

TEMA 3

1.- Abstracción de operaciones.

- Métodos
- Funciones y Procedimientos
- Paso de parámetros.
- ✓ Por valor
- ✓ Por referencia

2.- La clase String.

3.- La clase Math.

Abstracción de operaciones.

- ✗ En Java, la abstracción de operaciones se consigue a través de los llamados **métodos**.
- ✗ Los métodos implementan operaciones dentro de una clase, de manera que **un método podrá invocar a otros de los que solo le interesa saber qué es lo que hacen** y, en ningún caso **cómo** realizan internamente la operación.

Métodos

- ✗ El repertorio de instrucciones que ofrecen los lenguajes de programación se amplía con la posibilidad de hacer llamadas a métodos definidos por el propio programador.
- ✗ Hasta ahora hemos utilizado ciertos métodos que se definen en librerías del propio lenguaje, por ejemplo **Math** .
- ✗ Se han utilizado los métodos **pow ()** y **sqrt()** de la librería **Math**. Ejemplos de su utilización son:

```
double sx = Math.pow(x, y);
```

```
double rx= Math.sqrt(x);
```

Abstracción de operaciones.

Métodos

- ✖ En todos los casos se observa que la **utilización del método nos oculta los detalles de cómo se realiza internamente la operación** que se requiere.
- ✖ No sabemos como el método **Math.pow(x,y)** realizará la potencia, realmente no nos importa, lo importante es que cuando se invoque al método, éste realice la operación que se indica.

Características comunes en el uso de métodos:

- ✗ Todos los métodos tienen un identificador(un nombre):
sqrt, nextInt, nextLine.
- ✗ Después del identificador, y entre paréntesis, figuran los **parámetros del método.**
- ✗ Los métodos pueden **no tener parámetros**, como el método `nextLine()`;
- ✗ **Tener o no tener** parámetros como

`System.out.println();`

`System.out.println(" Resultado = "+ resul);`

Características comunes referentes al uso de métodos:

- ✗ Algunos métodos devuelven un resultado, por ejemplo **pow(x,y)** devuelve un tipo double, **readLine()** devuelve un String, otros métodos no devuelven ningún resultado explícitamente, como por ejemplo **System.out.print()**.
- ✗ A los métodos que devuelven un resultado explícitamente se les denomina **funciones**, a los otros **procedimientos**.

ventajas:

- ❖ **Ahorra esfuerzo y tiempo** cuando en la resolución de un problema complejo, se repite frecuentemente una misma secuencia de acciones.
- ❖ Facilita la resolución en términos de subproblemas más sencillos.
- ❖ Incrementa la legibilidad de los programas.
- ❖ Un programa estará compuesto de uno o más métodos que, invocándose unos a otros, resolverán el problema planteado.

ventajas:

- ❖ En JAVA, el método que siempre existe es el llamado 'main' o método principal y será el encargado de llevar **el control de ejecución del programa**, es decir, la secuencia de ejecución de las instrucciones y la llamada a los métodos necesarios para resolver el problema.

Ejemplo

- ✗ El problema del cálculo del mayor de dos números enteros cualesquiera .

Una solución podría ser:

```
int a,b;
```

```
int max;
```

```
//Asignación de valores a las variables a y b
```

```
if (a>=b) max=a;
```

```
    else max= b;
```

```
System.out.println("El mayor es "+max);
```

- Supongamos ahora el problema del cálculo del mayor de 4 números enteros cualesquiera (a, b, c y d).
- **Una solución podría ser:**

```
int a,b,c,d;  
int max1, max2, max;  
    //Asignación de valores  
if(a >= b)  max1=a;  
    else  max1=b;  
if(c >= d)  max2=c;  
    else  max2=d;  
  
if(max1 >= max2) max=max1;  
    else max=max2;  
  
System.out.println("El mayor es  
"+max);
```

