

Tarea AD-2 JavaDoc y JUnit

A continuacion, voy a mostrar el enunciado pedido en la actividad.

Requerimiento 1

Documentar y hacer las pruebas unitarias de la siguiente clase.

```
public class Soldado {  
    private boolean estasMuerto;  
    private int numeroBalas;  
    /*Crear los métodos "get" y "set" de los atributos cuando se vayan a  
hacer las pruebas y la documentación. Aquí no se han creado porque no apotan nada.  
*/  
  
    public boolean puedeDisparar(){  
        if(this.numeroBalas >0){  
            return true;  
        }  
        return false;  
    }  
    public void disparar(Soldado sol){  
        this.numeroBalas--;  
        sol.estaMuerto = true;  
    }  
}
```

Valoración: 5 puntos sobre 10

Requerimiento 2

Documentar y hacer las pruebas unitarias de la siguiente clase.

```
public class Jugador{  
    private int dorsal;  
    private int numeroTarjetasAmarillas;  
    private int numeroTarjetasRojas;  
    /*Crear los métodos "get" y "set" de los atributos cuando se vayan  
a hacer las pruebas y la documentación. Aquí no se han creado porque no apotan  
nada.  
*/
```

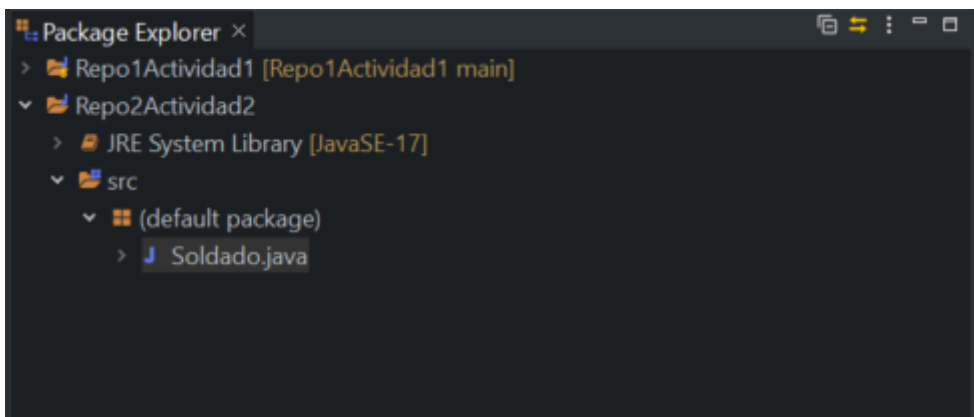
```
        public void ponerDorsal(int dorsal){
            if(dorsal)>=1 && dorsal <=30){
                this.dorsal = dorsal;
            } else{
                this.dorsal = -1;
            }
        }
    }
```

Para comenzar vamos a organizar la tarea de la siguiente manera, vamos a crear las clases en [Eclipse](#), vamos a generar la documentacion de la clase **Soldado** y **Jugador** con [JavaDoc](#), y posteriormente vamos a realizar las pruebas unitarias con [JUnit5](#).



Comenzamos

- Creamos la clase donde vamos a realizar la tarea :



- Creamos la clase :

```

1
2 public class Soldado {
3
4     private boolean estasMuerto;
5     private int numeroBalas;
6
7     /*
8      * Constructor con parámetros.
9      * Los constructores también se pueden comentar, teniendo en cuenta no devuelven nada
10     * por lo que la etiqueta @Return no habría que ponerla.
11     * En este caso no los voy a comentar con JavaDoc ya que el ejercicio no lo pedía
12     */
13     public Soldado(boolean estasMuerto, int numeroBalas) {
14         super();
15         this.estasMuerto = estasMuerto;
16         this.numeroBalas = numeroBalas;
17     }
18
19     //Constructor sin parametros
20     public Soldado() {
21         super();
22     }
23
24     /*
25     * Los getter and setter generalmente no se comentan. Solo en caso de que haga alguna funcionalidad
26     * diferente a la que tienen por defecto.
27     */
28     public boolean isEstasMuerto() {
29         return estasMuerto;
30     }
31
32     public void setEstasMuerto(boolean estasMuerto) {
33         this.estasMuerto = estasMuerto;
34     }
35
36     public int getNumeroBalas() {
37         return numeroBalas;
38     }
39
40     public void setNumeroBalas(int numeroBalas) {
41         this.numeroBalas = numeroBalas;
42     }
43
44
45     public boolean puedeDisparar() {
46
47         if (this.numeroBalas > 0) {
48             return true;
49         }
50         return false;
51     }
52
53     public void disparar(Soldado sol) {
54         this.numeroBalas--;
55         sol.estasMuerto = true;
56     }
57
58
59 }
60

```

En este caso, hemos creado los *constructores* y los *getters and setters* no los hemos comentado porque el ejercicio no nos lo soliciaba.

