Ejercicio 1) Se compone de 3 partes:

- **Primera parte: implementa una aplicación que ordena un conjunto indeterminado de números que recibe a través de su entrada estándar; y muestra el resultado de la ordenación en su salida estándar. La aplicación se llamará 'ordenarNumeros'.**

```java
package com.mycompany.aleatorios;

import java.util.Random;

/**
 * Clase que genera un arreglo de 40 números aleatorios entre 0 y 100 (inclusive),
 * y los imprime en la consola separados por espacios.
 *
 * @author David
 */

public class Main {

    public static void main(String[] args) {
        // Arreglo para almacenar 40 números aleatorios.
        int[] numeros = new int[40];
        Random r = new Random();

        // Generar 40 números aleatorios entre 0 y 100 (inclusive)
        for (int i = 0; i < numeros.length; i++) {
            numeros[i] = r.nextInt(101); // Genera un número entre 0 y 100
        }

        // Imprimir los números separados por espacios
        for (int i = 0; i < numeros.length; i++) {
            System.out.print(numeros[i]);
            if (i < numeros.length - 1) {
                System.out.print(" ");// Agregar un espacio entre los números.
            }
        }

        System.out.println(); // Salto de línea al final
    }
}
```

- **Segunda parte: implementa una aplicación, llamada 'aleatorios', que genere al menos 40 números aleatorios (entre 0 y 100), y que los escriba en su salida estándar.**

```java
package com.mycompany.ordenarnumeros;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

/**
 * Esta clase lee una serie de números enteros de la entrada estándar,
 * los ordena de forma ascendente y los imprime en la consola separados por espacios.
 * Ejemplo de uso:
 * Al ingresar: 3 1 4 1 5
 * La salida será: 1 1 3 4 5
 */

public class Main {

    public static void main(String[] args) {

        // Crear un objeto Scanner para leer los números desde la entrada estándar.
        Scanner entrada = new Scanner(System.in);

        // Lista para almacenar los números ingresados.
        List<Integer> numeros = new ArrayList<>();

        // Leer números de la entrada estándar hasta que no haya más entradas
        while (entrada.hasNextInt()) {
            numeros.add(entrada.nextInt());
        }

        // Ordenar la lista de números en orden ascendente
        Collections.sort(numeros);
        //Si hubiera que rdenarlos de manera descendente:
        //Collections.sort(numeros, Collections.reverseOrder());

        // Imprimir los números ordenados, separados por espacios
        for (int i = 0; i < numeros.size(); i++) {
            System.out.print(numeros.get(i));
            if (i < numeros.size() - 1) {
                System.out.print(" ");// Agregar un espacio entre los números.
            }
        }

        System.out.println();// Salto de línea al final.
        entrada.close();// Cerrar el objeto Scanner para liberar recursos.
    }
}
```

- **Tercera parte: Realiza un pequeño manual (tipo "¿Cómo se hace?" o "HowTo"), utilizando un editor de textos (tipo word o writer) en el que indiques, con pequeñas explicaciones y capturas, cómo has probado la ejecución de las aplicaciones que has implementado en este ejercicio. Entre las pruebas que hayas realizado, debes incluir una prueba en la que utilizando el operador "|" (tubería) redirijas la salida de la aplicación 'aleatorios' a la entrada de la aplicación 'ordenarNumeros'.**

