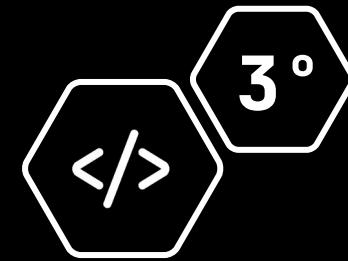




**PRÁCTICA 2. TEST DRIVEN
DEVELOPMENT**
**AMPLIACIÓN DE INGENIERÍA DEL
SOFTWARE**
TERCER CURSO
DIEGO PICAZO GARCÍA
LARA FERNÁNDEZ GUTIÉRRREZ

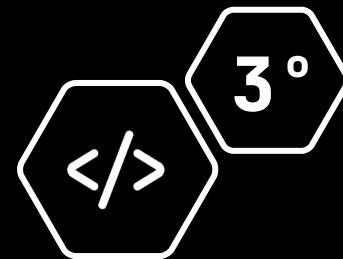




PRÁCTICA 2. TEST DRIVEN DEVELOPMENT

AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE

CASO INICIAL



ÍNDICE

- [Caso de Prueba 1](#)
- [Caso de Prueba 2](#)
- [Caso de Prueba 3](#)
- [Caso de Prueba 4](#)
- [Caso de Prueba 5](#)
- [Caso de Prueba 6](#)
- [Caso de Prueba 7](#)

TEST1.CE

```

  6 usages new *
3   public class Card {
      2 usages new *
4     public Card(String name, int attack, int defense, Position position) {
5       |
6       |
1 usage Laara2705 +2 *
3   public class Combat {
        1 usage Laara2705 +1 *
4     @ public static String combat(Card c1, Card c2) {
5       return null;
6     }
7   }

```

```

  3 usages Laara2705 *
public enum Position {
  2 usages
Attack, Defense;
}

```

```

1 package es.codeurjc.ais;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.DisplayName;
5 import org.junit.jupiter.api.Test;
6 no usages ER4UG.reverse +1 *
7 @DisplayName("Initial Case examples ... ")
8 public class InitialCaseTest {
9   no usages ER4UG.reverse +1 *
10  @Test
11  @DisplayName("First Example")
12  public void test1() {
13    String expected = """
14      Carta 1 (3000/2500/Posición: Ataque) vs Carta 2
15      (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500
16      puntos. Carta 2 destruido/a."";
17    Card card_1 = new Card("Carta 1", 3000, 2500, Position.Attack);
18    Card card_2 = new Card("Carta 2", 2500, 2100, Position.Attack);
19    String result = Combat.combat(card_1, card_2);
20    Assertions.assertEquals(expected, result);
21  }

```

Desarrollamos las clases de Java necesarias, con lo suficiente como para que el test compile.

```

1 package es.codeurjc.ais;
2
3 import org.junit.jupiter.api.Assertions;
4 import org.junit.jupiter.api.DisplayName;
5 import org.junit.jupiter.api.Test;
6 no usages ↗ ER4UG.reverse +1 *
7 @DisplayName("Initial Case examples ...")
8
9 ✘ public class InitialCaseTest {
10    no usages ↗ ER4UG.reverse +1 *
11    @Test
12    @DisplayName("First Example")
13    public void test1() {
14        String expected = """
15            Carta 1 (3000/2500/Posición: Ataque) vs Carta 2
16            (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500
17            puntos. Carta 2 destruido/a."""
18        Card card_1 = new Card( name: "Carta 1", attack: 3000, defense: 2500, Position.Attack);
19        Card card_2 = new Card( name: "Carta 2", attack: 2500, defense: 2100, Position.Attack);
20        String result = Combat.combat(card_1, card_2);
21        Assertions.assertEquals(expected, result);
22    }
23
24
25
26

```



Una vez desarrollado la compilación, tenemos un "AssertionFailedError" al correr el test que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado ningún cuerpo en las clases.

TEST1.AC

```
1 usage  ↗ Laara2705 +2
3
public class Combat {
    1 usage  ↗ Laara2705 +1
4 @
    public static String combat(Card c1, Card c2) {
        return "Carta 1 (3000/2500/Posición: Ataque) vs Carta 2\n" +
            "(2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500\n" +
            "puntos. Carta 2 destruido/a.";
8 }
8
    @Test
9     @DisplayName("First Example")
10    public void test1() {
11        String expected = """
12             Carta 1 (3000/2500/Posición: Ataque) vs Carta 2
13             (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500
14             puntos. Carta 2 destruido/a."";
15        Card card_1 = new Card( name: "Carta 1", attack: 3000, defense: 2500, Position.Attack);
16        Card card_2 = new Card( name: "Carta 2", attack: 2500, defense: 2100, Position.Attack);
17        String result = Combat.combat(card_1, card_2);
18        Assertions.assertEquals(expected, result);
19    }
```

Esta implementación mínima nos hace pasar el test. No tenemos nada que refactorizar ya que simplemente retornamos un String.



Tests passed: 1 of 1 test – 11ms

/Library/Java/JavaVirtualMachines/jdk-17.jdk/Con
Process finished with exit code 0

```
7 @DisplayName("Initial Case examples ...") 13
8 ✅ public class InitialCaseTest { 14
9     5 usages
10    static String expected; 15
11    5 usages
12    static String result; 16
13    5 usages
14    static Card card_1; 17
15    5 usages
16    static Card card_2; 18
17
18 }
```

```
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
```

```
32 @Test
33 @DisplayName("Second Example")
34 public void test2(){
35     expected = """
36         Carta 1 (1200/1000/Posición: Ataque) vs Carta 2
37         (1500/1500/Posición: Ataque) → Gana Carta 2. Atacante pierde 200
38         puntos. Carta 1 destruido/a."""
39
40
41
42
43 }
```

En este caso, no tenemos problemas de compilación en el segundo test pero si que refactorizamos ya que encontramos que ciertas variables pueden ser reutilizadas por cada test.

```
36
37
38
39
40
41
42
43
```

TEST2.WA

```

23     expected = """
24         Carta 1 (3000/2500/Posición: Ataque) vs Carta 2
25         (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500
26         puntos. Carta 2 destruido/a."";
27     card_1 = new Card( name: "Carta 1", attack: 3000, defense: 2500, Position.Attack);
28     card_2 = new Card( name: "Carta 2", attack: 2500, defense: 2100, Position.Attack);
29     result = Combat.combat(card_1, card_2);
30     Assertions.assertEquals(expected, result);
31 }
32 no usages ↗gu4re +1 *
33 @Test
34 @DisplayName("Second Example")
35 public void test2(){
36     expected = """
37         Carta 1 (1200/1000/Posición: Ataque) vs Carta 2
38         (1500/1500/Posición: Ataque) → Gana Carta 2. Atacante pierde 200
39         puntos. Carta 1 destruido/a."";
40     card_1 = new Card( name: "Carta 1", attack: 1200, defense: 1000, Position.Attack);
41     card_2 = new Card( name: "Carta 2", attack: 1500, defense: 1500, Position.Attack);
42     result = Combat.combat(card_1, card_2);
43     Assertions.assertEquals(expected, result);
44 }
```

Tenemos un *"AssertFailedError"* que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un cuerpo válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método *"combat"* para que pase el test.

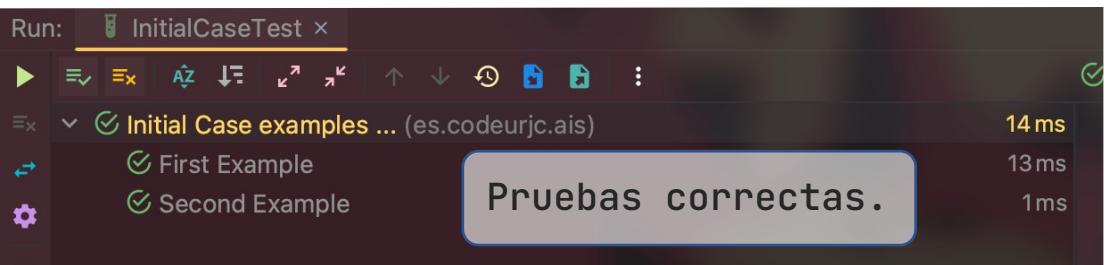


TEST2.AC

```

1 package es.codeurjc.ais;
2
3     2 usages ↗gu4re +2 *
4     public class Combat {
5         4 usages
6         private static final StringBuilder combat_resolution = new StringBuilder();
7         no usages ↗gu4re
8         private Combat(){}
9         2 usages ↗gu4re +1 *
10        @
11        public static String combat(Card c1, Card c2) {
12            // Clear StringBuilder
13            combat_resolution.setLength(0);
14            if (c1.getAttack() > c2.getAttack()) {
15                combat_resolution.append("""
16                    Carta 1 (3000/2500/Posición: Ataque) vs Carta 2
17                    (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500
18                    puntos. Carta 2 destruido/a."");
19            } else {
20                combat_resolution.append("""
21                    Carta 1 (1200/1000/Posición: Ataque) vs Carta 2
22                    (1500/1500/Posición: Ataque) → Gana Carta 2. Atacante pierde 200
23                    puntos. Carta 1 destruido/a."");
24            }
25            return combat_resolution.toString();
26        }
27    }

```



Esta implementación mínima nos hace pasar el segundo test. Hemos refactorizado la clase Combat y es que hemos introducido un StringBuilder ya que hace que el String sea más maleable. Luego comparando el ataque de ambas cartas somos capaces de saber si estamos en el caso donde C1 tiene mayor ataque o por el contrario, es C2 quien gana.

```

3     public class Card {
4         2 usages
5         private final int attack;
6         4 usages ↗gu4re +1
7         public Card(String name, int attack, int defense, Position position) {
8             this.attack = attack;
9         }
10        2 usages ↗gu4re +1
11        public int getAttack() {
12            return attack;
13        }

```

Implementación del atributo del ataque de la carta.

TEST3.CE

En este caso, no tenemos problemas de compilación en el tercer test pero si que refactorizamos el nombre del enumerado "Position" para que cumpla con los estándares de Java.

```
1 package es.codeurjc.ais;
2
3     7 usages ↗ER4UG.reverse *
4         public enum Position {
5             6 usages
6                 ATTACK, DEFENSE;
7             }
8 }
```

```
44
45 @Test
46 @DisplayName("Third Example")
47 public void test3(){
48     expected = """
49         Carta 1 (2000/0/Posición: Ataque) vs Carta 2
50         (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."";
51
52     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0, Position.ATTACK);
53     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.ATTACK);
54     result = Combat.combat(card_1, card_2);
55     Assertions.assertEquals(expected, result);
56 }
```

TEST3.WA

```

34 public void test2(){
35     expected = """
36         Carta 1 (1200/1000/Posición: Ataque) vs Carta 2
37         (1500/1500/Posición: Ataque) → Gana Carta 2. Atacante pierde 200
38         puntos. Carta 1 destruido/a."";
39     card_1 = new Card( name: "Carta 1", attack: 1200, defense: 1000, Position.ATTACK);
40     card_2 = new Card( name: "Carta 2", attack: 1500, defense: 1500, Position.ATTACK);
41     result = Combat.combat(card_1, card_2);
42     Assertions.assertEquals(expected, result);
43 }
44 no usages new *
45 @Test
46 @DisplayName("Third Example")
47 public void test3(){
48     expected = """
49         Carta 1 (2000/0/Posición: Ataque) vs Carta 2
50         (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."";
51     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0, Position.ATTACK);
52     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.ATTACK);
53     result = Combat.combat(card_1, card_2);
54     Assertions.assertEquals(expected, result);
55 }

```

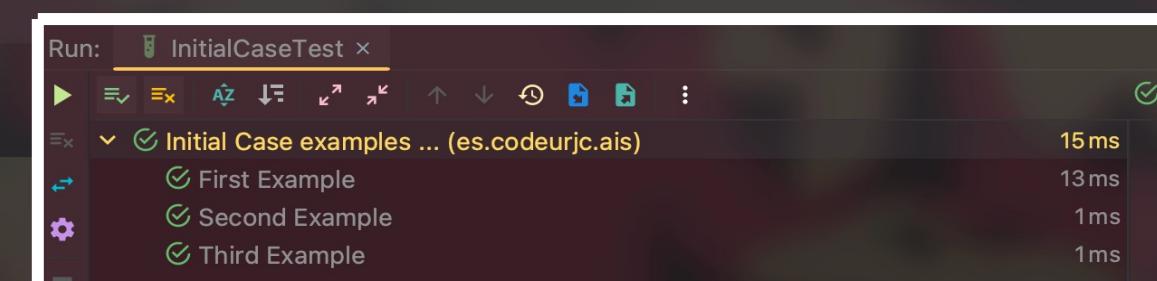
Tenemos un *"AssertFailedError"* que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un caso válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método *"combat"* para que pase el test.



TEST3.AC

```
6 @public static String combat(Card c1, Card c2) {  
7     // Clear StringBuilder  
8     combat_resolution.setLength(0);  
9     // Attacker > Defense  
10    if (c1.getAttack() > c2.getAttack()) {  
11        combat_resolution.append("""  
12            Carta 1 (3000/2500/Posición: Ataque) vs Carta 2  
13            (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500  
14            puntos. Carta 2 destruido/a."");  
15    // Attacker < Defense  
16    } else if (c1.getAttack() < c2.getAttack()){  
17        combat_resolution.append("""  
18            Carta 1 (1200/1000/Posición: Ataque) vs Carta 2  
19            (1500/1500/Posición: Ataque) → Gana Carta 2. Atacante pierde 200  
20            puntos. Carta 1 destruido/a."");  
21    // Attacker = Defense  
22    } else{  
23        combat_resolution.append("""  
24            Carta 1 (2000/0/Posición: Ataque) vs Carta 2  
25            (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."");  
26    }  
27    return combat_resolution.toString();  
28}
```

Hemos añadido el caso en el que tanto los puntos del atacante como los del defensor sean iguales consiguiendo así que el test sea correcto.



```
no usages new *
55 @Test
56 @DisplayName("Fourth Example")
57 public void test4(){
58     expected = """
59         Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
60         (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."";
61     card_1 = new Card( name: "Carta 1", attack: 1501, defense: 2850, Position.ATTACK);
62     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
63     result = Combat.combat(card_1, card_2);
64     Assertions.assertEquals(expected, result);
65 }
66 }
```

En este caso, no tenemos problemas de compilación en el cuarto test. Por lo tanto, damos paso a ejecutar el test y ver posteriormente qué ocurre.

