Table of Contents

[**Test Suite No.1** *(Class: NotationReader; Method:void parseDiceNotation(String diceNotation); type: Unit/Black Box )* 3](#_Toc464379909)

[Test Execution 3](#_Toc464379910)

[Source Code 3](#_Toc464379911)

[Output 5](#_Toc464379912)

[Test Result 6](#_Toc464379913)

[Test Summary 6](#_Toc464379914)

[**Test Suite No.2** *(Class: NotationReader; Method:void parseDiceNotation(String diceNotation); type: UnitWhite Box )* 7](#_Toc464379915)

[Test Execution 8](#_Toc464379916)

[Source Code 8](#_Toc464379917)

[Output 9](#_Toc464379918)

[Test Summary 9](#_Toc464379919)

[**Test Suite No.3** 10](#_Toc464379920)

[Test Execution 10](#_Toc464379921)

[Source Code 10](#_Toc464379922)

[Output 12](#_Toc464379923)

[Test Result 12](#_Toc464379924)

[Test Summary 12](#_Toc464379925)

[**Test Suite No.4** 13](#_Toc464379926)

[Test Execution 13](#_Toc464379927)

[Source Code 13](#_Toc464379928)

* Invalid inputs/parameters cannot materialize in the methods of objects that are below **NotationReader class** i.e. that are dependent on **NotationReader** class. This is because, all data generated is dependent on dice notation. And dice notation is **validated by NotationReader** before methods of other objects are invoked.
* Therefore, all objects dependent upon the **NotationReader** object, when executes its behavior will receive the correct input/parameter be it object passed by reference or primitive type passed by value.
* Taking account of the above points, all test data that follows besides that of methods in NotationReader will have **valid test data type only.**
* In order to **test private methods, from test package**, without breaking the encapsulation of original class. I have made duplicate class of the original class, in test package (Example – **Dice** class was duplicated in test package as “**publicForTest\_Dice**”) & changed the private modifier to protected in the duplicate class only.

## Test Suite No.1

**Testing** **class**: cardstacks.NotationReader  
**Testing** **type**: Black Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void parseDiceNotation(String diceNotation) | 4d4-2 | Valid (2 delimiters –  ‘d’ & ‘-’) | Sets values of instance variables as following:   * diceNotation = 4d4-2 * numDices = 4 * numFaces = 4 * toRemove = 2 |
| 2 | void parseDiceNotation(String diceNotation) | 4d4 | Valid (1 delimiter – ‘d’) | Sets values instance variables as following:   * diceNotation = 4d4 * numDices = 4 * numFaces = 4 * toRemove = 0 |
| 3 | void parseDiceNotation(String diceNotation) | xyz | Invalid | Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!" |
| 4 | void parseDiceNotation(String diceNotation) | “” | Null | Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!" |

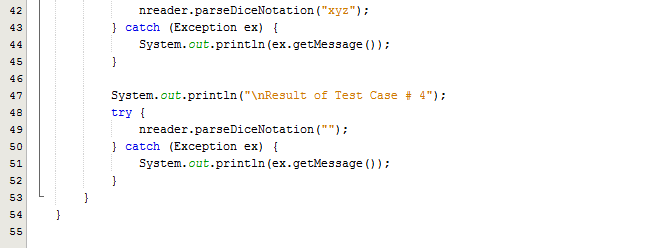
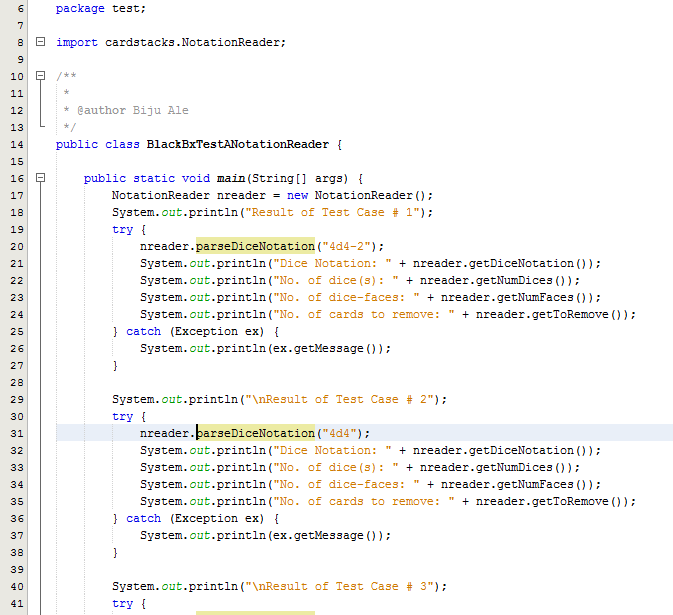
**Method of Equivalence partitioning:**

* Two types of valid (validated by Regex Pattern matcher) input is present – 1 sample was selected from each equivalence partition.
* Anything besides valid input’s Regex Pattern is another partition. 1 sample was selected.
* If no input is given, this is taken as another partition. Null is selected.
* No boundary value analysis required as per the nature of expected parameter.

