PROGRAMACION 3

TUDAI

CURSADA 2024

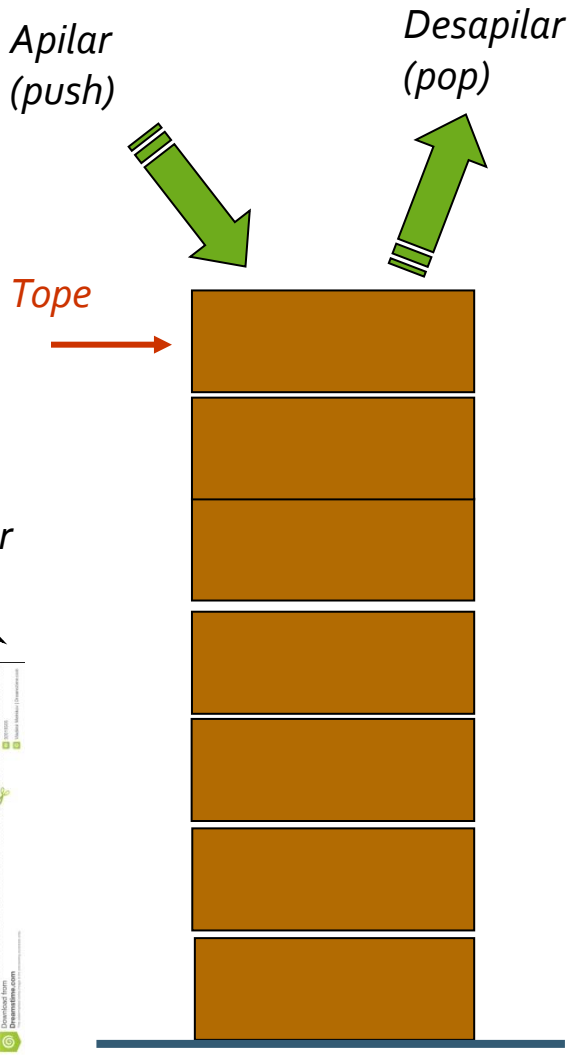
Federico Casanova
Facultad de Ciencias Exactas - UNICEN

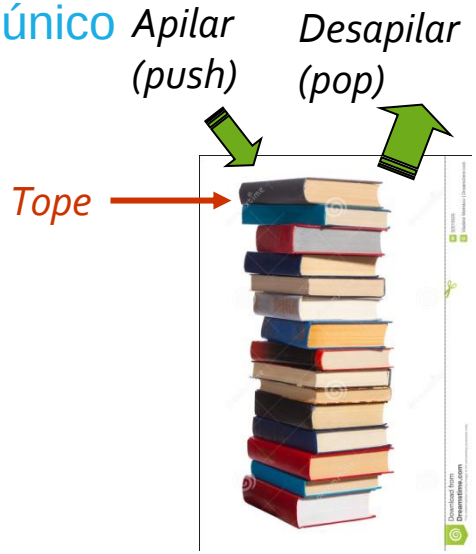
Iniciar Grabación

RECURSIVIDAD

Repaso: Estructura de Datos PILA

- Es una estructura en la cual el acceso está limitado al elemento más recientemente insertado.
- La inserción y extracción de elementos de la pila siguen el principio UEPS (Ultimo en Entrar es el Primero en Salir)
- Su nombre proviene de una metáfora de las pilas en el mundo real: pila de libros, pila de CDs, pila de expedientes, …
- El último elemento insertado es el único disponible para operar.





Ejecución de programas

```
public static void main(String[] args)
{
  G( );
}

public void G ( )
{
  a = 5 * F(22);
}

public int F (int x)
{
  h = x*x;
  return h;
}
```

