Sesión 01: En marcha (Conocimientos básicos) v20240507

Hoja de problemas

Programación 2

Ángel Herranz
aherranz@fi.upm.es
Universidad Politécnica de Madrid

2024-2025

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, se incluye explicaciones sobre cosas que vamos a usar durante todo el curso. Si no sabes hacerlos, no pasa nada, se aprende haciendo... y preguntando `;)`.

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*
- *buscar en internet o libros*
- *programar*
- *pensar, observar*
- *recordar*
- *peligro, atención*
- *éxito*
- *fracaso*
- *en casa*

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

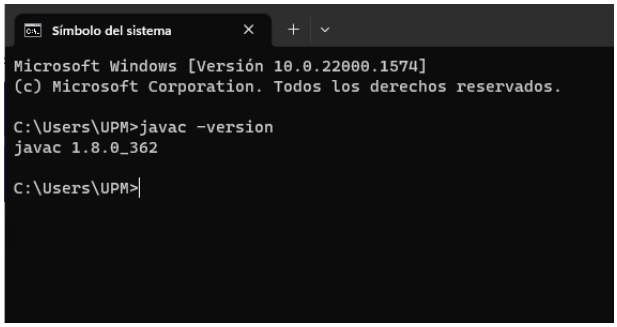
- Un ***kit de desarrollo*** de Java, o ***JDK*** (*Java Development Kit*).
- Un ***editor de texto plano*** como pueden ser `Atom`, `Sublime` o `Notepad++` (hay muchos otros).
- Un ***terminal*** como puede ser `Cmd` ("símbolo de sistema" en Windows) o `gnome-terminal` ("terminal" en Linux).

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
```

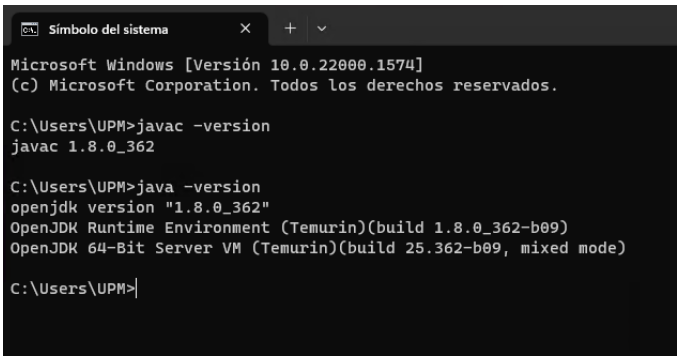
Deberías estar viendo una pantalla parecida a esta:



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:



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 Java 8).

`jshell` **Ejercicio 3.** Si tienes instalada una versión de la JDK por encima de Java 8, es altamente recomendable que aprendas a usar *JShell*, un interprete de Java que te permitirá probar cosas muy simples de una forma muy rápida. Para ponder en marcha *JShell* podrás ejecutar `jshell` en la línea de comandos. Mira un ejemplo de uso con un ejemplo de sentencias en Java:

💻 `Hola` **Ejercicio 4.** 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:

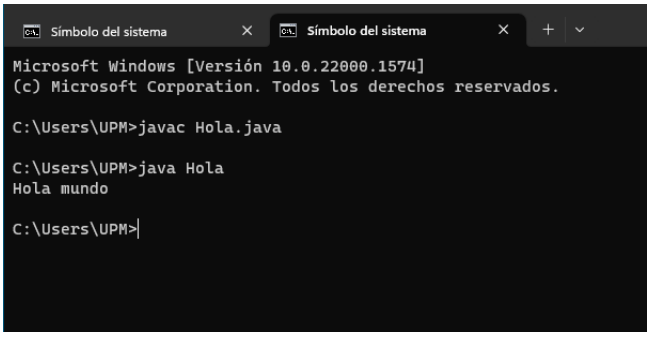
```
javac Hola.java
```

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