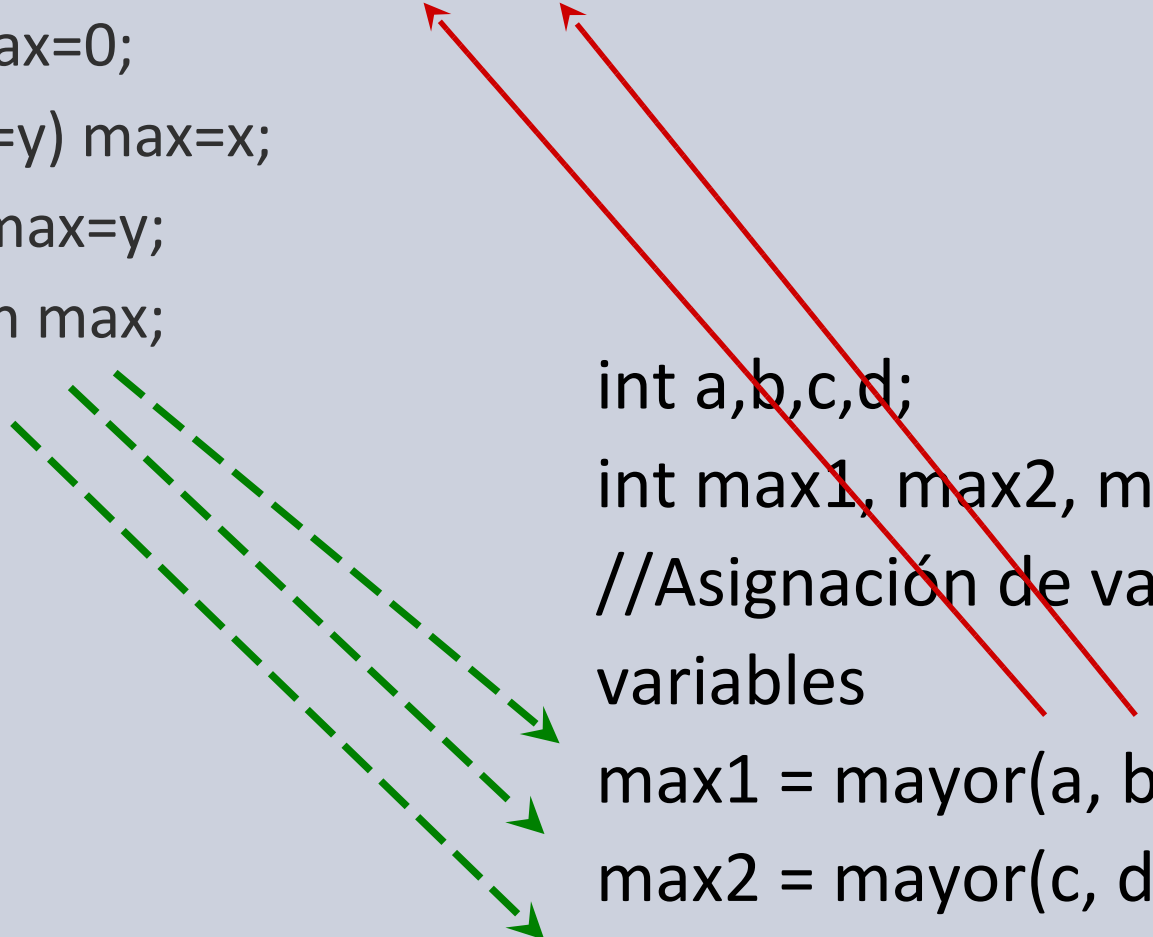
Ejemplo

- La comparación se realiza tres veces, la única diferencia es, qué valores se comparan en cada ocasión (**a con b**, **c con d**, y **max1 con max2**).
- A esos valores podemos llamarles **parámetros de la comparación** y podríamos escribir un método que, de manera genérica, compare 2 números enteros y calcule el mayor de ellos

Ejemplo

```
static int mayor (int x, int y){  
    int max=0;  
    if (x>=y) max=x;  
    else max=y;  
    return max;  
}
```

```
int a,b,c,d;  
int max1, max2, max;  
//Asignación de valores a las  
variables  
max1 = mayor(a, b);  
max2 = mayor(c, d);  
max = mayor(max1, max2);
```



