Argentina programa 4.0

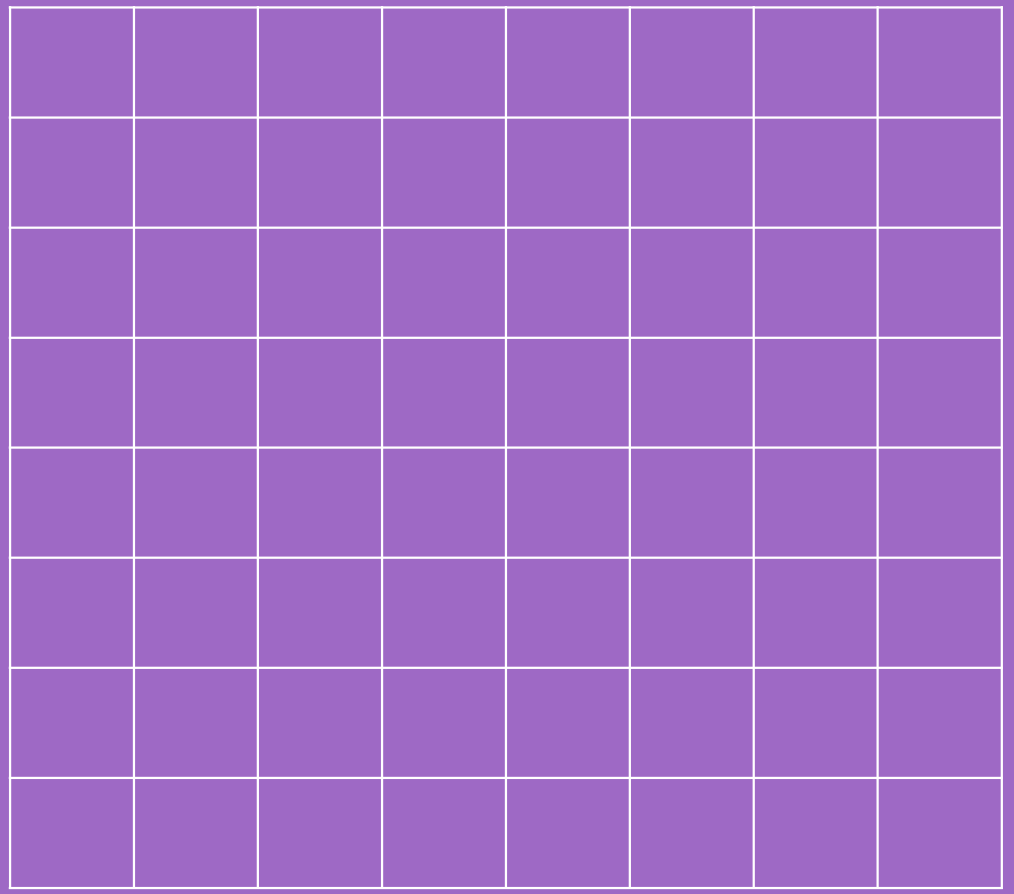
# Programas y Archivos

"Desarrollador Java Inicial"

# Agenda


- Funciones
  - Declaración
  - Usos
- Archivos
  - Sistema de archivos
  - Formatos de Archivo
  - Lectura y Escritura
  - Charsets
- Entradas en un programa
  - Argumentos
  - Lectura de la consola

# Funciones / Métodos



# Métodos en Java, funciones y procedimientos

Argentina programa 4.0

- 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 requerimos con frecuencia y que además se pueden 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).

# Métodos en Java, funciones y procedimientos


- Para estar claros en todo, 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:

# ¿Funciones, métodos o procedimientos?

Argentina programa 4.0

Es muy común entre programadores que se hable indistintamente de estos tres términos sin embargo poseen deferencias 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 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 retorna un valor usando la instrucción return, si no retorna algo, entonces no es una función. En java las funciones usan el modificador *static*.

# Funciones

Argentina programa 4.0

Ejemplo de una función que no devuelve valor:

```
1
2  void imprimirMensaje() {
3      System.out.println("Hola mundo");
4  }
5
```

Ejemplo de una función que devuelve un valor:

```
1
2  int sumar(int a, int b) {
3      return a + b;
4  }
5
```

Definir el tipo de dato de retorno:Si la función no devuelve ningún valor, usa la palabra clave “void”. Si devuelve un valor,por ejemplo, int, double, String, etc.

