David Fuentes Insa
Jan Planas Batllori
Dimecres 10:30-12:30
Poker

Classe Card

Funcionalitat: Constructor de la clase card

Localització: Card.java, class: Card, public Card(Suit suit, Rank rank)

Test: CardTest.java, class: Card,

1. Primera versió TDD:

```java
// Constructor que valida que el palo y el rango no sean nulos y que sean válidos.
   public Card(Suit suit, Rank rank) {
       this.suit = suit;
       this.rank = rank;
   }

  @Test
  void testCardConstructor_ValidSuitAndRank() {
      Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);
      assertNotNull(card, "Card should be created with valid suit and rank.");
      assertEquals(Card.Suit.HEARTS, card.getSuit(), "Suit should be HEARTS.");
      assertEquals(Card.Rank.A, card.getRank(), "Rank should be ACE.");
  }
```

2. Versió final TDD:

```java
// Constructor que valida que el palo y el rango no sean nulos y que sean válidos.
public Card(Suit suit, Rank rank) {
    try {
        if (suit == null || rank == null) {
            throw new IllegalArgumentException("Suit and rank cannot be null.");
        }

        // Verifica que los valores de suit y rank sean válidos.
        Card.Suit.valueOf(suit.name());
        Card.Rank.valueOf(rank.name());

    } catch (IllegalArgumentException e) {
        throw new IllegalArgumentException("Invalid Suit or Rank", e);
    }
    this.suit = suit;
    this.rank = rank;
}
```

```java
@Test
public void testConstructor_nullValues_throwsException() {
    assertThrows(IllegalArgumentException.class, () -> new Card(null, Card.Rank.A));
    assertThrows(IllegalArgumentException.class, () -> new Card(Card.Suit.HEARTS, null));
    assertThrows(IllegalArgumentException.class, () -> new Card(null, null));
}

@Test
void constructor_ShouldThrowException_WhenInvalidSuitOrRank() {
   assertThrows(IllegalArgumentException.class, () -> new Card(Card.Suit.valueOf("INVALID"), Card.Rank.A));
   assertThrows(IllegalArgumentException.class, () -> new Card(Card.Suit.HEARTS, Card.Rank.valueOf("INVALID")));
}
```

```java
@Test
void testCardConstructor_ValidSuitAndRank() {
    Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);
    assertNotNull(card, "Card should be created with valid suit and rank.");
    assertEquals(Card.Suit.HEARTS, card.getSuit(), "Suit should be HEARTS.");
    assertEquals(Card.Rank.A, card.getRank(), "Rank should be ACE.");
}
```

3. void constructor_ShouldCreateCard_WhenSuitAndRankAreValid()

Test de Caixa Negra: els valors frontera.

4. void constructor_ShouldCreateCard_WithValidSuitsAndRanks ()

Test de Caixa Negra: pairwise testing.

Tets de Caixa blanca: statement coverage.

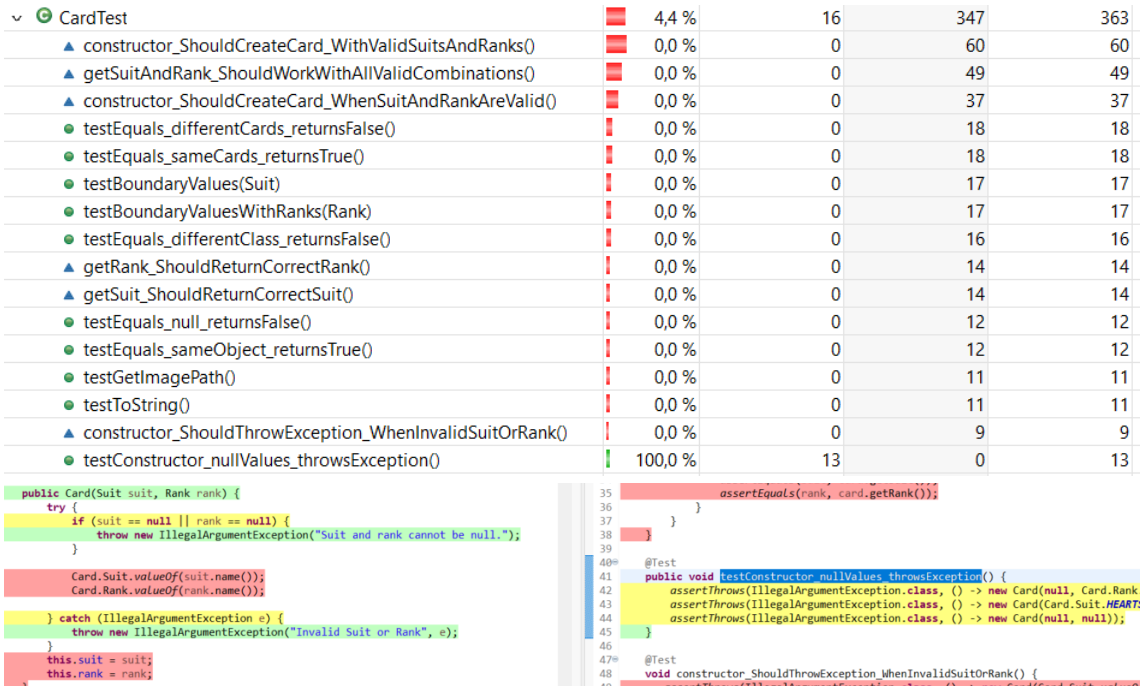
CardTest	17,4 %	63	300	363
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValuesOrInvalid_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	100,0 %	60	0	60
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	100,0 %	60	0	60

```java
public Card(Suit suit, Rank rank) {
    try {
        if (suit == null || rank == null) {
            throw new IllegalArgumentException("Suit and rank cannot be null.");
        }

        Card.Suit.valueOf(suit.name());
        Card.Rank.valueOf(rank.name());

    } catch (IllegalArgumentException e) {
        throw new IllegalArgumentException("Invalid Suit or Rank", e);
    }
    this.suit = suit;
    this.rank = rank;
}
```

```java
}

@Test
void constructor_ShouldCreateCard_WithValidSuitsAndRanks() {
    for (Card.Suit suit : Card.Suit.values()) {
        for (Card.Rank rank : Card.Rank.values()) {
            if ((suit == Card.Suit.HEARTS && rank == Card.Rank.A) ||
                (suit == Card.Suit.SPADES && rank == Card.Rank.TWO)) {
                continue;
            }
            Card card = new Card(suit, rank);
            assertEquals(suit, card.getSuit());
            assertEquals(rank, card.getRank());
        }
    }
}
```

5. void testConstructor_nullValues_throwsException ()

Tets de Caixa blanca: statement coverage, decision coverage i condition coverage.



CardTest	4,4 %	16	347	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testConstructor_nullValues_throwsException()	100,0 %	13	0	13

```
public Card(Suit suit, Rank rank) {
    try {
        if (suit == null || rank == null) {
            throw new IllegalArgumentException("Suit and rank cannot be null.");
        }

        Card.Suit.valueOf(suit.name());
        Card.Rank.valueOf(rank.name());

    } catch (IllegalArgumentException e) {
        throw new IllegalArgumentException("Invalid Suit or Rank", e);
    }
    this.suit = suit;
    this.rank = rank;
```

```
35              assertEquals(rank, card.getRank());
36          }
37      }
38  }
39
40  @Test
41  public void testConstructor_nullValues_throwsException() {
42      assertThrows(IllegalArgumentException.class, () -> new Card(null, Card.Rank
43      assertThrows(IllegalArgumentException.class, () -> new Card(Card.Suit.HEART
44      assertThrows(IllegalArgumentException.class, () -> new Card(null, null));
45  }
46
47  @Test
48  void constructor_ShouldThrowException_WhenInvalidSuitOrRank() {
```

6. void constructor_ShouldThrowException_WhenInvalidSuitOrRank()

Test de Caixa Negra: valors frontera.

Funcionalitat: Geter de l'atribut suit

Localització: Card.java, class: Card, public Suit getSuit()

Test: CardTest.java, class: Card,

void getSuit_ShouldReturnCorrectSuit()

Test de Caixa blanca: statement coverage.

CardTest	5,5 %	20	343	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testBoundaryValues(Suit)	100,0 %	17	0	17

Funcionalitat: Geter de l'atribut rank

Localització: Card.java, class: Card, public Rank getRank()

Test: CardTest.java, class: Card,

void getRank_ShouldReturnCorrectRank ()

Test de Caixa blanca: statement coverage.

CardTest	4,7 %	17	346	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
getRank_ShouldReturnCorrectRank()	100,0 %	14	0	14

Funcionalitat: Geter de l'atribut rank I suit

Localització: Card.java, class: Card, public Rank getRank() public Suit getSuit()

Test: CardTest.java, class: Card,

void getSuitAndRank_ShouldWorkWithAllValidCombinations()

Test de Caixa Negra: particions equivalents.

Funcionalitat: Comparador de dues cartes

Localització: Card.java, class: Card, public boolean equals(Object obj)

Test: CardTest.java, class: Card,

1. Primera versió TDD:

```java
public boolean equals(Object obj) {
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}

@Test
public void testEquals() {
    // Crear dos cartas con el mismo valor
    Card card1 = new Card(Card.Suit.HEARTS, Card.Rank.A);
    Card card2 = new Card(Card.Suit.HEARTS, Card.Rank.A);

    // Verificar que las dos cartas son iguales
    assertTrue(card1.equals(card2), "Cards with same suit and rank should be equal");

    // Crear una carta diferente
    Card card3 = new Card(Card.Suit.CLUBS, Card.Rank.A);

    // Verificar que las cartas son diferentes
    assertFalse(card1.equals(card3), "Cards with different suit should not be equal");

    // Crear una carta con un rank diferente
    Card card4 = new Card(Card.Suit.HEARTS, Card.Rank.K);

    // Verificar que las cartas son diferentes
    assertFalse(card1.equals(card4), "Cards with different rank should not be equal");
}
```

2. Versió final TDD

```java
// Método que compara dos cartas para verificar si son iguales (mismo palo y rango).
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}

@Test
public void testEquals2() {
    // Crear dos cartas con el mismo valor (mismo suit y rank)
    Card card1 = new Card(Card.Suit.HEARTS, Card.Rank.A);
    Card card2 = new Card(Card.Suit.HEARTS, Card.Rank.A);

    // Verificar que las dos cartas son iguales
    assertTrue(card1.equals(card2), "Cards with the same suit and rank should be equal");

    // Crear una carta diferente en suit
    Card card3 = new Card(Card.Suit.CLUBS, Card.Rank.A);

    // Verificar que las cartas son diferentes por el suit
    assertFalse(card1.equals(card3), "Cards with different suits should not be equal");

    // Crear una carta diferente en rank
    Card card4 = new Card(Card.Suit.HEARTS, Card.Rank.K);

    // Verificar que las cartas son diferentes por el rank
    assertFalse(card1.equals(card4), "Cards with different ranks should not be equal");

    // Verificar que la misma carta es igual a sí misma
    assertTrue(card1.equals(card1), "A card should be equal to itself");

    // Verificar que una carta no es igual a null
    assertFalse(card1.equals(null), "Card should not be equal to null");

    // Verificar que no son iguales a un objeto de otro tipo
    assertFalse(card1.equals(new Object()), "Card should not be equal to objects of other types");
}
```

3. void testEquals_sameCards_returnsTrue ()

Test de Caixa blanca: statement coverage. També es fa decision coverage.

CardTest	5,8 %	21	342	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testEquals_sameCards_returnsTrue()	100,0 %	18	0	18

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}
```

```
98    @Test
99    public void testEquals_sameCards_returnsTrue() {
100       Card card1 = new Card(Card.Suit.HEARTS, Card.Rank.A);
101       Card card2 = new Card(Card.Suit.HEARTS, Card.Rank.A);
102       assertTrue(card1.equals(card2), "The cards should be equal.");
103   }
104
```

4. void testEquals_sameObject_returnsTrue()

Test de Caixa blanca: statement coverage, decision coverage i condition coverage.

CardTest	4,1 %	15	348	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testEquals_sameObject_returnsTrue()	100,0 %	12	0	12

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}
```

```
104
105   @Test
106   public void testEquals_sameObject_returnsTrue() {
107       Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);
108       assertTrue(card.equals(card), "An object should always be equal to itself.");
109   }
110
```

5. void testEquals_differentCards_returnsFalse()

Test de Caixa blanca: condition coverage

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}
```

```
110
111   @Test
112   public void testEquals_differentCards_returnsFalse() {
113       Card card1 = new Card(Card.Suit.HEARTS, Card.Rank.A);
114       Card card2 = new Card(Card.Suit.CLUBS, Card.Rank.K);
115       assertFalse(card1.equals(card2), "Cards with different suit and rank should not b
116   }
117
```

6. void testEquals_null_returnsFalse()

Test de Caixa blanca: statement coverage, decision coverage i condition coverage.

CardTest	4,1 %	15	348	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testEquals_null_returnsFalse()	100,0 %	12	0	12

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}
```

```
119  public void testEquals_null_returnsFalse() {
120      Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);
121      assertFalse(card.equals(null), "A card should not be equal to null.");
122  }
123
124  @Test
125  public void testEquals_differentClass_returnsFalse() {
```

7. void testEquals_differentClass_returnsFalse()

Test de Caixa blanca: condition coverage.

```
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Card card = (Card) obj;
    return suit == card.suit && rank == card.rank;
}
```

```
122  }
123
124  @Test
125  public void testEquals_differentClass_returnsFalse() {
126      Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);
127      Object obj = new Object();
128      assertFalse(card.equals(obj), "A card should not be equal to an object of a diffe
129  }
```

Funcionalitat: Obtenir el path de la imatge de la carta solicitada

Localització: Card.java, class: Card, public String getImagePath()

Test: CardTest.java, class: Card,

void testGetImagePath()

Test de Caixa blanca: statement coverage.

CardTest	3,9 %	14	349	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testToString()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testGetImagePath()	100,0 %	11	0	11

Funcionalitat: Obtenir en text el valor i el rang de la carta solicitada

Localització: Card.java, class: Card, public String toString()

Test: CardTest.java, class: Card,

void testToString ()

Test de Caixa blanca: statement coverage.