```
java Hola
```

Deberías ver una pantalla parecida a esta:

[1] El 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 terminar `Cmd`). Otra opción es abrir el terminal y ***moverte hasta ese directorio*** usando el mandato `cd`.



Ejercicio 5. 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 del programa*** (en línea de comandos al ejecutar el programa).
- Lectura de datos de la ***entrada estándar*** (por defecto *el teclado*).
- Escritura de datos en la ***salida estándar*** (por defecto *la pantalla*).
- Escritura de datos en la ***salida de error*** (por defecto también *la pantalla*).
- ***Valor de terminación*** o ***exit status*** (es un valor numérico que dice si un programa ha terminado bien o mal, no es fácilmente accesible pero en algún momento veremos cómo explorarlo).

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

Con respecto a los argumentos del programa, los podrás ver en el ejercicio 7.

Con respecto a la entrada estándar tendrás que esperar al ejercicio 10.

Con respecto a la salida estándar, creo que con el *método*[2] `System.out.println(...)`[3] tienes más que suficiente para empezar. En todo caso ya el ejercicio 7 trata un poco más de la salida estándar.

Con respecto a la salida de error, sólo deberás saber que es la salida que se usa para emitir mensajes al usuario cuando tu programa detecta un problema. Creo que con el *método* `System.err.println(...)`[4] tienes más que suficiente para empezar.

Con respecto al valor de terminación, podrás practicar con él en el ejercicio 8.

Ejercicio 6. 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 el caso de la versión 8:

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

`args` **Ejercicio 7.** Antes de empezar a hacer programas más elaborados vamos a usar dos re-

[2] **Importante:** (la palabra método es el nombre que se da a las funciones y a los procedimientos en los lenguajes orientados a objeto, su característica clave es que el "objeto primer argumento" se pone delante del nombre del método con un punto en el medio, es decir, en vez de $f(a,b)$ muchas veces vamos a ver escrito $a.f(b)$.

[3] `System.out` es el objeto que representa la salida estándar.

[4] `System.err` es el objeto que representa la salida de error.

cursos fundamentales en programación: argumentos del programa y estado de salida de un programa.

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`).

💻 *exit status*

Ejercicio 8. Cuando un programa termina puede hacerlo *bien* o *mal* (por ejemplo si detecta algún error interno o en los datos). Todos los lenguajes de programación tienen una forma para que tu programa lo comunique, es el *valor de terminación* o *exit status* en inglés.

Si el valor de terminación es 0 (cero) entonces es que el programa ha terminado bien. Si es distinto de 0 es que ha terminado mal. En Java, el valor de terminación 0 es implícito y es el que devuelven nuestros programas si nada se rompe. Para que el programador pueda hacer hacer explícito el valor de termiación puede usar el método `System.exit(s)` donde `s` es un número.

En este ejercicio tienes que modificar el programa anterior (ejercicio 7) para hacer que tu programa termine mal si te piden calcular el factorial de un número menor que 1. Algo así:

```
public class Factorial {
  public static void main(String[] args) {
    String argumento = args[0];
    int resultado = 1;

    POR HACER: convertir argumento a entero
```

```
    if (POR HACER: el entero es menor que 1) {
      System.err.println("Error: argumento menor que 1");
      // Finalizar la ejecución del programa
      // con exit status distinto de 0
      System.exit(255);
    }

    POR HACER: calcular el factorial en resultado

    System.out.println(resultado);
    // Finalizar la ejecución del programa
    // con exit status igual a 0 (por defecto es lo que pasa)
    System.exit(0);
  }
}
```

Pero, una pregunta, ¿cómo puedo saber si mi programa termina bien o mal después de haber acabado? Para consultar el valor de terminación puedes hacerlo desde el terminal de esta forma:

En Windows:

```
C:\Users\UPM> java Factorial -3
Error: argumento menor que 1
C:\Users\UPM> echo %ERRORLEVEL%
255
```

En Linux:

```
$ java Factorial -3
Error: argumento menor que 1
$ echo $?
255
```

Ejercicio 9. 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` asi 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 6).

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 anos, 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 biblitoeca estándar, en concreto en `https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html`.

Scanner **Ejercicio 10.** 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.nextLine();
```

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

Transcribe el programa y ejecútalo para probar.

Ejercicio 11. Busca e intenta entender la documentación sobre la clase `Scanner` en la documentación de la biblitoeca estándar (ver ejercicio 6)

Redirect **Ejercicio 12.** 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 10) 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.

🔍 **Ejercicio 13.** Cualquier programador debería conocer algunos algorítmos *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 14.** 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 comuń divisor de los mismos. Por ejemplo, tu programar debería imprimir en pantalla `5` si la invocación del mismo es

```
java MCD 15 25
```

🔍 `String` **Ejercicio 15.** Como en todos los lenguajes de programación Java también 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 objetos de la clase `String`, i entero, las operaciones más importantes son:

Uso	**Resultado**	**Significado**
`"cualquier cosa"`	`String`	el propio *string* `"cualquier cosa"`
`new` `String(`s_1`)`	`String`	el resultado es una nueva instancia de `String` que es igual a s_1 caracter a caracter
s_1`.length()`	**`int`**	el resultado es la longitud de s_1
s_1`.charAt(`i`)`	**`char`**	el resultado es el caracter que ocupa la índice i en s_1 (precondición: i mayor o igual que 0 y menor que la longitud de s_1)
s_1`.codePointAt(`i`)`	**`int`**	el resultado es el caracter Unicode que ocupa el índice i en s_1 (precondición: i mayor o igual que 0 y menor que la longitud de s_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 *string* que tenga hacentos com `Herranz, Ángel` :)
s_1`.compareTo(`s_2`)`	**`int`**	compara s_1 y s_2 lexicográficamente: devuelve un número menor que 0 si s_1 va antes (lexicográficamente) que s_2, 0 si s_1 y s_2 sin iguales y un número mayor que 0 si s_2 va antes (lexicográficamente) que s_1
s_1`.concat(`s_2`)`	`String`	el resultado es una nueva instancia de `String` que representa la concatenación de los dos *strings* s_1 y s_2
s_1`.contains(`s_2`)`	**`boolean`**	el resultado es **`true`** si s_1 es una subcadena de s_2 y **`false`** en otro caso
s_1`.equals(`s_2`)`	**`boolean`**	el resultado es **`true`** si s_1 y s_2 son iguales y **`false`** en otro caso
s_1 `==` s_2	**`boolean`**	el resultado es **`true`** si s_1 y s_2 son la misma instancia y **`false`** en otro caso
s_1`.indexOf(`c`)`	**`int`**	el resultado es el índice que ocupa el caracter c en el *string* s_1, `-1` si el caracter no aparece en s_1
s_1`.indexOf(`s_2`)`	**`int`**	el resultado es el índice en el que empieza el *string* s_2 en el *string* s_1, `-1` si s_2 no está en s_1
s_1`.isEmpty()`	**`boolean`**	el resultado es **`true`** si el *string* s_1 es igual al *string* `""`, **`false`** en otro caso
s_1`.substring(`i`, `j`)`	`String`	el resultado es una nueva instancia de `String` que representa la subcadena entre los índices i y j (precondición: i, j mayores o igual que 0 y menores que la longitud de s_1 y i menor o igual que j)
s_1`.trim()`	`String`	el resultado es una nueva instancia de `String` que es idéntica a s_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 16. 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`**:

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

¿Se te ocurre a ti misma alguna propiedad nueva?

💻 `Grep`

Ejercicio 17. 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`[5] . 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 enseñes tu limpio desnudo
como un negro cactus abierto en los juncos.

Déjame en un ansia de oscuros planetas,
¡pero no me enseñes tu cintura fresca!
```

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

[5] *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`.

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

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

💻 Lc **Ejercicio 18.** 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 17. Así:

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

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

💻 Wc **Ejercicio 19.** Escribe un programa `Wc.java`[6] 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 17. 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 20.** 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 17.

💻 NumLinea **Ejercicio 21.** 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 17. Quizás veas que es un poco feo el cambio entre la línea `9` y la `10`.

💻 NumLinea^2 **Ejercicio 22.** En el ejercicio anterior (ejercicio 21) 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.
...
```

💻 NumLinea^3 **Ejercicio 23.** Siguiendo con el ejercicio anterior (ejercicio 22). ¿Cómo resolverías el problema si el número de líneas puede superar las 99999 y no tienes un límite conocido?

💻 SumaCole **Ejercicio 24.** Escribe un programa `SumaCole.java` que lea dos números naturales de la entrada estándar e imprima su suma usando como algorimo el algoritmo para sumar que nos enseñaron en el cole. A modo de recordatorio:

[6] *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)`.

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

💻 DibujaSum **Ejercicio 25.** Usando como punto de partida el ejercicio anterior (ejercicio 24), escribe un programa `DibujaSum.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.

💻 GendRand **Ejercicio 26.** 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--`.

💻 Factorial **Ejercicio 27.** En el ejercicio 7 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 28.** En el ejercicio 27 (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?