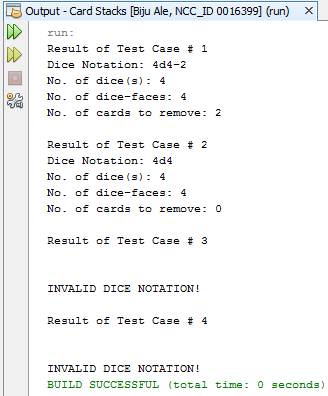
### Test Execution

#### Source Code

**[PLEASE TURN OVER]**



#### Output



### Test Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** | **Actual outcome as expected?** |
| 1 | void parseDiceNotation(String diceNotation) | 4d4-2 | Valid (2 delimiters –  ‘d’ & ‘-’) | Sets values of instance variables as following:   * diceNotation = 4d4-2 * numDices = 4 * numFaces = 4 * toRemove = 2 | Yes |
| 2 | void parseDiceNotation(String diceNotation) | 4d4 | Valid (1 delimiter – ‘d’) | Sets values instance variables as following:   * diceNotation = 4d4 * numDices = 4 * numFaces = 4 * toRemove = 0 | Yes |
| 3 | void parseDiceNotation(String diceNotation) | xyz | Invalid | Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!" | Yes |
| 4 | void parseDiceNotation(String diceNotation) | “” | Null | Throws NumberFormatException with message - "\n\nINVALID DICE NOTATION!" | Yes |

### Test Summary

From the above test results, all tests were executed as expected.

Test Suite No.1 **also implicitly covered white box & black box tests for getter methods**, which returned the respective values of instance variables. Hence, it too executed as expected without any errors.

## Test Suite **No**.2

**Testing** **class**: cardstacks.NotationReader  
**Testing** **type**: White Box / Unit Testing

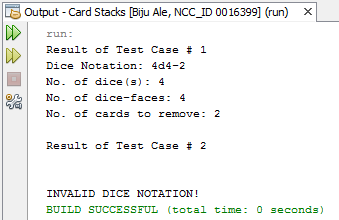
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void parseDiceNotation(String diceNotation) | 4d4-2 | Valid (2 delimiters –  ‘d’ & ‘-’) | Sets values of instance variables as following:   * diceNotation = 4d4-2 * numDices = 4 * numFaces = 4 * toRemove = 2 |
| 2 | void parseDiceNotation(String diceNotation) | xyz | Invalid | Throws NumberFormatException with message - "INVALID DICE NOTATION!" |

### Test Execution

#### Source Code



#### Output



### Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

## Test Suite No.3

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: Black Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | Dice(NotitficationReader nreader) | nreader | Valid object | Constructor should set the dice name |
| 2 | Dice (NotificationReader nreader) | nreader | \*Invalid object (due to invalid data member) | Constructor should not set the dice name. Exception should be thrown with message –  "No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n" |

