

PROGRAMACION 3

TUDAI

CURSADA 2022

Federico Améndola y Sebastián Vallejos
Facultad de Ciencias Exactas - UNICEN

Iniciar Grabación

RECURSIVIDAD



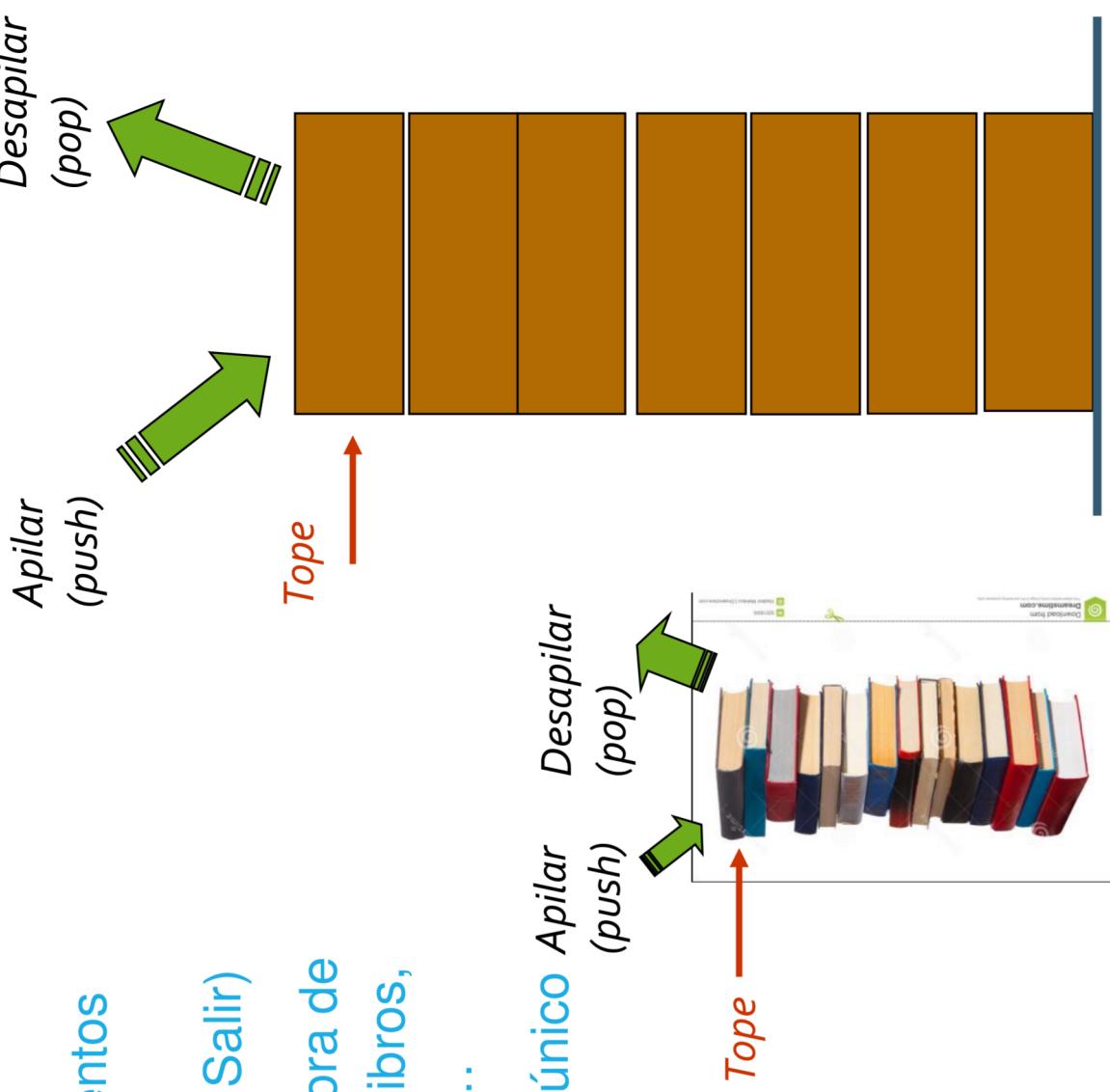
Repasso: Estructura de Datos PILA

- 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 (Último 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