TEST4.WA

```

45 @DisplayName("Third Example")
46 public void test3(){
47     expected = """
48         Carta 1 (2000/0/Posición: Ataque) vs Carta 2
49             (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."";
50     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0, Position.ATTACK);
51     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.ATTACK);
52     result = Combat.combat(card_1, card_2);
53     Assertions.assertEquals(expected, result);
54 }
55 no usages new *
56 @Test
57 @DisplayName("Fourth Example")
58 public void test4(){
59     expected = """
60         Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
61             (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."";
62     card_1 = new Card( name: "Carta 1", attack: 1501, defense: 2850, Position.ATTACK);
63     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
64     result = Combat.combat(card_1, card_2);
65     Assertions.assertEquals(expected, result);
66 }

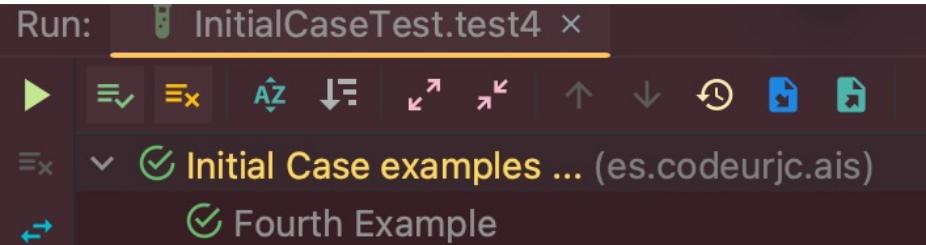
```



Tenemos un “*AssertionFailedError*” que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un caso válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método “*combat*” así como la implementación de la defensa para que pase el test.

```
1 package es.codeurjc.ais;
2
3     12 usages gu4re+1*
4 public class Card {
5     2 usages
6         private final int attack;
7     2 usages
8         private final Position position;
9     8 usages gu4re+1*
10    public Card(String name, int attack, int defense, Position position) {
11        this.attack = attack;
12        this.position = position;
13    }
14    4 usages gu4re+1
15    public int getAttack() {
16        return attack;
17    }
18    1 usage new*
19    public Position getPosition(){
20        return position;
21    }
22 }
```

Hemos implementado el position dentro de la clase de la carta.



```
public static String combat(Card c1, Card c2) {
    // Clear StringBuilder
    combat_resolution.setLength(0);
    // Defense is in attack mode
    if (c2.getPosition().equals(Position.ATTACK)) {
        // Attacker > Defense
        if (c1.getAttack() > c2.getAttack()) {
            combat_resolution.append("""
                Carta 1 (3000/2500/Posición: Ataque) vs Carta 2
                (2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500
                puntos. Carta 2 destruido/a.""");
        }
        // Attacker < Defense
    } else if (c1.getAttack() < c2.getAttack()) {
        combat_resolution.append("""
            Carta 1 (1200/1000/Posición: Ataque) vs Carta 2
            (1500/1500/Posición: Ataque) → Gana Carta 2. Atacante pierde 200
            puntos. Carta 1 destruido/a.""");
    }
    // Attacker = Defense
} else {
    combat_resolution.append("""
        Carta 1 (2000/0/Posición: Ataque) vs Carta 2
        (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."""");
}
// Defense is in defense mode
else{
    combat_resolution.append("""
        Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
        (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."""");
}
return combat_resolution.toString();
}
```

TEST4.AC

Hemos añadido el caso en el que el defensor tome la posición de defensa.

TEST5.CE

```
57 public void test4(){
58     expected = """
59         Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
60         (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."""";
61     card_1 = new Card( name: "Carta 1", attack: 1501, defense: 2850, Position.ATTACK);
62     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
63     result = Combat.combat(card_1, card_2);
64     Assertions.assertEquals(expected, result);
65 }
66 no usages new *
67 @Test
68 @DisplayName("Fifth Example")
69 public void test5(){
70     expected = """
71         Carta 1 (2000/2850/Posición: Ataque) vs Carta 2
72         (0/3000/Posición: Defensa) → Gana Carta 2. Atacante pierde 1000
73         puntos."""";
74     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850, Position.ATTACK);
75     card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000, Position.DEFENSE);
76     result = Combat.combat(card_1, card_2);
77     Assertions.assertEquals(expected, result);
78 }
```

En este caso, no tenemos problemas de compilación en el quinto test. Por lo tanto, damos paso a ejecutar el test y ver posteriormente qué ocurre.

TEST5.WA

```

57 public void test4(){
58     expected = """
59         Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
60         (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."""";
61     card_1 = new Card( name: "Carta 1", attack: 1501, defense: 2850, Position.ATTACK);
62     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
63     result = Combat.combat(card_1, card_2);
64     Assertions.assertEquals(expected, result);
65 }
66 no usages new *
67 @Test
68 @DisplayName("Fifth Example")
69 public void test5(){
70     expected = """
71         Carta 1 (2000/2850/Posición: Ataque) vs Carta 2
72         (0/3000/Posición: Defensa) → Gana Carta 2. Atacante pierde 1000
73         puntos."""";
74     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850, Position.ATTACK);
75     card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000, Position.DEFENSE);
76     result = Combat.combat(card_1, card_2);
77     Assertions.assertEquals(expected, result);
78 }
```

Tenemos un *"AssertFailedError"* que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un caso válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método *"combat"*.



```
14 usages &gu4re+1
3   public class Card {
4     2 usages
5       private final int attack;
6     2 usages
7       private final int defense;
8     2 usages
9       private final Position position;
10    10 usages &gu4re+1*
11      public Card(String name, int attack, int defense, Position position) {
12        this.attack = attack;
13        this.position = position;
14        this.defense = defense;
15      }
16      5 usages &gu4re+1
17      public int getAttack() {
18        return attack;
19      }
20      1 usage new *
21      public Position getPosition(){
22        return position;
23      }
24      1 usage new *
25      public int getDefense(){
26        return defense;
27      }
28    }
```

Run: InitialCaseTest.test5 ×

Initial Case examples ... (es.codeurjc.ais) 11ms

Fifth Example 11ms

Hemos implementado la defensa dentro de la clase de la carta. Además hemos añadido el nuevo caso al cuerpo del else dentro de la clase "combat" que corresponde a cuando el defensor gana por puntos de defensa.

```
30      // Defense is in defense mode
31      else{
32        if (c1.getAttack() > c2.getDefense()){
33          combat_resolution.append("""
34            Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
35            (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."");
36        }
37        else{
38          combat_resolution.append("""
39            Carta 1 (2000/2850/Posición: Ataque) vs Carta 2
40            (0/3000/Posición: Defensa) → Gana Carta 2. Atacante pierde 1000
41            puntos."");
42        }
43      }
```



```
68 public void test5(){
69     expected = """
70         Carta 1 (2000/2850/Posición: Ataque) vs Carta 2
71         (0/3000/Posición: Defensa) → Gana Carta 2. Atacante pierde 1000
72         puntos."";
73     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850, Position.ATTACK);
74     card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000, Position.DEFENSE);
75     result = Combat.combat(card_1, card_2);
76     Assertions.assertEquals(expected, result);
77 }
78 no usages new *
79 @Test
80 @DisplayName("Sixth Example")
81 public void test6(){
82     expected = """
83         Carta 1 (1500/2850/Posición: Ataque) vs Carta 2
84         (2000/1500/Posición: Defensa) → Empate."";
85     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.ATTACK);
86     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
87     result = Combat.combat(card_1, card_2);
88     Assertions.assertEquals(expected, result);
89 }
```

TEST6.CE

En este caso, no tenemos problemas de compilación en el sexto test. Por lo tanto, damos paso a ejecutar el test y ver posteriormente qué ocurre.



A1 A7 A45 ↑ ↓

TEST6.WA

```
68 public void test5(){
69     expected = """
70         Carta 1 (2000/2850/Posición: Ataque) vs Carta 2
71         (0/3000/Posición: Defensa) → Gana Carta 2. Atacante pierde 1000
72         puntos."";
73     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850, Position.ATTACK);
74     card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000, Position.DEFENSE);
75     result = Combat.combat(card_1, card_2);
76     Assertions.assertEquals(expected, result);
77 }
78 no usages new *
79 @Test
80 @DisplayName("Sixth Example")
81 public void test6(){
82     expected = """
83         Carta 1 (1500/2850/Posición: Ataque) vs Carta 2
84         (2000/1500/Posición: Defensa) → Empate."";
85     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.ATTACK);
86     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
87     result = Combat.combat(card_1, card_2);
88     Assertions.assertEquals(expected, result);
89 }
```

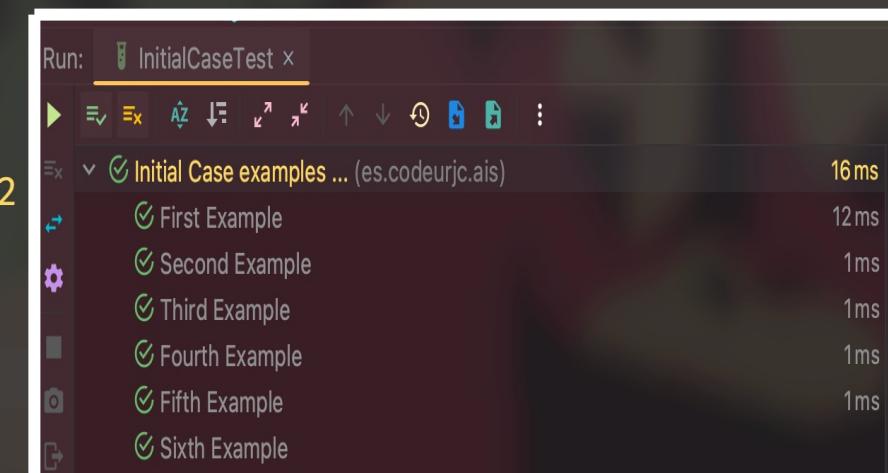
```
org.opentest4j.AssertionFailedError: ...
Debug 'InitialCaseTest.test6' ⌂ More actions...
  org.junit.jupiter.api.Assertions
    public static void assertEquals(
        Object expected,
        Object actual
    )
  Maven: org.junit.jupiter:junit-jupiter-api:5.9.2
(junit-jupiter-api-5.9.2.jar)
```

Tenemos un *"AssertionFailedError"* que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un caso válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método *"combat"*.

TEST6.AC

```
30 // Defense is in defense mode
31 else{
32     // Attack > Defense
33     if (c1.getAttack() > c2.getDefense()){
34         combat_resolution.append("""
35             Carta 1 (1501/2850/Posición: Ataque) vs Carta 2
36             (2000/1500/Posición: Defensa) → Gana Carta 1. Carta 2 destruido/a."");
37     }
38     // Attack < Defense
39     else if (c1.getAttack() < c2.getDefense()){
40         combat_resolution.append("""
41             Carta 1 (2000/2850/Posición: Ataque) vs Carta 2
42             (0/3000/Posición: Defensa) → Gana Carta 2. Atacante pierde 1000
43             puntos."");
44     }
45     // Attack = Defense
46     else{
47         combat_resolution.append("""
48             Carta 1 (1500/2850/Posición: Ataque) vs Carta 2
49             (2000/1500/Posición: Defensa) → Empate."");
50     }
51 }
52 return combat_resolution.toString();
```

Hemos implementado el caso en el que el ataque es igual que la defensa cuando el defensor está en modo defensa.



TEST7.CE

```

78 no usages new *
79 @Test
80 @DisplayName("Sixth Example")
81 public void test6(){
82     expected = """
83         Carta 1 (1500/2850/Posición: Ataque) vs Carta 2
84         (2000/1500/Posición: Defensa) → Empate."";
85     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.ATTACK);
86     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
87     result = Combat.combat(card_1, card_2);
88     Assertions.assertEquals(expected, result);
89 }
90 no usages new *
91 @Test
92 @DisplayName("Seventh Example")
93 public void test7(){
94     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.DEFENSE);
95     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
96     Assertions.assertThrows(IllegalPositionException.class, () →
97         result = Combat.combat(card_1, card_2)
98     );
}

```

En este caso, nuestro séptimo test arroja un “*Compilation Error*” ya que la excepción que buscamos arrojar en este test es personalizada y no está definida por Java. Procedemos a su creación para poder ejecutar el test.

```

1 usage new *
2
3 public class IllegalPositionException extends Exception{
4     no usages new *
5     public IllegalPositionException(){}
6         super();
7     }
8     no usages new *
9     public IllegalPositionException(String msg){
10        super(msg);
11    }
12 }

```

TEST7.WA

```

78
79
80  @Test
81 @DisplayName("Sixth Example")
82 public void test6(){
83     expected = """
84         Carta 1 (1500/2850/Posición: Ataque) vs Carta 2
85         (2000/1500/Posición: Defensa) → Empate.""";
86     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.ATTACK);
87     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
88     result = Combat.combat(card_1, card_2);
89     Assertions.assertEquals(expected, result);
90 }
91  no usages new *
92 @Test
93 @DisplayName("Seventh Example")
94 public void test7(){
95     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.DEFENSE);
96     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
97     Assertions.assertThrows(IllegalPositionException.class, () ->
98         result = C

```

Tenemos un “*AssertFailedError*” que nos comenta que el *assertThrows* se esperaba una excepción del tipo “*IllegalPositionException*”, sin embargo, nada fue lanzado. Procedemos a su respectiva implementación en el método *combat*.

org.opentest4j.AssertionFailedError: Expected es.codeurjc.ais.IllegalPositionException to be thrown, but nothing was thrown.