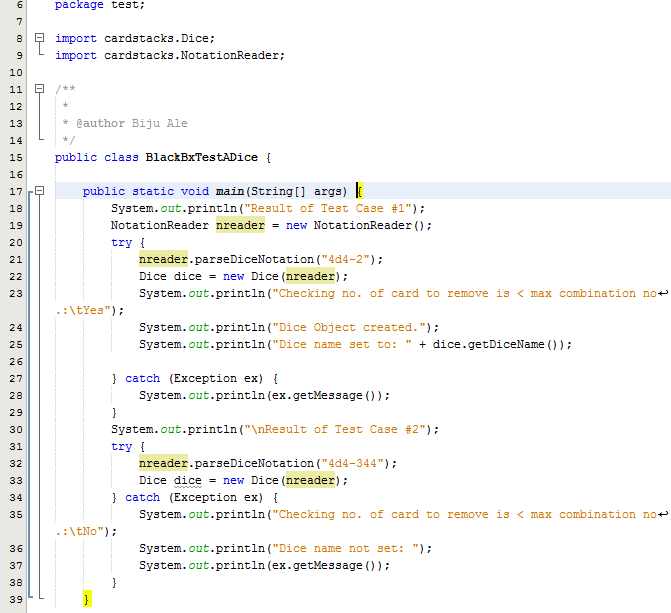
**Method of test data selection:**

\*Here, invalid ‘nreader’ means the parsing was correct (Test Suite No.1) but the number of card to remove exceeded the maximum combination number (total no. of cards). Correct parsing is checked in Test Suite No.1 whereas the valid no. of card to remove is checked in Test Suite No.3’s constructor.

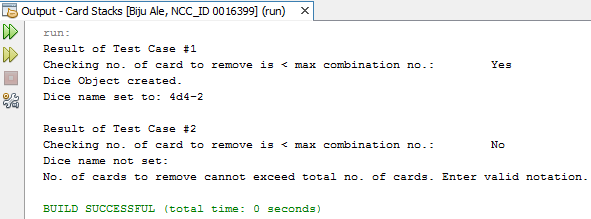
### Test Execution

#### Source Code

**[PLEASE TURN OVER]**



#### Output



### Test Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** | **Actual outcome as expected?** |
| 1 | Dice(NotitficationReader nreader) | nreader | Valid object | Constructor should set the dice name | Yes |
| 2 | Dice (NotificationReader nreader) | nreader | \*Invalid object (due to invalid data member) | Constructor should not set the dice name. Exception should be thrown with message –  "No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n" | Yes |

### Test Summary

From the above test results, all tests were executed as expected.

## Test Suite No.4

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: White Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | Dice(NotitficationReader nreader) | nreader | Valid object | Constructor should set the dice name |
| 2 | Dice (NotificationReader nreader) | nreader | \*Invalid object (due to invalid data member) | Constructor should not set the dice name. Exception should be thrown with message –  "No. of cards to remove cannot exceed total no. of cards. Enter valid notation.\n" |

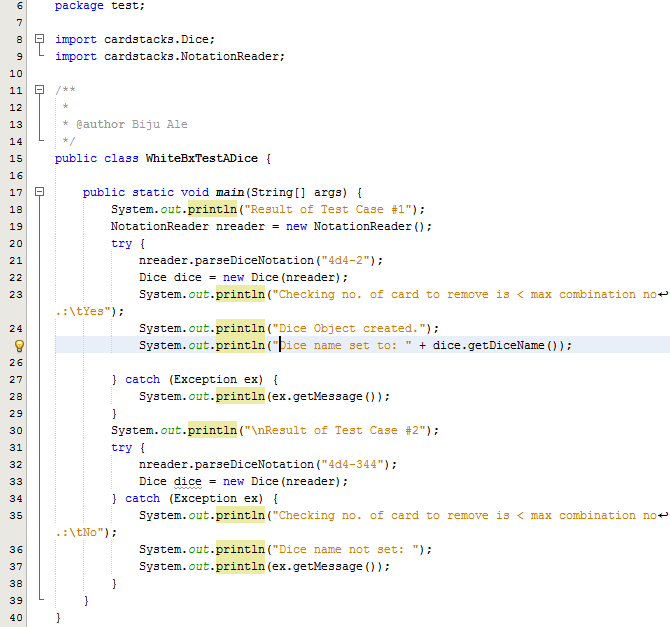
**Method of test data selection:**

\*Here, invalid ‘nreader’ means the parsing was correct (Test Suite No.1) but the number of card to remove exceeded the maximum combination number (total no. of cards). Correct parsing is checked in Test Suite No.1 whereas the valid no. of card to remove is checked in Test Suite No.3’s constructor.

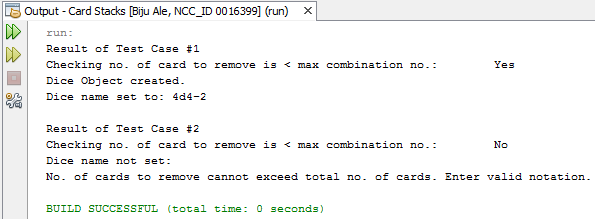
### Test Execution

#### Source Code

**[PLEASE TURN OVER]**



#### Output:



### Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

## Test Suite No. 5

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: Black Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void setMinMax(int minCombination, int maxCombination) | (4, 16) | \*Valid | Sets values of instance variables as following:   * minCombination = 4 * maxCombination = 16 |

