calculateScoreForCategory

A picture containing diagram, sketch, drawing, line

Description automatically generated

Edge-pair coverage:

TR: {[1,2,3], [1,2,4], [1,2,5], [1,2,6], [1,2,7], [1,2,8], [1,2,9], [1,2,10], [1,2,11], [1,2,12], [1,2,13], [1,2,14], [1,2,15], [1,2,16], [2,3,17], [2,4,17], [2,5,17], [2,6,17], [2,7,17], [2,8,17], [2,9,17], [2,10,17], [2,11,17], [2,12,17], [2,13,17], [2,14,17], [2,15,17], [2,16,17]}

TP: {[1,2,3,17], [1,2,4,17], [1,2,5,17], [1,2,6,17] , [1,2,7,17], [1,2,8,17], , [1,2,9,17], [1,2,10,17], [1,2,11,17], [1,2,12,17], [1,2,13,17], [1,2,14,17], [1,2,15,17], [1,2,16,17], }

DU coverage

|  |  |  |
| --- | --- | --- |
| Variable | Du-pair | Du path |
| score | (1, 17)  (3, 17)  (4, 17)  (5, 17)  (6, 17)  (7, 17)  (8, 17)  (9, 17)  (10, 17)  (11, 17)  (12, 17)  (13, 17)  (14, 17)  (15, 17) | [1, 2, 16, 17]  [3, 17]  [4, 17]  [5, 17]  [6, 17]  [7, 17]  [8, 17]  [9, 17]  [10, 17]  [11, 17]  [12, 17]  [13, 17]  [14, 17]  [15, 17] |
| category | (1, 2) | [1, 2] |
| diceList | (1, 3)  (1, 4)  (1, 5)  (1, 6)  (1, 7)  (1, 8)  (1, 9)  (1, 10)  (1, 11)  (1, 12)  (1, 13)  (1, 14)  (1, 15) | [1,2,3]  [1,2,4]  [1,2,5]  [1,2,6]  [1,2,7]  [1,2,8]  [1,2,9]  [1,2,10]  [1,2,11]  [1,2,12]  [1,2,13]  [1,2,14]  [1,2,15] |

JUnit test

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.ArrayList;

public class ScoreCalculatorTest {

private ArrayList<Dice> diceList;

@BeforeEach

public void setUp() {

diceList = new ArrayList<>();

diceList.add(new Dice(1));

diceList.add(new Dice(2));

diceList.add(new Dice(3));

diceList.add(new Dice(4));

diceList.add(new Dice(5));

}

@Test

public void testOnes() {

int score = ScoreCalculator.calculateScoreForCategory("Ones", diceList);

assertEquals(1, score);

}

@Test

public void testTwos() {

int score = ScoreCalculator.calculateScoreForCategory("Twos", diceList);

assertEquals(2, score);

}

// Similarly, write tests for other categories

@Test

public void testInvalidCategory() {

int score = ScoreCalculator.calculateScoreForCategory("Invalid Category", diceList);

assertEquals(0, score);

}

}

calcNumScore

A picture containing diagram, text, circle, screenshot

Description automatically generated

Edge-pair coverage

TR: {[1,2,3], [1,2,5], [2,3,4], [2,3,2], [3,4,2], [3,2,5], [3,2,3], [4,2,3], [4,2,5]}

TP: {[1,2,5], [1,2,3,4,2,5], [1,2,3,2,5], [1,2,3,2,3,4,2,5]}

Du coverage

|  |  |  |
| --- | --- | --- |
| variable | Du pair | Du path |
| Sum | (1, 4)  (4, 4)  (4, 5) | [1, 2, 3, 4]  [4, 2, 3, 4]  [4, 2, 5] |
| diceList | (1,2) | [1,2] |
| dice | (2,3) | [2, 3] |
| number | (1, 3)  (1, 4) | [1, 2, 3]  [1, 2, 3, 4] |

Junit Test

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import java.util.ArrayList;

public class ScoreCalculatorTest {

private ArrayList<Dice> diceList;

@BeforeEach

public void setUp() {

diceList = new ArrayList<>();

diceList.add(new Dice(1));

diceList.add(new Dice(2));

diceList.add(new Dice(3));

diceList.add(new Dice(2));

diceList.add(new Dice(5));

}

@Test

public void testCalcNumScoreOnes() {

int sum = ScoreCalculator.calcNumScore(diceList, 1);

assertEquals(1, sum);

