

Sesión 01: En marcha (*Deberías* saber hacerlo)

v0.6

Hoja de problemas Programación 2

Ángel Herranz
aherranz@fi.upm.es

Universidad Politécnica de Madrid

2022-2023

Esta hoja de ejercicios tiene dos objetivos. Por un lado exponer algunos ejemplos de programas que el profesor espera que tú puedas resolver. Por otro lado, que te sirvan de recordatorio de lo que se presupone que sabes sobre programación. Si no sabes hacerlos, no pasa nada, lo aprenderás ;), no dejes de preguntar lo que no sepas.

Nota: Quizás ya te hayas dado cuenta de que en las transparencias y en las hojas de ejercicios de vez en cuando aparecen algunos iconos. Aquí tienes un pequeño diccionario:

 leer, convenciones

 peligro, atención

 buscar en internet o libros

 éxito

 programar

 fracaso

 pensar, observar

 en casa

 recordar

 **Ejercicio 1.** Antes de empezar con la hoja vamos a ver qué **herramientas de desarrollo** vas a necesitar:

- Un **kit de desarrollo** de Java, o **JDK** (*Java Development Kit*).
- Un **editor de texto plano** como pueden ser Atom, Sublime o Notepad++ (hay muchos otros).
- Un **terminal** como puede ser Cmd (“símbolo de sistema” en Windows) o gnome-terminal (“terminal” en Linux).

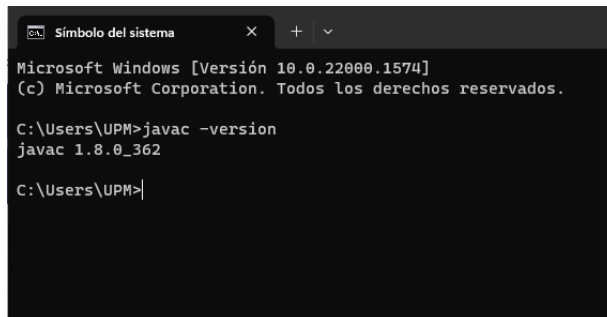
Para instalar un JDK te recomendamos usar la distribución de OpenJDK que puedes descargar desde <https://adoptium.net>.

 **Ejercicio 2.** Para asegurarnos de que todo está funcionando correctamente después de la

instalación, abre el terminal y en la “consola” ejecuta la orden

```
javac -version
```

Deberías estar viendo una pantalla parecida a esta:



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22000.1574]
(c) Microsoft Corporation. Todos los derechos reservados.

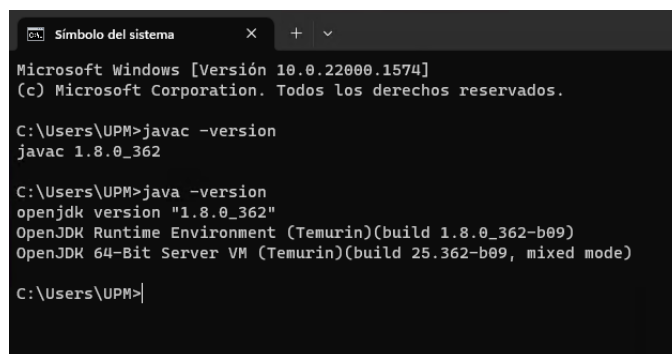
C:\Users\UPM>javac -version
javac 1.8.0_362

C:\Users\UPM>|
```

Eso ya nos indica que tienes el **compilador de Java** correctamente instalado. A continuación vamos a probar que puedes **ejecutar los programas** que escribas, en el mismo terminal ejecuta la orden:

```
java -version
```

Deberías estar viendo una pantalla parecida a esta:



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22000.1574]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\UPM>javac -version
javac 1.8.0_362

C:\Users\UPM>java -version
openjdk version "1.8.0_362"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_362-b09)
OpenJDK 64-Bit Server VM (Temurin)(build 25.362-b09, mixed mode)

C:\Users\UPM>|
```

Eso nos indica que tienes una **entorno de ejecución de Java** correctamente instalado. Asegúrate de comprobar que las versiones son las correctas (nunca por debajo de 1.8).