Empezamos por comentar la clase. Usamos las siguientes etiquetas:

- **@author**. Con esta etiqueta estamos indicando el desarrollador de la clase.
- **@since**. Se usa principalmente en método, indica desde cuando está creada la clase.
- **@version**. Indica la versión de la clase.

```

1
2 /**
3  * Clase que encapsula la información de un soldado. Cada soldado tiene un estado en el que indica si esta
4  * muerto o no, y un número de balas.
5  * @author Isaac Calderón López
6  * @since 20/02/2023
7  * @version 1.0
8  */
9
10 public class Soldado {
11

```

Seguimos comentando los atributos. He puesto la etiqueta **@return**, no estoy muy seguro de si se puede poner en los atributos, he revisado la documentación y he visto que dice que la

etiqueta `@return` no se puede usar en constructores o métodos `"void"`.

```
10 public class Soldado {
11     /**
12      * Este atributo representa el estado del soldado.
13      * @return Devuelve un booleano, true si esta muerto, false no esta muerto.
14      */
15     private boolean estasMuerto;
16     /**
17      * Este atributo representa el número de balas que tiene el soldado.
18      * @return devuelve un numero entero indicando la cantidad de balas que tiene.
19      */
20     private int numeroBalas;
```

Pasamos a comentar ahora los métodos que aparecen en el ejercicio.

En el primer método hemos hecho una breve descripción del método, hemos utilizado la etiqueta `@return` indicando los valores de retorno y su significado.

En el segundo método hemos hecho un resumen también, hemos utilizado la etiqueta `@param` para indicar que parámetros le hemos pasado. En este caso no utilizamos la etiqueta `@return` porque al ser de tipo `void` no devuelve nada.

```
58
59 /**
60  * Este método indica si el soldado puede disparar, si numeroBalas es superior a 0 puede disparar, en caso de que numeroBalas
61  * sea igual o inferior a 0 no podrá disparar.
62  *
63  * @return devuelve un booleano. TRUE en caso de que numeroBalas sea superior a 0, indicando tiene capacidad para disparar,
64  * FALSE en caso de que numeroBalas sea igual o inferior a 0
65  */
66
67 public boolean puedeDisparar() {
68
69     if (this.numeroBalas > 0) {
70         return true;
71     }
72     return false;
73 }
74
75 /**
76  * Este método reduce el numeroBalas en unidades e indica que el soldado al que ha disparado esta muerto.
77  *
78  * @param sol Soldados al que va a disparar
79  */
80 public void disparar(Soldado sol) {
81     this.numeroBalas --;
82     sol.estasMuerto= true;
83 }
84
85
```

Ahora vamos a comentar la siguiente clase que pedía la actividad, la clase **Jugador**.

Comentamos y utilizamos las mismas etiquetas que en la clase anterior:

- **@author.** Con esta etiqueta estamos indicando el desarrollador de la clase.
- **@since.** Se usa principalmente en método, indica desde cuando está creada la clase.
- **@version.** Indica la versión de la clase.

```

1 package Requerimiento2;
2 /**
3  * Clase que encapsula la información de un Jugador. Cada jugador debe tener un dorsal para poder jugar y además
4  * depende del tipo o numero de tarjeta estará expulsado o no.
5  *
6  * @author Isaac Calderón López
7  * @version 1.2
8  * @since 20/02/2023
9  *
10 */
11 public class Jugador {
12

```

Comentamos los atributos, en este caso no hemos puesto ninguna etiqueta por hacerlo algo mas diferente.

```

11 public class Jugador {
12     /**
13      * Este atributo indica el numero entero del dorsal del jugador.
14      */
15     private int dorsal;
16     /**
17      * Este atributo indica el numero entero de tarjetas Amarillas que tiene un jugador.
18      */
19     private int numeroTarjetasAmarillas;
20     /**
21      * Este atributo indica el numero entero de tarjetas rojas que tiene un jugador.
22      */
23     private int numeroTarjetasRojas;
24