}

@Test

public void testCalcNumScoreTwos() {

int sum = ScoreCalculator.calcNumScore(diceList, 2);

assertEquals(4, sum);

}

// You can add more test cases for other numbers if necessary

@Test

public void testCalcNumScoreInvalidNumber() {

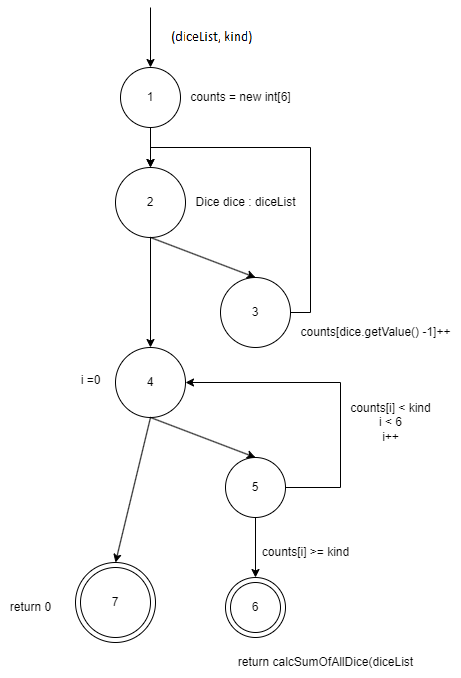
int sum = ScoreCalculator.calcNumScore(diceList, 7);

assertEquals(0, sum);

}

}

calcSumOfAKind



Edge-pair coverage

TR: {[1,2,3], [1,2,4], [2,3,2], [2,4,5], [2,4,7], [3,2,3], [3,2,4], [4,5,4], [4,5,6], [5,4,5]}

TP: {[1,2,4,7], [1,2,4,5,6], [1,2,3,2,4,5,4,7], [1,2,3,2,3,2,4,5,4,5,6]}

Du coverage

|  |  |  |
| --- | --- | --- |
| variable | Du pair | Du path |
| counts | (1, 3)  (1, 5)  (3, 5) | [1, 2, 3]  [1, 2, 4, 5]  [3, 2, 4, 5] |
| diceList | (1, 2)  (1, 6) | [1, 2]  [1, 2, 4, 5, 6] |
| kind | (1, 5) | [1,2,4,5] |
| dice | (2, 3) | [2, 3] |

Junit Test

import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.\*;

public class CalcSumOfAKindTest {

@Test

public void testCalcSumOfAKind() {

ArrayList<Dice> diceList = new ArrayList<>();

diceList.add(new Dice(1));

diceList.add(new Dice(2));

diceList.add(new Dice(2));

diceList.add(new Dice(2));

diceList.add(new Dice(5));

// Edge-pair coverage test cases

assertEquals(0, calcSumOfAKind(diceList, 1)); // min kind

assertEquals(6, calcSumOfAKind(diceList, 2)); // typical 2 of a kind

assertEquals(8, calcSumOfAKind(diceList, 3)); // typical 3 of a kind

assertEquals(0, calcSumOfAKind(diceList, 6)); // max kind

// All-DU-Path coverage test cases

assertEquals(6, calcSumOfAKind(diceList, 2)); // minimum values for all variables

assertEquals(10, calcSumOfAKind(diceList, 4)); // kind greater than number of dice

assertEquals(15, calcSumOfAKind(diceList, 5)); // kind equal to number of dice

assertEquals(0, calcSumOfAKind(new ArrayList<>(), 3)); // empty dice list

assertEquals(0, calcSumOfAKind(diceList, 4)); // no kind found

}

private static int calcSumOfAllDice(ArrayList<Dice> diceList) {

int sum = 0;

for (Dice dice : diceList) {

sum += dice.getValue();

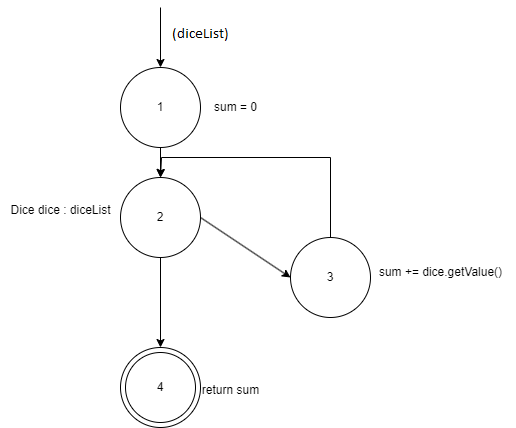
}

return sum;

