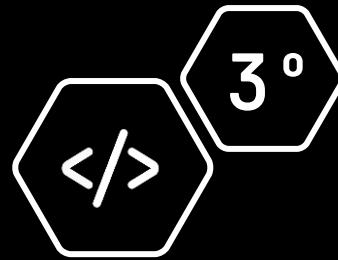




**PRÁCTICA 2. TEST DRIVEN  
DEVELOPMENT**  
**AMPLIACIÓN DE INGENIERÍA DEL  
SOFTWARE**  
**TERCER CURSO**  
**DIEGO PICAZO GARCÍA**  
**LARA FERNÁNDEZ GUTIÉRRREZ**

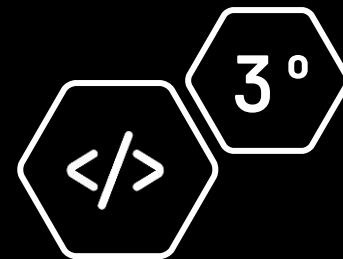




# PRÁCTICA 2. TEST DRIVEN DEVELOPMENT

## AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE

### CASO INICIAL



#### INDICE

- [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

```

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("Carta 1", 3000, 2500, Position.Attack);
19        Card card_2 = new Card("Carta 2", 2500, 2100, Position.Attack);
20        String result = Combat.combat(card_1, card_2);
21        Assertions.assertEquals(expected, result);
22    }
23 }
```

```

1 6 usages new *
2 public class Card {
3     2 usages new *
4     public Card(String name, int attack, int defense, Position position) {
5         |
6         |
7             1 usage ↗Laara2705 +2 *
8             public class Combat {
9                 1 usage ↗Laara2705 +1*
10                @ public static String combat(Card c1, Card c2) {
11                    return null;
12                }
13            }
14        }
15    }
16 }
```

```

3 usages ↗Laara2705 *
4 public enum Position {
5     2 usages
6     Attack, Defense;
7 }
```

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 {
4     @ 1 usage  ↗ Laara2705 +1
5       public static String combat(Card c1, Card c2) {
6         return "Carta 1 (3000/2500/Posición: Ataque) vs Carta 2\n" +
7             "(2500/2100/Posición: Ataque) → Gana Carta 1. Defensor pierde 500\n" +
8             "puntos. Carta 2 destruido/a.";
9
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( name: "Carta 1", attack: 3000, defense: 2500, Position.Attack);
18      Card card_2 = new Card( name: "Carta 2", attack: 2500, defense: 2100, Position.Attack);
19      String result = Combat.combat(card_1, card_2);
20      Assertions.assertEquals(expected, result);
21    }
22 }
```

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

11ms 11ms /Library/Java/JavaVirtualMachines/jdk-17.jdk/Com

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.

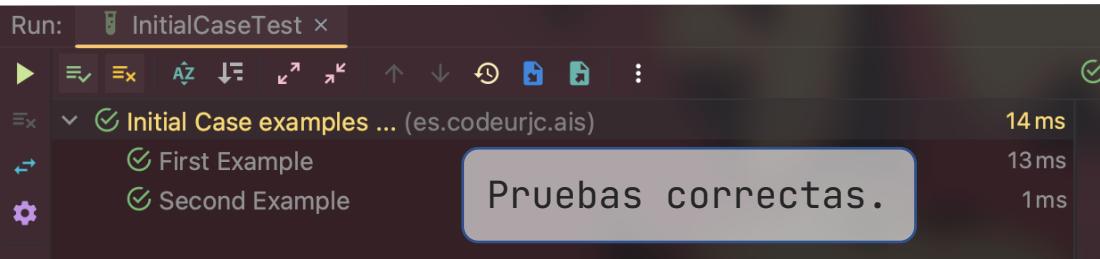


## TEST2.AC

```

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         }
```

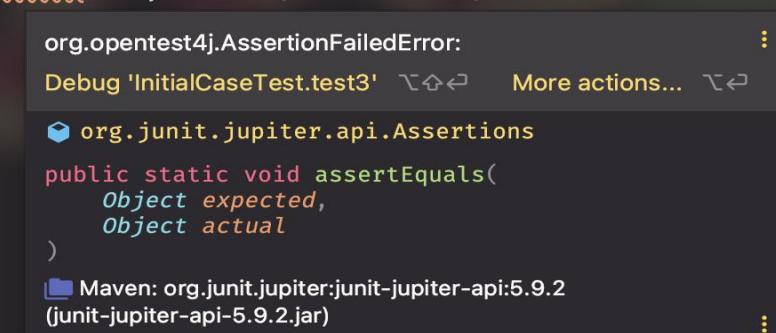
```
44
45 @Test
46 @DisplayName("Third Example")
47 public void test3(){
48     expected = """
49         Carta 1 (2000/0/Posición: Ataque) vs Carta 2
50         (2000/1500/Posición: Ataque) → Empate. Ambas cartas destruidas."";
51
52     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0,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