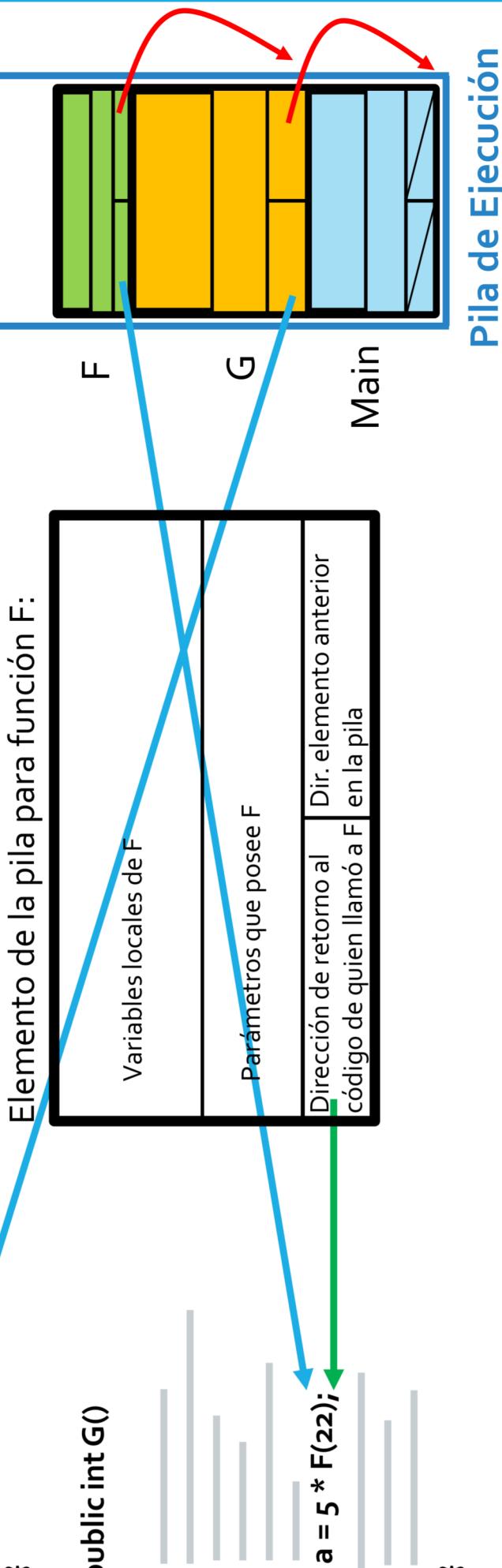
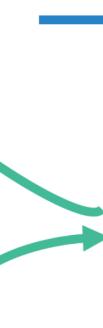
Describiremos un modelo simplificado de pila de ejecución:

El Sistema operativo mantiene una pila por cada aplicación en ejecución.

Por cada llamada a función (o método) se agrega un elemento en la pila.

Dicho elemento contendrá varios datos: los parámetros que se le pasan al método o función, la dirección de retorno (dónde debe continuar ejecutar luego de finalizar), la dirección del elemento anterior en la pila (ya que los elementos no son todos del mismo tamaño), y las variables locales de la función. Esto define el ámbito de la función.