💻 reverse **Ejercicio 29.** Escribe un *procedimiento* que reciba como entrada un array de enteros y lo invierta. La *interfaz*[7] de dicho procedimiento será algo parecido a

[7] **Importante:** otros nombres para *interfaz* pueden ser *cabecera*, *declaración*, *signatura* o simplemente *tipo*.

```
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 procedimiento reverse");
    System.exit(1);
  }
}
```

💻 reverse[2] **Ejercicio 30.** Escribe una versión nueva de `reverse` del ejercicio anterior para que esta vez devuelva un nuevo array sin tocar el array del parámetro. La interfaz de la función será ahora la 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);
  }
}
```

💻 insertar **Ejercicio 31.** ¿Se te ocurre alguna forma de insertar un dato en una determinada posición de un array de enteros sin perder ningún dato? En este ejercicio tienes que programar una función que tome un array de enteros, un índice y un entero y que devuelva un nuevo array *insertando* el elemento en la posición indicada.

La *interfaz* de dicha función será algo parecido a

```
public static int[] insertar(int[] a, int i, int e)
```

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

```
int[] ejemplo = new int[] {2,5,4,6,4};
int[] esperado;
esperado = insertar(ejemplo, 3, 27);
```

```
if (!(esperado.length + 1 == ejemplo.length
      && esperado[0] == ejemplo[0]
      && esperado[1] == ejemplo[1]
      && esperado[2] == ejemplo[2]
      && esperado[3] == 27
      && esperado[4] == ejemplo[3]
      && esperado[5] == ejemplo[4]))
  System.err.println("Error en función insertar");
```

borrar **Ejercicio 32.** En este ejercicio tienes que programar una función que tome un array de enteros y un índice y que devuelva un nuevo array *borrando* el elemento en la posición indicada.

La *interfaz* de dicha función será algo parecido a

```
public static int[] borrar(int[] a, int i)
```

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

```
int[] ejemplo = new int[] {2,5,4,6,4};
int[] esperado;
esperado = borrar(ejemplo, 3);
if (!(esperado.length - 1 == ejemplo.length
      && esperado[0] == ejemplo[0]
      && esperado[1] == ejemplo[1]
      && esperado[2] == ejemplo[2]
      && esperado[3] == ejemplo[4]))
  System.err.println("Error en función borrar");
```

rev[3] **Ejercicio 33.** Escribe un procedimiento que reciba un `String` como argumento y devuelva un nuevo *string* que sea la inversión del *string* de entrada. La interfaz de la función será la siguiente:

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

Intenta construir tú misma una prueba inspirándote en la prueba del ejercicio 30.

Rev **Ejercicio 34.** 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`[8]. Pruébalo con el fichero `gacela.txt` del ejercicio 17. Si lo ejecutas de esta forma:

```
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
```

[8] *Rev* es una utilidad de Unix que invierte las líneas de un fichero.

```
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 35. Escribe un programa `Primo.java` que recibe un número como argumento en la línea de comandos y determina si es primo o no. Puedes probar tu programa con la siguiente línea de comandos[9]:

```
java Primo 2147483647
```

💻 Factores

Ejercicio 36. 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:

```
java Factores 50
2
5
```

```
java Factores 8932
2
7
11
29
```

```
java Factores 2147483647
2147483647
```

🔍 Sobre la factorización

Ejercicio 37. Te habrás dado cuenta de que factorizar números es bastante costoso en términos 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!), dońde 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 lagoritmos que te animo a explorar. Algunos nombres de algoritmos importantes para factorizar son:

- Algoritmo de Dixon

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

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

DibujaFactores

Ejercicio 38. ¿Recuerdas la *forma* que tenía la descomposición en factores primos en el cole? Por ejemplo, para el número 72:

$$\begin{array}{r|l} 72 & 2 \\ 36 & 2 \\ 18 & 2 \\ 9 & 3 \\ 3 & 3 \\ 1 & \end{array}$$

$72 = 2^3 \times 3^2$

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
```

MulCole

Ejercicio 39. Escribe un programa `MulCole.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.

DibujaMul

Ejercicio 40. Usando como punto de partida el ejercicio anterior (ejercicio 39), escribe un programa `DibujaMul.java` que dibuje la multiplicación de la misma forma en la que lo hacíamos en el cole.

DivCole

Ejercicio 41. Escribe un programa `DivCole.java` que lea dos números naturales de la entrada estándar e imprima su división usando como algorimo el algoritmo para división que nos enseñaron en el cole.

DibujaDiv

Ejercicio 42. Usando como punto de partida el ejercicio anterior (ejercicio 41), escribe un programa `DibujaDiv.java` que dibuje la división de la misma forma en la que lo hacíamos en el cole.

Ultima **Ejercicio 43.** En el ejercicio 10 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(entrada);
    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 44.** 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 espera:

```
java Palindromo Aibofobia          java Palindromo listo

"Aibofobia" es palíndromo          "listo" no es palíndromo
```

Mayusculas **Ejercicio 45.** 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.

💻 `SnakeCase` **Ejercicio 46.** 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.

💻 `CamelCase` **Ejercicio 47.** 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.