



MANUAL DEL DESARROLLADOR

POR

JAVIER SANTIAGO BORBÓN BORBÓN Y JOSÉ ANDRÉS SANABRIA ARIAS

## Tabla de contenido

|                                  |    |
|----------------------------------|----|
| 1. INTRODUCCIÓN .....            | 2  |
| 2. DIAGRAMA UML.....             | 3  |
| 3. ELEMENTOS DEL PROYECTO.....   | 4  |
| 3.1. Archivo.java.....           | 5  |
| 3.2. FiguraBorrador.java.....    | 9  |
| 3.3. FiguraCuadrado.java.....    | 10 |
| 3.4. FiguraL.java.....           | 11 |
| 3.5. FiguraLinea.java.....       | 12 |
| 3.6. FiguraLnv.java.....         | 13 |
| 3.7. FiguraS.java.....           | 14 |
| 3.8. FiguraT.java.....           | 15 |
| 3.9. FiguraZ.java.....           | 16 |
| 3.10. Forma.java.....            | 17 |
| 3.11. Interfaz.java.....         | 19 |
| 3.12. Main.java.....             | 25 |
| 3.13. Pieza.java.....            | 26 |
| 3.14. Sonido.java.....           | 27 |
| 3.15. Tablero.java.....          | 28 |
| 3.16. VentanaPrincipal.java..... | 35 |
| 4. Index.html.....               | 39 |

## INTRODUCCIÓN

En este documento usted encontrará información que le ayudará a comprender mejor este proyecto realizado en Java, específicamente en el IDE de Eclipse. Con una versión de Java SE-13.

Así como se evidenció en la tabla de contenido, se le mostrará primero el diagrama UML, donde podrá evidenciar las clases utilizadas y la relación que tienen estas. Se le mostrará el índice de los archivos en el IDE y por último se le anexará cada una de las clases con su respectiva descripción.

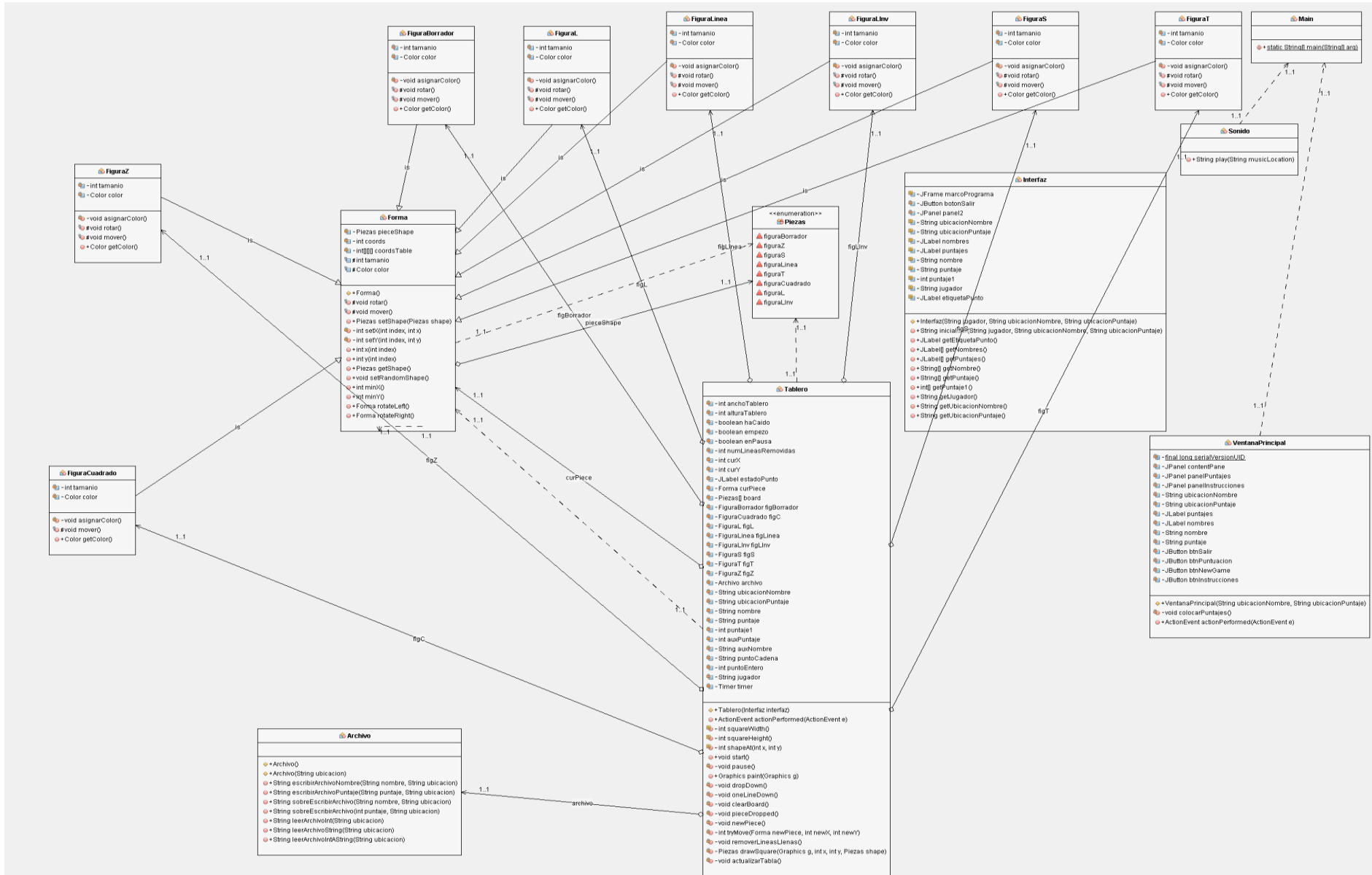
Esperemos le sea de utilidad este manual. Y que pueda modificar el código de la mejor manera.

Adicional a esto, hemos documentado el programa, usted puede acceder a la información abriendo el archivo “Index” ubicado en la carpeta doc, en este documento se le especificará más.

Para mayor información, inquietudes y dudas, por favor escribanos al siguiente correo:

[mjoasanabriaa@correo.udistrital.edu.co](mailto:mjoasanabriaa@correo.udistrital.edu.co)

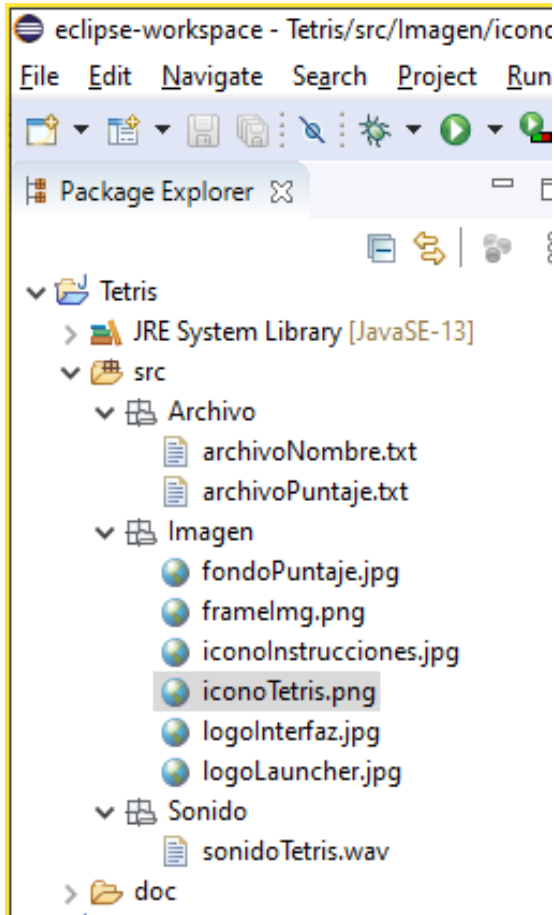
## ARCHIVO UML



Para mejor visualización, este documento se encuentra en la misma carpeta de este documento.