A1 A4 A26 ↑ ↓

```
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 }
```

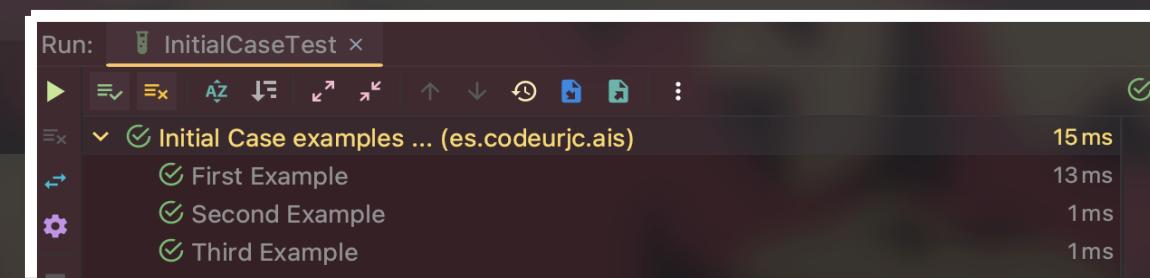
Tenemos un “`AssertionFailedError`” que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un caso válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método “`combat`” para que pase el test.



# 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
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 }
```

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 }

```



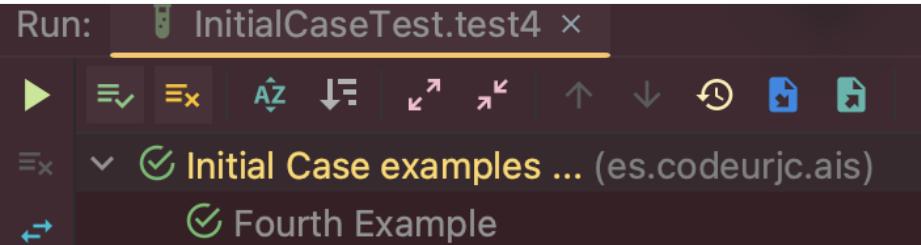
Tenemos un “AssertFailedError” que nos comenta que lo devuelto no coincide con lo esperado, ya que no hemos desarrollado un caso válido en las clases. Por lo tanto, nos toca ponernos manos a la obra en el método “combat” así como la implementación de la defensa para que pase el test.

```

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.



# TEST4.AC

```

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();
}

```

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)

Fifth Example

TEST5.AC 11ms

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 }
```

## TEST6.CE

```
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 }
```

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.

```
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)

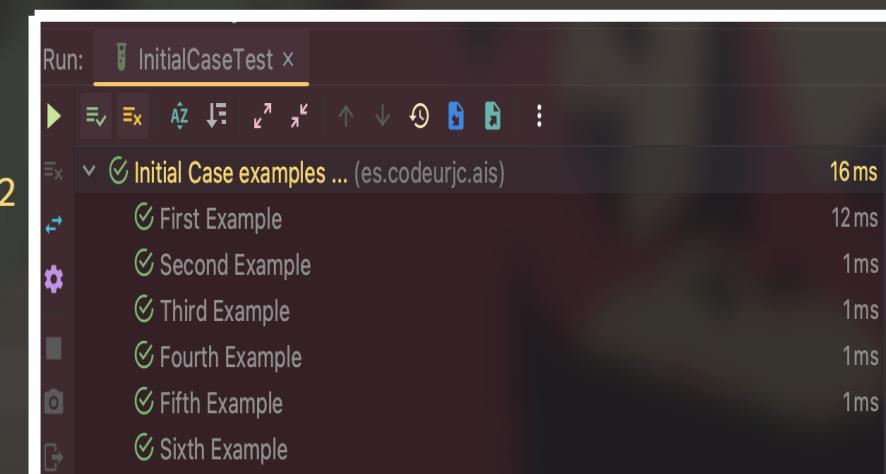
TEST6.WA

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".

# 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
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
92     no usages new *
93
94     @Test
95     @DisplayName("Seventh Example")
96     public void test7(){
97         card_1 = new Card( name: "Carta 1", attack: 1500, defense: 2850, Position.DEFENSE);
98         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500, Position.DEFENSE);

```

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.

```

    Assertions.assertThrows(IllegalPositionException.class, () →
        result = Combat.combat(card_1, card_2)
    );
}
}

3     usage new *
4     public class IllegalPositionException extends Exception{
5
6         no usages new *
7         public IllegalPositionException() {
8             super();
9         }
10
11        no usages new *
12        public IllegalPositionException(String msg){
13            super(msg);
14        }
15
16    }

```

## 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) → Gana Carta 2. Atacante pierde 200 puntos. Carta 1 destruida/a."  
    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);  
    });  
}  
  