```

Hemos puesto un comentario multilinea en los constructores indicando lo que se puede leer en imagen.

```

24
25     /**
26      * Constructor con parámetros.
27      * Los constructores también se pueden comentar, teniendo en cuenta no devuelven nada
28      * por lo que la etiqueta @Return no habría que ponerla.
29      * En este caso no los voy a comentar con JavaDoc ya que el ejercicio no lo pedía
30      */
31     public Jugador(int dorsal, int numeroTarjetasAmarillas, int numeroTarjetasRojas) {
32         super();
33         this.dorsal = dorsal;
34         this.numeroTarjetasAmarillas = numeroTarjetasAmarillas;
35         this.numeroTarjetasRojas = numeroTarjetasRojas;
36     }
37
38     //Constructor sin parámetros
39     public Jugador() {
40         super();
41     }
42

```

Hemo hecho lo mismo con los *Getters and Setters*.

```

44◦  /*
45     * Los getter and setter generalmente no se comentan. Solo en caso de que haga alguna funcionalidad
46     * diferente a la que tienen por defecto.
47     */
48◦  public int getDorsal() {
49      return dorsal;
50  }
51
52
53◦  public void setDorsal(int dorsal) {
54      this.dorsal = dorsal;
55  }
56
57
58◦  public int getNumeroTarjetasAmarillas() {
59      return numeroTarjetasAmarillas;
60  }
61
62
63◦  public void setNumeroTarjetasAmarillas(int numeroTarjetasAmarillas) {
64      this.numeroTarjetasAmarillas = numeroTarjetasAmarillas;
65  }
66
67
68◦  public int getNumeroTarjetasRojas() {
69      return numeroTarjetasRojas;
70  }
71
72
73◦  public void setNumeroTarjetasRojas(int tarjetasRojas) {
74      numeroTarjetasRojas = tarjetasRojas;
75  }
76

```

Ahora comentamos los métodos.

En el primer método utilizamos la etiqueta **@param** y la etiqueta **@author** para indicar los parámetros y el autor del método.

En el segundo utilizamos la etiqueta **@return** para indicar que es lo que devuelve el método. En

ambos hemos hecho una breve descripción del método.

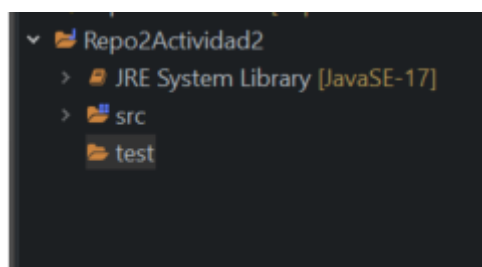
```
76° /**
77  * El siguiente método establece el numero de dorsal de un jugador, establece el rango entre el número 1 y 30
78
79  * @param dorsal se le pasa por parámetro el dorsal del jugador, en caso de que el dorsal no se encuentre entre
80  * 1 y 30 devuelve -1.
81
82  * @author Félix de Pablo.
83  */
84° public void ponerDorsal(int dorsal) {
85     if(dorsal >= 1 && dorsal <= 30) {
86         this.dorsal = dorsal;
87     }else {
88         this.dorsal = -1;
89     }
90 }
91° /**
92  * El siguiente método devuelve un booleano indicando si el jugador está expulsado o no.
93  * Si el jugador tiene el numeroTarjetasAmarillas igual a 2 es expulsado, si también tiene el numeroTarjetasRojas igual a 1
94  * también será expulsado
95  * @return true si el jugador está expulsado, false si el jugador no ha sido expulsado.
96  */
97° public boolean estaExpulsado() {
98     boolean expulsado = false;
99
100     if (numeroTarjetasAmarillas==2) {
101
102         expulsado = true;
103     }
104     if (numeroTarjetasRojas==1) {
105
106         expulsado = true;
107     }
108
109     return true;
110 }
111
112 }
```

Una vez finalizado el tema de la documentación procedemos a realizar las pruebas unitarias.

