

# UT3 Estructuras de Control

**1. Estructuras de selección.**

**1.1 Estructura If.**

**1.2 Switch.**

**1.3 Operador Condicional.**

**2. Estructuras de repetición.**

**2.1 Bucle For.**

**2.2 Bucle While.**

**2.3 Bucle Do-While**

## 1.- Estructuras de selección.

Las estructuras de selección son aquellas que se utilizan cuando se quiere ejecutar una instrucción o bloque de instrucciones si se cumple una determinada condición, de ahí que también se conozcan con el nombre de condicionales.

### 1.1 Estructura If.

Permite ejecutar una instrucción (o secuencia de instrucciones) si se da una condición. La sentencia **if** es la sentencia básica de selección y existen tres variantes: simple, doble y selección múltiple, pero las 3 se basan en la misma idea.

a) **Selección Simple**. Solo tiene la parte positiva de la selección. Su sintaxis es:

|  |
| --- |
| **if (*condición*){**  ***sentencias;***  **}** |

Donde **condición** es una expresión booleana y **sentencias** representa una sentencia o bloque de sentencias, las cuales se ejecutarán si el valor de la condición es True (cierta).

Si es una **sentencia única**, se pueden quitar las llaves

**Ej:**

if (numero%2 != 0) {

System.out.println(“El numero es impar “);

cont++;

}

b) **Selección Doble.** Se añade una parte else a la sentencia if la cual se ejecutará si la condición es falsa. Su sintaxis es:

|  |
| --- |
| ***if (condición){***  ***sentencias1;***  ***}***  ***else{***  ***sentencias2;***  ***}*** |

**Ej:**

if (a>b)

System.out.println (“El número mayor es “ + a);

else

System.out.println(“El número mayor es “ + b+ “ o son iguales”);

c) **Selección Múltiple**. Es habitual cuando hay más de una condición. Su sintaxis es:

|  |
| --- |
| ***if (condición1){***  ***sentencias1;***  ***}***  ***else if (condición2){***  ***sentencias2;***  ***}***  ***else if (condición3){***  ***sentencias3;***  ***else {***  ***sentencias;***  ***}*** |

**Ej:**

if (mes == 12 || mes == 1 || mes == 2)

System.out.println ( “Invierno”);

else if (mes == 3 || mes == 4 || mes == 5)

System.out.println( “Primavera”);

else if (mes == 6 || mes == 7 || mes == 8)

System.out.println (“Verano”);

else

System.out.println (“Otoño”);

### 1.2 Switch.

Es una estructura de selección múltiple más cómoda de leer y utilizar que el else-if. Selecciona un bloque de sentencias dependiendo del valor de una expresión.

Su sintaxis es:

|  |
| --- |
| ***switch (expresion) {***  ***case valor1:sentencias;***  ***break;***  ***case valor2:sentencias;***  ***break;***  ***case valor3:sentencias;***  ***break;***  ***...***  ***default: sentencias;***  ***break;***  ***}*** |

Donde ***expresión*** tiene que tomar un valor entero o un carácter. A partir de la versión 7 de Java también se permite que la expresión sea un String (cadena), ***break*** indica que ha acabado la ejecución de ese caso y seguiría ejecutando la sentencia siguiente al switch y ***default*** es opcional y se ejecutará cuando la expresión tome un valor que no esté recogido en los distintos casos especificados.

**Ej:**

switch (mes) {

case 4:

case 6:

case 9:

case 11: dias = 30;

break;

case 2: dias = 28;

break;

default: dias = 31;

break;

}

Es posible juntar distintos casos, dejándolos en blanco y especificando las instrucciones en el último de los casos de grupo.

En el ejemplo, si la variable mes toma los valores 4, 6, 9 u 11 se hace en los 4 casos lo mismo, se le asigna a la variable día el valor 30.

### 1.3 Operador Condicional.

El operador condicional es un operador ternario, es decir, consta de tres operandos y su función es asignar un valor entre dos posibles a una variable si se cumple o no una condición.

Su sintaxis es la siguiente:

|  |
| --- |
| ***tipo variable = (condicion) ? v\_cond\_true: v\_cond\_false;*** |

Realmente es un if…else simple que podemos utilizar si solo queremos asignar un valor a una variable si se cumple o no una condición.

Ejemplo:

**public** **class** EjemploOperadorCondicional

{

**public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I'm Feeling Lucky)[] args)

   {

**int** edad = 18;

*//Utilizando el operador condicional*

[String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I'm Feeling Lucky) resultado = (edad >=18) ? "Mayor de edad." : "Menor de edad.";

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I'm Feeling Lucky).out.println(resultado);

*//Utilizando if...else*

**if**(edad >= 18)

       {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I'm Feeling Lucky).out.println("Mayor de edad.");

       }

**else**

       {

[System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I'm Feeling Lucky).out.println("Menor de edad.");

       }

   }

}

## 2.- Estructuras de repetición.

Estas estructuras se utilizan para repetir un bloque de sentencias un número de veces. Son también llamadas sentencias de iteración o bucles.

Los tipos de bucles que hay son: ***for***, cuando se sabe el número de veces que se va realizar y ***while o do-while***, cuando no se conoce de antemano el número de veces que se repetirá el bloque de sentencias.

### 2.1 Bucle For (PARA)

La sintaxis de la sentencia es:

|  |
| --- |
| ***for(inicialización;condición;incremento/decremento){***  ***sentencias;***  ***}*** |

Dónde:

* **Inicialización** se realiza solo una vez, **antes de la primera iteración**.
* **Condición** se comprueba **cada vez, antes de entrar** al bucle y si es cierta entra, si no lo es se termina la ejecución de la sentencia y pasa a ejecutarse la siguiente sentencia al for.
* **Incremento/decremento** se realiza siempre **después de terminar de ejecutar las sentencias del cuerpo de la iteración** y antes de volver a comprobar la condición de nuevo.

**Ej**:

int i;

for (i=0; i<5; i++)

System.out.println (i);

El funcionamiento de esta instrucción seria el siguiente:

**1º.** Se inicializa la i.

En el ejemplo i toma el valor 0.

**2º.** Se comprueba si cumple la condición, si se cumple entra en el bucle pero si no la cumple no entraría.

En el ejemplo sí la cumple pues 0<5.

**3º.** Se ejecuta el bloque de sentencias que pertenezcan al for.

En el ejemplo escribirá en pantalla un 0

**4º.** Vuelve al principio del bucle, incrementa la i.

En el ejemplo se incrementa en 1, con lo cual pasa a tomar valor 1.

**5º.** Se repite el proceso desde el paso 2, hasta que deje de cumplirse la condición.

En el ejemplo se repetirá hasta que i tome el valor de 5 pues no se cumple que 5<5 Los siguientes dos ejemplos son un bucle infinito y un bucle que nunca se llega a ejecutar.

**Bucle infinito**: for (i=1;i<5;i--) pues nunca se dejará de cumplir que i<5.

**Bucle que no ejecuta nunca sus sentencias**: for (i=5; i<3;i++) pues desde el principio no se cumple la condición pues 5 no es menor que 3.

### 2.2 Bucle While (MIENTRAS)

La sintaxis de la sentencia es:

|  |
| --- |
| ***while (condición){***  ***sentencias;***  ***}*** |

Donde **condición** es una expresión booleana que se evalúa al principio del bucle y antes de cada iteración de las sentencias.

Si la ***condición es verdadera***, se ejecuta el bloque de sentencias, y se vuelve al principio del bucle.

Si la ***condición es falsa***, no se ejecuta el bloque de sentencias, y se continúa con las siguientes sentencias del programa.

Si la condición es falsa desde un principio, entonces el bucle nunca se ejecuta.

Si la condición nunca llega a ser falsa, se tiene un bucle infinito.

**Ej:**

int i=0;

while (i<5){

System.out.println (i);

i++;

}

El resultado de ejecutar este bloque de instrucciones es el mismo que para el for, escribirá en pantalla del 0 al 4 en diferentes líneas.

Los dos siguientes ejemplos son también un bucle infinito y un bucle que no se ejecutará nunca.

**Bucle infinito:**

int i=0;

while (i<5){

System.out.println(i);

}

Sera infinito pues al no cambiar el valor de la i nunca, la condición siempre se está cumpliendo.