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  
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 thro...")  
112     public void test7(){  
113         card_1 = new Card( name:  
114         card_2 = new Card( name:  
115         Assertions.assertThrows  
116         );  
117         @NotNull  
118         public static String combat(@NotNull Card c1, @NotNull Card c2) throws IllegalPositionException{  
119             return c1.name + " vs " + c2.name;  
120         }  
121     }  
122 }
```

# REFACTORIZACIÓN INTERMEDIA

19 usages ER4UG.reverse +1

public enum Position {

10 usages



ATTACK, DEFENSE;

}

s realizado una refactorización intermedia antes de continuar con los casos intermedios. En la clase los casos iniciales, por motivación de variables y sobre por seguridad frente a bles vulnerabilidades, el imo y Último ejemplo de los s iniciales está contenido en subclase con su propio test y `@BeforeEach...()`

...(\*)de esta forma, el séptimo caso de utilizar las variables de la clase principal al ser de este test 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.

Initial Case examples ...")

InitialCaseTest {

static String expected;

private static String result;

protected static Card card\_1;

16 usages

protected static Card card\_2;

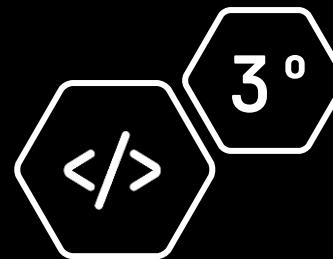
10



# PRÁCTICA 2. TEST DRIVEN DEVELOPMENT

## AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE

### CASO INTERMEDIO



#### INDICE

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

```

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 }

```

26 usages ER4UG.reverse \*

```

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

```

```

13     private Position.EFFECT effect;
14
15     15 usages gu4re *
16
17     public Card(String name, int attack, int defense, @NotNull Position position) {
18         this.attack = attack;
19         this.position = position;
20         this.defense = defense;
21         this.effect = Position.EFFECT.NA;
22     }
23
24     2 usages new *
25
26     public Card(String name, int attack, int defense, @NotNull Position position,
27                 Position.EFFECT effect) {
28         this(name, attack, defense, position);
29         this.effect = effect;
30     }

```

# TEST8.CE

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.com
35         Assertions.assertEquals(
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.

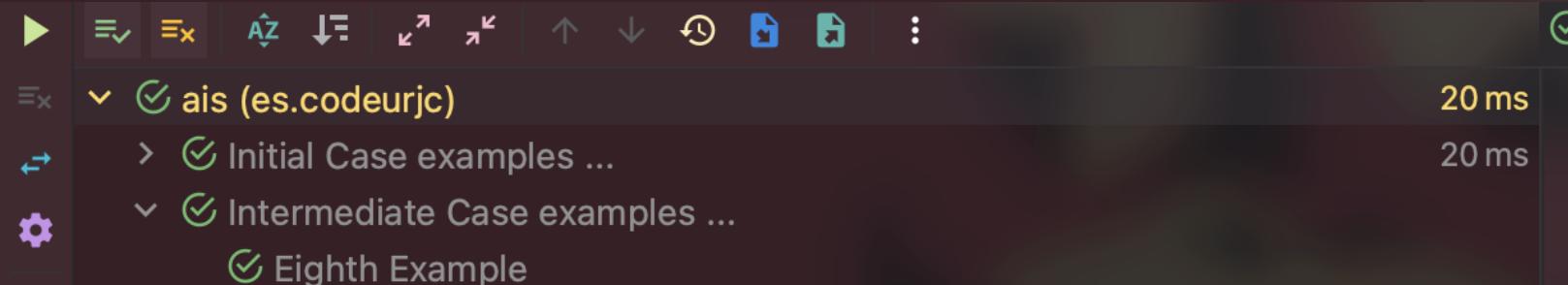
```

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.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.");
                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     }

```

```

public void test1() { // escogido el test 1 como ejemplo
    expected = "Carta 1 (3000/2500/Posición: Ataque) vs Carta 2 (2500/2100/Posición: Ataque) " +
               "→ Gana Carta 1. Defensor pierde 500 puntos. " +
               "Carta 2 destruido/a.";
}

```

```

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

```

# 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

```

23 public void test8(){
24     expected = "Carta 1 (2000/2850/Posición: Ataque/Efecto: IMMORTAL) vs Carta 2 " +
25             "(2000/1500/Posición: Ataque/Efecto: NA) → Empate. Carta 2 destruido/a.";
26     card_1 = new Card( name: "Carta 1", attack: 2000, defense: 2850,
27                         Position.ATTACK, Position.EFFECT.IMMORTAL);
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
37 @Test
38 @DisplayName("Ninth Example")
39 public void test9(){
40     expected = "Carta 1 (1800/2850/Posición: Ataque/Efecto: IMMORTAL) vs Carta 2 " +
41             "(2000/1500/Posición: Ataque/Efecto: NA) → Gana Carta 2. Atacante pierde 200 puntos.";
42     card_1 = new Card( name: "Carta 1", attack: 1800, defense: 2850,
43                         Position.ATTACK, Position.EFFECT.IMMORTAL);
44     card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
45                         Position.ATTACK, Position.EFFECT.NA);
46     Assertions.assertDoesNotThrow(() → {
47         result = Combat.combat(card_1, card_2);
48         Assertions.assertEquals(expected, result);
49     });
50 }
```

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.ETERNAL))
                    combat_resolution.append("Gana Carta 2. Atacante pierde 200 puntos.");
                else
                    combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruida/a.");
            }
            case 0 → {
                if (c1.getEffect() ≠ null && c1.getEffect().equals(Position.EFFECT.ETERNAL))
                    combat_resolution.append("Empate. Carta 2 destruida/a.");
                else
                    combat_resolution.append("Empate. Ambas cartas destruidas.");
            }
            case 1 → combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruida/a.");
        }
    }
}
```

n: IntermediateCaseTest.test9 ×

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".