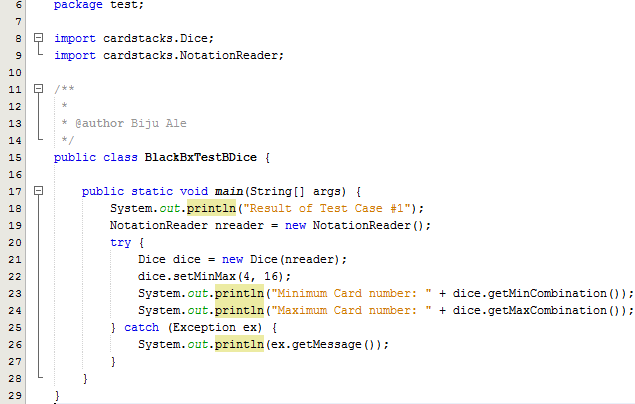
**Method of test data selection:**

\*Only 1 valid test case because, dice notation is already validated by NotationReader class and Dice class. Hence, setMinMax() always receives valid input/parameter.

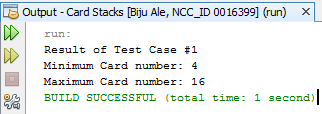
No boundary value analysis required as per the nature of expected parameter.

### Test Execution

#### Source Code



#### Output



### Test Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** | **Actual outcome as expected?** |
| 1 | void setMinMax(int minCombination, int maxCombination) | (4, 16) | \*Valid | Sets values of instance variables as following:   * minCombination = 4 * maxCombination = 16 | Yes |

### Test Summary

From the above test results, all tests were executed as expected.

## Test Suite No.6

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: White Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void setMinMax(int minCombination, int maxCombination) | (4, 16) | \*Valid | Sets values of instance variables as following:   * minCombination = 4 * maxCombination = 16 |

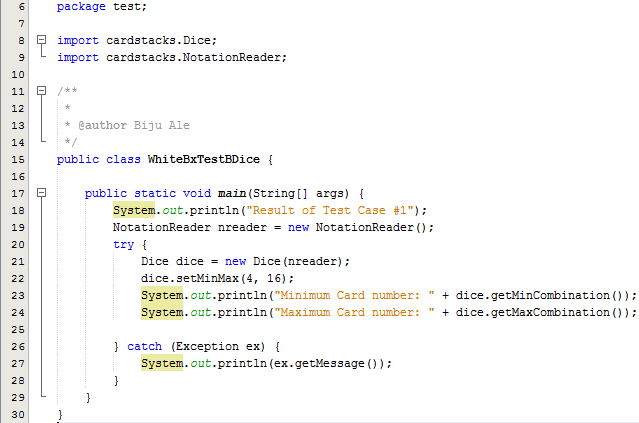
**Method of test data selection:**

\*Only 1 valid test case because, dice notation is already validated by NotationReader class and Dice class. Hence, setMinMax() always receives valid input/parameter.

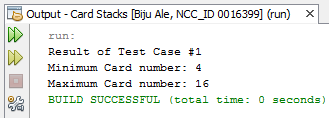
No boundary value analysis required as per the nature of the expected parameter.

### Test Execution

#### Source Code



#### Output



### Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

## Test Suite No. 7

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: Black Box / Unit Testing

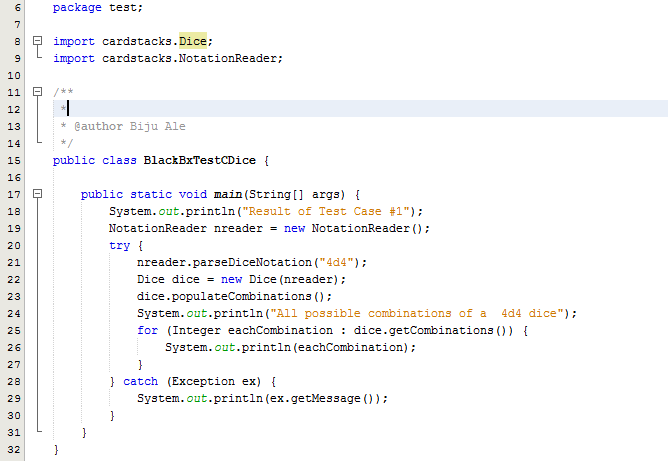
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void populateCombinations() | \*N/A | N/A | Populates Integer array with all correct possible combinations from the given dice roll (Sample used – 4d4). |

**Method of test data selection:**

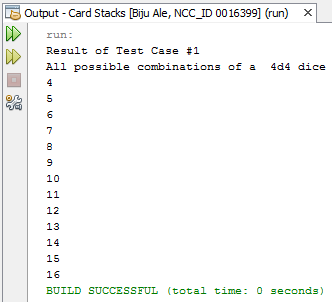
\* This is a parameter-less method therefore; no test data is given. The method depends on already-validated values of minCombination & maxCombination values’ set by method checked in Test Suite No. 6.

