```
import java.util.*;
import java.io.*;
/**
 * Esta clase contiene la función baldosas, que escribe los triminos, y también
 * algunas otras funciones y campos necesarios.
 *
 * @author (Ignacio García)
 */
class Baldosas {
    /**
     * Tabla de caracteres que llenaremos de triminos
     */
    private char[][] Suelo; // Esta es la tabla de caracteres que hay que llenar con
    // triminos. Inicialmente lo llenaremos de puntos ('.')
    /**
     * Dimensión de la tabla, de N x N. Por defecto será de 8 x 8, si no se especifica
     * otra cifra en los argumentos de entrada.
     */
    private int N;
    /**
     * Siguiente caracter a escribir en el próximo trimino
     */
    private char sigCaracter = '0'; // El siguiente caracter a ser escrito en la
    //tabla; comenzamos por el numero 0
    /**
     * Coordenada 'x' de la tabla donde se coloca la marca '#'; se mide de 1 a N de
     * izquierda a derecha.
     */
    private int x; // en la entrada de argumentos se le resta 1 para que vaya de 0 a
    // N - 1
    /**
     * Coordenada 'y' de la tabla donde se coloca la marca '#'; se mide de 1 a N de
     * arriba hacia abajo.
     */
    private int y; // en la entrada de argumentos se le resta 1 para que vaya de 0 a
    // N - 1
    /**
     * Cuando está en true indica la opción de trazar (-h) activada.
     */
    private boolean trazar;
    /**
     * Nombre (o nombre y ruta) del archivo que se usará como salida opcional.
     */
    String archivo;
    /**
     * Recogerá el texto que deberá salir en pantalla en caso de error en los
     * argumentos de entrada.
     */
    StringBuilder textoError;
    /**
     * Constante que representa el estado de error.
     */
    static final int ERROR = -1;
    /**
     * Constante que representa el estado de ayuda.
     */
    static final int AYUDA = -2;
    /**
     * Constructor de la clase baldosas.
```

```
     */
    Baldosas() {N = 8;}

    /**
     * Función que comprueba si un cuadrante está vacío (lleno de puntos '.')
     * @param a Objeto cuadrante a comprobar
     * @return true si el cuadrante está vacío
     */
    private boolean cuadranteVacio(Cuadrante c) {
        return(((c.lvi > y) || (c.lhi > x) || (c.lvs < y) || (c.lhs < x)) &&
        (Suelo[c.lvi][c.lhi] == '.') && (Suelo[c.lvs][c.lhi] == '.') &&
        (Suelo[c.lvi][c.lhs] == '.') && (Suelo[c.lvs][c.lhs] == '.'));
        // Comprueba las siguientes condiciones:
        // - Que la marca # esté fuera de los límites del cuadrante (ya que con esa
        // marca ya no estaría vacío)
        // - Que las cuatro esquinas del cuadrante estén vacías ('.'). Esto es así
        // porque el algoritmo recursivo que escribe los triminos en la tabla, en cada
        // recursión escribe los símbolos del trimino siguiente en las esquinas de los
        // hijos del cuadrante actual.
    }

    /**
     * Procedimiento que imprime la tabla Suelo en pantalla
     */
    private void escribeSuelo() {
        StringBuilder s = new StringBuilder(); // uso un objeto StringBuilder para
        // evitar el costo adicional del operador concatenación('+') de la clase
        // String, ya que en cada uso del operador se crea un nuevo objeto
        // StringBuilder y en el doble bucle no será rentable.
        for(int i = 0; i < N; i++) {
            for(int j = 0; j < N; j++) {
                s.append(Suelo[i][j]); s.append(" ");
            }
            s.append("\r\n");
        }
        s.append("\r\n\r\n\r\n\r\n");
        System.out.print(s);
    }

    /**
     * Función que devuelve el siguiente caracter a escribir en el próximo trimino
     * @param sigCaracter El caracter actual, el último que se ha escrito
     * @return El siguiente caracter, cambiando cíclicamente entre números, letras
     * mayúsculas y letras minúsculas
     */
    private void avanzaCaracter() { // Obtiene el siguiente caracter a escribir en la
        // tabla
        if (sigCaracter == '9') // primero los números del 0 al 9
            sigCaracter = 'A'; // luego las mayúsculas
        else if (sigCaracter == 'Z')
            sigCaracter = 'a'; // luego las minúsculas
        else if (sigCaracter == 'z')
            sigCaracter = '0'; // y vuelta a los números
        else
            sigCaracter ++; // si no cambia de tipo de caracter, solo avanza al
            // siguiente
    }

    /**
```

```
     * Procedimiento que escribe un caracter en la tabla Suelo, solo si el objeto
     * Cuadrante en cuestión está vacío
     * @param h Objeto Cuadrante del que tomaremos sus coordenadas más céntricas con
     * respecto a su cuadrante padre (EsquinaCentral), para escribir en ellas el
     * siguiente caracter en la tabla Suelo.