## ELEMENTOS DEL PROYECTO

Este proyecto consta de los siguientes elementos:



En la imagen anterior se ven las carpetas de los recursos que utiliza el juego.

La carpeta Archivo contiene los archivos relacionados al nombre y al puntaje.

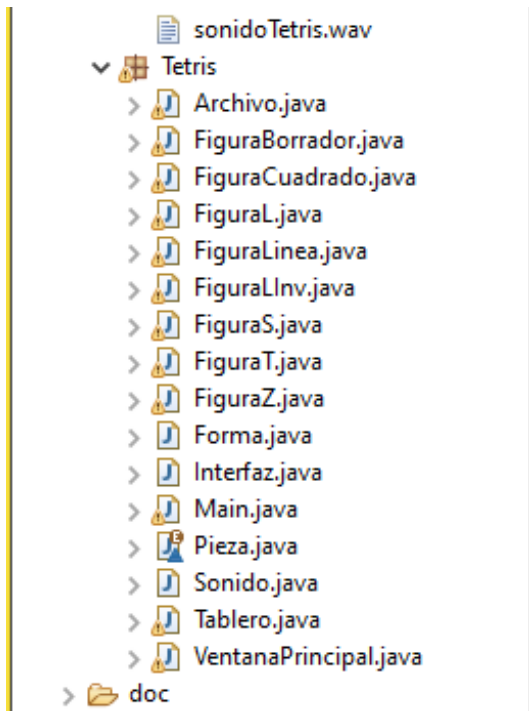
- archivoNombre.txt: Guarda los nombres en un documento de texto.
- archivoPuntaje.txt: Guarda los puntos (como String) en un documento de texto.

La carpeta Imagen contiene las diferentes imágenes que contiene el programa.

- fondoPuntaje.jpg: Corresponde a la imagen del fondo de la tabla de puntuación.
- frameImg.png: Corresponde a los bordes que tiene el tablero del juego.
- iconoInstrucciones.jpg: Corresponde a la imagen del panel de Instrucciones.
- iconoTetris.png: Corresponde al icono que aparece en la barra de tareas y al archivo .jar
- logoInterfaz.jpg: Corresponde a la imagen del panel derecho del juego.
- logoLauncher.jpg: Corresponde a la imagen que aparece en la ventana inicial.

La carpeta Sonido contiene el audio del juego: sonidoTetris.wav.

La carpeta: doc contiene todo lo relacionado a la documentación del proyecto



El paquete Tetris contiene toda la lógica del juego.

El proyecto cuenta con tres clases encargadas de la parte gráfica, estas son: VentanaPrincipal.java, Tablero.java e Interfaz.java

Una clase encargada de la gestión del sonido: Sonido.java

Una clase encargada de la gestión de Archivos: Archivo.java

Una clase encargada de la ubicación de las dos clases anteriores y de instanciar a la ventana inicial: Main.java

La clase Pieza.java encargada de enlistar las ocho figuras (siete del Tetris y una que borra).

Y las demás clases, encargadas de la lógica del juego, a continuación, se detallarán cada una de las clases.

## Archivo.java

```
1 package Tetris;
2
3 * NOMBRE DEL PROYECTO: TETRIS
4
5
6
7
8
9
10
11
12
13
14
15
16
17 import java.io.BufferedReader;
18
19
20
21
22
23
24
25
26
27
28
29
30 public class Archivo {
31
32     public Archivo() {
33
34     }
35
36     // Método constructor, el cual crea el archivo, lo abre y cierra.
37     public Archivo(String ubicacion) {
38
39         DataOutputStream archivo = null;
40
41         try {
42
43             archivo = new DataOutputStream(new FileOutputStream(ubicacion, true));
44
45             archivo.close();
46
47         } catch (FileNotFoundException fnfe) {
48
49         } catch (IOException ioe) {
50
51         }
52     }
53
54
55     // Métodos que se utilizaron para crear el primer archivo
56     public void escribirArchivoNombre(String nombre[], String ubicacion) {
57
58         DataOutputStream archivo = null;
59
60         try {
61
62             archivo = new DataOutputStream(new FileOutputStream(ubicacion, true));
63
64             for (int i = 0; i < 10; i++) {
65
66                 archivo.writeUTF(nombre[i]);
67
68             }
69
70             archivo.close();
71
72         } catch (FileNotFoundException fnfe) {
73
74         } catch (IOException ioe) {
75
76         }
77     }
78 }
```

```

80 // Métodos que se utilizaron para crear el primer archivo
81 public void escribirArchivoPuntaje(String puntaje[], String ubicacion) {
82
83     int[] puntaje1 = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
84
85     DataOutputStream archivo = null;
86
87     try {
88
89         archivo = new DataOutputStream(new FileOutputStream(ubicacion, true));
90
91         for (int i = 0; i < 10; i++) {
92             puntaje1[i] = Integer.parseInt(puntaje[i]);
93         }
94
95         for (int i = 0; i < 10; i++) {
96             archivo.writeInt(puntaje1[i]);
97         }
98
99         archivo.close();
100     } catch (FileNotFoundException fnfe) {
101     } catch (IOException ioe) {
102     }
103 }
104
105 // Método que hace la persistencia de los puntajes al final del juego
106 public void sobreEscribirArchivo(String nombre[], String ubicacion) {
107
108     // DataOutputStream archivo = null;
109
110     try {
111
112         // archivo = new DataOutputStream( new FileOutputStream(ubicacion,true));
113
114         RandomAccessFile archivo = new RandomAccessFile(ubicacion, "rw");
115
116         archivo.seek(0);
117
118         for (int i = 0; i < 10; i++) {
119             archivo.writeUTF(nombre[i]);
120         }
121
122         archivo.close();
123     } catch (FileNotFoundException fnfe) {
124     } catch (IOException ioe) {
125     }
126 }
127
128 }
129
130
131
132
133
134
135
136
137
138
139
140