Ejemplo

```
class metodo_1_a {  
    public static void main ( String arg[] ) {  
        int a,b,c,d;  
        int max1, max2, max;  
        //Asignación de valores a las variables  
        max1 = mayor(a, b);  
        max2 = mayor(c, d);  
        max = mayor(max1, max2);  
    }  
    static int mayor (int x, int y){  
        int max=0;  
        if (x>=y) max=x;  
        else max=y;  
        return max;  
    }  
}
```

Parámetros reales

Parámetros formales

Variable local al método

```
class metodo_1_b {  
    public static void main ( String arg[]  
    {  
        int a,b,c,d;  
        int max1, max2, max;  
        //Asignación de valores  
        if(a >= b) max1=a;  
        else max1=b;  
  
        if(c >= d) max2=c;  
        else max2=d;  
  
        if(max1 >= max2)  
            max=max1;  
        else max=max2;  
    }  
}
```

Ejemplo

- ★ Llamamos al método **mayor** tres veces con unos datos (parámetros) diferentes cada vez.
- ★ Desde el punto de llamada, no importa **cómo** el método realiza el cálculo del máximo, sino **qué** es lo que realiza .

Funciones

- ⚙ Una función es un método que define el cálculo de un determinado valor.
- ⚙ Distinguimos entre la **definición de la función** y la **llamada a la función**.

La **definición** consta de :

- Una cabecera,
- Una parte de declaraciones de variables locales.
- El bloque de instrucciones.

El resultado de la función es devuelto mediante la instrucción **return**.

Funciones

static **tipoResultado** **nombreFuncion** (**tipo1 param1, tipo2 param2,..**)

- El modificador **static** indica que la función es accesible como parte de la clase donde se define
- **tipoResultado**. El tipo del valor que devuelve la función.
- **nombreFuncion**. El nombre de la función.
- Entre paréntesis, los nombres de los parámetros y los tipos de los mismos

Funciones

- ⚙ Los parámetros que se usan en la definición de la función(param1, param2, etc.) se denominan **parámetros formales**. Son nombres genéricos que serán sustituidos por los valores que se indiquen en la llamada a la función, a éstos se les denomina **parámetros reales**.
- ⚙ En el momento de la llamada, ambos tipos de parámetros han de coincidir en **número y tipo** (según orden de aparición).

Funciones

- ⚙ El resultado que devuelve la función es el que se indica en la instrucción **return** y ha de ser del tipo especificado en la cabecera **tipoResultado**.

Por ejemplo:

```
static int maximo_2 (int x, int y) {  
    if (x >= y) return x;  
    else return y;  
}
```

parámetros formales



resultado que devuelve



Funciones

Efecto de la llamada a una función

Una llamada a una función se puede realizar, siempre que el contexto sea válido para el resultado que devuelve la función (concordancia de tipos).

Cuando se produce una llamada a una función en un contexto adecuado y con unos **parámetros reales** correctos (en número y tipo), se realizan los siguientes pasos:

Funciones

1. Se crean los objetos locales, parámetros formales y variables locales del bloque principal que define la función. Los nuevos objetos correspondientes a los parámetros formales, toman como **valores** los de los parámetros reales.
- Esta sustitución se denomina **paso de parámetros por valor**.

Funciones

2. Se ejecutan las instrucciones del cuerpo de la función, y la función devuelve su valor.
3. Los objetos locales dejan de existir.
4. La ejecución del programa continúa en la instrucción siguiente a la llamada a la función

Funciones

- ★ **Una función no se ejecuta hasta que no se produce una llamada a la misma.**
- ★ Los objetos locales a la función se crean cuando se produce la llamada, y se destruyen cuando termina la ejecución de la función.