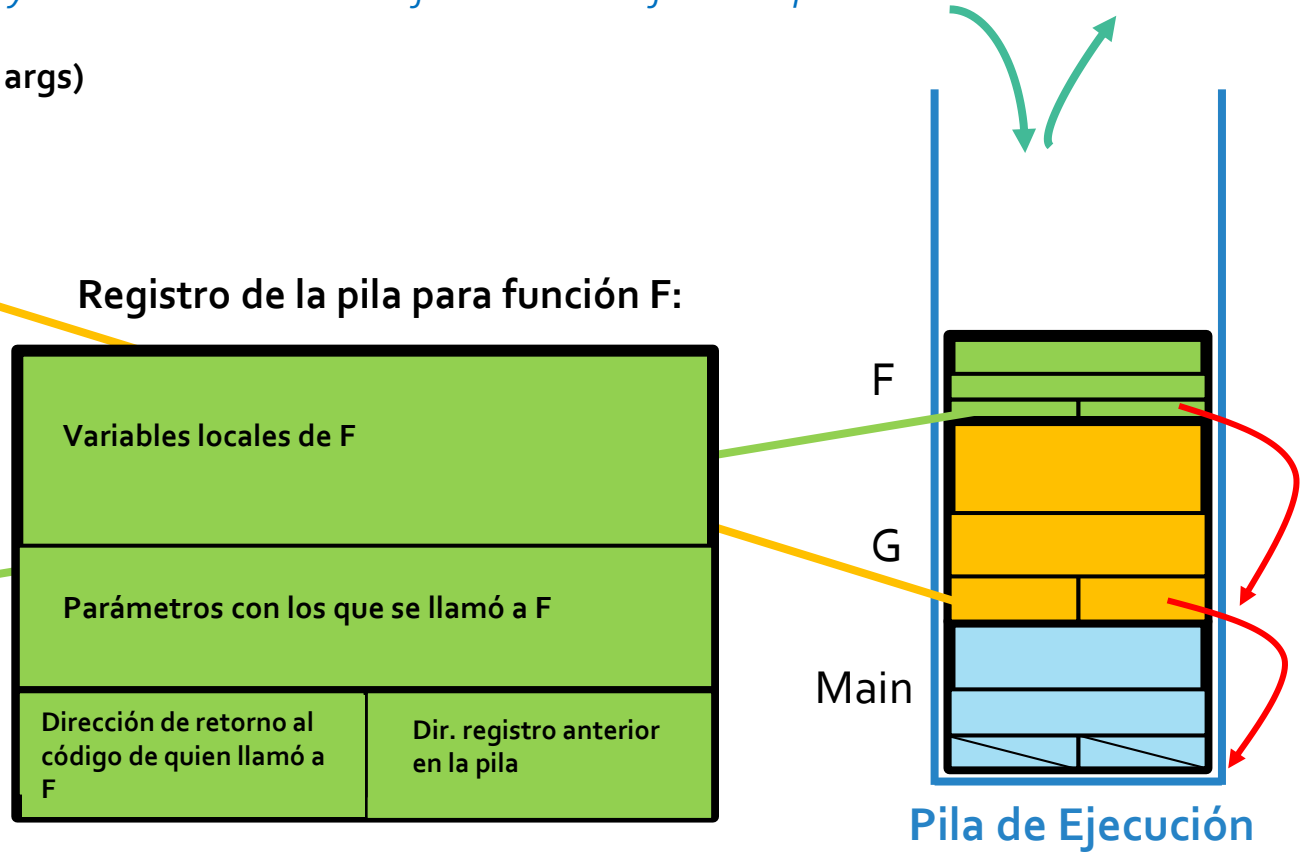
Ejecución de programas

Modelo de pila de ejecución

Describiremos un modelo simplificado de pila de ejecución:

El Sistema Operativo mantiene una pila por cada aplicación en ejecución .
Por cada llamada a función (o método) se agrega un registro en la pila.
Dicho registro contendrá varios datos: los parámetros que se le pasan al método o función, la dirección de retorno (dónde debe continuar ejecutando luego de finalizar), la dirección del elemento anterior en la pila (ya que los registros no son todos del mismo tamaño), y las variables locales de la función. Esto define el lo que se llama ámbito de la función.

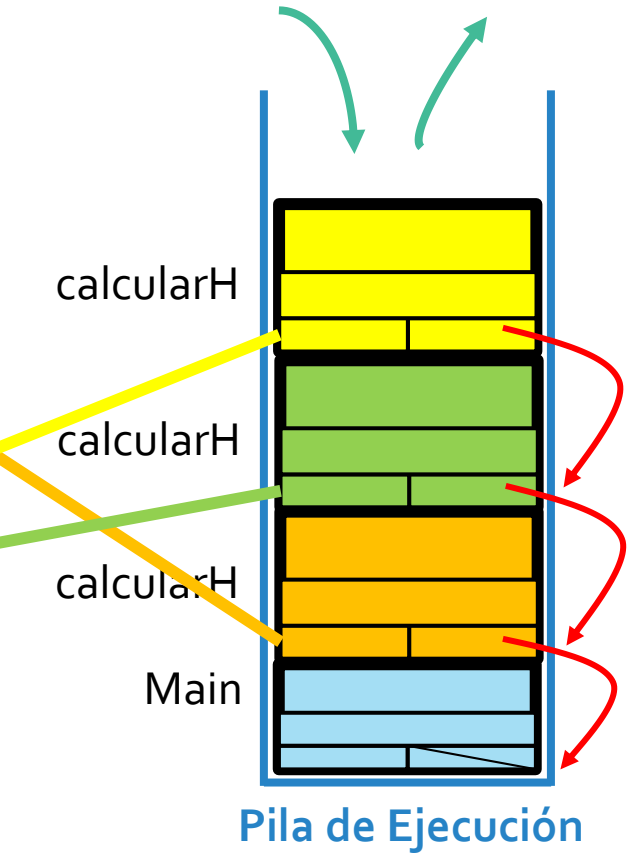
```
public static void main(String[] args)
{
  x = x * G();
}

public int G()
{
  a = 5 * F(22);
}
```



