

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Tema 3 – Métodos y acceso básico a archivos

Introducción y definición

Es evidente que con lo visto hasta el momento cuando un programa se hace demasiado grande van a aparecer zonas de código repetidas y todo bastante desorganizado. Esto lo hemos visto, por ejemplo, a la hora de pedir datos en un rango. Lo ideal sería poder realizar pequeños fragmentos de código que funcionaran como “nuevas instrucciones” y que pudiéramos ejecutarlas desde distintas partes del programa. Con esto conseguiríamos:

- No repetir código.
- Mantener el código más fácilmente ya que al cambiar/arreglar en un sitio lo cambiamos para todo el programa.
- Organizar el código incluso con la idea de repartir la realización del programa entre un equipo de programadores de forma que cada uno se ocupe de cierta parte lo más independiente posible del resto.
- Reutilizar código. Si hago un “comando nuevo” para un programa puede que me sirva para futuros programas.

Por supuesto que estamos en un lenguaje de POO y todo lo anterior lo vamos a conseguir de una forma muy eficiente con esta filosofía que veremos más adelante, pero la división de nuestro programa en los módulos denominados métodos o funciones conlleva una primera estructuración y organización modular que se ha usado como paradigma durante muchos años.

Una **función o método** es una subrutina (fragmento de código útil y específico) que puede ser invocada (ejecutada) desde distintas partes de un programa y que puede además de realizar cierta tarea, devolver un valor de cualquier tipo de dato.

En los temas pasados ya hemos estado usando funciones que han construido otros programadores. Algunos ejemplos:

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

print(datos): Función dentro de *System.out* que toma el contenido de *datos* y lo imprime en pantalla. Esta función "no devuelve" nada, simplemente **realiza una acción: muestra por pantalla.**

sqrt(numero) : Función dentro de *Math* que **devuelve la raíz cuadrada de un número.** Es decir, que cuando la llamamos se ejecuta cierto código que calcula la raíz de *numero* y luego lo devuelve. Con devolver quiero decir que la propia función es sustituida por el valor que calcula al finalizar. Por ejemplo si tengo el siguiente código:

```
double resultado;
double a=36;
double b=6;

resultado = 2*Math.sqrt(a)+25/b;
System.out.printf("Resultado = %.3f", resultado);
```

la segunda línea comienza ejecutando la función *Math.sqrt(a)* y por tanto se ejecuta un código que calcula la raíz de 36 (ya que ese es el valor de *a*) y devuelve 6 de forma que la línea para el ordenador quedaría (después de sustituir también *b*):

```
resultado = 2*6+25/6;
```

En ambos ejemplos (*print* y *sqrt*) se llama a un código escrito previamente y se ejecuta. La diferencia es que *sqrt* devuelve un valor y por tanto puede formar parte de una expresión sin embargo *print* no devuelve nada, solo realiza acciones.

Devolver un valor significa que tras hacer los cálculos, la función deja el resultado en una zona de memoria que luego puede ser recogida desde la función llamante por ejemplo en una variable:

```
resultado = Math.sqrt(a);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Métodos que no devuelven ningún valor

Es un subprograma que realiza una tarea específica pero no toma ningún valor como resultado de sus operaciones internas. La estructura básica es:

```
[public static] void nombreMetodo([lista de parámetros]) {
    cuerpo del método
}
```

Las partes entre corchetes son opcionales o dependerán de la situación. En nuestro caso, como aún no hemos profundizado en las características de orientación a objetos del lenguaje, es imprescindible que indiquemos todas las funciones con el modificador `static` y opcionalmente con `public`.


La lista de parámetros es opcional y es necesaria para darle datos a la función para poder actuar. Veamos un ejemplo que iremos explicando:

```
public class EjemploFunciones {

    static void estrellas1() {
        System.out.print("*****\n");
    }

    //Método principal
    public static void main(String args[]) {
        estrellas1();
        EjemploFunciones.estrellas1(); //Innecesario dentro de la misma clase.
        System.out.println();
        for (int i = 0; i < 5; i++) {
            estrellas1();
        }
        System.out.println("    Fin ");
    }
}
```

Para llamar a funciones dentro de la misma clase no es necesario poner el nombre de la clase. Si las funciones estuvieran en otra clase sí sería necesario hacerlo y además necesitaríamos anteponerle el modificador *public*.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Actividad: Prueba a crear otra clase (sin public delante) y mete ahí las funciones. También puedes hacer que sea public en otro archivo.

Importante: Las funciones se escriben **dentro de la clase pero fuera del main**. De hecho el main es una función más. Es la función que busca la **jvm** para empezar a ejecutar el programa.

En el ejemplo anterior , al empezar el main y tras la declaración de las variables se ejecuta la función estrellas1(). Esto hace que el código salte a dicha función, se ejecute, y a continuación regresa a seguir ejecutando el main. Como vuelve a haber otra ejecución de estrellas1(), pues vuelve a saltar a dicho código.

El caso de estrellas1() es el más sencillo: una función que no necesita de datos añadidos para trabajar (parámetros) y no devuelve ningún valor.

En este caso cuando se llama a la función **es obligatorio escribir los paréntesis aunque no lleve nada dentro**. Por eso se producen errores de compilación cuando escribías nextInt sin paréntesis, porque nextInt() es una función.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Paso de parámetros

El uso de parámetros es la forma que tenemos para darle información a una función para que pueda realizar tareas con datos externos a la misma. Por ejemplo si hacemos un método que sume dos números, lógicamente deberíamos de poder darle los números a sumar.

Otro ejemplo es una variante de la función `estrellas()`, que llamaremos `estrellas2()` al cual hay que decirle el número de estrellas que ha de pintar.

A una función se le puede pasar como parámetro un literal, una constante, una variable o una expresión.

Amplíemos por ejemplo el programa de estrellas con una segunda función, ahora con un parámetro:

```
public static void estrellas2(int cantidad) {
    for (int i = 0; i < cantidad; i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Nuevamente ten cuidado no meter esta función dentro del `main`, pero sí dentro de la clase.

En el **main** podrías usar `estrellas2` así:

```
Scanner sc = new Scanner(System.in);
int n;
System.out.println();
estrellas2(5);

System.out.println();
System.out.println(" ¿Cuántos asteriscos quieres pintar?");
n = sc.nextInt();
estrellas2(n);
```

Lo que pasa aquí, es que cuando se llama a la función, antes de ejecutarse, el parámetro `cantidad` toma el valor que se le pasa.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Es decir, en el primer caso hace cantidad = 5 y luego se ejecuta la función.

En en el segundo se hace cantidad = n y luego se ejecuta la función. La variable n lógicamente tiene el dato que ha metido al usuario, y mediante el parámetro pasamos ese dato al interior de la función.

Otro ejemplo de función con parámetros:

```
public static void suma(double sumando1, double sumando2) {
    System.out.printf("El resultado de sumar %.2f+%.2f es %.2f\n",
        sumando1, sumando2, sumando1 + sumando2);
}
```

La forma en que funcionan los parámetros es la siguiente: cuando se llama a una función se le da en la posición del parámetro (y en el mismo orden si son varios) un valor o variable. Por ejemplo:

```
suma(4,3);
```

En el momento que se va a ejecutar esa línea, como el 4 está en la posición de *sumando1* este valor es sustituido en toda la función y como 3 está en la posición de *sumando 2* se sustituye en toda la función de la misma manera. Como se explicó para estrellas2 es como si antes de ejecutar la función se ejecutaran las líneas:

```
sumando1 = 4;
sumando2 = 3;
```

Es decir, que al llamar a la función se ejecuta la siguiente línea:

```
System.out.printf("El resultado de sumar %.2f + %.2f es %.2f\n",
    4, 3, 4+3);
```

Es importante resaltar que en el caso de que se le pase una variable **esta no cambiará de valor** a lo largo de la función. Cuando profundicemos en el tema de objetos veremos que existe la posibilidad de que parezca que sí cambia.