Process finished with exit code 0

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

```

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 → "la posición de defensa"; a 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 *  
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 }
no usages new*
49 @Test
50 @DisplayName("Tenth Example")
51 public void test10(){
52     expected = "Carta 1 (2100/2850/Posición: Ataque/Efecto: NA) vs Carta 2 " +
53                 "(2000/1500/Posición: Ataque/Efecto: IMMORTAL) → Gana Carta 1. Defensor pierde 100 puntos.";
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 }

```

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     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 }
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)

```
1 usage  gu4re +1 *  ▲4 ▲1 ▲42 1
private static void attackVsAttack(@NotNull Card card1, @NotNull Card card2){    TEST10.AC
    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack()))){      case -1 → {
        if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.IMITARIAL))
            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.IMITARIAL))
            combat_resolution.append("Empate. Carta 2 destruido/a.");
        else
            combat_resolution.append("Empate. Ambas cartas destruidas.");
    }
    case 1 → {
        if (card2.getEffect() ≠ null && card2.getEffect().equals(Position.EFFECT.IMITARIAL))
            combat_resolution.append("Gana Carta 1. Defensor pierde 100 puntos.");
        else
            combat_resolution.append("Gana Carta 1. Defensor pierde 500 puntos. Carta 2 destruido/a.");
    }
    default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
}
```

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

Process finished with exit code 0

# 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.

# TEST11CE

```
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.";
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   }
45 1 usage  gu4re *
46   private static @NotNull String effectToString(@NotNull Position EFFECT, Effect effect)
47 }
```

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.

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 Ademástr hemos añadido la conversión correspondiente");
            }
        default → throw new UnsupportedOperationException("No se soporta la operación");
    }
}
1 usage ↗ gu4re *
```

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.

```

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

```

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
    @Test
    @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: orgjunitjupiterjunit-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

```

44
45     public void test13(){
46         expected = "Carta 1 (2000/1000/Posición: Ataque/Efecto: N/A) vs Carta 2 (2000/1500/Posición: Defensa/Efecto: Toque mortal) " +
47             "→ Gana Carta 1. Ambas cartas destruidas.";
48         card_1 = new Card( name: "Carta 1", attack: 2000, defense: 1000,
49                           Position.ATTACK, Position.EFFECT.NA);
50         card_2 = new Card( name: "Carta 2", attack: 2000, defense: 1500,
51                           Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
52         Assertions.assertDoesNotThrow(() → {
53             result = Combat.combat(card_1, card_2);
54             Assertions.assertEquals(expected, result);
55         });
56     }
57     no usages new*
58 @Test
59 @DisplayName("Fourteenth Example")
60 public void test14(){
61     expected = "Carta 1 (1000/1000/Posición: Ataque/Efecto: Toque mortal) vs Carta 2 (0/1000/Posición: " +
62                 "Defensa/Efecto: Toque mortal) → Empate. Ambas cartas destruidas.";
63     card_1 = new Card( name: "Carta 1", attack: 1000, defense: 1000,
64                           Position.ATTACK, Position.EFFECT.MORTAL_TOUCH);
65     card_2 = new Card( name: "Carta 2", attack: 0, defense: 1000,
66                           Position.DEFENSE, Position.EFFECT.MORTAL_TOUCH);
67     Assertions.assertDoesNotThrow(() → {
68         result = Combat.combat(card_1, card_2);
69         Assertions.assertEquals(expected, result);
70     });
71 }

```

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:
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     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     // 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             .append((card1.getEffect() ≠ null && card2.getEffect() ≠ null &&
46                     card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
47                     && card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH))
48                     ? " Ambas cartas destruidas." : "");
49         case 1 → {
50             combat_resolution.append("Gana Carta 1.");
51             if (card2.getEffect() = null)
52                 combat_resolution.append(" Carta 2 destruido/a.");
53             else if (card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
54                     && card1.getDefense() < DEFENSE_THRESHOLD)
55                 combat_resolution.append(" Ambas cartas destruidas.");
56         }
57         default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
58     }
59 }
60 1 usage  ↗gu4re
61 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.

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

## TEST15.WA