Llamada recursiva

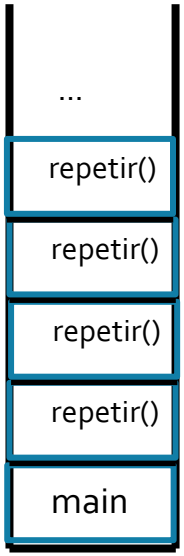
Modelo de pila de ejecución

```
public static void main(String[] args)
{
  X = calcularH(10) + 7;
}

public void calcularH(int x)
{
  a = 5 * calcularH(x+22);
}
```



Ejemplo recursión

```
public class Recursividad {

void repetir() {
   repetir();
  }

public static void main(String[] ar) {
  Recursividad re=new Recursividad();
  re.repetir();
  }
}
```



- Qué pasa con la pila de ejecución ????

"Exception in thread "main" java.lang.StackOverflowError"

Ejemplo recursión

```
public class Recursividad {

 void imprimir(int x) {
   System.out.println(x);
   imprimir(x-1);
 }

 public static void main(String[] ar) {
   Recursividad re=new Recursividad();
   re.imprimir(5);
  }
}
```

Por ej.
imprimir(5):

5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
.....

- Qué pasa con la pila de ejecución ????

Eventualmente dará "Exception in thread "main" java.lang.StackOverflowError"

Entonces cómo hacemos ?? Cuál es el error ??:
Les falta un caso base o condición de corte de la recursión.
Sin eso continuará su ejecución indefinidamente hasta agotar el espacio de la pila de ejecución (Stack Overflow)

Ejemplo recursión

Implementar un método recursivo que imprima en forma **descendente** desde **x** hasta **1** de uno en uno.

```
public class Recursividad {

 void imprimir(int x) {
   if (x>0){
      System.out.println(x);
      imprimir(x-1);
     }
 }

 public static void main(String[] ar) {
    Recursividad re=new Recursividad();
    re.imprimir(5);
 }
}
```

Condición de corte de la recursión

Por ej si x=5:

5
4
3
2
1

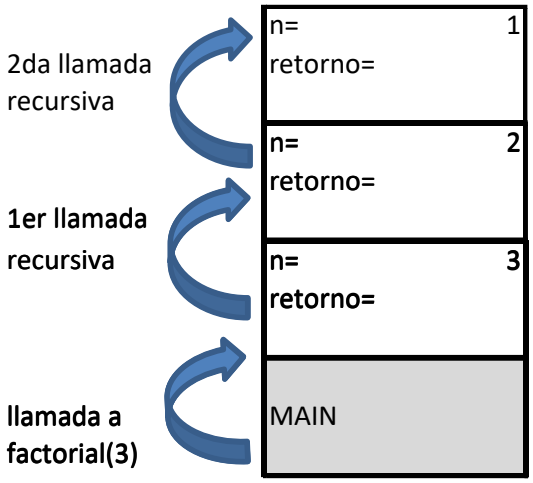
Conviene hacer este método recursivo en vez de iterativo ?

Ejemplo recursión – calcular factorial

Dado n un número natural
n!=n*(n-1)*(n-2)*…*1

Definición recurrente:
n! = n * (n-1)!
0!=1

Por ejemplo:
5!=5*4! =120
4!=4*3! =24
3!=3*2! =6
2!=2*1! =2
1!=1*0! =1
0!=1
Entonces 5!=5*4*3*2*1=120




```
public double factorial(int n) {

     if (n>1)
         return (n * factorial (n-1));
     else
         return 1;
}
```