     */
    private void escribeLetra (Cuadrante h) { // Escribe un caracter en la esquina más
        // centrica de un cuadrante
        if (cuadranteVacio(h))
            Suelo[h.EsquinaCentral.Fila][h.EsquinaCentral.Columna] = sigCaracter;
    }

    /**
     * Procedimiento que, tomando un objeto Cuadrante, escribe el caracter actual en
     * tres de sus cuatro casillas, que están libres. Sólo se usa en los cuadrantes
     * básicos (de 2 x 2 casillas).
     * @param c Objeto Cuadrante a rellenar
     */
    private void escribeMinimoCuadrante(Cuadrante c) {
        for (int i = c.lvi; i <= c.lvs; i ++) {
            for (int j = c.lhi; j <= c.lhs; j ++) {
                if(Suelo[i][j] == '.')
                    Suelo[i][j] = sigCaracter;
            }
        }
        avanzaCaracter(); // como se invoca dentro de escribeCuadrante, hay que
        // actualizar el siguiente caracter que se escribirá.
    }

    /**
     * Procedimiento recursivo que, tomando un objeto Cuadrante, genera sus cuatro
     * hijos, escribe el caracter actual en las coordenadas EsquinaCentral de tres de
     * ellos que están vacíos, hasta haber escrito en los subcuadrantes de 2 x 2
     * casillas; el procedimiento escribeMinimoCuadrante se encarga de terminar de
     * rellenar estos cuadrantes básicos.
     * @param cuadr Objeto Cuadrante inicial
     */
    private void escribeCuadrante(Cuadrante cuadr) { // escribe los triminos en
        // la tabla de caracteres recursivamente
        if (cuadr.lhs - cuadr.lhi == 1) { // pasamos a la funcion sencilla que sólo
            // escribe un trimino si el cuadrante actual sólo tiene  2 x 2 casillas
            escribeMinimoCuadrante(cuadr);
            return;
        }
        Cuadrante h1 = cuadr.Hijo(1); // generamos el subcuadrante numero 1 del
        // cuadrante actual
        escribeLetra(h1); // escribimos el siguiente caracter en su esquina mas
        // cercana al centro de su cuadrante padre
        Cuadrante h2 = cuadr.Hijo(2); // y asi sucesivamente hasta el subcuadrante 4
        escribeLetra(h2);
        Cuadrante h3 = cuadr.Hijo(3);
        escribeLetra(h3);
        Cuadrante h4 = cuadr.Hijo(4);
        escribeLetra(h4);
        // aunque en lugar de todas estas instrucciones en secuencia se podrían haber
        // usado sentencias for o while, he preferido no hacerlo para no tener que
        // usar una estructura de datos iterativa, como un vector o una lista
        // enlazada, por el gasto adicional que ello conlleva. Esto es, claro está,
        // porque se trata solamente de una recursividad "de uno a cuatro"; en casos
```

```
        // mayores, el algoritmo podría tener demasiadas líneas que escribir y
        // demasiadas variables que asignar a los hijos del cuadante.
        avanzaCaracter(); // obtenemos el siguiente caracter a escribir
        if(trazar)
            escribeSuelo();
        escribeCuadrante(h1); // repetición recursiva de la misma función para cada
        // uno del los subcuadrantes obtenidos
        if(trazar)
            escribeSuelo();
        escribeCuadrante(h2);
        if(trazar)
            escribeSuelo();
        escribeCuadrante(h3);
        if(trazar)
            escribeSuelo();
        escribeCuadrante(h4);
    }

    /**
     * Procedimiento principal: rellena la tabla Suelo con triminos, cada trimino con
     * un caracter distinto, aunque se repiten cíclicamente los caracteres. Escribe
     * en el medio de salida la tabla al principio, cuando está vacía y al final
     * llena de triminos.
     */
    void baldosas() {
      if (N > 0) {
        Suelo = new char[N][N];
        for(int i = 0; i < N; i++) {  // inicializo la tabla llena de puntos, es decir
            // simbolos de vacío
            for(int j = 0; j < N; j++) {
                Suelo[i][j] = '.';
            }
        }
        Suelo[y][x] = '#'; // coloco la marca especial en las coordenadas dadas;
        // obsérvese la inversión de orden: la 'y' es la fila de la tabla, mientras
        // que la 'x' es la columna.
        escribeSuelo(); // muestro la tabla inicialmente vacía.
        if (N > 1) // la funcion sólo es necesaria si hay alguna casilla más que
        // rellenar a parte de '#'
            escribeCuadrante(new Cuadrante(N)); // invoco al algoritmo
        escribeSuelo();
      }
    }

    /**
     * Función que analiza los parámetros de entrada, tomando valores y buscando
     * errores. Al primer error localizado escapará de la función, ya que el programa
     * no se ejecutará. Si alguno de los parámetros, en cualquier posición, es la
     * opción de ayuda ('-h'), muestra la ayuda de sintaxis y no se hace nada más.