**Bucle que nunca se ejecutará:**

int i=6;

while (i<5){

System.out.println(i);

i++;

}

No se ejecuta nunca el bucle pues desde el principio no se cumple la condición que permita entrar en el while, pues 6 no es menor que 5.

### 2.3 Bucle Do-While (REPETIR HASTA QUE)

La sintaxis de la sentencia es:

|  |
| --- |
| *do{*  *sentencias;*  *}while (condición);* |

Es muy parecida a while. El bloque de sentencias se repite mientras se cumpla la condición pero en este caso, la condición se comprueba después de ejecutar el bloque de sentencias por lo que el bloque **se ejecuta siempre al menos una vez**.

*Este tipo de bucle será muy útil cuando se quiere obligar a que una determinada variable solo pueda tomar unos ciertos valores, o cuando se quiere comprobar una contraseña antes de seguir ejecutando el bucle, etc…*

**Ej**:

int i;

do {

System.out.println (“Introducir un numero distinto de 0”);

i = teclado.nextInt();

} while(i == 0);

En este ejemplo se pide introducir un número que no sea 0, con lo cual no se saldrá del bucle mientras se introduzca un 0. (Probarlo en clase).

# UT3 Java printf para dar formato a los datos de salida

Vamos a ver cómo utilizar printf para dar formato a los datos que se imprimen por pantalla en Java. Este problema se nos plantea por ejemplo cuando queremos mostrar un número de tipo float o double con un número determinado de decimales y no con los que por defecto muestra Java.

A partir de la versión Java 5 se incorporan los métodos format y printf que permiten aplicar un formato a la salida de datos por pantalla. Ambos realizan la misma función, tienen exactamente el mismo formato y emulan la impresión con formato printf() de C.

Veamos primero varios ejemplos de printf en Java y después explicaremos en detalle la sintaxis de printf. Si queremos mostrar el número 12.3698 de tipo double con dos decimales:

**System.out.printf("%.2f %n", 12.3698);**

El primer % indica que en esa posición se va a escribir un valor. El valor a escribir se encuentra a continuación de las comillas, .2 indica el número de decimales. La f indica que el número es de tipo **float** o **double**. En la tabla que aparece más adelante podéis ver todos los caracteres de conversión para todos los tipos de datos. %n indica un salto de línea, equivale a \n. Con printf podemos usar ambos para hacer un salto de línea.

**La salida por pantalla es: 12,37**

Comprobamos que printf realiza un redondeo para mostrar los decimales indicados.

Lo más común será que tengamos el valor en una variable, en ese caso si queremos escribir el valor de n con tres decimales:

double n = 1.25036;

System.out.printf("%.3f %n", n);

Salida:

1,250

Para mostrar el signo + en un número positivo:

double n = 1.25036;

System.out.printf("%+.3f %n", n);

Salida:

+1.250

Si el número a mostrar es un **entero** se utiliza el carácter d:

int x = 10;

System.out.printf("%d %n", x);

Salida:

10

Para mostrarlo con signo:

int x = 10;

System.out.printf("%+d %n", x);

Salida:

+10

Para mostrar varias variables pondremos **tantos % como valores vamos a mostrar**. Las variables se escriben a continuación de las comillas separadas por comas:

double n = 1.25036;

int x = 10;

System.out.printf("n = %.2f x = %d %n", n, x);

Salida:

n = 1,25 x = 10

Cuando hay varias variables podemos indicar de cuál de ellas es el valor a mostrar escribiendo 1$, 2$, 3$, ... indicando que el valor a mostrar es el de la primera variable que aparece a continuación de las comillas, de la segunda, etc.

La instrucción anterior la podemos escribir así:

System.out.printf("n = %1$.2f x = %2$d %n", n, x);

Este número es opcional, si no aparece se entenderá que el primer valor proviene de la primera variable, el segundo de la segunda, etc.

Si queremos mostrar el número 123.4567 y su cuadrado ambos con dos decimales debemos escribir:

double n = 123.4567;

System.out.printf("El cuadrado de %.2f es %.2f\n", n, n\*n);

Salida:

El cuadrado de 123,46 es 15241,56

**printf** permite mostrar valores con un ancho de campo determinado. Por ejemplo, si queremos mostrar el contenido de n en un ancho de campo de 10 caracteres escribimos:

double n = 1.25036;

System.out.printf("%+10.2f %n", n);

Salida:

*bbbbb*+1.25

Donde cada *b* indica un espacio en blanco.

El 10 indica el tamaño en caracteres que ocupará el número en pantalla. Se cuentan además de las cifras del número el punto decimal y el signo si lo lleva. En este caso el número ocupa un espacio de 5 caracteres (3 cifras, un punto y el signo) por lo tanto se añaden 5 espacios en blanco al principio para completar el tamaño de 10.

Si queremos que en lugar de espacios en blancos nos muestre el número completando el ancho con ceros escribimos:

System.out.printf("%+010.2f %n", n);

Salida:

+000001.25

Más ejemplos de printf:

Mostrar el número 1.22 en un ancho de campo de 10 caracteres y con dos decimales.

double precio = 1.22;

System.out.printf("%10.2f", precio);

Salida:

*bbbbbb*1.22

(el carácter b indica un espacio en blanco)

El número ocupa un espacio total de 10 caracteres incluyendo el punto y los dos decimales.

Mostrar la cadena "Total:" con un ancho de 10 caracteres y alineada a la izquierda:

System.out.printf("%-10s", "Total:");

Salida:

Total:*bbbb*

El caracter s indica que se va a mostrar una cadena de caracteres.

El signo - indica alineación a la izquierda.

Mostrar la cadena "Total:" con un ancho de 10 caracteres y alineada a la derecha:

System.out.printf("%10s", "Total:");

Salida:

*bbbb*Total:

Al final puedes ver un ejemplo completo con distintos usos de printf.

Veamos ahora detenidamente la sintaxis de printf:

La sintaxis general de printf es:

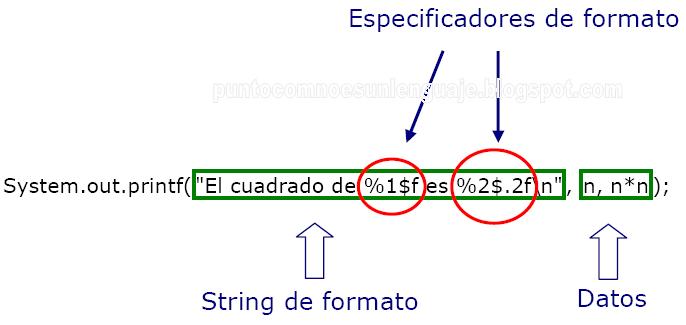
**printf (String de formato, Object … datos);**

El *String de formato* es una cadena de caracteres que contiene:

- **Texto fijo** que será mostrado tal cual.

- **Especificadores de formato** que determinan la forma en que se van a mostrar los datos.

Los *datos* representan la información que se va a mostrar y sobre la que se aplica el formato anterior. El número de datos que se pueden mostrar es variable.



Explicación de cada una de las partes que aparecen en la instrucción printf:

**Especificadores de formato:**

La sintaxis para los especificadores de formato de printf es:

%[posición\_dato$][indicador\_de\_formato][ancho][.precisión]carácter\_de\_conversión

Los elementos entre corchetes son opcionales.

**posición\_dato**: indica la posición del dato sobre el que se va aplicar el formato. El primero por la izquierda ocupa la posición 1.