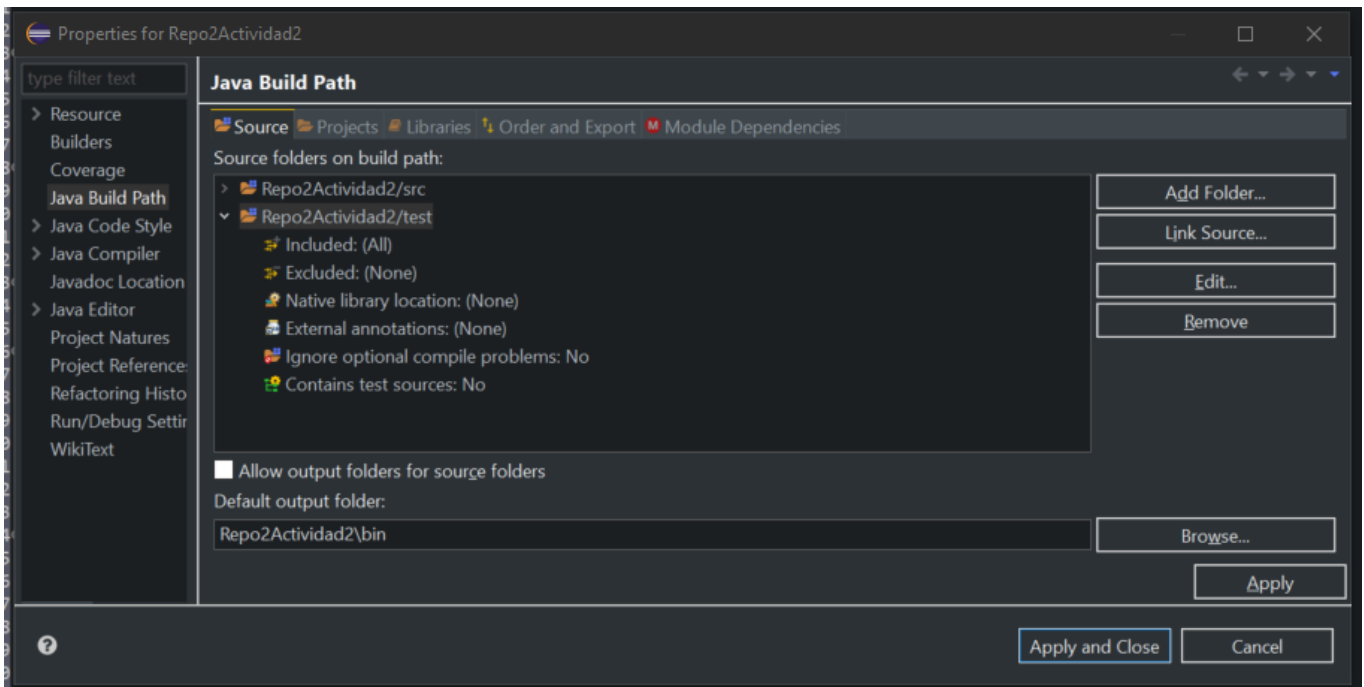


Pruebas unitarias de lo que pide el ejercicio.

Lo primero que hacemos es crear una carpeta que llamaremos *test*.



Configuramos el *BuildPath*.