```



```

141
142 // Método que hace la persistencia de los puntajes al final del juego
143 public void sobreEscribirArchivo(int puntaje[], String ubicacion) {
144
145     try {
146
147         RandomAccessFile archivo = new RandomAccessFile(ubicacion, "rw");
148
149         archivo.seek(0);
150
151         for (int i = 0; i < 10; i++) {
152
153             archivo.writeInt(puntaje[i]);
154
155         }
156
157         archivo.close();
158
159     } catch (FileNotFoundException fnfe) {
160
161     } catch (IOException ioe) {
162
163     }
164
165 }
166
167 // Método que leen el archivo, y transfieren sus datos a variables para su uso.
168 public int[] leerArchivoInt(String ubicacion) {
169
170     DataInputStream archivo = null;
171     int[] puntaje = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
172
173     try {
174
175         archivo = new DataInputStream(new FileInputStream(ubicacion));
176
177         for (int i = 0; i < 10; i++) {
178
179             puntaje[i] = archivo.readInt();
180
181         }
182
183     } catch (FileNotFoundException fnfe) {
184
185     } catch (IOException ioe) {
186
187     }
188
189     return puntaje;
190
191 }
192
193 // Método que leen el archivo, y transfieren sus datos a variables para su uso.
194 public String[] leerArchivoString(String ubicacion) {
195
196     DataInputStream archivo = null;
197     String[] nombre = { null, null, null, null, null, null, null, null, null, null };
198
199     try {
200
201         archivo = new DataInputStream(new FileInputStream(ubicacion));
202
203         for (int i = 0; i < 10; i++) {
204
205             nombre[i] = archivo.readUTF();
206
207         }
208
209     } catch (FileNotFoundException fnfe) {
210
211     } catch (IOException ioe) {
212
213     }
214
215     return nombre;
216
217 }

```

```

218
219 // Método que leen el archivo, y tranfieren sus datos a variables para su uso.
220 public String[] leerArchivoIntAString(String ubicacion) {
221     int puntajeEntero[] = leerArchivoInt(ubicacion);
222
223     String[] puntaje = { null, null, null, null, null, null, null, null, null, null };
224
225     for (int i = 0; i < 10; i++) {
226         puntaje[i] = Integer.toString(puntajeEntero[i]);
227     }
228
229     return puntaje;
230 }
231
232 }
233
234 }
235
236 }

```

FiguraBorrador.java

```

1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraBorrador extends Forma {
6
7     private int tamaño;
8     private Color color;
9
10    private void asignarColor(){
11        // Asignación de colores a las figuras
12        Color negro = new Color(0, 0, 0); // negro
13
14        this.color=negro;
15    }
16
17    public Color getColor() {
18        asignarColor();
19        return color;
20    }
21
22
23 }
24

```

FiguraCuadrado.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraCuadrado extends Forma {
6
7     private int tamano;
8     private Color color;
9
10    private void asignarColor(){
11        // Asignación de colores a las figuras
12        Color azul = new Color(12, 67, 194); // Cuadrado
13        this.color=azul;
14    }
15
16    protected void mover() {
17    }
18
19    public Color getColor() {
20        asignarColor();
21        return color;
22    }
23
24
25 }
```

FiguraL.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraL extends Forma {
6
7     private int tamano;
8     private Color color;
9     private void asignarColor(){
10         // Asignación de colores a las figuras
11
12         Color magenta = new Color(205, 46, 150); // L
13         this.color=magenta;
14     }
15     protected void rotar() {
16
17     }
18
19     protected void mover() {
20
21     }
22     public Color getColor() {
23         asignarColor();
24         return color;
25     }
26
27
28 }
```

FiguraLinea.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraLinea extends Forma {
6
7     private int tamano;
8     private Color color;
9
10    private void asignarColor(){
11        // Asignación de colores a las figuras
12        Color rojo = new Color(216, 29, 43); // Linea
13        this.color=rojo;
14    }
15    protected void rotar() {
16
17    }
18
19    protected void mover() {
20
21    }
22    public Color getColor() {
23        asignarColor();
24        return color;
25    }
26
27
28 }
```

FiguraLInv.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraLInv extends Forma {
6
7     private int tamano;
8     private Color color;
9     private void asignarColor(){
10         // Asignación de colores a las figuras
11
12         Color blanco = new Color(255, 255, 255); // L invertida
13         this.color=blanco;
14     }
15     protected void rotar() {
16
17     }
18
19     protected void mover() {
20
21     }
22     public Color getColor() {
23         asignarColor();
24         return color;
25     }
26
27
28
29 }
30
```

FiguraS.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraS extends Forma {
6
7     private int tamano;
8     private Color color;
9     private void asignarColor(){
10         // Asignación de colores a las figuras
11
12         Color verde = new Color(5, 162, 11); // S
13         this.color=verde;
14     }
15     protected void rotar() {
16
17     }
18
19     protected void mover() {
20
21     }
22     public Color getColor() {
23         asignarColor();
24         return color;
25     }
26
27
28 }
```



FiguraT.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraT extends Forma {
6
7     private int tamaño;
8     private Color color;
9
10    private void asignarColor() {
11        // Asignación de colores a las figuras
12
13        Color marron = new Color(158, 114, 15); // T
14        this.color = marron;
15    }
16
17    protected void rotar() {
18
19    }
20
21    protected void mover() {
22
23    }
24
25    public Color getColor() {
26        asignarColor();
27        return color;
28    }
29
30 }
```

FiguraZ.java

```
1 package Tetris;
2
3 import java.awt.Color;
4
5 public class FiguraZ extends Forma {
6
7     private int tamano;
8     private Color color;
9     private void asignarColor(){
10         // Asignación de colores a las figuras
11         Color cyan = new Color(0, 132, 144); // Z
12
13         this.color=cyan;
14     }
15     protected void rotar() {
16
17     }
18
19     protected void mover() {
20
21     }
22     public Color getColor() {
23         asignarColor();
24         return color;
25     }
26
27
28
29 }
```