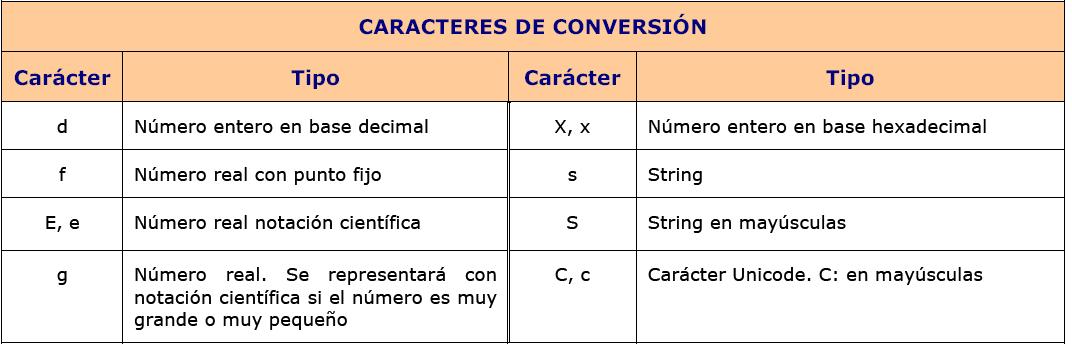
**indicador\_de\_formato**: es el conjunto de caracteres que determina el formato de salida. Los indicadores de formato de printf en Java son:



**ancho**: Indica el tamaño mínimo, medido en número de caracteres, que debe ocupar el dato en pantalla.

**.precisión**: Indica el número de decimales que serán representados. Solo aplicable a datos de tipo float o double.

**carácter\_de\_conversión**: Carácter que indica cómo tiene que ser formateado el dato. Los más utilizados se muestran en la tabla.



Ejemplo completo con distintos usos de printf en Java:

**public** **class** Printf2 {

**public** **static** **void** main(String[] args) {

**double** q = 1.0/3.0;

System.***out***.printf("1.0/3.0 = %5.3f %n", q);

System.***out***.printf("1.0/3.0 = %7.5f %n", q);

q = 1.0/2.0;

System.***out***.printf("1.0/2.0 = %09.3f %n", q);

q = 1000.0/3.0;

System.***out***.printf("1000/3.0 = %7.1e %n", q);

q = 3.0/4567.0;

System.***out***.printf("3.0/4567.0 = %7.3e %n", q);

q = -1.0/0.0;

System.***out***.printf("-1.0/0.0 = %7.2e %n", q);

q = 0.0/0.0;

System.***out***.printf("0.0/0.0 = %5.2e %n", q);

System.***out***.printf("pi = %5.3f; e = %10.4f %n", Math.***PI***, Math.***E***);

**double** r = 1.1;

System.***out***.printf("%1$.5f \*%1$.5f %n", Math.***PI***, r);

System.***out***.printf("%1$6.6f \*%1$6.6f %n", Math.***PI***, r);

System.***out***.printf("C = 2 \* %1$5.5f \* %2$4.1f, "+"A = %2$4.1f \* %2$4.1f \* %1$5.5f %n", Math.***PI***, r);

}

}

**Salida**:

1.0/3.0 = 0,333

1.0/3.0 = 0,33333

1.0/2.0 = 00000,500

1000/3.0 = 3,3e+02

3.0/4567.0 = 6,569e-04

-1.0/0.0 = -Infinity

0.0/0.0 = NaN

pi = 3,142; e = 2,7183

3,14159 \*3,14159

3,141593 \*3,141593

C = 2 \* 3,14159 \* 1,1, A = 1,1 \* 1,1 \* 3,14159

## 1. Formatter (clase) java.util.Formatter

Esta clase permite generar String a partir de datos usando una especificación de formato.

La funcionalidad se presta a diferentes clases para un uso cómodo:

String s =String.format(formato, valores ...);

System.out.printf(formato, valores ...);

System.out.format(formato, valores ...);

Es muy similar a printf.

Formatter fmtr = new Formatter();

String saludo =   
String.format( "Hola amigos, bienvenidos a %s !", "Madrid");

System.out.println(saludo);

saludo =   
String.format("Hola %2$s, bienvenidos %1$s !", "Madrid", "chicos");

System.out.println(saludo);

int numero = 425;

fmtr.format("%08d", numero);

System.out.println("El numero formateado " + fmtr);

Object result = fmtr.format("%1$4d - el año %2$4.2f", 1951, Math.PI);

System.out.println(result);

// Otra forma de hacerlo

System.out.format("%1$04d - el año %2$4.2f%n", 1951, Math.PI);

// Parecido al printf que ya vimos:

System.out.printf("%1$04d - el año %2$4.2f%n", 1951, Math.PI);

System.out.printf("PI es mas o menos %1$4.2f%n", Math.PI);

System.out.printf("Tu nombre es %1$s y el mio es %2$s%n", "Juan", "Pepi");

fmtr.close();

### 1.1. Como usar el Formateador decimal en Java

La clase **DecimalFormat**: Es una **clase de Java** que nos permite mostrar los números con un formato deseado, puede ser limitando los dígitos decimales, si usaremos punto o coma, etc, incluso podemos tomar los valores desde un **JTextField** y reconstruir el número, ejemplo:

//import requerido para el Formateador

import java.text.DecimalFormat;

DecimalFormat formateador = new DecimalFormat("####.####");

// Imprime esto con cuatro decimales, es decir: 7,1234

System.out.println (formateador.format (7.12342383));

Si utilizamos ceros en lugar de los #, los dígitos no existentes se rellenarán con ceros, ejemplo:

DecimalFormat formateador = new DecimalFormat("0000.0000");

// Imprime con 4 cifras enteras y 4 decimales: 0001,8200

System.out.println (formateador.format (1.82));

También podemos utilizar el signo de porcentaje (%) en la máscara y así el número se multiplicará automáticamente por 100 al momento de Imprimir.

DecimalFormat formateador = new DecimalFormat("###.##%");

// Imprime: 68,44%

System.out.println (formateador.format(0.6844));

## DecimalFormat (clase)

La clase **DecimalFormat** usa por defecto el formato para el lenguaje que tengamos instalado en el ordenador. Es decir, si nuestro sistema operativo está en español, se usará la coma para los decimales y el punto para los separadores de miles. Si estamos en inglés, se usará el punto decimal. Una opción para cambiar esto, es utilizar la clase **DecimalFormatSymbols**, que vendrá rellena con lo del idioma por defecto, y cambiar en ella el símbolo que nos interese. Por ejemplo, si estamos en español y queremos usar el punto decimal en vez de la coma, podemos:

import java.text.DecimalFormat;

import java.text.DecimalFormatSymbols;

DecimalFormatSymbols simbolos = new DecimalFormatSymbols();

simbolos.setDecimalSeparator('.');

DecimalFormat formateador = new DecimalFormat("####.####",simbolos);

// Imprime: 3.4324

System.out.println(formateador.format (3.43242383));

Reconstruyendo un Número:

Supongamos que leemos algún número desde consola o un JTextField y deseamos reconstruirlo con DecimalFormat, se hace de la siguiente forma:

JTextField textField = new JTextField();

DecimalFormat formateador = new DecimalFormat("####.####");

String texto = textField.getText();

try

{

// parse() lanza una ParseException en caso de fallo que hay que capturar.

Number numero = formateador.parse(texto);

double valor = numero.doubleValue();

// Optimizando las Lineas anteriores:

// double valor = formateador.parse(texto).doubleValue();

}

catch (ParseException e)

{

// Error. El usuario ha escrito algo que no se puede convertir a número.  
  
}

# UT3 Otra forma de leer del usuario

## 1. Alertas y diálogos en Java

La clase JOptionPane es una clase que hereda de JComponent. Esta clase nos permitirá crear alertas o cuadros de diálogo simples para poder solicitar o mostrar información al usuario.

Los métodos que veremos son:

1. JOptionPane.showMessageDialog(…);
2. JOptionPane.showConfirmDialog(….);
3. JOptionPane.showInputDialog(…);
4. JOptionPane.showOptionDialog(…);

Cada uno de estos métodos presenta una particularidad distinta pero todos ellos nos muestran una ventana pop up que nos permitirá captar información del usuario.

### showMessageDialog

**JOptionPane. showMessageDialog (Component componentePadre, Object mensaje, String titulo, int tipoDeMensaje)**

Nos sirve para mostrar información, por ejemplo alguna alerta que queremos hacerle al usuario. Veamos cuales son los principales argumentos del método:

***componentePadre***: es el Frame desde el cual lo llamamos. Si queremos lo podemos poner null de momento. Si hubiera un padre, el dialogo se mostraría sobre él.

***mensaje***: es lo que queremos que diga el cuadro de dialogo.

***titulo***: el título de la ventana.

***tipoDeMensaje***: son constantes que le dirán a java qué tipo de mensaje queremos mostrar. De acuerdo a esto serán los iconos que se mostrarán en el cuadro de dialogo.