✓ **La siguiente función comprueba si un determinado carácter es una letra minúscula o no:**

```
static boolean minuscula? (char letra) {  
    return ('a'<=letra) && (letra <='z'); }  
}
```

Funciones

- ✓ La siguiente función convierte una letra minúscula en mayúscula:

// x es minúscula

```
static char minusAMayus(char x) {  
    return(char)(((int)x-(int)'a'+(int)'A'));  
}
```

Estas funciones se podrían utilizar como parte de un programa que lea una letra, y si es minúscula convertirla a mayúscula.

```
if (minuscula?(ch))
```

```
System.out.println(minusAMayus(ch));
```

Funciones

- ★ Se desea crear un método que nos permita operar con números enteros como si se tratara de una calculadora básica ('+', '-', '*', '/'). La función quedaría:

```
static double calcula (int a, char op, int b) {  
    // b <> 0 y op es un carácter valido ('+', '-',  
    '*', '/')  
    switch (op) {  
        case '+': return (a+b);  
        case '-': return (a-b);  
        case '*': return (a*b);  
        case '/': return (a/(double)b);  
    }  
}
```


Funciones

Ejemplos de llamada a la función:

```
double result;
```

```
int a, b;
```

```
// Supongamos a y b valores enteros
```

```
result=calcula (a, '+', b);
```

```
result=calcula (result, '-', a);
```

```
result=a*calcula (a, '*', a);
```

```
result=calcula (8, '/', b);
```

Procedimientos

Un procedimiento es un método que define un conjunto de instrucciones que **no devuelven explícitamente un valor**.

Por ejemplo, el conjunto de instrucciones que permite presentar por pantalla, de forma ordenada, tres valores de tipo entero, pasados como parámetros (sin ningún orden).

En la definición de un procedimiento también se distingue entre : cabecera y cuerpo del procedimiento.

Procedimientos

- ❖ La sintaxis de la cabecera de un procedimiento en Java es la de una función que devuelve un valor de tipo vacío,

void

static void nombreProcedimiento (tipo1 param1, tipo2 param2,...)

- ❖ Los parámetros que se usan en la definición de la función (param1, param2) son los parámetros formales que serán sustituidos por los valores de los parámetros reales, en el momento de la llamada a la función.
- ❖ Ambos parámetros (formales y reales) han de coincidir en número, orden y tipo.

Llamada a un procedimiento

- Una llamada a un procedimiento se puede realizar como una instrucción más del lenguaje.
- Cuando el método acaba de ejecutarse, se transfiere el control a la instrucción siguiente a la llamada al procedimiento.

Procedimientos:

Ejemplos

- Un procedimiento para mostrar por pantalla, **de forma ordenada**, 2 valores de tipo entero pasados como parámetros sin ningún orden previo:

```
static void MuestraOrdenados (int A, int B) {  
    int max, min; // variables locales  
  
    if (A>=B) { max=A;  
                min=B; }  
    else { max=B;  
           min=A; }  
    System.out.println(max+" "+min);  
}
```

Posibles llamadas válidas al procedimiento serían:

// x e y variables de tipo entero

MuestraOrdenados (12, 22);

MuestraOrdenados (x, y) ;

MuestraOrdenados (x, 7) ;

parámetros reales



Paso de parámetros por Valor

- ✓ Estos parámetros se consideran variables locales al método y se inicializan con los valores de los parámetros reales, cuando se realiza la llamada al método.
- ✓ Por ser variables locales al método, cualquier modificación sobre los parámetros formales no afectará a los valores de los parámetros reales, cuando termine la llamada.
- ✓ En Java, todos los parámetros de tipo simple (int, double, char, boolean) se pasan por valor.

Paso de parámetros por Valor

- ✓ Cuando se realiza un paso de parámetros por valor, ambos parámetros (formales y reales) son variables diferentes en memoria, la única relación entre ellas es que en el momento de la llamada al método, el valor de los parámetros reales se copia en los parámetros formales y a partir de ese momento son variables totalmente independientes

Paso de parámetros por referencia

- En el paso de parámetros por referencia, lo que se pasa en realidad es la dirección de la variable u objeto, por lo tanto el parámetro formal se convierte en una referencia al parámetro real.
- La llamada al método no provoca la creación de una nueva variable, de forma que las variaciones que el método pueda realizar sobre estos parámetros, afectan directamente a los parámetros reales.