Debug 'InitialCaseTest.test7' More actions... ⚡

org.junit.jupiter.api.Assertions

```
public static <T extends Throwable> T assertThrows(
    @NotNull Class<T> expectedType,
    org.junit.jupiter.api.function.Executable executable
)
```

Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

TEST7.AC

```
@Test  
@DisplayName("Sixth Example")  
public void test6(){  
    expected = "";  
    Carta 1 (1500/2850/Posición: Ataque) vs Carta 2 (2000/1500/Posición: Defensa).  
    card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.ATTACK);  
    card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);  
    Assertions.assertDoesNotThrow(() -> {  
        result = Combat.combat(card_1, card_2);  
        Assertions.assertEquals(expected, result);  
    });  
}  
  
    First Example  
    Second Example  
    Third Example  
    Fourth Example  
    Fifth Example  
    Sixth Example  
    Seventh Example  
    18 ms  
    13 ms  
    1 ms  
    1 ms  
    1 ms  
    1 ms  
    1 ms  
    1 ms  
    (*)Test 2  
    ón: Ataque) vs Carta 2  
    ue) → Gana Carta 2. Atacante pierde 200  
    puntos. Carta 1 destruida/a."";  
    card_1 = new Card( name: "Carta 1", attack: 1200, defense: 1000, Position.ATTACK);  
    card_2 = new Card( name: "Carta 2", attack: 1500, defense: 1500, Position.ATTACK);  
    Assertions.assertDoesNotThrow(() -> {  
        result = Combat.combat(card_1, card_2);  
        Assertions.assertEquals(expected, result);  
    });  
}
```

```
public static String combat(Card c1, Card c2) throws IllegalPositionException{
```

```
// Clear StringBuilder  
combat_resolution.setLength(0);  
// Attacker is in defense mode  
if (c1.getPosition().equals(Position.DEFENSE))  
    throw new IllegalPositionException();  
// Attacker is in attack mode  
else {
```

Simplemente para pasar el séptimo test, hemos permitido que el método combat pueda lanzar una "IllegalPositionException" cumpliéndose que si el atacante está en modo defensa, se lance. Esto provoca un "Compilation Error" en todos los tests anteriores al no manejar la excepción mencionada, por lo que simplemente hemos englobado la acción de combatir dentro de un assertDoesNotThrow en los tests del 1 al 6, ya que lo esperado es un String de retorno y no una excepción. De esta forma controlamos también otros posibles casos.

```

99
100 @Nested
101 @DisplayName("Seventh Example")
102 class SeventhExample {
103     no usages ✎gu4re
104     @BeforeEach
105     public void setUp(){
106         card_1 = null;
107         card_2 = null;
108     }
109     no usages ✎gu4re
110     @Test
111     @DisplayName("Exception thrown")
112     public void test7(){
113         card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.DEFENSE);
114         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);
115         Assertions.assertThrows(IllegalPositionException.class, () →
116             Combat.combat(card_1, card_2)
117         );
118     }
119 }

```

...(*) de esta forma, el séptimo test solo es capaz de utilizar las variables card_1 y card_2 de la clase principal al ser de tipo protected, ya que este test recordemos que solo son necesarios estos atributos para provocar el lanzamiento de la excepción que buscamos que entregue por parte del método combat. Al final no deja de ser una refactorización ideológica.

REFACTORIZACIÓN INTERMEDIA

Hemos realizado una refactorización intermedia antes de continuar con los casos intermedios. En la clase de los casos iniciales, por optimización de variables y sobre todo por seguridad frente a posibles vulnerabilidades, el séptimo y último ejemplo de los casos iniciales está contenido en una subclase con su propio test y su propio método @BeforeEach...(*)

5	<code>@DisplayName("Initial Case examples ... ")</code>
6	<code>public class InitialCaseTest {</code>
13 usages	
7	<code>private static String expected;</code>
13 usages	
8	<code>private static String result;</code>
16 usages	
9	<code>protected static Card card_1;</code>
16 usages	
10	<code>protected static Card card_2;</code>

```
99 @Nested  
100 @DisplayName("Seventh Example")  
101 class SeventhExample {  
102     no usages ■gu4re  
103     @BeforeEach  
104     public void setUp(){  
105         card_1 = null;  
106         card_2 = null;  
107     }  
108     no usages ■gu4re  
109     @Test  
110     @DisplayName("Exception thrown")  
111     public void test7(){  
112         card_1 = new Card( name:  
113         card_2 = new Card( name:  
114         Assertions.assertThrows  
115             );  
116     }  
117 }
```

REFACTORIZACIÓN INTERMEDIA

19 usages ER4UG.reverse +1

public enum Position {

10 usages



ATTACK, DEFENSE;

}

public static String combat(@NotNull Card c1, @NotNull Card c2) throws IllegalPositionException{

5 @DisplayName("Initial Case examples ... ")

...(*) de esta forma, el séptimo caso de prueba se basa en la utilización de las variables de clase principal al ser de tipo Card. Al final recordemos que estos atributos para provocar la excepción que buscamos que entregue por parte del método combat. Al final no deja de ser una refactorización ideológica.

También se han realizado cambios menores como la eliminación de un ';' en la clase Position, así como la adición de anotaciones @NotNull al método combat para de esa forma ayudar al desarrollador para que sepa que tanto los argumentos como el valor de retorno no deberían de ser nunca nulo.

s realizado una refactorización intermedia antes de continuar con los casos intermedios. En la clase Position se han hecho modificaciones de variables y sobre todo por seguridad frente a posibles vulnerabilidades, el primero y último ejemplo de los casos iniciales está contenido en la subclase con su propio test y el segundo en el método @BeforeEach...(*)

InitialCaseTest {

static String expected;

private static String result;

protected static Card card_1;

16 usages

protected static Card card_2;

9

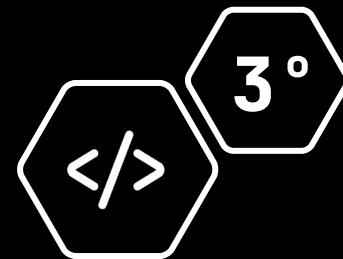
10



PRÁCTICA 2. TEST DRIVEN DEVELOPMENT

AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE

CASO INTERMEDIO



ÍNDICE

<u>Caso de Prueba</u>	8
<u>Caso de Prueba</u>	9
<u>Caso de Prueba</u>	10
<u>Caso de Prueba</u>	11
<u>Caso de Prueba</u>	12
<u>Caso de Prueba</u>	13
<u>Caso de Prueba</u>	14
<u>Caso de Prueba</u>	15
<u>Caso de Prueba</u>	16
<u>Caso de Prueba</u>	17

TEST8.CE

```
21 @Test
22 @DisplayName("Eighth Example")
23 public void test8(){
24     expected = """
25         Carta 1 (2000/2850/Posición: Ataque/Efecto: Inmortal) vs Carta 2
26         (2000/1500/Posición: Ataque/Efecto: N/A) → Empate.
27         Carta 2 destruido/a."";
28     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850,
29                       Position.ATTACK, Position.EFFECT.INMORTAL);
30     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
31                       Position.ATTACK, Position.EFFECT.NA);
32     Assertions.assertDoesNotThrow(() → {
33         result = Combat.combat(card_1, card_2);
34         Assertions.assertEquals(expected, result);
35     });
36 }
```

```
3 26 usages ↗ER4UG.reverse *
4 public enum Position {
5     12 usages
6     ATTACK, DEFENSE;
7     4 usages new *
8     enum EFFECT{
9         1 usage
10        IMMORTAL, NA
11    }
12 }
```

```
13     private Position.EFFECT effect;
14     15 usages ↗gu4re *
15     public Card(String name, int attack, int defense, @NotNull Position position) {
16         this.attack = attack;
17         this.position = position;
18         this.defense = defense;
19         this.effect = Position.EFFECT.NA;
20     }
21     2 usages new *
22     public Card(String name, int attack, int defense, @NotNull Position position,
23                 Position.EFFECT effect) {
24         this(name, attack, defense, position);
25         this.effect = effect;
26     }
27 }
```

Al comenzar el octavo test nos encontramos con un nuevo atributo, y es que ahora las cartas, a parte de atacar o defender, pueden tener efectos. Tenemos un "Compilation Error" debido a que debemos generar esta nueva característica dentro de la clase Position y dentro de la clase Card.

TEST8.WA

```

17     result = "";
18     card_1 = null;
19     card_2 = null;
20 }
21 no usages ↗gu4re*
22 @Test
23 @DisplayName("Eighth Example")
24 public void test8(){
25     expected = """
26         Carta 1 (2000/2850/Posición: Ataque/Efecto: Inmortal) vs Carta 2
27         (2000/1500/Posición: Ataque/Efecto: N/A) → Empate.
28         Carta 2 destruido/a."""
29     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850,
30                         Position.ATTACK, Position.EFFECT.IMMORTAL);
31     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
32                         Position.ATTACK, Position.EFFECT.NA);
33     Assertions.assertDoesNotThrow(() → {
34         result = Combat.combat(card_1, card_2);
35         Assertions.assertEquals(expected, result);
36     });
37 }
38

```

En este caso, nos falla el test ya que se esperaba un caso, y se devolvió otra cosa distinta. De hecho, el resultado del combate es empate, pero como podemos ver, la diferencia está en que al usarse el efecto INMORTAL pues solo se destruye la carta 2 y no ambas.

```

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1
(2000/2850/Posición: Ataque/Efecto: Inmortal) vs Carta 2
(2000/1500/Posición: Ataque/Efecto: N/A) -> Empate.
Carta 2 destruido/a.> but was: <Carta 1 (2000/0/Posición: Ataque) vs Carta 2
(2000/1500/Posición: Ataque) -> Empate. Ambas cartas destruidas.>
Debug 'IntermediateCaseTest...' ⌂ More actions...
 org.junit.jupiter.api.Assertions
@API(status = Status.STABLE, since = "5.2")
public static void assertDoesNotThrow(
    @NotNull org.junit.jupiter.api.function.Executable executable
)
Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

```

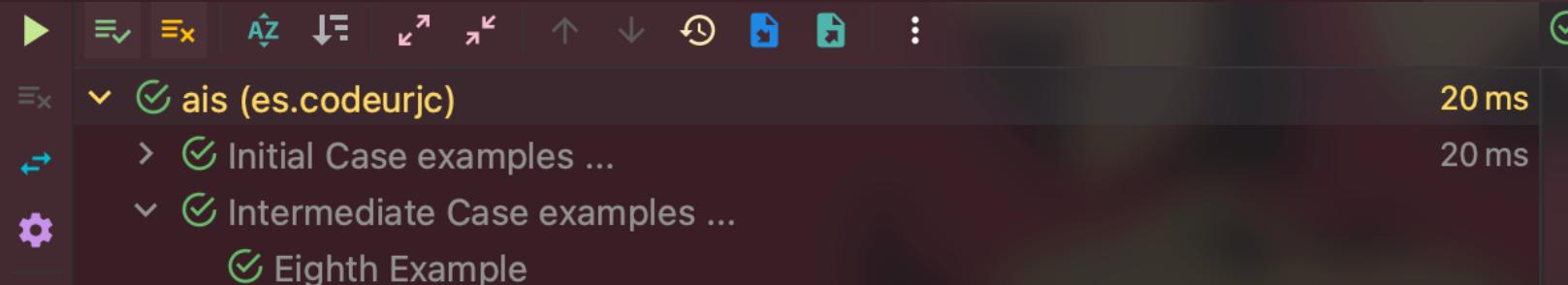
```

32 // Attack = Defense
33 } else {
34     // C1 has IMMORTAL
35     if(c1.getEffect().equals(Position.EFFECT.IMITARIAL)){
36         combat_resolution.append(""""
37             Carta 1 (2000/2850/Posición: Ataque/Efecto: Inmortal) vs Carta 2
38             (2000/1500/Posición: Ataque/Efecto: N/A) → Empate.
39             Carta 2 destruido/a."");
40     } else{
41         combat_resolution.append(""""
42             Carta 1 (2000/0/Posición: Ataque) vs Carta 2
43             (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."");
44     }
45 }
```

Hemos hecho una implementación mínima para que el test pase, sin embargo, vamos a tener que hacer una serie de refactorizaciones futuras antes de pasar al siguiente test debido a problemas como "Cognitive Complexity" así como el tratamiento de este nuevo atributo llamado "EFFECT".

```

32     public Position.EFFECT getEffect(){
33         return effect;
34     }
35 }
```



```
public static @NotNull String combat(@NotNull Card c1, @NotNull Card c2) throws IllegalPositionException,
    UnsupportedOperationException {
    final String NSY = "Not supported yet";
    // Clear StringBuilder
    combat_resolution.setLength(0);
    if (c1.getPosition() == Position.DEFENSE)
        throw new IllegalPositionException("La carta atacante no puede estar en una posición de Defensa");
    combat_resolution.append(String.format("Carta 1 (%d/%d/Posición: Ataque", c1.getAttack(), c1.getDefense()));
    //combat_resolution.append(String.format("/Efecto: %s", c1.getEffect().toString() != null ? c1.getEffect().toString() : ""));
    if (c1.getEffect() != null)
        combat_resolution.append(String.format("/Efecto: %s", c1.getEffect().toString()));
    combat_resolution.append(") vs ");
    switch (c2.getPosition()) {
        case ATTACK -> {
            combat_resolution.append(String.format("Carta 2 (%d/%d/Posición: Ataque", c2.getAttack(), c2.getDefense()));
            if (c2.getEffect() != null)
                combat_resolution.append(String.format("/Efecto: %s", c2.getEffect().toString()));
            combat_resolution.append(") → ");
            switch (Integer.signum(Integer.compare(c1.getAttack(), c2.getAttack()))) {
                case -1 -> combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");
                case 0 -> {
                    if (c1.getEffect() != null && c1.getEffect().equals(Position.EFFECT.ETERNAL))
                        combat_resolution.append("Empate. Carta 2 destruido/a.");
                    else
                        combat_resolution.append("Empate. Ambas cartas destruidas.");
                }
                case 1 -> combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");
                default -> throw new UnsupportedOperationException(NSY);
            }
        }
        case DEFENSE -> {
            combat_resolution.append(String.format("Carta 2 (%d/%d/Posición: Defensa) → ", c2.getAttack(), c2.getDefense()));
            switch (Integer.signum(Integer.compare(c1.getAttack(), c2.getDefense()))) {
                case -1 -> combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");
                case 0 -> combat_resolution.append("Empate.");
                case 1 -> combat_resolution.append("Gana Carta 1. Carta 2 destruido/a.");
                default -> throw new UnsupportedOperationException(NSY);
            }
        }
    default -> throw new UnsupportedOperationException(NSY);
    }
    return combat_resolution.toString();
}
```

Refactorización de combat para darle una mejor visibilidad al código evitando "Cognitive Complexity".