}

}

calcSumOfAllDice



Edge-pair coverage

TR: {[1,2,3], [1,2,4], [2,3,2], [3,2,3], [3,2,4]}

TP: {[1, 2, 4], [1, 2, 3, 2, 4], [1, 2, 3, 2, 3, 2, 4]}

Du coverage

|  |  |  |
| --- | --- | --- |
| variable | Du pair | Du path |
| Sum | (1, 3)  (1, 4) | [1, 2, 3]  [1, 2, 4] |
| diceList | (1, 2) | [1, 2] |
| dice | (2, 3) | [2, 3] |

Junit Test

import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.\*;

public class CalcSumOfAllDiceTest {

@Test

public void testCalcSumOfAllDice() {

ArrayList<Dice> diceList = new ArrayList<>();

diceList.add(new Dice(1));

diceList.add(new Dice(2));

diceList.add(new Dice(3));

// Edge-pair coverage test cases

assertEquals(0, calcSumOfAllDice(new ArrayList<>())); // empty dice list

assertEquals(1, calcSumOfAllDice(new ArrayList<>(List.of(new Dice(1))))); // list with 1 dice

assertEquals(6, calcSumOfAllDice(diceList)); // typical list of dice

assertEquals(18, calcSumOfAllDice(new ArrayList<>(List.of(new Dice(6), new Dice(6), new Dice(6))))); // list with all 6s

// All-DU-Path coverage test cases

assertEquals(0, calcSumOfAllDice(new ArrayList<>())); // empty dice list

assertEquals(1, calcSumOfAllDice(new ArrayList<>(List.of(new Dice(1))))); // list with 1 dice

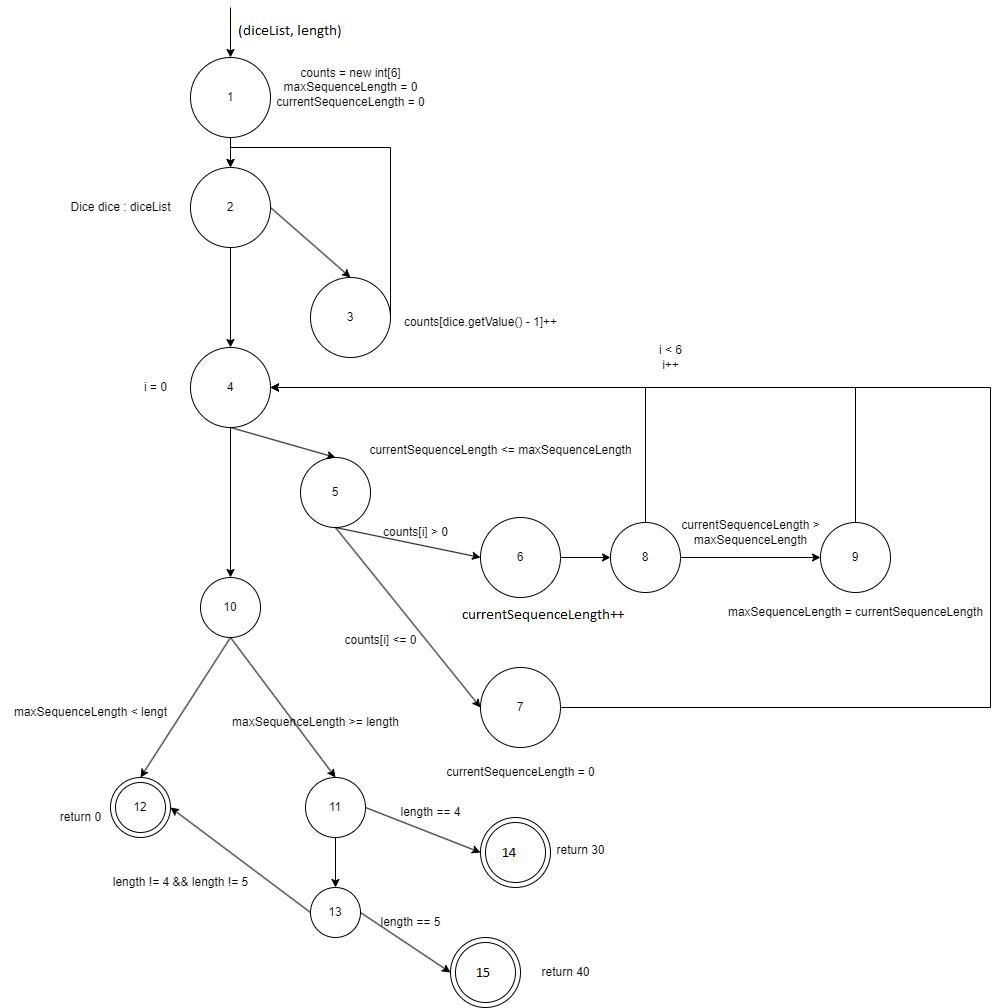
assertEquals(9, calcSumOfAllDice(new ArrayList<>(List.of(new Dice(4), new Dice(5))))); // typical list of dice