## Forma.java

```
1 package Tetris;
2
3
4 * NOMBRE DEL PROYECTO: TETRIS
5
6
7
8
9
10
11
12
13
14
15
16
17 import java.util.Random;
18
19
20
21 public class Forma {
22     // Variables fundamentales para manipular las piezas en el tablero, mediante su
23     // forma y coordenadas.
24
25     private Pieza pieceShape;
26     private int coords[][];
27     private int[][][] coordsTable;
28
29     protected int tamano;
30     protected Color color;
31
32     protected void rotar() {
33         //MÉTODO POR EL CUAL LAS FICHAS ROTAN
34     }
35
36     protected void mover() {
37         //MÉTODO POR EL CUAL LAS FICHAS SE MUEVEN
38     }
39
40     // Método constructor que genera un contorno de pieza en el tablero. Ya en el
41     // tablero toma una forma gracias al random.
42     public Forma() {
43
44         coords = new int[4][2];
45         setShape(Pieza.figuraBorrador);
46
47     }
48
49     // Método fundamental que crea una pieza con su forma, ubicacion.
50     public void setShape(Pieza shape) {
51
52         coordsTable = new int[][][] { { { 0, 0 }, { 0, 0 }, { 0, 0 }, { 0, 0 } },
53             { { 0, -1 }, { 0, 0 }, { -1, 0 }, { -1, 1 } }, { { 0, -1 }, { 0, 0 }, { 1, 0 }, { 1, 1 } },
54             { { 0, -1 }, { 0, 0 }, { 0, 1 }, { 0, 2 } }, { { -1, 0 }, { 0, 0 }, { 1, 0 }, { 0, 1 } },
55             { { 0, 0 }, { 1, 0 }, { 0, 1 }, { 1, 1 } }, { { -1, -1 }, { 0, -1 }, { 0, 0 }, { 0, 1 } },
56             { { 1, -1 }, { 0, -1 }, { 0, 0 }, { 0, 1 } } };
57
58         for (int i = 0; i < 4; i++) {
59             for (int j = 0; j < 2; ++j) {
60                 coords[i][j] = coordsTable[shape.ordinal()][i][j]; //CoordsTable guarda el #de la ficha segun la clase Piezas
61             }
62         }
63         pieceShape = shape;
64
65     }
66
67     private void setX(int index, int x) {
68         coords[index][0] = x;
69     }
70
71     private void setY(int index, int y) {
72         coords[index][1] = y;
73     }
74 }
```

```

75 public int x(int index) {
76     return coords[index][0];
77 }
78
79 public int y(int index) {
80     return coords[index][1];
81 }
82
83 public Pieza getShape() {
84     return pieceShape;
85 }
86
87 // Método por el cual crea una pieza aleatoria de 7 posibles.
88 public void setRandomShape() {
89     Random r = new Random();
90     int x = Math.abs(r.nextInt()) % 7 + 1;
91     Pieza[] values = Pieza.values();
92     setShape(values[x]);
93 }
94
95 // Método que devuelve el mínimo de la ficha en X.
96 public int minX() {
97     int m = coords[0][0];
98     for (int i = 0; i < 4; i++) {
99         m = Math.min(m, coords[i][0]);
100     }
101     return m;
102 }
103
104 // Método que devuelve el mínimo de la ficha en Y.
105 public int minY() {
106     int m = coords[0][1];
107     for (int i = 0; i < 4; i++) {
108         m = Math.min(m, coords[i][1]);
109     }
110     return m;
111 }
112
113 // Método que rota la pieza
114 public Forma rotateLeft() {
115     if (pieceShape == Pieza.figuraCuadrado)
116         return this;
117
118     Forma result = new Forma();
119     result.pieceShape = pieceShape;
120
121     for (int i = 0; i < 4; ++i) {
122         result.setX(i, y(i));
123         result.setY(i, -x(i));
124     }
125     return result;
126 }
127
128 // Método que rota la pieza
129 public Forma rotateRight() {
130     if (pieceShape == Pieza.figuraCuadrado)
131         return this;
132
133     Forma result = new Forma();
134     result.pieceShape = pieceShape;
135
136     for (int i = 0; i < 4; ++i) {
137         result.setX(i, -y(i));
138         result.setY(i, x(i));
139     }
140     return result;
141 }
142 }

```

## Interfaz.java

```
1 package Tetris;
2
3
4 * NOMBRE DEL PROYECTO: TETRIS
5
6
7
8
9
10
11
12
13
14
15
16
17 import java.awt.Color;
18
19
20
21
22
23
24
25
26
27
28
29
30 public class Interfaz {
31
32     // Objetos para la interfaz, ventana, paneles y botón.
33     JFrame marcoPrograma;
34     private JButton botonSalir;
35     private JPanel panel2;
36
37     // Objetos para la persistencia mediante Archivos.
38     String ubicacionNombre = null;
39     String ubicacionPuntaje = null;
40     JLabel nombres[] = new JLabel[10];
41     JLabel puntajes[] = new JLabel[10];
42     String nombre[];
43     String puntaje[];
44     int puntaje1[];
45     String jugador = null;
46     JLabel etiquetaPunto;
47
48     // El constructor de la clase Interfaz la cual contiene el método Inicializador.
49     public Interfaz(String jugador, String ubicacionNombre, String ubicacionPuntaje) {
50         inicializar(jugador, ubicacionNombre, ubicacionPuntaje);
51     }
52
53
54     // Método inicializador, acá pasa toda la magia.
55     public void inicializar(String jugador, String ubicacionNombre, String ubicacionPuntaje) {
56
57         // Asignación del nombre del jugador a la variable de la clase Interfaz.
58
59         this.jugador = jugador;
60
61         // Creación de la ventana, con sus dimensiones, icono y estableciendo el manejo
62         // de los paneles.
63         marcoPrograma = new JFrame();
64         marcoPrograma.setUndecorated(true);
65         marcoPrograma.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
66         marcoPrograma.getContentPane().setBackground(Color.BLACK);
67         marcoPrograma.setBounds(0, 0, 600, 400);
68         marcoPrograma.setLocationRelativeTo(null);
69         marcoPrograma
70             .setIconImage(Toolkit.getDefaultToolkit().getImage(Main.class.getResource("/Imagen/iconoTetris.png")));
71         marcoPrograma.getContentPane().setLayout(null);
72
73         // Asignación de las ubicaciones de los archivos a las variables de la clase
74         // Interfaz.
75         this.ubicacionNombre = ubicacionNombre;
76         this.ubicacionPuntaje = ubicacionPuntaje;
77
78         // Creación de los archivos.
79         Archivo archivoNombre = new Archivo(ubicacionNombre);
80         Archivo archivoPuntaje = new Archivo(ubicacionPuntaje);
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```

82 // Asignación de los datos de los archivos para trabajar durante todo el
83 // programa
84 nombre = archivoNombre.leerArchivoString(ubicacionNombre);
85 puntaje = archivoPuntaje.leerArchivoIntAString(ubicacionPuntaje);
86 puntaje1 = archivoPuntaje.leerArchivoInt(ubicacionPuntaje);
87
88 // Creación del panel 2 con sus respectivas etiquetas, texto y botón. El panel 2
89 // se encarga de llevar el nombre del jugador, el puntaje del jugador y el botón
90 // para salir del juego.
91 panel2 = new JPanel();
92 panel2.setBounds(237, 0, 200, 420);
93 panel2.setBackground(Color.DARK_GRAY);
94 marcoPrograma.getContentPane().add(panel2);
95 panel2.setLayout(null);
96
97 // Logo que aparece en el panel derecho
98 JLabel lblLogo = new JLabel("");
99 lblLogo.setVerticalAlignment(SwingConstants.TOP);
100 lblLogo.setIcon(new ImageIcon(Interfaz.class.getResource("/Imagen/logoInterfaz.jpg")));
101 lblLogo.setBounds(10, 11, 180, 101);
102 panel2.add(lblLogo);
103
104 botonSalir = new JButton("SALIR");
105 botonSalir.setBackground(Color.BLACK);
106 botonSalir.setBounds(43, 346, 80, 36);
107 botonSalir.setForeground(Color.RED);
108 botonSalir.addActionListener(new ActionListener() {
109     // Cuando se le da click al botón de salir se cierra el juego.
110     public void actionPerformed(ActionEvent e) {
111
112         System.exit(0);
113     }
114 });
115 panel2.add(botonSalir);
116
117 etiquetaPunto = new JLabel("0");
118 etiquetaPunto.setForeground(Color.CYAN);
119 etiquetaPunto.setFont(new Font("Segoe UI", Font.PLAIN, 25));
120 etiquetaPunto.setBackground(Color.BLACK);
121 etiquetaPunto.setBounds(73, 266, 117, 36);
122 panel2.add(etiquetaPunto);
123
124 JLabel lblHighScoreTitle = new JLabel("PUNTAJE M\u00C1S ALTO:");
125 lblHighScoreTitle.setForeground(Color.PINK);
126 lblHighScoreTitle.setBounds(10, 123, 164, 14);
127 panel2.add(lblHighScoreTitle);
128
129 JLabel lblScore = new JLabel("999999 puntos");
130 lblScore.setForeground(Color.CYAN);
131 lblScore.setFont(new Font("Dialog", Font.PLAIN, 20));
132 lblScore.setBounds(20, 148, 170, 36);
133 lblScore.setText(puntaje[0] + " puntos");
134 panel2.add(lblScore);
135
136 JLabel etiquetaPlayer = new JLabel("JUGADOR:");
137 etiquetaPlayer.setForeground(Color.PINK);
138 etiquetaPlayer.setBackground(Color.BLACK);
139 etiquetaPlayer.setBounds(55, 196, 68, 23);