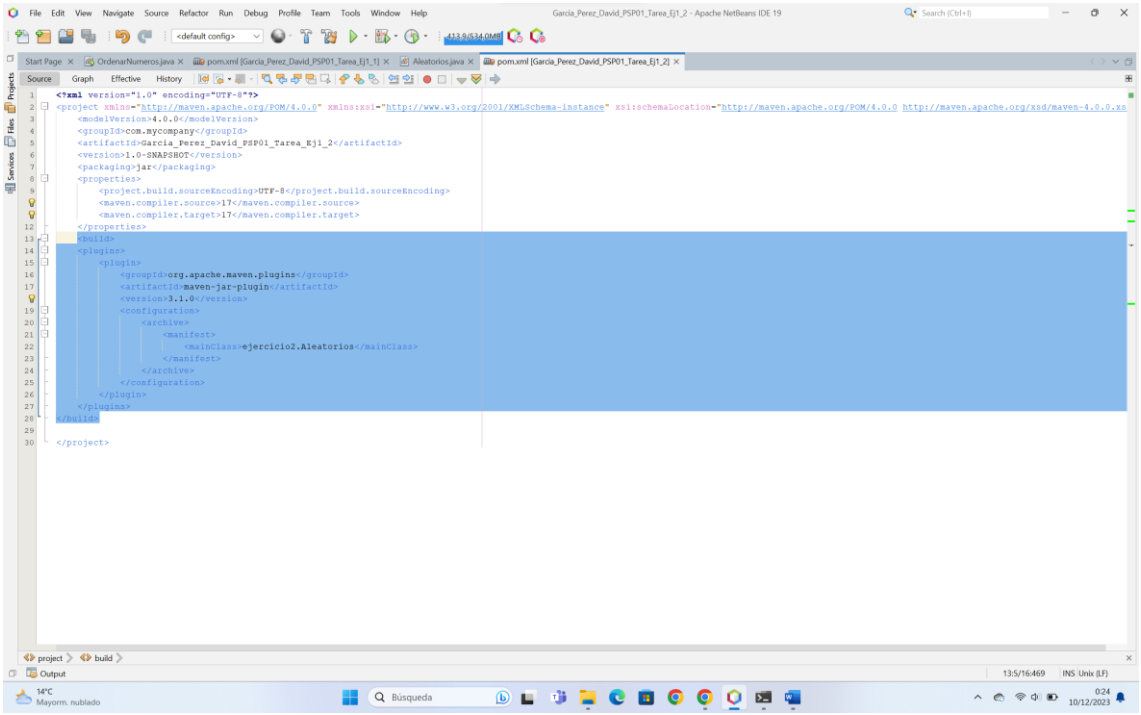
Ejecución de la aplicación Aleatorios

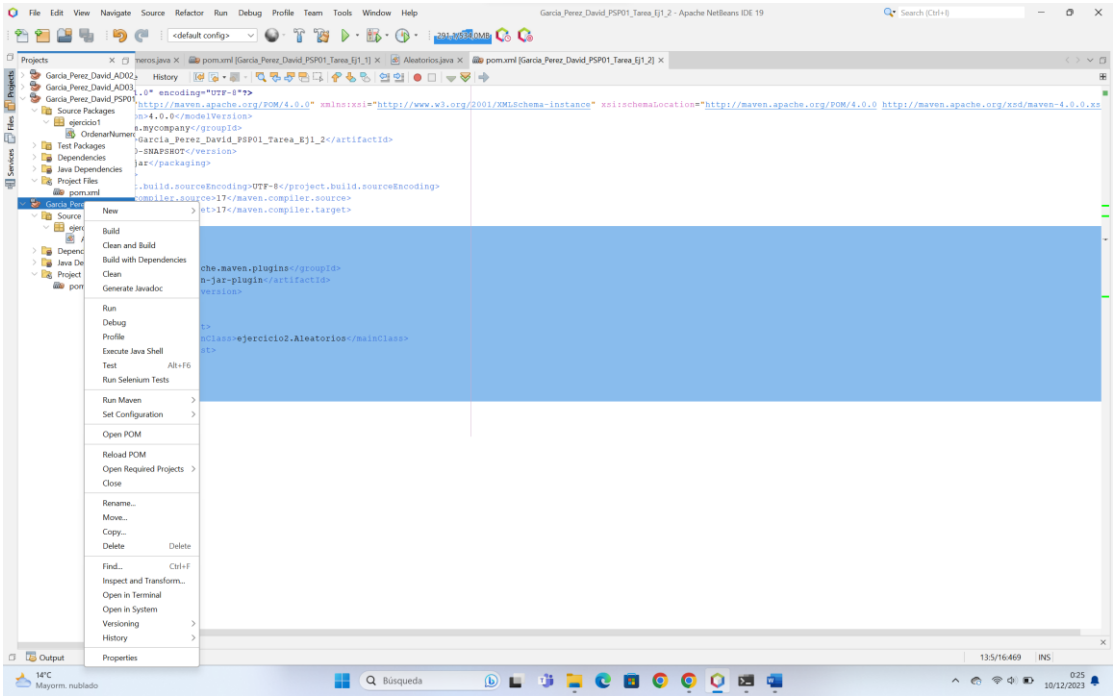
• Descripción: La aplicación aleatorios genera una lista de 40 números enteros aleatorios entre 0 y 100.

• Pasos:

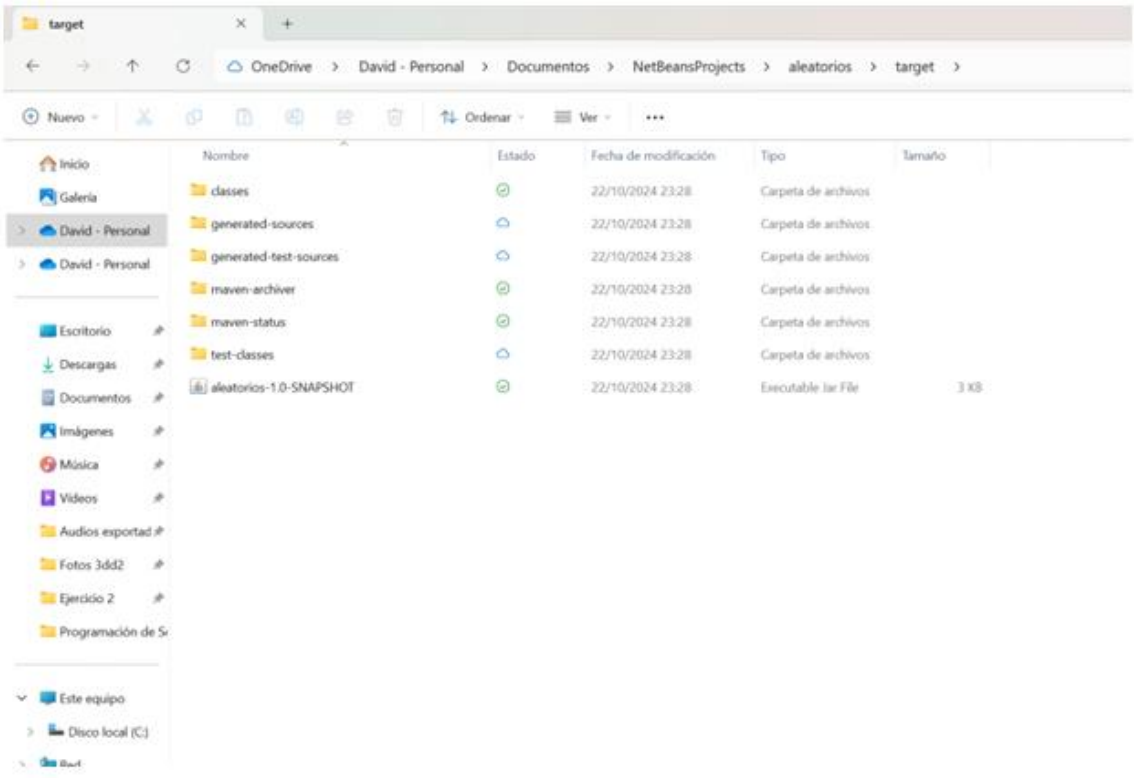
1.1 Crear el archivo JAR: Para ello, en el apartado de proyectos de NetBeans, seleccionaríamos con el botón derecho la aplicación y pulsaríamos Clean and Build.

Pero antes de esto, debemos agregar la siguiente configuración en el archivo POM.xml del proyecto para crear un JAR ejecutable: Esto generará el archivo .jar en la carpeta target del proyecto:

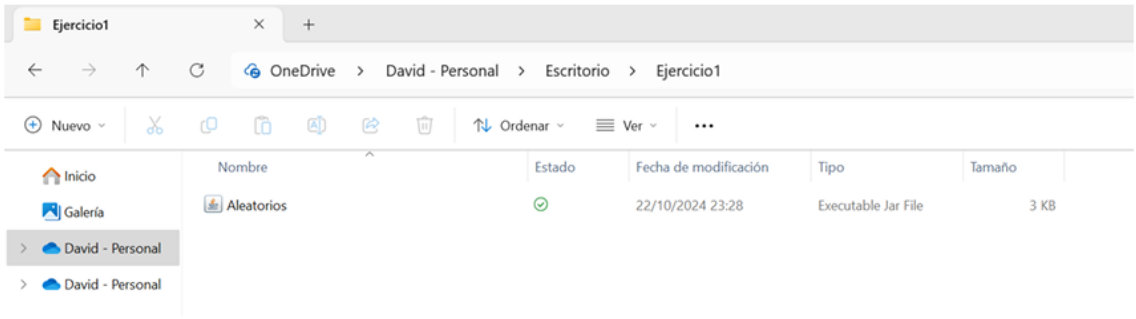




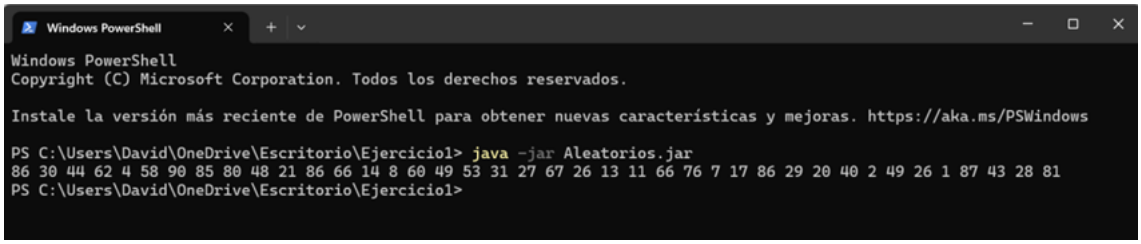
Esto generará el archivo .jar en la carpeta target del proyecto:



1.2 Ejecutar la aplicación: Para ejecutar la aplicación, copiamos el archivo aleatorios.jar en la carpeta ejercicio 1.



Desde esa carpeta abrimos la terminal y comprobamos el funcionamiento de la aplicación con la siguiente línea de comandos: java -jar aleatorios.jar



De esta manera obtenemos una lista de 40 números aleatorios entre el 1 y el 100 separados por espacios.

2. Ejecución de la Aplicación ordenarNumeros

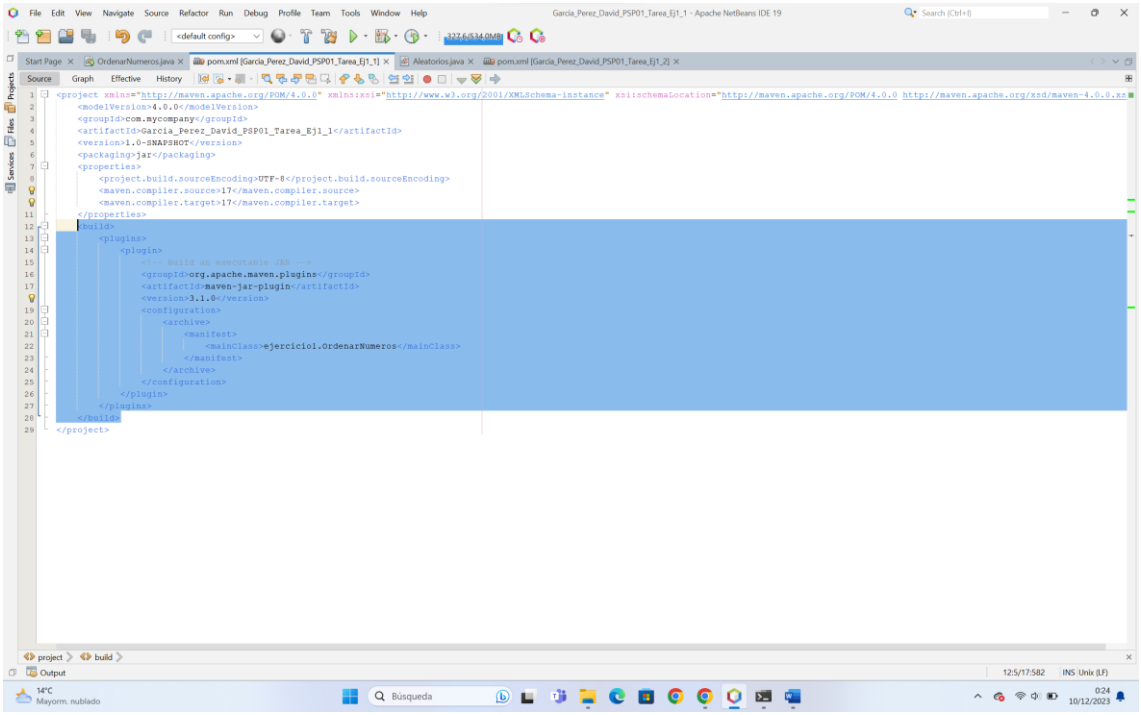
• **Descripción:** La aplicación ordenarNumeros lee una lista de números de la entrada estándar, los ordena y los imprime.

• **Pasos:**

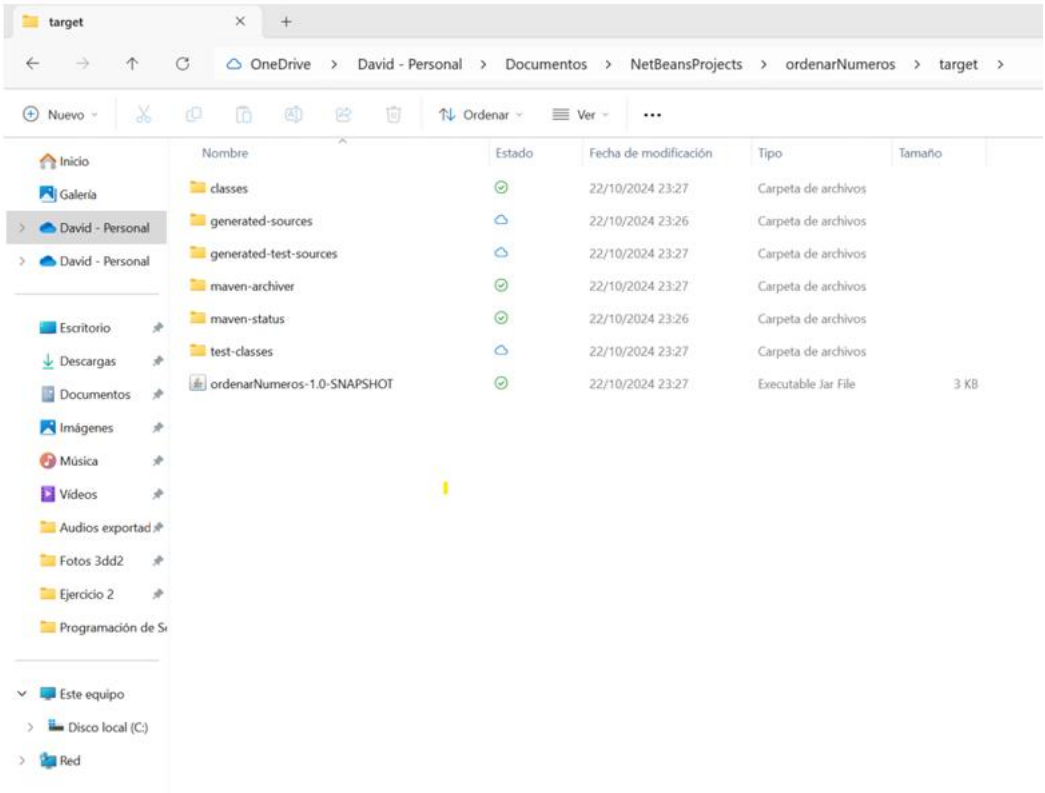
2.1 Crear el archivo JAR:

Para ello, en el apartado de proyectos de NetBeans, seleccionaríamos con el botón derecho la aplicación y pulsaríamos Clean and Build.

