TEMA 3 Abstracción de operaciones. Métodos

3.1.- Abstracción de operaciones.
3.2.- Métodos
- Declaración, definición.

3.3.- Funciones
- Declaración y llamada

3.4.-Procedimientos
- Declaración y llamada

3.5.-Paso de parámetros.
- Por valor
- Por referencia

3.6.- La clase String
- Operaciones con cadenas.
- Métodos de la clase String.

3.7.- La clase Math.
- Métodos.

3.1.- Abstracción de operaciones.

El proceso de abstracción consiste en hacer caso omiso de ciertas propiedades o circunstancias poco relevantes y enfatizar en lo que realmente interesa de una situación o enunciado.

En nuestro contexto, la abstracción de operaciones significa hacer caso omiso de **cómo** 'se realizan ciertas operaciones enfatizando el **qué** es lo que hacen.

En el lenguaje Java, la abstracción de operaciones se consigue a través de los llamados **métodos** (que implementan operaciones dentro de una clase), de manera que unos métodos podrán invocar a otros de los que les interesa saber únicamente qué es lo que hacen y, en ningún caso, cómo realizan internamente la operación.

De esta manera, el repertorio de instrucciones que ofrecen los lenguajes de programación (y Java en concreto) se amplía con la posibilidad de hacer llamadas a métodos definidos por el propio programador o por otros.

Hasta ahora hemos utilizado ciertos métodos que se definen en librerías del propio lenguaje, como por ejemplo Math o BufferedReader en concreto, se han utilizado los métodos pow y sqrt de la librería Math, y los métodos readLine().

```
double sx=Math.sin(x);              // x es una variable de tipo double
double rx=Math.sqrt(x);             // x es una variable de tipo double y positiva
BufferedReader teclado = new BufferedReader (new InputStreamReader(System.in));
String texto=teclado.readLine();
```

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; por ejemplo, no sabemos cómo la llamada al 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. La posibilidad de definir métodos es una herramienta de gran importancia en la programación, ya que, permite la abstracción modular de las operaciones y la descomposición de problemas.

Si se observan los ejemplos descritos, se pueden extraer algunas características comunes referentes al uso de métodos:

- Todos los métodos tienen un identificador: **sqrt**(), **readLine**().
- Después del identificador, y entre paréntesis, figuran los parámetros del método. Los métodos también pueden no tener parámetros, como el método readLine(), y también puede ocurrir que algunos métodos puedan tener o no parámetros, como es el caso del System.out.println().
- Algunos métodos devuelven un resultado, por ejemplo pow o sqrt devuelven un tipo double, readLine() devuelve un resultado de tipo String, otros métodos no devuelven ningún resultado explícitamente.

A los métodos que devuelven un resultado explícitamente se les denominan **funciones**, a los otros **procedimientos**.

El programador también puede definir funciones o procedimientos propios. De hecho, la utilización de procedimientos y funciones en el desarrollo de programas es la base del diseño por refinamientos sucesivos y presenta las siguientes ventajas:

- Ahorra esfuerzo y tiempo cuando en la resolución de un problema se repite frecuentemente una misma secuencia de acciones de problemas complejos describiendo su solución.
- Facilita la resolución en términos de subproblemas más sencillos.
- Incrementa la legibilidad de los programas.

De esta manera, un programa estará compuesto de uno o más métodos que, invocándose unos a otros, resolverán el problema planteado. 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 ejecución de los demás métodos necesarios para resolver el problema.

Un programa siempre empieza a ejecutarse en el método 'main' o principal que, en el momento adecuado, ira invocando a otros métodos que, a su vez, pueden invocar a otros (incluso un método puede invocarse a sí mismo, es la denominada recursividad).

A esta forma de diseño descendente de algoritmos o programas se le llama 'Diseño Top Down' o por 'refinamientos sucesivos'. Los pasos a seguir para emplear esta metodología son:

1. Se considera el problema como un 'todo'.
2. Se divide el problema en subproblemas tales que, siendo de menor complejidad, forman parte del problema completo.
3. Se vuelve a aplicar el paso 2 a los subproblemas obtenidos hasta llegar a problemas de fácil solución o triviales.
4. Se solucionan los subproblemas 'triviales' y se agregan estas soluciones para obtener la solución al problema inicial.

De esta forma, el programador se concentra en solucionar problemas de menor complejidad, tales que, la solución combinada de todos ellos, es la solución al problema completo. En Java, la solución a cada uno de estos problemas, se corresponde con un método.