📄 **Hola Ejercicio 3.** Para terminar de comprobar que todo está correcto vamos a escribir un programa “hola mundo”. Tendrás que crear un fichero llamado `Hola.java` con tu editor de texto plano. El programa es el siguiente:

```
public class Hola {
    public static void main(String[] args) {
        System.out.println("Hola mundo");
    }
}
```

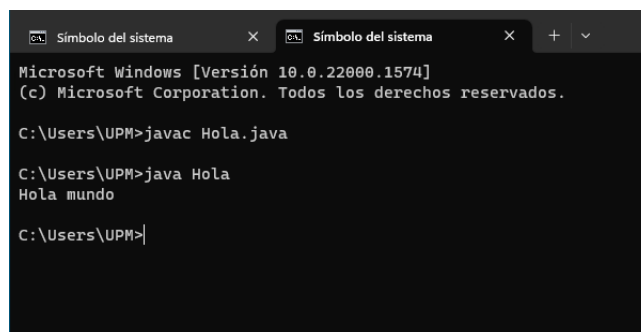
Guarda el fichero `Hola.java` en una carpeta (**directorio**) para todos los programas de esta hoja, **abre el terminal en ese directorio**¹ y ejecuta la orden de compilación:

```
javac Hola.java
```

y luego la orden de ejecución del programa `Hola`:

```
java Hola
```

Deberías ver una pantalla parecida a esta:



```
Microsoft Windows [Versión 10.0.22000.1574]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\UPM>javac Hola.java

C:\Users\UPM>java Hola
Hola mundo

C:\Users\UPM>|
```

Ejercicio 4. En casi todos los ejercicios de esta hoja te vamos a pedir que crees un programa con el que te comunicarás usando las siguientes *fuentes*:

- **Argumentos** de entrada (los escribes en línea de comandos al ejecutar el programa).
- Lectura de datos de la **entrada estándar** (el teclado).
- Escritura de datos en la **salida estándar** (la pantalla).

Para poder abordar todos estos ejercicios tendrás que atender y entender la primera clase donde ya se explica cómo usar la línea de comandos.

Con respecto a la salida estándar, creo que con el *método* `println` tienes más que suficiente para empezar. En todo caso ya el ejercicio 6 trata un poco más de la salida estándar.

Con respecto a la entrada estándar tendrás que esperar al ejercicio 8.

Ejercicio 5. Los lenguajes de programación siempre vienen acompañados de una **biblioteca estándar**, es decir, un conjunto de datos y funciones ya incorporadas en el propio lenguaje. En el caso de Java dicha biblioteca se denomina **API de la edición estándar**, o en inglés **Java Platform, Standard Edition API**. Siempre tienes que tener a mano la documentación de la biblioteca estándar del lenguaje. En nuestro caso:

<https://docs.oracle.com/javase/8/docs/api/index.html>

args y **status**

Ejercicio 6. Antes de empezar a hacer programas más elaborados vamos a usar dos recursos fundamentales en programación: argumentos del programa y estado de salida de un programa.

¹El concepto de **ejecutar en un directorio** es un concepto fundamental. Puedes abrir el terminal en ese directorio (en Windows, por ejemplo, desde el explorador, en una carpeta, puedes abrir un terminal `Cmd`). Otra opción es abrir el terminal y **moverte hasta ese directorio** usando el mandato `cd`.

Imagina que quieres hacer un programa que justo al ejecutarlo puedas pasarle *datos*. Imagina que quieres implementar un programa que calcule el factorial de cualquier número que le pases. Imagina que puedes ejecutar

```
java Factorial 5
```

y que tu programa contesta por la salida estándar con

```
120
```

Todos los lenguajes de programación te ofrecen una forma de leer ese 5 (el argumento de entrada del programa `Factorial`).

¿Qué te parece si te digo que ese 5 vas a poder leerlo del índice 0 del array `args` que es el argumento del `main`? Vamos a probarlo, escribe el programa `Factorial.java`:

```
public class Factorial {  
    public static void main(String[] args) {  
        String argumento = args[0];  
        int resultado = 1;  
  
        POR HACER: convertir argumento a entero  
  
        POR HACER: calcular el factorial en resultado  
  
        System.out.println(resultado);  
    }  
}
```

En tus manos: completa la implementación para calcular el factorial, compila y ejecuta tal y como se ha muestra más arriba (`java Factorial 5`).

 **Ejercicio 7.** La *salida estándar* se refiere a la forma en que se muestran los datos o resultados generados por un programa en la consola o terminal de tu ordenador. En Java, se puede imprimir en la salida estándar (consola) utilizando el objeto `System.out`, que representa la salida estándar.