Pero antes de esto, debemos agregar la siguiente configuración en el archivo POM.xml del proyecto para crear un JAR ejecutable:

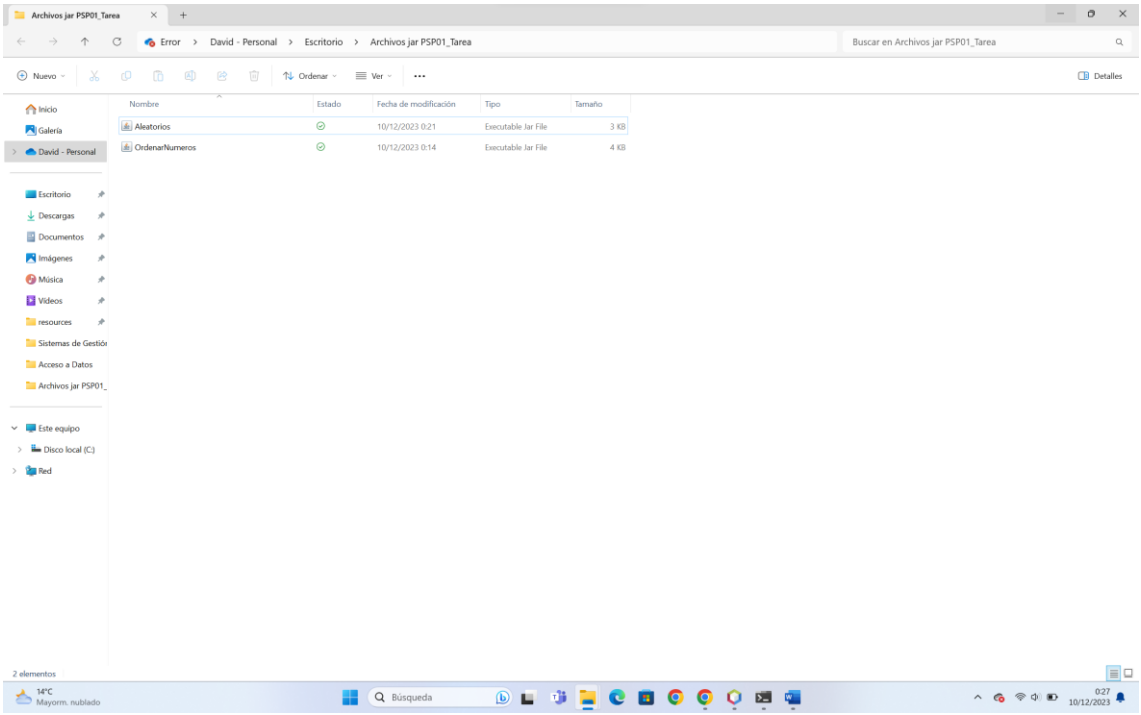


Esto generará el archivo .jar en la carpeta target del proyecto:



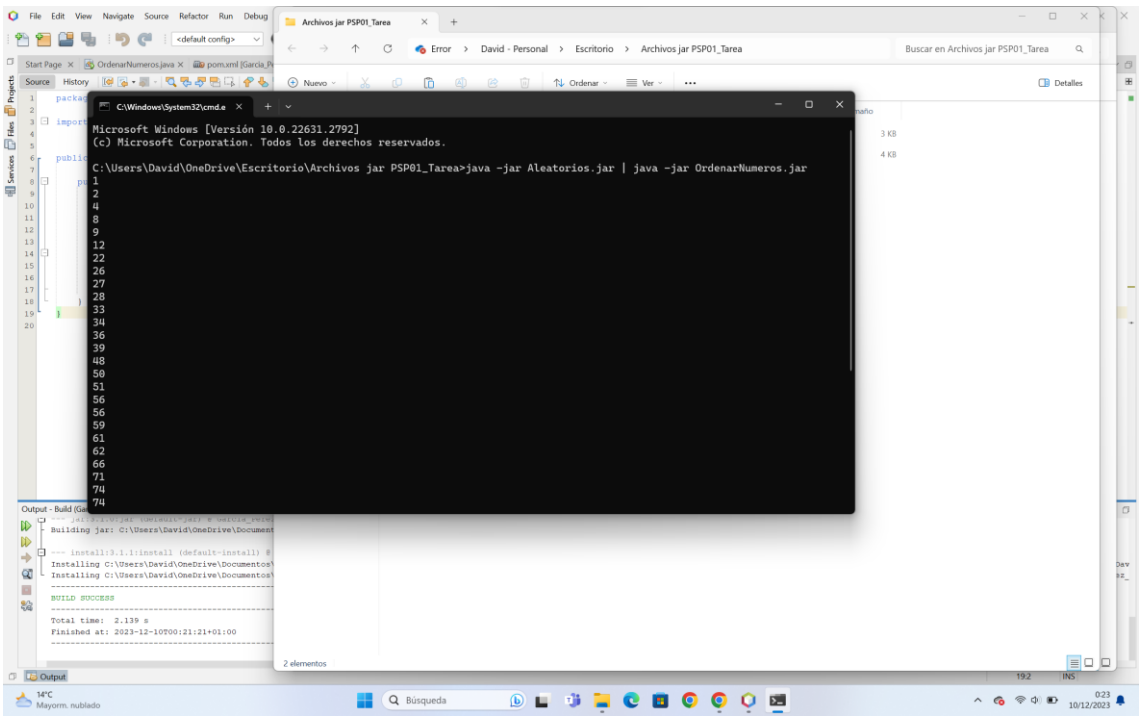
2.2 Ejecutar la aplicación:

Para ejecutar la aplicación, copiamos el archivo ordenarNumeros.jar en la carpeta ejercicio 1.