assertEquals(18, calcSumOfAllDice(new ArrayList<>(List.of(new Dice(6), new Dice(6), new Dice(6))))); // list with all 6s

}

}

calcStraightScore



Edge-pair coverage

TR: {[1,2,3], [1,2,4], [2,3,2], [2,4,5], [2,4,10], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [4,10,11], [4,10,12], [5,7,4], [5,6,8], [6,8,9], [6,8,4], [7,4,5], [7,4,10], [8,4,5], [8,4,10], [8,9,4], [9,4,5], [9,4,10], [10,11,13], [10,11,14], [11,13,12], [11,13,15]}

TP: {[1, 2, 4, 10, 12], [1, 2, 3, 2, 4, 5, 7, 4, 10, 11, 13, 12], [1, 2, 3, 2, 3, 2, 4, 5, 6, 8, 4, 10, 11, 13, 15], [1, 2, 3, 2, 3, 2, 4, 5, 6, 8, 4, 5, 6, 8, 9, 4, 10, 11, 14], [1, 2, 3, 2, 3, 2, 4, 5, 6, 8, 9, 4, 5, 7, 4, 10, 11, 14]}

Du coverage

|  |  |  |
| --- | --- | --- |
| Variable | Du pair | Du path |
| Counts | (1, 3)  (1, 5)  (3, 5) | [1, 2, 3]  [1, 2, 4, 5]  [3, 2, 4, 5] |
| maxSequenceLength | (1, 8)  (1, 10)  (9, 10) | [1, 2, 4, 5, 6, 8]  [1, 2, 4, 10]  [9, 4, 10] |
| currentSequenceLength | (1, 6)  (1, 8)  (1, 9)  (6, 8)  (6, 9)  (7, 6)  (7, 8)  (7, 9) | [1, 2, 4, 5, 6]  [1, 2, 4, 5, 6, 8]  [1, 2, 4, 5, 6, 8, 9]  [6, 8]  [6, 8, 9]  [7, 4, 5, 6]  [7, 4, 5, 6, 8]  [7, 4, 5, 6, 8, 9] |
| diceList | (1, 2) | [1, 2] |
| Length | (1, 10)  (1, 11)  (1, 13) | [1, 2, 4, 10]  [1, 2, 4, 10, 11]  [1, 2, 4, 10, 11, 13] |
| Dice | (2, 3) | [2, 3] |
| i | (4, 5)  (4, 7)  (4, 8)  (4, 9)  (7, 7)  (7, 8)  (7, 9)  (8, 7)  (8, 8)  (9, 7)  (9, 8)  (9, 9) | [4, 5]  [4, 5, 7]  [4, 5, 6, 8]  [4, 5, 6, 8, 9]  [7, 4, 5, 6, 8]  [7, 4, 5, 6, 8, 9]  [8, 4, 5, 7]  [9, 4, 5, 7]  [9, 4, 5, 6, 8] |

Junit Test

import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.\*;

public class CalcStraightScoreTest {

@Test

public void testCalcStraightScore() {

ArrayList<Dice> diceList = new ArrayList<>();

diceList.add(new Dice(1));

diceList.add(new Dice(2));

diceList.add(new Dice(3));

diceList.add(new Dice(4));

diceList.add(new Dice(6)); // Note that there is no 5, so this is not a large straight