Seguro que ya conoces el método `println` así que te sugerimos que explores otros métodos que puedes usar sobre la salida estándar y en particular te sugerimos que explores el método `printf` en la documentación de la biblioteca estándar (ver ejercicio 5).

Veamos un ejemplo con el que puedes practicar. Supongamos que tenemos las siguientes variables:

```
String nombre = "Luisa";  
int edad = 25;  
double altura = 1.75;
```

Podemos utilizar `System.out.printf()` para imprimir estas variables en un formato específico. Por ejemplo:

```
System.out.printf("Mi nombre es %s, edad %d años, altura %.2f metros.\n",  
                 nombre, edad, altura);
```

Observa con cuidado como se usan ciertas **etiquetas de formato** para indicar cómo queremos que se impriman las variables. La etiqueta %s indica una cadena de caracteres, %d indica un número entero y %.2f indica un número decimal con dos decimales. Finalmente la *etiqueta* \n indica un cambio de línea. Luego, pasamos las variables correspondientes como argumentos en el orden en que aparecen en la cadena de formato. Mucho más sencillo que hacerlo con println, ¿no crees?

Puedes consultar todas las etiquetas en el manual de la biblioteca estándar, en concreto en <https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html>.

📖 **Scanner** **Ejercicio 8.** La salida estándar se refiere a una forma en que un programa recibe datos de entrada desde una fuente externa. En general la entrada estándar está de algún modo *conectada al teclado*: tú escribes en el teclado y tu programa puede leer lo que hayas escrito. En Java, la entrada estándar se puede leer desde la consola utilizando el objeto `System.in`, que representa la entrada estándar.

Lamentablemente, para poder leer datos de la entrada estándar en Java, se necesita utilizar una clase extra que muy habitualmente es la clase `Scanner`. Los objetos de la clase `Scanner` permiten leer diferentes tipos de datos desde la entrada estándar. Veamos un ejemplo:

```
import java.util.Scanner;

public class Hola {
    public static void main(String[] args) {
        String nombre, apellido;
        Scanner entrada = new Scanner(System.in);
        System.out.printf("Introduce tu nombre: ");
        nombre = entrada.nextLine();
        System.out.printf("¡Hola, %s! encantado.\n", nombre);
        System.out.printf("Introduce tu apellido: ");
        apellido = entrada.next();
        System.out.printf("Hola, %s %s, ya nos conocemos mejor.\n", nombre, apellido);
    }
}
```

Quizás la línea más importante en ese código sea la que dice

```
nombre = entrada.nextLine();
```

Esa línea provoca que el *scanner* entrada lea una línea de la entrada estándar. El segundo uso del *scanner*

```
apellido = entrada.next();
```

provoca que se lea la **siguiente** línea del teclado.

Transcribe el programa y ejecútalo para probar.

🔍 **Ejercicio 9.** Busca e intenta entender la documentación sobre la clase `Scanner` en la documentación de la biblioteca estándar (ver ejercicio 5)

📖 **Redirect** **Ejercicio 10.** Un recurso muy interesante es que el sistema operativo te permite cambiar

fácilmente la entrada estándar para que en lugar de que sea el teclado sea directamente un fichero (a este recurso se le llama **redirección de la entrada estándar**). Vamos a ver un ejemplo muy ilustrativo.

Crea un fichero de texto plano `datos.txt` con tu nombre y tu apellido, por ejemplo:

```
Luisa
Amigo
```

Y ahora ejecuta tu programa `Hola` del ejercicio anterior (ejercicio 8) usando la siguiente orden en la línea de comandos:

```
java Hola < datos.txt
```

Podrás observar que tu programa ha leído los datos desde el fichero en vez de desde el teclado. Esto se debe a que al ejecutar, el símbolo `<` le dice al sistema operativo que *redirija* la entrada estándar desde un fichero.

Q Ejercicio 11. Cualquier programador debería conocer algunos algoritmos *míticos*. Entre ellos se encuentra el algoritmo de Euclides para calcular el máximo común divisor. Busca en internet información al respecto del algoritmo.

 **Euclides** **Ejercicio 12.** Llegó la hora de comprobar si eres capaz de escribir un programa que deberías ser capaz de escribir ;). Si no eres capaz, no te preocupes, quizás sea un buen punto de partida para preguntar a tu profesor.

Escribe un programa `MCD.java` que lea dos números desde la línea de comandos y escriba en la salida estándar (pantalla) el máximo común divisor de los mismos. Por ejemplo, tu programar debería imprimir en pantalla 5 si la invocación del mismo es

```
java MCD 15 25
```

Q String **Ejercicio 13.** Como en casi todos los lenguajes de programación Java ofrece una construcción para representar cadenas de caracteres. En el caso de Java es el tipo `String`.