```

09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
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);
});
no usages new*
@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.comb...
        Assertions.assertEquals(expected, result);
    });
}
}

```

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.  
 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 ~~st~~ extrayendo ~~ring~~ la condición "Not supported yet"; "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 st extrayendo ring la condición "Not supported yet";
7     "card1.getEffect() != null" afuera para solo realizarse una única vez. Además de esta forma
8     evitar también gracias al operador ternario,
9     un aviso por "Cognitive Complexity".
10    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack())))
11        case -1 → {
12            if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.IMITORIAL))
13                combat_resolution.append("Gana Carta 2. Atacante pierde 200 puntos.");
14            else if (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH))
15                combat_resolution.append("Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas.");
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.IMITORIAL))
21                combat_resolution.append("Empate. Carta 2 destruido/a.");
22            else
23                combat_resolution.append("Empate. Ambas cartas destruidas.");
24        }
25    switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getAttack())))
26        case -1 → {
27            if (card1.getEffect() ≠ null)
28                combat_resolution.append((card1.getEffect().equals(Position.EFFECT.IMITORIAL)
29                    ? "Gana Carta 2. Atacante pierde 200 puntos."
30                    : "Gana Carta 2. Atacante pierde 500 puntos. Ambas cartas destruidas."));
31            else
32                combat_resolution.append("Gana Carta 2. Atacante pierde 300 puntos. Carta 1 destruido/a.");
33        }
}

```

```
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
```

Este Test produce un problema en compilación, ya que aparece un nro denominado presión, que tendremos de la forma que se esquina inferior derecha, para posteriormente ejecutar el test y ver qué ocurre.

```
    @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 *
```

Este Test produce un problema en compilación, ya que aparece un nro denominado presión, que tendremos de la forma que se esquina inferior derecha, para posteriormente ejecutar el test y ver qué ocurre.

```
    @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);
        });
    }
```

```
public enum Position {
    ATTACK, DEFENSE;
}
enum EFFECT{
    MORTAL_TOUCH, NA, MORAL_INJURY;
}
```

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

```
public enum Position {  
    24 usages  
    ATTACK, DEFENSE;  
    29 usages   ➔ gu4re *  
    enum EFFECT{  
        7 usages  
        IMMORTAL, NA, MORTAL_TOUCH, PRESSURE  
    }  
}
```

```

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

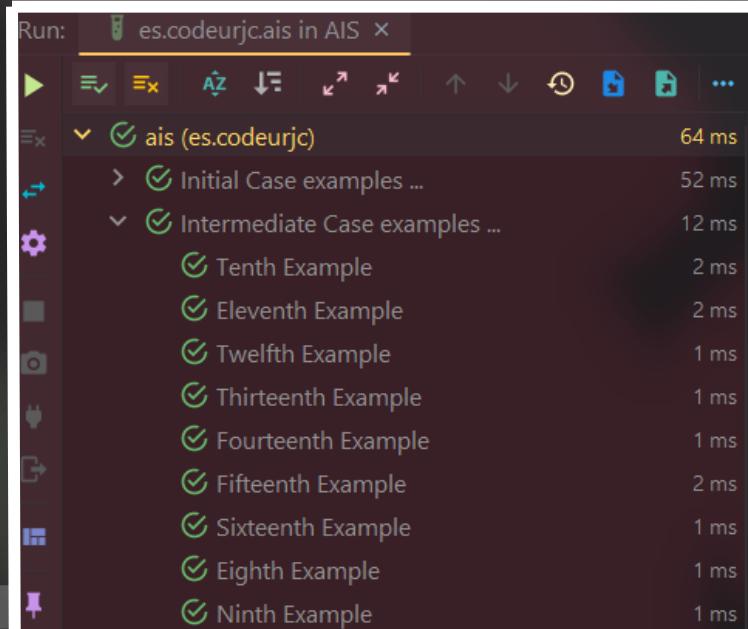
Nos ha pasado algo sorprendente en este test, y es que el test ha dado acierto con tan solo traducir "PRESSURE -> Presión" por el formato del efecto a String, y no hemos tenido que implementar ningún tipo de funcionalidad para el efecto de presión, ya que como venimos de Test anteriores, coincide que 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 gu4re +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("Carta %d. Atacante pierde %d puntos.", cardNumber, card.getAttack() - DEFENSE_THRESHOLD));  
    cardInfoBuilder.append(String.format("Carta %d. Defendiente pierde %d puntos.", cardNumber, card.getDefense() - DEFENSE_THRESHOLD));  
    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 destruida/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

```
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);
});
no usages ↗ gu4re *
@Test
@DisplayName("Seventeenth Example")
public void test17(){
    expected = "Carta 1 (2000/0/Posición: Ataque/Efecto: Presión) → Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.";
    "Ataque/Efecto: Presión) → Gana Carta 1. Defensor pierde 250 puntos. Carta 2 destruido/a.";
    card_1 = new Card( name: "Carta 1", attack: 2000, defense: 0,
        Position.ATTACK, Position.EFFECT.NA);
    card_2 = new Card( name: "Carta 2", attack: 1500, defense: 0,
        Position.ATTACK, Position.EFFECT.PRESSURE);
    Assertions.assertDoesNotThrow(() -> {
        result = Combat.combat(card_1, card_2);
        Assertions.assertEquals(expected, result);
    });
}
```

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 o/0 y pesten 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

e Refactor this method to reduce its Cognitive Complexity from 17 to the 15 allowed

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.

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 nuestro 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.IMMORTAL)  
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.IMMORTAL)  
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.IMMORTAL)  
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 estándolos. Sigamos con el método `attackVsDefense`.

# REFACTORIZACIÓN POSTERIOR 2