CardTest	3,9 %	14	349	363
constructor_ShouldCreateCard_WithValidSuitsAndRanks()	0,0 %	0	60	60
getSuitAndRank_ShouldWorkWithAllValidCombinations()	0,0 %	0	49	49
constructor_ShouldCreateCard_WhenSuitAndRankAreValid()	0,0 %	0	37	37
testEquals_differentCards_returnsFalse()	0,0 %	0	18	18
testEquals_sameCards_returnsTrue()	0,0 %	0	18	18
testBoundaryValues(Suit)	0,0 %	0	17	17
testBoundaryValuesWithRanks(Rank)	0,0 %	0	17	17
testEquals_differentClass_returnsFalse()	0,0 %	0	16	16
getRank_ShouldReturnCorrectRank()	0,0 %	0	14	14
getSuit_ShouldReturnCorrectSuit()	0,0 %	0	14	14
testConstructor_nullValues_throwsException()	0,0 %	0	13	13
testEquals_null_returnsFalse()	0,0 %	0	12	12
testEquals_sameObject_returnsTrue()	0,0 %	0	12	12
testGetImagePath()	0,0 %	0	11	11
constructor_ShouldThrowException_WhenInvalidSuitOrRank()	0,0 %	0	9	9
testToString()	100,0 %	11	0	11

Classe Deck

Funcionalitat: Constructor de la classe deck

Localització: Deck.java, class: Deck, public Deck()

Test: DeckTest.java, class: Deck,

1. Primera versió TDD

```
    public Deck() {
        this.cards = new ArrayList<>();
}
```

```
@Test
void testDeckConstructor_initializesCardsList() {
    assertNotNull(deck.getCards(), "The cards list should not be null after constructor is called.");
}
```

2. Versió final TDD

```
// Constructor que inicializa el mazo y valida que no haya más de 52 cartas
public Deck() {
        this.cards = new ArrayList<>();
        initializeDeck();
        if (this.cards.size() > 52) {
            throw new IllegalArgumentException("Too many cards in the deck.");
        }
}
```

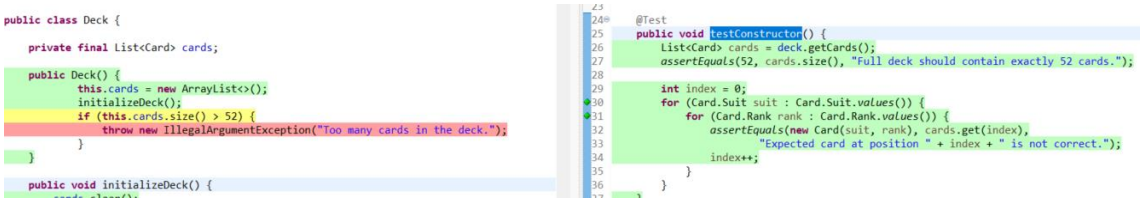
```java
// Constructor que inicializa el mazo y valida que no haya más de 52 cartas
public Deck() {
        this.cards = new ArrayList<>();
        initializeDeck();
}

// Inicializa el mazo con todas las cartas posibles (por cada palo y rango)
private void initializeDeck() {
    cards.clear();
    for (Card.Suit suit : Card.Suit.values()) {
        for (Card.Rank rank : Card.Rank.values()) {
            cards.add(new Card(suit, rank));
        }
    }
}
```

```java
@Test
public void testConstructor() {
    List<Card> cards = deck.getCards();
    assertEquals(52, cards.size(), "Full deck should contain exactly 52 cards.");

    int index = 0;
    for (Card.Suit suit : Card.Suit.values()) {
        for (Card.Rank rank : Card.Rank.values()) {
            assertEquals(new Card(suit, rank), cards.get(index),
                    "Expected card at position " + index + " is not correct.");
            index++;
        }
    }
}
```

3. public void testConstructor()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

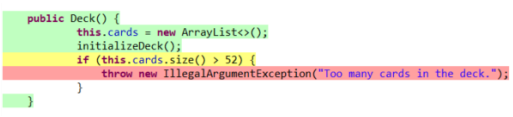
DeckTest	28,3 %	72	182	254
testInitializeDeck()	0,0 %	0	73	73
testResetDeck()	0,0 %	0	39	39
testDealCard()	0,0 %	0	26	26
testShuffle()	0,0 %	0	25	25
testConstructor_tooManyCards()	0,0 %	0	12	12
setUp()	100,0 %	6	0	6
testConstructor()	100,0 %	63	0	63



4. void testConstructor_tooManyCards()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DeckTest	11,0 %	28	226	254
testInitializeDeck()	0,0 %	0	73	73
testConstructor()	0,0 %	0	63	63
testResetDeck()	0,0 %	0	39	39
testDealCard()	0,0 %	0	26	26
testShuffle()	0,0 %	0	25	25
setUp()	100,0 %	6	0	6
testConstructor_tooManyCards()	100,0 %	12	0	12



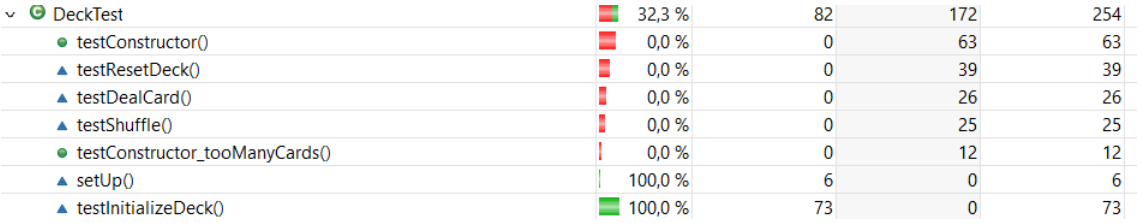
Funcionalitat: Inicialitzar deck

Localització: Deck.java, class: Deck, public initializeDeck()

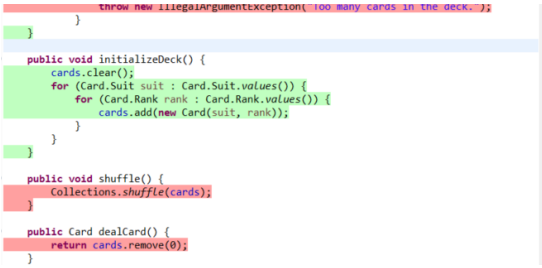
Test: DeckTest.java, class: Deck,

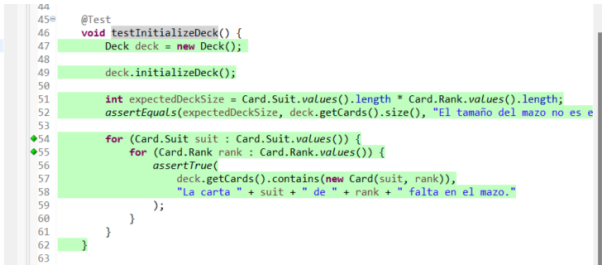
public void initializeDeck()

Test de Caixa blanca: statement coverage i loop testing de loops aniuats.



DeckTest	32,3 %	82	172	254
testConstructor()	0,0 %	0	63	63
testResetDeck()	0,0 %	0	39	39
testDealCard()	0,0 %	0	26	26
testShuffle()	0,0 %	0	25	25
testConstructor_tooManyCards()	0,0 %	0	12	12
setUp()	100,0 %	6	0	6
testInitializeDeck()	100,0 %	73	0	73





Funcionalitat: Barreja de les cartes de deck

Localització: Deck.java, class: Deck, public shuffle()

Test: DeckTest.java, class: Deck,

public void testShuffle()

1. Primera versió TDD

```
@Test
void testShuffle() {
    List<Card> originalOrder = new ArrayList<>(deck.getCards());
    deck.shuffle();
    assertNotEquals(originalOrder, deck.getCards(), "El orden de las cartas debería ser diferente tras barajar.");
}
```

```
//Mezcla las cartas del mazo de forma aleatoria
public void shuffle() {
    cards = cards.reversed();
}
```

2. Versió final TDD

```java
@Test
void testShuffle() {
    List<Card> originalOrder = new ArrayList<>(deck.getCards());
    deck.shuffle();
    assertEquals(originalOrder.size(), deck.getCards().size(), "El tamaño del mazo no debe cambiar al barajar.");
    assertNotEquals(originalOrder, deck.getCards(), "El orden de las cartas debería ser diferente tras barajar.");
}
```

```java
//Mezcla las cartas del mazo de forma aleatoria
public void shuffle() {
    Collections.shuffle(cards);
}
```

3. Test de Caixa blanca: statement coverage.

DeckTest	18,8 %	34	147	181
testConstructor()	0,0 %	0	63	63
testResetDeck()	0,0 %	0	39	39
testDealCard()	0,0 %	0	26	26
testConstructor_tooManyCards()	0,0 %	0	12	12
setUp()	100,0 %	6	0	6
testShuffle()	100,0 %	25	0	25

Funcionalitat: Treure carta de deck

Localització: Deck.java, class: Deck, public dealCard()

Test: DeckTest.java, class: Deck,

public void testDealCard ()

1. Primera versió TDD

```java
@Test
void testDealCard0() {
    Card c = deck.getCards().get(0);
}
```

```java
public Card dealCard() {
    return cards.get(0);
}
```

2. Versió final TDD

```java
@Test
void testDealCard() {
    int initialSize = deck.getCards().size();
    Card dealtCard = deck.dealCard();
    assertEquals(initialSize - 1, deck.getCards().size(), "El tamaño del mazo debe disminuir en 1 tras repartir una carta.");
    assertFalse(deck.getCards().contains(dealtCard), "La carta repartida no debería estar en el mazo.");
}
```

```java
// Reparte una carta, eliminándola del mazo
public Card dealCard() {
    return cards.remove(0);
}
```

3. Test de Caixa blanca: statement coverage.

DeckTest	19,3 %	35	146	181
testConstructor()	0,0 %	0	63	63
testResetDeck()	0,0 %	0	39	39
testShuffle()	0,0 %	0	25	25
testConstructor_tooManyCards()	0,0 %	0	12	12
setUp()	100,0 %	6	0	6
testDealCard()	100,0 %	26	0	26

Funcionalitat: Tornar a crear una deck i barrejarla

Localització: Deck.java, class: Deck, public resetDeck ()

Test: DeckTest.java, class: Deck,

public void testDealCard ()

Primera versió TDD

```
public void resetDeck() {
    cards.clear();
    for (Card.Suit suit : Card.Suit.values()) {
        for (Card.Rank rank : Card.Rank.values()) {
            cards.add(new Card(suit, rank));
        }
    }
}
```

```
@Test
void testResetDeck0() {
    deck.resetDeck();
    List<Card> cards = deck.getCards();
    int index = 0;
    for (Card.Suit suit : Card.Suit.values()) {
        for (Card.Rank rank : Card.Rank.values()) {
            assertEquals(new Card(suit, rank), cards.get(index),
                    "Expected card at position " + index + " is not correct.");
            index++;
        }
    }
}
```

Versió final TDD

```
// Restablece el mazo a su estado original y lo mezcla
public void resetDeck() {
    initializeDeck();
    shuffle();
}
```

```
@Test
void testResetDeck() {
    List<Card> originalOrder = new ArrayList<>(deck.getCards());
    deck.resetDeck();
    assertEquals(52, deck.getCards().size(), "El mazo debe contener 52 cartas tras resetear.");
    assertEquals(originalOrder.size(), deck.getCards().size(), "El tamaño del mazo no debe cambiar al barajar.");
    assertNotEquals(originalOrder, deck.getCards(), "El orden de las cartas debería ser diferente tras barajar.");
}
```

Test de Caixa blanca: statement coverage.

DeckTest	26,5 %	48	133	181
testConstructor()	0,0 %	0	63	63
testDealCard()	0,0 %	0	26	26
testShuffle()	0,0 %	0	25	25
testConstructor_tooManyCards()	0,0 %	0	12	12
setUp()	100,0 %	6	0	6
testResetDeck()	100,0 %	39	0	39

Classe Dealer

Funcionalitat: Constructor sense paràmetres de la classe Dealer

Localització: Dealer.java, class: Dealer, public Dealer()

Test: DealerTest.java, class: Dealer,

void testConstructor_emptyDealer_noExceptionThrown()

Test de Caixa blanca: statement coverage.

DealerTest	3,7 %	14	369	383
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_emptyDealer_noExceptionThrown()	100,0 %	11	0	11

Funcionalitat: Constructor paramètric de la classe Dealer

Localització: Dealer.java, class: Dealer, public Dealer(List<Card> hand)

Test: DealerTest.java, class: Dealer,

Primera versió TDD

```
public Dealer(List<Card> hand) {
    this.hand = new ArrayList<>(hand);
}
```

```
@Test
public void testConstructor0() {
    List<Card> hand = Arrays.asList(
            new Card(Card.Suit.DIAMONDS, Card.Rank.THREE),
            new Card(Card.Suit.HEARTS, Card.Rank.J)
    );

    Dealer dealer = new Dealer(hand);
    assertNotNull(dealer);
}
```

Versió final TDD

```java
// Constructor que valida y establece una mano específica
public Dealer(List<Card> hand) {
    // Verifica que la mano no sea nula ni vacía
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }

    // Verifica que la mano no tenga más de 2 cartas
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }

    // Asigna una nueva lista con las cartas proporcionadas
    this.hand = new ArrayList<>(hand);
}
```

```java
@Test
public void testConstructor_withHand_noExceptionThrown() {
    List<Card> hand = Arrays.asList(
            new Card(Card.Suit.DIAMONDS, Card.Rank.THREE),
            new Card(Card.Suit.HEARTS, Card.Rank.J)
    );

    Dealer dealer = new Dealer(hand);
    assertNotNull(dealer);
    assertEquals(2, dealer.getHand().size());
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.THREE)
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.J)));
}

@Test
public void testConstructor_withHandOversized_ExceptionThrown() {
    List<Card> hand = Arrays.asList(
            new Card(Card.Suit.DIAMONDS, Card.Rank.THREE),
            new Card(Card.Suit.HEARTS, Card.Rank.J),
            new Card(Card.Suit.CLUBS, Card.Rank.K)
    );

    assertThrows(IllegalArgumentException.class, () -> new Dealer(hand));
}
```

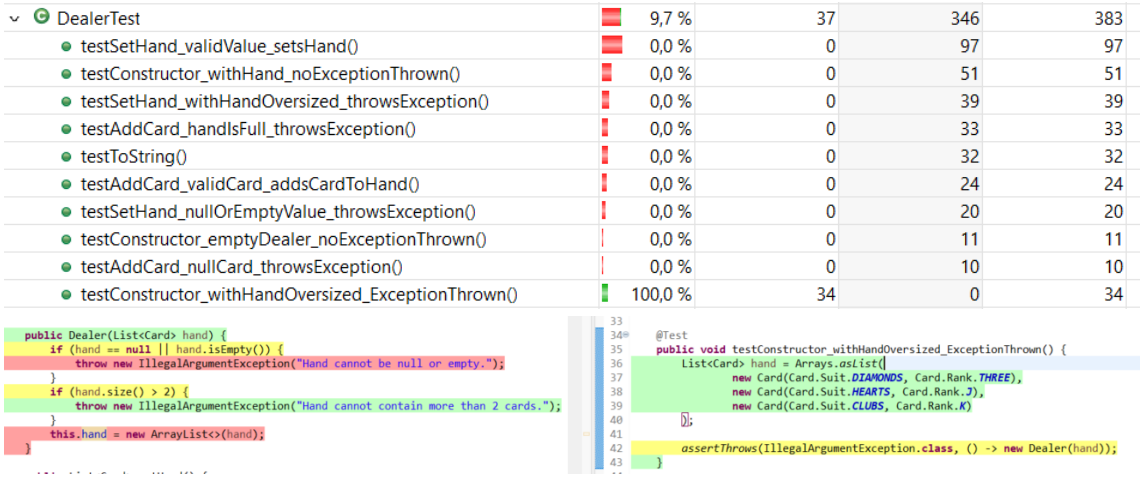
1. void testConstructor_withHand_noExceptionThrown ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	14,1 %	54	329	383
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_withHand_noExceptionThrown()	100,0 %	51	0	51

2. void testConstructor_withHandOversized_ExceptionThrown()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