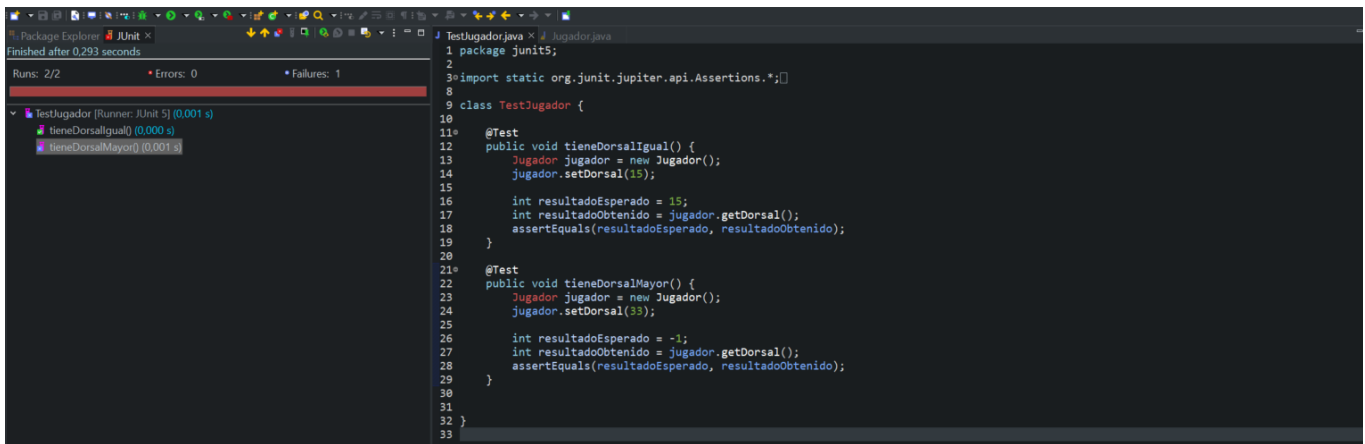


Empezamos haciendo la prueba del metodo `puedeDispara()` de la clase `Soldado`. En este caso vamos a realizar tres pruebas. Si el `numeroBalas` es superior, igual o inferior a 0. Despues, probaremos las clase `estasMuerto()`



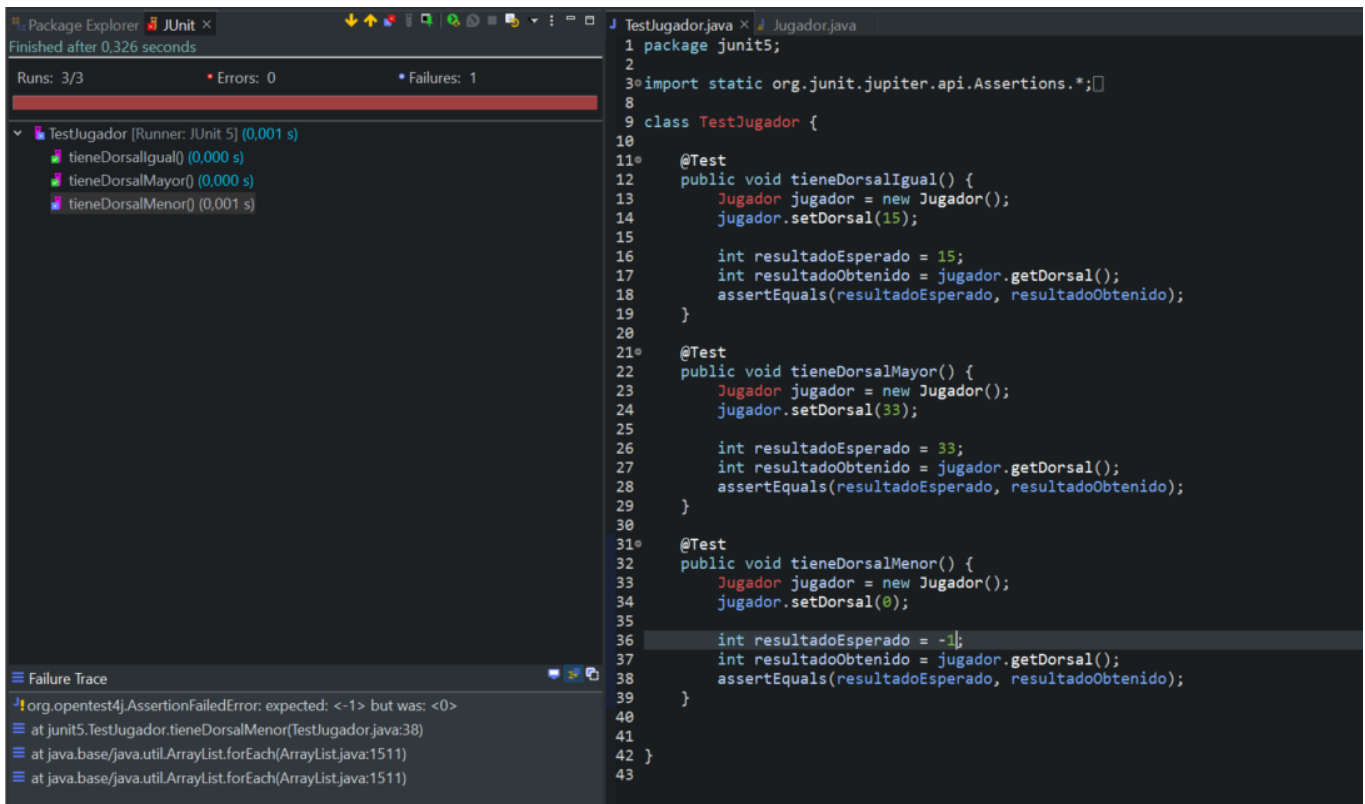
Procedemos a realizar las pruebas unitarias de la clase `Jugador`. En este caso, vamos a realizar tres purebas con el dorsal, un número que sea entre 1 y 30, un número mayor de 30 y otro numero por debajo o igual de 1.