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

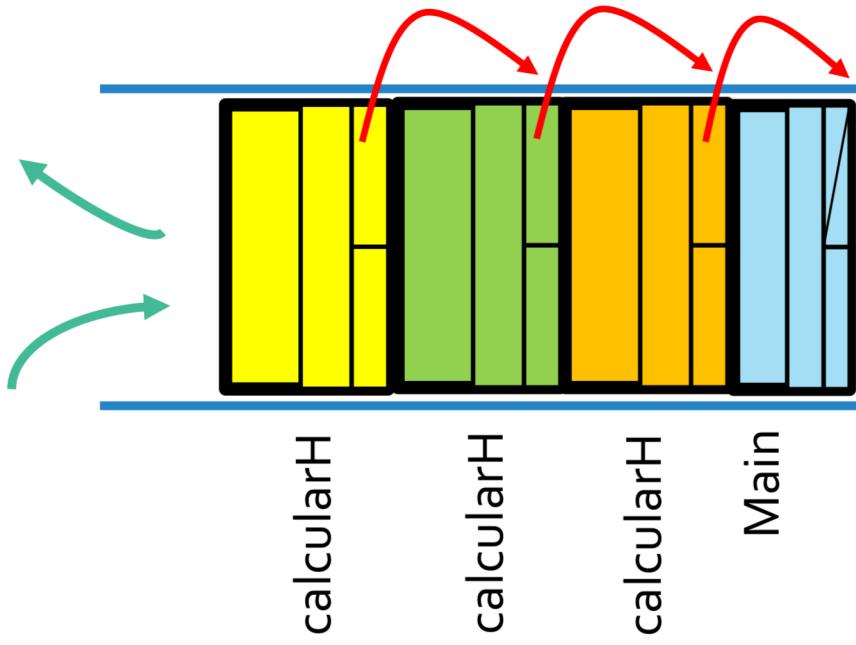


Llamada recursiva

Modelo de pila de ejecución

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

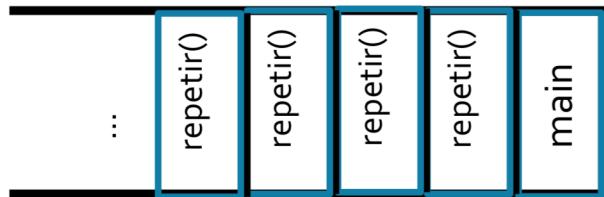
```
public void calcularH(int x)
{
    a = 5 * calcularH(x+22);
}
```



Pila de Ejecución

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

Por ej.

imprimir(5):

```
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);  
    }  
}
```

- 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 caso base o condición de corte de la recursión.

Por lo que 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.

Por ej si **x=5**:

```
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);  
    }  
}
```

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:

$$5!=5*4! = 120$$

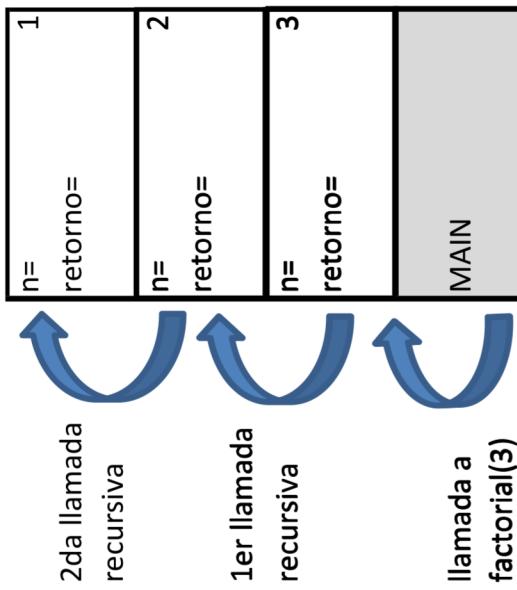
$$4!=4*3! = 24$$

$$3!=3*2! = 6$$

$$2!=2*1! = 2$$

$$1!=1*0! = 1$$

$$\text{Entonces } 5!=5*4*3*2*1=120$$



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

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

Ejemplo recursión – calcular factorial

Dado n un número natural

$$n! = n * (n-1) * (n-2) * \dots * 1$$

Definición recurrente:

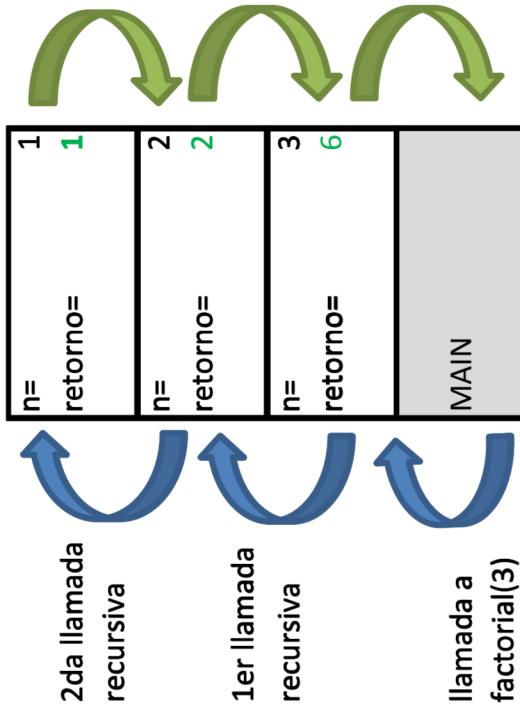
$$n! = n * (n-1)!$$

$$0!=1$$

Por ejemplo:

$$\begin{aligned} 5! &= 5 * 4! &= 120 \\ 4! &= 4 * 3! &= 24 \\ 3! &= 3 * 2! &= 6 \\ 2! &= 2 * 1! &= 2 \\ 1! &= 1 * 0! &= 1 \\ 0! &= 1 \end{aligned}$$

