```java
package Ejercicio04;

import java.io.BufferedReader;
import java.io.Console;
import java.io.File;
import java.io.FileFilter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.Writer;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class U06E04 {

    //lo usaremos para filtrar archivos gps
    public static FileFilter filter = new FileFilter() {
        public boolean accept(File file) {
                String tmp = file.getName().toLowerCase();
                if (tmp.endsWith(".gps")){
                    return true;
                }
                    return false;
                }
            };

    public static void main(String[] args) {

        //variables locales
        String fichero; //almacena el fichero
        String origen ="origen";    //almacena la ciudad de origen
        String destino ="destino";  //almacena la ciudad de destino
        String carpetaActual;   //almacena la carpeta actual

        Scanner stdin = new Scanner(System.in); //entrada por teclado

        boolean flag=false;

        //preguntamos por el fichero, en este caso suponemos que lo tenemos en el directorio actual
        System.out.print("Fichero (.GPS): ");
        fichero = stdin.nextLine();

        //suponiendo que vamos a trabajar desde el actual directorio, lo almacenamos
        carpetaActual = System.getProperty("user.dir");

        //creamos el objeto File
        File d = new File(carpetaActual);

        //vemos si hay ficheros con la extension gps dentro del directorio
        File[] listaFicheros = d.listFiles(filter);
```

```
52
53          //si hay archivos gps, buscamos el que queremos
54          if(listaFicheros.length > 0) {
55              for(int i = 0; i < listaFicheros.length; i++) {
56                  if(listaFicheros[i].getName().toLowerCase().equals(fichero.toLowerCase())) flag=true;
57              }
58          }
59
60          //si no hemos encotrado nada finalizamos el programa
61          if(flag == false) {
62              System.out.println("No hemos encontrado el fichero " + fichero);
63              System.exit(0);
64          }
65
66          //si existe el fichero continuamos con el programa
67
68          //mientras el origen y el destino no esten vacios
69          while(!origen.isEmpty() && !destino.isEmpty()) {
70
71              //preguntamos por la ciudad de origen
72              System.out.print("Origen: ");
73              origen = stdin.nextLine();
74
75              //preguntamos por la ciudad de destino
76              System.out.print("Destino: ");
77              destino = stdin.nextLine();
78
79              //si origen y destino no estan vacias
80              if(!origen.isEmpty() && !destino.isEmpty()) {
81                  Grafo unGrafo = new Grafo(fichero);
82                  String cadena = unGrafo.caminoMasCorto(origen, destino);
83                  System.out.println(cadena);
84              }//fin del if
85
86          }//fin del while
87
88      }//fin del main
89
90  }//fin de la clase
91
92  //*****************************************clase Grafo*********************************************
93  class Grafo{
94
95      public static String[] datos;   //para almacenar los datos del fichero
96      public static int contador = 0;     //las posiciones del array de datos
97      public static int lineasTotales = 0; //lineas totales del fichero
98
99      //variables miembro
100     private int nV; //numero de vertices
101     private int nA; //numero de aristas
102     private int[][] tA; //tabla de adyacencias
```

```
103     private String[] nombreVertice; //nombre de los vertices
104
105     //constructor
106     public Grafo(String nomFichero) {
107
108         //declaramos una variable BufferedReader
109         BufferedReader br = null;
110
111         try {
112             //crear un objeto BufferedReader al que se le pasa
113             //un objeto FileReader con el nombre del fichero
114             br = new BufferedReader(new FileReader(nomFichero));
115
116
117             //leer la primera linea, guardando en un string
118             String texto = br.readLine();
119
120             //repetir mientras no se llegue al final del fichero
121             while(texto != null) {
122
123                 //contamos las lineas del fichero y leemos
124                 contador++;
125                 texto = br.readLine();
126
127
128             }//fin del while
129
130             //almacenamos las lineas totales leidas
131             lineasTotales = contador;
132