Como este recurso es tan importante, merece la pena que explores qué operaciones sobre *strings* te regala la biblioteca estándar. Pero antes de mostrarte la URL que tú misma deberías explorar voy a mostrarte una tabla con, posiblemente, los métodos más útiles. Suponiendo que s_1 y s_2 son instancias de la clase `String`, i entero, las operaciones más importantes son:

Uso	Resultado	Significado
"cualquier cosa"	String	el propio <i>string</i> "cualquier cosa"
new String(<i>s</i> ₁)	String	el resultado es una nueva instancia de String que es igual a <i>s</i> ₁ caracter a caracter
<i>s</i> ₁ .length()	int	el resultado es la longitud de <i>s</i> ₁
<i>s</i> ₁ .charAt(<i>i</i>)	char	el resultado es el caracter que ocupa la índice <i>i</i> en <i>s</i> ₁ (precondición: <i>i</i> mayor o igual que 0 y menor que la longitud de <i>s</i> ₁)
<i>s</i> ₁ .codePointAt(<i>i</i>)	int	el resultado es el caracter Unicode que ocupa el índice <i>i</i> en <i>s</i> ₁ (precondición: <i>i</i> mayor o igual que 0 y menor que la longitud de <i>s</i> ₁) Nota: los caracteres unicode pueden ocupar más de 8 bits, que es lo que ocupa un char , por eso devuelve un int , puedes hacer pruebas con un <i>string</i> que tenga hacentos com Herranz, Ángel :)
<i>s</i> ₁ .compareTo(<i>s</i> ₂)	int	compara <i>s</i> ₁ y <i>s</i> ₂ lexicográficamente: devuelve un número menor que 0 si <i>s</i> ₁ va antes (lexicográficamente) que <i>s</i> ₂ , 0 si <i>s</i> ₁ y <i>s</i> ₂ son iguales y un número mayor que 0 si <i>s</i> ₂ va antes (lexicográficamente) que <i>s</i> ₁
<i>s</i> ₁ .concat(<i>s</i> ₂)	String	el resultado es una nueva instancia de String que representa la concatenación de los dos <i>strings</i> <i>s</i> ₁ y <i>s</i> ₂
<i>s</i> ₁ .contains(<i>s</i> ₂)	boolean	el resultado es true si <i>s</i> ₁ es una subcadena de <i>s</i> ₂ y false en otro caso
<i>s</i> ₁ .equals(<i>s</i> ₂)	boolean	el resultado es true si <i>s</i> ₁ y <i>s</i> ₂ son iguales y false en otro caso
<i>s</i> ₁ == <i>s</i> ₂	boolean	el resultado es true si <i>s</i> ₁ y <i>s</i> ₂ son la misma instancia y false en otro caso
<i>s</i> ₁ .indexOf(<i>c</i>)	int	el resultado es el índice que ocupa el caracter <i>c</i> en el <i>string</i> <i>s</i> ₁ , -1 si el caracter no aparece en <i>s</i> ₁
<i>s</i> ₁ .indexOf(<i>s</i> ₂)	int	el resultado es el índice en el que empieza el <i>string</i> <i>s</i> ₂ en el <i>string</i> <i>s</i> ₁ , -1 si <i>s</i> ₂ no está en <i>s</i> ₁
<i>s</i> ₁ .isEmpty()	boolean	el resultado es true si el <i>string</i> <i>s</i> ₁ es igual al <i>string</i> "", false en otro caso
<i>s</i> ₁ .substring(<i>i</i> , <i>j</i>)	String	el resultado es una nueva instancia de String que representa la subcadena entre los índices <i>i</i> y <i>j</i> (precondición: <i>i</i> , <i>j</i> mayores o igual que 0 y menores que la longitud de <i>s</i> ₁ y <i>i</i> menor o igual que <i>j</i>)
<i>s</i> ₁ .trim()	String	el resultado es una nueva instancia de String que es idéntica a <i>s</i> ₁ tras eliminar posibles espacios al principio y al final

¿Quieres saber más de String? No dejes de visitar la documentación:

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

💬 Propiedades de String

Ejercicio 14. A continuación te dejo algunas propiedades sobre los *strings*. Asume que todas las propiedades están escritas con un para todo sobre s_1 y s_2 y asumiendo que s_1 y s_2 son distintos de **null**:

$$\begin{aligned}s_1 == s_2 &\Rightarrow s_1.equals(s_2) \\ s_1.equals(s_2) &\not\Rightarrow s_1 == s_2 \\ s_1.compareTo(s_2) == 0 &\Leftrightarrow s_1.equals(s_2) \\ s_1.length() == 0 &\Leftrightarrow s_1.isEmpty() \\ s_1.concat(s_2).length() &== s_1.length() + s_2.length() \\ s_1.contains(s_2) &\Leftrightarrow \exists i, j : s_1.substring(i, j).equals(s_2) \\ s_1.trim().length() &\leq s_1.length()\end{aligned}$$

¿Se te ocurre a ti misma alguna propiedad nueva?

📄 Grep

Ejercicio 15. Tu programa tiene que admitir un argumento en la línea de comandos. Ese argumento es una palabra que tendrás que buscar en la entrada estándar. Cada vez que esa palabra aparezca en una línea de la entrada estándar tendrás que imprimir dicha línea.

Llama a tu programa `Grep.java`². Crea un fichero de texto plano con algunas líneas como por ejemplo este poema de Lorca:

Yo quiero que el agua se quede sin cauce.
Yo quiero que el viento se quede sin valles.

Quiero que la noche se quede sin ojos
y mi corazón sin la flor del oro.

Que los bueyes hablen con las grandes hojas
y que la lombriz se muera de sombra.

Que brillen los dientes de la calavera
y los amarillos inunden la seda.

Puedo ver el duelo de la noche herida
luchando enroscada con el mediodía.

Resisto un ocaso de verde veneno
y los arcos rotos donde sufre el tiempo.

Pero no me ensenes tu limpio desnudo
como un negro cactus abierto en los juncos.

Déjame en un ansia de oscuros planetas,
ipero no me ensenes tu cintura fresca!

Voy a asumir que el fichero se llama `gacela.txt`.

Tu programa debería ser capaz de imprimir las líneas que tengan la palabra “*los*” usando la siguiente invocación:

²*Grep* es una utilidad de la línea de comandos escrita originalmente para ser usada con el sistema operativo Unix, para más información consulta <https://es.wikipedia.org/wiki/Grep>.

```
java Grep los < gacela.txt
```

-  **Lc** **Ejercicio 16.** Escribe un programa `Lc.java` que lea líneas de la entrada estándar e imprima un número que indique el número de líneas leídas (`Lc = line count`). Prueba el programa con el fichero `gacela.txt` del ejercicio 15. Así:

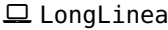
```
java Lc < gacela.txt
```

En la salida estándar deberías ver el número 23 (número de líneas del fichero `gacela.txt`).

-  **Wc** **Ejercicio 17.** Escribe un programa `Wc.java`³ que lea líneas de la entrada estándar e imprima un número que indique el número de palabras leídas (`Wc = word count`). Prueba el programa con el fichero `gacela.txt` del ejercicio 15. Así:

```
java Wc < gacela.txt
```

En la salida estándar deberías ver un número cercano al 119 (número de palabras del fichero `gacela.txt`).

-  **LongLinea** **Ejercicio 18.** Escribe un programa `LongLinea.java` que lea líneas de la entrada estándar e imprima sus longitudes, una por línea en la salida estándar. Prueba el programa con el fichero `gacela.txt` del ejercicio 15.

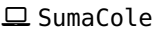
-  **LongLinea** **Ejercicio 19.** Escribe un programa `LongLinea.java` que lea líneas de la entrada estándar e imprima sus longitudes, una por línea en la salida estándar. Prueba el programa con el fichero `gacela.txt` del ejercicio 15.

-  **NumLinea** **Ejercicio 20.** Escribe un programa `NumLinea.java` que lea líneas de la entrada estándar y las imprima en la salida estándar pero colocando delante el número de línea (empezando en 1). Prueba el programa con el fichero `gacela.txt` del ejercicio 15. Quizás veas que es un poco feo el cambio entre la línea 9 y la 10.