```

```

135
136 JLabel etiquetaPlayer = new JLabel("JUGADOR:");
137 etiquetaPlayer.setForeground(Color.PINK);
138 etiquetaPlayer.setBackground(Color.BLACK);
139 etiquetaPlayer.setBounds(55, 196, 68, 23);
140 panel2.add(etiquetaPlayer);
141
142 JLabel etiquetaJugador = new JLabel("<dynamic>");
143 etiquetaJugador.setHorizontalAlignment(SwingConstants.CENTER);
144 etiquetaJugador.setForeground(Color.CYAN);
145 etiquetaJugador.setFont(new Font("Tahoma", Font.PLAIN, 25));
146 etiquetaJugador.setBackground(Color.BLACK);
147 etiquetaJugador.setBounds(20, 217, 151, 23);
148 etiquetaJugador.setText(jugador);
149 panel2.add(etiquetaJugador);
150
151 JLabel etiquetaPuntaje2 = new JLabel("PUNTAJE");
152 etiquetaPuntaje2.setForeground(Color.PINK);
153 etiquetaPuntaje2.setBackground(Color.BLACK);
154 etiquetaPuntaje2.setBounds(55, 251, 68, 23);
155 panel2.add(etiquetaPuntaje2);
156
157 JLabel logo = new JLabel("");
158 logo.setVerticalAlignment(SwingConstants.TOP);
159 logo.setIcon(new ImageIcon(Interfaz.class.getResource("/Imagen/frameImg.png")));
160 logo.setBounds(0, 0, 237, 420);
161 marcoPrograma.getContentPane().add(logo);
162
163 // Creación del panel central, panel 3 o tablero. Este tablero se encarga del
164 // juego.
165 Tablero tablero = new Tablero(this);
166 tablero.setBounds(18, 0, 200, 400);
167 marcoPrograma.getContentPane().add(tablero);
168 panel2.add(lblScore);
169
170 // Inicia el juego.
171 tablero.start();
172
173 marcoPrograma.setSize(438, 420);
174
175 }
176
177 // Métodos getters para que Tablero realice sus respectivos procesos.
178
179 public JLabel getEtiquetaPunto() {
180     return etiquetaPunto;
181 }
182
183 public JLabel[] getNombres() {
184
185     return nombres;
186
187 }
188
189 public JLabel[] getPuntajes() {
190
191     return puntajes;
192
193 }

```

```
194
195 public String[] getNombre() {
196     return nombre;
197 }
198
199
200
201 public String[] getPuntaje() {
202     return puntaje;
203 }
204
205
206
207 public int[] getPuntaje1() {
208     return puntaje1;
209 }
210
211
212
213 public String getJugador() {
214     return jugador;
215 }
216
217
218
219 public String getUbicacionNombre() {
220     return ubicacionNombre;
221 }
222
223
224
225 public String getUbicacionPuntaje() {
226     return ubicacionPuntaje;
227 }
228
229
230
231 }
232
```



## Main.java

```
1 package Tetris;
2
3
4 * NOMBRE DEL PROYECTO: TETRIS
5
6
7
8
9
10
11
12
13
14
15
16
17 import java.awt.EventQueue;
18
19
20 public class Main {
21
22     public static void main(String[] arg) throws Exception {
23
24
25         EventQueue.invokeLater(new Runnable() {
26             public void run() {
27                 try {
28
29                     // Se crean los archivos
30                     String ubicacionNombre = "src/Archivo/archivoNombre.txt";
31                     String ubicacionPuntaje = "src/Archivo/archivoPuntaje.txt";
32                     VentanaPrincipal window = new VentanaPrincipal(ubicacionNombre, ubicacionPuntaje);
33                     window.setVisible(true);
34
35                     // Creacion del objeto que reproduce la música en el juego.
36                     String filepath = "src/Sonido/sonidoTetris.wav";
37                     Sonido music = new Sonido();
38                     music.play(filepath);
39                 } catch (Exception e) {
40                     e.printStackTrace();
41                 }
42             }
43         });
44     }
45 }
46 }
```

## Pieza.java

```
1 package Tetris;
2
3 /*
4  * NOMBRE DEL PROYECTO: TETRIS
5  *
6  * AUTORES: Javier Santiago Borbón Borbón y José Andrés Sanabria Arias
7  * Códigos: 20182020085 y 2018202095
8  *
9  *
10 * FECHA: 28/03/2020 (Marzo 28 de 2020)
11 *
12 *
13 * Versión 1.0;
14 *
15 */
16
17 enum Pieza {
18     figuraBorrador, figuraZ, figuraS, figuraLinea, figuraT, figuraCuadrado, figuraL, figuraLInv
19
20 }
21
22
```

## Sonido.java

```
1 package Tetris;
2
3 * NOMBRE DEL PROYECTO: TETRIS
4
5
6
7
8
9
10
11
12
13
14
15
16
17 import java.io.File;
18
19
20
21
22 public class Sonido {
23     public void play(String musicLocation) {
24         try {
25             File musicPath = new File(musicLocation);
26             if (musicPath.exists()) {
27                 AudioInputStream audioInput = AudioSystem.getAudioInputStream(musicPath);
28                 Clip clip = AudioSystem.getClip();
29                 clip.open(audioInput);
30                 clip.start();
31                 clip.loop(Clip.LOOP_CONTINUOUSLY);
32             }
33             } else {
34                 System.out.println("No se encuentra la música");
35             }
36         } catch (Exception ex) {
37             ex.printStackTrace();
38         }
39     }
40 }
41
42
```

