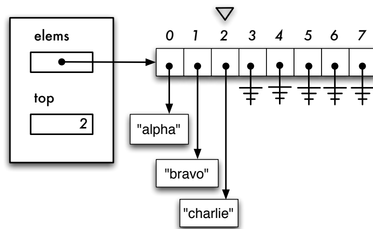
Adding an element when top designates top element

What are the **steps** for **adding** an element?



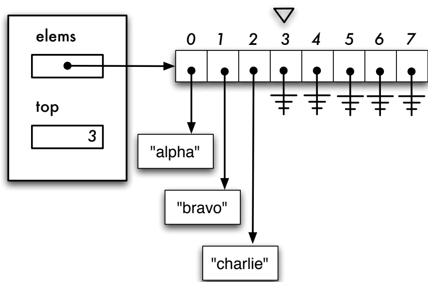
The variable top contains the index to top element.

What is the value of **top** if the stack is **empty**?



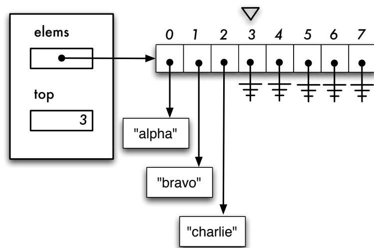
Adding an element when top designates the empty cell that follows the element on top

What are the **steps** for adding an element?



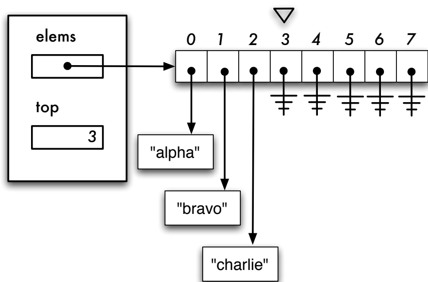
The variable top designates the empty cell that follows the top element

What is the value of **top** if the stack is **empty**?

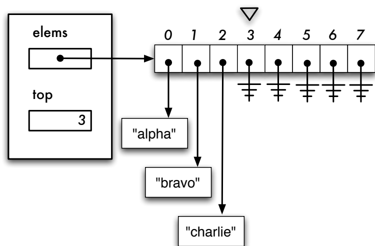


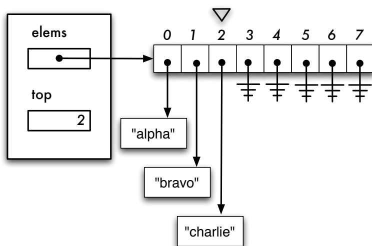
Remove an element when top designates the number of elements

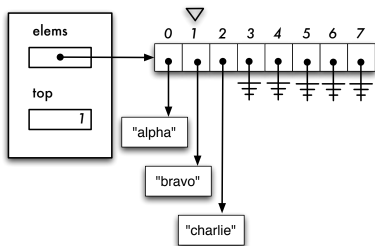
What are the **steps** for **removing** an item?

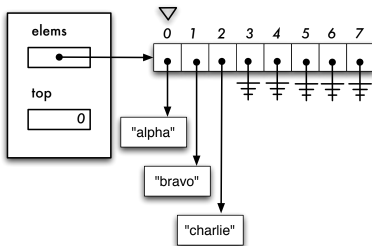


What do you think?









Memory leaks

Memory leak

Steps for removing an element :

- Decrementing the value of **top**
- Save in a local (temporary) variable the element of the array located at the **position** designated by **top**.

- Returning the saved value

The proposed approach leads to memory leaks!

Indeed, the **garbage collector** cannot reclaim the memory space associated with the **accessible** objects.

What solution do you propose?

Java implementation

Designing a generic type : ArrayStack<E>

Examples of stack usage:

```
Stack<String> s1;
name = new ArrayStack<String>(100);

Stack<Time> s2;
name = new ArrayStack<Time>(1024);

s1.push("alpha");
s2.push(new Time( 23,0,0 ));

String a;
a = s1.pop();
```

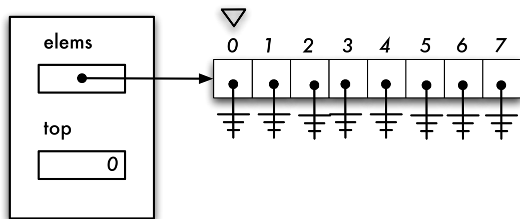
Implementation using an array : ArrayStack

- **Java** implementation of a stack using an array.
- Here, **top** refers to the **empty cell** that follows the top element.
- The implementation where **top** designates the top element is left as an exercise.