Veámoslo:

```
public static void pruebaCambio(int dato){
    System.out.println("Dentro de la función antes del incremento: "+dato);
    dato++; // Es lo mismo que poner dato = dato + 1;
    System.out.println("Dentro de la función después del incremento: "+dato);
}
```

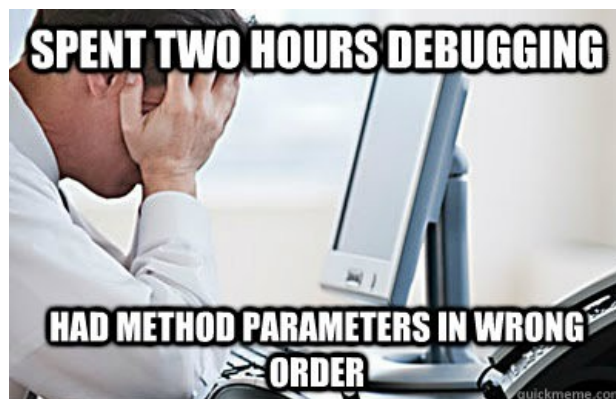
COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Para llamarlo hagamos los siguiente:

```
int valor = 10;

pruebaCambio(valor);
System.out.println("Tras acabar la función: " + valor);
```

Comprueba el resultado.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Métodos que devuelven valores

En los casos anteriores hemos visto funciones que realizan cierto proceso y, a lo sumo, sacan datos y resultados por pantalla, sin embargo habrá muchos casos en los que nos interese que una función tras hacer algún tipo de operación nos deje un valor de resultado para poder seguir trabajando con él. Es lo que se denomina “**valor devuelto por una función**”.

Para que una función devuelva un valor se necesitan dos cambios, por un lado en lugar de usar **void** en la cabecera se indica **el tipo de dato** que se quiere devolver. Además se usará el comando **return** para realizar la devolución. En el momento que se ejecuta return se termina el método aunque se esté dentro de un bucle, un if, o cualquier otra estructura.

Realmente cuando una función llega a return lo que sucede es que deja el resultado en una zona temporal de memoria y el programa principal recoge el dato de esa posición.

Veamos algunos ejemplos:

```
public static int suma(int sumando1, int sumando2) {
    return sumando1 + sumando2;
}
```

```
public static int signo(int numero) {
    if (numero < 0) {
        return -1;
    } else {
        if (numero > 0) {
            return 1;
        }
    }
    return 0; // Puedo hacerlo sin else ¿Por qué?
}
```

¿Por qué crees que esta función suma que devuelve el resultado podría ser mejor que la anterior que además lo muestra?

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

En el programa principal:

```
int a;
int b = 5;
int c;
int d;

a = suma(12, b) * 10;
b = (4 + a) * signo(a - 300);
c = signo(suma(a, b));
d = suma(c, signo(c));
System.out.printf("a:%d\nb:%d\nc:%d\nd:%d\n", a, b, c, d);
```

Prueba en el main anterior a extraer las tres última líneas para comprobar el funcionamiento.

Actividad:

1. Realizar una función que sume $1+2+3+\dots+n$ y devuelva el resultado. **n** será un parámetro. En el programa principal ejecútala **dos** veces para luego mostrar la suma **hasta 10** y **hasta 10000**.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Ámbito de variables

Hasta ahora el único sitio donde hemos declarado variables es dentro de métodos (incluido el método main), sin embargo las variables también se pueden declarar dentro de la clase pero fuera de todas las funciones. Esto hace que la variable pueda ser utilizada por todas las funciones o incluso como "intercambio de información" entre las funciones en lugar de los parámetros.