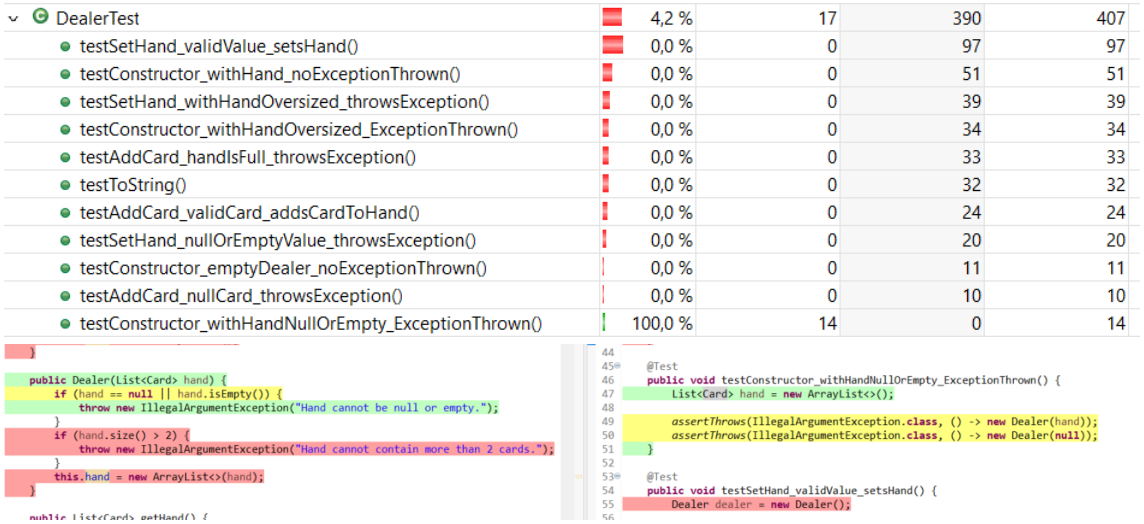
DealerTest	9,7 %	37	346	383
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_withHandOversized_ExceptionThrown()	100,0 %	34	0	34

```
public Dealer(List<Card> hand) {
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }
    this.hand = new ArrayList<>(hand);
}
```

```
@Test
public void testConstructor_withHandOversized_ExceptionThrown() {
    List<Card> hand = Arrays.asList(
            new Card(Card.Suit.DIAMONDS, Card.Rank.THREE),
            new Card(Card.Suit.HEARTS, Card.Rank.J),
            new Card(Card.Suit.CLUBS, Card.Rank.K)
    );

    assertThrows(IllegalArgumentException.class, () -> new Dealer(hand));
}
```

3. void testConstructor_withHandNullOrEmpty_ExceptionThrown()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



DealerTest	4,2 %	17	390	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_withHandNullOrEmpty_ExceptionThrown()	100,0 %	14	0	14

```
}

public Dealer(List<Card> hand) {
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }
    this.hand = new ArrayList<>(hand);
}
```

```
@Test
public void testConstructor_withHandNullOrEmpty_ExceptionThrown() {
    List<Card> hand = new ArrayList<>();

    assertThrows(IllegalArgumentException.class, () -> new Dealer(hand));
    assertThrows(IllegalArgumentException.class, () -> new Dealer(null));
}

@Test
public void testSetHand_validValue_setsHand() {
    Dealer dealer = new Dealer();
```

Funcionalitat: Setter del atribut hand

Localització: Dealer.java, class: Dealer, public void setHand()

Test: DealerTest.java, class: Dealer,

1. void testSetHand_validValue_setsHand()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	24,6 %	100	307	407
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testSetHand_validValue_setsHand()	100,0 %	97	0	97

```
public List<Card> getHand() {
    return hand;
}

public void setHand(List<Card> hand) {
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }
    this.hand = new ArrayList<>(hand);
}

public void addCard(Card card) {
    if (hand.size() >= 2) {
        throw new IllegalArgumentException("Cannot add a card because the hand is full");
    }
    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
}
```

```
@Test
public void testSetHand_validValue_setsHand() {
    Dealer dealer = new Dealer();

    List<Card> initialHand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );
    List<Card> newHand = Arrays.asList(
            new Card(Card.Suit.CLUBS, Card.Rank.Q),
            new Card(Card.Suit.SPADES, Card.Rank.J)
    );

    dealer.setHand(initialHand);
    assertEquals(2, dealer.getHand().size());
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));

    dealer.setHand(newHand);
    assertEquals(2, dealer.getHand().size());
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.CLUBS, Card.Rank.Q)));
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.SPADES, Card.Rank.J)));
}
```

2. void testSetHand_withHandOversized_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	10,3 %	42	365	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testSetHand_withHandOversized_throwsException()	100,0 %	39	0	39

```
public void setHand(List<Card> hand) {
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }
    this.hand = new ArrayList<>(hand);
}

public void addCard(Card card) {
```

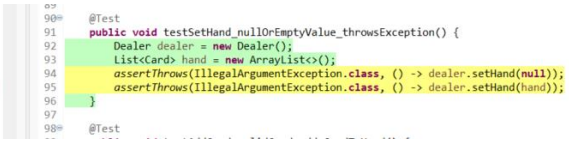
```
@Test
public void testSetHand_withHandOversized_throwsException() {
    Dealer dealer = new Dealer();

    List<Card> hand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K),
        new Card(Card.Suit.CLUBS, Card.Rank.SEVEN)
    );

    assertThrows(IllegalArgumentException.class, () -> dealer.setHand(hand));
}
```

3. void testSetHand_nullOrEmptyValue_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	5,7 %	23	384	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testSetHand_nullOrEmptyValue_throwsException()	100,0 %	20	0	20

```
public void setHand(List<Card> hand) {
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }
    this.hand = new ArrayList<>(hand);
}
```



```
@Test
public void testSetHand_nullOrEmptyValue_throwsException() {
    Dealer dealer = new Dealer();
    List<Card> hand = new ArrayList<>();
    assertThrows(IllegalArgumentException.class, () -> dealer.setHand(null));
    assertThrows(IllegalArgumentException.class, () -> dealer.setHand(hand));
}

@Test
```

Funcionalitat: Afegir carta a dealer

Localització: Dealer.java, class: Dealer, public void addCard(Card card)

Test: DealerTest.java, class: Dealer,

Primera versió TDD

```
public void setHand(List<Card> hand) {
    this.hand = new ArrayList<>(hand);
}
```

```
@Test
public void testSetHand() {
    Dealer dealer = new Dealer();

    List<Card> initialHand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );

    dealer.setHand(initialHand);
}
```

Versió final TDD

```java
// Establece una nueva mano del dealer validando que sea correcta
public void setHand(List<Card> hand) {
    // Verifica que la mano no sea nula ni vacía
    if (hand == null || hand.isEmpty()) {
        throw new IllegalArgumentException("Hand cannot be null or empty.");
    }

    // Verifica que la mano no tenga más de 2 cartas
    if (hand.size() > 2) {
        throw new IllegalArgumentException("Hand cannot contain more than 2 cards.");
    }

    // Asigna una nueva lista con las cartas proporcionadas
    this.hand = new ArrayList<>(hand);
}

@Test
public void testSetHand_validValue_setsHand() {
    Dealer dealer = new Dealer();

    List<Card> initialHand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );
    List<Card> newHand = Arrays.asList(
            new Card(Card.Suit.CLUBS, Card.Rank.Q),
            new Card(Card.Suit.SPADES, Card.Rank.J)
    );

    dealer.setHand(initialHand);
    assertEquals(2, dealer.getHand().size());
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));

    dealer.setHand(newHand);
    assertEquals(2, dealer.getHand().size());
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.CLUBS, Card.Rank.Q)));
    assertTrue(dealer.getHand().contains(new Card(Card.Suit.SPADES, Card.Rank.J)));
}

@Test
public void testSetHand_withHandOversized_throwsException() {
    Dealer dealer = new Dealer();

    List<Card> hand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K),
        new Card(Card.Suit.CLUBS, Card.Rank.SEVEN)
    );

    assertThrows(IllegalArgumentException.class, () -> dealer.setHand(hand));
}
```

1. void testAddCard_validCard_addsCardToHand()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	6,6 %	27	380	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testAddCard_validCard_addsCardToHand()	100,0 %	24	0	24

```
public void addCard(Card card) {
    if (hand.size() >= 2) {
        throw new IllegalArgumentException("Cannot add a card because the hand is full
    }
    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
}

@Override
public String toString() {
```

```
@Test
public void testAddCard_validCard_addsCardToHand() {
    Dealer dealer = new Dealer();
    Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);

    dealer.addCard(card);

    assertEquals(1, dealer.getHand().size());
    assertTrue(dealer.getHand().contains(card));
}
```

2. void testAddCard_nullCard_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	3,2 %	13	394	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	100,0 %	10	0	10

```
public void addCard(Card card) {
    if (hand.size() >= 2) {
        throw new IllegalArgumentException("Cannot add a card because the hand is full
    }
    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
}
```

```
@Test
public void testAddCard_nullCard_throwsException() {
    Dealer dealer = new Dealer();

    assertThrows(IllegalArgumentException.class, () -> dealer.addCard(null));
}

@Test
public void testAddCard_handIsFull_throwsException() {
```

3. void testAddCard_handIsFull_throwsException ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

DealerTest	8,8 %	36	371	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testToString()	0,0 %	0	32	32
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testAddCard_handIsFull_throwsException()	100,0 %	33	0	33

```
public void addCard(Card card) {
    if (hand.size() >= 2) {
        throw new IllegalArgumentException("Cannot add a card because the hand is full
    }
    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
}

@Override
public String toString() {
    return "Dealer{hand=" + hand + "}";
```

```
@Test
public void testAddCard_handIsFull_throwsException() {
    Dealer dealer = new Dealer();
    Card card1 = new Card(Card.Suit.HEARTS, Card.Rank.A);
    Card card2 = new Card(Card.Suit.DIAMONDS, Card.Rank.K);

    dealer.addCard(card1);
    dealer.addCard(card2);

    assertThrows(IllegalArgumentException.class, () -> dealer.addCard(new Card(Card.S
    assertEquals(2, dealer.getHand().size());
}
```

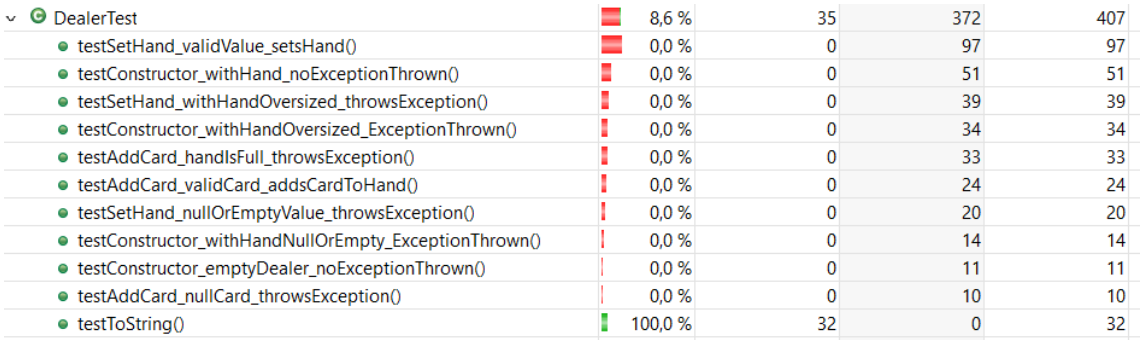
Funcionalitat: Mostrar en forma de text les cartes del dealer

Localització: Dealer.java, class: Dealer, public String toString()

Test: DealerTest.java, class: Dealer,

void testToString()

Test de Caixa blanca: statement coverage.



DealerTest	8,6 %	35	372	407
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withHand_noExceptionThrown()	0,0 %	0	51	51
testSetHand_withHandOversized_throwsException()	0,0 %	0	39	39
testConstructor_withHandOversized_ExceptionThrown()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testConstructor_withHandNullOrEmpty_ExceptionThrown()	0,0 %	0	14	14
testConstructor_emptyDealer_noExceptionThrown()	0,0 %	0	11	11
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testToString()	100,0 %	32	0	32

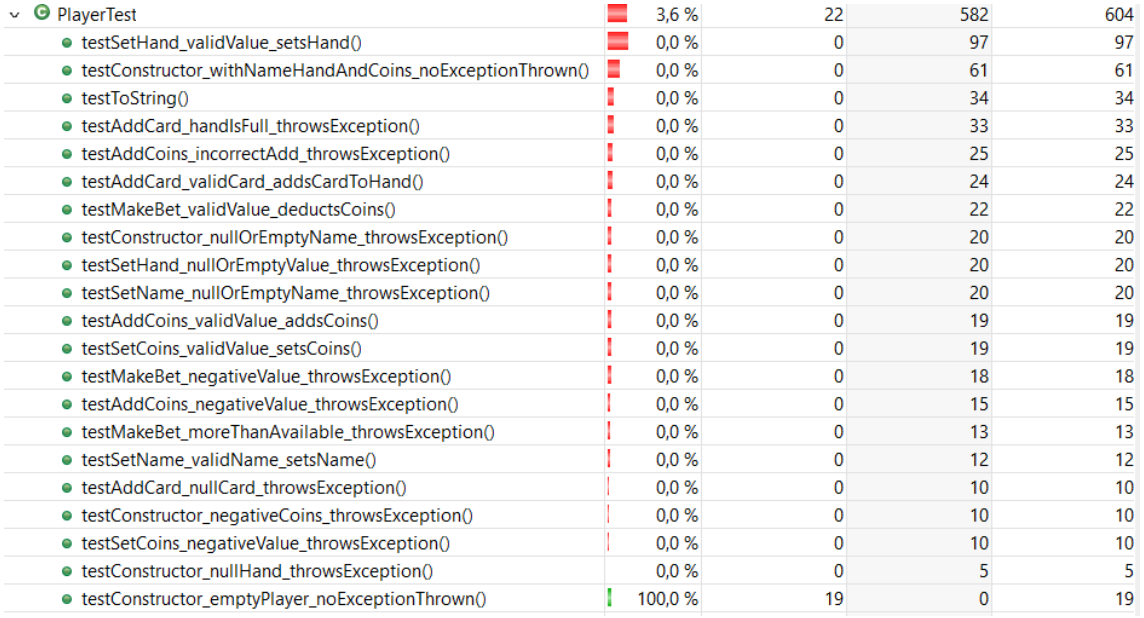
Classe Player

Funcionalitat: Constructor de la classe player

Localització: Player.java, class: Player, public Player ()

Test: PlayerTest.java, class: Player,

Test de Caixa blanca: statement coverage.



PlayerTest	3,6 %	22	582	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testConstructor_emptyPlayer_noExceptionThrown()	100,0 %	19	0	19

Funcionalitat: Constructor paramètric de la classe player

Localització: Player.java, class: Player, public Player(String name, List<Card> hand, **int** coins)

Test: PlayerTest.java, class: Player,

Primera versió TDD

```java
public Player(String name, List<Card> hand, int coins) {
    this.name = name;
    this.hand = hand;
    this.coins = coins;
}
```

```java
@Test
public void constructorTest() {
    Player player = new Player();
    assertNotNull(player);
    assertEquals(0, player.getCoins());
    assertEquals("Uknown Player", player.getName());
}
```

Versió final TDD

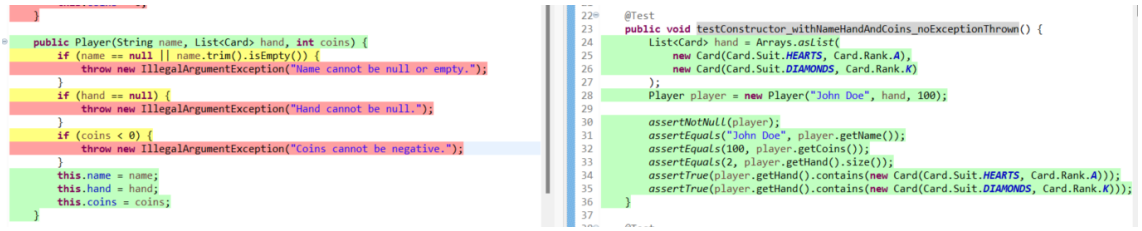
```java
// Constructor que inicializa el jugador con un nombre, mano y monedas
public Player(String name, List<Card> hand, int coins) {
    if (name == null || name.trim().isEmpty()) {
        throw new IllegalArgumentException("Name cannot be null or empty.");
    }
    if (hand == null) {
        throw new IllegalArgumentException("Hand cannot be null.");
    }
    if (coins < 0) {
        throw new IllegalArgumentException("Coins cannot be negative.");
    }
    this.name = name;
    this.hand = hand;
    this.coins = coins;
}
```

```java
@Test
public void testConstructor_emptyPlayer_noExceptionThrown() {
    Player player = new Player();
    assertNotNull(player);
    assertEquals(0, player.getCoins());
    assertTrue(player.getHand().isEmpty());
    assertEquals("Uknown Player", player.getName());
}

@Test
public void testConstructor_withNameHandAndCoins_noExceptionThrown() {
    List<Card> hand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );
    Player player = new Player("John Doe", hand, 100);

    assertNotNull(player);
    assertEquals("John Doe", player.getName());
    assertEquals(100, player.getCoins());
    assertEquals(2, player.getHand().size());
    assertTrue(player.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
    assertTrue(player.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));
}

@Test
public void testConstructor_nullOrEmptyName_throwsException() {
    List<Card> hand = new ArrayList<>();
    assertThrows(IllegalArgumentException.class, () -> new Player(null, hand, 100));
    assertThrows(IllegalArgumentException.class, () -> new Player("", hand, 100));
    assertThrows(IllegalArgumentException.class, () -> new Player("   ", hand, 100));
}
```