## Tablero.java

```
1 package Tetris;
2
3
4 * NOMBRE DEL PROYECTO: TETRIS
5
6
7
8
9
10
11
12
13
14
15
16
17 import java.awt.Color;
18
19
20
21
22
23
24
25
26
27
28
29
30 public class Tablero extends JPanel implements ActionListener {
31
32     // Tamaño del tablero del tetris
33     private final int anchoTablero = 10;
34     private final int alturaTablero = 20;
35
36     // Variables necesarias para procesar el juego en el tablero.
37
38     private boolean haCaido = false;
39     private boolean empezo = false;
40     private boolean enPausa = false;
41     private int numLineasRemovidas = 0;
42     private int curX = 0;
43     private int curY = 0;
44     private JLabel estadoPunto;
45     private Forma curPiece;
46     private Pieza[] board;
47
48     // Llamamos a las figuras
49
50     private FiguraBorrador figBorrador = new FiguraBorrador();
51     private FiguraCuadrado figC = new FiguraCuadrado();
52     private FiguraL figL = new FiguraL();
53     private FiguraLinea figLinea = new FiguraLinea();
54     private FiguraLInv figLInv = new FiguraLInv();
55     private FiguraS figS = new FiguraS();
56     private FiguraT figT = new FiguraT();
57     private FiguraZ figZ = new FiguraZ();
58
59
60     // Variables necesarias para hacer la persistencia mediante Archivos.
61     private Archivo archivo = new Archivo();
62     private String ubicacionNombre;
63     private String ubicacionPuntaje;
64     private String nombre[];
65     private String puntaje[];
66     private int puntaje1[];
67     private int auxPuntaje = 0;
68     private String auxNombre = null;
69     private String puntoCadena = null;
70     private int puntoEntero = 0;
71     private String jugador = null;
72     private Timer timer;
73 }
```

```

74 // Constructor de la clase Tablero, en la cual se encarga del Panel3, central o
75 // tablero. Se requiere de parametro la Interfaz para hacer un traspaso de
76 // Variables.
77 public Tablero(Interfaz interfaz) {
78     setFocusable(true);
79     setBackground(Color.BLACK);
80     curPiece = new Forma();
81     timer = new Timer(400, this);
82     timer.start();
83     ubicacionNombre = interfaz.getUbicacionNombre();
84     ubicacionPuntaje = interfaz.getUbicacionPuntaje();
85     jugador = interfaz.getJugador();
86     estadoPunto = interfaz.getEtiquetaPunto();
87     nombre = interfaz.getNombre();
88     puntaje = interfaz.getPuntaje();
89     puntaje1 = interfaz.getPuntaje1();
90
91     board = new Pieza[anchoTablero * alturaTablero];
92     addKeyListener(new TAdapter());
93     clearBoard();
94 }
95
96 // Método que hace que la pieza caiga.
97 public void actionPerformed(ActionEvent e) {
98     if (haCaído) {
99         haCaído = false;
100         newPiece();
101     } else {
102         oneLineDown();
103     }
104 }
105
106 // Establece los límites del juego en el tablero.
107 int squareWidth() {
108     return (int) getSize().getWidth() / anchoTablero;
109 }
110
111 int squareHeight() {
112     return (int) getSize().getHeight() / alturaTablero;
113 }
114
115 Pieza shapeAt(int x, int y) {
116     return board[(y * anchoTablero) + x];
117 }
118
119 // Método principal del tablero, da inicio al timer, creación de piezas.
120 public void start() {
121     if (enPausa)
122         return;
123
124     empezo = true;
125     haCaído = false;
126     numLineasRemovidas = 0;
127     clearBoard();
128
129     newPiece();
130     timer.start();
131 }
132
133

```

```

133
134 // Método por el cual se logra detener el juego.
135 private void pause() {
136     if (!empezo)
137         return;
138
139     enPausa = !enPausa;
140     if (enPausa) {
141         timer.stop();
142         estadoPunto.setText("Pausado");
143     } else {
144         timer.start();
145         estadoPunto.setText(String.valueOf(numLineasRemovidas));
146     }
147     repaint();
148 }
149
150 // Método por el cual se pintan las pizas gracias a la clase Forma.
151 public void paint(Graphics g) {
152     super.paint(g);
153
154     Dimension size = getSize();
155     int boardTop = (int) size.getHeight() - alturaTablero * squareHeight(); // Pintar Tablero
156
157     for (int i = 0; i < alturaTablero; ++i) {
158         for (int j = 0; j < anchoTablero; ++j) {
159             Pieza shape = shapeAt(j, alturaTablero - i - 1);
160             if (shape != Pieza.figuraBorrador)
161                 drawSquare(g, 0 + j * squareWidth(), boardTop + i * squareHeight(), shape);
162         }
163     }
164
165     if (curPiece.getShape() != Pieza.figuraBorrador) {
166         for (int i = 0; i < 4; ++i) {
167             int x = curX + curPiece.x(i);
168             int y = curY - curPiece.y(i);
169             drawSquare(g, 0 + x * squareWidth(), boardTop + (alturaTablero - y - 1) * squareHeight(),
170                 curPiece.getShape());
171         }
172     }
173 }
174
175 // Método para hacer caer la ficha (con BarraEspaciadora en el juego).
176 private void dropDown() {
177     int newY = curY;
178     while (newY > 0) {
179         if (!tryMove(curPiece, curX, newY - 1))
180             break;
181         --newY;
182     }
183     pieceDropped();
184 }
185
186 // Método por el cual la ficha baja si no hay un obstaculo.
187 private void oneLineDown() {
188     if (!tryMove(curPiece, curX, curY - 1))
189         pieceDropped();
190 }
191
192 // Método para mover la ficha horizontalmente.

```

```

192 // Método para limpiar el castro de las fichas al caer.
193 private void clearBoard() {
194     for (int i = 0; i < alturaTablero * anchoTablero; ++i)
195         board[i] = Pieza.figuraBorrador;
196 }
197
198 // Método por el cual
199 private void pieceDropped() {
200     for (int i = 0; i < 4; ++i) {
201         int x = curX + curPiece.x(i);
202         int y = curY - curPiece.y(i);
203         board[(y * anchoTablero) + x] = curPiece.getShape();
204     }
205
206     removerLineasLlenas();
207
208     if (!haCaido)
209         newPiece();
210 }
211
212 // Método fundamental para crear piezas aleatorias, además de ser el método que
213 // crea la persistencia.
214 private void newPiece() {
215     curPiece.setRandomShape();
216     curX = anchoTablero / 2 + 1;
217     curY = alturaTablero - 1 + curPiece.minY();
218
219     if (!tryMove(curPiece, curX, curY)) {
220         curPiece.setShape(Pieza.figuraBorrador);
221         timer.stop();
222         empezo = false;
223         puntoCadena = estadoPunto.getText();
224         puntoEntero = Integer.parseInt(puntoCadena);
225         JOptionPane.showMessageDialog(null, "JUEGO TERMINADO!!\n\nSu puntaje es: " + puntoEntero, "GAME OVER",
226             JOptionPane.INFORMATION_MESSAGE, null);
227
228         actualizarTabla();
229     }
230 }
231
232 // Método que ayuda a no sobre poner las fichas al rotar.
233 private boolean tryMove(Forma newPiece, int newX, int newY) {
234     for (int i = 0; i < 4; ++i) {
235         int x = newX + newPiece.x(i);
236         int y = newY - newPiece.y(i);
237         if (x < 0 || x >= anchoTablero || y < 0 || y >= alturaTablero)
238             return false;
239         if (shapeAt(x, y) != Pieza.figuraBorrador)
240             return false;
241     }
242
243     curPiece = newPiece;
244     curX = newX;
245     curY = newY;
246     repaint();
247     return true;
248 }
249
250 // Método por el cual identifica cuando hay una línea y aumenta el puntaje del