Las opciones son:

* **ERROR\_MESSAGE**
* **INFORMATION\_MESSAGE**
* **WARNING\_MESSAGE**
* **QUESTION\_MESSAGE**
* **PLAIN\_MESSAGE**

Si quisiéramos mostrar un icono personalizado, podemos agregarlo al final como un argumento más.

|  |
| --- |
| Ejemplo:  **JOptionPane.showMessageDialog(null, "Acceso Denegado", "Error", JOptionPane.ERROR\_MESSAGE);** |

### showConfirmDialog

Este método sirve para pedirle al usuario una confirmación. Por ejemplo, una confirmación de salida del sistema

**Int respuesta = JOptionPane. showConfirmDialog(Component componentePadre, Object mensaje, String titulo, int tipoDeOpcion);**

**Lo anterior es la versión corta de los argumentos del método. La versión larga incluye el tipo de mensaje y el icono, por si queremos personalizarlo.**

Los argumentos son idénticos a los del método anterior. Excepto por el tipo de opción, que es otra constante y los valores pueden ser:

* **DEFAULT\_OPTION**
* **YES\_NO\_OPTION**
* **YES\_NO\_CANCEL\_OPTION**
* **OK\_CANCEL\_OPTION**

Como vemos, el método devuelve un entero que nos permitirá captar cual es la opción elegida por el usuario. Los valores serán **0** para **Sí**, **1** para **No**, **2** para **Cancelar** y **-1** para el **cierre de la ventana**. Así podremos preguntar cuál es el valor devuelto y realizar la acción que deseamos.

|  |
| --- |
| Ejemplo:  **int i =JOptionPane.showConfirmDialog(this, "¿Realmente Desea Salir de Hola Swing?",  "Confirmar Salida",JOptionPane.YES\_NO\_OPTION);**  **if(i= =0){**  **System.exit(0);**  **}** |

### showInputDialog

Este método nos muestra una ventana donde podremos insertar un String. Por ejemplo, cuando queremos que el usuario inserte su nombre. **La versión corta del método es**:

**String respuesta = JOptionPane.showInputDialog(Object mensaje)**

Este método devolverá un String para poder utilizarlo después. La versión larga de los argumentos del método es similar a los anteriores.

|  |
| --- |
| Ejemplo:  **String nombre =JOptionPane.showInputDialog("Inserte su nombre");** |

También podemos crear un cuadro de dialogo que contenga un combo con las opciones predeterminadas que le queremos dar al usuario.

|  |
| --- |
| Ejemplo:  **Object[] valoresPosibles = {"Pedro", "Juan", "Carlos" };**  **Object jefe = JOptionPane.showInputDialog(null, "Seleccione cual es su Jefe Inmediato", "Seleccionar Jefe", JOptionPane.QUESTION\_MESSAGE, null, valoresPosibles, valoresPosibles[0]);** |

El array de valores posibles nos muestra en un combo cuáles serán los jefes que podemos mostrar. El último argumento del método nos muestra cuál será la opción seleccionada por defecto

### showOptionDialog

Con este método podemos crear cuadros de diálogos con botones personalizados. Está bien para personalizar los cuadros de dialogo.

El método tiene la forma:

**int res = JOptionPane.showOptionDialog(Component componentePadre, Object mensaje, String titulo, int tipoDeOpcion, int tipoMensaje, Icon icono, Object[] botones, Object botonDefault)**

**Aquí lo único que varía con el resto, es el array de botones que vamos a tener, debemos destacar que no hace falta que creemos botones, solo tenemos que poner cuál será el texto que saldrá en él.**

|  |
| --- |
| Object[] botones = {"No, nada", "Un poquito", "Me estalla" };  int i=JOptionPane.showOptionDialog(null, "¿Te duele la cabeza?", "ventanita",  JOptionPane.DEFAULT\_OPTION, JOptionPane.WARNING\_MESSAGE,  null, botones, botones[0]);  System.out.println(i); |

Después podemos tomar la respuesta como lo hacíamos con el confirmDialog.

Se puede consultar el API y https://serprogramador.es/programando-mensajes-de-dialogo-en-java-parte-1/

## 2. Convertir String a número

**Instrucciones de Java: ParseInt, ParseDouble y ParseFloat:**

La función parseFloat() analiza una cadena y devuelve un número de punto flotante.

Float.parseFloat(String str)

La función parseInt() analiza una cadena y devuelve un entero.

Integer.parseInteger(String str)

La función parseDouble() analiza una cadena y devuelve un número decimal doble.

Double.parseDouble(String str)

# UT3 Sus Métodos en Java, funciones y procedimientos. Cómo hacerlos y usarlos

Los métodos en Java, las funciones y los procedimientos, especialmente en Java, son una herramienta indispensable para programar. Java nos permite crear o hacer nuestros propios métodos y usarlos sencillamente como también nos facilita hacer uso de los métodos de otras librerías (funciones matemáticas, aritméticas, de archivos, de fechas, etc. Cualquiera que sea el caso, las funciones permiten automatizar tareas que requiramos con frecuencia y que además se puedan generalizar por medio de parámetros o argumentos.

Aprender a crear métodos en Java y usarlos correctamente es de gran importancia, separar nuestro código en módulos y según las tareas que requerimos. En java una función debe contener la implementación de una utilidad de nuestra aplicación, esto nos pide que por cada utilidad básica (abrir, cerrar, cargar, mover, etc.) sería adecuado tener al menos una función asociada a ésta, pues sería muy complejo usar o crear un método que haga todo de una sola vez, por esto es muy buena idea separar cada tarea en una función o método (según corresponda).

En Java es mucho más común hablar de métodos que de funciones y procedimientos y esto se debe a que en realidad un método, una función y un procedimiento NO son lo mismo, veamos la diferencia:

## 1. ¿Funciones, métodos o procedimientos?

**Es muy común entre programadores que se hable indistintamente de estos tres términos, sin embargo poseen diferencias fundamentales.**

### Funciones:

Las funciones son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente reciben parámetros, cuyos valores utilizan para efectuar operaciones y adicionalmente devuelven/retornan un valor. En otras palabras, una función puede recibir parámetros o argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y devuelve un valor usando la instrucción **return**. **Si no devuelve algo, entonces no es una función (se suele llamar procedimiento en este caso)**.

### Métodos:

Los métodos y las funciones (y/o procedimientos) en Java pueden realizar las mismas tareas, es decir, son funcionalmente idénticos, pero su diferencia radica en la manera en que hacemos uso de uno u otro (el contexto). Un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo un método está asociado a un objeto, **SIEMPRE**, **básicamente un método es una función que pertenece a un objeto o clase, mientras que una función existe por sí sola, sin necesidad de un objeto para ser usada**.

**Nota:** En Java se debe hablar de métodos y no de funciones, pues en Java estamos siempre obligados a crear un objeto para usar el método. Para que sea una función/procedimiento, esta debe ser static, para que no requiera de un objeto para ser llamada.

### Procedimientos:

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, pueden recibir o no argumentos. En el contexto de Java un procedimiento es básicamente un método cuyo tipo de retorno es void, que no nos obliga a utilizar una sentencia return.

## 2. Crear un método en Java

La sintaxis para declarar una función es muy simple, veamos:

*Ejemplo:* ***public static void main (String[] args)***

**[acceso] [modificador] tipoDevuelto nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento]...])**

***//Argumento es lo mismo que variable***

{

/\*

\* Bloque de instrucciones

\*/

return valor;

*//return: devuelve solo el valor de la variable, no la variable*

}

El primer componente corresponde al **[modificador de acceso](https://www.programarya.com/Cursos/Java/Modificadores-de-Acceso" \t "Modificadores de acceso en Java)**, que puede ser public, private o protected, este último es opcional, si no ponemos nada, se asume el modificador de acceso por defecto.

El segundo componente es el **[modificador](https://www.programarya.com/Cursos/Java/Sistema-de-Tipos/Final-y-Constantes" \t "Final y constantes en Java)** que puede ser final o static (o ambas), también es opcional. Recordemos que un método o función siempre retorna algo, por lo tanto es obligatorio declararle un **tipo** (el tercer componente de la sintaxis anterior), puede ser entero (int), booleano (boolean), o cualquiera que consideremos, incluso tipos complejos. Si no es función, pondremos void.