1. void testConstructor_withNameHandAndCoins_noExceptionThrown()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	10,6 %	64	540	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testConstructor_withNameHandAndCoins_noExceptionThrown()	100,0 %	61	0	61



```
    }

    public Player(String name, List<Card> hand, int coins) {
        if (name == null || name.trim().isEmpty()) {
            throw new IllegalArgumentException("Name cannot be null or empty.");
        }
        if (hand == null) {
            throw new IllegalArgumentException("Hand cannot be null.");
        }
        if (coins < 0) {
            throw new IllegalArgumentException("Coins cannot be negative.");
        }
        this.name = name;
        this.hand = hand;
        this.coins = coins;
    }
```

```
@Test
public void testConstructor_withNameHandAndCoins_noExceptionThrown() {
    List<Card> hand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );
    Player player = new Player("John Doe", hand, 100);

    assertNotNull(player);
    assertEquals("John Doe", player.getName());
    assertEquals(100, player.getCoins());
    assertEquals(2, player.getHand().size());
    assertTrue(player.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
    assertTrue(player.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));
}
```

2. void testConstructor_nullOrEmptyName_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	3,8 %	23	581	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testConstructor_nullOrEmptyName_throwsException()	100,0 %	20	0	20

```
    public Player(String name, List<Card> hand, int coins) {
        if (name == null || name.trim().isEmpty()) {
            throw new IllegalArgumentException("Name cannot be null or empty.");
        }
        if (hand == null) {
            throw new IllegalArgumentException("Hand cannot be null.");
        }
        if (coins < 0) {
            throw new IllegalArgumentException("Coins cannot be negative.");
        }
        this.name = name;
        this.hand = hand;
        this.coins = coins;
    }
```

```
    }

@Test
public void testConstructor_nullOrEmptyName_throwsException() {
    List<Card> hand = new ArrayList<>();
    assertThrows(IllegalArgumentException.class, () -> new Player(null, hand, 100));
    assertThrows(IllegalArgumentException.class, () -> new Player("", hand, 100));
    assertThrows(IllegalArgumentException.class, () -> new Player("   ", hand, 100));
}

@Test
public void testConstructor_negativeCoins_throwsException() {
    List<Card> hand = new ArrayList<>();
    assertThrows(IllegalArgumentException.class, () -> new Player("Jhon", hand, -10))
}
```

3. void testConstructor_negativeCoins_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	2,2 %	13	591	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testConstructor_negativeCoins_throwsException()	100,0 %	10	0	10

4. void testConstructor_nullHand_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	1,3 %	8	596	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	100,0 %	5	0	5

Funcionalitat: Editar atribut name del player

Localització: Player.java, class: Player, public void setName(String name)

Test: PlayerTest.java, class: Player,

Primera versió TDD

```
@Test
public void testSetName() {
    Player player = new Player();
    player.setName("Alice");
    assertEquals("Alice", player.getName());
}
```

```
public void setName(String name) {
    this.name = name;
}
```

Versió final TDD

```java
@Test
public void testSetName_validName_setsName() {
    Player player = new Player();
    player.setName("Alice");
    assertEquals("Alice", player.getName());
}

@Test
public void testSetName_nullOrEmptyName_throwsException() {
    Player player = new Player();
    assertThrows(IllegalArgumentException.class, () -> player.setName(null));
    assertThrows(IllegalArgumentException.class, () -> player.setName(""));
    assertThrows(IllegalArgumentException.class, () -> player.setName("    "));
}
```

```java
public void setName(String name) {
    if (name == null || name.trim().isEmpty()) {
        throw new IllegalArgumentException("Name cannot be null or empty.");
    }
    this.name = name;
}
```

1. void testSetName_validName_setsName()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

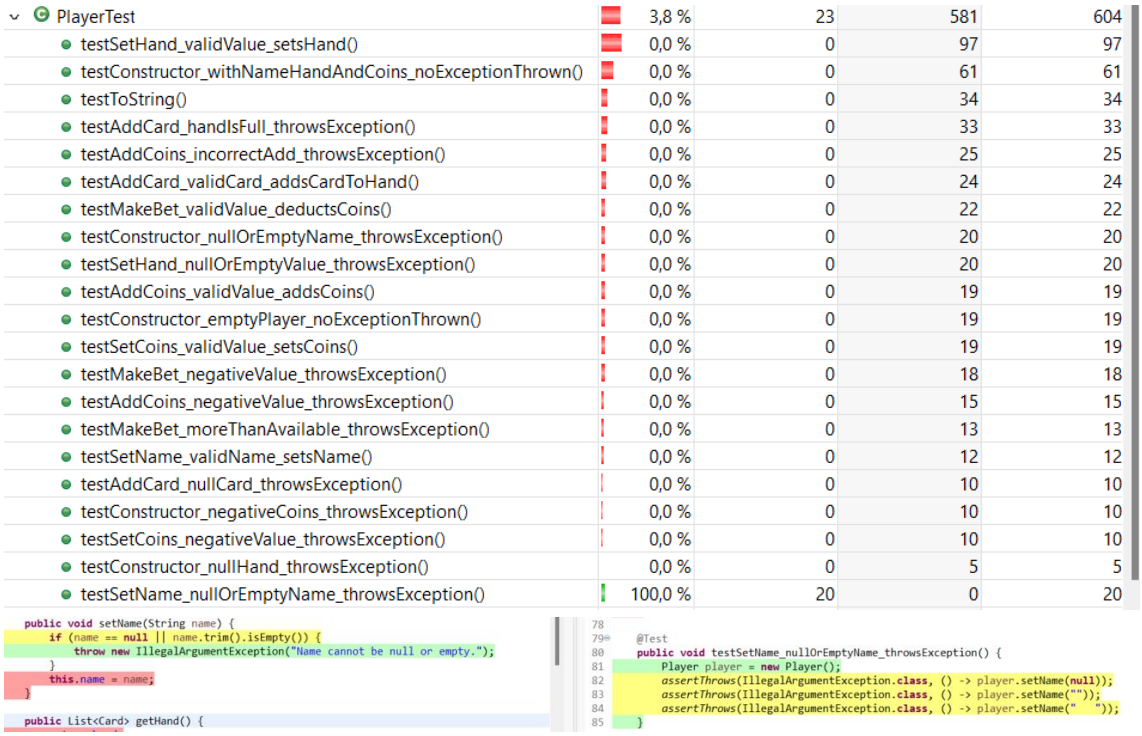
PlayerTest	2,5 %	15	589	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testSetName_validName_setsName()	100,0 %	12	0	12

```java
public void setName(String name) {
    if (name == null || name.trim().isEmpty()) {
        throw new IllegalArgumentException("Name cannot be null or empty.");
    }
    this.name = name;
}
```

```java
@Test
public void testSetName_validName_setsName() {
    Player player = new Player();
    player.setName("Alice");
    assertEquals("Alice", player.getName());
}
```

2. void testSetName_nullOrEmptyName_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



PlayerTest	3,8 %	23	581	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testSetName_nullOrEmptyName_throwsException()	100,0 %	20	0	20

```
public void setName(String name) {
    if (name == null || name.trim().isEmpty()) {
        throw new IllegalArgumentException("Name cannot be null or empty.");
    }
    this.name = name;
}

public List<Card> getHand() {
```

```
78
79  @Test
80  public void testSetName_nullOrEmptyName_throwsException() {
81      Player player = new Player();
82      assertThrows(IllegalArgumentException.class, () -> player.setName(null));
83      assertThrows(IllegalArgumentException.class, () -> player.setName(""));
84      assertThrows(IllegalArgumentException.class, () -> player.setName("   "));
85  }
```

Funcionalitat: Editar atribut coins del player

Localització: Player.java, class: Player, public void setCoins(int coins)

Test: PlayerTest.java, class: Player,

Primera versió TDD

```
@Test
public void setCoinsTest() {
    Player player = new Player();
    player.setCoins(200);
    assertEquals(200, player.getCoins());
}
```

```
public void setCoins(int coins) {
    this.coins = coins;
}
```

Versió final TDD

```java
@Test
public void testSetCoins_validValue_setsCoins() {
    Player player = new Player();
    player.setCoins(200);
    assertEquals(200, player.getCoins());
    player.setCoins(0);
    assertEquals(0, player.getCoins());
}

@Test
public void testSetCoins_negativeValue_throwsException() {
    Player player = new Player();
    assertThrows(IllegalArgumentException.class, () -> player.setCoins(-50));
}
```

```java
public void setCoins(int coins) {
    if (coins < 0) {
        throw new IllegalArgumentException("Coins cannot be negative.");
    }
    this.coins = coins;
}
```

1. void testSetCoins_validValue_setsCoins ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	3,6 %	22	582	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testSetCoins_validValue_setsCoins()	100,0 %	19	0	19

```java
    return coins;
}

public void setCoins(int coins) {
    if (coins < 0) {
        throw new IllegalArgumentException("Coins cannot be negative.");
    }
    this.coins = coins;
}
```

```java
@Test
public void testSetCoins_validValue_setsCoins() {
    Player player = new Player();
    player.setCoins(200);
    assertEquals(200, player.getCoins());
    player.setCoins(0);
    assertEquals(0, player.getCoins());
}
```

2. void testSetCoins_negativeValue_throwsException ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	2,2 %	13	591	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testSetCoins_negativeValue_throwsException()	100,0 %	10	0	10

```
public void setCoins(int coins) {
    if (coins < 0) {
        throw new IllegalArgumentException("Coins cannot be negative.");
    }
    this.coins = coins;
}
```

```
@Test
public void testSetCoins_negativeValue_throwsException() {
    Player player = new Player();
    assertThrows(IllegalArgumentException.class, () -> player.setCoins(-50));
}
```

Funcionalitat: Editar atribut hand del player

Localització: Player.java, class: Player, public void setHand(List<Card> hand)

Test: PlayerTest.java, class: Player,

Primera versió TDD

```
public void setHand(List<Card> hand) {
    this.hand = hand;
}
```

```
@Test
public void setHandTestCorrect() {
    Player player = new Player();

    List<Card> initialHand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );
    player.setHand(initialHand);
    assertEquals(2, player.getHand().size());
    assertTrue(player.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
    assertTrue(player.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));
}
```

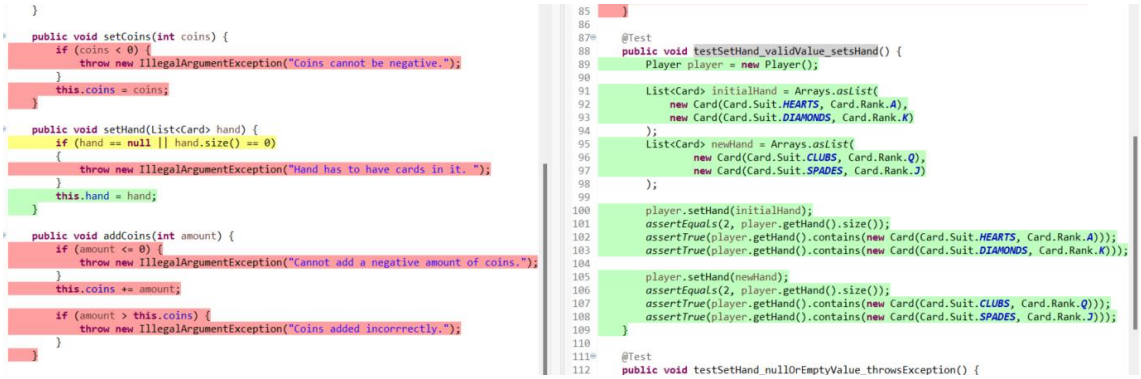
Versió final TDD

```java
// Establece la mano del jugador
public void setHand(List<Card> hand) {
    if (hand == null || hand.size() == 0)
    {
        throw new IllegalArgumentException("Hand has to have cards in it. ");
    }
    this.hand = hand;
}
```

```java
@Test
public void testSetHand_validValue_setsHand() {
    Player player = new Player();

    List<Card> initialHand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.A),
        new Card(Card.Suit.DIAMONDS, Card.Rank.K)
    );
    List<Card> newHand = Arrays.asList(
            new Card(Card.Suit.CLUBS, Card.Rank.Q),
            new Card(Card.Suit.SPADES, Card.Rank.J)
    );

    player.setHand(initialHand);
    assertEquals(2, player.getHand().size());
    assertTrue(player.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
    assertTrue(player.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));

    player.setHand(newHand);
    assertEquals(2, player.getHand().size());
    assertTrue(player.getHand().contains(new Card(Card.Suit.CLUBS, Card.Rank.Q)));
    assertTrue(player.getHand().contains(new Card(Card.Suit.SPADES, Card.Rank.J)));
}

@Test
public void testSetHand_nullOrEmptyValue_throwsException() {
    Player player = new Player();
    List<Card> hand = new ArrayList<>();
    assertThrows(IllegalArgumentException.class, () -> player.setHand(null));
    assertThrows(IllegalArgumentException.class, () -> player.setHand(hand));

}
```

1. void testSetHand_validValue_setsHand ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



```java
    }

    public void setCoins(int coins) {
        if (coins < 0) {
            throw new IllegalArgumentException("Coins cannot be negative.");
        }
        this.coins = coins;
    }

    public void setHand(List<Card> hand) {
        if (hand == null || hand.size() == 0)
        {
            throw new IllegalArgumentException("Hand has to have cards in it. ");
        }
        this.hand = hand;
    }

    public void addCoins(int amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Cannot add a negative amount of coins.");
        }
        this.coins += amount;

        if (amount > this.coins) {
            throw new IllegalArgumentException("Coins added incorrrectly.");
        }
    }
```

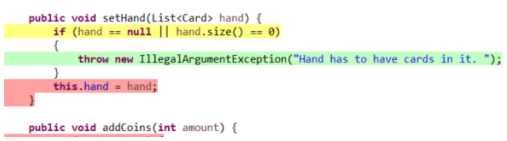
```java
    }

    @Test
    public void testSetHand_validValue_setsHand() {
        Player player = new Player();

        List<Card> initialHand = Arrays.asList(
            new Card(Card.Suit.HEARTS, Card.Rank.A),
            new Card(Card.Suit.DIAMONDS, Card.Rank.K)
        );
        List<Card> newHand = Arrays.asList(
                new Card(Card.Suit.CLUBS, Card.Rank.Q),
                new Card(Card.Suit.SPADES, Card.Rank.J)
        );

        player.setHand(initialHand);
        assertEquals(2, player.getHand().size());
        assertTrue(player.getHand().contains(new Card(Card.Suit.HEARTS, Card.Rank.A)));
        assertTrue(player.getHand().contains(new Card(Card.Suit.DIAMONDS, Card.Rank.K)));

        player.setHand(newHand);
        assertEquals(2, player.getHand().size());
        assertTrue(player.getHand().contains(new Card(Card.Suit.CLUBS, Card.Rank.Q)));
        assertTrue(player.getHand().contains(new Card(Card.Suit.SPADES, Card.Rank.J)));
    }

    @Test
    public void testSetHand_nullOrEmptyValue_throwsException() {
```