     * @param args El mismo vector de argumentos del método main de la clase trimino.
     * @return Un entero que representa un estado correcto de salida, o bien el estado
     * de ayuda, o bien el estado de error.
     */
    int examinaEntrada(String[] args) {
        // recorre un "diagrama de estados" en cada uno de los cuales evalúa uno de
        // los argumentos de la entrada; tiene un comportamiento similar a un AFD
        // (autómata finito determinista), pero en cada estado no evalúa caracteres
        // sino argumentos completos.
        for(int i = 0; i < args.length; i ++) { // Lo primero que hago, antes de nada,
```

```
            // es buscar en toda la cadena de entrada la opción '-h'; en caso de
            // encontrarlo, se muestra la ayuda y nada más; y se sale de la función.
            if(args[i].matches("-h"))
                return AYUDA;
        }

        textoError = new StringBuilder(); // inicializo aquí el objeto StringBuilder
        // que contendrá el texto del posible error.

        if (args.length < 2) { // si hay menos de dos argumentos (los obligatorios 'x'
            // e 'y')
            textoError.append("Error: faltan argumentos obligatorios");
            return ERROR; // salida inmediata por error
        }

        if (args.length > 5) { // como ya hemos descartado la opción de ayuda, lo
            // máximo serán cinco argumentos.
            textoError.append("Error: demasiados argumentos");
            return ERROR; // salida inmediata por error
        }

        int estado = 0, sigEstado = 0; int i = 0; int naux = -1; int arch = -2;
        while (i < args.length) {
            String entrada = args[i]; // argumento actual
            estado = sigEstado; // avanzar en el diagrama al siguiente estado
            // (recorrer el 'arco' entre estados)
            switch (estado) { // hago un switch para hacer lo que corresponda en cada
                // estado
                case 0: // estado inicial: el primer argumento de la cadena puede ser
                // bien '-h' o bien la coordenada x. Evalúo los dos arcos:
                    if (entrada.matches("-t")) { // opción de trazar
                        trazar = true;
                        sigEstado = 1; // Arco a la coordenada x
                    }
                    else if((x = Funciones.compruebaEntero(entrada) - 1) >= 0)
                    // coordenada x: le restamos 1, para adaptarlo a la tabla, y
                    // comprobamos que sea un entero y además mayor o igual que 0.
                        sigEstado = 2; // arco a la coordenada y
                    else { // error
                        textoError.append("Error en el argumento ");
                        textoError.append((i + 1)); textoError.append(": ");
                        textoError.append(entrada); textoError.append(" no es ");
                        // muestro el argumento motivo del error
                        if (entrada.matches("-?\\d+.*"))
                        // si los primeros caracteres son números con o sin signo,
                        // suponemos que hubo un error al escribir la coordenada x.
                            textoError.append("una coordenada x válida");
                        else
                            textoError.append("una opción válida"); // en caso
                            // contrario, suponemos que hay un error en las opciones.
                        return ERROR; // salida inmediata por error ("el autómata no
                        // acepta la cadena de entrada")
                    }
                    break;

                case 1: // coordenada x
                    if((x = Funciones.compruebaEntero(entrada) - 1) >= 0)
                        sigEstado = 2; // arco a la coordenada y
                    else {
```

```
                    textoError.append("Error en el argumento ");
                    textoError.append((i + 1)); textoError.append(": ");
                    textoError.append(entrada); textoError.append(" no es ");
                    textoError.append("una coordenada x válida");
                    return ERROR; // salida inmediata por error
                }
                break;

            case 2: // coordenada y: posible salida correcta
                if((y = Funciones.compruebaEntero(entrada) - 1) >= 0)
                    sigEstado = 3; // arco a la dimensión
                else {
                    textoError.append("Error en el argumento ");
                    textoError.append((i + 1)); textoError.append(": ");
                    textoError.append(entrada); textoError.append(" no es ");
                    textoError.append("una coordenada y válida");
                    return ERROR; // salida inmediata por error
                }
                if (i == args.length - 1) { // si éste es el último argumento
                    if ( x >= N) { // coordenada x mayor que la dimensión
                        textoError.append("Error en el argumento ");
                        textoError.append(i); textoError.append(": ");
                        textoError.append(args[i - 1]); textoError.append(" es ");
                        textoError.append("una coordenada 'x' mayor que la ");
                        textoError.append("dimensión");
                        return ERROR; // salida inmediata por error
                    }
                    else if ( y >= N) { // coordenada y mayor que la dimensión
                        textoError.append("Error en el argumento ");
                        textoError.append((i + 1)); textoError.append(": ");
                        textoError.append(args[i]); textoError.append(" es ");
                        textoError.append("una coordenada 'y' mayor que la ");
                        textoError.append("dimensión");
                        return ERROR; // salida inmediata por error
                    }
                }
                break;

            case 3:
            //este argumento puede bien ser la dimensión o bien el archivo de
            // salida, ambos posibles salidas correctas.