-  **Ejercicio 21.** En el ejercicio anterior (ejercicio 20) habrás observado que es un poco feo el cambio entre la línea 9 y la 10. Quizás puedas solucionarlo haciendo que los números de línea ocupen los primeros 5 caracteres de la línea y alineando los números hasta esa *columna*. El resultado debería ser algo como esto:

```
...
    7 Que los bueyes hablen con las grandes hojas
    8 y que la lombriz se muera de sombra.
    9
   10 Que brillen los dientes de la calavera
   11 y los amarillos inunden la seda.
...
```

-  **Ejercicio 22.** Siguiendo con el ejercicio anterior (ejercicio 21). ¿Cómo resolverías el problema si el número de líneas puede superar las 99999 y no tienes un límite conocido?

-  **SumaCole** **Ejercicio 23.** Escribe un programa `SumaCole.java` que lea dos números naturales de la

³ `Wc` es una utilidad de la línea de comandos escrita originalmente para ser usada con el sistema operativo Unix, para más información consulta [https://es.wikipedia.org/wiki/Wc_\(Unix\)](https://es.wikipedia.org/wiki/Wc_(Unix)).

entrada estándar e imprima su suma usando como algoritmo el algoritmo para sumar que nos enseñaron en el cole. A modo de recordatorio:

$$\begin{array}{r}
 (1) \\
 3 2 1 5 \\
 + 8 3 7 9 \\
 \hline
 1 1 5 9 4
 \end{array}$$

 **Ejercicio 24.** Usando como punto de partida el ejercicio anterior (ejercicio 23), escribe un programa `DibujaSuma.java` que dibuje la suma de la misma forma en la que lo hacíamos en el cole. Es decir, para una entrada como la siguiente:

```
3215
8379
```

deberá mostrar una salida como esta:

```

      1
    3 2 1 5
+ 8 3 7 9
-----
1 1 5 9 4

```

respetando los espaciados que se pueden ver.

 **Ejercicio 25.** Escribe un programa `GenRand.java` que genera números enteros aleatorios entre -2^{31} y $2^{31} - 1$. El programa recibe como argumentos el número de números aleatorios a generar e imprime cada uno en una línea.

Por ejemplo, ejecutando tu programa así

```
java GenRand 5
```

la salida podría ser algo como esto:

```
32539
-18498
6671
27011
-14641
```

Para poder realizar este programa puedes ver cómo generar números aleatorios en Java consultando el manual: <https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html#random->.

 **Ejercicio 26.** En el ejercicio 6 realizaste un programa que calcula el factorial. Quizás tu implementación era iterativa (usando un bucle) o recursiva. El reto ahora es hacer lo contrario de lo que hicieras entonces.

 **Overflow** **Ejercicio 27.** En el ejercicio 26 (da igual si en su versión iterativa o recursiva) realizaste un programa que calcula el factorial. Pero ¿Hasta dónde llega? ¿En qué punto empieza a fallar? ¿Por qué? ¿Se te ocurre alguna solución?

 **Ejercicio 28.** Escribe una función que reciba como entrada un array de enteros y lo invierta. La *interfaz* o la *signatura* o el *tipo* de dicha función será algo parecido a

```
public static void reverse(int[] a)
```

Podrás probar tu código con la siguiente prueba (test):

```
int[] ejemplo = new int[] {2,5,4,6,4};
int[] esperado = new int[] {4,6,4,5,2};
reverse(ejemplo);
for (int i = 0; i < ejemplo.length; i++) {
    if (ejemplo[i] != esperado[i]) {
        System.err.println("Error en función reverse");
        System.exit(1);
    }
}
```

- 📄 reverse(2) **Ejercicio 29.** Escribe una versión nueva de la función reverse para que esta vez devuelva un nuevo array sin tocar el array de entrada. El tipo de la función será el siguiente:

```
public static int[] reverse(int[] a)
```

Podrás probar tu código con la siguiente prueba, algo más complicada que en la versión anterior:

```
int[] ejemplo = new int[] {2,5,4,6,4};
int[] esperado = new int[] {4,6,4,5,2};
int[] resultado;
resultado = reverse(ejemplo);
if (ejemplo.length != esperado.length || resultado.length != esperado.length) {
    System.err.println("Error en función reverse: las longitudes no coinciden");
    System.exit(1);
}
for (int i = 0; i < ejemplo.length; i++) {
    if (ejemplo[i] != esperado[ejemplo.length - i - 1]) {
        System.err.println("Error en función reverse: no debe modificarse el array ejemplo");
        System.exit(1);
    }
    if (resultado[i] != esperado[i]) {
        System.err.println("Error en función reverse");
        System.exit(1);
    }
}
```

- 📄 rev(3) **Ejercicio 30.** Escribe una función que reciba un String como argumento y devuelva un nuevo *string* que sea la inversión del *string* de entrada. El tipo de la función será el siguiente:

```
public static String rev(String s)
```

Intenta construir tú misma una prueba inspirándote en la prueba del ejercicio 29.