Desde esa carpeta abrimos la terminal y comprobamos el funcionamiento de la aplicación dando valores de números por la entrada estándar y usando el operador | para redirigir esta entrada con la aplicación ordenarNumeros con la siguiente línea de comandos:

echo "23 5 89 34 67 12 45" | java -jar ordenarNumeros.jar



3. Redirigir la salida de aleatorios a la entrada de ordenarnumeros :

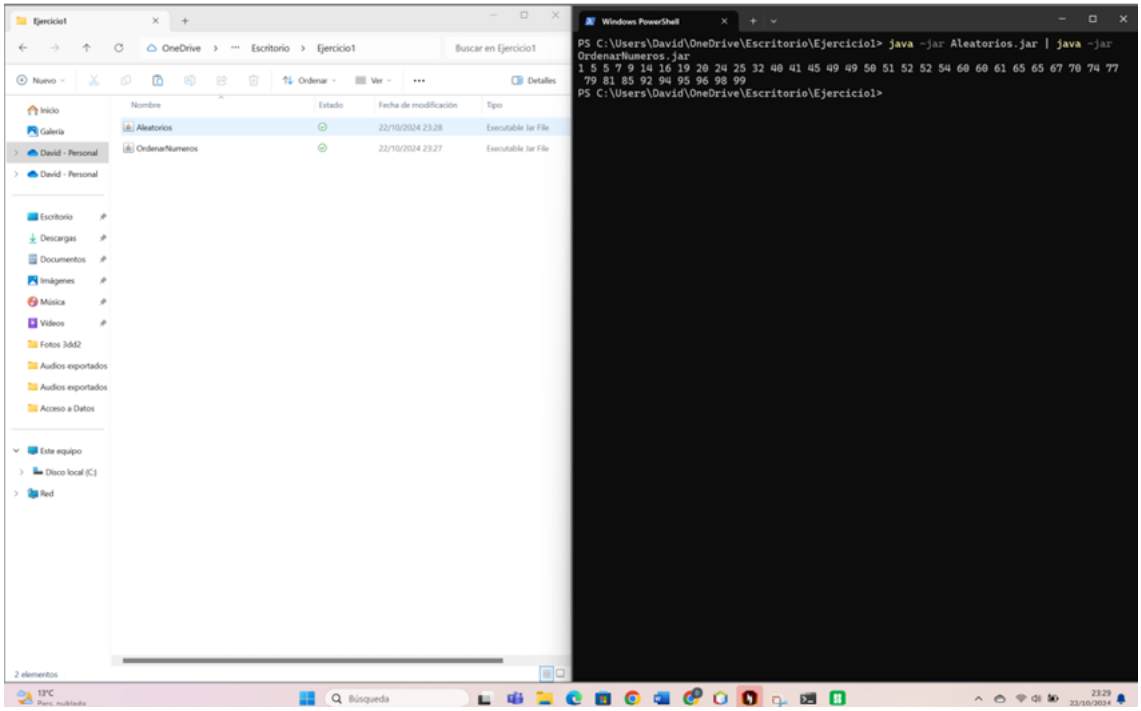
Abrimos la terminal en la carpeta Ejercicio 1 y usamos el operador | para redirigir la salida de Aleatorios como entrada de ordenarNumeros.

Ejecutar ambos JARs con la tubería con el siguiente código:

java -jar aleatorios.jar | java -jar ordenarnumeros.jar

Explicación:

- java -jar aleatorios.jar genera una lista de números aleatorios.
- | (pipe) redirige esa lista como entrada a java -jar ordenarnumeros.jar, que ordena y muestra los números.



Estas aplicaciones trabajando conjuntamente nos muestran una lista de 40 números ordenados de menor a mayor.

4. Conclusiones

- Las aplicaciones Aleatorios y ordenarNumeros pueden trabajar de forma independiente o de manera conjunta usando una tubería para generar y ordenar números.
- Esto permite que Aleatorios actúe como un generador de datos y ordenarNumeros como un procesador, una técnica útil en el procesamiento de datos en línea de comandos.

Ejercicio 2) Se compone de 3 partes:

- **Primera parte: implementa una aplicación que escriba en un fichero indicado por el usuario conjuntos de letras generadas de forma aleatoria (sin sentido real). Escribiendo cada conjunto de letras en una línea distinta. El número de conjuntos de letras a generar por el proceso, también será dado por el usuario en el momento de su ejecución. Esta aplicación se llamará "lenguaje" y como ejemplo, podrá ser invocada así:**

java -jar lenguaje 40 miFicheroDeLenguaje.txt

Indicando que se generarán 40 palabras del lenguaje y serán guardadas en miFicheroDeLenguaje.txt.

```java
package com.mycompany.lenguaje;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

/**
 * Aplicación que genera conjuntos de letras aleatorios y los escribe en un fichero.
 * Se invoca como: java -jar lenguaje.jar numeroConjuntos nombreArchivo
 * Ejemplo: java -jar lenguaje.jar 40 miFicheroDeLenguaje.txt
 *
 * @author David
 */
public class Main {

    public static void main(String[] args) {
        // Verificar que se han pasado dos argumentos
        if (args.length < 2){
            System.out.println("Instrucción incorrecta. Ejemplo: java -jar lenguaje.jar 40 archivo.txt ");
            return;
        }


        try {
            int numeroPalabras = Integer.parseInt(args[0]); //Primer argumento es una instanca de tipo entero (cantidad de palabras generadas)
            String nombreDocumento = args[1];//Segundo argumento de tipo texto con el nombre del fichero.

            generarConjuntoPalabras(numeroPalabras, nombreDocumento);// Instanciamos la clase

        } catch (IOException | NumberFormatException e) {
            System.out.println("Error al escribir el argumento.");
        }
    }
```

```java
    /**
     * Genera el número de palabras dadas como parámetro y las escribe en un archivo con un nombre dado por parámetro.
     *
     * @param cantidadPalabras Número de conjuntos de letras a generar.
     * @param nombreArchivo    Nombre del archivo donde escribir las palabras.
     * @throws IOException     Si ocurre un error de escritura.
     */

    public static void generarConjuntoPalabras(int cantidadPalabras, String nombreArchivo) throws IOException {
        //Utilizamos un String[] letras que contiene todas las letras del abecedario excepto la ñ.
        String[] letras = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"};
        Random r = new Random();

        //Se utiliza un FileWriter dentro de un try-with-resources para asegurar el cierre automático del archivo al terminar.
        //Ponemos el valor true como parámetro en FileWriter para que permita rellenar el archivo de texto sin sobreescribirlo
        try (FileWriter writer = new FileWriter(nombreArchivo, true)) {
            for (int i = 0; i < cantidadPalabras; i++) {
                StringBuilder palabra = new StringBuilder();//Cada palabra se construye usando un StringBuilder ya que permite la concatenación de caracteres.
                int longitudPalabra = r.nextInt(26) + 1;// Longitud entre 1 y 26

                for (int j = 0; j < longitudPalabra; j++) {

                    int posicionLetra = r.nextInt(letras.length);
                    String caracter = letras[posicionLetra];
                    palabra = palabra.append(caracter);
                }
                //Cada palabra generada se escribe en una línea diferente usando writer.write() seguido de System.lineSeparator().
                writer.write(palabra.toString());
                writer.write(System.lineSeparator());

            }
        }

    }
}
```