```

```

251 // Método por el cual identifica cuando hay una línea, aumenta el puntaje del
252 // jugador.
253 private void removerLineasLlenas() {
254     int numFullLines = 0;
255
256     for (int i = alturaTablero - 1; i >= 0; --i) {
257         boolean lineIsFull = true;
258
259         for (int j = 0; j < anchoTablero; ++j) {
260             if (shapeAt(j, i) == Pieza.figuraBorrador) {
261                 lineIsFull = false;
262                 break;
263             }
264         }
265
266         if (lineIsFull) {
267             ++numFullLines;
268             for (int k = i; k < alturaTablero - 1; ++k) {
269                 for (int j = 0; j < anchoTablero; ++j)
270                     board[(k * anchoTablero) + j] = shapeAt(j, k + 1);
271             }
272         }
273     }
274
275     if (numFullLines > 0) {
276         numLineasRemovidas += numFullLines;
277         estadoPunto.setText(String.valueOf(numLineasRemovidas));
278         haCaído = true;
279         curPiece.setShape(Pieza.figuraBorrador);
280         repaint();
281     }
282 }
283
284 private void drawSquare(Graphics g, int x, int y, Pieza shape) {
285
286     Color colors[] = { figBorrador.getColor(), fig2.getColor(), fig5.getColor(), figLinea.getColor(), figT.getColor(), figC.getColor(), figL.getColor(), figLInv.getColor() };
287
288     Color color = colors[shape.ordinal()]; //Se crea un arreglo de 7 figuras
289
290     g.setColor(color);
291     g.fillRect(x + 1, y + 1, squareWidth() - 2, squareHeight() - 2);
292
293     g.setColor(color.brighter());
294     g.drawLine(x, y + squareHeight() - 1, x, y);
295     g.drawLine(x, y, x + squareWidth() - 1, y);
296
297     g.setColor(color.darker());
298     g.drawLine(x + 1, y + squareHeight() - 1, x + squareWidth() - 1, y + squareHeight() - 1);
299     g.drawLine(x + squareWidth() - 1, y + squareHeight() - 1, x + squareWidth() - 1, y + 1);
300 }

```



```

301
302 // Clase que capta los eventos por teclado posteriormente mueve la pieza o
303 // detiene el juego.
304 class TAdapter extends KeyAdapter {
305     public void keyPressed(KeyEvent e) {
306         if (!empezo || curPiece.getShape() == Pieza.figuraBorrador) {
307             return;
308         }
309
310         int keycode = e.getKeyCode();
311
312         if (keycode == 'p' || keycode == 'P') {
313             pause();
314             return;
315         }
316
317         if (enPausa)
318             return;
319
320         switch (keycode) {
321             case KeyEvent.VK_LEFT:
322                 tryMove(curPiece, curX - 1, curY);
323                 break;
324             case KeyEvent.VK_RIGHT:
325                 tryMove(curPiece, curX + 1, curY);
326                 break;
327             case KeyEvent.VK_DOWN:
328                 oneLineDown();
329                 break;
330             case KeyEvent.VK_UP:
331                 tryMove(curPiece.rotateLeft(), curX, curY);
332                 break;
333             case KeyEvent.VK_SPACE:
334                 dropDown();
335                 break;
336         }
337     }
338 }
339
340 private void actualizarTabla() {
341     nombre[10] = jugador;
342     puntaje1[10] = puntoEntero;
343
344     for (int i = 0; i < 10; i++) {
345         for (int j = 0; j < 10; j++) {
346             if (puntaje1[j] < puntaje1[j + 1]) {
347                 auxPuntaje = puntaje1[j];
348                 puntaje1[j] = puntaje1[j + 1];
349                 puntaje1[j + 1] = auxPuntaje;
350
351                 auxNombre = nombre[j];
352                 nombre[j] = nombre[j + 1];
353                 nombre[j + 1] = auxNombre;
354             }
355         }
356     }
357 }
358
359
360

```

```
360         }
361     }
362 }
363
364 }
365
366 for (int i = 0; i < 10; i++) {
367     puntaje[i] = Integer.toString(puntaje1[i]);
368 }
369
370 }
371
372 archivo.sobreEscribirArchivo(nombre, ubicacionNombre);
373 archivo.sobreEscribirArchivo(puntaje1, ubicacionPuntaje);
374
375 }
376
377 }
378
```