// Edge-pair coverage test cases

assertEquals(0, calcStraightScore(new ArrayList<>(), 4)); // empty dice list

assertEquals(0, calcStraightScore(diceList, 3)); // typical short straight

assertEquals(30, calcStraightScore(diceList, 4)); // typical small straight

assertEquals(40, calcStraightScore(diceList, 5)); // typical large straight

// All-DU-Path coverage test cases

assertEquals(0, calcStraightScore(new ArrayList<>(), 4)); // empty dice list

assertEquals(0, calcStraightScore(diceList, 3)); // short straight

assertEquals(0, calcStraightScore(new ArrayList<>(List.of(new Dice(1), new Dice(2), new Dice(3), new Dice(5))), 4)); // small straight with missing value

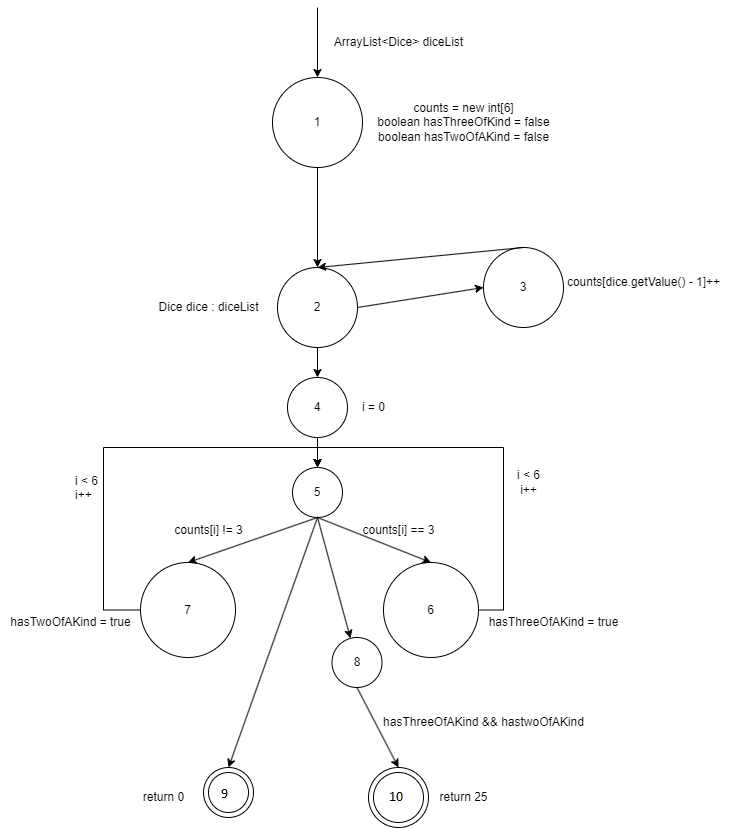
assertEquals(30, calcStraightScore(diceList, 4)); // small straight

assertEquals(40, calcStraightScore(new ArrayList<>(List.of(new Dice(1), new Dice(2), new Dice(3), new Dice(4), new Dice(5))), 5)); // large straight

}

}

calcFullHouseScore



Edge-pair coverage

TR: {[1,2,3], [1,2,4], [2,3,2], [2,4,5], [3,2,3], [3,2,4], [4,5,6], [4,5,7], [4,5,8], [4,5,9], [5,6,5], [5,7,5], [5,8,10], [6,5,6], [6,5,7], [6,5,8], [6,5,9], [7,5,7], [7,5,6], [7,5,8], [7,5,9]}

TP: {[1,2,4,5,10], [1,2,3,2,4,5,8,10], [1,2,3,2,3,2,4,5,6,5,6,5,9], [1,2,4,5,6,5,8,10], [1,2,4,5,7,5,7,5,9], [1,2,4,5,6,5,7,5,8,10], [1,2,4,5,7,5,6,5,9] }

Du coverage

|  |  |  |
| --- | --- | --- |
| variable | Du pair | Du path |
| Counts | (1, 3)  (1, 5)  (3, 5) | [1, 2, 3]  [1, 2, 4, 5]  [3, 2, 4, 5] |
| hasThreeOfAKind | (1, 6)  (1, 8) | [1, 2, 4, 5, 6]  [1, 2, 4, 5, 8] |
| hasTwoOfAKind | (1, 7)  (1, 8) | [1, 2, 4, 5, 7]  [1, 2, 4, 5, 8] |
| diceList | (1, 2) | [1, 2] |
| Dice | (2, 3) | [2, 3] |
| I | (4, 5) | [4, 5] |