- **Segunda parte: implementa una aplicación, llamada 'colaborar', que lance al menos 10 instancias de la aplicación "lenguaje". Haciendo que todas ellas, colaboren en generar un gran fichero de palabras. Cada instancia generará un número creciente de palabras de 10, 20, 30, … Por supuesto, cada proceso seguirá escribiendo su palabra en una línea independiente de las otras. Es decir, si lanzamos 10 instancias de "lenguaje", al final, debemos tener en el fichero 10 + 20 + 30 + … + 100 = 550 líneas.**

```java
package com.mycompany.colaborar;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Clase que lanza 10 instancias de la aplicación "lenguaje" para generar un archivo de palabras.
 * Las instancias generarán un número creciente de palabras: 10, 20, 30, ..., 100.
 *
 * Se invoca como: java -jar colaborar.jar
 *
 * @author David
 */
public class Main {

    public static void main(String[] args){
        // Lanzar 10 instancias de la aplicación "lenguaje" con un número creciente de palabras
        for (int i = 1; i <= 10; i++) {
            int numeroInstancias;
            numeroInstancias = i*10;// 10, 20, 30, ..., 100
            try {
                // Crear el comando para ejecutar la aplicación "lenguaje"
                ProcessBuilder builder = new ProcessBuilder("java", "-jar", "Lenguaje.jar", String.valueOf(numeroInstancias), "FicheroConjunto.txt");
                // Iniciar el proceso
                Process proceso = builder.start();
                // Esperar a que el proceso termine antes de lanzar el siguiente
                proceso.waitFor();

            } catch (IOException | NumberFormatException e) {
            System.out.println("Error al iniciar el proceso.");
            } catch (InterruptedException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
        System.out.println("Archivo FicheroConjunto.txt generado.");

    }
}
```

- **Tercera parte: Realiza un pequeño manual (tipo "¿Cómo se hace?" o "HowTo"), utilizando un editor de textos (tipo word o writer) en el que indiques, con pequeñas explicaciones y capturas, cómo has probado la ejecución de las aplicaciones que has implementado en este ejercicio.**

1. Ejecución de la aplicación Lenguaje

• Descripción:

La aplicación lenguaje genera un número de conjuntos de letras aleatorias especificado por el usuario y guarda cada conjunto en una línea del fichero especificado.

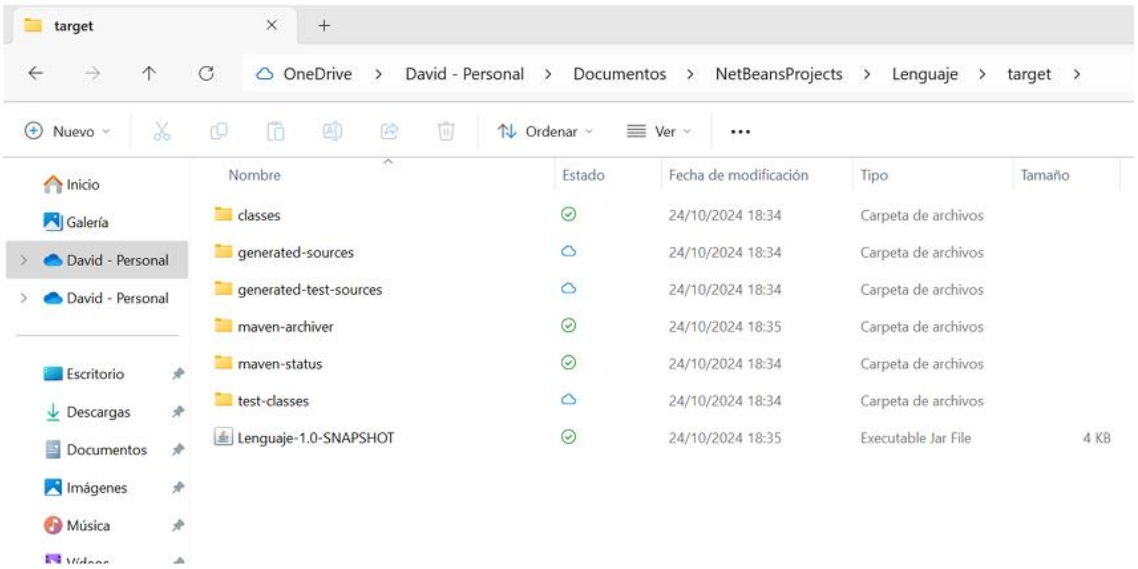
• Pasos:

1.1 Crear el archivo JAR:

Para ello, en el apartado de proyectos de NetBeans, seleccionaríamos con el botón derecho la aplicación y pulsaríamos Clean and Build. Pero antes de esto, debemos agregar la siguiente configuración en el archivo POM.xml del proyecto para crear un JAR ejecutable:

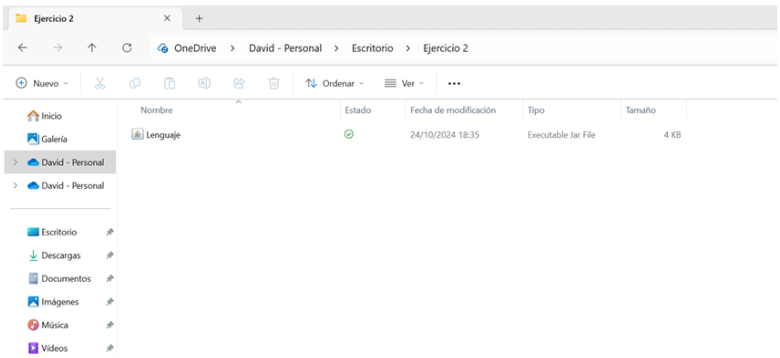
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://mave
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany</groupId>
    <artifactId>Lenguaje</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.release>21</maven.compiler.release>
        <exec.mainClass>com.mycompany.lenguaje.Main</exec.mainClass>
    </properties>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.1.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
                            <mainClass>com.mycompany.lenguaje.Main</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Esto generará el archivo .jar en la carpeta target del proyecto:



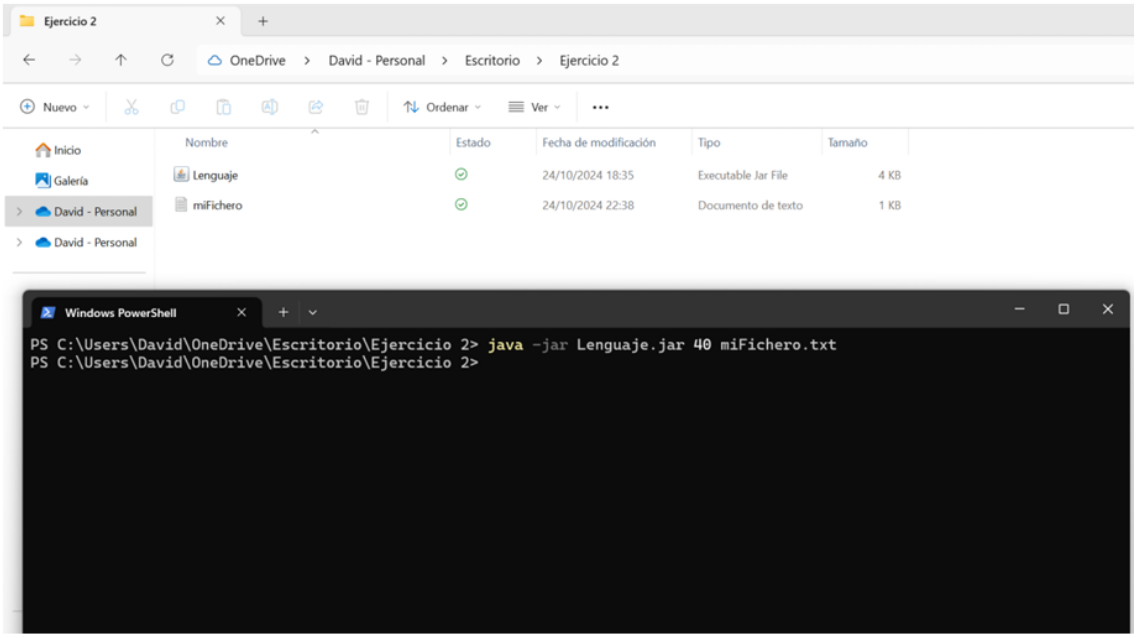
1.2 Ejecutar la aplicación:

Para ejecutar la aplicación, copiamos el archivo aleatorios.jar en la carpeta ejercicio 2.

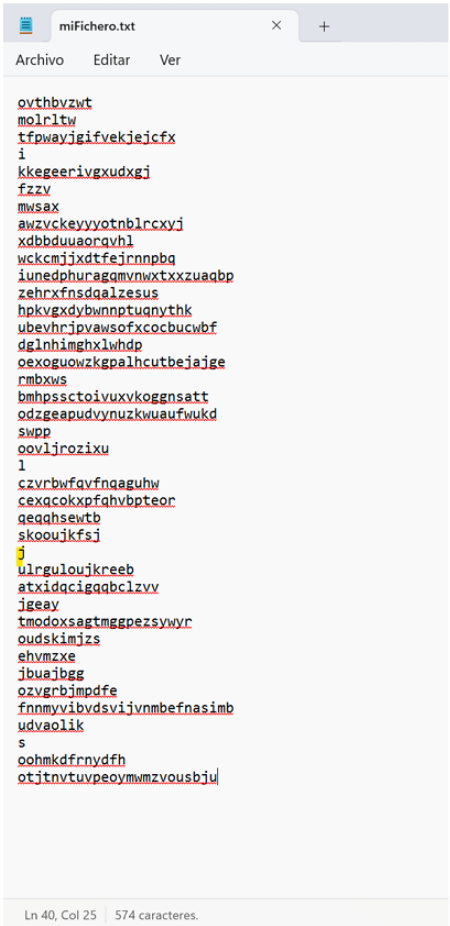


Desde esa carpeta abrimos la terminal y comprobamos el funcionamiento de la aplicación con la siguiente línea de comandos:

java -jar lenguaje.jar 40 miFicheroDeLenguaje.txt



De esta manera obtenemos un archivo llamado miFicheroDeLenguaje.txt que contiene 40 líneas de conjuntos de letras aleatorias:



2. **Ejecución de la aplicación Colaborar**

• **Descripción:** La aplicación colaborar lanza múltiples instancias de lenguaje para generar un gran fichero con un número creciente de conjuntos de letras.

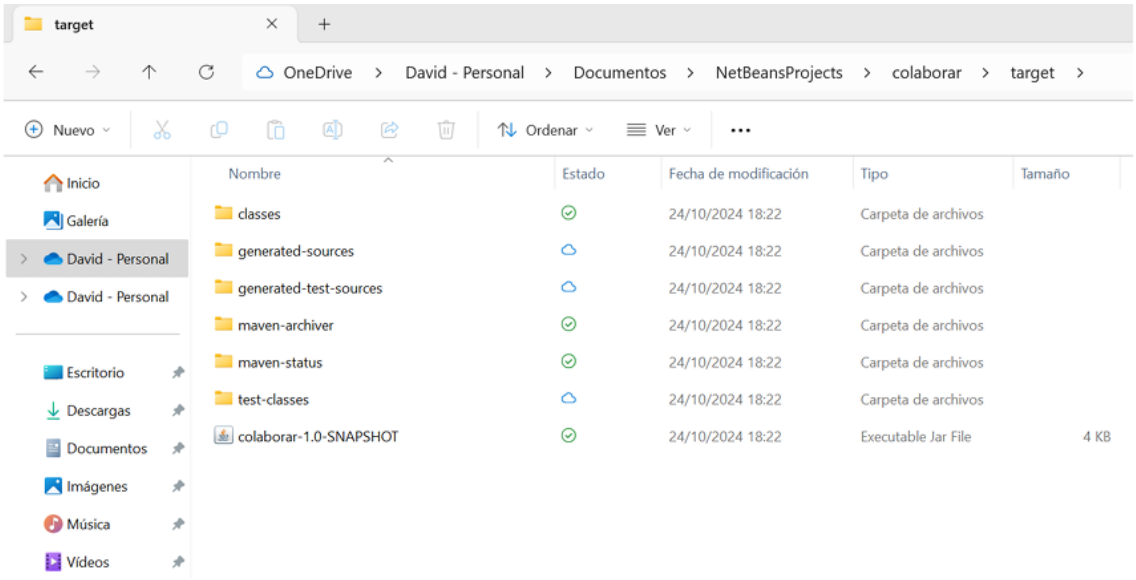
• **Pasos:**

2.1 Crear el archivo JAR:

Para ello, en el apartado de proyectos de NetBeans, seleccionaríamos con el botón derecho la aplicación y pulsaríamos Clean and Build.
Pero antes de esto, debemos agregar la siguiente configuración en el archivo POM.xml del proyecto para crear un JAR ejecutable:

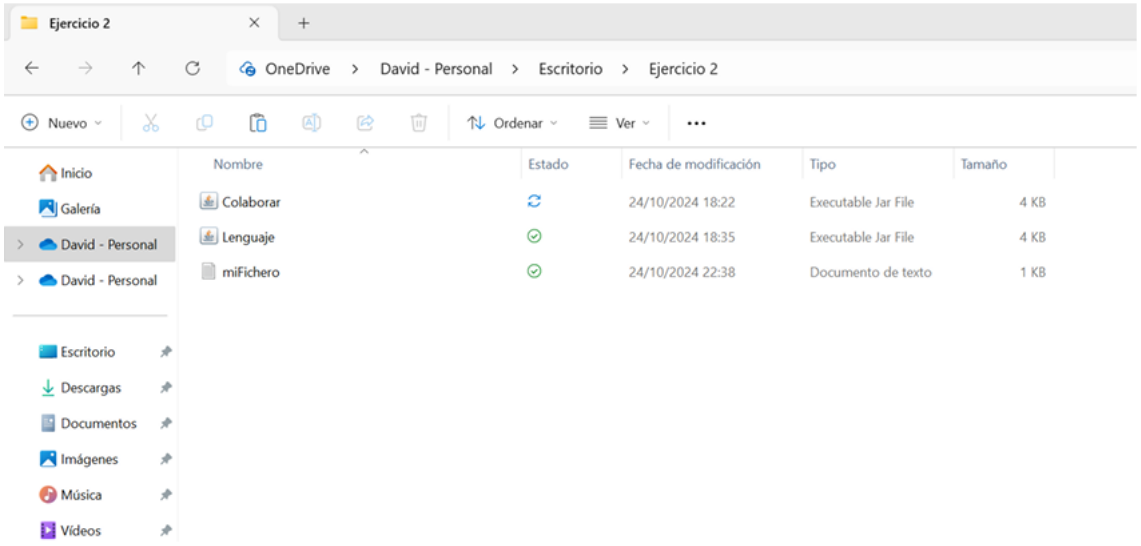
```
Source  Graph  Effective  History
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany</groupId>
    <artifactId>colaborar</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.release>21</maven.compiler.release>
        <exec.mainClass>com.mycompany.colaborar.Main</exec.mainClass>
    </properties>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.1.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
                            <mainClass>com.mycompany.colaborar.Main</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Esto generará el archivo .jar en la carpeta target del proyecto:



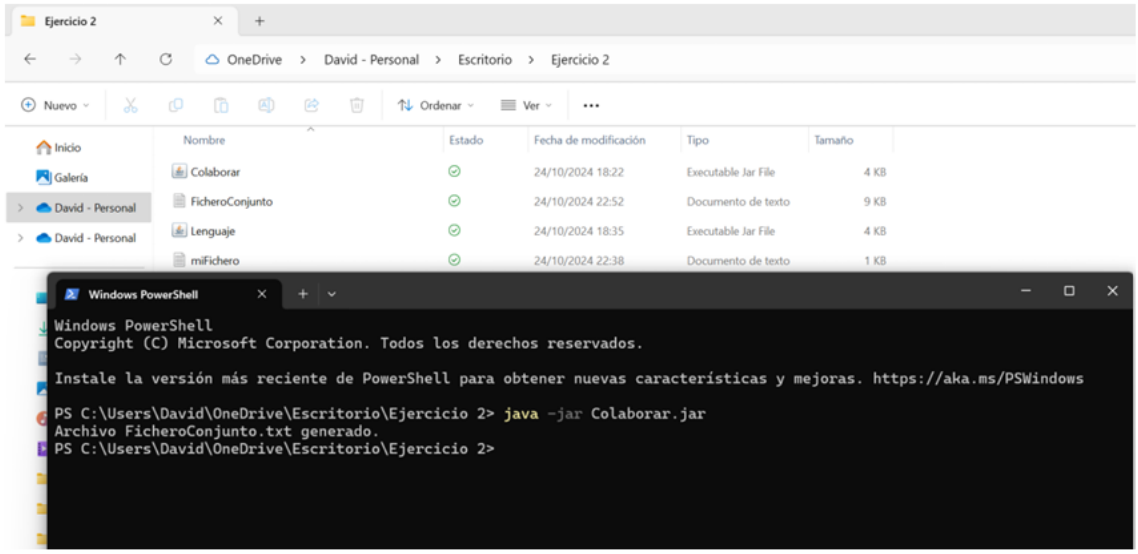
2.2 Ejecutar la aplicación:

Para ejecutar la aplicación, copiamos el archivo aleatorios.jar en la carpeta ejercicio 2.

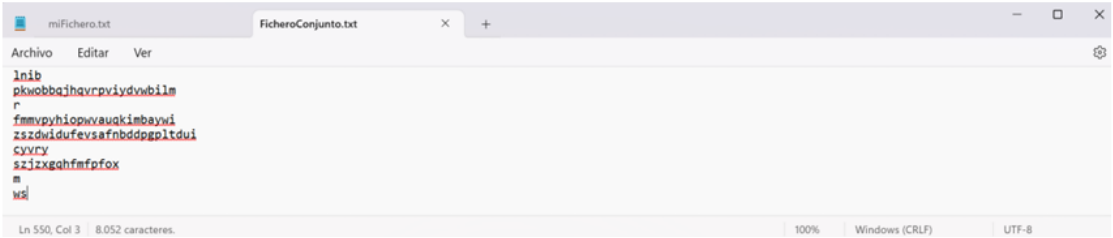


Desde esa carpeta abrimos la terminal y comprobamos el funcionamiento de la aplicación con la siguiente línea de comandos:

java -jar colaborar.jar



De esta manera obtenemos un archivo llamado FicheroConjunto.txt que contiene todas las instancias de lenguaje generando un archivo de texto con un total de 550 palabras:



3. Conclusiones

• Las aplicaciones lenguaje y colaborar permiten generar archivos de texto con conjuntos de letras aleatorias, ya sea de manera independiente o colaborando para generar un archivo más grande.