Luego debemos darle un **nombre** a dicha función/método/procedimiento, para poder identificarla y llamarla (invocarla) durante la ejecución. Después, en el interior del paréntesis, podemos poner los **argumentos** o parámetros.

Cuando hemos acabado con la definición de la "firma" o prototipo del método, definimos su funcionamiento entre llaves. Todo lo que esté dentro de las llaves, es parte del cuerpo del método y este se ejecuta hasta llegar a una instrucción *return* o cuando se alcanza la última sentencia a ejecutar.

## 3. Return y void

En algunas ocasiones, no es necesario que el método estático tenga que devolver un valor al finalizar su ejecución. En este caso, el tipo de dato que debe indicar en la cabecera de declaración del método es el tipo void y la sentencia return no viene seguida de ninguna expresión. Este return solitario se puede omitir.

Sintaxis:

return;

//Quiere decir vuelvo al sitio donde me han llamado para seguir con la siguiente instrucción, no continua a partir de aqui

**Ejemplo:**

Seguidamente se muestra un ejemplo de declaración y uso de un método que devuelve el cubo de un valor numérico real con una sentencia return:

/\*\*

\* Demostracion del metodo cubo

\*/

public class PruebaCubo {

public static void main (String [] args){

System.out.println("El cubo de 7.5 es: " + cubo(7.5));

// llamada dentro de la sentencia System.out.println

}

public static double cubo (double x) {

// devuelve un double

// declaracion

return x\*x\*x;

}

}

Si no hay sentencia return dentro de un método, su ejecución continúa hasta que se alcanza el final del método y entonces se devuelve la secuencia de ejecución al lugar dónde se invocó al método.

Un método cuyo tipo de retorno no es void necesita siempre devolver algo. Si el código de un método contiene varias sentencias if debe asegurarse de que cada una de las posibles opciones devuelve un valor. En caso contrario, se generaría un error de compilación.

Por ejemplo:

/\*\*

\* Demostracion de la funcion esPositivo

\*/

public class PruebaPositivo {

public static void main (String [] args) {

for (int i=5; i>=-5; i--)

System.out.println(i + " es positivo: " + esPositivo(i));

}

public static boolean esPositivo(int x) {

if (x<0) return false;

if (x>0) return true;

}

}

Ejemplo de intento de compilación del código anterior:

$>javac PruebaPositivo.java

pruebaPositivo.java:14: missing return statement

}^

***Lo que pasa es que falta el valor 0, que no es ni positivo ni negativo, por eso te da error.***

# UT3 Generación de números aleatorios

## 1. Números aleatorios en Java

Para generar números aleatorios en Java tenemos dos opciones. Por un lado, podemos usar *Math.random()*, por otro la clase *java.util.Random*. La primera es de uso más sencillo y rápido. La segunda nos da más opciones.

* [Math.random()](http://chuwiki.chuidiang.org/index.php?title=Generar_números_aleatorios_en_Java" \l "Math.random.28.29)
* [Clase java.util.Random](http://chuwiki.chuidiang.org/index.php?title=Generar_números_aleatorios_en_Java" \l "Clase_java.util.Random)

## 2. Math.random()

La llamada a *Math.random()* devuelve un número aleatorio entre 0.0 y 1.0, excluido este último valor, es decir, puede devolver 0.346442, 0.2344234, 0.98345,....

En muchas de nuestras aplicaciones no nos servirá este rango de valores. Por ejemplo, si queremos simular una tirada de dado, queremos números entre 1 y 6 sin decimales. Debemos echar unas cuentas para obtener lo deseado.

En primer lugar, miramos cuántos valores queremos. En nuestro caso del dado son 6 valores, del 1 al 6 ambos incluido. Debemos entonces multiplicar *Math.random()* por 6.

Math.random()\*6 // Esto da valores de 0.0 a 6.0, excluido el 6.0

[Math](http://www.google.com/search?hl=en&q=allinurl%3AMath+java.sun.com&btnI=I'm Feeling Lucky)**.random()\*limite+1;**

Donde límite será el número más alto del rango. Los números genéricos que generaremos irán entre el 1 y el límite.

Math.random()\*6 + 1

// Esto da valores entre 1.0 y 7.0 excluido el 7.0

Finalmente, para conseguir un entero, quitamos los decimales usando la clase *Math.floor()*

int valorDado = Math.floor(Math.random()\*6+1);

*// A mi me sale que Math.floor te lo pasa a un double, igual es Math.round*

O también haciendo un casting a un entero:

**int numeroAleatorio = (int) (**[Math](http://www.google.com/search?hl=en&q=allinurl%3AMath+java.sun.com&btnI=I'm Feeling Lucky)**.random()\*6+1);**

En general, para conseguir un número entero entre [N, M] con N < M y ambos incluidos, debemos usar esta fórmula:

int valorEntero = Math.floor(Math.random()\*(M-N+1)+N);

// Valor entre M y N, ambos incluidos.

## 3. Clase java.util.Random

La clase *java.util.Random* debemos instanciarla, a diferencia del método *Math.random()*. A cambio, tendremos bastantes más posibilidades.

Podemos usar un constructor sin parámetros o bien pasarle una semilla. Si instanciamos varias veces la clase con la misma semilla, tendremos siempre la misma secuencia de números aleatorios.

Random r1 = new Random();

Random r2 = new Random(4234);

Random r3 = new Random(4234); // r2 y r3 darán la misma secuencia.

Lo más fácil es usar el constructor sin parámetros, que normalmente dará secuencias distintas en cada instancia. De todas formas, una manera de obtener una semilla que sea distinta cada vez que ejecutemos nuestro programa puede ser obtener el tiempo actual en milisegundos con *System.currentTimeMillis()*, que dará números distintos salvo que hagamos la instancia justo en el mismo instante de tiempo.

Con esta clase, una vez instanciada, nuestro problema del dado sería bastante más sencillo, usando el método *nextInt(int n)*, que devuelve un valor entre 0 y n, excluido n

Random r = new Random();

int valorDado = r.nextInt(6)+1; // Entre 0 y 5, más 1. Es decir, 1-6

También tenemos funciones que nos dan un valor aleatorio siguiendo una curva de Gauss o que nos rellenan un array de bytes de forma aleatoria. Y por supuesto, el *nextDouble()* que devuelve un valor aleatorio entre 0.0 y 1.0, excluido este último.

La clase *Random* proporciona un generador de números aleatorios que es más flexible que la función estática [random](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/estatico/math/math.htm" \l "azar) de la clase *Math*.

Para crear una secuencia de números aleatorios tenemos que seguir los siguientes pasos:

1. Proporcionar a nuestro programa información acerca de la clase *Random.* Al principio del programa escribiremos la siguiente sentencia.

import java.util.Random;

1. Crear un objeto de la clase *Random*
2. Llamar a una de las funciones miembro que generan un número aleatorio
3. Usar el número aleatorio.

### 3.1 Constructores

La clase dispone de dos constructores, el primero crea un generador de números aleatorios cuya semilla es inicializada en base al instante de tiempo actual.

Random rnd = new Random();

El segundo, inicializa la semilla con un número del tipo **long**.

Random rnd = new Random(3816L);

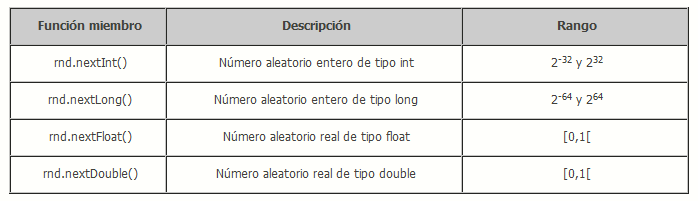
El sufijo L no es necesario, ya que aunque 3816 es un número **int** por defecto, es promocionado automáticamente a **long**.

Aunque no podemos predecir que números se generarán con una semilla particular, podemos sin embargo, duplicar una serie de números aleatorios usando la misma semilla. Es decir, cada vez que creamos un objeto de la clase *Random* con la misma semilla obtendremos la misma secuencia de números aleatorios. Esto no es útil en el caso de loterías, pero puede ser útil en el caso de juegos, exámenes basados en una secuencia de preguntas aleatorias, las mismas para cada uno de los estudiantes, simulaciones que se repitan de la misma forma una y otra vez, etc.