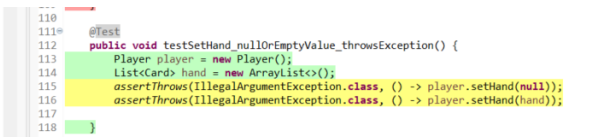
PlayerTest	16,6 %	100	504	604
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testSetHand_validValue_setsHand()	100,0 %	97	0	97

2. void testSetHand_nullOrEmptyValue_throwsException

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	3,8 %	23	581	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testSetHand_nullOrEmptyValue_throwsException()	100,0 %	20	0	20



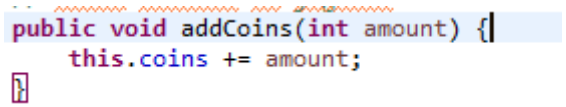


Funcionalitat: Afegir monedes al atribut coins

Localització: Player.java, class: Player, public void addCoins(int amount)

Test: PlayerTest.java, class: Player,

Primera versió TDD



```java
@Test
public void testAddCoins() {
    Player player = new Player();
    player.addCoins(50);
    assertEquals(50, player.getCoins());
}
```

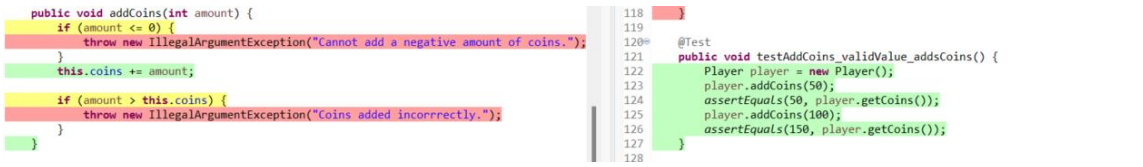
Versió Final TDD

```java
// Añade monedas al jugador
public void addCoins(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot add a negative amount of coins.");
    }
    this.coins += amount;

    if (amount > this.coins) {
        throw new IllegalArgumentException("Coins added incorrrectly.");
    }
}

@Test
public void testAddCoins_validValue_addsCoins() {
    Player player = new Player();
    player.addCoins(50);
    assertEquals(50, player.getCoins());
    player.addCoins(100);
    assertEquals(150, player.getCoins());
}

@Test
public void testAddCoins_negativeValue_throwsException() {
    Player player = new Player();
    assertThrows(IllegalArgumentException.class, () -> player.addCoins(-30));
    assertThrows(IllegalArgumentException.class, () -> player.addCoins(0));
}

@Test
public void testAddCoins_incorrectAdd_throwsException() {
    Player player = new Player("John", new ArrayList<>(), 10);
    int amountToAdd = 20;
    assertDoesNotThrow(() -> player.addCoins(amountToAdd));
    assertTrue(player.getCoins() > 20, "Coins should be correctly added.");
}
```

1. void testAddCoins_validValue_addsCoins()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	3,6 %	22	582	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testAddCoins_validValue_addsCoins()	100,0 %	19	0	19



```
public void addCoins(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot add a negative amount of coins.");
    }
    this.coins += amount;

    if (amount > this.coins) {
        throw new IllegalArgumentException("Coins added incorrrectly.");
    }
}
```

```
}

@Test
public void testAddCoins_validValue_addsCoins() {
    Player player = new Player();
    player.addCoins(50);
    assertEquals(50, player.getCoins());
    player.addCoins(100);
    assertEquals(150, player.getCoins());
}
```

2. void testAddCoins_negativeValue_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	3,0 %	18	586	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testAddCoins_negativeValue_throwsException()	100,0 %	15	0	15

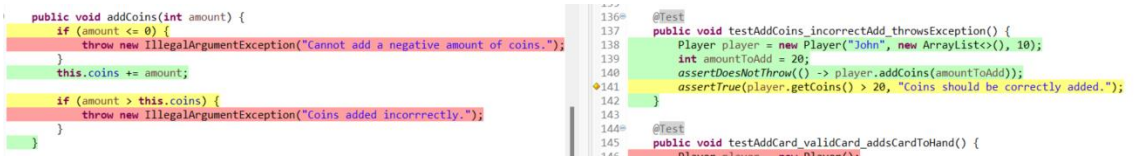
```
public void addCoins(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot add a negative amount of coins.");
    }
    this.coins += amount;

    if (amount > this.coins) {
        throw new IllegalArgumentException("Coins added incorrrectly.");
    }
}
```

```
}

@Test
public void testAddCoins_negativeValue_throwsException() {
    Player player = new Player();
    assertThrows(IllegalArgumentException.class, () -> player.addCoins(-30));
    assertThrows(IllegalArgumentException.class, () -> player.addCoins(0));
}

@Test
```

3. void testAddCoins_incorrectAdd_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	5,1 %	31	573	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testAddCoins_incorrectAdd_throwsException()	96,0 %	24	1	25



```
public void addCoins(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot add a negative amount of coins.");
    }
    this.coins += amount;

    if (amount > this.coins) {
        throw new IllegalArgumentException("Coins added incorrectly.");
    }
}
```

```
@Test
public void testAddCoins_incorrectAdd_throwsException() {
    Player player = new Player("John", new ArrayList<>(), 10);
    int amountToAdd = 20;
    assertDoesNotThrow(() -> player.addCoins(amountToAdd));
    assertTrue(player.getCoins() > 20, "Coins should be correctly added.");
}

@Test
public void testAddCard_validCard_addsCardToHand() {
```

Funcionalitat: Afegir cartes al atribut hand

Localització: Player.java, class: Player, public void addCard(Card card)

Test: PlayerTest.java, class: Player,

Primera versió TDD:

```
public void addCard(Card card) {
    hand.add(card);
}
```

```
@Test
public void testAddCard() {
    Player player = new Player();
    Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);

    player.addCard(card);

    assertEquals(1, player.getHand().size());
    assertTrue(player.getHand().contains(card));
}
```

Versió final TDD:

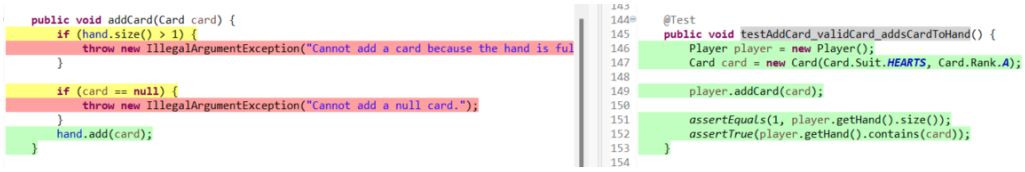
```java
// Añade una carta a la mano del jugador con validación de espacio
public void addCard(Card card) {
    if (hand.size() > 1) {
        throw new IllegalArgumentException("Cannot add a card because the hand is full.");
    }

    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
}
```

```java
@Test
public void testAddCard_validCard_addsCardToHand() {
    Player player = new Player();
    Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);

    player.addCard(card);

    assertEquals(1, player.getHand().size());
    assertTrue(player.getHand().contains(card));
}

@Test
public void testAddCard_nullCard_throwsException() {
    Player player = new Player();

    assertThrows(IllegalArgumentException.class, () -> player.addCard(null));
}

@Test
public void testAddCard_handIsFull_throwsException() {
    Player player = new Player();
    Card card1 = new Card(Card.Suit.HEARTS, Card.Rank.A);
    Card card2 = new Card(Card.Suit.DIAMONDS, Card.Rank.K);

    player.addCard(card1);
    player.addCard(card2);

    assertThrows(IllegalArgumentException.class, () -> player.addCard(new Card(Card.Suit.SPADES, Card.Rank.Q)));
    assertEquals(2, player.getHand().size());
}
```

1. void testAddCard_validCard_addsCardToHand ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	4,5 %	27	577	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testAddCard_validCard_addsCardToHand()	100,0 %	24	0	24



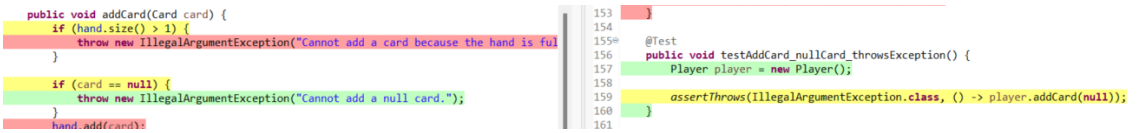
```
public void addCard(Card card) {
    if (hand.size() > 1) {
        throw new IllegalArgumentException("Cannot add a card because the hand is ful
    }

    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
}
```

```
@Test
public void testAddCard_validCard_addsCardToHand() {
    Player player = new Player();
    Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);

    player.addCard(card);

    assertEquals(1, player.getHand().size());
    assertTrue(player.getHand().contains(card));
}
```

2. void testAddCard_nullCard_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	2,2 %	13	591	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testAddCard_nullCard_throwsException()	100,0 %	10	0	10



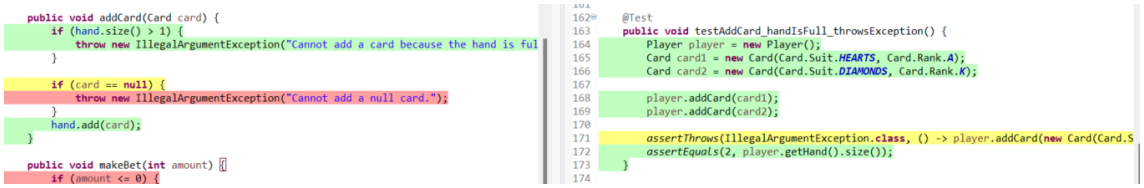
```
public void addCard(Card card) {
    if (hand.size() > 1) {
        throw new IllegalArgumentException("Cannot add a card because the hand is ful
    }

    if (card == null) {
        throw new IllegalArgumentException("Cannot add a null card.");
    }
    hand.add(card);
```

```
}

@Test
public void testAddCard_nullCard_throwsException() {
    Player player = new Player();

    assertThrows(IllegalArgumentException.class, () -> player.addCard(null));
}
```

3. void testAddCard_handIsFull_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

PlayerTest	6,0 %	36	568	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testAddCard_handIsFull_throwsException()	100,0 %	33	0	33



Funcionalitat: Apostar diners en la partida cosa que fa restar les coins del player

Localització: Player.java, class: Player, public void makeBet(int amount)

Test: PlayerTest.java, class: Player,

Primera versió TDD

```
public void makeBet(int amount) {
    this.coins -= amount;
}
```

```
@Test
public void testMakeBet() {
    Player player = new Player();
    player.setCoins(100);
    player.makeBet(40);
    assertEquals(60, player.getCoins());
}
```

Versió final TDD

```
@Test
public void testMakeBet_validValue_deductsCoins() {
    Player player = new Player();
    player.setCoins(100);
    player.makeBet(40);
    assertEquals(60, player.getCoins());
    player.makeBet(60);
    assertEquals(0, player.getCoins());
}
```

```
@Test
public void testMakeBet_moreThanAvailable_throwsException() {
    Player player = new Player();
    player.setCoins(50);
    assertThrows(IllegalArgumentException.class, () -> player.makeBet(60));
}
```

```
@Test
public void testMakeBet_negativeValue_throwsException() {
    Player player = new Player();
    player.setCoins(50);
    assertThrows(IllegalArgumentException.class, () -> player.makeBet(-10));
    assertThrows(IllegalArgumentException.class, () -> player.makeBet(0));
}
```

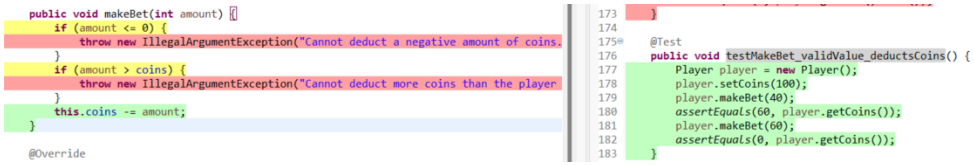
```java
public void makeBet(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot deduct a negative amount of coins.");
    }
    if (amount > coins) {
        throw new IllegalArgumentException("Cannot deduct more coins than the player has.");
    }
    this.coins -= amount;
}
```

1. void testMakeBet_validValue_deductsCoins()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

Test de Caixa negra: particions equivalents i valors límit.

PlayerTest	4,1 %	25	579	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testMakeBet_validValue_deductsCoins()	100,0 %	22	0	22



2. void testMakeBet_moreThanAvailable_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

Test de Caixa negra: valors frontera.

PlayerTest	2,6 %	16	588	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testMakeBet_moreThanAvailable_throwsException()	100,0 %	13	0	13

```
public void makeBet(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot deduct a negative amount of coins.
    }
    if (amount > coins) {
        throw new IllegalArgumentException("Cannot deduct more coins than the player
    }
    this.coins -= amount;
}
```

```
@Test
public void testMakeBet_moreThanAvailable_throwsException() {
    Player player = new Player();
    player.setCoins(50);
    assertThrows(IllegalArgumentException.class, () -> player.makeBet(60));
}

@Test
```

3. void testMakeBet_negativeValue_throwsException()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

Test de Caixa negra: valors frontera.

PlayerTest	3,5 %	21	583	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testToString()	0,0 %	0	34	34
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testMakeBet_negativeValue_throwsException()	100,0 %	18	0	18

```
public void makeBet(int amount) {
    if (amount <= 0) {
        throw new IllegalArgumentException("Cannot deduct a negative amount of coins.
    }
    if (amount > coins) {
        throw new IllegalArgumentException("Cannot deduct more coins than the player
    }
    this.coins -= amount;
}
```

```
public void testMakeBet_negativeValue_throwsException() {
    Player player = new Player();
    player.setCoins(50);
    assertThrows(IllegalArgumentException.class, () -> player.makeBet(-10));
    assertThrows(IllegalArgumentException.class, () -> player.makeBet(0));
}

@Test
public void testToString() {
```

Funcionalitat: Mostrar en forma de text els atributs de la calse player

Localització: Player.java, class: Player, public String toString()

Test: PlayerTest.java, class: Player,

void testToString()

Test de Caixa blanca: statement coverage.

PlayerTest	6,1 %	37	567	604
testSetHand_validValue_setsHand()	0,0 %	0	97	97
testConstructor_withNameHandAndCoins_noExceptionThrown()	0,0 %	0	61	61
testAddCard_handIsFull_throwsException()	0,0 %	0	33	33
testAddCoins_incorrectAdd_throwsException()	0,0 %	0	25	25
testAddCard_validCard_addsCardToHand()	0,0 %	0	24	24
testMakeBet_validValue_deductsCoins()	0,0 %	0	22	22
testConstructor_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testSetHand_nullOrEmptyValue_throwsException()	0,0 %	0	20	20
testSetName_nullOrEmptyName_throwsException()	0,0 %	0	20	20
testAddCoins_validValue_addsCoins()	0,0 %	0	19	19
testConstructor_emptyPlayer_noExceptionThrown()	0,0 %	0	19	19
testSetCoins_validValue_setsCoins()	0,0 %	0	19	19
testMakeBet_negativeValue_throwsException()	0,0 %	0	18	18
testAddCoins_negativeValue_throwsException()	0,0 %	0	15	15
testMakeBet_moreThanAvailable_throwsException()	0,0 %	0	13	13
testSetName_validName_setsName()	0,0 %	0	12	12
testAddCard_nullCard_throwsException()	0,0 %	0	10	10
testConstructor_negativeCoins_throwsException()	0,0 %	0	10	10
testSetCoins_negativeValue_throwsException()	0,0 %	0	10	10
testConstructor_nullHand_throwsException()	0,0 %	0	5	5
testToString()	100,0 %	34	0	34

Classe Table

Funcionalitat: Constructor de la classe table

Localització: Table.java, class: Table, public Table ()

Test: TableTest.java, class: Table,

void testTableConstructor()

Test de Caixa blanca: statement coverage.