                if (((naux = Funciones.compruebaEntero(entrada)) >= 0) &&
                (naux > y) && (naux > x) && (Funciones.potenciaDeDos(naux))) {
                    // comprobamos que sea un entero mayor o igual que 0, que
                    // también sea mayor que las coordenadas x e y de entrada
                    // y también que sea potencia de 2.
                    N = naux;
                    sigEstado = 4; // arco al archivo
                }
                else if ((arch = Funciones.patronArchivo(entrada)) >= 0) {
                    // comprobamos si es un archivo de salida válido
                    // (bien existe ya o bien tiene un formato correcto);
                    // si lo es, significa que no se tomó argumento
                    // de dimensión, y por tanto no se comprobó:
                    if ( x >= N) { // coordenada x mayor que la dimensión
                        textoError.append("Error en el argumento ");
                        textoError.append((i - 1)); textoError.append(": ");
                        textoError.append(args[i - 2]); textoError.append(" es ");
                        textoError.append("una coordenada 'x' mayor que la ");
```

```
                        textoError.append("dimensión");
                        return ERROR; // salida inmediata por error
                    }
                    else if ( y >= N) { // coordenada y mayor que la dimensión
                        textoError.append("Error en el argumento ");
                        textoError.append(i); textoError.append(": ");
                        textoError.append(args[i - 1]); textoError.append(" es ");
                        textoError.append("una coordenada 'y' mayor que la ");
                        textoError.append("dimensión");
                        return ERROR; // salida inmediata por error
                    }
                    archivo = entrada;
                    sigEstado = 5; // arco al estado final
                }
                else {
                    textoError.append("Error en el argumento ");
                    textoError.append((i + 1)); textoError.append(": ");
                    textoError.append(entrada); textoError.append(" no es ");
                    if (entrada.matches("-?\\d+.*")) {
                        // si los primeros caracteres son números con o sin signo,
                        // suponemos que hubo un error al escribir la dimensión.
                        textoError.append("un número válido para la dimensión");
                        textoError.append(" de la tabla");
                    }
                    else
                        textoError.append("un archivo válido"); // en caso
                        // contrario, suponemos que el error está en el nombre del
                        // archivo
                    return ERROR; // salida inmediata por error
                }
                break;

            case 4: // archivo: posible salida correcta
                if ((arch = Funciones.patronArchivo(entrada)) >= 0) {
                    // comprobamos si es un archivo de salida válido
                    // (bien existe ya o bien tiene un formato correcto).
                    archivo = entrada;
                    sigEstado = 5; // arco al estado final
                }
                else {
                   textoError.append("Error en el argumento ");
                   textoError.append((i + 1)); textoError.append(": ");
                   textoError.append(entrada); textoError.append(" no es ");
                   textoError.append("un archivo válido");
                   return ERROR; // salida inmediata por error
                }
                break;

            case 5: // estado final: si este arco se recorre, significa que no hay
            // demasiados argumentos (eso se comprueba antes de entrar en el
            // while); pero después del último argumento posible (el archivo de
            // salida) todavía hay más. Es un arco que lleva a un estado de error.
                textoError.append("Error: sobran argumentos al final: ");
                textoError.append(entrada);
                textoError.append(" y los que le siguen");
                return ERROR; // salida inmediata por error
        }
        i ++; // pasamos al siguiente argumento
    }
```

```
        if ((arch == 0) && !Funciones.leerConsola(archivo).equals("s")) {
            // preguntamos si se quiere sobreescribir el archivo
            textoError.append("No se usará el archivo");
            return ERROR; // aprovecho el estado de error para este caso
        }
        return estado;
    }

    /**
     * Procedimiento que escribe el encabezado del archivo de salida, con los valores
     * de los argumentos dados en la entrada.
     */
    void titulo() {
        StringBuilder s = new StringBuilder();
        s.append("SALIDA DEL COMANDO TRIMINO\r\n");
        s.append("==========================\r\n\r\n");
        s.append("Los argumentos fueron:\r\n");
        s.append("trazar = "); s.append(trazar); s.append("; x = ");
        s.append((x + 1)); s.append("; y = "); s.append((y + 1));
        s.append("; dimensión = "); s.append(N);
        s.append("; fichero = "); s.append(archivo); s.append("\r\n\r\n\r\n\r\n");
        System.out.print(s);
    }
}
```