JAVA

MÓDULO Nro.3

1000 PROGRAMADORES

Ministerio de Economía y Servicios Públicos
Gobierno de Salta

Ministerio de Educación Cultura, Ciencia y Tecnología
Gobierno de Salta

Universidad Nacional de Salta

Temas:

- Entrada y Salida por consola
- Variables de texto. Leer/escribir textos.
- Pilas.
- Colas.
- Listas.

Clase nro. 3

- **Hash Map**
- **Recursión**

MÓDULO 3.

1000 PRO GRA MA DORES

Zurita Natalia

HASHMAP

1000 PROGRAMADORES

Cuando vimos vectores y ArrayList aprendimos que los arrays y vectores son una colección ordenada de elementos en los cuales tenemos acceso a cada elemento que aloja el vector o ArrayList por medio de un índice de tipo entero (indice/elemento).

Hashmap es una estructura de datos que sigue una idea muy parecida a la de ***vectores*** o ***ArrayList*** *(indice/elemento)* pero con la diferencia que ***Hashmap*** almacena dos ítems una ***Clave*** y un ***Valor*** y en este caso podemos acceder a cada valor por medio de la clave asociada.

Zurita Natalia

HASHMAP - ¿COMO CREAR?

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();

    }
}
```


Zurita Natalia

HASHMAP - AÑADIR UN ELEMENTO

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
       hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
    }
}
```


Zurita Natalia

HASHMAP-ACCESO A UN ELEMENTO

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
       hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
       System.out.println(hm.get("2"));
    }
}
```


Zurita Natalia

HASHMAP - ELIMINAR ELEMENTO

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
       hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
       System.out.println(hm.get("2"));
       hm.remove("3");
    }
}
```

MÓDULO 3.

Zurita Natalia

HASHMAP - TAMAÑO

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
       hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
       System.out.println(hm.get("2"));
       hm.remove("3");
       System.out.println(hm.size());
    }
}
```

HASHMAP - ACCESO A CLAVES

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
        hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
        System.out.println(hm.get("2"));
        hm.remove("3");
        System.out.println(hm.size());
        for (String i : hm.keySet()) {
            System.out.println(i);
        }
    }
}
```

MÓDULO 3.

Zurita Natalia

HASHMAP - ACCESO A VALORES

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
        hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
        System.out.println(hm.get("2"));
        hm.remove("3");
        System.out.println(hm.size());
        for (String i : hm.keySet()) {System.out.println(i);}
        for (String i : hm.values()) { System.out.println(i);}
    }
}
```

MÓDULO 3.

Zurita Natalia

HASHMAP - ACCESO A VALORES Y CLAVES

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
        hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
        for (String i : hm.keySet()) {
            System.out.println("Clave: " + i + " Valor: " + hm.get(i));
        }
    }
}
```

MÓDULO 3.

HASHMAP - ELIMINAR TODOS

1000 PROGRAMADORES

```
public class Test {
    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<String,String> hm=new HashMap<String,String>();
        hm.put("1", "Argentina");
            hm.put("3", "Chile");
            hm.put("2", "Bolivia");
           hm.clear();
    }
}
```

MÓDULO 3.

RECURSIÓN - CONCEPTO

1000 PROGRAMADORES

La recursividad es una técnica de programación que permite diseñar algoritmos que dan soluciones elegantes y simples.

Es una técnica por la cual un algoritmo se invoca a sí mismo para resolver una versión más pequeña del problema original para el cual fue diseñado.

Zurita Natalia

RECURSIÓN - DISEÑO

1000 PRO GRA MA DORES

- El diseño de una solución recursiva para un problema P consta fundamentalmente de los siguientes pasos:
- Plantear P de forma que pueda ser descompuesto en k subproblemas (P_1, P_2,..., P_k) de la misma naturaleza que el problema original.
- Determinar el ***Caso base (condición de parada o terminación):*** es una condición que no produce auto invocación.
- ***Llamada recursiva:*** es la invocación del algoritmo a sí mismo.