Paso de parámetros por referencia

- En este caso los parámetros formales y reales se pueden considerar como la misma variable con dos nombres diferentes, uno en el método llamante y otro en el método llamado, pero hacen referencia a la misma posición de memoria.
- En java, los parámetros de tipo vector, y en general cualquier objeto, se pasa siempre por referencia

Ámbito de definición de métodos

- ★ La declaración de métodos incluye tanto la descripción de la cabecera como el cuerpo de los mismos.
- ★ Esta declaración se puede realizar bien donde se ha indicado anteriormente o bien antes de la cabecera de la función `main()`.
- ★ No hay restricciones en el orden en el que se escriben los métodos.
- ★ Es recomendable que los identificadores de los métodos estén relacionados con el significado o papel de los mismos

Ámbito de definición de métodos

- ★ Los métodos pueden estar sobrecargados, es decir, puede haber varios métodos con el mismo identificador (nombre). El número y tipo de parámetros de la función servirá para discriminar el método que ha sido llamado en su momento.
- ★ Son objetos locales a un método los parámetros formales y las variables locales que en él se definen.
- ★ Estos sólo son accesibles desde el método en el que se han definido.

Ámbito de definición de métodos

- ★ La comunicación entre el método `main()` y los diferentes métodos se realizará a través de parámetros. No obstante, se podría realizar a través de variables globales, pero esto lo veremos más adelante.
- ★ Un método **estático** definido en una clase se puede utilizar desde cualquier punto de la clase.

En temas posteriores veremos el uso de métodos no estáticos y la posibilidad de acceder a métodos de otras clases

Ejercicios

1. Escribe un programa que calcule el área y la longitud de una circunferencia en función del radio (leído desde teclado). Se ha de escribir un método para calcular el área y otro para la longitud. Las fórmulas del área y la longitud de una circunferencia: $A = \pi * r^2$; $L = 2 * \pi * r$.
2. Escribe un programa en Java que, dado el nombre de una persona y el idioma de preferencia de esa persona, escriba en pantalla un saludo a esa persona en el idioma elegido, con el estilo:

"Buenos días Pepe Sánchez" Los idiomas disponibles serán

(a) Valenciano

(b) Castellano

(c) Inglés

El saludo se mostrará desde un procedimiento al que se le pasan el nombre y el código del idioma. Para el ejemplo anterior, la llamada sería:
saludo ("Pepe Sánchez ", 'b')

La clase String



La clase String

Aspectos básicos

Cualquier grupo de caracteres entre comillas dobles, como: "Esto es un ejemplo de String" es una referencia a un objeto de tipo String con dicho valor.

Son equivalentes las dos inicializaciones siguientes:

```
String st1 = "Esto es un ejemplo de String";
```

```
String st2 = new String("Esto es un ejemplo de  
String ");
```

La clase String

- En el primer caso se asigna a la variable referencia st1 el valor de la referencia a la String,
- En el segundo se construye un nuevo objeto de tipo String cuyo valor es una referencia al texto correspondiente.

En Java se distingue entre caracteres individuales (tipo básico **char**) y objetos de tipo String (tipo referencia).

Las constantes de tipo **carácter** se forman mediante **comillas simples**, mientras que los objetos de tipo **String** lo hacen con comillas dobles.

La clase String

Concatenación y comparación.

La concatenación se efectúa mediante el método

concat(String)

que se puede abreviar mediante los operadores sobrecargados **+** y **+=**.

El resultado es una referencia al objeto de tipo String que se construye inmediatamente.

La clase String

Por ejemplo:

```
String cad1 = "ejemplo1";
```

```
String cad2 = "Ejemplo2";
```

```
String cad3 = cad1+cad2;
```

```
// cad3 vale "ejemplo1Ejemplo2"
```

```
    cad2 += cad1;
```

```
// cad2 vale "Ejemplo2ejemplo1"
```

La clase String

- Para comprobar la **igualdad de dos objetos** se utiliza el método **equals(String)** y, de forma más general, se puede utilizar el método **compareTo(String)** para determinar, según devuelva un número negativo, nulo o positivo si el objeto es menor, igual o mayor al objeto argumento.