Al realizar la prueba del número mayor de 30 observamos que nos marca un fallo, por lo que podemos decir que el código no es correcto.



```
1 package junit5;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class TestJugador {
6
7     @Test
8     public void tieneDorsalIgual() {
9         Jugador jugador = new Jugador();
10        jugador.setDorsal(15);
11
12        int resultadoEsperado = 15;
13        int resultadoObtenido = jugador.getDorsal();
14        assertEquals(resultadoEsperado, resultadoObtenido);
15    }
16
17     @Test
18     public void tieneDorsalMayor() {
19         Jugador jugador = new Jugador();
20        jugador.setDorsal(33);
21
22        int resultadoEsperado = 33;
23        int resultadoObtenido = jugador.getDorsal();
24        assertEquals(resultadoEsperado, resultadoObtenido);
25    }
26 }
```

Observamos lo mismo en el caso de meterle un numero inferior a 1.

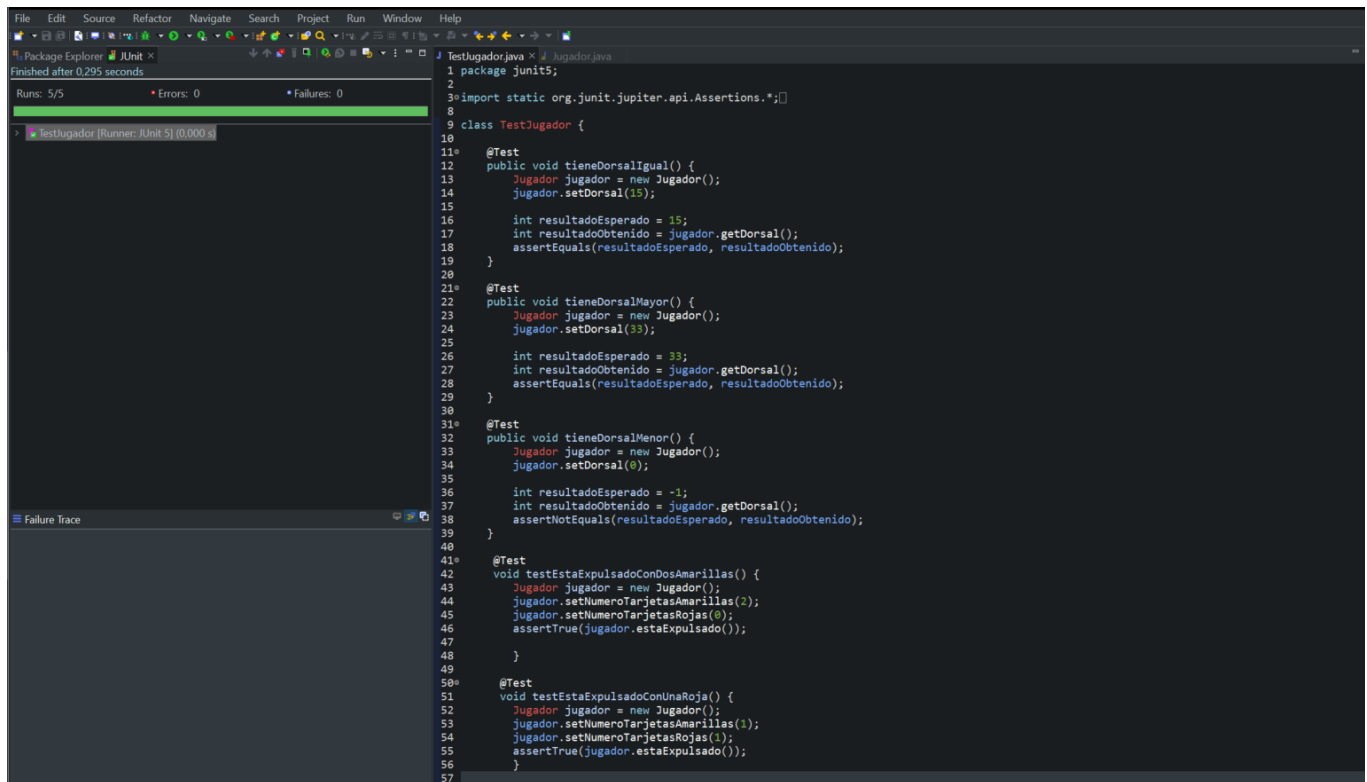


```
1 package junit5;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class TestJugador {
6
7     @Test
8     public void tieneDorsalIgual() {
9         Jugador jugador = new Jugador();
10        jugador.setDorsal(15);
11
12        int resultadoEsperado = 15;
13        int resultadoObtenido = jugador.getDorsal();
14        assertEquals(resultadoEsperado, resultadoObtenido);
15    }
16
17     @Test
18     public void tieneDorsalMayor() {
19         Jugador jugador = new Jugador();
20        jugador.setDorsal(33);
21
22        int resultadoEsperado = 33;
23        int resultadoObtenido = jugador.getDorsal();
24        assertEquals(resultadoEsperado, resultadoObtenido);
25    }
26
27     @Test
28     public void tieneDorsalMenor() {
29         Jugador jugador = new Jugador();
30        jugador.setDorsal(0);
31
32        int resultadoEsperado = -1;
33        int resultadoObtenido = jugador.getDorsal();
34        assertEquals(resultadoEsperado, resultadoObtenido);
35    }
36 }
```

