```
import java.io.*;
import java.util.*;
/**
 * Esta clase contiene algunas funciones auxiliares para el funcionamiento del
 * programa. Es pública, ya que algunas funciones pueden ser de utilidad en otros
 * problemas.
 * @author (Ignacio García)
 */
public class Funciones {

    /**
     * Constante que indica un archivo erróneo si es devuelta por la función
     * patronArchivo.
     */
    public static final int ERROR_ARCHIVO = -1;
    /**
     * Constante que indica un archivo ya existente si es devuelta por la función
     * patronArchivo.
     */
    public static final int EXISTE_ARCHIVO = 0;
    /**
     * Constante que indica un archivo válido si es devuelta por la función
     * patronArchivo.
     */
    public static final int ARCHIVO_VALIDO = 1;
    /**
     * Procedimiento que crea el texto de ayuda y lo muestra por pantalla.
     */
    static void mostrarAyuda() throws IOException {
        StringBuilder s = new StringBuilder(); // uso un objeto stringBuilder para
        // evitar el coste de la concatenación (+) de la clase String
        s.append("                    SINTAXIS \n");
        s.append("            ====================== \n\n");

        s.append(" java trimino [-t] [-h] x y [d] [fichero] \n");
        s.append(" Nota: Los argumentos entre corchetes ([ ]) son opcionales.");
        s.append("\n\n");

        s.append(" Descripción de los argumentos: \n");
        s.append(" ---------------------------- \n\n");

        s.append(" '-t' (opcional): Traza el algoritmo, imprimiendo la tabla de");
        s.append(" caracteres en cada trimino colocado. \n\n");

        s.append(" '-h' (opcional): Ayuda que vuelve a recordar la sintaxis; este");
        s.append(" argumento puede estar en cualquier posición, y se ejecuta aunque");
        s.append(" haya errores en los demás argumentos.\n\n");

        s.append(" 'x' e 'y' (los únicos argumentos obligatorios): Coordenadas");
        s.append(" (horizontal de izquierda a derecha y vertical de arriba a abajo,");
        s.append(" respectivamente), donde se colocará la marca \n");
        s.append(" especial '#'. Deben estar entre 1 y d, ambos inclusive. \n\n");

        s.append(" 'd' (opcional): Dimensión de la tabla (número de filas y");
        s.append(" columnas) a rellenar; debe ser mayor que 0 y potencia de 2.");
        s.append(" Por defecto, si no se introduce este argumento, se hará \n");
        s.append(" con una tabla de 8 x 8. \n\n");

        s.append(" 'fichero' (opcional):  Archivo de salida donde se desee imprimir");
```

```
        s.append(" la salida del programa. Si el archivo existe, el programa");
        s.append(" pregunta al usuario si quiere sobreescribirlo. \n");
        s.append(" Si la respuesta es no, el programa termina en ese punto. Si no");
        s.append(" existe el archivo lo crea el programa, siempre que tenga un");
        s.append(" formato válido. \n");
        s.append(" Para que el formato sea correcto, la ruta de directorios escrita");
        s.append(" en el argumento (si se escribió una ruta) ha de existir; el");
        s.append(" nombre del archivo ha de estar formado por letras, \n números, o");
        s.append(" símbolos '-', '_', 'ñ' ó 'Ñ'; después ha de tener un punto '.' y");
        s.append(" una extensión, formada por caracteres (entre uno y tres, ambos\n");
        s.append(" inclusive) que han de ser letras o números.\n");
        s.append(" Si no se introduce este argumento se imprimirá la salida en la");
        s.append(" consola.");
        s.append("\n\n\n\n");
        System.err.print(s); // imprimimos el texto creado en la consola
    }

    /**
     * Función que comprueba si el texto dado representa un número entero; diseñada
     * para cuando buscamos un número que ha de ser mayor o igual que 0.
     * @param a Texto a evaluar
     * @return Si el texto representa un entero, devuelve ese entero, y si no, -1.
     */
    public static int compruebaEntero(String a) {
        try {
            return Integer.parseInt(a); // uso esta función estática de la clase
            // Integer; si el texto 'a' no representa un entero, produce una excepción
            // que capturo debajo.
        }
        catch(NumberFormatException e) {
            return -1; // en el caso de que quisiéramos generalizar la función a la
            // comprobación de enteros con cualquier signo, bastaría cambiar el tipo
            // de retorno a un objeto Integer y en el catch devolver null.