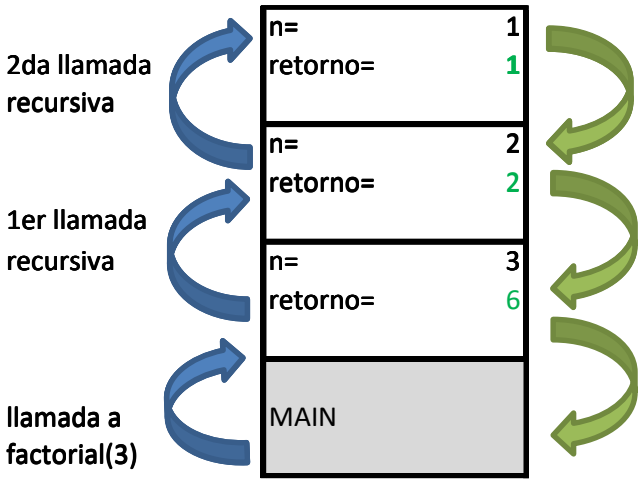
Por ejemplo hacemos una llamada desde Main a la función factorial(3)

Ejemplo recursión – calcular factorial

Dado n un número natural
n!=n*(n-1)*(n-2)*…*1

Definición recurrente:
n! = n * (n-1)!
0!=1

Por ejemplo:
5!=5*4! =120
4!=4*3! =24
3!=3*2! =6
2!=2*1! =2
1!=1*0! =1
0!=1
Entonces 5!=5*4*3*2*1=120




```
public double factorial(int n) {
    if (n>1)
        return (n * factorial (n-1));
    else
        return 1;
}
```

Condición de corte de la recursión

Por ejemplo hacemos una llamada desde Main a la función factorial(3)

Devuelve a Main el valor 6

Ejemplo recursión

Implementar un método recursivo que imprima en forma **ascendente** de 1 hasta **x** de uno en uno.

```
public class Recursividad {

  void imprimirRec(int x) {
    if (x>0) {
      imprimirRec(x-1);
      System.out.println(x);
    }
  }


public static void main(String[] ar) {
  Recursividad re=new Recursividad();
  re.imprimirRec(5);
 }

}
```

Por ej si x=5

1
2
3
4
5

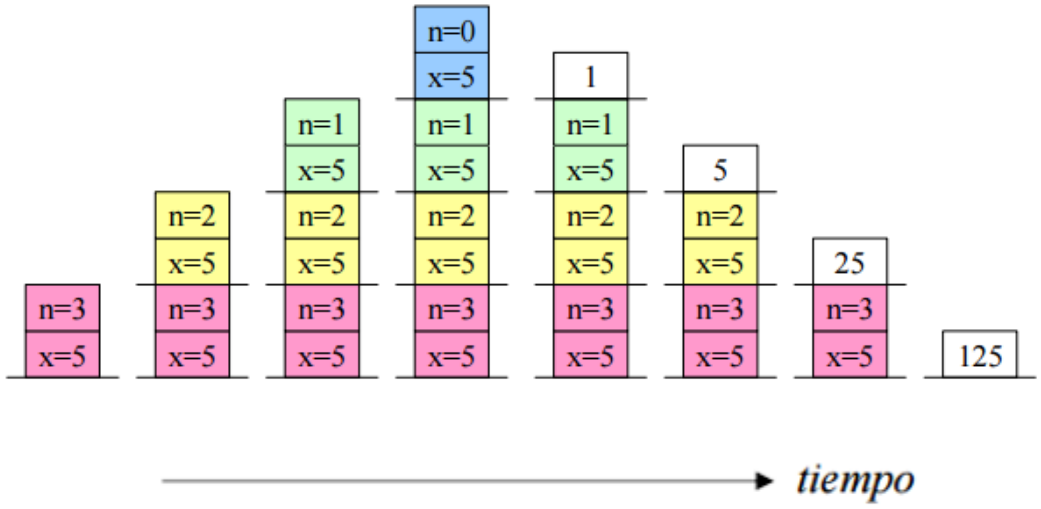
```
public class Recursividad {

 void imprimirRec(int x) {
   if (x>0) {
     System.out.println(x);
     imprimirRec(x-1);
    }
 }

 public static void main(String[] ar) {
   Recursividad re=new Recursividad();
   re.imprimirRec(5);
 }
}
```

5
4
3
2
1

Ejemplo recursión

Qué hace el método calcular ? Por ej. calcular(5, 3)

```
public static int calcular(int x, int n) {

    if (n <= 0) {

        return 1;
    }

    else {

       return x * calcular(x, n - 1);

    }

}
```



