



Programas y Archivos

“Desarrollador Java Inicial”

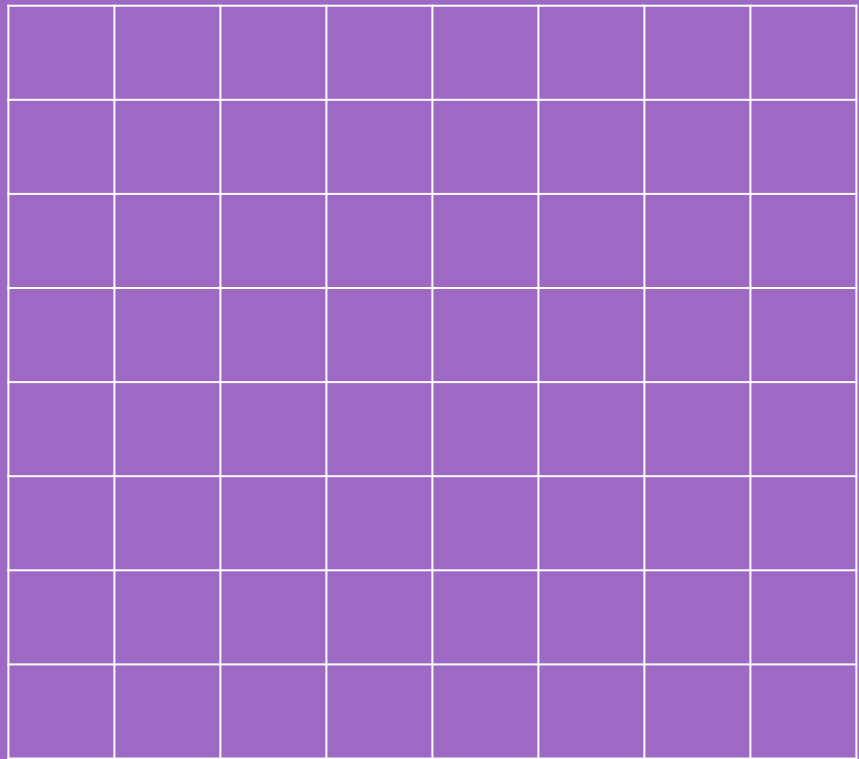
Agenda



- 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



Mirando una función principal o “main”

Cuando desarrollamos, es muy común que necesitemos reutilizar parte de nuestros programas. Por ejemplo, en el ejercicio de la guía: “*dado un vector de números, y un número X, que sume todos los números >X y retorne el resultado*”, probablemente planteamos algo así:

```
public class Sumatoria {  
    public static void main(String[] args) {  
        int numeros[] = new int[] { 1, 37, 16 };  
        int resultado = 0;  
        for (int i = 0; i < numeros.length; i++) {  
            resultado = resultado + numeros[i]  
        }  
        System.out.println(resultado);  
    }  
}
```

Por un lado, si quiero cambiar lo que debo calcular, tengo que modificar la variable números, por el otro, el resultado se imprime por pantalla y se “pierde” al final del programa.

Primer función reutilizable

Ahora, vamos a suponer que quiero calcular un promedio.
Una forma de hacerlo es volver a escribir el "main":

```
public class Promedio {  
    public static void main(String[] args) {  
        int numeros[] = new int[] { 1, 37, 16 };  
        int resultado = 0;  
        for (int i = 0; i < numeros.length; i++) {  
            resultado = resultado + numeros[i];  
        }  
        resultado = resultado/numeros.length;  
        System.out.println( resultado );  
    }  
}
```

Pero, suponiendo que no quiero perder el trabajo que ya tengo, conceptualmente sería usar el código que use para sumar y dividirlo por su largo

```
public class Promedio {  
    public static void main(String[] args) {  
        int numeros[] = new int[] { 1, 37, 16 };  
        int resultado = sumatoria(numeros);  
        resultado = resultado/numeros.length;  
        System.out.println( resultado );  
    }  
}
```

La pregunta es como representamos esa "sumatoria(numeros)", para utilizarla en nuestro programa de promedios.

Primer función reutilizable

Entonces queda algo así:

```
public class Promedio {  
    public static void main(String[] args) {  
        int numeros[] = new int[] { 1, 37, 16  
    };  
        int resultado = sumatoria(numeros);  
        resultado = resultado /  
        numeros.length;  
        System.out.println(resultado);  
    }  
  
    private static int sumatoria(int[]  
numerosASumar)  
    {  
        int suma = 0;  
        for (int i = 0; i <  
numerosASumar.length; i++) {  
            suma = suma + i;  
        }  
        return suma;  
    }  
}
```

Vemos entonces la función *sumatoria*, que recibe la variable **números** (puede llamarse con cualquier valor) y retorna un valor en la variable **suma** que se almacena en la variable "resultado" dentro de **main**.