### 3.2 Funciones miembro

Podemos cambiar la semilla de los números aleatorios en cualquier momento, llamando a la función miembro *setSeed*.

rnd.setSeed(3816);



Podemos generar números aleatorios de cuatro formas diferentes:

rnd.nextInt();

Genera un número aleatorio entero de tipo **int**

rnd.nextLong();

Genera un número aleatorio entero de tipo **long**

rnd.nextFloat();

Genera un número aleatorio de tipo **float** entre 0.0 y 1.0, aunque siempre menor que 1.0

rnd.nextDouble();

Genera un número aleatorio de tipo **double** entre 0.0 y 1.0, aunque siempre menor que 1.0

Casi siempre usaremos esta última versión. Por ejemplo, para generar una secuencia de 10 números aleatorios entre 0.0 y 1.0 escribimos

for (int i = 0; i < 10; i++) {

System.out.println(rnd.nextDouble());;

}

Para crear una secuencia de 10 números aleatorios enteros comprendidos entre 0 y 9 ambos incluidos escribimos

int x;

for (int i = 0; i < 10; i++) {

x = (int)(rnd.nextDouble() \* 10.0);

System.out.println(x);

}

*(int)* transforma un número decimal **double** en entero **int** eliminando la parte decimal.

## 4. Comprobación de la uniformidad de los números aleatorios

Podemos comprobar la uniformidad de los números aleatorios generando una secuencia muy grande de números aleatorios enteros comprendidos entre 0 y 9 ambos inclusive. Contamos cuantos ceros aparecen en la secuencia, cuantos unos, ... cuantos nueves, y guardamos estos datos en los elementos de un array.

Primero creamos un array *ndigitos* de 10 de elementos que son enteros.

int[] ndigitos = new int[10];

Inicializamos los elementos del array a cero.

for (int i = 0; i < 10; i++) {

ndigitos[i] = 0;

}

Creamos una secuencia de 100000 números aleatorios enteros comprendidos entre 0 y 9 ambos inclusive (véase el apartado anterior)

for (long i=0; i < 100000L; i++) {

n = (int)(rnd.nextDouble() \* 10.0);

ndigitos[n]++;

}

Si *n* sale cero suma una unidad al contador de ceros, *ndigitos[0]*. Si *n* sale uno, suma una unidad al contador de unos, *ndigitos[1]*, y así sucesivamente.

Finalmente, se imprime el resultado, los números que guardan cada uno de los elementos del array *ndigitos*

for (int i = 0; i < 10; i++) {

System.out.println(i+": " + ndigitos[i]);

}

Observaremos en la consola que cada número 0, 1, 2...9 aparece aproximadamente 10000 veces.

## 5. Secuencias de números aleatorios

En la siguiente porción de código, se imprime dos secuencias de cinco números aleatorios uniformemente distribuídos entre [0, 1), separando los números de cada una de las secuencias por un carácter tabulador.

System.out.println("Primera secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

System.out.println("Segunda secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

Comprobaremos que los números que aparecen en las dos secuencias son distintos.

En la siguiente porción de código, se imprime dos secuencias iguales de números aleatorios uniformemente distribuidos entre [0, 1). Se establece la semilla de los números aleatorios con la función miembro *setSeed*.

**rnd.setSeed(3816);**

System.out.println("Primera secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

**rnd.setSeed(3816);**

System.out.println("Segunda secuencia");

for (int i = 0; i < 5; i++) {

System.out.print("\t"+rnd.nextDouble());

}

System.out.println("");

|  |
| --- |
| package azar;  import java.util.Random;  public class AzarApp {  public static void main (String[] args) {  int[] ndigitos = new int[10];  int n;    Random rnd = new Random();  // Inicializar el array  for (int i = 0; i < 10; i++) {  ndigitos[i] = 0;  }  // verificar que los números aleatorios están uniformente distribuídos  for (long i=0; i < 100000L; i++) {  // genera un número aleatorio entre 0 y 9  n = (int)(rnd.nextDouble() \* 10.0);  //Cuenta las veces que aparece un número  ndigitos[n]++;  }  // imprime los resultados  for (int i = 0; i < 10; i++) {  System.out.println(i+": " + ndigitos[i]);  }  //Dos secuencias de 5 número (distinta semilla)  System.out.println("Primera secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  System.out.println("Segunda secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  //Dos secuencias de 5 número (misma semilla)  rnd.setSeed(3816L);  System.out.println("Primera secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  rnd.setSeed(3816);  System.out.println("Segunda secuencia");  for (int i = 0; i < 5; i++) {  System.out.print("\t"+rnd.nextDouble());  }  System.out.println("");  }  } |

# UT3 Ejercicios

## Funciones 1

**1.-** Crea una subrutina en java a la que se le pase un número N que se le pedirá al usuario y muestre por pantalla la frase “Módulo ejecutándose” N veces.

**2.-**Muestra por pantalla los resultados de las 4 operaciones básicas entre 2 números que se le pedirán al usuario, realizando una subrutina para cada una de dichas operaciones. El resultado de la operación se imprime dentro de la subrutina. Y fuera

**3.-** Crea una función en java que reciba dos números reales y devuelva su suma.

**4.-** Crea una función en java que reciba un número entero y devuelva su factorial.

**5.-** Escribe una aplicación en java que le pida al usuario que escriba el nombre de una operación en java entre las siguientes: suma, resta, multiplicación y división. En caso de introducir una distinta se le comunicará el error y se pedirá de nuevo la operación.

Con una instrucción switch, diseña un menú que en función de cuál sea la operación elegida ejecutará una de las cuatro funciones que escribirás para llevar a cabo la operación (sumar(float, float), multiplicar(float, float), restar(float, float), dividir(float, float)). Cada función de las anteriores recibirá los dos operadores y devolverá al programa principal el resultado, desde donde se imprimirá. En el menú habrá una opción de salir para poder terminar el programa.

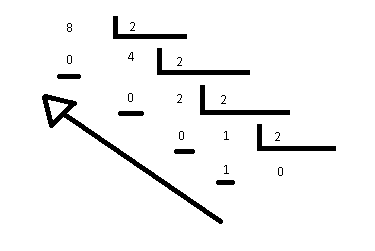
**6.-** Crea una aplicación que nos calcule el área de un círculo, cuadrado o un triángulo. Pediremos de qué figura queremos calcular su área (nos darán una de las tres) y según lo introducido pedirá los valores necesarios para calcular el área. Crea un método por cada figura para calcular cada área que devolverá un número real como valor del área. Muestra el resultado por pantalla. Añade un menú para mejorar el ejercicio. En el menú habrá una opción de salir para poder terminar el programa.

Datos para el cálculo del área de cada figura:

* **Circulo**: (radio^2)\*PI
* **Triangulo**: (base \* altura) / 2
* **Cuadrado**: lado \* lado

**7.-** Crea una aplicación que nos convierta un número en base decimal a binario. Esto lo realizará un método al que le pasaremos el número como parámetro, devolverá un String con el número convertido a binario. Para convertir un número decimal a binario, debemos dividir entre 2 el número y el resultado de esa división se divide entre 2 de nuevo hasta que no se pueda dividir más (la división da 0), el resto que obtengamos de cada división formara el número binario, de abajo a arriba.

Ejemplo: si introducimos un **8** nos deberá devolver **1000**



**8.-** Crea una aplicación que nos cuente el número de cifras de un número **entero positivo** (hay que controlarlo) pedido por teclado. Crea un método que realice esta acción, pasando el número por parámetro, devolverá el número de cifras.

**9.-** Crea un aplicación que nos convierta una cantidad de euros introducida por teclado a otra moneda, estas pueden ser dolares, yenes o libras. El método tendrá como parámetros, la cantidad de euros y la moneda a pasar que será una cadena. Haz dos versiones del programa:

El método no devolverá ningún valor, mostrará un mensaje indicando el cambio (void).

El método devolverá un valor que indica el cambio y este se mostrará en el programa principal.