REFACTORIZACIÓN POSTERIOR

Constructor de Card cambiado, añadido Tags para la ayuda y se ha considerado, que para ser puristas, en los Test del 1 al 7, EFFECT no existe y por lo tanto vale null. Además, por tratamiento y desuso, se ha modificado los "expected" de los Test del caso inicial así como la eliminación del constructor de la excepción personalizada, respectivamente.

```
13     public Card(String name, int attack, int defense, @NotNull Position position) {
14         this.attack = attack;
15         this.position = position;
16         this.defense = defense;
17         this.effect = null;
18     }
19     2 usages  ↗gu4re +1
20     public Card(String name, int attack, int defense, @NotNull Position position,
21                 @Nullable Position.EFFECT effect) {
22         this(name,attack,defense,position);
23         this.effect = effect;
24     }
25 }
```

```
public void test1() { // escogido el test 1 como ejemplo
    expected = "Carta 1 (3000/2500/Posición: Ataque) vs Carta
    "→ Gana Carta 1. Defensor pierde 500 puntos. " +
    "Carta 2 destruido/a.";
```

```
public class IllegalPositionException extends Exception{  
    1 usage   ↗ gu4re  
    public IllegalPositionException(String msg) { super(m  
}
```

TEST9.CE

```
► @Test  
  @DisplayName("Ninth Example")  
  public void test9(){  
    expected = "Carta 1 (1800/2850/Posición: Ataque/Efecto: IMMORTAL) vs Carta 2 " +  
              "(2000/1500/Posición: Ataque/Efecto: NA) → Gana Carta 2. Atacante pierde 200 puntos."  
    card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,  
                      Position.ATTACK, Position.EFFECT.IMMORTAL);  
    card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,  
                      Position.ATTACK, Position.EFFECT.NA);  
    Assertions.assertDoesNotThrow(() → {  
      result = Combat.combat(card_1, card_2);  
      Assertions.assertEquals(expected, result);  
    });  
  }  
}
```

💡 En este caso, no tenemos problemas de compilación en el noveno test. Por lo tanto, damos paso a ejecutar el test y ver posteriormente qué ocurre.

TEST9.WA

```

24
25     public void test8(){
26         expected = "Carta 1 (2000/2850/Posición: Ataque/Efecto: IMMORTAL) vs Carta 2 " +
27             "(2000/1500/Posición: Ataque/Efecto: NA) → Empate. Carta 2 destruido/a.";
28         card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850,
29             Position.ATTACK, Position.EFFECT.IMMORTAL);
30         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
31             Position.ATTACK, Position.EFFECT.NA);
32         Assertions.assertDoesNotThrow(() -> {
33             result = Combat.combat(card_1, card_2);
34             Assertions.assertEquals(expected, result);
35         });
36     }
37     no usages new *
38
39     @Test
40     @DisplayName("Ninth Example")
41     public void test9(){
42         expected = "Carta 1 (1800/2850/Posición: Ataque/Efecto: IMMORTAL) vs Carta 2 " +
43             "(2000/1500/Posición: Ataque/Efecto: NA) → Gana Carta 2. Atacante pierde 200 puntos.";
44         card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,
45             Position.ATTACK, Position.EFFECT.IMMORTAL);
46         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
47             Position.ATTACK, Position.EFFECT.NA);
48         Assertions.assertDoesNotThrow(() -> {
49             result = Combat.combat(card_1, card_2);
50             Assertions.assertEquals(expected, result);
51         });
52     }

```

En este caso, nos falla el test ya que se esperaba un caso, y se devolvió otra cosa distinta. De hecho, el resultado del combate es a favor de carta 2, pero como podemos ver, la diferencia está en que al usarse el efecto INMORTAL pues la carta 1 no debería de destruirse.

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (1800/2850/Posición:
 Debug 'IntermediateCaseTest.' Alt+Mayús+Intro More actions... Alt+Intro
 org.junit.jupiter.api.Assertions
 @API(status = Status.STABLE, since = "5.2")
 public static void assertDoesNotThrow(
 @NotNull org.junit.jupiter.api.function.Executable executable
)
 Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

TEST9.AC

```
combat_resolution.append(String.format("/Efecto: %s", c1.getEffect().toString()));
combat_resolution.append(") vs ");
switch (c2.getPosition()) {
    case ATTACK → {
        combat_resolution.append(String.format("Carta 2 (%d/%d/Posición: Ataque", c2.getAttack(), c2.getDefense()));
        if (c2.getEffect() ≠ null)
            combat_resolution.append(String.format("/Efecto: %s", c2.getEffect().toString()));
        combat_resolution.append(") → ");
        switch (Integer.signum(Integer.compare(c1.getAttack(), c2.getAttack()))) {
            case -1 → {
                if (c1.getEffect() ≠ null && c1.getEffect().equals(Position.EFFECT.IMMORTAL))
                    combat_resolution.append("Gana Carta 2. Atacante pierde 200 puntos.");
                else
                    combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");
            }
            case 0 → {
                if (c1.getEffect() ≠ null && c1.getEffect().equals(Position.EFFECT.IMMORTAL))
                    combat_resolution.append("Empate. Carta 2 destruido/a.");
                else
                    combat_resolution.append("Empate. Ambas cartas destruidas.");
            }
            case 1 → combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");
        }
    }
}
```

n: IntermediateCaseTest.test9 x

Tests passed: 1 of 1 test – 35 ms

Intermediate Case e 35 ms
Ninth Example 35 ms

"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" ...

Implementamos la mínima opción que nos permite distinguir entre carta 1 con efecto de inmortal o no cuando ésta pierde, por lo tanto, nuestro test tiene éxito. Ahora queda encargarnos de la refactorización para evitar nuestro famoso enemigo, el "Cognitive Complexity".

REFACTORIZACIÓN POSTERIOR 1

```
private static final String NOT_SUPPORTED_YET = "Not supported yet";
no usages  ↵ gu4re
private Combat(){}
1 usage  new *
private static void attackVsAttack(@NotNull Card card1, @NotNull Card card2){
    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack()))){
        case -1 → {
            if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.IMITAR))
                combat_resolution.append("Gana Carta 2. Atacante pierde 200 puntos.");
            else
                combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");
        }
        case 0 → {
            if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.IMITAR))
                combat_resolution.append("Empate. Carta 2 destruido/a.");
            else
                combat_resolution.append("Empate. Ambas cartas destruidas.");
        }
        case 1 → combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");
        default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
    }
}
1 usage  new *
private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense()))){
        case -1 → combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");
        case 0 → combat_resolution.append("Empate.");
        case 1 → combat_resolution.append("Gana Carta 1. Carta 2 destruido/a.");
        default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
}
```

En primer lugar, hemos encapsulado las resoluciones de los combates en dos métodos, uno que resuelve el conflicto Ataque vs Ataque, y otro para Ataque vs Defensa. Simplemente se ha extraído el código del anterior método combat, y se ha generalizado la variable NOT_SUPPORTED_YET para que pueda ser usada por todos los métodos de la clase. Cabe recalcar que todos estos métodos y métodos futuros son privados para que solo sea visible de puertas para fuera el método combat.

```

35
36     private static @NotNull String getCardInfo(@NotNull Card card){
37         // Local variables
38         final String CARD1_NAME = "Carta 1"; REFACTORIZACIÓN POSTERIOR 2
39         StringBuilder cardInfoBuilder = new StringBuilder();
40         int cardNumber = card.getName().equals(CARD1_NAME) ? 1 : 2;
41         cardInfoBuilder.append(String.format("Carta %d (%d/%d/Posición: %s", cardNumber, card.getAttack(),
42                                         card.getDefense(), (card.getPosition().equals(Position.ATTACK) ? "Ataque" : "Defensa")));
43         if (card.getEffect() != null)
44             cardInfoBuilder.append(String.format("/Efecto: %s", card.getEffect().toString()));
45         cardInfoBuilder.append(")");
46         return cardInfoBuilder.toString();
47     }
48
49     9 usages ↗gu4re +1 *
50
51     public static @NotNull String combat(@NotNull Card card1, @NotNull Card card2) throws IllegalPositionException {
52         UnsupportedOperationException {
53             // Local variables
54             final String ERROR_MESSAGE = "La carta atacante no puede estar en la posición de defensa";
55             final int ZERO = 0;
56             // Clear StringBuilder
57             combat_resolution.setLength(ZERO);
58             if (card1.getPosition().equals(Position.DEFENSE))
59                 throw new IllegalPositionException(ERROR_MESSAGE);
60             combat_resolution.append(getCardInfo(card1))
61                 .append(" vs ").append(getCardInfo(card2)).append(" ");
62             switch (card2.getPosition()){
63                 case ATTACK → attackVsAttack(card1, card2);
64                 case DEFENSE → attackVsDefense(card1, card2);
65                 default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
66             }
67             return combat_resolution.toString();
68         }
69     }

```

En segundo lugar, siguiendo la misma técnica que antes, hemos generalizado la información de las cartas, ahora de esta forma devolvemos la información de cada carta dependiendo de sus valores y atributos → "información de la resolución del combate". Además es visible la refactorización del método principal combat donde su esencia sigue siendo la misma solo que ahora llamando a las funciones mencionadas anteriormente. También se han evitado el uso de → "antiguas variables mágicas" ahora reemplazadas por 'CARD1_NAME', 'ZERO' y 'ERROR_MESSAGE'.

```
private final String name;  
15 usages  ↗ gu4re +1 *      REFACTORIZACIÓN POSTERIOR 3  
public Card(String name, int attack, int defense, @NotNull Position position) {  
    this.name = name;  
    this.attack = attack;  
    this.position = position;  
    this.defense = defense;  
    this.effect = null;  
}  
4 usages  ↗ gu4re +1
```

```
public Card(String name, int attack, int defense, @NotNull Position position,  
           @Nullable Position.EFFECT effect) {
```

```
    this(name, attack, defense, position);  
    this.effect = effect;  
}
```

```
1 usage  new *  
public String getName(){  
    return name;  
}
```

Y por último, como hemos mencionado anteriormente, hemos echado mano del nombre para saber qué carta tratamos en el método `getCardInfo`, por lo tanto, fue necesaria su respectiva implementación ya que previamente solo se encontraba mencionado 'name' como parámetro en los constructores.

TEST10.CE

```

39         "(2000/1500/Posición: Ataque/Efecto: NA) → Gana Carta 2. Atacante pierde 200 puntos.";
40 card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,
41                     Position.ATTACK, Position.EFFECT.IMMORTAL);
42 card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
43                     Position.ATTACK, Position.EFFECT.NA);
44 Assertions.assertDoesNotThrow(() → {
45     result = Combat.combat(card_1, card_2);
46     Assertions.assertEquals(expected, result);
47 });
48 }
49 no usages new*
50 @Test
51 @DisplayName("Tenth Example")
52 public void test10(){
53     expected = "Carta 1 (2100/2850/Posición: Ataque/Efecto: NA) vs Carta 2 " +
54             "(2000/1500/Posición: Ataque/Efecto: IMMORTAL) → Gana Carta 1. Defensor pierde 100 puntos.";
55     card_1 = new Card( name: "Carta 1", attack: 2100, defense: 2850,
56                     Position.ATTACK, Position.EFFECT.NA);
57     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
58                     Position.ATTACK, Position.EFFECT.IMMORTAL);
59     Assertions.assertDoesNotThrow(() → {
60         result = Combat.combat(card_1, card_2);
61         Assertions.assertEquals(expected, result);
62     });
63 }

```

Como podemos ver en este décimo test no tenemos problemas de compilación así que vamos a pasar a la siguiente fase, que es la ejecución del test.