Las entradas de las funciones se denominan parámetros, en el caso de **sumatoria**, el único es **numerosASumar**. Las funciones pueden tener ninguno o más parámetros.

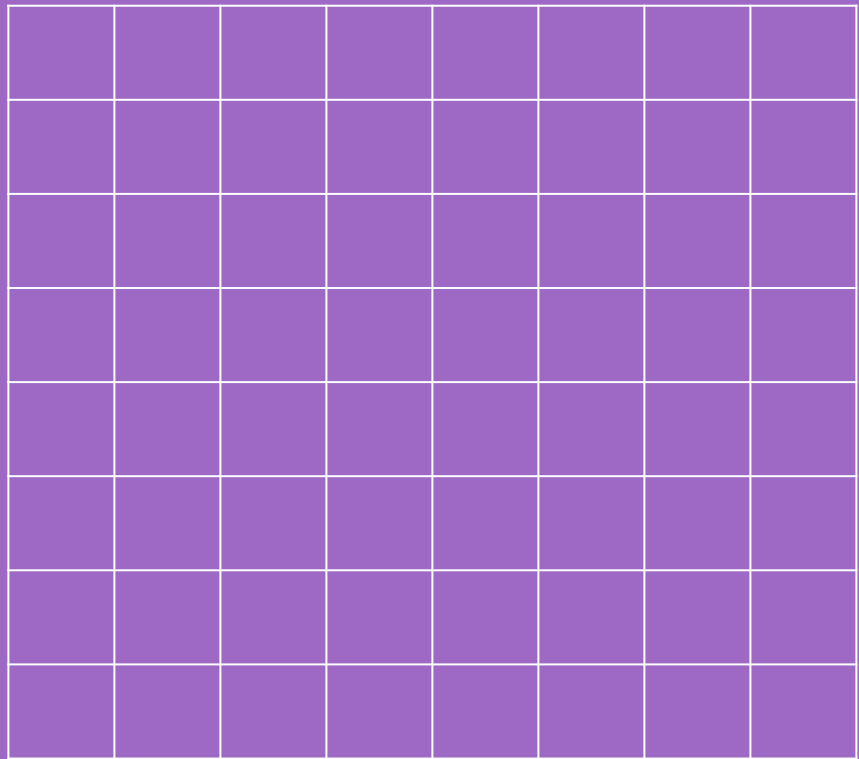
Aclaraciones importantes



- En realidad, Java no tiene funciones, sino "métodos", ya que están asociados a un objeto o clase, pero recién la próxima clase podremos apreciar la diferencia.
- Ya venimos usando Métodos, por ejemplo:
 - `println : System.out.println("un texto")`
 - `split: "hola que tal?".split(" ")`



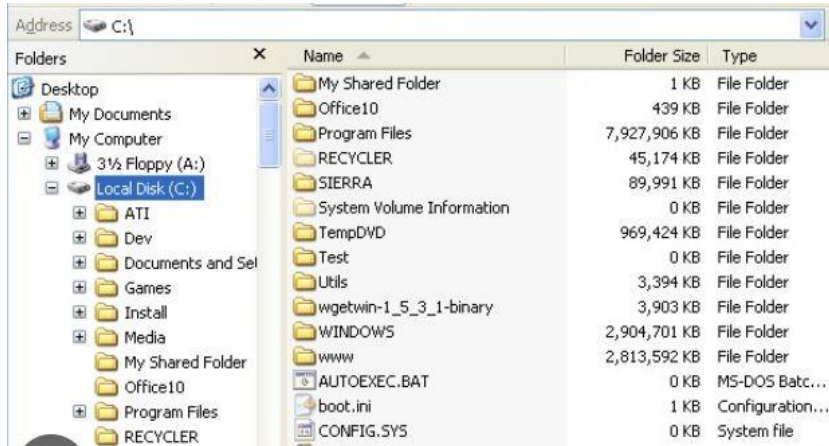
Sistema de Archivos



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

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) {

        //...
```

El código mostrado en esta PPT funciona con la versión 17 de la JDK

Archivos - Operaciones - Lectura

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);  
}
```

- 3) 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.