Ejemplo : El problema del cálculo del máximo de dos números enteros cualesquiera a y b. Este sencillo algoritmo resuelve el problema mencionado.

```
İnt a,b;
int max;
//Asignación de valores a las variables 'a' y 'b'
if(a>=b) max=a;
        else  max=b;
```

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

```
Public static void main(String argm[]){
int a,b,c,d;
int max1, max2, max;
//Asignación de valores a las variables
   if(a>=b) max1=a;
        else   max1=b;
   if(c>=d) max2=c;
        else  max2=d;
   if(max1>=max2) max=max1;
         else  max=max2;
```

Se puede observar que la comparación que se realiza tres veces es exactamente la misma, 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 llamarlos parámetros de la comparación y podríamos escribir un método que, de manera genérica, compare 2 números enteros y calcule el máximo de ellos. En JAVA quedaría:

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

Si la solución al problema del cálculo del máximo de 4 números enteros cualesquiera se rescribe haciendo uso de esta definición del método **maximo**:

```
public static void main(String argm[]){
int a,b,c,d;
int max1, max2, max;
//Asignar valores a las variables
max1=maximo(a, b);
max2=maximo(c, d);
max=maximo(max1,max2);
System.out.println( "El mayor es : "+max);
}
```

Se puede observar que se invoca al método 'maximo' 3 veces con unos 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.

3.3.- 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 y el bloque de instrucciones que indican el cálculo a realizar. El resultado de la función es devuelto mediante la instrucción **return.**

La sintaxis de la cabecera de una función en Java es:

> **static tipoResultado nombreFuncion (tipo1 param1, tipo2 param2,..)**
>
> La cabecera de la función: contiene la siguiente información:
>
> - El tipo del valor que devuelve la función.
> - El nombre de la función.
> - Entre paréntesis, los tipos de los parámetros y los nombres de los mismos.

El modificador **static** indica que la función es **accesible** como parte de la clase donde se define.

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).

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**.

En una función sólo se ejecutará una instrucción 'return', aunque en el cuerpo de la misma aparezcan varias.

Por ejemplo:

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

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). Por ejemplo, si una función devuelve un resultado de tipo int, puede ser llamada formando parte de una expresión aritmética, y en general en la parte derecha de una instrucción de asignación sobre una variable de tipo int.

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:

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 llama **paso de parámetros por valor**.
2. Se ejecutan las instrucciones del cuerpo de la función y devuelve un valor con **return.**
3. Los objetos locales dejan de existir.
4. La ejecución del programa continuará en la instrucción siguiente a la llamada a la función.

Una función no se ejecuta hasta que no se produce una *llamada a la misma*. De la misma manera, los objetos locales a la función se crean cuando se produce la llamada y se destruyen cuando termina la ejecución. de la función. Ejemplos :

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

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

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 para hacer que una letra leída minúscula se convierta a mayúscula :

```
//Supongamos que se ha leído un carácter sobre la variable ch
    if ( minuscula(ch) )  System.out.println( minusAMayus(ch) );
```

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

```
static int 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/b);
  }
}
```

Ejemplos de llamadas serían:

```
int result, a, b;
// Supongamos a y b valores de tipo entero

result=calcula(a,'+',b);
result=calcula(result, '- ',a);
result=a*calcula(a, '*',a);
result=calcula(8, '/', b);
```

3.4.- 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 la cabecera y el cuerpo del procedimiento. La cabecera de un procedimiento contiene la siguiente información:

- El nombre del procedimiento.
- Entre paréntesis los nombres de los parámetros y los tipos de los mismos.

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 (param1, param2, etc.) del procedimiento son parámetros formales, que serán sustituidos por los valores de los parámetros reales en el momento de la llamada. Ambos parámetros (formales y reales) han de coincidir en número orden de aparición y tipo, exactamente igual que en las funciones.

Llamada a un procedimiento

Una llamada a un procedimiento se puede realizar como una instrucción más del lenguaje. Cuando se produce la llamada se realizan los siguientes pasos:

1. Se crean los objetos locales del procedimiento (parámetros formales y variables locales). Los nuevos objetos correspondientes a los parámetros formales toman como valores los de los parámetros reales (paso de parámetros por valor).
2. Se ejecutan las instrucciones del cuerpo del procedimiento.
3. Los objetos locales dejan de existir.
4. La ejecución del programa continúa en la instrucción inmediatamente siguiente a la llamada al procedimiento.

Ejemplo. Un procedimiento para mostrar por pantalla, de forma ordenada, 2 valores de tipo entero pasados como parámetros, sin ningún orden.

```
static void MuestraOrdenados (int a, int b) {
int max, min;
if (a>=b) { max=a;
            min=b; }
   else { max=b;
          min=a; }
System.out.println ("EL máximo es: "+max);
System.out.println ("El mínimo es: "+min);
}
```

Posibles llamadas válidas al procedimiento serían:

```
// Siendo x e y variables de tipo entero
MuestraOrdenados(12, 22);
MuestraOrdenados(x, y);
MuestraOrdenados(x, 7)
```

3.5.- Paso de parámetros por valor y por referencia.

Hasta el momento sólo hemos hablado del 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 termina la llamada.

En Java, todos los parámetros de tipo simple (int, double, char, booleam , se pasan por valor. Cuando se realiza un paso de parámetros por valor, ambas 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.

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.

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 refe rencia.