TEST10.WA

```

43     Position.ATTACK, Position.EFFECT.NA);
44
45     Assertions.assertDoesNotThrow(() -> {
46         result = Combat.combat(card_1, card_2);
47         Assertions.assertEquals(expected, result);
48     });
49
50     no usages new *
51 @Test
52 @DisplayName("Tenth Example")
53 public void test10(){
54     expected = "Carta 1 (2100/2850/Posición: Ataque/Efecto: NA) vs Carta 2 " +
55                 "(2000/1500/Posición: Ataque/Efecto: IMMORTAL) -> Gana Carta 1. Defensor pierde 100 puntos.";
56     card_1 = new Card( name: "Carta 1", attack: 2100, defense: 2850,
57                         Position.ATTACK, Position.EFFECT.NA);
58     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
59                         Position.ATTACK, Position.EFFECT.IMMORTAL);
60
61     Assertions.assertDoesNotThrow(() -> {
62         result = Combat.combat(card_1, card_2);
63         Assertions.assertEquals(expected, result);
64     });
}

```

El test nos muestra que se esperaba un caso y se ha devuelto otro, ya que no estaba contemplado este nuevo caso, vamos a realizar una implementación mínima.

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (2100/2850/Posición: Ataque/Efecto: NA) vs Carta 2 (2000/1500/Posición: Ataque/Efecto: IMMORTAL) -> Gana Carta 1. Defensor pierde 100 puntos.> but was: <Carta 1 (2100/2850/Posición: Ataque/Efecto: NA) vs Carta 2 (2000/1500/Posición: Ataque/Efecto: IMMORTAL) -> Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.>

Debug 'IntermediateCaseTest...' ⌂ More actions... ⌂

org.junit.jupiter.api.Assertions

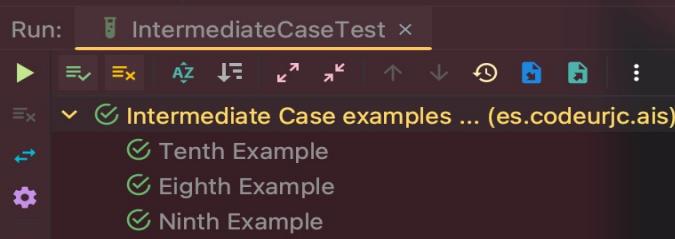
@API(status = Status.STABLE, since = "5.2")
public static void assertDoesNotThrow(
 @NotNull org.junit.jupiter.api.function.Executable executable
)

Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

```
11 private static void attackVsAttack(@NotNull Card card1, @NotNull Card card2){  
12     switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack()))){  
13         case -1 → {  
14             if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.IMITARIAL))  
15                 combat_resolution.append("Gana Carta 2. Atacante pierde 200 puntos.");  
16             else  
17                 combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");  
18         }  
19         case 0 → {  
20             if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.IMITARIAL))  
21                 combat_resolution.append("Empate. Carta 2 destruido/a.");  
22             else  
23                 combat_resolution.append("Empate. Ambas cartas destruidas.");  
24         }  
25         case 1 → {  
26             if (card2.getEffect() ≠ null && card2.getEffect().equals(Position.EFFECT.IMITARIAL))  
27                 combat_resolution.append("Gana Carta 1. Defensor pierde 100 puntos.");  
28             else  
29                 combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");  
30         }  
31     default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);  
32 }
```

TEST10.AC

Hemos realizado esta implementación mínima que nos hace pasar el décimo Test, concluyendo que podríamos tener que refactorizar posteriormente.



REFACTORIZACIÓN POSTERIOR

```
40 private static @NotNull String effectToString(@NotNull Position.EFFECT effect){  
41     if (effect.toString().equals("NA"))  
42         return "N/A";  
43     else if(effect.toString().equals("IMMORTAL"))  
44         return "Inmortal";  
45     return effect.toString();  
46 }  
47 2 usages ✎gu4re +1 *  
48 private static @NotNull String getCardInfo(@NotNull Card card){  
49     // Local variables  
50     final String CARD1_NAME = "Carta 1";  
51     StringBuilder cardInfoBuilder = new StringBuilder();  
52     int cardNumber = card.getName().equals(CARD1_NAME) ? 1 : 2;  
53     cardInfoBuilder.append(String.format("Carta %d (%d/%d/Posición: %s", cardNumber, card.getAttack(),  
54         card.getDefense(), (card.getPosition().equals(Position.ATTACK) ? "Ataque" : "Defensa")));  
55     if (card.getEffect() != null)  
56         cardInfoBuilder.append(String.format("/Efecto: %s", effectToString(card.getEffect())));  
57     cardInfoBuilder.append(")");  
58     return cardInfoBuilder.toString();  
59  
60 }  
61  
62  
63 @DisplayName("Ninth Example")  
64 public void test9(){  
65     expected = "Carta 1 (1800/2850/Posición: Ataque/Efecto: Inmortal) vs Carta 2 " +  
66     "(2000/1500/Posición: Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 200 puntos.";  
67     card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,  
68         Position.ATTACK, Position.EFFECT.IMMORTAL);  
69     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,  
70         Position.ATTACK, Position.EFFECT.NA);  
71     Assertions.assertDoesNotThrow() -> {  
72         result = Combat.combat(card_1, card_2);  
73         Assertions.assertEquals(expected, result);  
74     };  
75 }  
76 no usages ✎gu4re *  
77 @Test  
78 @DisplayName("Tenth Example")  
79 public void test10(){  
80     expected = "Carta 1 (2100/2850/Posición: Ataque/Efecto: N/A) vs Carta 2 " +  
81     "(2000/1500/Posición: Ataque/Efecto: Inmortal) → Gana Carta 1. Defensor pierde 100 puntos.";  
82     card_1 = new Card( name: "Carta 1", attack: 2100, defense: 2850,  
83         Position.ATTACK, Position.EFFECT.NA);  
84     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,  
85         Position.ATTACK, Position.EFFECT.IMMORTAL);  
86     Assertions.assertDoesNotThrow() -> {  
87         result = Combat.combat(card_1, card_2);  
88         Assertions.assertEquals(expected, result);  
89     };  
90 }
```

Hemos realizado una refactorización posterior al código, ya que nos hemos percatado que hasta ahora en este noveno test y en el anterior del caso intermedio, mostrábamos los efectos de forma distinta al enunciado y tal cual venía el enumerado. Ahora aplicando una nueva función que hemos denominado “effectToString”, mostramos la salida adecuándonos al enunciado.

```
51 public void test10(){
52     expected = "Carta 1 (2100/2850/Posición: Ataque/Efecto: N/A) vs Carta 2 " +
53             "(2000/1500/Posición: Ataque/Efecto: Inmortal) → Gana Carta 1. Defensor pierde 100 puntos." TEST11CE;
54     card_1 = new Card( name: "Carta 1", attack: 2100, defense: 2850,
55                         Position.ATTACK, Position.EFFECT.NA);
56     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
57                         Position.ATTACK, Position.EFFECT.IMMORTAL);
58     Assertions.assertDoesNotThrow() → {
59         result = Combat.combat(card_1, card_2);
60         Assertions.assertEquals(expected, result);
61     });
62 }
63 no usages new *
64 @Test
65 @DisplayName("Eleventh Example")
66 public void test11(){
67     expected = "Carta 1 (1800/2850/Posición: Ataque/Efecto: N/A) vs Carta 2 " +
68             "(2000/1500/Posición: Defensa/Efecto: Inmortal) → Gana Carta 1.";
69     card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,
70                         Position.ATTACK, Position.EFFECT.NA);
71     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
72                         Position.DEFENSE, Position.EFFECT.IMMORTAL);
73     Assertions.assertDoesNotThrow() → {
74         result = Combat.combat(card_1, card_2);
75         Assertions.assertEquals(expected, result);
76     });
77 }
```

Como podemos ver en este undécimo test no tenemos problemas de compilación así que vamos a pasar a la siguiente fase, que es la ejecución del test.

TEST11.WA

```

57     Position.ATTACK, Position.EFFECT. IMMORTAL);
58     Assertions.assertDoesNotThrow(() -> {
59         result = Combat.combat(card_1, card_2);
60         Assertions.assertEquals(expected, result);
61     });
62 }
63 no usages new *
64 @Test
65 @DisplayName("Eleventh Example")
66 public void test11(){
67     expected = "Carta 1 (1800/2850/Posición: Ataque/Efecto: N/A) vs Carta 2 " +
68             "(2000/1500/Posición: Defensa/Efecto: Inmortal) -> Gana Carta 1.";
69     card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,
70                         Position.ATTACK, Position.EFFECT.NA);
71     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
72                         Position.DEFENSE, Position.EFFECT. IMMORTAL);
73     Assertions.assertDoesNotThrow(() -> {
74         result = Combat.combat(card_1, card_2);
75         Assertions.assertEquals(expected, result);
76     });
77 }
78

```

El test nos muestra que se esperaba un caso y se ha devuelto otro, ya que no estaba contemplado este nuevo caso, vamos a realizar una implementación mínima.

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (1800/2850/Posición: Ataque/Efecto: N/A) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: Inmortal) -> Gana Carta 1.> but was: <Carta 1 (1800/2850/Posición: Ataque/Efecto: N/A) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: Inmortal) -> Gana Carta 1. Carta 2 destruido/a>

Debug 'IntermediateCaseTest...' ⌂ More actions... ⌂

org.junit.jupiter.api.Assertions

@API(status = Status.STABLE, since = "5.2")
public static void assertDoesNotThrow(
 @NotNull org.junit.jupiter.api.function.Executable executable
)

Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

TEST11.AC

```

29     default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
30   }
31 }
32 1 usage  gu4re +1 *
33 private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
34   switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense())))
35   {
36     case -1 → combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");
37     case 0 → combat_resolution.append("Empate.");
38     case 1 → {
39       if (card2.getEffect() ≠ null && card2.getEffect().equals(Position.EFFECT.IMPETUOUS))
40         combat_resolution.append("Gana Carta 1.");
41       else
42         combat_resolution.append("Gana Carta 1. Carta 2 destruido/a.");
43     }
44     default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
45   }
46 }
47 1 usage  gu4re *
48 private static @NotNull String effectToString(@NotNull Position.EFFECT effect)
49 
```

Run: IntermediateCaseTest x

The screenshot shows an IDE interface with a code editor and a tool palette. In the code editor, there is a tooltip for the method 'effectToString'. The tooltip contains the following text:

Con esta implementación mínima podemos hacer que nuestro caso de prueba acierte, pudiendo realizar una pequeña refactorización con el "append" como vemos en la esquina inferior derecha.

- Intermediate Case examples ... (es.codeurjc.ais)
- ✓ Tenth Example
- ✓ Eleventh Example
- ✓ Eighth Example
- ✓ Ninth Example

Tests passed: 4 of 4 tests – 22 ms

```

case 1 → { // Refactorización posterior
  combat_resolution.append("Gana Carta 1.");
  if (card2.getEffect() = null)
    combat_resolution.append("Carta 2 destruido/a.");
}

```

TEST12.CE

```

70     Position.ATTACK, Position.EFFECT.NA),
71     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
72                         Position.DEFENSE, Position.EFFECT.IMMORTAL);
73     Assertions.assertDoesNotThrow(() -> {
74         result = Combat.combat(card_1, card_2);
75         Assertions.assertEquals(expected, result);
76     });
77     no usages new *
78 @Test
79 @DisplayName("Twelfth Example")
80 public void test12(){
81     expected = "Carta 1 (1000/2850/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 " +
82                 "(2000/1500/Posición: Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Carta 2 " +
83                 "destruido/a.";
84     card_1 = new Card( name: "Carta 1", attack: 1000, defense: 2850,
85                         Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
86     card_2 = new Card( name: "Carta 2", attack: 2000,
87                         Position.DEFENSE, Position.EFFECT.NA);|
88     Assertions.assertDoesNotThrow(() -> {
89         result = Combat.combat(card_1, card_2);
90         Assertions.assertEquals(expected, result);
91     });
92 }

```

En este caso, el duodécimo test nos presenta un problema de compilación ya que TOQUE MORTAL no está definido dentro del enumerado de efectos. Vamos a añadirlo al grupo y ejecutar el test.



```

public enum Position {
    19 usages
    ATTACK, DEFENSE;
    17 usages  ↗gu4re *
    enum EFFECT{
        7 usages
        IMMORTAL, NA, MORTAL_TOUCH
    }
}

```

TEST12.WA

```

71     Position.DEFENSE, Position.EFFECT. IMMORTAL);
72     Assertions.assertDoesNotThrow(() -> {
73         result = Combat.combat(card_1, card_2);
74         Assertions.assertEquals(expected, result);
75     });
76 }
77 no usages new *
78 @Test
79 @DisplayName("Twelfth Example")
80 public void test12(){
81     expected = "Carta 1 (1000/2850/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 " +
82                 "(2000/1500/Posición: Defensa/Efecto: N/A) -> Gana Carta 2. Atacante pierde 500 puntos. Carta 2 " +
83                 "destruido/a.";
84     card_1 = new Card( name: "Carta 1", attack: 1000, defense: 2850,
85                         Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
86     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
87                         Position.DEFENSE, Position.EFFECT.NA);
88     Assertions.assertDoesNotThrow(() -> {
89         result = Combat.combat(card_1, card_2);
90         Assertions.assertEquals(expected, result);
91     });
92 }

```

Tenemos un problema como podemos apreciar y es que no tenemos contemplado el caso con el nuevo efecto ya que previamente desconocíamos su existencia, por lo tanto vamos a implementarlo para que el test pueda pasar.