TableTest	19,3 %	27	113	140
testResetTable()	0,0 %	0	32	32
testAddCommunityCard_correctlyAddsCard()	0,0 %	0	27	27
testAddCommunityCard()	0,0 %	0	26	26
testSetAndGetAnteBet()	0,0 %	0	12	12
testSetAndGetCallBet()	0,0 %	0	12	12
testTableConstructor()	100,0 %	24	0	24

Funcionalitat: Afegir cartes al atribut communityCards que es la taula compartida on juguen els jugadors

Localització: Table.java, class: Table, public void addCommunityCard(Card card)

Test: TableTest.java, class: Table,

1. void testAddCommunityCard()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

TableTest	20,7 %	29	111	140
testResetTable()	0,0 %	0	32	32
testAddCommunityCard_correctlyAddsCard()	0,0 %	0	27	27
testTableConstructor()	0,0 %	0	24	24
testSetAndGetAnteBet()	0,0 %	0	12	12
testSetAndGetCallBet()	0,0 %	0	12	12
testAddCommunityCard()	100,0 %	26	0	26

```
        this.callBet = 0;
    }

    public void addCommunityCard(Card card) {
        int quantity_communty_cards_before = communityCards.size();
        communityCards.add(card);
        if (quantity_communty_cards_before >= communityCards.size()) {
            throw new IllegalArgumentException("Card added incorrrectly.");
        }
    }
```

```
    @Test
    void testAddCommunityCard() {
        Table table = new Table();
        Card card = new Card(Card.Suit.DIAMONDS, Card.Rank.A);
        table.addCommunityCard(card);
        List<Card> communityCards = table.getCommunityCards();

        assertEquals(1, communityCards.size());
        assertEquals(card, communityCards.get(0));
    }
```

2. void testAddCommunityCard_correctlyAddsCard()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

TableTest	24,3 %	34	106	140
testResetTable()	0,0 %	0	32	32
testAddCommunityCard()	0,0 %	0	26	26
testTableConstructor()	0,0 %	0	24	24
testSetAndGetAnteBet()	0,0 %	0	12	12
testSetAndGetCallBet()	0,0 %	0	12	12
testAddCommunityCard_correctlyAddsCard()	100,0 %	27	0	27

```
public void addCommunityCard(Card card) {
    int quantity_communty_cards_before = communityCards.size();
    communityCards.add(card);
    if (quantity_communty_cards_before >= communityCards.size()) {
        throw new IllegalArgumentException("Card added incorrrectly.");
    }
}

public List<Card> getCommunityCards() {
    return communityCards;
}
```

```
        assertEquals(card, communityCards.get(0));
    }

    @Test
    public void testAddCommunityCard_correctlyAddsCard() {
        Table table = new Table();
        Card card = new Card(Card.Suit.HEARTS, Card.Rank.A);
        int quantityBefore = table.getCommunityCards().size();

        assertDoesNotThrow(() -> table.addCommunityCard(card));
        assertEquals(quantityBefore + 1, table.getCommunityCards().size(), "Card added inc
    }
```

Funcionalitat: Setter i getter del atribut anteBet

Localització: Table.java, class: Table, public void setAnteBet(int amount)

public int getAnteBet()

Test: TableTest.java, class: Table,

Void testSetAndGetAnteBet()

Test de Caixa blanca: statement coverage.

TableTest	10,7 %	15	125	140
testResetTable()	0,0 %	0	32	32
testAddCommunityCard_correctlyAddsCard()	0,0 %	0	27	27
testAddCommunityCard()	0,0 %	0	26	26
testTableConstructor()	0,0 %	0	24	24
testSetAndGetCallBet()	0,0 %	0	12	12
testSetAndGetAnteBet()	100,0 %	12	0	12

Funcionalitat: Setter i getter del atribut callBet

Localització: Table.java, class: Table, public void setCallBet(int amount)

public int getCallBet()

Test: TableTest.java, class: Table,

Void testSetAndGetCallBet ()

Test de Caixa blanca: statement coverage.

TableTest	10,7 %	15	125	140
testResetTable()	0,0 %	0	32	32
testAddCommunityCard_correctlyAddsCard()	0,0 %	0	27	27
testAddCommunityCard()	0,0 %	0	26	26
testTableConstructor()	0,0 %	0	24	24
testSetAndGetAnteBet()	0,0 %	0	12	12
testSetAndGetCallBet()	100,0 %	12	0	12

Funcionalitat: Reiniciar els atributs de table deixant-los buits o a zero

Localització: Table.java, class: Table, public void resetTable ()

Test: TableTest.java, class: Table,

Void testResetTable()

Test de Caixa blanca: statement coverage.

TableTest	25,0 %	35	105	140
testAddCommunityCard_correctlyAddsCard()	0,0 %	0	27	27
testAddCommunityCard()	0,0 %	0	26	26
testTableConstructor()	0,0 %	0	24	24
testSetAndGetAnteBet()	0,0 %	0	12	12
testSetAndGetCallBet()	0,0 %	0	12	12
testResetTable()	100,0 %	32	0	32

Classe Game

Funcionalitat: Constructor paramètric de la classe game

Localització: Game.java, class: Game, public Game (int playerChips)

Test: GameTest.java, class: Game,

void testGameConstructorInitializationAndGetters()

Test de Caixa blanca: statement coverage.

GameTest	1,8 %	31	1.706	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57
testPlayerCalls()	0,0 %	0	37	37
testOnePair()	0,0 %	0	57	57
testNonConsecutiveRanks()	0,0 %	0	40	40
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testHighCard()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	100,0 %	21	0	21
testFullHouse()	0,0 %	0	57	57

Funcionalitat: Calcula el valor de les teves cartes en la ronda clasifican-les per a poder saber quina de les mans dels jugadors es millor

Localització: Game.java, class: Game, public static HandRank classifyHand(List<Card> hand)

Test: GameTest.java, class: Game,

1. void testRoyalFlush()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	100,0 %	57	0	57

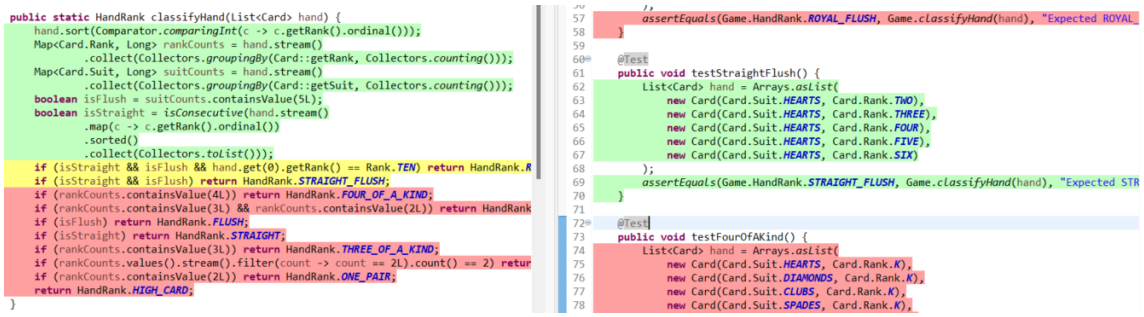
```
public static HandRank classifyHand(List<Card> hand) {
    hand.sort(Comparator.comparingInt(c -> c.getRank().ordinal()));
    Map<Card.Rank, Long> rankCounts = hand.stream()
            .collect(Collectors.groupingBy(Card::getRank, Collectors.counting()));
    Map<Card.Suit, Long> suitCounts = hand.stream()
            .collect(Collectors.groupingBy(Card::getSuit, Collectors.counting()));
    boolean isFlush = suitCounts.containsValue(5L);
    boolean isStraight = isConsecutive(hand.stream()
            .map(c -> c.getRank().ordinal())
            .sorted()
            .collect(Collectors.toList()));
    if (isStraight && isFlush && hand.get(0).getRank() == Rank.TEN) return HandRank.R
    if (isStraight && isFlush) return HandRank.STRAIGHT_FLUSH;
    if (rankCounts.containsValue(4L)) return HandRank.FOUR_OF_A_KIND;
    if (rankCounts.containsValue(3L) && rankCounts.containsValue(2L)) return HandRank
    if (isFlush) return HandRank.FLUSH;
    if (isStraight) return HandRank.STRAIGHT;
    if (rankCounts.containsValue(3L)) return HandRank.THREE_OF_A_KIND;
    if (rankCounts.values().stream().filter(count -> count == 2L).count() == 2) retur
    if (rankCounts.containsValue(2L)) return HandRank.ONE_PAIR;
    return HandRank.HIGH_CARD;
}
```

```
48  @Test
49  public void testRoyalFlush() {
50      List<Card> hand = Arrays.asList(
51          new Card(Card.Suit.HEARTS, Card.Rank.TEN),
52          new Card(Card.Suit.HEARTS, Card.Rank.J),
53          new Card(Card.Suit.HEARTS, Card.Rank.Q),
54          new Card(Card.Suit.HEARTS, Card.Rank.K),
55          new Card(Card.Suit.HEARTS, Card.Rank.A)
56      );
57      assertEquals(Game.HandRank.ROYAL_FLUSH, Game.classifyHand(hand), "Expected ROYAL_
58  }
59
60  @Test
61  public void testStraightFlush() {
62      List<Card> hand = Arrays.asList(
63          new Card(Card.Suit.HEARTS, Card.Rank.TWO),
64          new Card(Card.Suit.HEARTS, Card.Rank.THREE),
65          new Card(Card.Suit.HEARTS, Card.Rank.FOUR),
66          new Card(Card.Suit.HEARTS, Card.Rank.FIVE),
67          new Card(Card.Suit.HEARTS, Card.Rank.SIX)
68      );
69      assertEquals(Game.HandRank.STRAIGHT_FLUSH, Game.classifyHand(hand), "Expected STR
```

2. void testStraightFlush ()

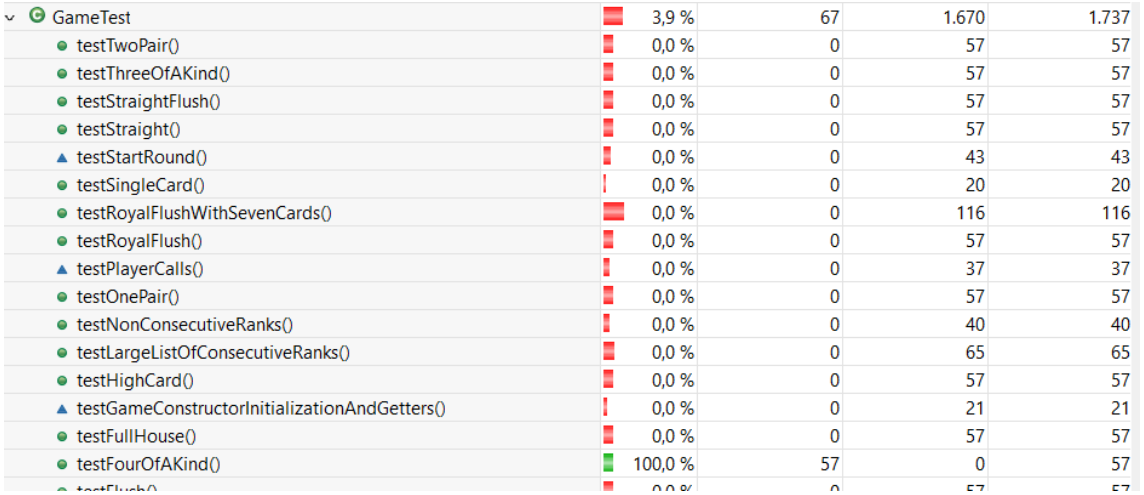
Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	100,0 %	57	0	57

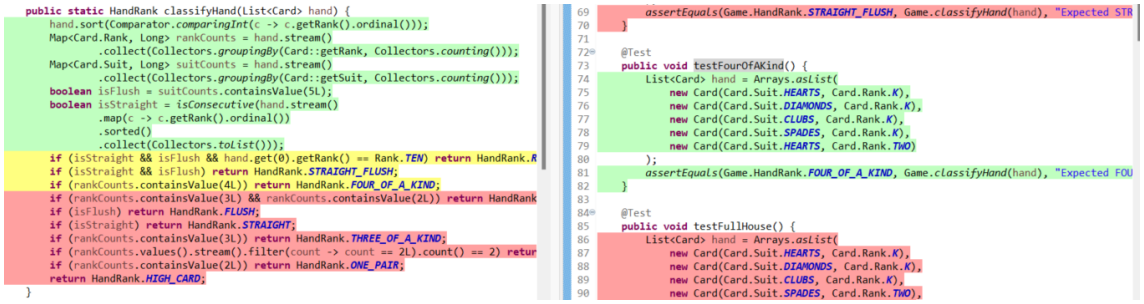


3. void testFourOfAKind ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



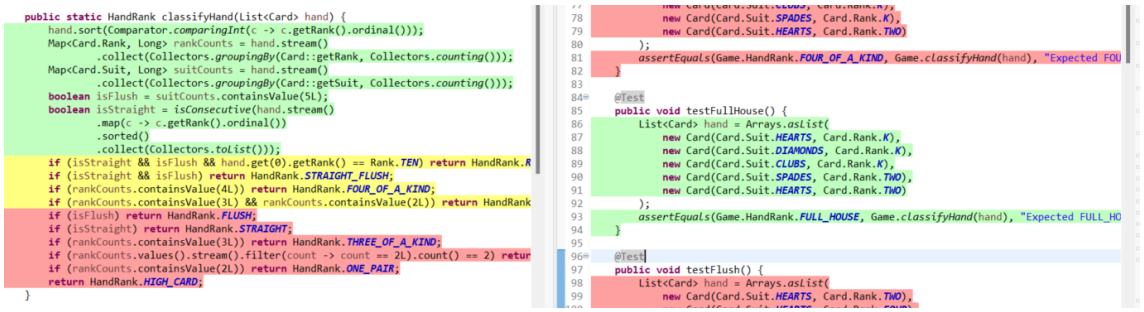
GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57
testPlayerCalls()	0,0 %	0	37	37
testOnePair()	0,0 %	0	57	57
testNonConsecutiveRanks()	0,0 %	0	40	40
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testHighCard()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	0,0 %	0	21	21
testFullHouse()	0,0 %	0	57	57
testFourOfAKind()	100,0 %	57	0	57



4. void testFullHouse ()

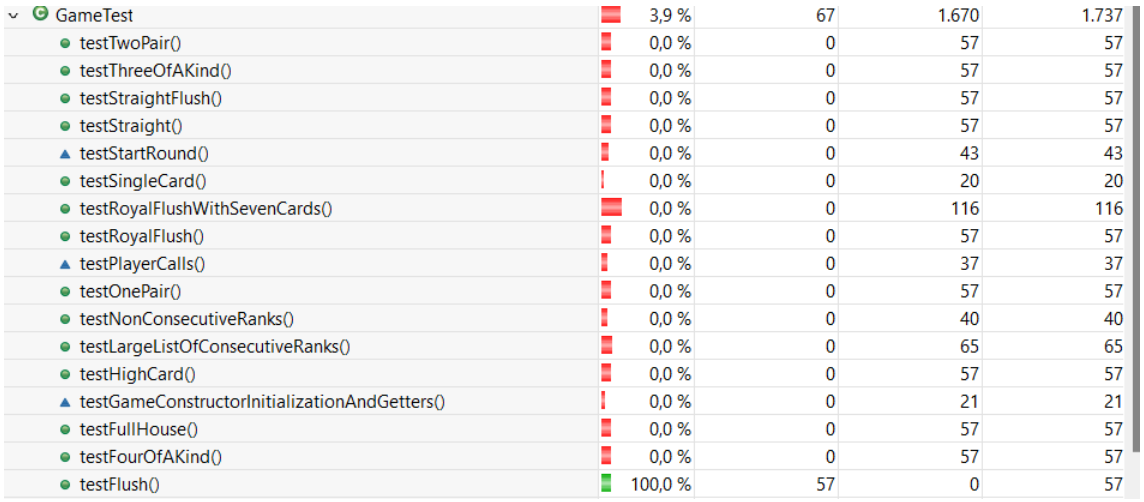
Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57
testPlayerCalls()	0,0 %	0	37	37
testOnePair()	0,0 %	0	57	57
testNonConsecutiveRanks()	0,0 %	0	40	40
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testHighCard()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	0,0 %	0	21	21
testFullHouse()	100,0 %	57	0	57