  Nota: el valor de los parámetros cambiará en cada llamada, volviéndolo un caso más sencillo que el anterior y aproximándose en cada llamado al caso base.

RECURSIÓN - DISEÑO

1000 PRO GRA MA DORES

Para determinar si un algoritmo recursivo está bien diseñado, se puede utilizar el método de las 3 preguntas:

1. ¿Existe una salida no recursiva o caso base, y esta funciona correctamente para ella?
2. ¿Cada llamada recursiva se refiere a un caso más pequeño del problema original?
3. Suponiendo que 1 y 2 se cumplen, ¿ Funciona correctamente en todo el proceso?

Zurita Natalia

RECURSIÓN - EJEMPLO

Supongamos que deseamos sumar los n elementos de una lista

6	40	13	-1	0	9

1. **PROBLEMA P:** sumar los elementos de una lista

P

6	40	13	-1	0	9

Subproblema P_1

6	40	13	-1	0

Subproblema P_2

6	40	13	-1

Zurita Natalia

RECURSIÓN - EJEMPLO

1000 PRO GRA MA DORES

Subproblema P_3

6	40	13

Subproblema P_4

6	40

Subproblema P_5

6

2. **CASO BASE:** n=0 (lista sin elementos).
3. **LLAMADA RECURSIVA:** supongamos que nuestro algoritmo lo vamos a llamar sumarLista este módulo recibirá como parámetros la lista y su longitud. De acuerdo a como se planteó la división en subproblema de P vemos que vamos reduciendo en 1 la lista quitando siempre el último elemento, es decir en cada subproblema lo vamos "perdiendo" pero antes de perderlo deberíamos acumularlo ya que queremos sumar todos los elementos de la lista. Entonces la llamada recursiva queda: ultimoElemento + RestoLista.

Zurita Natalia

RECURSIÓN - CÓDIGO JAVA

1000 PROGRAMADORES

```
public Int sumarLista(int lista, int n){
        If(t==0){
                        return 0;
        }
        else return lista[n-1] + sumarLista(lista,n-1);
}
```

MÓDULO 3.

Zurita Natalia

RECURSIÓN - TRAZA

1000 PRO GRA MA DORES

Para comprender mejor el funcionamiento de un algoritmo recursivo es fundamental conocer cómo funciona la pila de llamadas[1] de un programa en ejecución.

Todo programa en ejecución tiene una pila asociada para este propósito. En los lenguajes de alto nivel (como C o Java) la gestión de la pila de llamadas la realiza de forma automática el compilador, por lo tanto, el programa no necesita preocuparse de su correcto funcionamiento.

Cuando en un punto del programa se llama a una función (recursiva o no) se reserva un espacio en la pila para la siguiente información:

- Dirección de retorno de la función
- Parámetros de la llamada
- Espacio para las variables locales
- Resultado devuelto

1 segmento de memoria basado en una estructura de datos de tipo pila utilizada para almacenar la información relacionadas con las llamadas a funciones dentro de un programa