Entonces $5! = 5 * 4 * 3 * 2 * 1 = 120$



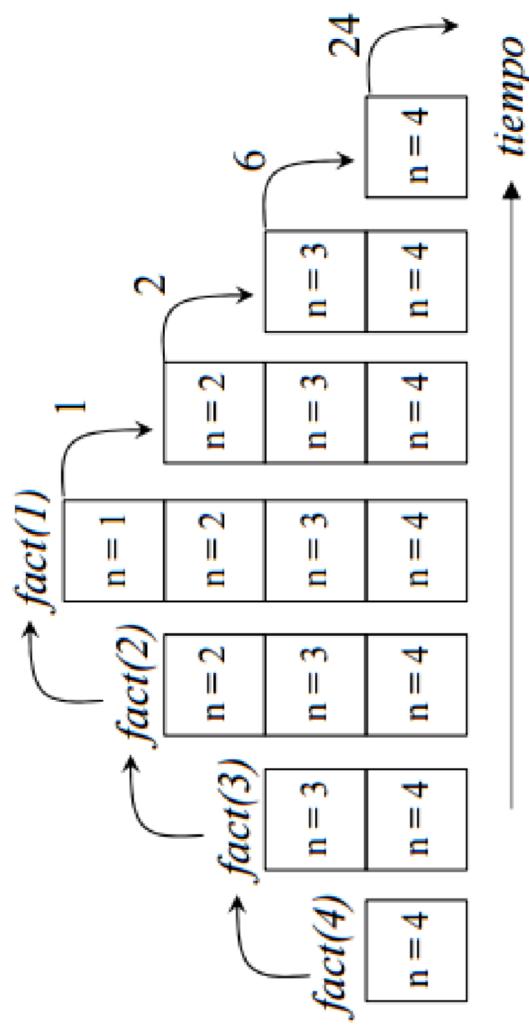
Por ejemplo hacemos una llamada desde Main a la función factorial(3)

Devuelve a Main el valor 6

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

Ejemplo recursión – calcular factorial

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



Ejemplo recursión

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

Por ej si **x=5**

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

1 2
3 4
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
```

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

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);  
  
    }  
  
}
```

n=0	x=5	1
n=1	x=5	5
n=2	x=5	25
n=3	x=5	125

tiempo

Ejemplo recursión

Imaginemos que un amigo desea que adivinemos un número que tiene en un papel anotado y que solo él conoce, antes que empiecemos nos advierte que el número está comprendido del 0 al 20.

Podemos intentar cuantas veces como queramos, y por cada intento que hagamos el nos dirá si el número es mayor, menor o igual al número que tiene anotado en el papel.
Supongamos él piensa en el 14....

¿Qué nos conviene hacer para minimizar la cantidad de preguntas? Poner el rango de nros ordenados.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Preguntar por elemento central.

Es el 10? Respuesta: Es mayor...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Es el 15? Respuesta: Es menor...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Es el 12? Respuesta: Es mayor...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Es el 13? Respuesta: Es mayor...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Es el 14? Respuesta: Adivinaste !!!

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).

¿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 (fin < inicio) return -1; //sucederá si no se encuentra el elemento  
    else {  
        medio = (inicio + fin) / 2; //al ser medio un int, se realiza un truncado  
        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 inicializa en el tamaño del array -1

Ejemplo recursión

Dado el conjunto $\{5, 33, 35, 3, 2, 37, 10, 9, 13, 7, 18, 24, 27\}$

Queremos aplicar Búsqueda Binaria, para ello ponemos el conjunto en un array **ordenado**.

Array A	0	1	2	3	4	5	6	7	8	9	10	11	12
	2	3	5	7	9	10	13	18	24	27	33	35	37

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 $O(\log_2 n)$

Tarea: Hacer seguimiento de la ejecución “¿Existe el 17?”.

IMPACTO EN LA EFICIENCIA:

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

Y queremos saber si determinado DNI está en el conjunto.

- Si hacemos búsqueda secuencial => máximo **1.000.000** de comparaciones $O(n)$
- Si hacemos búsqueda binaria => máximo cuántas comparaciones ? **20 !!!** $O(\log_2 n)$
- Y si ahora tenemos guardados **2.000.000** de DNI ?
- Si hacemos búsqueda secuencial => máximo **2.000.000** de comparaciones $O(n)$
- Si hacemos búsqueda binaria => máximo cuántas comparaciones ? **21 !!!** $O(\log_2 n)$

...Obviamente tenemos que consumir el tiempo de ordenarlo, pero se hace una sola vez.
Cómo lo ordenamos y cuánto “cuesta”? ... Clase que viene...