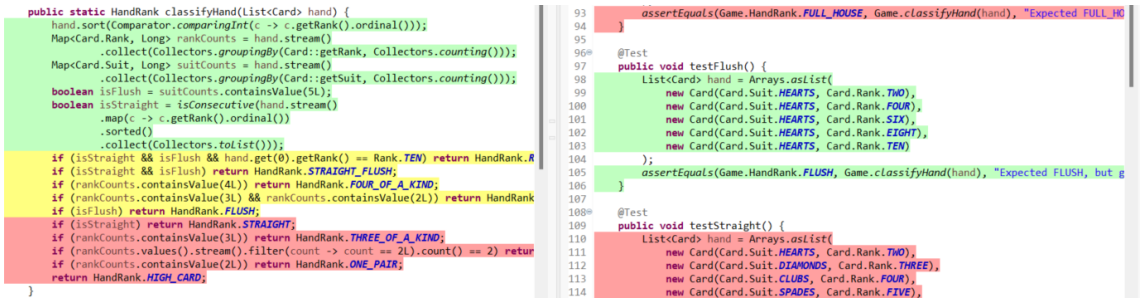


5. void testFlush ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



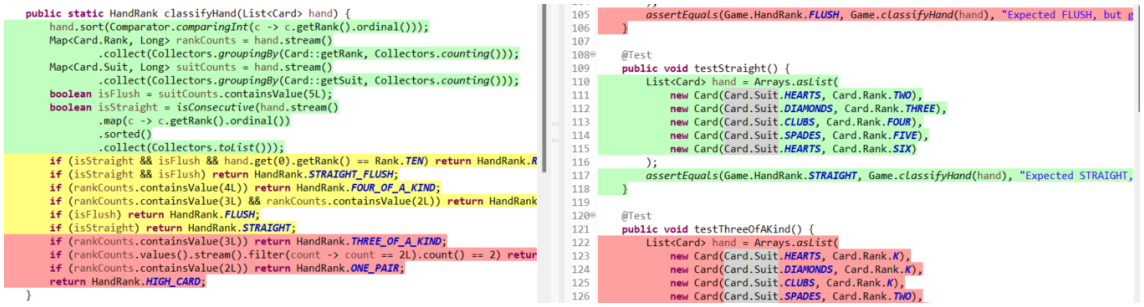
GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57
testPlayerCalls()	0,0 %	0	37	37
testOnePair()	0,0 %	0	57	57
testNonConsecutiveRanks()	0,0 %	0	40	40
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testHighCard()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	0,0 %	0	21	21
testFullHouse()	0,0 %	0	57	57
testFourOfAKind()	0,0 %	0	57	57
testFlush()	100,0 %	57	0	57



6. void testStraight()

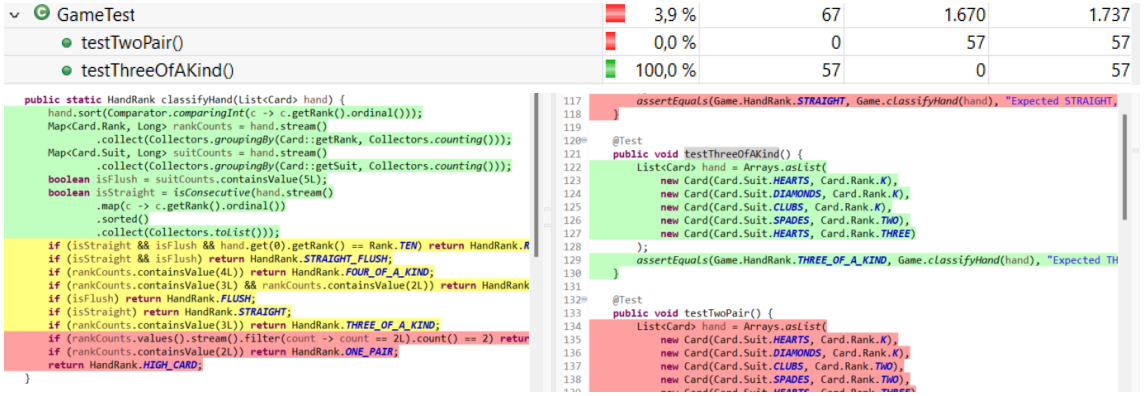
Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	100,0 %	57	0	57



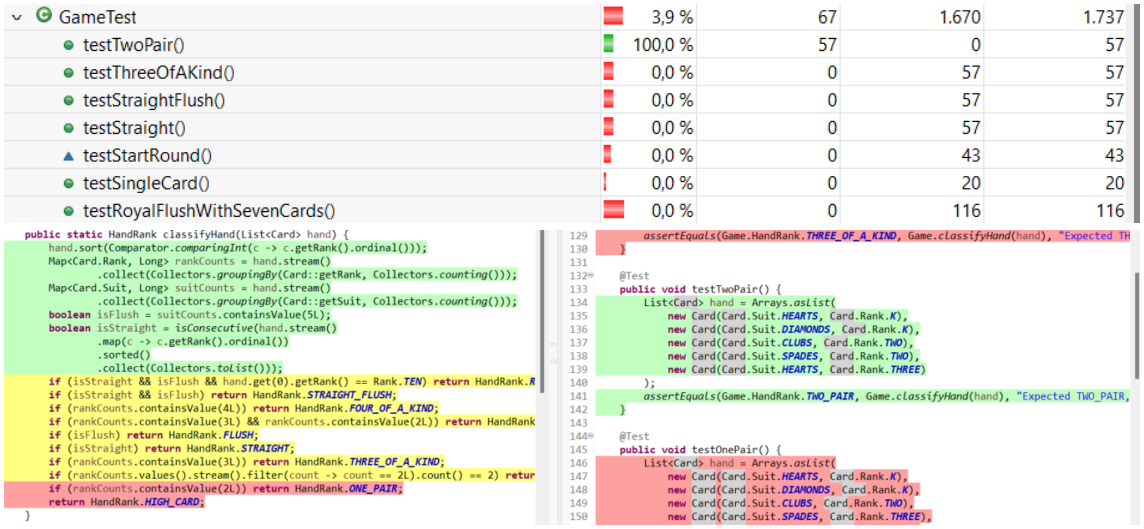
7. void testThreeOfAKind ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



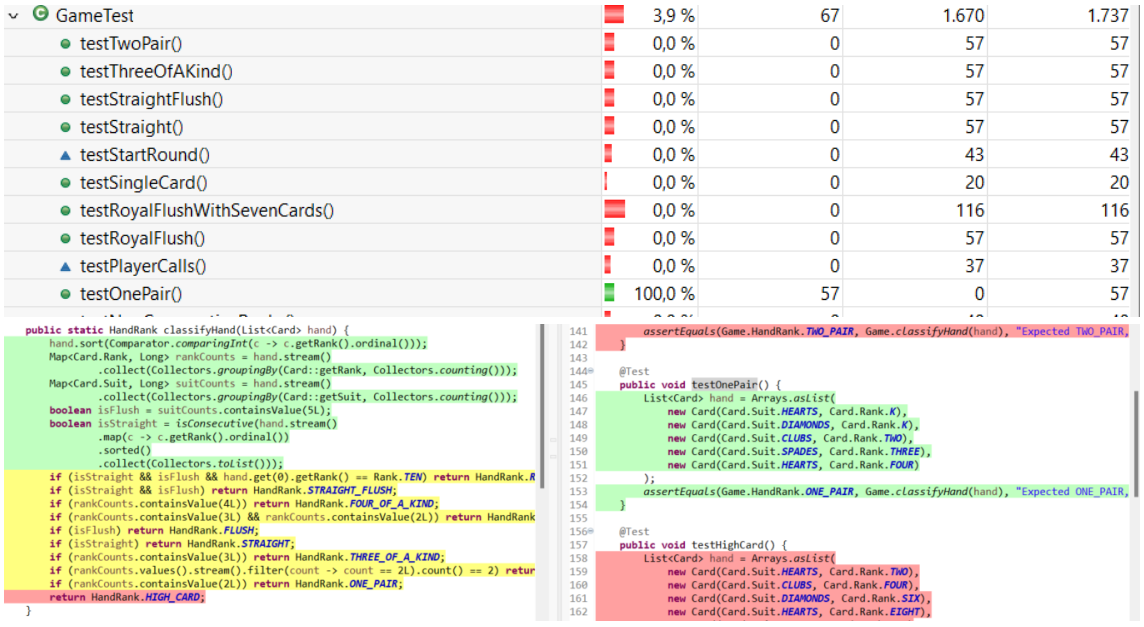
8. void testTwoPair ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



9. void testOnePair ()

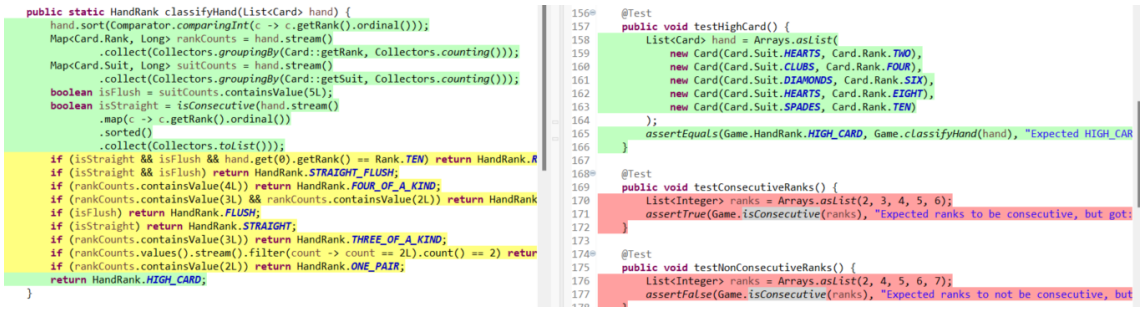
Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.



10. void testHighCard ()

Test de Caixa blanca: statement coverage, decisión coverage i condition coverage.

GameTest	3,9 %	67	1.670	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57
testPlayerCalls()	0,0 %	0	37	37
testOnePair()	0,0 %	0	57	57
testNonConsecutiveRanks()	0,0 %	0	40	40
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testHighCard()	100,0 %	57	0	57



```
public static HandRank classifyHand(List<Card> hand) {
    hand.sort(Comparator.comparingInt(c -> c.getRank().ordinal()));
    Map<Card.Rank, Long> rankCounts = hand.stream()
            .collect(Collectors.groupingBy(Card::getRank, Collectors.counting()));
    Map<Card.Suit, Long> suitCounts = hand.stream()
            .collect(Collectors.groupingBy(Card::getSuit, Collectors.counting()));
    boolean isFlush = suitCounts.containsValue(5L);
    boolean isStraight = isConsecutive(hand.stream()
            .map(c -> c.getRank().ordinal())
            .sorted()
            .collect(Collectors.toList()));
    if (isStraight && isFlush && hand.get(0).getRank() == Rank.TEN) return HandRank.R
    if (isStraight && isFlush) return HandRank.STRAIGHT_FLUSH;
    if (rankCounts.containsValue(4L)) return HandRank.FOUR_OF_A_KIND;
    if (rankCounts.containsValue(3L) && rankCounts.containsValue(2L)) return HandRank
    if (isFlush) return HandRank.FLUSH;
    if (isStraight) return HandRank.STRAIGHT;
    if (rankCounts.containsValue(3L)) return HandRank.THREE_OF_A_KIND;
    if (rankCounts.values().stream().filter(count -> count == 2L).count() == 2) retur
    if (rankCounts.containsValue(2L)) return HandRank.ONE_PAIR;
    return HandRank.HIGH_CARD;
}
```

```
@Test
public void testHighCard() {
    List<Card> hand = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.TWO),
        new Card(Card.Suit.CLUBS, Card.Rank.FOUR),
        new Card(Card.Suit.DIAMONDS, Card.Rank.SIX),
        new Card(Card.Suit.HEARTS, Card.Rank.EIGHT),
        new Card(Card.Suit.SPADES, Card.Rank.TEN)
    );
    assertEquals(Game.HandRank.HIGH_CARD, Game.classifyHand(hand), "Expected HIGH_CAR
}

@Test
public void testConsecutiveRanks() {
    List<Integer> ranks = Arrays.asList(2, 3, 4, 5, 6);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
}

@Test
public void testNonConsecutiveRanks() {
    List<Integer> ranks = Arrays.asList(2, 4, 5, 6, 7);
    assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
```

Funcionalitat: Metode per saber si les cartes son consecutives i formen una escala

Localització: Game.java, class: Game, public static boolean isConsecutive(List<Integer> ranks)

Test: GameTest.java, class: Game,

1. void testConsecutiveRanks ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage i loop testing.

Test de Caixa negra: particions equivalents.

GameTest	2,9 %	50	1.687	1.737
setUp()	100,0 %	7	0	7
testCompareFullHouse()	0,0 %	0	100	100
testCompareHighCard()	0,0 %	0	100	100
testCompareOnePair()	0,0 %	0	100	100
testCompareRoyalFlushVsStraightFlush()	0,0 %	0	100	100
testCompareTwoPairs()	0,0 %	0	100	100
testCompareTwoPairsOfAces()	0,0 %	0	100	100
testConsecutiveRanks()	100,0 %	40	0	40

```
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```
        new Card(Card.Suit.DIAMONDS, Card.Rank.SIX),
        new Card(Card.Suit.HEARTS, Card.Rank.EIGHT),
        new Card(Card.Suit.SPADES, Card.Rank.TEN)
    );
    assertEquals(Game.HandRank.HIGH_CARD, Game.classifyHand(hand), "Expected HIGH_CAR
}

@Test
public void testConsecutiveRanks() {
    List<Integer> ranks = Arrays.asList(2, 3, 4, 5, 6);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
}

@Test
public void testNonConsecutiveRanks() {
    List<Integer> ranks = Arrays.asList(2, 4, 5, 6, 7);
    assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
}

@Test
public void testConsecutiveRanksWithLowAce() {
    List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with low
}

@Test
public void testConsecutiveRanksWithHighAce() {
    List<Integer> ranks = Arrays.asList(10, 11, 12, 13, 14);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with high
}
```

2. void testNonConsecutiveRanks ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage i loop testing.

Test de Caixa negra: particions equivalents.

GameTest	2,9 %	50	1.687	1.737
setUp()	100,0 %	7	0	7
testCompareFullHouse()	0,0 %	0	100	100
testCompareHighCard()	0,0 %	0	100	100
testCompareOnePair()	0,0 %	0	100	100
testCompareRoyalFlushVsStraightFlush()	0,0 %	0	100	100
testCompareTwoPairs()	0,0 %	0	100	100
testCompareTwoPairsOfAces()	0,0 %	0	100	100
testConsecutiveRanks()	0,0 %	0	40	40
testConsecutiveRanksWithHighAce()	0,0 %	0	40	40
testConsecutiveRanksWithJumps()	0,0 %	0	40	40
testConsecutiveRanksWithLowAce()	0,0 %	0	40	40
testConsecutiveRanksWithWrapAroundAce()	0,0 %	0	40	40
testEmptyList()	0,0 %	0	15	15
testFlush()	0,0 %	0	57	57
testFourOfAKind()	0,0 %	0	57	57
testFullHouse()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	0,0 %	0	21	21
testHighCard()	0,0 %	0	57	57
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testNonConsecutiveRanks()	100,0 %	40	0	40

```
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```
        );
        assertEquals(Game.HandRank.HIGH_CARD, Game.classifyHand(hand), "Expected HIGH_CAR
    }

    @Test
    public void testConsecutiveRanks() {
        List<Integer> ranks = Arrays.asList(2, 3, 4, 5, 6);
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
    }

    @Test
    public void testNonConsecutiveRanks() {
        List<Integer> ranks = Arrays.asList(2, 4, 5, 6, 7);
        assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
    }

    @Test
    public void testConsecutiveRanksWithLowAce() {
        List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with low
    }

    @Test
    public void testConsecutiveRanksWithHighAce() {
        List<Integer> ranks = Arrays.asList(10, 11, 12, 13, 14);
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with high
    }

    @Test
    public void testSingleCard() {
```

3. void testConsecutiveRanksWithLowAce ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage i loop testing.