```

35     default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
36 }
37
38 1 usage  ↗ gu4re +1 *
39
40 private static void attackVsDefense(@NotNull Card card1, @NotNull Card card2){
41     // Local variables
42     final int DEFENSE_THRESHOLD = 2000;
43     lostPoints = (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.PRESSURE))
44         ? Math.abs(card1.getAttack() - card2.getDefense()) / 2
45         : Math.abs(card1.getAttack() - card2.getDefense());
46     switch (Integer.signum(Integer.compare(card1.getAttack(), card2.getDefense()))) {
47         case -1 → combat_resolution.append(String.format("Gana Carta 2. Atacante pierde %d puntos.%s", lostPoints,
48             (card1.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
49                 && card2.getDefense() < DEFENSE_THRESHOLD)
50                 ? CARD2_DESTROYED
51                 : EMPTY));
52         case 0 → combat_resolution.append("Empate.");
53             && card2.getEffect() ≠ null && card1.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
54             && card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
55             ? BOTH_CARDS_DESTROYED
56             : EMPTY);
57         case 1 → {
58             combat_resolution.append("Gana Carta 1.");
59             if (card2.getEffect() = null)
60                 combat_resolution.append(CARD2_DESTROYED);
61             else if (card2.getEffect().equals(Position.EFFECT.MORTAL_TOUCH)
62                 && card1.getDefense() < DEFENSE_THRESHOLD)
63                 combat_resolution.append(BOTH_CARDS_DESTROYED);
64             }
65             default → throw new UnsupportedOperationException(NOT_S
66 }
67
68 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

```

6   2 usages  gu4re +1 *
7
8     private static @NotNull String getCardInfo(@NotNull Card card) {
9       // Local variables
10      StringBuilder cardInfoBuilder = new StringBuilder();
11      int cardNumber = card.getName().equals(CARD1_NAME) ? 1 : 2;
12      cardInfoBuilder.append(String.format("Carta %d (%d/%d/Posición: %s)", cardNumber, card.getAttack(), card.getDefense(), card.getPosition()));
13      if (card.getEffect() != null)
14        cardInfoBuilder.append(String.format("/Efecto: %s", effectToObserve(card.getEffect())));
15      cardInfoBuilder.append(")");
16      return cardInfoBuilder.toString();
17    }
18
19    17 usages  gu4re +1 *
20
21    public static @NotNull String combat(@NotNull Card card1, @NotNull Card card2) throws IllegalPositionException,
22      UnsupportedOperationException {
23      // Local variables
24      final String ERROR_MESSAGE = "La carta atacante no puede estar en una posición de Defensa";
25      final int ZERO = 0;
26      // Clearing StringBuilder at the beginning of the method
27      combat_resolution.setLength(ZERO);
28      if (card1.getPosition().equals(Position.DEFENSE))
29        throw new IllegalPositionException(ERROR_MESSAGE);
30      combat_resolution.append(getCardInfo(card1)).append(" vs ").append(getCardInfo(card2)).append(" ");
31      switch (card2.getPosition()) {
32        case ATTACK → attackVsAttack(card1, card2);
33        case DEFENSE → attackVsDefense(card1, card2);
34        default → throw new UnsupportedOperationException(NOT_SUPPORTED_YET);
35      }
36      return combat_resolution.toString();
37    }
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
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
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
5010
5011
5012
5013
5014
5015
5016
5017
5018
5019
5020
5021
5022
5023
5024
5025
5026
5027
5028
5029
5030
5031
5032
5033
5034
5035
5036
5037
5038
5039
5040
5041
5042
5043
5044
5045
5046
5047
5048
5049
5050
5051
5052
5053
5054
5055
5056
5057
5058
5059
5060
5061
5062
5063
5064
5065
5066
5067
5068
5069
5070
5071
5072
5073
5074
5075
5076
5077
5078
5079
5080
5081
5082
5083
5084
5085
5086
5087
5088
5089
5090
5091
5092
5093
5094
5095
5096
5097
5098
5099
50100
50101
50102
50103
50104
50105
50106
50107
50108
50109
50110
50111
50112
50113
50114
50115
50116
50117
50118
50119
50120
50121