133             //conocido el numero de lineas construimos el array de datos
134             datos = new String[contador];
135
136             //ponemos el contador a cero
137             contador = 0;
138
139             //creamos un bufferedReader
140             //con el metodo br.mark y br.reset deberia haber reiniciado el buffer anterior
141             //pero me da error al aplicarlos
142             BufferedReader br2 = new BufferedReader(new FileReader(nomFichero));
143
144             //volvemos a correr el fichero rellenando de nuevo los datos
145             texto = br2.readLine();
146             while(texto != null) {
147
148                 datos[contador]=texto.trim();
149                 contador++;
150                 texto = br2.readLine();
151
152
153             } //fin del while
```

```
154
155             //ahora vamos rellenando las variables miembro
156             contador=0;
157
158             //la primera linea indica el numero de indices
159             nV = Integer.parseInt(datos[0]);
160             contador++;
161
162             //la segunda linea indica el numero de aristas
163             nA = Integer.parseInt(datos[1]);
164             contador++;
165
166             //creamos el array de vertices
167             nombreVertice = new String[nV];
168
169             //creamos el array de adyacentes
170             tA = new int[nV][nV];
171
172             //rellenamos el array de vertices
173             for(int i = 2; i < nV+2; i++) {
174                 int pos = datos[i].indexOf(" "); //el espacio separa el nº vertice del nombre de vertice
175                 nombreVertice[i-2] = datos[i].substring(pos+1);
176                 contador++;
177             }
178
179             //rellenamos la tabla de adyacentes
180             for(int i = contador; i < lineasTotales; i++) {
181
182                 //como son 3 campos, cogemos la primera y ultima posicion del separador
183                 int pos1 = datos[i].indexOf(" ");
184                 int pos2 = datos[i].lastIndexOf(" ");
185
186                 //volcamos cada campo en variables
187                 int temp1 = Integer.parseInt(datos[i].substring(0, pos1));
188                 int temp2 = Integer.parseInt(datos[i].substring(pos1+1, pos2));
189
190                 //los valores no pueden ser negativos o mayores que el numero de vertices
191                 if(temp1 >= nV || temp1 < 0 || temp2 >= nV || temp2 < 0) {
192                     throw new IllegalArgumentException ("Error formato en " + nomFichero + " linea " + i);
193                 }
194
195                 //el primer campo no puede igual que el segundo campo
196                 if(temp1 == temp2) {
197                     throw new IllegalArgumentException ("Error formato en " + nomFichero + " linea " + i);
198                 }
199
200                 //si esta todo correcto
201                 tA[temp1][temp2] = Integer.parseInt(datos[i].substring(pos2+1));
202                 tA[temp2][temp1] = Integer.parseInt(datos[i].substring(pos2+1));
203
204             }
```

```
205             //tratamiento de errores
206             } catch (FileNotFoundException e) {
207                 System.out.println("Error: Fichero no encontrado");
208                 System.out.println(e.getMessage());
209             } catch (Exception e) {
210                 System.out.println("Error de lectura del fichero");
211                 System.out.println(e.getMessage());
212             }
213             finally {
214                 try {
215                     if(br != null)
216                         br.close();
217                 } catch (Exception e) {
218                     System.out.println("Error al cerrar el fichero");
219                     System.out.println(e.getMessage());
220                 }
221             }
222 
223         } //fin del constructor
224 
225         //metodos
226 
227         public String caminoMasCorto(String origen, String destino) {
228 
229             //variables locales
230             boolean cOrigen = false;
231             boolean cDestino = false;
232             String cadena = "";
233             int d=0,o=0; //numero de vertice de origen y destino
234 
235             //comprobamos que la ciudad de origen estan dentro de los datos
236             for(int i = 0; i < nombreVertice.length; i++) {
237                 if(nombreVertice[i].toLowerCase().equals(origen.toLowerCase())) {
238                     cOrigen = true;
239                     o = i;
240                 }