Tendríamos por tanto dos ámbitos de trabajo para variables:

Variables de clase: declaradas fuera de las funciones, son vistas por todas las funciones definidas en la clase. Hay que anteponerles el modificador **static** (ya entenderemos por qué). **Por ahora no se debe usar esta manera** salvo, si queréis, el Scanner.

Variables locales: declaradas dentro de las funciones y por tanto son visibles sólo dentro de la función en la que fue declarada. Es posible, por tanto, usar los mismos nombres en variables dentro de distintas funciones.

Vimos también que una **variable definida dentro de una estructura** es visible sólo dentro de la estructura (zona entre llaves).

Por ahora **vamos a usar sólo los dos últimos tipos aquí definidos** ya que el primero (static) tiene una visión más amplia que estudiaremos en el tema de POO.

Hay un caso particular en una variable definida dentro de la zona de inicialización del **for**, solo puede ser vista dentro del for (es como si se hubiera declarado dentro de las llaves del for).

Ejemplo de código incorrecto:

```
for(int i=0;i<10;i++){
    System.out.println("i dentro del bucle"+i);
}
System.out.println("i fuera del bucle"+i);
```

Esto nos permite usar la misma variable como contador en distintos bucles for.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

También hay que tener cuidado porque una variable definida dentro de un **do-while** **NO** puede ser usada como parte de la condición del while.

Ejemplo de código incorrecto:

```
do {
    System.out.println("Introduce un n°");
    int num=sc.nextInt();
} while (num<10);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Algunas funciones de interés

Funciones de conversión: Las vistas en el segundo tema.

```
String → byte:      Byte.parseByte(str)
String → double:    Double.parseDouble(str)
String → float:     Float.parseFloat(str)
String → int:       Integer.parseInt(str)
String → long:      Long.parseLong(str)
String → short:     Short.parseShort(str)
```

Funciones matemáticas: Se encuentran en la clase Math. La mayoría las usaremos poco o nada este primer curso pero es interesante conocerlas. Algunas de ellas son:

```
Math.abs(x):        Valor absoluto. Devuelve el mismo tipo que el
                    parámetro.
Math.ceil(x):       Aproxima al entero mayor más cercano. P.ej.
                    Devuelve 5.0 si se le pasa 4.2.
Math.floor(x):      Aproxima al entero menor más cercano. P.ej.
                    Devuelve 4.0 si se le pasa 4.8.
Math.cos(x):        Devuelve el coseno de x (x en radianes).
Math.sin(x):        Devuelve el seno de x (x en radianes).
Math.tan(x):        Devuelve la tangente de x (x en radianes).
Math.sqrt(x):       Raíz cuadrada de x.
Math.pow(x,y):      Devuelve x elevado a y.
Math.max(x,y):      Devuelve el máximo de x e y.
Math.min(x,y):      Devuelve el mínimo de x e y.
```

También existen las constantes Math.PI y Math.E

Números aleatorios: Pertenece a las funciones matemáticas pero vamos a tratarla aparte.

Math.random() Función que devuelve un double entre 0 y 1 (1 no incluido)

Para sacar un número aleatorio entero en cierto rango habría que hacer lo siguiente:

```
(int) (Math.random()*cantidadDeValores+valorMinimo)
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Ejemplos:

Entre 1 y 10 (ambos incluidos): `(int) (Math.random()*10+1)`

Entre 20 y 30 (ambos incluidos): `(int) (Math.random()*11+20)`

Puedes ver otras posibilidades en:

<https://stackoverflow.com/questions/363681/how-do-i-generate-random-integers-within-a-specific-range-in-java>

También como curiosidad, si alguien quiere ver el código fuente de la clase Math puede obtenerlo aquí:

<http://developer.classpath.org/doc/java/lang/Math-source.html>

Algunas de las funciones llaman a VMMath que contiene referencias a funciones "native", es decir, ya compiladas en binario nativo con las librerías java por temas de eficiencia.