50122
50123
50124
50125
50126
50127
50128
50129
50130
50131
50132
50133
50134
50135
50136
50137
50138
50139
50140
50141
50142
50143
50144
50145
50146
50147
50148
50149
50150
50151
50152
50153
50154
50155
50156
50157
50158
50159
50160
50161
50162
50163
50164
50165
50166
50167
50168
50169
50170
50171
50172
50173
50174
50175
50176
50177
50178
50179
50180
50181
50182
50183
50184
50185
50186
50187
50188
50189
50190
50191
50192
50193
50194
50195
50196
50197
50198
50199
50200
50201
50202
50203
50204
50205
50206
50207
50208
50209
50210
50211
50212
50213
50214
50215
50216
50217
50218
50219
50220
50221
50222
50223
50224
50225
50226
50227
50228
50229
50230
50231
50232
50233
50234
50235
50236
50237
50238
50239
50240
50241
50242
50243
50244
50245
50246
50247
50248
50249
50250
50251
50252
50253
50254
50255
50256
50257
50258
50259
50260
50261
50262
50263
50264
50265
50266
50267
50268
50269
50270
50271
50272
50273
50274
50275
50276
50277
50278
50279
50280
50281
50282
50283
50284
50285
50286
50287
50288
50289
50290
50291
50292
50293
50294
50295
50296
50297
50298
50299
50299
50300
50301
50302
50303
50304
50305
50306
50307
50308
50309
50310
50311
50312
50313
50314
50315
50316
50317
50318
50319
50320
50321
50322
50323
50324
50325
50326
50327
50328
50329
50330
50331
50332
50333
50334
50335
50336
50337
50338
50339
50340
50341
50342
50343
50344
50345
50346
50347
50348
50349
50350
50351
50352
50353
50354
50355
50356
50357
50358
50359
50360
50361
50362
50363
50364
50365
50366
50367
50368
50369
50370
50371
50372
50373
50374
50375
50376
50377
50378
50379
50380
50381
50382
50383
50384
50385
50386
50387
50388
50389
50390
50391
50392
50393
50394
50395
50396
50397
50398
50399
50399
50400
50401
50402
50403
50404
50405
50406
50407
50408
50409
50409
50410
50411
50412
50413
50414
50415
50416
50417
50418
50419
50419
50420
50421
50422
50423
50424
50425
50426
50427
50428
50429
50429
50430
50431
50432
50433
50434
50435
50436
50437
50438
50439
50439
50440
50441
50442
50443
50444
50445
50446
50447
50448
50449
50449
50450
50451
50452
50453
50454
50455
50456
50457
50458
50459
50459
50460
50461
50462
50463
50464
50465
50466
50467
50468
50469
50469
50470
50471
50472
50473
50474
50475
50476
50477
50478
50479
50479
50480
50481
50482
50483
50484
50485
50486
50487
50488
50489
50489
50490
50491
50492
50493
50494
50495
50496
50497
50498
50499
50499
50500
50501
50502
50503
50504
50505
50506
50507
50508
50509
50509
50510
50511
50512
50513
50514
50515
50516
50517
50518
50519
50519
50520
50521
50522
50523
50524
50525
50526
50527
50528
50529
50529
50530
50531
50532
50533
50534
50535
50536
50537
50538
50539
50539
50540
50541
50542
50543
50544
50545
50546
50547
50548
50549
50549
50550
50551
50552
50553
50554
50555
50556
50557
50558
50559
50559
50560
50561
50562
50563
50564
50565
50566
50567
50568
50569
50569
50570
50571
50572
50573
50574
50575
50576
50577
50578
50579
50579
50580
50581
50582
50583
50584
50585
50586
50587
50588
50589
50589
50590
50591
50592
50593
50594
50595
50596
50597
50598
50599
50599
50600
50601
50602
50603
50604
50605
50606
50607
50608
50609
50609
50610
50611
50612
50613
50614
50615
50616
50617
50618
50619
50619
50620
50621
50622
50623
50624
50625
50626
50627
50628
50629
50629
50630
50631
50632
50633
50634
50635
50636
50637
50638
50639
50639
50640
50641
50642
50643
50644
50645
50646
50647
50648
50649
50649
50650
50651
50652
50653
50654
50655
50656
50657
50658
50659
50659
50660
50661
50662
50663
50664
50665
50666
50667
50668
50669
50669
50670
50671
50672
50673
50674
50675
50676
50677
50678
50679
50679
50680
50681
50682
50683
50684
50685
50686
50687
50688
50689
50689
50690
50691
50692
50693
50694
50695
50696
50697
50698
50699
50699
50700
50701
50702
50703
50704
50705
50706
50707
50708
50709
50709
50710
50711
50712
50713
50714
50715
50716
50717
50718
50719
50719
50720
50721
50722
50723
50724
50725
50726
50727
50728
50729
50729
50730
50731
50732
50733
50734
50735
50736
50737
50738
50739
50739
50740
50741
50742
50743
50744
50745
50746
50747
50748
50749
50749
50750
50751
50752
50753
50754
50755
50756
50757
50758
50759
50759
50760
50761
50762
50763
50764
50765