Failure Trace

```
org.opentest4j.AssertionFailedError: expected: <-1> but was: <0>
    at junit5.TestJugador.tieneDorsalMenor(TestJugador.java:38)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

Vamos haciendo las pruebas con el resto de métodos y ya finalizamos.



The screenshot shows an IDE with a dark theme. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The left sidebar shows the Package Explorer with a project named 'JUnit' and a file named 'TestJugador.java'. The main editor displays the code for 'TestJugador.java'. The code is as follows:

```
1 package junit5;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class TestJugador {
6
7     @Test
8     public void tieneDorsalIgual() {
9         Jugador jugador = new Jugador();
10        jugador.setDorsal(15);
11
12        int resultadoEsperado = 15;
13        int resultadoObtenido = jugador.getDorsal();
14        assertEquals(resultadoEsperado, resultadoObtenido);
15    }
16
17     @Test
18     public void tieneDorsalMayor() {
19         Jugador jugador = new Jugador();
20        jugador.setDorsal(33);
21
22        int resultadoEsperado = 33;
23        int resultadoObtenido = jugador.getDorsal();
24        assertEquals(resultadoEsperado, resultadoObtenido);
25    }
26
27     @Test
28     public void tieneDorsalMenor() {
29         Jugador jugador = new Jugador();
30        jugador.setDorsal(0);
31
32        int resultadoEsperado = -1;
33        int resultadoObtenido = jugador.getDorsal();
34        assertEquals(resultadoEsperado, resultadoObtenido);
35    }
36
37     @Test
38     void testEstaExpulsadoConDosAmarillas() {
39         Jugador jugador = new Jugador();
40        jugador.setNumeroTarjetasAmarillas(2);
41        jugador.setNumeroTarjetasRojas(0);
42        assertTrue(jugador.estaExpulsado());
43    }
44
45     @Test
46     void testEstaExpulsadoConUnaRoja() {
47         Jugador jugador = new Jugador();
48        jugador.setNumeroTarjetasAmarillas(1);
49        jugador.setNumeroTarjetasRojas(1);
50        assertTrue(jugador.estaExpulsado());
51    }
52
53 }
```

The bottom of the IDE shows a 'Failure Trace' panel, which is currently empty. The top status bar indicates 'Runs: 5/5', 'Errors: 0', and 'Failures: 0'. The bottom status bar shows 'Finished after 0.295 seconds'.

He hecho un par de modificaciones y vemos que al final las pruebas estan correctas. Pero como veredicto final tendríamos que revisar el código y ver porque no cumple como debería.