El cambio de divisas es:

* 0.86 libras es un 1 €
* 1.28611 $ es un 1 €
* 129.852 yenes es un 1 €

## Funciones 2

Crea una biblioteca de funciones matemáticas que contenga las siguientes funciones. Recuerda que puedes usar unas dentro de otras si es necesario. Los números serán todos long. Todas las funciones devuelven al código llamante lo que se pide.

**1.-** **voltea**: Le da la vuelta a un número.

**2.-** **esCapicua**: Devuelve verdadero si el número que se pasa como parámetro es capicúa y falso en caso contrario.

**3.- digitos**: Cuenta el número de dígitos de un número.

**4.-** **digitoN**: Devuelve el dígito que está en la posición n de un número. Se empieza contando por el 0 y de izquierda a derecha.

**5.- posicionDeDigito**: Da la posición de la primera ocurrencia de un dígito dentro de un número. Si no se encuentra, devuelve -1.

**6.-** **quitaPorDetras**: Le quita a un número n dígitos por detrás (por la derecha).

**7.-** **quitaPorDelante**: Le quita a un número n dígitos por delante (por la izquierda).

**8.-** **pegaPorDetras**: Añade un número a otro número por detrás.

**9.-** **pegaPorDelante**: Añade un número a otro número por delante.

**10.-** **trozoDeNumero**: Toma como parámetros las posiciones inicial y final dentro de un número y devuelve el trozo correspondiente.

**11.-** **juntaNumeros**: Pega dos números para formar uno. Se usa para apoyar la solución de 8 y 9.

## Ejercicios Cadenas (Strings)

**1**.- Realiza un programa que lea 11 cadenas de caracteres, para mostrar al final la que contenga el mayor número de vocales, ya sean mayúsculas o minúsculas. Se consideran además de las 5 vocales, todas las vocales acentuadas y la ü.

**2.-** Realiza un programa que lea un número entero y creando una función llamada esCapicua devuelva si es capicúa o no lo es (true y false respectivamente). Para ello, conviértelo en una cadena de caracteres y utiliza métodos de cadenas/caracteres.

**3.-** Crea una función en Java que reciba una cadena de caracteres y devuelva esta cadena invertida.

**4.-** Crea una función en Java que reciba una palabra y la devuelva invertida con efecto espejo, esto es, se concatena a la palabra original su inversa, compartiendo la última letra, que hará de espejo, por lo que la palabra obtenida se lee igual hacia adelante que hacia atrás. Por ejemplo, al introducir “teclado” devolverá “tecladodalcet”  y al introducir “goma” devolverá “gomamog”.

**5.-** Crea una función en Java que reciba dos cadenas, la primera cadena será en la que se buscará y la segunda será la cadena buscada. La función devolverá cuántas veces aparece la segunda cadena en la primera.

**6.-** Crea una función en Java que reciba dos cadenas de caracteres y que devuelva la primera cadena, pero transformando en mayúsculas la parte que coincide con la segunda cadena introducida. Por ejemplo, si se introducen las cadenas “Este es mi amigo Juan” y “amigo”, devolverá “Este es mi AMIGO Juan”.

## Básicos Hoja 1

**1.** Hacer un programa que sume dos números leídos por teclado y escriba el resultado con el mayor detalle posible.

**2.** Haz un programa que convierta un número de galones en litros y escriba el resultado.

En un galón hay 3.7854 litros.

**3.** Haz un programa que solicite una medida en pies y realice la conversión a pulgadas, yardas, cm y metros. Ten en cuenta que un pie tiene 12 pulgadas, una pulgada equivale a 2.54 cm y 1 yarda son 0.9144 metros.

**4.** Un año es bisiesto si es múltiplo de 4, exceptuando los múltiplos de 100, que solo son bisiestos cuando son múltiplos además de 400.

Por ejemplo, el año 1900 no fue bisiesto, pero el año 2000 sí.

Haz un programa que dado un año A nos diga si es o no bisiesto.

**5.** Hacer un programa que dado un día D, un mes M y un año A, calcule cual es el día siguiente. Se debe tener en cuenta que en los años bisiestos Febrero tiene 29 días y en los no bisiestos 28.

**6.** Hacer un programa para resolver una ecuación de segundo grado, sin tener en cuenta las soluciones complejas.

**7.** Modificar el ejercicio 1 para sumar 10 números leídos por teclado.

**8.** Modificar el ejercicio 7 para que permita sumar N números. El valor de N se debe leer previamente por teclado.

**9.** Modifica el programa 2 para que imprima una tabla de conversión de 1 hasta 100 galones, cada 10 galones imprimirá una línea de salida en blanco.

**10.** Hacer un programa que permita escribir los primeros 100 números pares.

**11.** La sucesión de Fibonacci se define de la siguiente forma:

a1=1

a2=1

…

an=an-1+an-2 para n>2,

Es decir, los dos primeros son 1 y el resto cada uno es la suma de los dos anteriores, los primeros son: 1, 1, 2, 3, 5, 8, 13, 21,...

Haz un programa que calcule e imprima N términos de la sucesión, N se pedirá por teclado.

**12.** Haz un programa que calcule el total de una factura, partiendo de una lista de parejas importe, IVA pertenecientes a la misma factura. La lista finaliza cuando el importe sea 0. El IVA puede ser el 4%, el 10% o el 21%, en cualquier otro caso se rechazan importe e IVA y se deben introducir de nuevo.

Finalmente, hay que realizar un descuento en función de la suma de los importes. Dicho descuento es del 0% si es menor que 1000, del 5% si es menor que 10000 (y mayor o igual que 1000) y del 10% si es mayor o igual que 10000. El descuento se debe aplicar al importe final.

El programa debe imprimir el **importe total** sin descuento y el **importe final** al aplicar el descuento que corresponda al importe total.

**13.** Hacer un programa que lea N números (se piden al usuario), calcule la suma de los pares y el producto de los impares y escriba ambos resultados.

**14.** Hacer un diagrama de flujo para calcular el máximo común divisor de dos números enteros positivos N y M siguiendo el algoritmo de Euclides, que es el siguiente:

1. Se divide N por M, sea R el resto.

2. Si R=0, el máximo común divisor es M y se acaba.

3. Si no, se asigna a N el valor de M y a M el valor de R y volvemos al paso 1.

**15.** Hacer un programa para calcular el factorial de N (N!=1·2·3·...·N).

**16.** Hacer un programa para sumar los N primeros términos de una progresión geométrica de primer término A y razón R (dados por teclado). Se debe realizar la suma sin emplear la fórmula que existe para ello. Muestra también los términos de la serie.

***Ejemplo de ejecución:***

Introducir número de términos

6

Introducir el primer término

5

Introducir la razón

3

***Salida***:

5 15 45 135 405 1215

La suma de los términos de la serie es 1820

**17.** Se dice que un número N es perfecto si la suma de sus divisores, excluido el propio número es N. Por ejemplo, 28 es perfecto, pues sus divisores son: 1, 2, 4, 7 y 14 y su suma es 1+2+4+7+14=28.

Haz un programa que dado un número N nos diga si es o no perfecto. Cambia el programa para que siga pidiendo números mientras el número introducido sea distinto de cero, que será la señal para parar el programa.

**18.** Hacer un programa que sea capaz de calcular el impuesto sobre la renta y lo escriba por pantalla hasta que se introduzca un salario igual a cero. El impuesto de la renta es el 15% del salario anual de cada persona, al que previamente se debe realizar una deducción en función del número de hijos, que es del 0% si tiene 0, del 5% si tiene 1 o 2 y del 15% si tiene más de 2.

Tanto el salario como el número de hijos se pedirán por teclado.

**19.** Hacer un programa que lea por teclado un numero N e imprima un triángulo rectángulo, de N filas. EJ: N=5, se pintará lo siguiente:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**20.** Hacer un programa en el que se pida por teclado un número mayor que 2 (el programa controlará que cumpla esto), y lo imprima de todas las formas posibles como producto de dos factores (no se tiene en cuenta la multiplicación por 1).

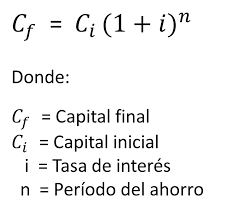
**Ej:** Con el número 36, tendría que visualizarse: 18\*2, 12\*3, 9\*4, 6\*6, 3\*12, 4\*9, 2\*18

## Básicos Hoja 2

**1.** Escribir un programa que imprima cada uno de los términos de la serie 2, 5, 7, 10, 12, 15, 17,..., 1800. Además calcule e imprima la suma de los términos.