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (1000/2850/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: N/A) -> Gana Carta 2. Atacante pierde 500 puntos. Carta 2 destruido/a.> but was: <Carta 1 (1000/2850/Posición: Ataque/Efecto: MORTAL_TOUCH) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: N/A) -> Gana Carta 2. Atacante pierde 1000 puntos.>

Debug 'IntermediateCaseTest...' ⌂ More actions... ⌂

↳ org.junit.jupiter.api.Assertions

```

@API(status = Status.STABLE, since = "5.2")
public static void assertDoesNotThrow(
    @NotNull org.junit.jupiter.api.function.Executable executable
)

```

↳ Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

TEST12.AC

```

private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense())))
    {
        case -1 → {
            if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
                && card2.getDefense() < 2000)
                combat_resolution.append("Gana Carta 2. Atacante pierde 500 puntos. Carta 2 destruido/a.");
            else
                combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");
        }
        case 0 → combat_resolution.append("Empate.");
        case 1 → {
            combat_resolution.append("Gana Carta 1.");
            if (card2.getEffect() = null)
                combat_resolution.append("Carta Ademástremos añadido la conversión correspondiente");
            else
                combat_resolution.append("Carta Ademástremos añadido la conversión correspondiente");
        }
        default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
    }
}

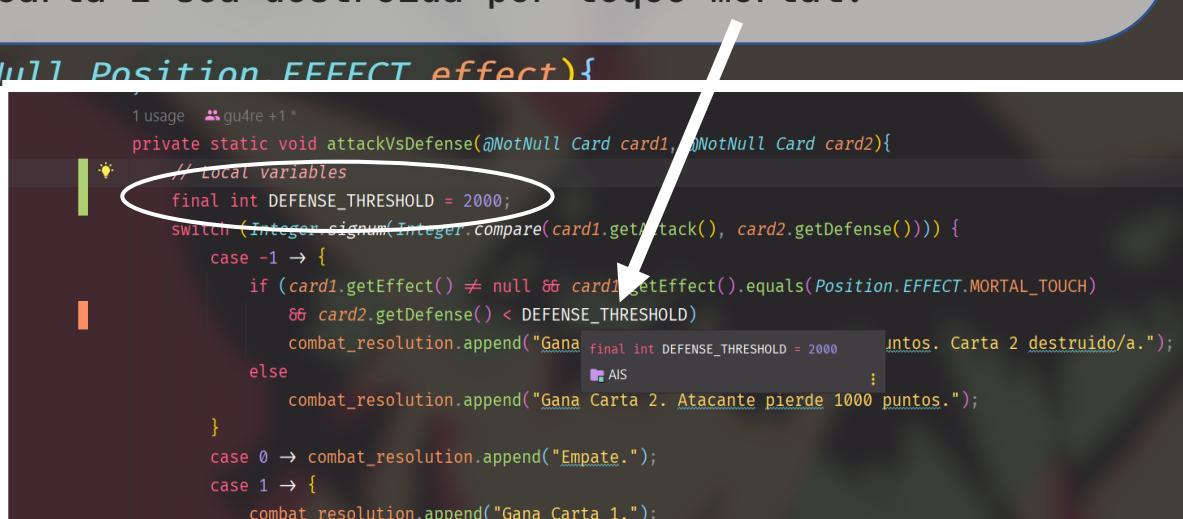
```

En este caso, está marcado en naranja los cambios realizados como mínima implementación para que nuestro test pase. Simplemente hemos añadido el caso en el que tengamos toque mortal en la carta 1 y por lo tanto la defensa de la carta 2 sea menor que 2000. Además hemos añadido la conversión correspondiente del efecto toque mortal a su valor en String. Podríamos plantear una refactorización con la eliminación de la variable mágica '2000' que indica el límite superior de puntos de defensa como para que la carta 2 sea destruida por toque mortal.

```

1 usage ↗gu4re *
private static @NotNull String effectToString(@NotNull Position EFFECT effect)
{
    if (effect.toString().equals("NA"))
        return "N/A";
    else if(effect.toString().equals("IMMORTAL"))
        return "Inmortal";
    else if (effect.toString().equals("MORTAL_TOUCH"))
        return "Toque mortal";
    return effect.toString();
}

```



```

1 usage ↗gu4re +1 *
private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
    // Local variables
    final int DEFENSE_THRESHOLD = 2000;
    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense())))
    {
        case -1 → {
            if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
                && card2.getDefense() < DEFENSE_THRESHOLD)
                combat_resolution.append("Gana final int DEFENSE_THRESHOLD = 2000 puntos. Carta 2 destruido/a.");
            else
                combat_resolution.append("Gana final int DEFENSE_THRESHOLD = 2000 puntos. Carta 2 destruido/a.");
        }
        case 0 → combat_resolution.append("Empate.");
        case 1 → {
            combat_resolution.append("Gana Carta 1.");
        }
    }
}

```

TEST13.CE

```
77
78
79
80
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
```

wTest
@DisplayName("Twelfth Example")
public void test12(){
 expected = "Carta 1 (1000/2850/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 " +
 "(2000/1500/Posición: Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Carta 2 " +
 "destruido/a.";
 card_1 = new Card(name: "Carta 1", attack: 1000, defense: 2850,
 Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
 card_2 = new Card(name: "Carta 2", attack: 2000, defense: 1500,
 Position.DEFENSE, Position.EFFECT.NA);
 Assertions.assertDoesNotThrow(() → {
 result = Combat.combat(card_1, card_2);
 Assertions.assertEquals(expected, result);
 });
}
no usages new *
@Test
@DisplayName("Thirteenth Example")
public void test13(){
 expected = "Carta 1 (2000/1000/Posición: Ataque/Efecto: N/A) vs Carta 2 " +
 "(2000/1500/Posición: Defensa/Efecto: Toque Mortal) → Gana Carta 1. Ambas cartas destruidas.";
 card_1 = new Card(name: "Carta 1", attack: 2000, defense: 1000,
 Position.ATTACK, Position.EFFECT.NA);
 card_2 = new Card(name: "Carta 2", attack: 2000, defense: 1500,
 Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
 Assertions.assertDoesNotThrow(() → {
 result = Combat.combat(card_1, card_2);
 Assertions.assertEquals(expected, result);
 });
}

Como podemos observar en este decimotercer Test no tenemos ningún problema de compilación por lo que vamos a pasar a la ejecución y prueba del mismo.

TEST13.WA

```

81      "(2000/1500/Posición: Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Carta 2 " +
82          "destruido/a." ;
83      card_1 = new Card( name: "Carta 1", attack: 1000, defense: 2850,
84                      Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
85      card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
86                      Position.DEFENSE, Position.EFFECT.NA);
87      Assertions.assertDoesNotThrow(() → {
88          result = Combat.combat(card_1, card_2);
89          Assertions.assertEquals(expected, result);
90      });
91  }
92  no usages new*
93  @Test
94  @DisplayName("Thirteenth Example")
95  public void test13(){
96      expected = "Carta 1 (2000/1000/Posición: Ataque/Efecto: N/A) vs Carta 2 " +
97                  "(2000/1500/Posición: Defensa/Efecto: Toque Mortal) → Gana Carta 1. Ambas cartas destruidas.";
98      card_1 = new Card( name: "Carta 1", attack: 2000, defense: 1000,
99                      Position.ATTACK, Position.EFFECT.NA);
100     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
101                     Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
102     Assertions.assertDoesNotThrow(() → {
103         result = Combat.combat(card_1, card_2);
104         Assertions.assertEquals(expected, result);
105     });
106  }

```

No tenemos el caso en cuestión implementado ya que nunca se ha dado la ocasión de que la defensa tenga Toque Mortal y por lo tanto ambas cartas sean destruidas. Vamos a implementar dicha funcionalidad.

```

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (2000/1000/Posición:
Debug 'IntermediateCaseTest..' Alt+Mayús+Intro More actions... Alt+Intro
:
```

```

  org.junit.jupiter.api.Assertions
    @API(status = Status.STABLE, since = "5.2")
    public static void assertDoesNotThrow(
        @NotNull org.junit.jupiter.api.function.Executable executable
    )
  Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)
:
```

TEST13.AC

```

31
32     1 usage  ↗gu4re +1 *
33     private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
34         // Local variables
35         final int DEFENSE_THRESHOLD = 2000;
36         switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense()))) {
37             case -1 → {
38                 if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
39                     && card2.getDefense() < DEFENSE_THRESHOLD)
40                     combat_resolution.append("Gana Carta 2. Atacante pierde 500 puntos. Carta 2 destruido/a.");
41                 else
42                     combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");
43             }
44             case 0 → combat_resolution.append("Empate.");
45             case 1 → {
46                 combat_resolution.append("Gana Carta 1.");
47                 if (card2.getEffect() = null)
48                     combat_resolution.append(" Carta 2 destruido/a.");
49                 else if (card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH) && card1.getDefense() < DEFENSE_THRESHOLD)
50                     combat_resolution.append(" Ambas cartas destruidas.");
51             }
52             default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
53         }
54     1 usage  ↗gu4re
55     private static @NotNull String effectToString(@NotNull Position.EFFECT effect)
56         if (effect.toString().equals("NA"))
57             return "N/A";
58         else if(effect.toString().equals("IMMORTAL"))
59             return "Immortal";
60         else if (effect.toString().equals("MORTAL_TOUCH"))
61             return "Toque mortal";
62         return effect.toString();

```

Con esta mínima implementación hacemos pasar nuestro test, dando la posibilidad de que si la carta 1 gana, en vez de destruirse solo la carta 2, exista la posibilidad de que la carta 1 también desaparezca por culpa del efecto de Toque Mortal de la carta 2. No hay una refactorización obvia del método por lo que pasamos al siguiente test.

TEST14.CE

```
public void test14() {
    expected = "Carta 1 (2000/1000/Posición: Ataque/Efecto: N/A) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: Toque mortal) " +
               "→ Gana Carta 1. Ambas cartas destruidas.";
    card_1 = new Card( name: "Carta 1", attack: 2000, defense: 1000,
                      Position.ATTACK, Position.EFFECT.NA);
    card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
                      Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
    Assertions.assertDoesNotThrow(() -> {
        result = Combat.combat(card_1, card_2);
        Assertions.assertEquals(expected, result);
    });
}
no usages  new *
@Test
@DisplayName("Fourteenth Example")
public void test14(){
    expected = "Carta 1 (1000/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (0/1000/Posición: " +
               "Defensa/Efecto: Toque mortal) → Empate. Ambas cartas destruidas.";
    card_1 = new Card( name: "Carta 1", attack: 1000, defense: 1000,
                      Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
    card_2 = new Card( name: "Carta 2", attack: 0, defense: 1000,
                      Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
    Assertions.assertDoesNotThrow(() -> {
        result = Combat.combat(card_1, card_2);
        Assertions.assertEquals(expected, result);
    });
}
```

Podemos apreciar que a la hora de realizar el decimocuarto Test no tenemos ningún problema de compilación, vamos a ver la ejecución...

TEST14.WA

```

94
95     public void test13(){
96         expected = "Carta 1 (2000/1000/Posición: Ataque/Efecto: N/A) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: Toque mortal) " +
97             "→ Gana Carta 1. Ambas cartas destruidas.";
98         card_1 = new Card( name: "Carta 1", attack: 2000, defense: 1000,
99             Position.ATTACK, Position.EFFECT.NA);
100        card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
101            Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
102        Assertions.assertDoesNotThrow(() → {
103            result = Combat.combat(card_1, card_2);
104            Assertions.assertEquals(expected, result);
105        });
106    }
107
108    @Test
109    @DisplayName("Fourteenth Example")
110    public void test14(){
111        expected = "Carta 1 (1000/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (0/1000/Posición: " +
112            "Defensa/Efecto: Toque mortal) → Empate. Ambas cartas destruidas.";
113        card_1 = new Card( name: "Carta 1", attack: 1000, defense: 1000,
114            Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
115        card_2 = new Card( name: "Carta 2", attack: 0, defense: 1000,
116            Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
117        Assertions.assertDoesNotThrow(() → {
118            result = Combat.combat(card_1, card_2);
119            Assertions.assertEquals(expected, result);
120        });
121    }

```

Nos encontramos ante un caso totalmente nuevo y desconocido para nuestro programa, y por lo tanto, nuestra aplicación no funciona como lo esperado. Vamos a implementar este caso mínimamente para que el Test pase.

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (1000/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (0/1000/Posición: Defensa/Efecto: Toque mortal) → Empate. Ambas cartas destruidas.>
 Debug 'IntermediateCaseTest.' Alt+Mayús+Intro More actions... Alt+Intro

org.junit.jupiter.api.Assertions
 @API(status = Status.STABLE, since = "5.2")
 public static void assertDoesNotThrow(
 @NotNull org.junit.jupiter.api.function.Executable executable
)
 Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

TEST14.AC

```

28
29
30
31
32     default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
33 }
34
35     }
36
37     1 usage  ↗gu4re +1 *
38
39 private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
40     // Local variables
41     final int DEFENSE_THRESHOLD = 2000;
42     switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense())))
43     {
44         case -1 → {
45             if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
46                 && card2.getDefense() < DEFENSE_THRESHOLD)
47                 combat_resolution.append("Gana Carta 2. Atacante pierde 500 puntos. Carta 2 destruido/a.");
48             else
49                 combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");
50         }
51
52         case 0 → combat_resolution.append("Empate.")
53             .append((card1.getEffect() ≠ null && card2.getEffect() ≠ null &&
54                     card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
55                     && card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH))
56                     ? " Ambas cartas destruidas." : "");
57
58         case 1 → {
59             combat_resolution.append("Gana Carta 1.");
60             if (card2.getEffect() = null)
61                 combat_resolution.append(" Carta 2 destruido/a.");
62             else if (card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
63                     && card1.getDefense() < DEFENSE_THRESHOLD)
64                 combat_resolution.append(" Ambas cartas destruidas.");
65         }
66
67         default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
68     }
69
70     }
71
72     1 usage  ↗gu4re
73
74 private static @NotNull String effectToString(@NotNull Position.EFFECT effect){
```

Con esta mínima implementación nos pasa el test, echando mano del operador ternario incluimos la opción de que las cartas puedan llegar a ser destruidas a la vez si ambos tienen Toque Mortal, y en caso contrario, hacemos 'append' de un String vacío. Procedemos al siguiente Test.