- 📄 Rev **Ejercicio 31.** Tendrás que escribir un programa que lea líneas de la entrada estándar y las imprima en el mismo orden pero invertidas en la salida estándar. Llama a tu programa `Rev.java`⁴. Pruébalo con el fichero `gacela.txt` del ejercicio 15. Si lo ejecutas de esta forma:

⁴Rev es una utilidad de Unix que invierte las líneas de un fichero.

```
java Rev < gacela.txt
```

La salida estándar debería ser lo siguiente:

```
.ecuac nis edeuq es auga le euq oreiuq oY
.sellav nis edeuq es otneiv le euq oreiuq oY
```

```
sojo nis edeuq es ehcon al euq oreiuQ
.oro led rolf al nis nózaroc im y
```

```
sajoh sednarg sal noc nelbah seyeub sol euQ
.arbmos ed areum es zirbmol al euq y
```

```
arevalac al ed setneid sol nellirb euQ
.ades al nednuni sollirama sol y
```

```
adireh ehcon al ed oleud le rev odeuP
.aídoidem le noc adacsorne odnahcul
```

```
onenev edrev ed osaco nu otsiseR
.opmeit le erfus ednod sotor socra sol y
```

```
odunsed oipmil ut senesne em on oreP
.socnuj sol ne otreiba sutcac orgen nu omoc
```

```
,satenalp sorucso ed aisna nu ne emajéD
!acserf arutnic ut senesne em on orepi
```

 **Primo** **Ejercicio 32.** Escribe un programa `Primo.java` que recibe un número como argumento en la línea de comandos y determina si es primo o no. Pruebas probar tu programa con la siguiente línea de comandos⁵:

```
java Primo 2147483647
```

 **Factores** **Ejercicio 33.** Escribe un programa `Factores.java` que recibe un número como argumento en la línea de comandos e imprime todos sus factores primos en la salida estándar (uno por línea). Se muestran a continuación tres ejemplos de ejecución y su salida espera:

<pre>java Factores 50</pre>	<pre>java Factores 8932</pre>	<pre>java Factores 2147483647</pre>
<pre>2</pre>	<pre>2</pre>	<pre>2147483647</pre>
<pre>5</pre>	<pre>7</pre>	
	<pre>11</pre>	
	<pre>29</pre>	

 **Sobre factorización** **Ejercicio 34.** Te habrás dado cuenta de que factorizar números es bastante costoso en tér-

⁵El número 2147483647 es el máximo valor para un entero con signo en máquinas de 32 bits.

menos de tiempo de ejecución. Seguro que tu misma te has dado cuenta, o quizás recuerdas de haber estudiado, que no es necesario buscar factores mayores que $\sqrt{n}$ (¡por que no los hay!), donde n es el número a factorizar.

El problema de la factorización de números naturales es fundamental en muchas áreas de las matemáticas y la informática. Por eso hay muchos lagorismos que te animo a explorar. Algunos nombres de algoritmos importantes par factorizar son:

- Algoritmo de Dixon
- Factorización con fracciones continuas
- Criba cuadrática
- Criba racional
- Algoritmo general de criba del cuerpo de números
- Factorización de formas cuadradas de Shanks

 Dibuja Factores

Ejercicio 35. ¿Recuerdas la *forma* que tenía la descomposición en factores primos en el cole? Por ejemplo, para el número 72:

```

72 | 2
36 | 2
18 | 2
 9 | 3
 3 | 3
 1 |
72 = 23 × 32

```

La idea es que implementes un programa `DibujaFactores.java` que reciba un número natural a través de la línea de comandos y que ofrezca resultados de esta forma:

```
java DibujaFactores 72
```

```

72 | 2
36 | 2
18 | 2
 9 | 3
 3 | 3
 1 |

```

```
72 = 2^3 x 3^2
```

 MultCole

Ejercicio 36. Escribe un programa `MultCole.java` que lea dos números naturales de la entrada estándar e imprima su multiplicación usando como algorimo el algoritmo para multiplicación que nos enseñaron en el cole.

 DibujaMult

Ejercicio 37. Usando como punto de partida el ejercicio anterior (ejercicio 36), escribe

un programa `DibujaMult.java` que dibuje la multiplicación de la misma forma en la que lo hacíamos en el cole.