Implementation using an array : ArrayStack

Implementation using an array : ArrayStack

Give the implementation of the **constructor** that will produce the memory diagram below:



Give the **memory diagram** corresponding to the execution of the constructor below:

```
public class ArrayStack implements Stack {
    private E[] elements;
    private int top;

    public ArrayStack(int capacity) {
        top = 0;
    }
}
```

Arrays and generics

Arrays and generics

- In order for programs to run on **virtual machines of previous generations (before 1.5)**, we cannot create arrays with generic element types.
- We still have **a problem to solve**.

Arrays and generics

Solution :

```
public class ArrayStack<E> implements Stack<E> {
    private E[] elems;
    private int top;

    public ArrayStack( int capacity ) {
        elems = (E[]) new Object[ capacity ];
        top = 0;
    }
    // ...
```

Alas, this solution will produce a **warning when compiling**.

```
Note: ArrayStack.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Arrays and generics

Local directives to remove a warning at compile time :

```
public class ArrayStack<E> implements Stack<E> {
    private E[] elems;
    private int top;

    @SuppressWarnings( "unchecked" )
    public ArrayStack( int capacity ) {
        elems = (E[]) new Object[ capacity ];
        top = 0;
    }
    // ...
```

Which is preferable to global warning suppression!

```
> javac -Xlint:unchecked ArrayStack.java
```

ArrayStack<E> : isEmpty()

ArrayStack<E> : E peek()

ArrayStack<E> : void push(E element)

ArrayStack<E> : E pop()

ArrayStack : isEmpty()

ArrayStack : isEmpty() (take 2)

ArrayStack : E peek()

Dynamic arrays

Weakness of our implementation

- Our implementation has a **major weakness**, what is it?

Dynamics arrays

Some programming languages allow us to **change the size of the arrays** when running programs. These languages use the technique below.

Pitfall

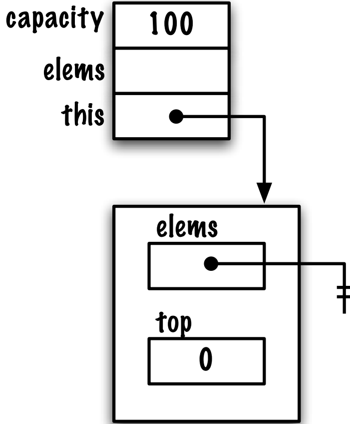
Pitfall!

```
public class ArrayStack<E> implements Stack<E> {

    // Instance variables
    private E[] elems;
    private int top;

    // Constructor
    public ArrayStack( int capacity ) {
        E[] elems = (E[]) new Object[ capacity ];
        top = 0;
    }
    // Returns true if this ArrayStack is empty
    public boolean isEmpty() {
        return top == 0;
    }
    // ...
}
```

Pitfall

Working memory for ArrayStack

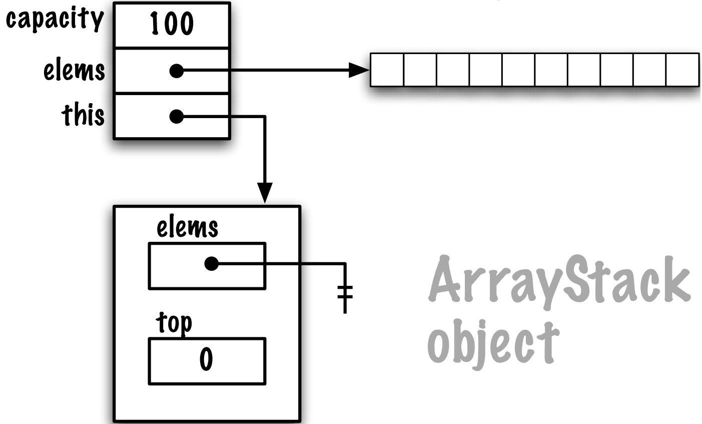
formal parameter(s)
local variable(s)



ArrayStack object

Activation Frame for ArrayStack

formal parameter(s)
local variable(s)

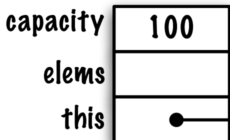


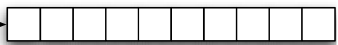
Pitfall

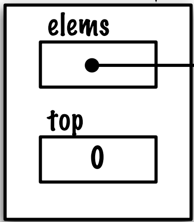
Activation Frame for ArrayStack

formal parameter(s)

local variable(s)







ArrayStack object

Summary

- A **stack** is used when you want to process the elements in **reverse order**.
- There are two main implementations : **using an array** and **using linked elements**.

Next module

- **Stack** : applications

References I

E. B. Koffman and Wolfgang P. A. T.
***Data Structures: Abstraction and Design Using Java*.**
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

Université d'Ottawa