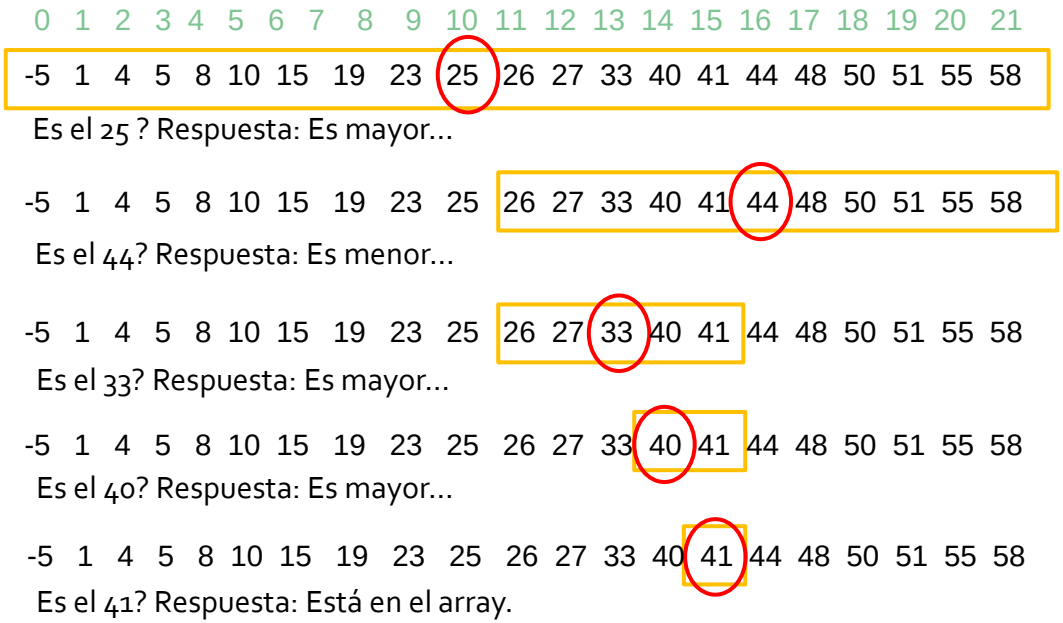
Ejemplo recursión

Supongamos que tenemos un array A de números ordenados.
Y queremos contestar a la pregunta: ¿El número 41 está en el array?

¿Qué nos conviene hacer para minimizar la cantidad de preguntas?

Preguntar por elemento central.
i = floor((iizquierdo + iderecho)/2)

iderecho = A.length -1
iizquierdo = 0
Entonces i = 10



Algoritmo de Búsqueda Binaria

Sirve para buscar eficientemente un elemento dentro de cualquier array de números ordenados.
Hace como mucho $\lceil \log_2 n \rceil$ preguntas (accesos a memoria).
$O(\log_2 n)$

¿La eficiencia sería la misma si en vez de usar un array ordenado usara una lista vinculada ordenada?

NO !!

Ejemplo recursión

Algoritmo de Búsqueda Binaria

```
public int BinariaRecursiva(int [] A,  int X, int inicio, int fin)
{
      int medio;
      if (inicio > fin) return -1; //sucederá si no se encuentra el elemento
      else {
        medio = (inicio + fin) / 2;    //al ser medio un int, se realiza un truncado (pierde la parte decimal)
        if (X > A[medio])
            return BinariaRecursiva(A, X, medio+1, fin);
        else
            if (X < A[medio])
                return BinariaRecursiva(A, X, inicio, medio -1);
            else
                return medio;
       }
}
```

- Donde A es el array de enteros ordenado de menor a mayor.
- X es el número buscado.
- **inicio** se inicializa en 0.
- **fin** se iniciaiza en A.length-1

Algoritmo de Búsqueda Binaria

Algoritmo de Búsqueda Binaria

Sirve para buscar eficientemente un elemento dentro de cualquier array de números ordenados. Hace como mucho $\lceil \log_2 n \rceil$ preguntas. Es $O(\log_2 n)$

IMPACTO EN LA EFICIENCIA:

Tenemos un array ordenado con **1.000.000** de DNI.

Y queremos saber si determinado DNI está en el conjunto.

$O(n)$

- Si hacemos busqueda secuencial => maximo **1.000.000** de comparaciones (accesos a memoria)
- Si hacemos busqueda binaria => máximo cuántas comparaciones ? $O(\log_2 n)$

$\log_2$ 1.000.000 = 20 !!!!

- Y si ahora tenemos guardados **2.000.000** de DNI ?
- Si hacemos busqueda secuencial => maximo **2.000.000** de comparaciones $O(n)$
- Si hacemos busqueda binaria => máximo cuántas comparaciones ? $O(\log_2 n)$

$\log_2$ 2.000.000 = 21 !!!!

Entonces en estos casos usar búsqueda binaria puede ser una MUY BUENA decision de diseño.

...Obviamente tenemos que consumir el tiempo de ordenar el array, pero se hace **una sola vez**. Cómo lo ordenamos y cuánto "cuesta" ?... Clase que viene...