Manipulación de archivos, creación e información extra



```
Path unArchivo = Paths.get("../recursos/algo.txt");  
System.out.println(unArchivo.toAbsolutePath());
```

```
FileTime lastModifiedTime = Files.getLastModifiedTime(unArchivo);  
System.out.println(lastModifiedTime);
```

```
Path otroArch = Paths.get("../recursos2/algoQueNoExiste.txt");  
System.out.println(Files.exists(otroArch));  
Files.createFile(otroArch);  
System.out.println(Files.exists(otroArch));
```

```
Path unTemporal = Files.createTempFile("unPrefijo", ".txt");  
System.out.println(unTemporal);
```

```
System.out.println( Files.isDirectory(unTemporal) );  
Files.copy(arch1, arch2);  
Files.move(arch1, arch3);
```

<https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/nio/files/Paths.html>

Encoding

```
Path utfFile = Files.createTempFile("some", ".txt");

Files.write(utfFile, "un texto \n con eñes ".getBytes()); / UTF 8

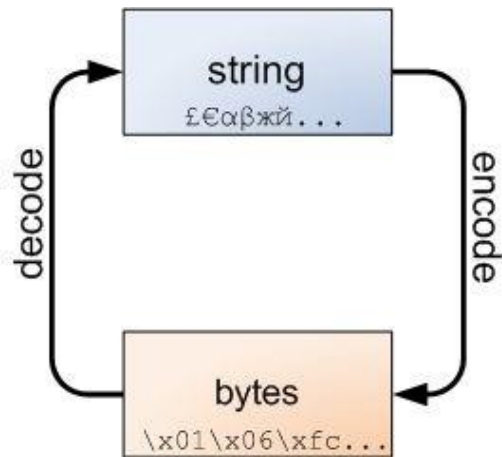
Files.write(utfFile, (System.lineSeparator() +
"y acentos á ").getBytes(),StandardOpenOption.APPEND);

System.out.println(Files.readAllLines(utfFile) .size());

for (String line: Files.readAllLines(utfFile)) {
    System.out.println(line);
}

Path iso88591File = Files.createTempFile("some", ".txt");

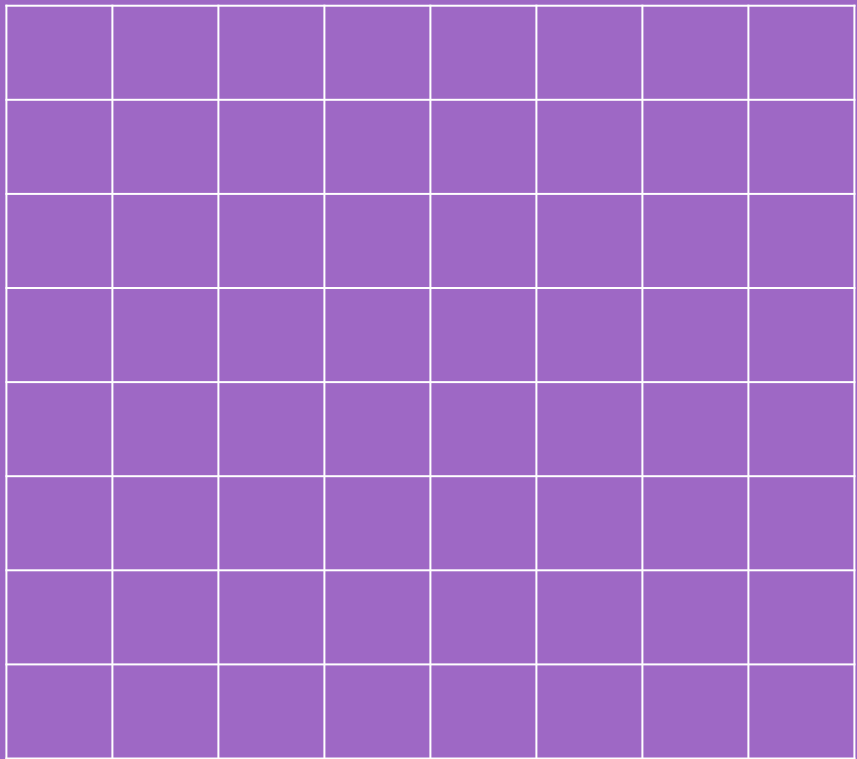
Files.write(iso88591File, "otro texto con eñes".getBytes(StandardCharsets.ISO_8859_1));
```



`StandardOpenOption` y `StandardCharsets` son Enums o valores enumerados, por ahora solo diremos que son fijos y no se pueden cambiar.



Entradas de un programa



Parámetros de comando

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



```
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 numeroSt : numeros.split(" ")) {
            int número = Integer.parseInt(numeroSt);
            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!