Nota: a lo largo de este curso **usaremos solo Math.random()** para la obtención de números aleatorios y no se permite el uso de otras librerías o clases que, si es cierto que facilitan la labor, pero se pierde generalidad pues en cualquier lenguaje de programación va a existir siempre una función que da un número aleatorio entre 0 y 1.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Comentando las funciones

Cuando se desarrollan funciones es necesario informar de la utilidad de dichos subprogramas para lo cual se suele hacer un tipo de comentario estándar justo antes de empezar el método. En la asignatura de Entornos de Desarrollo veremos una herramienta muy potente denominada **javadoc** que permite la automatización de la creación de documentación técnica. Por el momento vamos a comentar las funciones de la siguiente manera.

- Usamos comentario de documentación: **/****
- **Descripción** de lo que hace el método. Siempre lo primero que se escribe después del inicio del comentario. No se pone título pues el propio javadoc se encarga de poner como título la cabecera de la función.
- **@param** por cada parámetro y una descripción si fuera necesaria.
- **@returns** indicando el valor que devuelve.

Ejemplo:

```
/**
 * Resta de valores.
 *
 * @param num1 Minuendo de la resta
 * @param num3 Sustraendo de la resta
 * @return La resta de los parámetros
 */
public static int resta(int num1, int num2) {
    return num1 - num2;
}
```

La principal ventaja de hacer esto a medida que se programa es doble: Por un lado tenemos siempre a nuestra disposición o a disposición del equipo de programación una descripción de los distintos elementos que componen nuestro programa. Por otro lado, la realización del Manual Técnico de cualquier aplicación se realiza muy rápido mediante el programa **javadoc**.

No vamos a pararnos ahora con javadoc pero podemos ver un "preview" del resultado al ponernos encima de la cabecera de la función en vscode, viendo cuál sería el resultado de dicha documentación.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Actividad: Para practicar la creación de funciones haz el boletín básico de métodos. Este no es necesario validarlo pero aprovecha para aclarar todas las dudas que puedas antes de pasar al boletín estándar.

Si acabas este boletín puedes hacer los ejercicios del boletín estándar que no estén marcados con doble asterisco (**) pues esos incluyen uso de archivos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Aún más entrada y salida: Introducción a los archivos de texto y persistencia de datos.

En este apartado vamos a introducir como podemos guardar datos en un archivo de texto y como leerlos. Esto podemos hacerlo porque se realiza de forma muy similar a como se gestiona la consola y, aunque habrá alguna línea que no tengamos aún claro su funcionamiento (como el Scanner), nos permitirá precisamente leer y guardar esos datos para que sigan existiendo aunque nuestro programa deje de ejecutarse o incluso se apague el ordenador.

Escribir datos en un archivo

Durante la ejecución de un programa es muy habitual desear que ciertos datos persistan (sobrevivan) al cierre de la aplicación e incluso al apagado del ordenador. Para ello existen diversos métodos como guardar dichos datos en una base de datos, en la nube o, en el caso más sencillo y que veremos aquí, en un archivo.

Como acercamiento inicial guardaremos datos en un archivo pero en modo texto (es decir, solo caracteres que estén en algún sistema alfanumérico como ASCII o Unicode. Esto incrementa la sencillez pues usaremos los mismos comandos que tenemos para escribir en consola. Veamos esto con un ejemplo:

```
import java.io.PrintWriter;
public class EscrituraArchivos {
    public static void main (String[] a) throws Exception {
        // Apertura
        PrintWriter f = new PrintWriter("prueba.txt");

        // Escritura
        f.print("Prueba de archivos. ");
        f.println("Hola que tal.\n\n");
        f.printf("%5.2f\n", Math.PI);
        for (int i=10; i<=20; i++){
            f.printf("%4d",i);
        }
        // Cierre
        f.close();
    }
}
```


COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Como se indica en los comentarios, el uso de un archivo implica 3 pasos:

- **Apertura del archivo:** Cuando una aplicación necesita usar un archivo, exista o no, tiene que informar al sistema operativo que es el encargado de la gestión de archivos. Al abrir el archivo se realiza precisamente eso: informamos al sistema operativo que vamos a usar determinado archivo y lo bloquea para que solo lo pueda usar mi aplicación (evita conflictos).