241                 if(nombreVertice[i].toLowerCase().equals(destino.toLowerCase())) {
242                     cDestino = true;
243                     d = i;
244                 }
245             }
246 
247             if(cOrigen == false && cDestino == false) cadena = cadena + "ciudad de origen " + origen + " y destino " + destino +
            " no encontradas.";
248             else if(cOrigen == false) cadena = cadena + "ciudad de origen " + origen +  " no encontrada.";
249             else if(cDestino == false) cadena = cadena + "ciudad de destino " + destino +  " no encontrada.";
250             else if(origen.equals(destino)) cadena = cadena + origen +"(0) -> " + origen;
251             //llamada al metodo
252             else cadena = Dijsktra(o,d);
253 
254             return cadena;
```

```
255       } //fin del metodo
256
257       //metodo dijsktra
258       private String Dijsktra(int ori, int des) {
259           //variables locales
260           ArrayList S = new ArrayList();
261           int[] D = new int[nV];
262           int[] P = new int[nV];
263           int menos = 0;
264           //int pos = 0;
265           int tramo;
266           int distancia=0;
267           int w=-1;
268           int v=-1;
269           String camino = "";
270
271           //inicialmente S contiene el vertice origen(x)
272           S.add(ori);
273
274           //para cada vertice
275           for(int i = 0; i < nV; i++) {
276               //si i no es el origen
277               if(i != ori) {
278                   //si hay arista entre i y origen
279                   if(tA[ori][i] > 0) D[i] = tA[ori][i];
280                   //si no hay arista ponemos el maximo valor entero
281                   else D[i] = Integer.MAX_VALUE;
282               }
283           } //fin del for
284
285           //explorar el grafo
286           for(int i = 0; i < nV; i++) {
287               if(S.size() == nV) break;
288               //corremos el array de distancias, buscando la posicion que no esté en S y que tenga la menor distancia
289               menos = Integer.MAX_VALUE; //inicializamos la variable para ver que posicion tiene menos
290               for(int j = 0; j < nV; j++) {
291                   //comprobamos que j no está en S
292                   if(!S.contains(j)) {
293                   //comprobamos que tenga el valor mas bajo
294                       if(D[j] < menos) {
295                           menos = D[j];
296                           w = j; //asignamos a v la posicion en D[i]
297                       }
298                   }
299               }
300               //añadimos el vertice w al conjunto S
301               S.add(w);
302
303               //volvemos a correr V-S, o sea todos los vertices que no esten en S
304               for(int j = 0; j < nV; j++) {
305                   //comprobamos que j no está en S
```

```
306                 if(!S.contains(j)) {
307                 //comprobamos que hay una arista w - j
308                     if(tA[w][j] > 0) {
309                         v = j;
310                         if (D[v] + tA[w][v] < D[v]) {
311                             D[v] = D[w] + tA[w][v];
312                             P[v] = w;
313                         }
314                     }
315                 }
316             }
317
318
319
320         } //fin exploracion del grafo
321
322         //test de arrays
323         /*for(int i = 0; i<nV; i++) System.out.print(i + ":" + D[i] + " ");
324         System.out.println();
325         for(int i = 0; i<nV; i++) System.out.print(i + ":" + P[i] + " ");
326         System.out.println();*/
327
328         distancia = 0;
329
330         camino = nombreVertice[des];
331
332
333         v = des;
334
335
336         while (v != ori){
337             //en el vertice previo al destino se me va a 0, asi que añado esto
338             if(P[v]==0) {
339                 tramo = tA[v][ori];
340                 distancia = distancia + tramo;
341                 camino = nombreVertice[ori] + " (+" + tramo + ") -> " + camino;
342                 v=ori;
343             } else {
344                 tramo = tA[v][P[v]];
345                 distancia = distancia + tramo;
346                 camino = nombreVertice[P[v]] + " (+" + tramo + ") ->" + camino;
347                 v = P[v];
348             }
349         }
350
351         camino ="  " + camino + " Total: " + distancia;
352         return camino;
353
354
355
356     } //fin de dijsktra
```

```
357
358
359
360   } //fin de la clase grafo
361
```