Escribir el nombre de la función

Escribir los parámetros: Si la función requiere algún parámetro, debes escribirlos entre paréntesis.

```
1
2  void imprimirMensaje(String mensaje) {
3      System.out.println(mensaje);
4  }
5
```

•Escribir el cuerpo de la función: El cuerpo de la función va entre llaves {} y contiene las instrucciones que se ejecutan cuando se llama a la función.

# ¿Funciones, métodos o procedimientos?

Argentina programa 4.0

- ## Métodos:

- Los métodos y las funciones en Java están en capacidad de 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 en 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:*** *Es aquí donde digo que 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 esta debe ser* static*, para que no requiera de un objeto para ser llamada.*


Tipo de retorno (tipo de dato)
Nombre del método
Modificador de acceso
Bloque de código
Lista de parámetros
1
2 class Ejemplo {
3 public int suma(int a, int b) {
4 return a + b;
5 }
6 }
7

# ¿Funciones, métodos o procedimientos?

Argentina programa 4.0

- ## Procedimientos:

- Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo en la definición no hay nada que se lo impida.

- 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.

# Ejemplos 1 de métodos

```
int metodoEntero()//Función sin parámetros
{
   int suma = 5+5;
   return suma;          //Acá termina la ejecución del método
                          //return 5+5;//Este return nunca se ejecutará
                          //Intenta intercambiar la línea 3 con la 5
                         //int x = 10; //Esta línea nunca se ejecutará
}
```

- Como puedes ver es un ejemplo sencillo, es un método llamado metodoEntero, si ejecutas esto, la función te retornará el valor de suma que es 10 (5+5). Las líneas posteriores no se ejecutarán nunca, aunque no generan error alguno, no tienen utilidad. Puedes notar que para este caso es lo mismo haber escrito return suma que escribir return 5+5. Ambas líneas funcionan equivalentemente.

# Ejemplos 2 de métodos

Argentina programa 4.0

```
public String metodoString(int n)//método con un parámetro
{
   if(n == 0)//Usamos el parámetro en la función
   {
      return "a"; //Si n es cero retorna a
      //Notar que de aquí para abajo no se ejecuta nada más
   }
   return "x";//Este return sólo se ejecuta cuando n NO es cero
}
```

Aquí creamos un método público, hicimos uso de múltiples sentencia return y aprovechamos la característica de que al ser ejecutadas finalizan inmediatamente la ejecución de la parte restante del método. De este modo podemos asegurar que la función retornará "a" únicamente cuando el valor del parámetro n sea cero y retornará un "x" cuando dicho valor no sea cero.

# Ejemplos 3 de métodos

Argentina programa 4.0

```
static boolean metodoBoolean(boolean n, String mensaje)//Método con dos parámetros
{
   if(n)//Usamos el parámetro en el método
   {
      System.out.println(mensaje);//Mostramos el mensaje
   }
   return n; //Usamos el parámetro como valor a retornar
}
```

Aquí ya tenemos una función (digo función y no método porque es static) que recibe dos parámetros, uno de ellos es usado en el condicional y el otro para mostrar su valor por pantalla con *System.out.println*, esta vez retornamos valores booleanos true o false y utilizamos el valor propio recibido en el parámetro. Toma en cuenta que en esta ocasión únicamente usamos una sentencia return, pues usar una al interior del if habría sido innecesario y el resultado sería el mismo.

# Hablemos un poco de los procedimientos

Argentina programa 4.0

- Los procedimientos son similares a las funciones, aunque más resumidos. Debido a que los procedimientos no retornan valores, no hacen uso de la sentencia return para devolver valores y no tienen tipo específico, sólo *void*. Veamos un ejemplo:

```
void procedimiento(int n, String nombre) //Notar el void
{
   if(n > 0 && !nombre.equals(""))//usamos los dos parámetros
   {
      System.out.println("hola " + nombre);
      return; //Si no ponemos este return se mostraría hola y luego adiós
   }
   //También podríamos usar un else en vez del return
   System.out.println("adios");
}
```

# Hablemos un poco de los procedimientos


- De este ejemplo podemos ver que ya no se usa un tipo sino que se pone void, indicando que no retorna valores, también podemos ver que un procedimiento también puede recibir parámetros o argumentos.
- **Recuerda:** Los procedimientos también pueden usar la sentencia return, pero no con un valor. En los procedimientos el *return* sólo se utiliza para finalizar allí la ejecución.