📄 **DivCole** **Ejercicio 38.** Escribe un programa `DivCole.java` que lea dos números naturales de la entrada estándar e imprima su división usando como algoritmo el algoritmo para división que nos enseñaron en el cole.

📄 **DibujaDiv** **Ejercicio 39.** Usando como punto de partida el ejercicio anterior (ejercicio 38), escribe un programa `DibujaDiv.java` que dibuje la división de la misma forma en la que lo hacíamos en el cole.

📄 **Ultima** **Ejercicio 40.** En el ejercicio 8 viste cómo se pueden leer datos de la entrada estándar usando la clase `Scanner`. Esa misma clase puede usarse para leer datos de cualquier fichero de texto. Por ejemplo, supongamos que queremos leer la última línea de un fichero que se llama `palabras.txt`. Primero tendremos que *abrir el fichero*:

```
File fichero = new File("palabras.txt");
```

y entonces tendremos que crear un *Scanner conectado* a dicho fichero:

```
Scanner entrada = new Scanner(fichero);
```

y a partir de ese momento cada ejecución del método `entrada.nextLine()` devolverá un *String* con la línea leída desde el fichero `palabras.txt`. El siguiente código imprimirá la última línea de un fichero:

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

```
public class Ultima {
    public static void main(String[] args) {
        File fichero = new File("palabras.txt");
        Scanner entrada = new Scanner(fichero);
        String ultima, aux;
        aux = entrada.nextLine();
        while (aux != null) {
            ultima = aux;
            aux = entrada.nextLine();
        }
        System.out.printf(ultima);
    }
}
```

Compila y prueba el programa `Ultima.java`.

📄 **Palindromo** **Ejercicio 41.** Escribe un programa `Palindromo.java` que recibe una palabra como argumento en la línea de comandos y decide si es palíndromo o no. Se muestran a continuación dos ejemplos de ejecución y su salida esperada:

```
java Palindromo Aibofobia
```

```
java Palindromo listo
```

```
"Aibofobia" es palíndromo
```

```
"listo" no es palíndromo
```



Ejercicio 42. Escribe un programa `Mayusculas.java` que convierta una cadena de caracteres en mayúsculas. El programa debe recibir palabras como argumentos en la línea de comandos y mostrar la versión en mayúsculas de todas las palabras. Se muestran a continuación dos ejemplos de ejecución y su salida esperada:

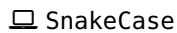
```
java Mayusculas Hola, mundo!
```

```
java Mayusculas Lorem ipsum dolor sit amet.
```

```
HOLA, MUNDO!
```

```
LOREM IPSUM DOLOR SIT AMET.
```

Para convertir una cadena a mayúsculas en Java, puedes utilizar el método `toUpperCase()` de la clase `String`. Asegúrate de probar tu programa con varias cadenas y verificar que el resultado sea correcto.



Ejercicio 43. Escribe un programa `SnakeCase.java` que transforme un identificador dado a *snake case* (ver https://es.wikipedia.org/wiki/Snake_case). Se muestran dos ejemplos de ejecución y su salida esperada:

```
java SnakeCase miIdentificador
```

```
java SnakeCase lastElementInList
```

```
mi_identificador
```

```
last_element_in_list
```

Nota: Para transformar un identificador a *snake case*, debes seguir los siguientes pasos:

1. Convertir todas las letras del identificador a minúsculas.
2. Reemplazar los espacios y guiones bajos con guiones bajos.
3. Si el identificador empieza con un número, agregar un guión bajo al principio.

Asegúrate de probar tu programa con varios identificadores y verificar que el resultado sea correcto.

Ejercicio 44.  **CamelCase** Escribe un programa `CamelCase.java` que transforme un identificador dado a *CamelCase* (ver https://es.wikipedia.org/wiki/Camel_case). Se muestran dos ejemplos de ejecución y su salida esperada:

```
java CamelCase mi_identificador
```

```
java CamelCase last_element_in_list
```

```
miIdentificador
```

```
lastElementInList
```

Nota: Para transformar un identificador a *CamelCase*, debes seguir los siguientes pasos:

1. Convertir todas las letras del identificador a minúsculas.
2. Reemplazar los espacios y guiones bajos con espacios.

3. Convertir la primera letra de cada palabra en mayúscula, excepto la primera palabra.
4. Eliminar los espacios.

Asegúrate de probar tu programa con varios identificadores y verificar que el resultado sea correcto.