Ámbito de definición de métodos

La estructura general de un programa en Java que define y usa sus propios métodos sería la siguiente:

```
class ... {
public static void main (...) {
...
...
}

//Declaración 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.*** *N*o 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.

Los métodos pueden estar sobrecargados; esto es, puede haber varios métodos con el mismo identificador. 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.

La comunicación entre la función main y los diferentes métodos , se realizará a través de los parámetros. También se podrá establecer comunicación, mediante variables globales como 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.

Problemas

1. Escribe una función que dados 4 números enteros pasados como parámetros, compruebe si dicha secuencia de números es capicúa.

4. Escribe un programa en Java que, dado el nombre de una persona y el idioma de preferencia, escriba en pantalla un saludo 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’)

3.6.- 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 una 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), y las constantes de tipo carácter están formadas mediante comillas simples, mientras que los objetos de tipo String lo están por comillas dobles.

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. Así, por ejemplo:

```
String exp1 = "ejemplo1";
String exp2 = "Ejemplo2";
String exp3 = exp1+exp2;          // exp3 vale  “ejemplo1Ejemplo2"
String exp2 += exp1;              // exp2 vale "Ejemplo2ejemplo1“
```

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 de la clase String.

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.

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.
boolean	equals(String)	Si la cadena es igual a la String que , negativo svse indica , o no.
int	compareTo(String)	Compara la cadena con el String que se indica , devolviendo 0 si es igual, negativo si esmenor y positivo si es mayor
int	compareToIgnoreCase(String)	Igual que el anterior, sin diferenciar entre mayúsculas y minúsculas

A continuación se indican, mediante ejemplos, los 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 caracter = sT1.charAt(1);             //   caracter  vale 'j'
String sub = sT1.substring(3,5);           //   sub vale "mpl"
String sT= sT1.concat(" y 2");             //   sT vale "Ejemplo 1 y 2"
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);          // inicio=10,
```

Existe otro 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.
Así por ejemplo, toString(1618) devuelve la String "1618".

- 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.
- Existen métodos inversos que permiten pasar de un número escrito como una String al valor 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();
```

Ejemplo 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[] args) {
 Scanner ent = new Scanner(System.in);
 String frase;

 System.out.println("Dame una frase ");
 frase= ent.nextLine();
 frase=frase.toUpperCase();
 System.out.println("En mayúscula :"+
frase);
  }
}
```

```
public class DameTuEmail{
 public static void main(String args[])  throws IOException {
  String correoE;
  Scanner teclado = new iScanner(System.in);
   do{
      System.out.println("Introduce tu email");
       correoE=teclado.readLine();
       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);
 }
}
```

3.7.- La clase Math :Métodos.

Desde la versión J2SE 5, se incluye la importación de la clase de forma estática, lo que permite acceder a sus métodos sin tener que incluir el nombre de la clase.

Ejemplo.

```
import java.lang.Math.*;
System.out.println("Valor absoluto "+ abs(N);
```

Métodos de la clase Math	Descripción
Math.abs(x)	Valor absoluto de x
Math.sqrt(x)	Raíz cuadrada de x
Math.ceil(double x)	El número completo más pequeño mayor o igual a x
Math.floor(double x)	El número completo más grande menor o igual a x
Math.rint(double x)	Valor double truncado de x
Math.pow(double x, double y)	X elevado a y
Math.round(x)	Para double y float
Math.randon(x)	Double entre 0.0 y 1.0
Math.max(x , y)	Para int, long, float, double
Math.min(x, y)	Para int, long, float, double
Math.PI	Para PI aproximadamente 3.14
Math.sin(double x)	Seno de x en radianes
th.cosl(double x)	Coseno de x en radianes
Math.tan(double x)	Tangente de x en radianes
Math.atan2(double x, double y)	El ángulo cuya tangente es a/b

```
package trim1;
class Mates {
 public static void main(String[] args) {
 double x= 17.83, y=2   ;
 System.out.println(Math.ceil(x));                    // Visualiza 18.0
 System.out.println(Math.floor(x));                   // Visualiza 17.0
 System.out.println(Math.ceil(x));                    // Visualiza 18.0
 System.out.println(Math.rint(x));                    // Visualiza 18.0
 System.out.println(Math.round(x));                   // Visualiza 18
 System.out.println(Math.pow(4, y));                  // Visualiza 16.0
 System.out.println(Math.random());                   // Visualiza 0.3682705870072899
 System.out.println(Math.round(Math.random()));       //  Visualiza 0
 System.out.println((Math.PI));                       // Visualiza 3.141592653589793  }
}
```

Ejercicios de 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. Por ejemplo, si la frase es “el programa en java pide el dato al usuario” y la palabra es “el”, el programa me dirá “la palabra el aparece 2 veces 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 meten un espacio).
4. Escribir un programa en java que pida una frase al usuario y nos diga cuantas vocales tiene la frase. (Vocales son todas, las minúsculas y las mayúsculas)
5. Escribir un programa en java que pida una frase al usuario y la visualice por palabras en una columna, indicando cuantas letras tiene cada palabra.

Hola como estas	Hola 4
	como 4
	estas 5

6. Escribir un programa en java que pida el nombre de una persona (que puede ser compuesto) y sus dos apellidos, y nos saque por pantalla las iniciales (de su/s nombre/s) junto con las dos iniciales de sus dos apellidos.
7. Escribir un programa en java que pida una frase al usuario y la visualice después de eliminar todos los espacios en blanco.
8. Escribir un programa en java que pida una palabra, y guarde en otra String la palabra al revés.