Test de Caixa negra: valors límit.

```
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```

    @Test
    public void testNonConsecutiveRanks() {
        List<Integer> ranks = Arrays.asList(2, 4, 5, 6, 7);
        assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
    }

    @Test
    public void testConsecutiveRanksWithLowAce() {
        List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with low
    }

    @Test
    public void testConsecutiveRanksWithHighAce() {
        List<Integer> ranks = Arrays.asList(10, 11, 12, 13, 14);
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with high
    }

    @Test
    public void testSingleCard() {
        List<Integer> ranks = Arrays.asList(5);
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
    }

    @Test
    public void testEmptyList() {
        List<Integer> ranks = Arrays.asList();
        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive (empty li
    }
```

4. void testConsecutiveRanksWithHighAce ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage.

GameTest	2,9 %	50	1.687	1.737
setUp()	100,0 %	7	0	7
testCompareFullHouse()	0,0 %	0	100	100
testCompareHighCard()	0,0 %	0	100	100
testCompareOnePair()	0,0 %	0	100	100
testCompareRoyalFlushVsStraightFlush()	0,0 %	0	100	100
testCompareTwoPairs()	0,0 %	0	100	100
testCompareTwoPairsOfAces()	0,0 %	0	100	100
testConsecutiveRanks()	0,0 %	0	40	40
testConsecutiveRanksWithHighAce()	100,0 %	40	0	40

```
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```
182     List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
183     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with low
184 }
185
186 @Test
187 public void testConsecutiveRanksWithHighAce() {
188     List<Integer> ranks = Arrays.asList(10, 11, 12, 13, 14);
189     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with high
190 }
191
192 @Test
193 public void testSingleCard() {
194     List<Integer> ranks = Arrays.asList(5);
195     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
196 }
197
198 @Test
199 public void testEmptyList() {
200     List<Integer> ranks = Arrays.asList();
201     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive (empty li
202 }
203
204 @Test
205 public void testConsecutiveRanksWithWrapAroundAce() {
206     List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
207     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with wrap
208 }
209
210 @Test
211 public void testLargeListOfConsecutiveRanks() {
```

5. void testSingleCard ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage i loop testing.

GameTest	1,7 %	30	1.707	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	100,0 %	20	0	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57

```
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```
185
186 @Test
187 public void testConsecutiveRanksWithHighAce() {
188     List<Integer> ranks = Arrays.asList(10, 11, 12, 13, 14);
189     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with high
190 }
191
192 @Test
193 public void testSingleCard() {
194     List<Integer> ranks = Arrays.asList(5);
195     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
196 }
197
198 @Test
199 public void testEmptyList() {
200     List<Integer> ranks = Arrays.asList();
201     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive (empty li
202 }
203
204 @Test
205 public void testConsecutiveRanksWithWrapAroundAce() {
206     List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
207     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with wrap
208 }
209
210 @Test
211 public void testLargeListOfConsecutiveRanks() {
212     // Test a larger list of consecutive ranks (e.g., 1 to 10)
213     List<Integer> ranks = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
214     assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
215 }
```

6. void testEmptyList ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage.

GameTest	1,4 %	25	1.712	1.737
testTwoPair()	0,0 %	0	57	57
testThreeOfAKind()	0,0 %	0	57	57
testStraightFlush()	0,0 %	0	57	57
testStraight()	0,0 %	0	57	57
testStartRound()	0,0 %	0	43	43
testSingleCard()	0,0 %	0	20	20
testRoyalFlushWithSevenCards()	0,0 %	0	116	116
testRoyalFlush()	0,0 %	0	57	57
testPlayerCalls()	0,0 %	0	37	37
testOnePair()	0,0 %	0	57	57
testNonConsecutiveRanks()	0,0 %	0	40	40
testLargeListOfConsecutiveRanks()	0,0 %	0	65	65
testHighCard()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	0,0 %	0	21	21
testFullHouse()	0,0 %	0	57	57
testFourOfAKind()	0,0 %	0	57	57
testFlush()	0,0 %	0	57	57
testEmptyList()	100,0 %	15	0	15

```
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```
191
192    @Test
193    public void testSingleCard() {
194        List<Integer> ranks = Arrays.asList(5);
195        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
196    }
197
198    @Test
199    public void testEmptyList() {
200        List<Integer> ranks = Arrays.asList();
201        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive (empty li
202    }
203
204    @Test
205    public void testConsecutiveRanksWithWrapAroundAce() {
206        List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
207        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with wrap
208    }
209
210    @Test
211    public void testLargeListOfConsecutiveRanks() {
212        // Test a larger list of consecutive ranks (e.g., 1 to 10)
213        List<Integer> ranks = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
214        assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
215    }
216
217    @Test
218    public void testConsecutiveRanksWithJumps() {
219        List<Integer> ranks = Arrays.asList(3, 5, 6, 7, 8);
220        assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
```

7. void testConsecutiveRanksWithWrapAroundAce ()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage i loop testing.

GameTest	2,9 %	50	1.687	1.737
setUp()	100,0 %	7	0	7
testCompareFullHouse()	0,0 %	0	100	100
testCompareHighCard()	0,0 %	0	100	100
testCompareOnePair()	0,0 %	0	100	100
testCompareRoyalFlushVsStraightFlush()	0,0 %	0	100	100
testCompareTwoPairs()	0,0 %	0	100	100
testCompareTwoPairsOfAces()	0,0 %	0	100	100
testConsecutiveRanks()	0,0 %	0	40	40
testConsecutiveRanksWithHighAce()	0,0 %	0	40	40
testConsecutiveRanksWithJumps()	0,0 %	0	40	40
testConsecutiveRanksWithLowAce()	0,0 %	0	40	40
testConsecutiveRanksWithWrapAroundAce()	100,0 %	40	0	40

```java
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```java
@Test
public void testEmptyList() {
    List<Integer> ranks = Arrays.asList();
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive (empty li
}

@Test
public void testConsecutiveRanksWithWrapAroundAce() {
    List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with wrap
}

@Test
public void testLargeListOfConsecutiveRanks() {
    // Test a larger list of consecutive ranks (e.g., 1 to 10)
    List<Integer> ranks = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
}

@Test
public void testConsecutiveRanksWithJumps() {
    List<Integer> ranks = Arrays.asList(3, 5, 6, 7, 8);
    assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
}

@Test
public void testCompareRoyalFlushVsStraightFlush() {
    List<Card> royalFlush = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.TEN),
```

8. void testLargeListOfConsecutiveRanks ()

Test de Caixa blanca: statement coverage.

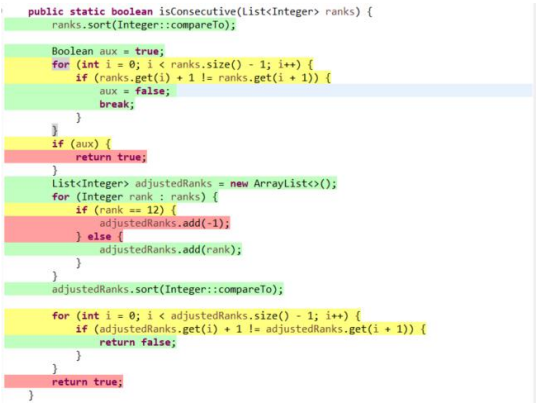
GameTest	4,3 %	75	1.662	1.737
setUp()	100,0 %	7	0	7
testCompareFullHouse()	0,0 %	0	100	100
testCompareHighCard()	0,0 %	0	100	100
testCompareOnePair()	0,0 %	0	100	100
testCompareRoyalFlushVsStraightFlush()	0,0 %	0	100	100
testCompareTwoPairs()	0,0 %	0	100	100
testCompareTwoPairsOfAces()	0,0 %	0	100	100
testConsecutiveRanks()	0,0 %	0	40	40
testConsecutiveRanksWithHighAce()	0,0 %	0	40	40
testConsecutiveRanksWithJumps()	0,0 %	0	40	40
testConsecutiveRanksWithLowAce()	0,0 %	0	40	40
testConsecutiveRanksWithWrapAroundAce()	0,0 %	0	40	40
testEmptyList()	0,0 %	0	15	15
testFlush()	0,0 %	0	57	57
testFourOfAKind()	0,0 %	0	57	57
testFullHouse()	0,0 %	0	57	57
testGameConstructorInitializationAndGetters()	0,0 %	0	21	21
testHighCard()	0,0 %	0	57	57
testLargeListOfConsecutiveRanks()	100,0 %	65	0	65

9. void testConsecutiveRanksWithJumps()

Test de Caixa blanca: statement coverage, decision coverage, condition coverage.

Test de Caixa negra: valors frontera.

GameTest	2,9 %	50	1.687	1.737
setUp()	100,0 %	7	0	7
testCompareFullHouse()	0,0 %	0	100	100
testCompareHighCard()	0,0 %	0	100	100
testCompareOnePair()	0,0 %	0	100	100
testCompareRoyalFlushVsStraightFlush()	0,0 %	0	100	100
testCompareTwoPairs()	0,0 %	0	100	100
testCompareTwoPairsOfAces()	0,0 %	0	100	100
testConsecutiveRanks()	0,0 %	0	40	40
testConsecutiveRanksWithHighAce()	0,0 %	0	40	40
testConsecutiveRanksWithJumps()	100,0 %	40	0	40
testConsecutiveRanksWithLowAce()	0,0 %	0	40	40



```java
public static boolean isConsecutive(List<Integer> ranks) {
    ranks.sort(Integer::compareTo);

    Boolean aux = true;
    for (int i = 0; i < ranks.size() - 1; i++) {
        if (ranks.get(i) + 1 != ranks.get(i + 1)) {
            aux = false;
            break;
        }
    }
    if (aux) {
        return true;
    }
    List<Integer> adjustedRanks = new ArrayList<>();
    for (Integer rank : ranks) {
        if (rank == 12) {
            adjustedRanks.add(-1);
        } else {
            adjustedRanks.add(rank);
        }
    }
    adjustedRanks.sort(Integer::compareTo);

    for (int i = 0; i < adjustedRanks.size() - 1; i++) {
        if (adjustedRanks.get(i) + 1 != adjustedRanks.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

```java
    List<Integer> ranks = Arrays.asList(12, 0, 1, 2, 3);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive with wrap
}

@Test
public void testLargeListOfConsecutiveRanks() {
    // Test a larger list of consecutive ranks (e.g., 1 to 10)
    List<Integer> ranks = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    assertTrue(Game.isConsecutive(ranks), "Expected ranks to be consecutive, but got:
}

@Test
public void testConsecutiveRanksWithJumps() {
    List<Integer> ranks = Arrays.asList(3, 5, 6, 7, 8);
    assertFalse(Game.isConsecutive(ranks), "Expected ranks to not be consecutive, but
}

@Test
public void testCompareRoyalFlushVsStraightFlush() {
    List<Card> royalFlush = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.TEN),
        new Card(Card.Suit.HEARTS, Card.Rank.J),
        new Card(Card.Suit.HEARTS, Card.Rank.Q),
        new Card(Card.Suit.HEARTS, Card.Rank.K),
        new Card(Card.Suit.HEARTS, Card.Rank.A)
    );

    List<Card> straightFlush = Arrays.asList(
        new Card(Card.Suit.HEARTS, Card.Rank.FIVE),
        new Card(Card.Suit.HEARTS, Card.Rank.SIX),
```

Classe Game

Funcionalitat: Métode per saber quina mà es millor entre dues

Localització: Game.java, class: Game, public static int compareHands(List<Card> hand1, List<Card> hand2)

Test: GameTest.java, class: Game

1. public void testCompareRoyalFlushVsStraightFlush()

2. public void testCompareTwoPairs()

3. public void testCompareFullHouse()

4. public void testCompareOnePair()

5. public void testCompareHighCard()

6. public void testCompareTwoPairsOfAces()

Funcionalitat: Métode per saber quina mà es millor entre unes cartes

Localització: Game.java, class: Game, public static List<Card> bestHand(List<Card> cards)

Test: GameTest.java, class: Game

1. public void testRoyalFlushWithSevenCards()

Funcionalitat: Métode per saber el que es paga per cada mà

Localització: Game.java, class: Game, public static int payRatio(HandRank hr)

Test: GameTest.java, class: Game

void testRoyalFlushMultiplier()

void testStraightFlushMultiplier()

void testFourOfAKindMultiplier()

void testFullHouseMultiplier()

void testFlushMultiplier()

void testStraightMultiplier()
void testThreeOfAKindMultiplier()
void testTwoPairMultiplier()
void testOnePairMultiplier()
void testHighCardMultiplier()
void testDefaultMultiplier()

Funcionalitat: Métode per saber quina mà es millor entre unes cartes

Localització: Game.java, class: Game, public static List<Card> bestHand(List<Card> cards)

Test: GameTest.java, class: Game, void testBestHand()

Funcionalitat: Actualitzar les monedes del jugador a la vista

Localització: MainController.java, class: MainController, public void updateViewPlayerCoins()

Test: MainControllerTest.java, class: MainController, void testUpdateViewPlayerCoins()

Funcionalitat: Actualitzar l'aposta inicial a la vista

Localització: MainController.java, class: MainController, public void updateViewAnteBet()

Test: MainControllerTest.java, class: MainController, void testUpdateViewAnteBet()

Funcionalitat: Actualitzar l'aposta de trucada a la vista

Localització: MainController.java, class: MainController, public void updateViewCallBet()

Test: MainControllerTest.java, class: MainController, void testUpdateViewCallBet()

Funcionalitat: Permetre al jugador col·locar monedes

Localització: MainController.java, class: MainController, public void placeCoin(int value)

Test: MainControllerTest.java, class: MainController, void testPlaceCoin()

Funcionalitat: Eliminar les monedes col·locades

Localització: MainController.java, class: MainController, public void removePlacedCoins()

Test: MainControllerTest.java, class: MainController, void testRemovePlacedCoins()

Funcionalitat: Permetre al jugador seleccionar una carta

Localització: MainController.java, class: MainController, public int pickCard(Boolean table, int index)

Test: MainControllerTest.java, class: MainController, void testPickCard()

Funcionalitat: Avançar al següent estat del joc

Localització: MainController.java, class: MainController, public void next()

Test: MainControllerTest.java, class: MainController, void testNext()

Funcionalitat: Seleccionar el guanyador del joc

Localització: MainController.java, class: MainController, public void selectWinner()

Test: MainControllerTest.java, class: MainController, void testSelectWinner()

Funcionalitat: Realitzar l'aposta inicial o de trucada

Localització: MainController.java, class: MainController, public void makeBet()

Test: MainControllerTest.java, class: MainController, void testMakeBet()

Funcionalitat: Permetre al jugador retirar-se i reiniciar el joc

Localització: MainController.java, class: MainController, public void retire()

Test: MainControllerTest.java, class: MainController, void testRetire()

Funcionalitat: Mostrar les cartes del joc

Localització: MainController.java, class: MainController, public int viewGameCards()

Test: MainControllerTest.java, class: MainController, void testViewGameCards()