# Invocando funciones y procedimientos en Java


- Ya hemos visto cómo hacer funciones en Java, cómo se crean y cómo se ejecutan, ahora veamos cómo usar un método, función o procedimiento.

**nombre([valor,[valor]...]);**

- Como puedes notar es bastante sencillo invocar o llamar funciones en Java, sólo necesitas el nombre del método, función o procedimiento y enviarle el valor de los parámetros. Hay que hacer algunas salvedades respecto a esto.

# Detalles para invocar métodos funciones y procedimientos


- No importa si se trata de un método en Java o de una función o de un método, sólo debes ocuparte de enviar los parámetros de la forma correcta para invocarlos.
- El nombre debe coincidir exactamente al momento de invocar, pues es la única forma de identificarlo.
- El orden de los parámetros y el tipo debe coincidir. Hay que ser cuidadosos al momento de enviar los parámetros, debemos hacerlo en el mismo orden en el que fueron declarados y deben ser del mismo tipo (número, texto u otros).
- Cada parámetro enviado también va separado por comas.

# Detalles para invocar métodos funciones y procedimientos


- Si una función no recibe parámetros, simplemente no ponemos nada al interior de los paréntesis, pero SIEMPRE debemos poner los paréntesis.
- Invocar una función sigue siendo una sentencia común y corriente en Java, así que ésta debe finalizar con ';' como siempre.
- El valor retornado por un método o función puede ser asignado a una variable del mismo tipo, pero no podemos hacer esto con un procedimiento, pues no retornan valor alguno.
- Una función puede llamar a otra dentro de sí misma o incluso puede ser enviada como parámetro a otra (mira el siguiente ejemplo).

# Ejemplos de uso de funciones

Argentina programa 4.0

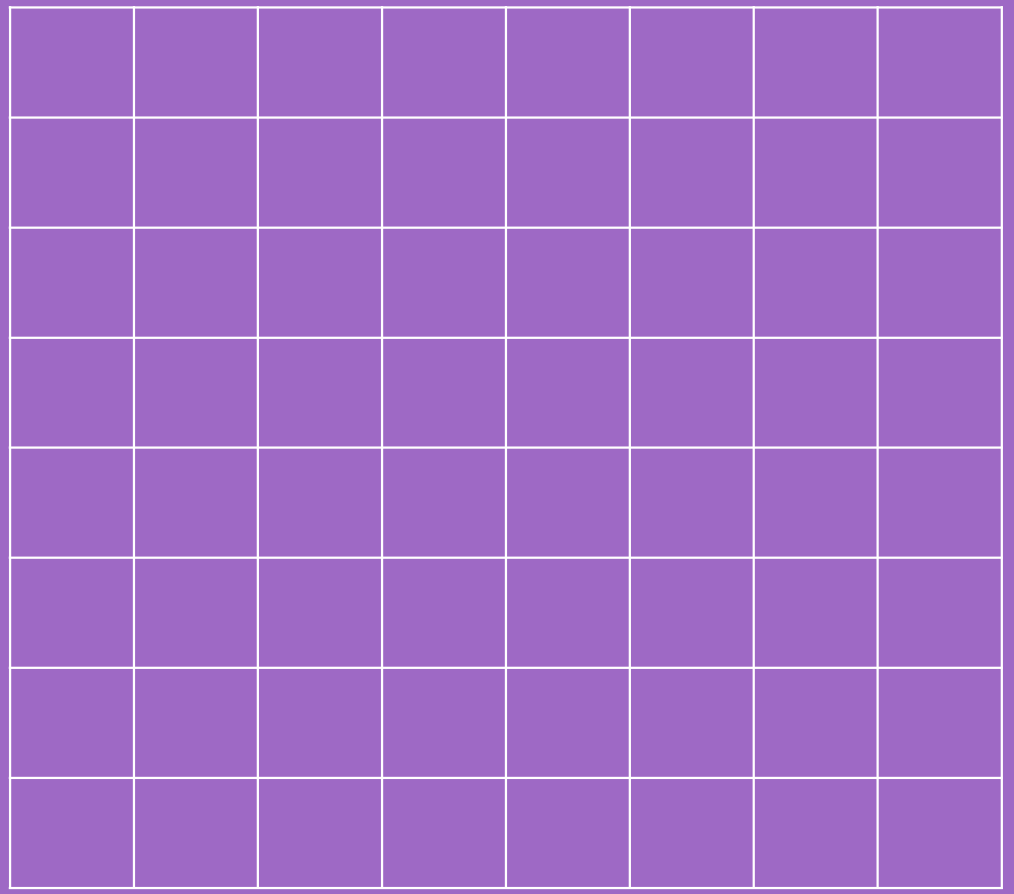
```java

class MiClase {
    // Función que imprime el doble de un número
    void imprimirDoble(int num) {
        // Calcula el doble del número pasado como parámetro
        int resultado = num * 2;
        // Imprime el resultado en la consola
        System.out.println("El doble de " + num + " es " + resultado);
    }
}

public static void main(String[] args) {
    MiClase objeto = new MiClase();
    // Llama a la función pasando el número 5 como parámetro
    objeto.imprimirDoble(5);
}