Algunos métodos

La clase String define un gran número de métodos para operar sobre objetos pertenecientes a la clase.

Retorno	Método	Función que realiza
String	toUpperCase();	Convierte a mayúsculas
String	toLowerCase();	Convierte a minúsculas
int	length();	Longitud de la cadena
char	charAt(int);	Obtiene el carácter que ocupa la posición indicada por el entero
String	substring(int, int)	Extrae una subcadena desde las posiciones que indican los parámetros.
String	concat(String);	Enlaza o une una cadena a otra.
boolean	startsWith(String)	Si la cadena comienza por la String que se indica , o no.
int	indexOf(String)	Posición donde se encuentra una cadena.

La clase String

Ejemplos de los métodos más significativos:

```
String sT1 = "Ejemplo 1";
```

```
String mayus = sT1.toUpperCase();
```

```
// mayus vale "EJEMPLO 1"
```

```
String minus = sT1.toLowerCase();
```

```
// minus vale "ejemplo 1"
```

```
int longitud = sT1.length();
```

```
// longitud = 9
```

```
char character = sT1.charAt(1);
```

```
// character vale 'j'
```

```
String sub = sT1.substring(3,5);
```

```
// sub vale "mpl"
```

```
String sT= sT1.concat(" y 2");
```

```
// sT vale "Ejemplo 1 y 2"
```

La clase String

```
boolean b = sT1.startsWith("Eje");  
//true si comienza por "Eje"  
int inicio = sT1.indexOf("mpl");  
//inicio=3, la pos. de la secuencia      "mpl" es la 3  
int inicio = sT1.indexOf("y", 5);  
// busca la cadena y a partir de la posición 5
```

La clase String

Además de estos métodos que operan sobre objetos de tipo String, existe un método importante que se encuentra definido para todos los valores de tipo primitivo y también para la mayoría de las clases predefinidas, se trata del método:

toString(), con el que se convierte un valor primitivo u objeto a una String que lo representa.

Por ejemplo:

toString(1618) devuelve la String "1618".

La clase String

- Una variación interesante del método, es que se permite especificar la base de representación cuando se trata de un número entero, así:

`toString(1618, 2)`, devuelve la String representación binaria del número.

La clase String

- Existen métodos inversos que permiten pasar de un número escrito como una String al valor numérico correspondiente, los más habituales relativos a valores int y double, son los siguientes:

```
int num1 = Integer.parseInt("1618");
```

```
int num2 = new Integer("1618").intValue();
```

```
double d = new Double("1618").doubleValue();
```

•Ejercicio 1.-

Escribe un programa en java que pida una frase al usuario y visualice en pantalla la misma frase pero con todos los caracteres en mayúsculas.

```
class cade1 {  
    public static void main(String [] arg) {  
        String frase;  
        BufferedReader ent=new BufferedReader(new  
InputStreamReader(System.in);  
        System.out.println("Dame una frase ");  
        frase= ent.readLine();  
        frase=frase.toUpperCase();  
        System.out.println("En mayúscula : "+ frase);  
    }  
}
```

La clase String

```
class DameTuEmail{
    public static void main(String args[]) {
        Scanner teclado= new Scanner(System.in);
        String correoE;
        do{
            System.out.println("Introduce tu email :");
            correoE=teclado. nextLine();
            if(correoE.length()<3 || correoE.indexOf('@') < 0)
                System.out.println("Dirección no valida");
        }while(correoE.length() < 3 || correoE.indexOf('@') < 0);
        System.out.println("Tu email es: "+correoE);
    }
}
```

La clase String

1. Hacer un programa en java que pida una frase y una palabra al usuario y le diga cuántas veces aparece esa palabra en la frase.
2. Codificar un programa en java que pida una frase y un entero al usuario y escriba por pantalla el carácter que hay en esa posición en la frase, o un mensaje de error si la frase es más corta que la posición que nos pide el usuario.
3. Escribir un programa en java que pida una frase al usuario y le diga cuántas palabras hay en esa frase (consideraremos que entre cada dos palabras únicamente hay un espacio).