        }
    }

    /**
     * Función que comprueba si el entero dado es potencia de dos.
     * @param n Entero a evaluar
     * @return Devuelve true si es potencia de dos, y si no, false.
     */
    public static boolean potenciaDeDos(int n) {
        return (Integer.toBinaryString(n).matches("1?0*")); // si el número en binario
        // es bien 1, bien 0 ó de la forma 1000 ... 000, entonces es potencia de 2
    }

    /**
     * Función que pregunta si se desea sobreescribir un archivo ya existente.
     * @return Devuelvelve la respuesta entrada por consola.
     */
    public static String leerConsola(String a) {
        String respuesta = null; // respuesta que se leerá por consola
        int fallos = 0; // número de fallos al escribir en consola
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        // uso un objeto BufferedReader para leer la respuesta
        System.err.print(
        "AVISO: el archivo " + a + " ya existe; ¿desea sobreescribirlo? (s, n): ");
        // Saco el texto por System.err por precaución: En el caso de que se haya
        // redireccionado la salida del programa con ">"; si por un fallo se hubiera
```

```
        // dado el argumento del archivo, al usar System.out la pregunta se escribiría
        // en el archivo, mientras que la consola del sistema operativo estaría
        // esperando una respuesta sin haber escrito la pregunta.
         while(fallos < 5) { // permitiremos un máximo de 5 fallos
            try {
                respuesta = br.readLine(); // leemos por consola la respuesta
            }
            catch(IOException e) {
                System.err.println(
                "error de lectura de consola; el fichero no se sobreescribirá.");
                respuesta = "n"; // en caso de un error de lectura, no sobreescribimos
                // el fichero
            }
            if((respuesta.equals("s")) || (respuesta.equals("n"))) // romper el bucle
            // si la respuesta es válida
                break;
            else {
                if (fallos < 4)
                    System.err.print("error; escriba 's' o 'n': ");
                    // pedir una respuesta válida
                fallos ++; // sumar un fallo
            }
        }
        return respuesta;
    }

    /**
     * Función que comprueba si el texto de entrada representa un archivo válido.
     * @param a Texto a comprobar
     * @return - Devuelve EXISTE_ARCHIVO si:
     * Se trata de un nombre de archivo existente, con o sin
     * ruta (si no se escribe ruta, la función sólo comprobará el propio directorio
     * del programa).
     *
     * - Devuelve ARCHIVO_VALIDO si:
     * El archivo no existe (se creará), pero tanto su nombre como la ruta (en su
     * caso) tienen un formato correcto.
     *
     * - Devuelve ERROR_ARCHIVO en otro caso.
     *
     * Para que el formato sea correcto, la ruta de directorios escrita en el
     * argumento ha de existir; el nombre del archivo ha de estar formado por letras,
     * números, o símbolos '-', '_', 'ñ' ó 'Ñ'; después ha de tener un punto '.' y una
     * extensión, formada por caracteres que han de ser letras o números, de los
     * cuales debe tener como mínimo uno y como máximo tres.
     */
    public static int patronArchivo(String a) {
        if (new File(a).isFile())  // si el archivo existe
            return EXISTE_ARCHIVO; // lo notifica la función y termina
        else if (a.matches("(\\w+ñ*Ñ*-*_*)+[.]\\w\\w?\\w?")) // si el nombre de
        // archivo es simple (no tiene ruta) y tiene un formato correcto
            return ARCHIVO_VALIDO;
        else if (a.matches(".?\\w+.?[.]\\w\\w?\\w?")) { // si no es un nombre simple
            // de  archivo pero la extensión es correcta
            String[] tabla = a.split(File.separator); // parto la ruta por los
            // símbolos '/' que indican los directorios
            StringBuilder s = new StringBuilder();
            int i;
            for (i = 1; i < tabla.length - 1; i ++) { // copio todos los trozos menos
```

```
                // el último, que debería ser el nombre del archivo, para recoger la
                // ruta;
                s.append(File.separator); s.append(tabla[i]); // vuelvo a poner los
                // símbolos '//' que deben separar los directorios
            }

            if ((new File(s.toString()).isDirectory()) &&
            (tabla[i].matches("(\\w+ñ*Ñ*-*_*)+[.]\\w\\w?\\w?")))  // si el texto así
            // formado representa un directorio existente y la última parte cortada es
            // un nombre de archivo válido
                return ARCHIVO_VALIDO;
            else
                return ERROR_ARCHIVO;
        }
        else
            return ERROR_ARCHIVO;
    }
}
```