Zurita Natalia

RECURSIÓN - TRAZA

1000 PRO GRA MA DORES

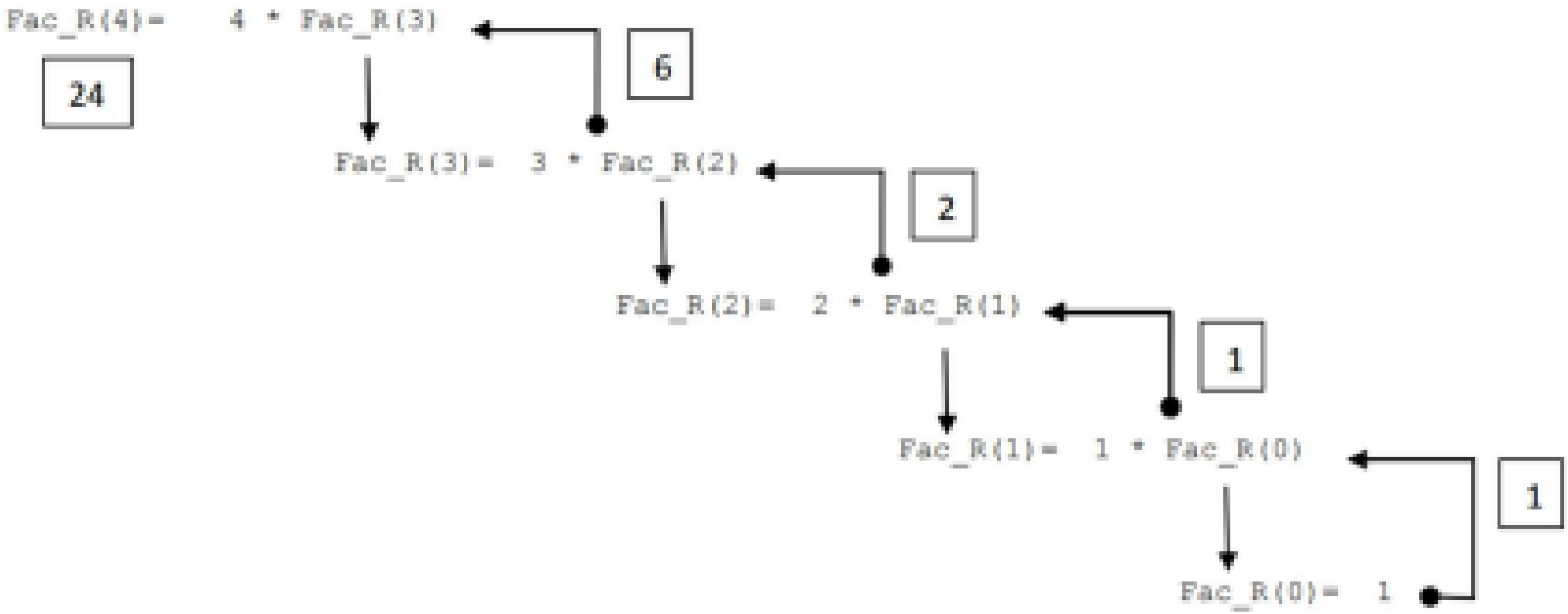
Ahora que sabemos cómo se funciona un programa en ejecución pasemos a un ejemplo de la ejecución de un algoritmo recursivo.
Supongamos un programa que calcula la factorial de un número n. Sabemos que la de matemáticamente el factorial se define como n!=n*(n-1), esta definición traducida a código Java:

```
public static int factorial(n){
        if(n==0)
                return 1;
        else return n*factorial(n-1);
}
```

Si asignamos a n el valor 4

1. Dentro de factorial, cada llamada return n*factorial(n-1); genera una nueva zona de memoria en la pila, siendo n-1 el correspondiente parámetro actual para esta zona de memoria. En cada llamada también queda pendiente la evaluación de la ejecución de la expresión y la ejecución del return.
2. El proceso 1 se repite hasta que la condición del caso base se hace verdadera. Se ejecuta return 1 y empieza la vuelta hacia atrás de la recursión. Se evalúan las expresiones y se ejecutan los return pendientes.

RECURSIÓN - TRAZA



Zurita Natalia

RECURSIÓN

1000 PRO GRA MA DORES

¿Por qué escribir algoritmos recursivos?

- Generalmente son más fáciles de analizar
- Se adaptan mejor a las estructuras de datos cuya naturaleza es recursiva
- Ofrecen soluciones estructuradas, modulares y elegantemente simples

Zurita Natalia

Gracias.

WEB: http://milprogramadores.unsa.edu.ar/

CANAL TELEGRAM: https://t.me/milprogramadoressaltenios

CENTRO DE AYUDA: http://ayudamilprogramadores.com/

1000 PROGRAMADORES