```

# Ejemplos de uso de funciones



- En el código anterior podemos ver cómo todo ha sido invocado al interior la función main (la función principal), esto nos demuestra que podemos hacer uso de funciones al interior de otras. También vemos cómo se asigna el valor retornado por el método a la variable 'respuesta' y finalmente, antes del return, vemos cómo hemos usado el valor retornado por 'funcionBool' como parámetro del procedimiento.

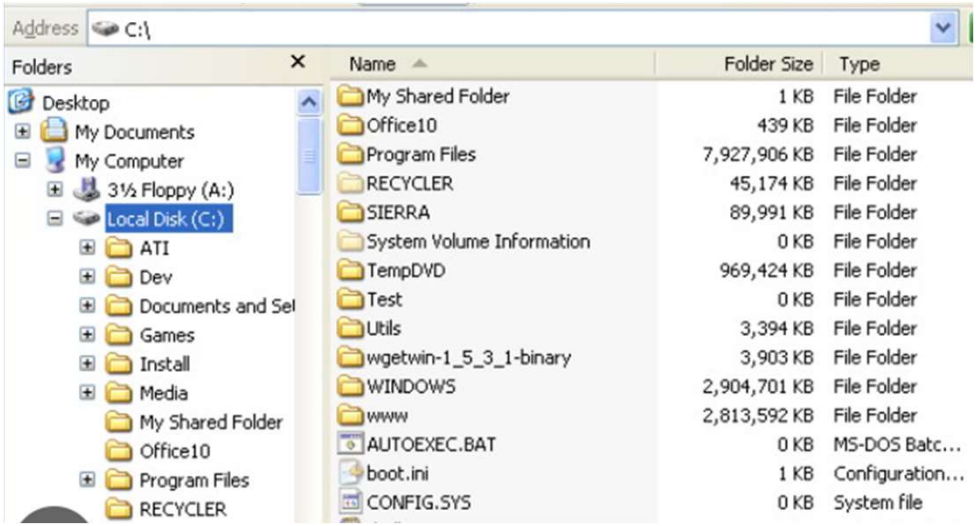
# Sistema de Archivos



# Sistema de Archivos

En general, la información duradera dentro de una computadora, se almacena en una estructura de archivos (files) y directorios o carpetas(folders). Las carpetas contienen archivos y otras carpetas, mientras que los archivos contienen información, cuya interpretación depende de su formato ( por ejemplo texto, video, sonido, pdf, programa ejecutable, etc.)

En el Sistema Operativo Windows, el padre de todas las carpetas es la unidad de almacenamiento o disco rígido en uso, por ejemplo C: , D: etc...




También en Windows, en general, el formato del archivo viene acompañado de una extensión, por ejemplo pdf, mp3, .ext, .png, etc.
Por ejemplo, el código “fuente” de java se almacena en archivos .java y como veremos, su versión “compilada” se almacena en archivos .class.

# Sistema de Archivos

Cuando se quiere acceder a un archivo o directorio, debe conocerse su “ruta”, es decir dentro de qué directorios está lo que queremos buscar. En general dicha ruta se expresa con un String y el carácter “/” (dividido) se usa para separar los directorios y archivos (en Windows también se puede usar el carácter “\” contrabarra, pero como vimos en la unidad anterior, al ser el carácter de escape, hay que utilizarlo 2 veces para que se represente a sí mismo).

