

Sesión 10: Pilas Acotadas

Programación 2

Ángel Herranz

Febrero 2019




Universidad Politécnica de Madrid

En capítulos anteriores

 Tema 1: Clases y Objetos

 Tema 2: Colecciones acotadas de Objetos

En capítulos anteriores

-  Tema 1: Clases y Objetos
-  Tema 2: Colecciones acotadas de Objetos
-  “Programa que lea de la entrada estándar órdenes para insertar y borrar canciones y que imprima la lista resultante final”

En el capítulo de hoy

Reverse Polish Notation 

En el capítulo de hoy

Reverse Polish Notation

“Programa que lea de la entrada estándar líneas en notación RPN y que por cada línea calcule el valor y lo imprima en la salida estándar”

Ejemplos de entrada

$$3 \ 2 \ - \ =$$

$$5 \ 3 \ 2 \ * \ + \ =$$

$$15 \ 7 \ 1 \ 1 \ + \ - \ / \ 3 \ * \ 2 \ 1 \ 1 \ + \ + \ - \ =$$

$$12 \ 3 \ - \ 3 \ / \ =$$

$$9 \ 5 \ 3 \ + \ 2 \ 4 \ ^ \ - \ + \ =$$

El *token* = indica el final de la expresión

Ejemplos de entrada y salida esperada

1

11

5

3

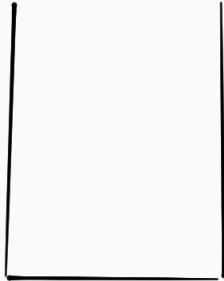
1

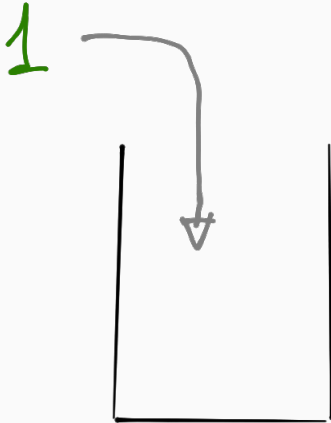
En Wikipedia

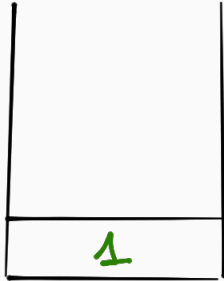
```
for each token in the postfix expression:
    if token is an operator:
        operand_2 = pop from the stack
        operand_1 = pop from the stack
        result = evaluate token with operand_1 and operand_2
        push result back onto the stack
    else if token is an operand:
        push token onto the stack
result = pop from the stack
```

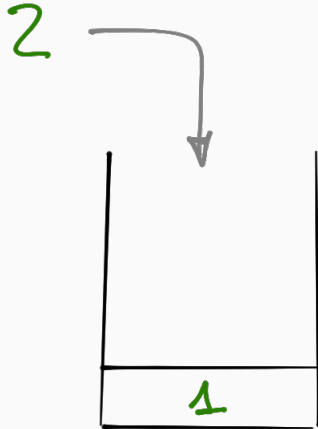

¿Tokens?

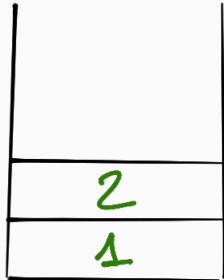
- Cada uno de los elementos “atómicos” en los que podemos dividir una *entrada*
- En nuestro ejemplo:
- **Operando:** números enteros
- **Operadores:** "+", "-", "*", "/", "^"
- **Fin:** "="
- Usamos `hasNextInt()` (detecta si hay un operando) `hasNext()` (detecta si hay más tokens), `nextInt()` (consume el operando), `next()` (consume operador o fin)

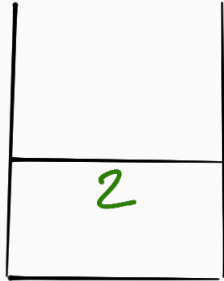


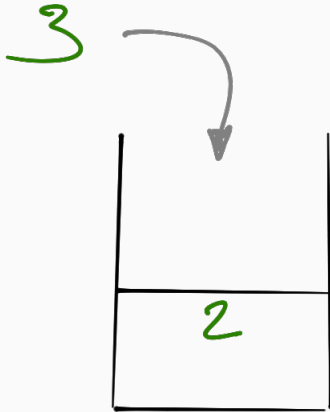


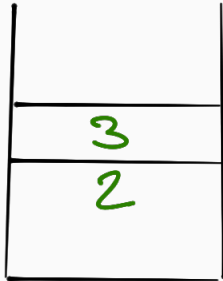


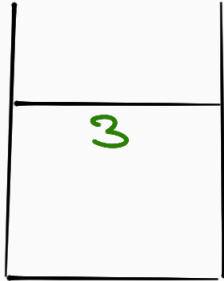


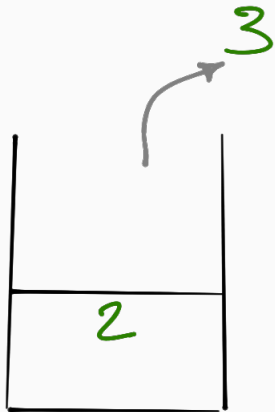


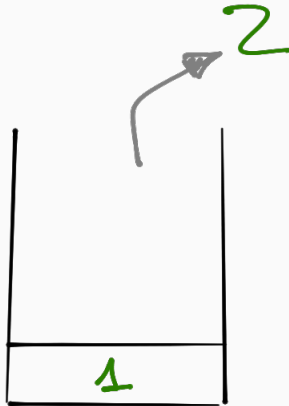


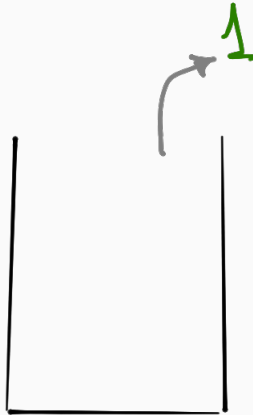













Ejecutad a mano cada expresión

```
public class RPN {  
    public static void main(String[] args) {  
        // Se crea un objeto Scanner para poder  
        // leer de la entrada estándar  
        java.util.Scanner stdin =  
            new java.util.Scanner(System.in);  
  
        int[] pila = new int[1000];  
        int siguiente = 0;  
        while (stdin.hasNext()) {  
            if (stdin.hasNextInt()) {  
                pila[siguiente] =  
                    stdin.nextInt()
```

Pilas “acotadas”

- Sólo *arrays* de Java
- Asumimos un límite (pongamos 1000)


 Clase para representar pilas de enteros y así no tener que manejar los arrays directamente:

```
PilaEnteros pila = new PilaEnteros();  
p.apilar(42);
```

 ¿API?

Pilas “acotadas”

- Sólo *arrays* de Java
- Asumimos un límite (pongamos 1000)

 Clase para representar pilas de enteros y así no tener que manejar los arrays directamente:


```
PilaEnteros pila = new PilaEnteros();  
p.apilar(42);
```

 ¿API?

- Crear, apilar, desapilar, cima

Pilas “acotadas”


- Sólo *arrays* de Java
- Asumimos un límite (pongamos 1000)

 Clase para representar pilas de enteros y así no tener que manejar los arrays directamente:

```
PilaEnteros pila = new PilaEnteros();  
p.apilar(42);
```

 ¿API?

- Crear, apilar, desapilar, cima

 Reimplementar RPN haciendo uso de la nueva clase que representa pilas de enteros