## VentanaPrincipal.java

```
1 package Tetris;
2
3
4@ = @title Ventana Principal
5
24
25# import java.awt.Color;
26
39
40 public class VentanaPrincipal extends JFrame implements ActionListener {
41 // Objetos para la persistencia mediante Archivos.
42
43 private static final long serialVersionUID = 1L;
44 private JPanel contentPane;
45 private JPanel panelPuntajes;
46 private JPanel panelInstrucciones;
47 private String ubicacionNombre = null;
48 private String ubicacionPuntaje = null;
49 private JLabel puntajes[] = new JLabel[10];
50 private JLabel nombres[] = new JLabel[10];
51 private String nombre[];
52 private String puntaje[];
53
54 // BOTONES
55 private JButton btnSalir;
56 private JButton btnPuntuacion;
57 private JButton btnNewGame;
58 private JButton btnInstrucciones;
59
59# public VentanaPrincipal(String ubicacionNombre, String ubicacionPuntaje) {
60 this.ubicacionNombre = ubicacionNombre;
61 this.ubicacionPuntaje = ubicacionPuntaje;
62 setBounds(100, 100, 600, 300);
63 setUndecorated(true);
64 setLocationRelativeTo(null);
65 contentPane = new JPanel();
66 setContentPane(contentPane);
67 contentPane.setLayout(null);
68
69 // Asignación de las ubicaciones de los archivos a las variables de la clase
70 // Interfaz.
71 this.ubicacionNombre = ubicacionNombre;
72 this.ubicacionPuntaje = ubicacionPuntaje;
73
74 // Creación de los archivos.
75 Archivo archivoNombre = new Archivo(ubicacionNombre);
76 Archivo archivoPuntaje = new Archivo(ubicacionPuntaje);
77 nombre = archivoNombre.leerArchivoString(ubicacionNombre);
78 puntaje = archivoPuntaje.leerArchivoIntAString(ubicacionPuntaje);
79
80 // Creación del Panel en donde se muestran las instrucciones
81 panelInstrucciones = new JPanel();
82 panelInstrucciones.setBackground(Color.BLACK);
83 panelInstrucciones.setBounds(170, 0, 259, 262);
84 contentPane.add(panelInstrucciones);
85 panelInstrucciones.setLayout(null);
86 JTextPane textoInstruccion = new JTextPane();
87 textoInstruccion.setEditable(false);
88 textoInstruccion.setForeground(new Color(255, 255, 255));
89 textoInstruccion.setBackground(Color.BLACK);
90 textoInstruccion.setText(
91 "P : Pausar el juego.\n\n \u2191 : Rotar la figura.\n\n \u2192 : Moverse a la derecha.\n\n \u2190 : Moverse a la izquierda.\n\n \u2193 : Moverse hacia abajo.\n\n Barra Espaciadora: Bajar la figura.");
92 textoInstruccion.setBounds(42, 161, 217, 181);
93 panelInstrucciones.add(textoInstruccion);
94
95 JLabel lblIcono = new JLabel("");
96 lblIcono.setIcon(new ImageIcon(VentanaPrincipal.class.getResource("/Imagen/iconoInstrucciones.jpg")));
97 lblIcono.setBounds(0, 0, 259, 168);
98 panelInstrucciones.add(lblIcono);
99 panelInstrucciones.setVisible(false);
100
101 // Creación del Panel en donde se muestran los puntajes
102 panelPuntajes = new JPanel();
103 panelPuntajes.setBounds(112, 0, 375, 262);
104 contentPane.add(panelPuntajes);
105 JLabel lblTituloPuntajes = new JLabel("MAESTROS CAMPEONES");
106 lblTituloPuntajes.setForeground(Color.BLUE);
107 lblTituloPuntajes.setFont(new Font("Tw Cen MT Condensed Extra Bold", Font.PLAIN, 27));
108 lblTituloPuntajes.setBounds(72, 5, 240, 30);
109 colocarPuntajes();
110 panelPuntajes.add(lblTituloPuntajes);
111 JLabel lblName = new JLabel("NOMBRE");
112 lblName.setFont(new Font("Times New Roman", Font.PLAIN, 16));
113 lblName.setBounds(30, 40, 244, 14);
114 panelPuntajes.add(lblName);
115
116 JLabel lblPuntos = new JLabel("PUNTOS");
117 lblPuntos.setFont(new Font("Times New Roman", Font.PLAIN, 16));
118 lblPuntos.setBounds(266, 40, 99, 14);
119 panelPuntajes.add(lblPuntos);
120
121 JLabel lblImagenFondo = new JLabel();
122 lblImagenFondo.setBounds(0, 0, 375, 262);
123 lblImagenFondo.setIcon(new ImageIcon(VentanaPrincipal.class.getResource("/Imagen/fondo.jpg")));
124 panelPuntajes.add(lblImagenFondo);
125 panelPuntajes.setVisible(false);
126 btnInstrucciones = new JButton("Instrucciones");
127 btnInstrucciones.setBounds(170, 266, 118, 23);
128 btnInstrucciones.addActionListener(this);
129 contentPane.add(btnInstrucciones);
130
131 btnNewGame = new JButton("Nuevo Juego");
132 btnNewGame.setBounds(26, 266, 124, 23);
133 btnNewGame.addActionListener(this);
134 contentPane.add(btnNewGame);
135
136 btnPuntuacion = new JButton("Puntuaci\u00f3n");
137 btnPuntuacion.setBounds(312, 266, 100, 23);
138 btnPuntuacion.addActionListener(this);
139 contentPane.add(btnPuntuacion);
```

```

135     btnPuntuacion = new JButton("Puntuaci\u00F3n");
136     btnPuntuacion.setBounds(312, 266, 100, 23);
137     btnPuntuacion.addActionListener(this);
138     contentPane.add(btnPuntuacion);
139
140     btnSalir = new JButton("Salir");
141     btnSalir.addActionListener(this);
142     btnSalir.setBounds(445, 266, 106, 23);
143     contentPane.add(btnSalir);
144
145     JLabel lblImgVentana = new JLabel("");
146     lblImgVentana.setBounds(0, 0, 600, 300);
147     lblImgVentana.setIcon(new ImageIcon(VentanaPrincipal.class.getResource("/Imagen/Main Logo.jpg")));
148     contentPane.add(lblImgVentana);
149
150 }
151
152 private void colocarPuntajes() {
153     panelPuntajes.setLayout(null);
154
155     for (int i = 0; i < puntajes.length; i++) {
156         nombres[i] = new JLabel(nombre[i]);
157         nombres[i].setBounds(10, 60 + 20 * i, 264, 14);
158         panelPuntajes.add(nombres[i]);
159         if (Integer.parseInt(puntaje[i]) < 10) {
160             puntajes[i] = new JLabel(" " + puntaje[i]);
161         } else if (Integer.parseInt(puntaje[i]) > 100) {
162             puntajes[i] = new JLabel(puntaje[i]);
163         } else {
164             puntajes[i] = new JLabel(" " + puntaje[i]);
165         }
166         puntajes[i].setBounds(285, 60 + 20 * i, 80, 14);
167         panelPuntajes.add(puntajes[i]);
168     }
169 }
170
171
172 // Aqu\u00ed es donde se le da funcionamiento a los botones
173 @Override
174 public void actionPerformed(ActionEvent e) {
175     if (e.getSource().equals(btnNewGame)) {
176         panelPuntajes.setVisible(false);
177         panelInstrucciones.setVisible(false);
178         String name = "";
179         name = JOptionPane.showInputDialog("Ingrese el nombre del jugador:");
180         if (name == null || name.isEmpty() || name == "") { // Si el jugador no ingresa nombre alguno
181             name = "Player 1";
182         }
183         Interfaz juegoTetris = new Interfaz (name, ubicacionNombre, ubicacionPuntaje);
184         dispose();
185         juegoTetris.marcoPrograma.setVisible(true);
186     }
187
188
189     if (e.getSource().equals(btnPuntuacion)) {
190         panelInstrucciones.setVisible(false);
191         if (panelPuntajes.isVisible()) {
192             panelPuntajes.setVisible(false);
193         } else {

```

```

154
155     for (int i = 0; i < puntajes.length; i++) {
156         nombres[i] = new JLabel(nombre[i]);
157         nombres[i].setBounds(10, 60 + 20 * i, 264, 14);
158         panelPuntajes.add(nombres[i]);
159         if (Integer.parseInt(puntaje[i]) < 10) {
160             puntajes[i] = new JLabel(" " + puntaje[i]);
161         } else if (Integer.parseInt(puntaje[i]) > 100) {
162             puntajes[i] = new JLabel(puntaje[i]);
163         } else {
164             puntajes[i] = new JLabel(" " + puntaje[i]);
165         }
166         puntajes[i].setBounds(285, 60 + 20 * i, 80, 14);
167         panelPuntajes.add(puntajes[i]);
168     }
169
170 }
171
172 // Aquí es donde se le da funcionamiento a los botones
173 @Override
174 public void actionPerformed(ActionEvent e) {
175     if (e.getSource().equals(btnNewGame)) {
176         panelPuntajes.setVisible(false);
177         panelInstrucciones.setVisible(false);
178         String name = "";
179         name = JOptionPane.showInputDialog("Ingrese el nombre del jugador:");
180         if (name == null || name.isEmpty() || name == "") { // Si el jugador no ingresa nombre alguno
181             name = "Player 1";
182         }
183         Interfaz juegoTetris = new Interfaz (name, ubicacionNombre, ubicacionPuntaje);
184         dispose();
185         juegoTetris.marcoPrograma.setVisible(true);
186     }
187
188
189     if (e.getSource().equals(btnPuntuacion)) {
190         panelInstrucciones.setVisible(false);
191         if (panelPuntajes.isVisible()) {
192             panelPuntajes.setVisible(false);
193         } else {
194             panelPuntajes.setVisible(true);
195         }
196     }
197
198     if (e.getSource().equals(btnSalir)) {
199         System.exit(0);
200     }
201     if (e.getSource().equals(btnInstrucciones)) {
202         panelPuntajes.setVisible(false);
203
204         if (panelInstrucciones.isVisible()) {
205             panelInstrucciones.setVisible(false);
206         } else {
207             panelInstrucciones.setVisible(true);
208         }
209     }
210 }
211 }
212 }
213

```

## Index.html

En este documento se puede ver cada una de las clases, este índice además de mostrar los diferentes archivos, también le muestra a la persona los métodos, las variables y otros elementos empleados en este proyecto. Si necesita ayuda con este sumario, tiene la opción de ver la ayuda, en la pestaña que dice “HELP”.