Por ejemplo:

C:\\Users\\JuanPablo\\unArchivo.txt

Establece que: dentro del disco rígido que tenemos configurado como **C:** , en el directorio **Users**, se encuentra el directorio **JuanPablo** y dentro de ese, esta el archivo de texto, **unArchivo.txt**

# Archivos - Operaciones - Utilidades

Argentina programa 4.0

Por su importancia, prácticamente todos los lenguajes de programación poseen la forma de acceder al sistema de archivos, en el caso de Java, en esta unidad hablaremos de las clases “Files” y “Path”, nos enfocaremos sobre los archivos de texto y su lectura secuencial.

Nuevamente, si bien no llegamos a ver el concepto de clase, los vamos a tratar como utilidades para cumplir nuestros objetivos.

Primero que nada, antes de usar estas clases, es necesario importarlas, para que sean más fáciles de referenciar, para eso, antes de la declaración de nuestra clase principal, vamos a importarlas:

```
import java.nio.file.Files;

import java.nio.file.Path;

public class MiPrograma {

    public static void main(String[] args) {

    //...
```

# Clase File

La clase File además de proporcionarnos información sobre los archivos y directorios nos permite crearlos y eliminarlos.

Para ello esta clase nos permite crearlos utilizando:
**File nombreFile = new File("/carpeta/archivo");**

y borrarlos con:
**nombreFile.delete();**

# Además de los anteriores disponemos de estos métodos para gestionarlos:

Argentina programa 4.0

| Método | Descripción |
|---|---|
| *createNewFile()* | Crea (si se puede) el fichero indicado |
| *delete()* | Borra el fichero indicado |
| *mkdirs()* | Crea el directorio indicado |
| *getName()* | Devuelve un *String* con el nombre del fichero |
| *getPath()* | Devuelve un *String* con la ruta relativa |
| *getAbsolutePath()* | Devuelve un *String* con la ruta absoluta |
| *getParent()* | Devuelve un *String* con el directorio que tiene por encima |
| *renameTo()* | Renombra un fichero al nombre del fichero pasado como parámetro (se puede mover el fichero resultante a otra ruta, en caso de ser la misma se sobrescribirá) |
| *exists()* | *Boolean* que nos indicará si el fichero existe |
| *canWrite()* | *Boolean* que nos indicará si el fichero puede ser escrito |
| *canRead()* | *Boolean* que nos indicará si el fichero puede ser leído |
| *isFile()* | *Boolean* que indica si el fichero es un archivo |
| *listFiles()* | Método que devuelve un *array* con los ficheros contenidos en el directorio indicado |
| *isDirectory()* | *Boolean* que indica si el fichero es un directorio |
| *lastModified()* | Devuelve la última hora de modificación del archivo |
| *length()* | Devuelve la longitud del archivo |

# Archivos - Operaciones - Lectura

Argentina programa 4.0

Lectura de las líneas de un archivo

```
Files.readAllLines(Paths.get("C:\\Users\\JuanPablo\\unArchivo.txt")
```

- De esto hay varios comentarios para hacer

1) `Paths.get("...")` prepara la ruta ingresada para que la pueda aceptar el método readAllLines

2) readAllLines recibe la ruta generada por Paths.get y retorna una lista de Strings. Aún no vimos ese tipo de dato, pero sabemos que puede ser iterado. Por ejemplo:

```
String archivo = "C:\\Users\\JuanPablo\\unArchivo.txt";
for (String linea : Files.readAllLines(Paths.get(archivo))){
    System.out.println(linea);
}
```

1) Esta forma es útil siempre y cuando el archivo no sea muy grande (unos pocos megas)