Es esta línea:

```
PrintWriter f = new PrintWriter("prueba.txt");
```

En caso de que el archivo a usar no exista, si se tienen los permisos suficientes se crea.

Esta acción además enlaza un elemento externo al programa (el archivo) con un elemento interno al programa (la variable f). De esta manera dentro del programa usaremos f para acceder al archivo.

- **Proceso de datos (escritura):** Esto consta de todas las líneas que acceden al archivo para escribir datos en él. En este caso:

```
f.print("Prueba de archivos. ");
f.println("Hola que tal.\n\n");
f.printf("%5.2f\n", Math.PI);
```

Como puedes observar, los comandos son los mismos que usamos para la consola y además tienen exactamente el mismo comportamiento. Es decir, si el archivo escrito lo lees con el bloc de notas, verás que tiene el mismo aspecto que si hubieras hecho esos print en la consola. Algo así:

```
Prueba de archivos. Hola que tal.
```

```
3,14
10 11 12 13 14 15 16 17 18 19 2
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

- **Cierre:** Una vez que termino de usar el archivo, debe cerrarse. Esta acción indica al sistema operativo que dicho archivo queda libre para ser usado por otras aplicaciones y además, si hubiera datos en memoria que aún no se han volcado (pues el proceso de datos se hace sobre todo en memoria) se guardan realmente en el archivo. El cierre es mediante el comando close:

```
f.close();
```

Para probar esto último prueba a quitar esta línea, ejecuta el programa y mira el archivo prueba.txt. Verás que está vacío por no haberlo cerrado.

Fíjate además que en la cabecera del main o de la función donde esté la gestión del archivo es necesario ampliarlo con la clausula:

```
throws Exception
```

Por ahora hay que hacerlo así, ya entenderemos el motivo en el tema de excepciones.


Leer datos de un archivo

En este caso el proceso es similar a la apertura en cuanto a que consta de tres fases: Apertura proceso y cierre. En este caso se usa la clase Scanner y se pueden usar funciones como nextLine para leer.

Al Scanner se le indica que en vez de usar el teclado va a usar un archivo poniendo **new File("nombrearchivo")** en lugar del **System.in**

En principio es necesario conocer la estructura del archivo para saber lo que leo en cada caso pues en una línea puede haber un string, en otra un double y me puede interesar leer un double, etc. De todas formas por simplicidad puedo leer todo con nextLine y luego hacer conversiones con parseInt o parseDouble.

Veamos un ejemplo que lee el archivo prueba.txt que hemos grabado previamente y muestra cada una de sus líneas por pantalla:

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

```

import java.io.File;
import java.util.Scanner;

public class LecturArchivo {
    public static void main(String[] args) throws Exception {
        String linea;

        // Apertura
        Scanner f = new Scanner(new File("prueba.txt"));

        // Lectura de datos
        linea = f.nextLine(); // Prueba de archivos. Hola que tal.
        System.out.println(linea);

        f.nextLine(); // lee linea vacia
        f.nextLine(); // lee linea vacia

        // Para dejar en consola los dos retornos de carro del archivo
        System.out.println();
        System.out.println();

        linea = f.nextLine(); // 3,14
        System.out.println(linea);

        linea = f.nextLine(); // 10 11 12 13 14 15 16 17 18 19 20
        System.out.println(linea);

        // Cierre
        f.close();
    }
}

```

Fíjate que por cada línea del archivo se hace un `nextLine` para recuperar dicha línea. En este caso simplemente se muestra por pantalla pero podría hacerse otras tareas con los datos leídos.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Otras posibilidades en la lectura

Uso de nextDouble o nextInt:

Al igual que hacemos en consola, existe la posibilidad de usar la lectura con conversión directa, por ejemplo, las líneas que leen el número pi:

```
linea = f.nextLine(); // 3,14
System.out.println(linea);
```

Si fuera necesario usar ese dato como double hay dos posibilidades. La primera ya es conocida, pues es usar un Double.parseDouble(). La segunda, en el fondo también es conocida pues es usar el nextDouble pero en vez de con el usuario, con lo leído en el archivo, así sustituiríamos las líneas anteriores por estas:

```
double num = f.nextDouble(); // 3,14
f.nextLine(); // Recoge retorno de carro
System.out.printf("Dato: %.1f\n", num);
```

El nextLine suelto es como en la consola, para recoger el retorno de carro que queda tras el n.º double.

Leer en bucle (hasNext):

En ocasiones queremos leer un archivo todo de golpe sin saber el tamaño o estructura del mismo. Para ello la función hasNext nos devuelve true si hay algún dato a continuación o false si ha llegado al final del archivo.

Esto se puede usar para leer de golpe un archivo en muy pocas líneas (Fíjate que en el caso anterior repetíamos bastante código), haciendo que el bucle continúe mientras haya datos en el archivo.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

El siguiente ejemplo lee un archivo y lo muestra por pantalla.

```
// Apertura
Scanner f = new Scanner(new File("prueba.txt"));
String linea;

// Lectura de datos en bucle
while (f.hasNext()) {
    linea = f.nextLine();
    System.out.println(linea);
}

// Cierre
f.close();
```

Existen otras posibilidades pero por el momento nos quedamos con los métodos vistos.



<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Apéndice I: Recursividad.

Se define la recursividad como una técnica en la que una función se llama a si misma. El caso más simple de recursividad es el siguiente:

```
public static void infinito(){
    infinito();
}
```

Si lo pruebas llamándola desde un programa principal saltará un stack overflow debido a que se está llamando a si misma continuamente hasta que se llena la memoria ya que en cada llamada tiene que guardar la dirección de memoria de retorno.

Por tanto una función recursiva debe tener dos partes: una que sí es recursiva pero debe tener también una condición de salida para que termine en un momento dado.

Veamos un ejemplo con la serie de Fibonacci que es una sucesión de números definida de forma recursiva (https://es.wikipedia.org/wiki/Sucesión_de_Fibonacci):

```
Fib(1)=1
Fib(2)=1
Fib(3)= Fib(1)+ Fib(2)=2
Fib(4)= Fib(2)+ Fib(3)=3
```

Sería: 1,1,2,3,5,8,13,21,34,55, ...

La definiríamos de forma recursiva así:

1. Salida: si $n=1$ o $n=2$ (Fibonacci(n)=1)
2. Parte recursiva: Fibonacci(n)=Fibonacci($n-1$)+Fibonacci($n-2$)

```
public static int fibonacci(int n){
    if (n==1 || n==2){
        return 1;
    }
    return fibonacci(n-1)+fibonacci(n-2);
}
```

La recursividad no es siempre imprescindible pero para ciertas tareas facilita mucho la labor. Juegos como el buscaminas u operaciones como un factorial o ciertas sucesiones son un claro ejemplo.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice II: Añadir datos a un archivo (append).

Si lo que se desea es que al abrir un archivo no se sobrescriban los datos si no que queremos añadir más datos al final hay que realizar lo que se denomina un **"append"**, es decir, abrir el archivo para **añadir datos**.

Para ello la apertura es un poco más compleja. Sería algo así:

```
PrintWriter f = new PrintWriter(new FileWriter("prueba.txt", true));
```

Puedes probarlo con este código:

```
PrintWriter f = new PrintWriter(new FileWriter("prueba.txt", true));
f.println("Añadimos texto al final");
f.close();
```

En este caso un booleano que indica si **añade** datos (**true**) o **sobreescribe** (**false**).

Por supuesto también se podría hacer leyendo el archivo de forma clásica metiéndolo todo en un string y luego guardando en el archivo ese string con el nuevo texto al final.