50766
50767
50768
50769
50769
50770
50771
50772
50773
50774
50775
50776
50777
50778
50779
50779
50780
50781
50782
50783
50784
50785
50786
50787
50788
50789
50789
50790
50791
50792
50793
50794
50795
50796
50797
50798
50799
50799
50800
50801
50802
50803
50804
50805
50806
50807
50808
50809
50809
50810
50811
50812
50813
50814
50815
50816
50817
50818
50819
50819
50820
50821
50822
50823
50824
50825
50826
50827
50828
50829
50829
50830
50831
50832
50833
50834
50835
50836
50837
50838
50839
50839
50840
50841
50842
50843
50844
50845
50846
50847
50848
50849
50849
50850
50851
50852
50853
50854
50855
50856
50857
50858
50859
50859
50860
50861
50862
50863
50864
50865
50866
50867
50868
50869
50869
50870
50871
50872
50873
50874
50875
50876
50877
50878
50879
50879
50880
50881
50882
50883
50884
50885
50886
50887
50888
50889
50889
50890
50891
50892
50893
50894
50895
50896
50897
50898
50899
50899
50900
50901
50902
50903
50904
50905
50906
50907
50908
50909
50909
50910
50911
50912
50913
50914
50915
50916
50917
50918
50919
50919
50920
50921
50922
50923
50924
50925
50926
50927
50928
50929
50929
50930
50931
50932
50933
50934
50935
50936
50937
50938
50939
50939
50940
50941
50942
50943
50944
50945
50946
50947
50948
50949
50949
50950
50951
50952
50953
50954
50955
50956
50957
50958
50959
50959
50960
50961
50962
50963
50964
50965
50966
50967
50968
50969
50969
50970
50971
50972
50973
50974
50975
50976
50977
50978
50979
50979
50980
50981
50982
50983
50984
50985
50986
50987
50988
50989
50989
50990
50991
50992
50993
50994
50995
50996
50997
50998
50999
50999
51000
51001
51002
51003
51004
51005
51006
51007
51008
51009
51009
51010
51011
51012
51013
51014
51015
51016
51017
51018
51019
51019
51020
51021
51022
51023
51024
51025
51026
51027
51028
51029
51029
51030
51031
51032
51033
51034
51035
51036
51037
51038
51039
51039
51040
51041
51042
51043
51044
51045
51046
51047
51048
51049
51049
51050
51051
51052
51053
51054
51055
51056
51057
51058
51059
51059
51060
51061
51062
51063
51064
51065
51066
51067
51068
51069
51069
51070
51071
51072
51073
51074
51075
51076
51077
51078
51079
51079
51080
51081
51082
51083
51084
51085
51086
51087
51088
51089
51089
51090
51091
51092
51093
51094
51095
51096
51097
51098
51099
51099
51100
51101
51102
51103
51104
51105
51106
51107
51108
51109
51109
51110
51111
51112
51113
51114
51115
51116
51117
51118
51119
51119
51120
51121
51122
51123
51124
51125
51126
51127
51128
51129
51129
51130
51131
51132
51133
51134
51135
51136
51137
51138
51139
51139
51140
51141
51142
51143
51144
51145
51146
51147
51148
51149
51149
51150
51151
51152
51153
51154
51155
51156
51157
51158
51159
51159
51160
51161
51162
51163
51164
51165
51166
51167
51168
51169
51169
51170
51171
51172
51173
51174
51175
51176
51177
51178
51179
51179
51180
51181
51182
51183
51184
51185
51186
51187
51188
51189
51189
51190
51191
51192
51193
51194
51195
51196
51197
51198
51199
51199
51200
51201
51202
51203
51204
51205
51206
51207
51208
51209
51209
51210
51211
51212
51213
51214
51215
51216
51217
51218
51219
51219
51220
51221
51222
51223
51224
51225
51226
51227
51228
51229
51229
51230
51231
51232
51233
51234
51235
51236
51237
51238
51239
51239
51240
51241
51242
51243
51244
51245
51246
51247
51248
51249
51249
51250
51251
51252
51253
51254
51255
51256
51257
51258
51259
51259
51260
51261
51262
51263
51264
51265
51266
51267
51268
51269
51269
51270
51271
51272
51273
51274
51275
51276
51277
51278
51279
51279
51280
51281
51282
51283
51284
51285
51286
51287
51288
51289
51289
51290
51291
51292
51293
51294
51295
51296
51297
51298
51299
51299
51300
51301
51302
51303
51304
51305
51306
51307
51308
51309
51309
51310
51311
51312
51313
51314
51315
51316
51317
51318
51319
51319
51320
51321
51322
51323
51324
51325
51326
51327
51328
51329
51329
51330
51331
51332
51333
51334
51335
51336
51337
51338
51339
51339
51340
51341
51342
51343
51344
51345
51346
51347
51348
51349
51349
51350
51351
51352
51353
51354
51355
51356
51357
51358
51359
51359
51360
51361
51362
51363
51364
51365
51366
51367
51368
51369
51369
51370
51371
51372
51373
51374
51375
51376
51377
51378
51379
51379
51380
51381
51382
51383
51384
51
```