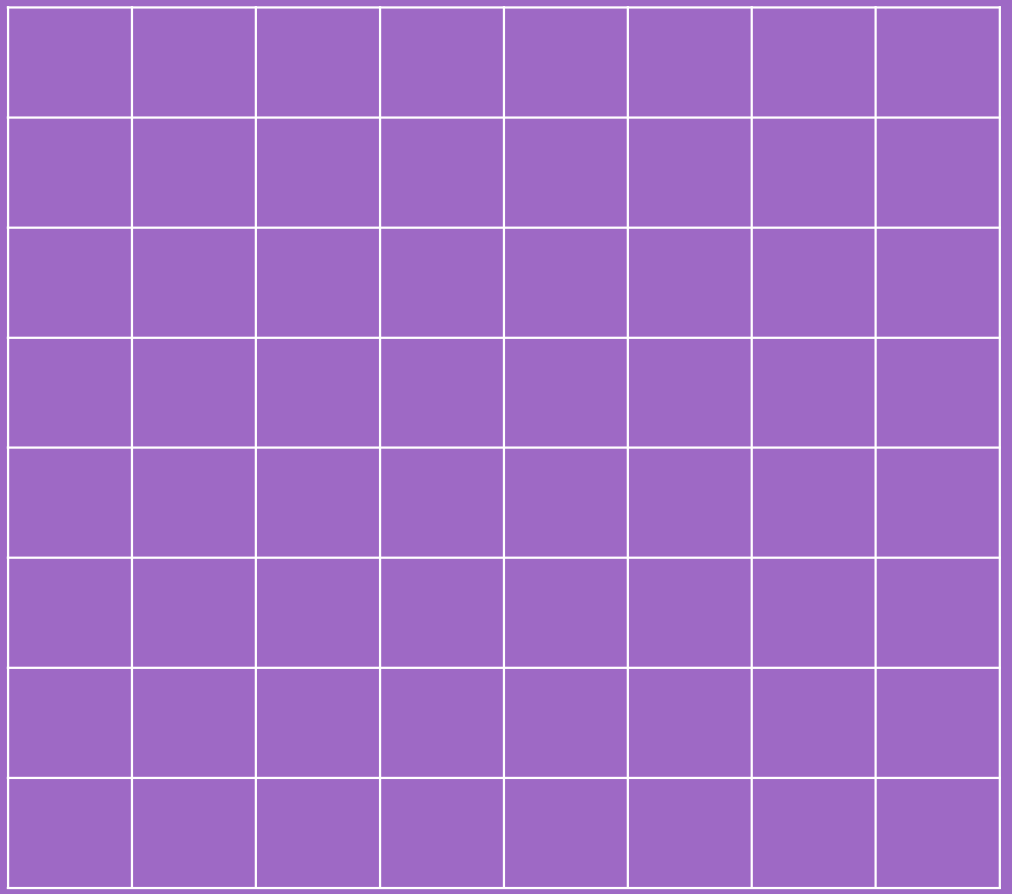
# Archivos - Operaciones - Escritura

Lectura de las líneas de un archivo

```
Files.writeString(Paths.get("/home/eze/nuevoArch"), "hola\n que tal?\n");
```

`writeString` recibe la ruta generada y como segundo parámetro un String con contenido. Nuevamente, este método es efectivo, cuando el String a escribir no es muy grande.

# Entradas de un programa



# Parámetros de comando

Argentina programa 4.0

Cuando se inicia un programa cualquiera, al mismo se le pueden pasar parámetros. En el caso de Java, si al programa se la pasan, este los leerá desde la variable **args** en **main**. Los parámetros pueden ser cualquier string y en cualquier cantidad.

```
public class Sumatoria {
    public static void main(String[] args) {
        int resultado = 0;
        for (int i = 0; i < args.length; i++) {
            int numero = Integer.parseInt(args[i]);
            resultado = resultado + numero;
        }
        System.out.println(resultado);
    }
}
```

# Leer de consola

```java
import java.util.Scanner;

public class Ejercicio1 {

	public static void main(String[] args) {
		Scanner scn = new Scanner(System.in);
		System.out.println(
			"Ingrese números separados por UN
		espacio");
		String numeros = scn.nextLine();
		int resultado = 0;
		for (String numeroStr : numeros.split(" ")) {
			int numero = Integer.parseInt(numeroStr);
			resultado = resultado + numero;
		}
		System.out.println(resultado);
	}
}
```

Otra forma de ingresar datos a un programa es mediante la consola, en este ejemplo el programa nos pedirá que escribamos algo y apretemos la tecla “enter” para terminar

# Referencias



- https://www.baeldung.com/java-nio-2-file-api
- https://www.marcobehler.com/guides/java-files
- http://www.javalearningacademy.com/streams-in-java-concepts-and-usage/

Argentina
programa
4.0

# Gracias!