TEST15.CE

```

108
109     public void test14(){
110         expected = "Carta 1 (1000/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (0/1000/Posición: " +
111             "Defensa/Efecto: Toque mortal) → Empate. Ambas cartas destruidas.";
112         card_1 = new Card( name: "Carta 1", attack: 1000, defense: 1000,
113             Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
114         card_2 = new Card( name: "Carta 2", attack: 0, defense: 1000,
115             Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
116         Assertions.assertDoesNotThrow(() -> {
117             result = Combat.combat(card_1, card_2);
118             Assertions.assertEquals(expected, result);
119         });
120     }
121     no usages    new *
122
123     @Test
124     @DisplayName("Fifteenth Example")
125     public void test15(){
126         expected = "Carta 1 (1500/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (2000/1500/Posición: " +
127             "Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.";
128         card_1 = new Card( name: "Carta 1", attack: 1500, defense: 1000,
129             Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
130         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
131             Position.ATTACK, Position.EFFECT.NA);
132         Assertions.assertDoesNotThrow(() -> {
133             result = Combat.combat(card_1, card_2);
134             Assertions.assertEquals(expected, result);
135         });
136     }

```

No tenemos ningún “Compilation Error” presente, por lo que vamos a ver en ejecución qué ocurre...

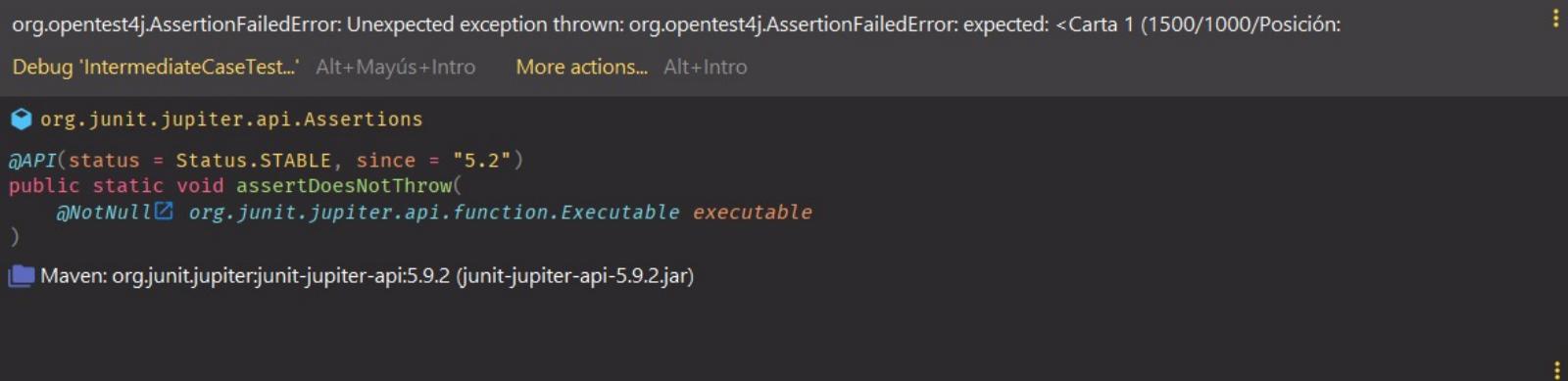
TEST15.WA

```

09
10     expected = "Carta 1 (1000/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (0/1000/Posición: " +
11         "Defensa/Efecto: Toque mortal) → Empate. Ambas cartas destruidas.";
12     card_1 = new Card( name: "Carta 1", attack: 1000, defense: 1000,
13         Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
14     card_2 = new Card( name: "Carta 2", attack: 0, defense: 1000,
15         Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
16     Assertions.assertDoesNotThrow(() → {
17         result = Combat.combat(card_1, card_2);
18         Assertions.assertEquals(expected, result);
19     });
20 no usages new *
21 @Test
22 @DisplayName("Fifteenth Example")
23 public void test15(){
24     expected = "Carta 1 (1500/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (2000/1500/Posición: " +
25         "Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.";
26     card_1 = new Card( name: "Carta 1", attack: 1500, defense: 1000,
27         Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
28     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
29         Position.ATTACK, Position.EFFECT.NA);
30     Assertions.assertDoesNotThrow(() → {
31         result = Combat.comb
32         Assertions.assertEquals(expected, result);
33     });
34 }

```

Como podemos ver, tenemos un caso nuevo no contemplado, y es que a pesar de ganar la carta 2 por estar en posición de ataque y tener mayor ataque, se destruye junto con la carta 1 al tener ésta Toque Mortal. Realizaremos la implementación mínima para hacer pasar el Test.



org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (1500/1000/Posición: Ataque/Efecto: N/A) vs Carta 2 (2000/1500/Posición: Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.> but actual: <Carta 1 (1500/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (2000/1500/Posición: Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.>

Debug 'IntermediateCaseTest...' Alt+Mayús+Intro More actions... Alt+Intro

org.junit.jupiter.api.Assertions

@API(status = Status.STABLE, since = "5.2")

public static void assertDoesNotThrow(
 @NotNull org.junit.jupiter.api.function.Executable executable
)

Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

TEST15.AC

16 usages

Esta mínima implementación nos hace pasar el Test, sin embargo, es obvio la posible refactorización de extrayendo la condición "card1.getEffect() != null" afuera para solo realizarse una única vez. Además de esta forma evitar también gracias al operador ternario, un aviso por "Cognitive Complexity".

```

public class Combat {
    ...
    16 usages
6     Esta mínima implementación nos hace pasar el Test, sin embargo, es obvio la posible refactorización de extrayendo la condición "card1.getEffect() != null" afuera para solo realizarse una única vez. Además de esta forma evitar también gracias al operador ternario, un aviso por "Cognitive Complexity".
7         ...
8             ...
9             ...
10            ...
11             ...
12             ...
13             ...
14             ...
15             ...
16             ...
17             ...
18             ...
19             ...
20             ...
21             ...
22             ...
23             ...
24             ...
25             ...
26             ...
27             ...
28             ...
29             ...
30             ...
31             ...
32             ...
33             ...
}

```

switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack()))){

- case -1 → {
- if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT. IMMORTAL))
- combat_resolution.append("Gana Carta 2. Atacante pierde 200 puntos.");
- else if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT. MORTAL_TOUCH))
- combat_resolution.append("Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.");
- else
- combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");

}

case 0 → {

- if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT. IMMORTAL))
- combat_resolution.append("Empate. Carta 2 destruido/a.");
- else
- combat_resolution.append("Empate. Ambas cartas destruidas.");

}

switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack()))){

- case -1 → {
- if (card1.getEffect() ≠ null)
- combat_resolution.append((card1.getEffect().equals(Position.EFFECT. IMMORTAL) ? "Gana Carta 2. Atacante pierde 200 puntos." : "Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas."));
- else
- combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");

}

TEST16.CE

```

20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

    @Test
    @DisplayName("Fifteenth Example")
    public void test15(){
        expected = "Carta 1 (1500/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (2000/1500/Posición: " +
            "Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.";
        card_1 = new Card( name: "Carta 1", attack: 1500, defense: 1000,
            Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
        card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
            Position.ATTACK, Position.EFFECT.NA);
        Assertions.assertDoesNotThrow(() → {
            result = Combat.combat(card_1, card_2);
            Assertions.assertEquals(expected, result);
        });
    }
    no usages new *
    @Test
    @DisplayName("Sixteenth Example")
    public void test16(){
        expected = "Carta 1 (1000/0/Posición: Ataque/Efecto: Presión) vs Carta 2 (0/3000/Posición: " +
            "Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 1000 puntos.";
        card_1 = new Card( name: "Carta 1", attack: 1000, defense: 0,
            Position.ATTACK, Position.EFFECT.PRESSURE);
        card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000,
            Position.DEFENSE, Position.EFFECT.NA);
        Assertions.assertDoesNotThrow(() → {
            result = Combat.combat(card_1, card_2);
            Assertions.assertEquals(expected, result);
        });
    }
}

```

Este Test produce un problema en tiempo de compilación, ya que aparece un nuevo efecto denominado presión, que tenemos que implementar de la forma que se ve en la esquina inferior derecha, para posteriormente ejecutar el test y ver qué ocurre.



```

public enum Position {
    ATTACK, DEFENSE;
    enum EFFECT{
        IMMORTAL, NA, MORTAL_TOUCH, PRESSURE
    }
}

```

```

22
23     public void test15(){
24         expected = "Carta 1 (1500/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (2000/1500/Posición: " +
25             "Ataque/Efecto: N/A) → Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.";
26         card_1 = new Card( name: "Carta 1", attack: 1500, defense: 1000,
27             Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
28         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
29             Position.ATTACK, Position.EFFECT.NA);
30         Assertions.assertDoesNotThrow(() → {
31             result = Combat.combat(card_1, card_2);
32             Assertions.assertEquals(expected, result);
33         });
34     }
35     no usages new *
36     @Test
37     @DisplayName("Sixteenth Example")
38     public void test16(){
39         expected = "Carta 1 (1000/0/Posición: Ataque/Efecto: Presión) vs Carta 2 (0/3000/Posición: " +
40             "Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 1000 puntos.";
41         card_1 = new Card( name: "Carta 1", attack: 1000, defense: 0,
42             Position.ATTACK, Position.EFFECT.PRESSURE);
43         card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000,
44             Position.DEFENSE, Position.EFFECT.NA);
45         Assertions.assertDoesNotThrow(() → {
46             result = Combat.combat(card_1, card_2);
47             Assertions.assertEquals(expected, result);
48         });
49     }

```

Como podemos ver el caso nuevo que se nos presenta no está implementado claro está, ya que nos ha aparecido un nuevo efecto que hace que el resultado varíe. Vamos a implementar lo mínimo para que el Test pase y luego vemos qué posibles refactorizaciones podemos hacer.

```

org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (1000/0/Posición: Ataque/Efecto: Presión) vs Carta 2 (0/3000/Posición: Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 1000 puntos.>
Debug 'IntermediateCaseTest...' Alt+Mayús+Intro More actions... Alt+Intro
  org.junit.jupiter.api.Assertions
    @API(status = Status.STABLE, since = "5.2")
    public static void assertDoesNotThrow(
        @NotNull org.junit.jupiter.api.function.Executable executable
    )
  Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)

```

TEST16.AC

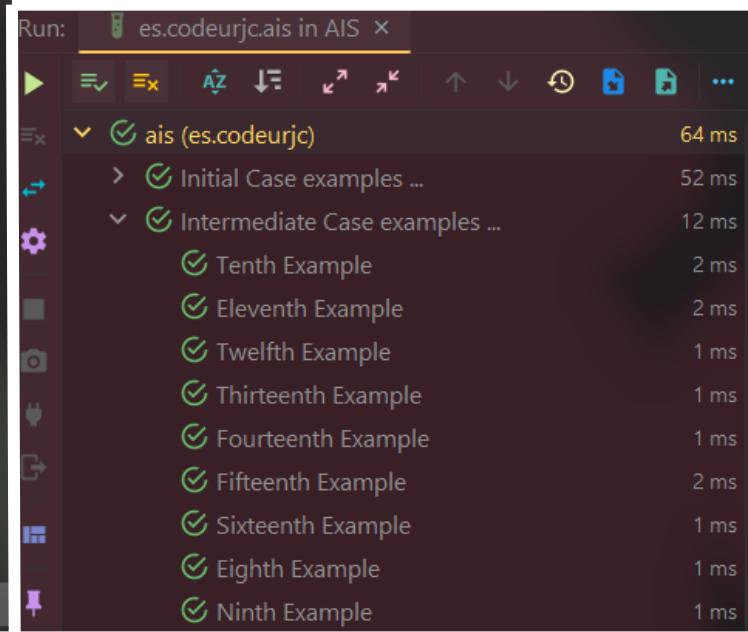
Nos ha pasado algo sorprendente en este test, y es que el test ha dado acierto con tan solo traducir "PRESSURE -> Presión" por el formato del efecto a String, y no hemos tenido que implementar ningún tipo de funcionalidad para el efecto de presión, ya que como venimos de Test anteriores, coincide que cuando se introduce en el if-else que se encuentra en "attackVsDefense" cuando la "Carta 1 < Carta 2", nuestra estructura else, ya contiene justo el valor que necesitamos devolver, por lo tanto en este punto, nos estamos planteando si continuar, ya que bueno al fin y al cabo, TDD nos dice que después de dar acierto el Test, refactoricemos y en este caso como el código no ha sido modificado, optamos por continuar y luego al final hacer una refactorización general no solo del código sino de la propia idea que tenemos sobre el enunciado por si habría alguna idea que estuviéramos malinterpretando.

```
private static @NotNull String effectToString(@NotNull Position.EFFECT effect){  
    if (effect.toString().equals("NA"))  
        return "N/A";  
    else if(effect.toString().equals("IMMORTAL"))  
        return "Inmortal";  
    else if (effect.toString().equals("MORTAL_TOUCH"))  
        return "Toque mortal";  
    else if (effect.toString().equals("PRESSURE"))  
        return "Presión";  
    return effect.toString();  
}
```