Junit Test

import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.\*;

public class CalcFullHouseScoreTest {

@Test

public void testCalcFullHouseScore() {

ArrayList<Dice> diceList = new ArrayList<>();

diceList.add(new Dice(2));

diceList.add(new Dice(2));

diceList.add(new Dice(3));

diceList.add(new Dice(3));

diceList.add(new Dice(3));

// Edge-pair coverage test cases

assertEquals(0, calcFullHouseScore(new ArrayList<>())); // empty dice list

assertEquals(0, calcFullHouseScore(new ArrayList<>(List.of(new Dice(1), new Dice(1), new Dice(1), new Dice(2), new Dice(2))))); // no full house

assertEquals(25, calcFullHouseScore(diceList)); // typical full house

// All-DU-Path coverage test cases

assertEquals(0, calcFullHouseScore(new ArrayList<>())); // empty dice list

assertEquals(0, calcFullHouseScore(new ArrayList<>(List.of(new Dice(1), new Dice(2), new Dice(2), new Dice(2), new Dice(2))))); // no three of a kind

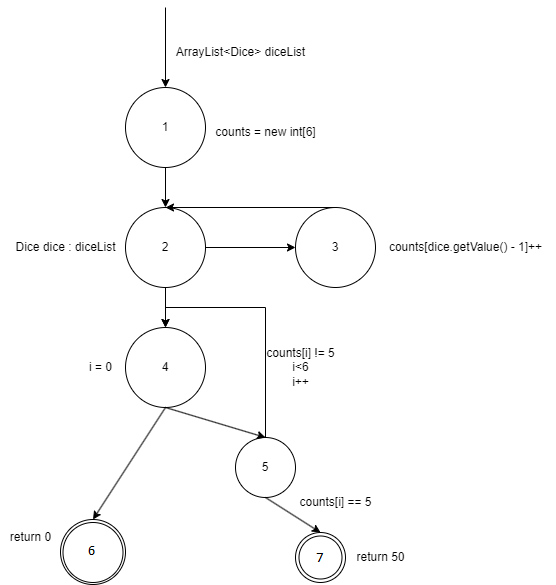
assertEquals(0, calcFullHouseScore(new ArrayList<>(List.of(new Dice(1), new Dice(1), new Dice(1), new Dice(2), new Dice(3))))); // no two of a kind

assertEquals(25, calcFullHouseScore(diceList)); // typical full house

}

}

calcYahtzeeScore



Edge-pair coverage

TR: {[1,2,3], [1,2,4], [2,3,2], [2,4,5], [2,4,6], [3,2,3], [3,2,4], [4,5,4], [4,5,7], [5,4,5], [5,4,6]}

TP: {[1, 2, 4, 6], [1, 2, 3, 2, 4, 5, 4, 6], [1, 2, 3, 2, 3, 2, 4, 5, 4, 5, 7]}

Du coverage

|  |  |  |
| --- | --- | --- |
| Variable | Du pair | Du path |
| Counts | (1, 3)  (1, 5) | [1, 2, 3]  [1, 2, 4, 5] |
| Dice | (2, 3) | [2, 3] |
| I | (4, 5)  (5, 5) | [4, 5] |
| diceList | (1, 2) | [1, 2] |

Junit Test

import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.\*;

public class CalcYahtzeeScoreTest {

@Test

public void testCalcYahtzeeScore() {

ArrayList<Dice> diceList = new ArrayList<>();

diceList.add(new Dice(1));

diceList.add(new Dice(1));

diceList.add(new Dice(1));

diceList.add(new Dice(1));

diceList.add(new Dice(1));

// Edge-pair coverage test cases

assertEquals(0, calcYahtzeeScore(new ArrayList<>())); // empty dice list

assertEquals(0, calcYahtzeeScore(new ArrayList<>(List.of(new Dice(1), new Dice(1), new Dice(1), new Dice(1), new Dice(2))))); // no Yahtzee

assertEquals(50, calcYahtzeeScore(diceList)); // typical Yahtzee

// All-DU-Path coverage test cases

assertEquals(0, calcYahtzeeScore(new ArrayList<>())); // empty dice list

assertEquals(0, calcYahtzeeScore(new ArrayList<>(List.of(new Dice(1), new Dice(1), new Dice(1), new Dice(2), new Dice(2))))); // no five of a kind

assertEquals(50, calcYahtzeeScore(diceList)); // typical Yahtzee

}

}