**2.** Leer un capital C y averiguar e imprimir en cuantos meses se duplica, si lo colocamos a un interés compuesto del 5% mensual.

La fórmula a aplicar es:



**3.** En 1980 la ciudad A tenía 3.5 millones de habitantes y una tasa de crecimiento del 7% anual; y la ciudad B tenía 5 millones de habitantes y una tasa de crecimiento del 5% anual.

Si el crecimiento poblacional se mantiene constante en las dos ciudades, hacer un programa que calcule e imprima en qué año la población de la ciudad A es mayor que la de la ciudad B.

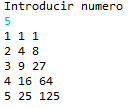
**4.** Haz un programa que lea un capital y calcule e imprima en cuantos meses se triplica si se coloca a un interés del 6% mensual.

**5.** Haz un programa que lea un número entero N y calcule el resultado de la siguiente serie: 1+1/2+1/3+1/4+1/5+...+1/N.

**6.** Haz un programa que lea un número entero N y calcule el resultado de la siguiente serie: 1-1/2+1/3-1/4+1/5-...+1/N.

**7.** Haz un programa que calcule la suma de los números pares comprendidos entre 10 y 50.

**8.** Haz un programa para imprimir una tabla de tres columnas y N filas con los cuadrados y los cubos de los N primeros números. Pide N al usuario.



**9.** Haz un programa que pida 2 números por teclado y calcule su producto mediante sumas sucesivas. Imprimir su resultado.

**10.** Hacer un programa que pida 2 números por teclado y calcule su división mediante restas sucesivas. Imprimir su resultado. Divide siempre el más grande entre el más pequeño.

***Ejemplo***: 1324 entre 312.

1324 - 312 = 1012 contamos una vez y seguimos porque 1012 >= 312

1012 - 312 = 700 contamos 2 veces y continuamos porque 700 >= 312

700 - 312 = 388 contamos 3 veces y continuamos porque 388 >= 312

388 - 312 = 76 contamos 4 veces y paramos porque 76 < 312

Luego la división es 4 y el resto 76.

**11.** Haz un programa que lea un número entero y genere y escriba el número resultante de invertir sus cifras.

**12.** Haz un programa que indique si un número entero N es primo o no.   
//Ejercicio 11 Estructuras repetitivas Pseint

**13.** Haz un programa que imprima el triángulo de Floyd hasta un valor dado. El triángulo contiene los números naturales correlativos, uno en la primera línea, dos en la segunda, etc.; es decir, en la fila n-ésima aparecen n valores.

Ejemplo:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

**14.** Escribe un programa para calcular A elevado a B, siendo A un número real cualquiera y B un valor entero positivo o nulo (sin emplear métodos ya hechos, claro). Si el exponente no es positivo o nulo se sigue pidiendo hasta que lo sea.

//Ejercicio 9 Estructuras repetitivas Pseint

## Básicos Hoja 3

**1.** Haz un programa en Java para jugar contra el ordenador a adivinar un número, generado aleatoriamente, entre 1 y 200. El usuario debe introducir un número por teclado y el programa le dirá mediante los símbolos '<' o '>', si el número introducido es menor o mayor que el generado por el ordenador.

Finalmente, se mostrará un mensaje informando de cuantos intentos se han necesitado para adivinar el número y si no se adivina se mostrará un mensaje diciendo que ha perdido.

El número máximo de intentos se pedirá por teclado.

**Explicación:** Para generar el número aleatorio poner:

int numAleatorio = (int) (Math.random()\*200+1);

**2.** Haz un programa en Java que muestre si dos números son o no amigos. Los números se pedirán por teclado.

**Explicación:** Se dice que dos números son amigos si la suma de los divisores del primero (excluido el propio número) es el segundo número y viceversa.

Ej: Los números 220 y 284 son amigos

220 = 1+2+4+5+10+11+20+22+44+55+110 = 284

284 = 1+2+4+71+142 = 220

**3.** Haz un programa que muestre un contador con 3 dígitos. Mostrará los números del 0-0-0 al 9-9-9, con la particularidad que cada vez que aparezca un 3 lo sustituya por una E.

**4.**  Para obtener el número del tarot de una persona, hay que sumar los números de su fecha de nacimiento y reducirlo a un solo dígito.

Ej: 1 de Julio de 1990 →1+7+1990 = 1998 → 1+9+9+8 = 27 → 2+7=9 El número del tarot es el 9.

Haz un programa que lea la fecha de nacimiento por teclado y escriba el número del tarot. La fecha estará formada por 3 números enteros, el día, el mes y el año (4 dígitos).

## Básicos Hoja 4 *– Nivel Examen*

**1.** Haz un programa en Java que pida números hasta que se teclee uno negativo, y mostrar cuántos números se han introducido.

**2.** Crear un programa para calcular el salario semanal de los empleados a los que se les paga 15 euros por hora si estas no superan las 35 horas. Cada hora por encima de 35 se considerará extra y se paga a 22 €. El programa pide las horas del trabajador y devuelve el salario que se le debe pagar.

Además, el programa debe preguntar si deseamos calcular otro salario, si es así el programa se vuelve a repetir.

**3.** Haz un programa que pida un número n, y diga cuántos y cuáles son los números primos que hay entre 1 y n.

**4.** Haz un programa que vaya pidiendo números, hasta introducir uno negativo, y diga cuál es el mayor número introducido y cuantas veces se repite.

**5.** Haz un programa que pida un número, entre 0 y 10, y escriba un triángulo invertido con dichos números.

Ej: Si n=10, quedaría

0,1,2,3,4,5,6,7,8,9

1,2,3,4,5,6,7,8,9

2,3,4,5,6,7,8,9

3,4,5,6,7,8,9

4,5,6,7,8,9

5,6,7,8,9

6,7,8,9

7,8,9

8,9

9

**6.** Un alumno desea saber que nota necesita en el tercer examen para aprobar una evaluación. El promedio de la evaluación se calcula con la siguiente formula.

NC=(E1+E2+E3)/3

NF =NC\*0.7+NL\*0.3

Donde NC es el promedio de los exámenes, NL el promedio de laboratorio y NF la nota final.

Escribe un programa que pregunte al usuario las notas de los dos primeros exámenes y la nota promedio de laboratorio, y muestre la nota que necesita el alumno en el tercer examen para aprobar la evaluación con nota final 6.0.

*Ejemplo:*

Ingrese nota examen 1: 4.5

Ingrese nota examen 2: 5.5

Ingrese nota laboratorio: 6.5

Necesita nota 7.4 en el examen 3

**7.** La secuencia de Collatz de un número entero se construye de la siguiente forma:

* si el número es par, se divide por dos;
* si es impar, se multiplica tres y se le suma uno;
* la sucesión termina al llegar a uno.

La conjetura de Collatz afirma que, al partir desde cualquier número, la secuencia siempre llegará a 1. A pesar de ser una afirmación a simple vista muy simple, no se ha podido demostrar si es cierta o no.

Usando computadores, se ha verificado que la sucesión efectivamente llega a 1 partiendo desde cualquier número natural menor que 258.

a) Hacer un programa que muestre la secuencia de Collatz de un número entero, que se pedirá por teclado.

Ej:

n: **18**

18 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

n: **19**

19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

n: **20**

20 10 5 16 8 4 2 1

b) Hacer un programa que grafique los largos de las secuencias de Collatz (número de elementos que hay que generar hasta que se llega al 1) de los números enteros positivos menores que el ingresado por el usuario:

n: **20**

1 \*

2 \*\*

3 \*\*\*\*\*\*\*\*

4 \*\*\*

5 \*\*\*\*\*\*

6 \*\*\*\*\*\*\*\*\*

7 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8 \*\*\*\*

9 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10 \*\*\*\*\*\*\*

11 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12 \*\*\*\*\*\*\*\*\*\*

13 \*\*\*\*\*\*\*\*\*\*

14 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

16 \*\*\*\*\*

17 \*\*\*\*\*\*\*\*\*\*\*\*\*

18 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

19 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20 \*\*\*\*\*\*\*\*