2 usages +1

```
private static @NotNull String getCardInfo(@NotNull Card card){  
    // Local variables  
    final String CARD1_NAME = "Carta 1";  
    StringBuilder cardInfoBuilder = new StringBuilder();  
    int cardNumber = card.getName().equals(CARD1_NAME) ? 1 : 2;  
    cardInfoBuilder.append((String.format("%s %s", cardNumber, card.getName())));  
    cardInfoBuilder.append(" Atacante pierde 500 puntos. ");  
    cardInfoBuilder.append("Carta 2 destruido/a.");  
    return cardInfoBuilder.toString();  
}
```

```
private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){  
    // Local variables  
    final int DEFENSE_THRESHOLD = 2000;  
    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense()))){  
        case -1 → {  
            if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)  
                && card2.getDefense() < DEFENSE_THRESHOLD)  
                combat_resolution.append("Gana Carta 2. Atacante pierde 500 puntos. Carta 2 destruido/a.");  
            else // Por aquí se cuela el test 16 de Presión  
                combat_resolution.append("Gana Carta 2. Atacante pierde 1000 puntos.");  
        }  
    }  
}
```



TEST17.CE

```

136
137 @DisplayName("Sixteenth Example")
138
139 public void test16(){
140     expected = "Carta 1 (1000/0/Posición: Ataque/Efecto: Presión) vs Carta 2 (0/3000/Posición: " +
141         "Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 1000 puntos.";
142     card_1 = new Card( name: "Carta 1", attack: 1000, defense: 0,
143                         Position.ATTACK, Position.EFFECT.PRESSURE);
144     card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000,
145                         Position.DEFENSE, Position.EFFECT.NA);
146     Assertions.assertDoesNotThrow(() → {
147         result = Combat.combat(card_1, card_2);
148         Assertions.assertEquals(expected, result);
149     });
150 }
151 no usages  ↗ gu4re *
152
153 @Test
154 @DisplayName("Seventeenth Example")
155
156 public void test17(){
157     expected = "Carta 1 (2000/0/Posición: Ataque/Efecto: N/A) vs Carta 2 (1500/0/Posición: " +
158         "Ataque/Efecto: Presión) → Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.";
159     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0,
160                         Position.ATTACK, Position.EFFECT.NA);
161     card_2 = new Card( name: "Carta 2", attack: 1500, defense: 0,
162                         Position.ATTACK, Position.EFFECT.PRESSURE);
163     Assertions.assertDoesNotThrow(() → {
164         result = Combat.combat(card_1, card_2);
165         Assertions.assertEquals(expected, result);
166     });
167 }
168 }
```

Como podemos observar no tenemos ningún problema en tiempo de compilación por lo tanto vamos a ver en tiempo de ejecución que sucede, si volverá a pasar directamente como nos ocurría en el anterior Test o no.

TEST17.WA

```
37     expected = "Carta 1 (1000/0/Posición: Ataque/Efecto: Presión) vs Carta 2 (0/3000/Posición: " +
38             "Defensa/Efecto: N/A) → Gana Carta 2. Atacante pierde 1000 puntos.";
39     card_1 = new Card( name: "Carta 1", attack: 1000, defense: 0,
40                       Position.ATTACK, Position.EFFECT.PRESSURE);
41     card_2 = new Card( name: "Carta 2", attack: 0, defense: 3000,
42                       Position.DEFENSE, Position.EFFECT.NA);
43     Assertions.assertDoesNotThrow(() → {
44         result = Combat.combat(card_1, card_2);
45         Assertions.assertEquals(expected, result);
46     });
47 }
48 no usages ↗ gu4re *
49 @Test
50 @DisplayName("Seventeenth Example")
51 public void test17(){
52     expected = "Carta 1 (2000/0/Posición: Ataque/Efecto: Presión) → Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.";
53     "Ataque/Efecto: Presión) → Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.";
54     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0,
55                       Position.ATTACK, Position.EFFECT.NA);
56     card_2 = new Card( name: "Carta 2", attack: 1500, defense: 0,
57                       Position.ATTACK, Position.EFFECT.PRESSURE);
58     Assertions.assertDoesNotThrow(() → {
59         result = Combat.combat(card_1, card_2);
60         Assertions.assertEquals(expected, result);
61     });
62 }
63 }
```

En este caso, este Test si que falla su funcionalidad así que habrá que implementarlo, analizando nuevamente qué hace realmente el efecto de Presión, cómo se puede aplicar en un combate y de esa forma, llegar a una conclusión próxima de por qué el anterior Test (si 0/0 pierden no.)

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown: org.opentest4j.AssertionFailedError: expected: <Carta 1 (2000/0/Posición: Ataque/Efecto: Presión) → Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.>
Debug 'IntermediateCaseTest...' Alt+Mayús+Intro More actions... Alt+Intro
  ↗ org.junit.jupiter.api.Assertions
    @API(status = Status.STABLE, since = "5.2")
    public static void assertDoesNotThrow(
        @NotNull org.junit.jupiter.api.function.Executable executable
    )
  Maven: org.junit.jupiter:junit-jupiter-api:5.9.2 (junit-jupiter-api-5.9.2.jar)
```

TEST17.AC

```

7     private static final String NOT_SUPPORTED_YET = "Not supported yet";
8     no usages  ↗ gu4re
9     private Combat(){}
10    1 usage   ↗ gu4re +1 *
11    private static void attackVsAttack(@NotNull Card card1, @NotNull Card card2){
12      switch (Integer.signum(Inte Refactor this method to reduce its Cognitive Complexity from 17 to the 15 allowed. ⋮
13      case -1 → {
14        if (card1.getEffect() combat_resolution.append("Gana Carta 1. Defensor pierde 100 puntos."));
15        else if (card1.getEffect() != null & card1.getEffect().equals(Position.EFFECT.STRONG)) {
16          combat_resolution.append("Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.");
17        else if (card1.getEffect() != null & card1.getEffect().equals(Position.EFFECT.PRESSURE))
18          combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");
19        else
20          combat_resolution.append("Empate. Ambas cartas se destruyen.");
21      }
22      else
23        combat_resolution.append("Gana Carta 1. Defensor pierde 300 puntos. Carta 2 destruido/a.");
24    }
25    case 1 → {
26      if (card2.getEffect() ≠ null & card2.getEffect().equals(Position.EFFECT.STRONG))
27        combat_resolution.append("Gana Carta 1. Defensor pierde 100 puntos.");
28      else if (card2.getEffect() ≠ null & card2.getEffect().equals(Position.EFFECT.PRESSURE))
29        combat_resolution.append("Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.");
30      else
31        combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");
32    }
33    default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
34  }
35  }
36  1 usage  ↗ gu4re +1
private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
```

Esta implementación mínima nos hace pasar el último test del caso intermedio, sin embargo, hemos llegado a conclusiones notorias: Tenemos de refactorizar el método attackVsAttack para reducir nuestro warning ya muy conocido, "Cognitive Complexity", para mejorar la lectura y legibilidad del código. Además, nos hemos percatado de algo importante a la hora de tratar con el efecto de Presión.

SonarLint: Show issue locations Alt+Mayús+Intro More actions... Alt+Intro

Hasta ahora no habíamos implementado ni habíamos necesitado utilizar los puntos de cambio de vida que se devuelven como resultado del combate al perdedor, pero, como justo este nuevo efecto hace uso de ello, necesitamos implementarlo, lo que nos va a llevar a una última refactorización global de prácticamente toda la clase para poder incluir este nuevo atributo. Nos ayudará seguramente a optimizar los "append" que realizamos, por lo que mejorará la lectura del código.

REFACTORIZACIÓN POSTERIOR

```
13     private static void attackVsAttack(@NotNull Card card1, @NotNull Card card2){  
14         lostPoints = (card2.getEffect() != null && card2.getEffect().equals(Position.EFFECT.PRESSURE))  
15             ? Math.abs(card1.getAttack() - card2.getAttack()) / 2  
16             : Math.abs(card1.getAttack() - card2.getAttack());  
17         switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack()))){  
18             case -1 → {  
19                 combat_resolution.append(String.format("Gana Carta 2. Atacante pierde %d puntos.", lostPoints));  
20                 if (card1.getEffect() != null)  
21                     combat_resolution.append((card1.getEffect().equals(Position.EFFECT.ETERNAL)  
22                         ? EMPTY  
23                         : BOTH_CARDS_DESTROYED));  
24             else  
25                 combat_resolution.append(" Carta 1 destruido/a.");  
26         }  
27         case 0 → combat_resolution.append(String.format("Empate.%s",  
28             (card1.getEffect() != null && card1.getEffect().equals(Position.EFFECT.ETERNAL)  
29                 ? CARD2_DESTROYED  
30                 : BOTH_CARDS_DESTROYED)));  
31         case 1 → combat_resolution.append(String.format("Gana Carta 1. Defensor pierde %d puntos.%s", lostPoints,  
32             (card2.getEffect() != null && card2.getEffect().equals(Position.EFFECT.ETERNAL)  
33                 ? EMPTY  
34                 : CARD2_DESTROYED)));  
35         default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);  
36     }
```

La primera refactorización que hemos realizado es el método en el que justamente intervenía el efecto de presión, que es el método `attackVsAttack`. Hemos eliminado el problema de "Cognitive Complexity" así como la declaración de constantes de algunos String que se repetían e incluso es probable que cuando refactoricemos el resto de métodos, nos encontraremos que los String que actualmente no están encapsulados en constantes, acaben estandarizados. Sigamos con el método `attackVsDefense`.

REFACTORIZACIÓN POSTERIOR 2

```

35     default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
36 }
37 }
38 1 usage  ↗ gu4re +1 *
39 private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
40     // Local variables
41     final int DEFENSE_THRESHOLD = 2000;
42     lostPoints = (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.PRESSURE))
43         ? Math.abs(card1.getAttack() - card2.getDefense()) / 2
44         : Math.abs(card1.getAttack() - card2.getDefense());
45     switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense()))) {
46         case -1 → combat_resolution.append(String.format("Gana Carta 2. Atacante pierde %d puntos.%s", lostPoints,
47             (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
48                 && card2.getDefense() < DEFENSE_THRESHOLD)
49                 ? CARD2_DESTROYED
50                 : EMPTY));
51         case 0 → combat_resolution.append("Empate.");
52             && card2.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
53             && card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
54             ? BOTH_CARDS_DESTROYED
55             : EMPTY);
56         case 1 → {
57             combat_resolution.append("Gana Carta 1.");
58             if (card2.getEffect() = null)
59                 combat_resolution.append(CARD2_DESTROYED);
60             else if (card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
61                 && card1.getDefense() < DEFENSE_THRESHOLD)
62                 combat_resolution.append(BOTH_CARDS_DESTROYED);
63             }
64         default → throw new UnsupportedOperationException(NOT_S
65     }
66 }
67 1 usage  ↗ gu4re

```

Esta segunda refactorización corresponde con el método `attackVsDefense`, que básicamente hemos compactado y refactorizado sobre todo los String así como la reutilización de constantes. Además, hemos refactorizado el método `effectToString`, antes implementado con un bloque extenso de if-else, por una estructura `switch` que permite una mejor lectura del código.

```

private static @NotNull String effectToString(@NotNull Position.EFFECT effect){
    return switch (effect) {
        case NA → "N/A";
        case IMMORTAL → "Inmortal";
        case MORTAL_TOUCH → "Toque mortal";
        case PRESSURE → "Presión";
    };
}

```

REFACTORIZACIÓN FINAL

2 usages  gu4re +1 *

```
private static @NotNull String getCardInfo(@NotNull Card card) {
    // Local variables
    StringBuilder cardInfoBuilder = new StringBuilder();
    int cardNumber = card.getName().equals(CARD1_NAME) ? 1 : 2;
    cardInfoBuilder.append(String.format("Carta %d (%d/%d/Posición: %s) %s", cardNumber, card.getAttack(), card.getDefense(), card.getPosition(), card.getEffect()));
    if (card.getEffect() != null)
        cardInfoBuilder.append(String.format("/Efecto: %s", effectToString(card.getEffect())));
    cardInfoBuilder.append(")");
    return cardInfoBuilder.toString();
}
```

17 usages  gu4re +1 *

```
public static @NotNull String combat(@NotNull Card card1, @NotNull Card card2) throws IllegalPositionException {
    // Local variables
    final String ERROR_MESSAGE = "La carta atacante no puede estar en una posición de Defensa";
    final int ZERO = 0;
    // Clearing StringBuilder at the beginning of the method
    combat_resolution.setLength(ZERO);
    if (card1.getPosition().equals(Position.DEFENSE))
        throw new IllegalPositionException(ERROR_MESSAGE);
    combat_resolution.append(getCardInfo(card1)).append(" vs ").append(getCardInfo(card2));
    switch (card2.getPosition()) {
        case ATTACK → attackVsAttack(card1, card2);
        case DEFENSE → attackVsDefense(card1, card2);
        default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
    }
    return combat_resolution.toString();
}
```

En el caso de los métodos restantes, no se ha aplicado ninguna refactorización, ya que no ha sido necesaria. Como podemos observar, todos los Test pasan de forma adecuada, así que con esto concluiría el caso intermedio y por ende, la práctica por contemplarse el caso final como optativo.

▶	≡x	AZ	↔
≡x	ais (es.codeurjc)		58 ms
▼	Initial Case examples ...		46 ms
	First Example		37 ms
	Second Example		2 ms
	Third Example		1 ms
	Fourth Example		2 ms
	Fifth Example		1 ms
	Sixth Example		1 ms
>	Seventh Example		2 ms
▼	Intermediate Case examples ...		12 ms
	Tenth Example		1 ms
	Eleventh Example		1 ms
	Twelfth Example		1 ms
	Thirteenth Example		1 ms
	Fourteenth Example		1 ms
	Fifteenth Example		1 ms
	Sixteenth Example		1 ms
	Seventeenth Example		2 ms
	Eighth Example		1 ms
	Ninth Example		2 ms