### Test Execution

#### Source Code



#### Output



### Test Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** | **Actual outcome as expected?** |
| 1 | void populateCombinations() | \*N/A | N/A | Populates Integer array with all correct possible combinations from the given dice roll (Sample used – 4d4). | Yes |

**Method of test data selection:**

\* This is a parameter-less method therefore; no test data is given. The method depends on already-validated values of minCombination & maxCombination values’ set by method checked in Test Suite No. 6.

No boundary value analysis required as per the nature of the expected parameter.

### Test Summary

From the above test results, all tests were executed as expected.

## Test Suite No. 8

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: White Box / Unit Testing

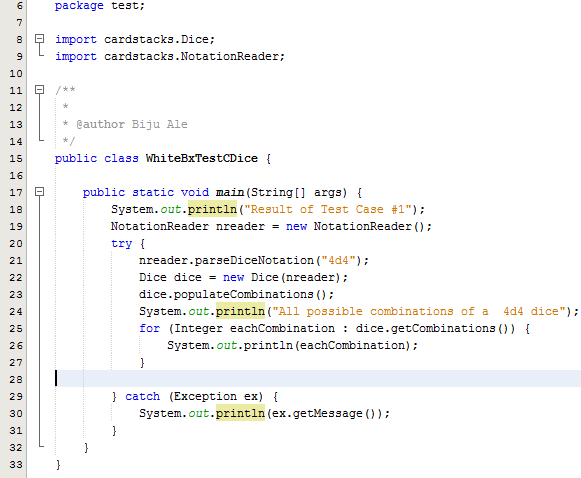
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void populateCombinations() | \*N/A | N/A | Populates Integer array with all correct possible combinations from the given dice roll (Sample used – 4d4). |

**Method of test data selection:**

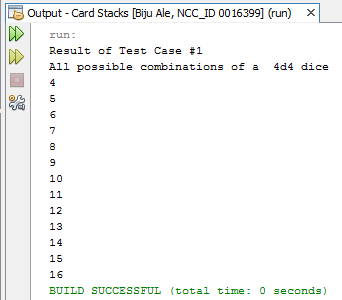
\* This is a parameter-less method therefore; no test data is given. The method depends on already-validated values of minCombination & maxCombination values’ set by method checked in Test Suite No. 6.

### Test Execution

#### Source Code



#### Output



### Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

## Test Suite No. 9

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: Black Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void roll(Integer[] Combinations) | dice.getCombinations()  which returns Integer[] Combinations containing:  (4,5,6,7,8,9,10.11,12,13,14,15,16) | Valid | Roll the dice 10000 times & record the frequencies of each combination/card in frequencies[] Integer array. |

**Method of test data selection:**

\* The method depends on already-validated values in the Combination[] Integer generated by method checked in Test Suite No. 7. Therefore, this method always receives valid input/parameter. Here, a sample dice – 4d4 is used.

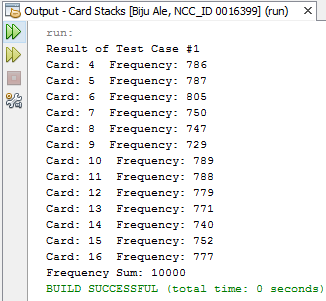
### Test Execution

#### Source Code

**[PLEASE TURN OVER]**



#### Output



### Test Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Method** | **Test Data** | **Input type** | **Expected Outcome** | **Actual outcome as expected?** |
| 1 | void roll(Integer[] Combinations) | dice.getCombinations()  which returns Integer[] Combinations containing:  (4,5,6,7,8,9,10.11,12,13,14,15,16) | Valid | Roll the dice 10000 times & record the frequencies of each combination/card in frequencies[] Integer array. | Yes |

### Test Summary

From the above test results, all tests were executed as expected.

## Test Suite No.10

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: White Box / Unit Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | void roll(Integer[] Combinations) | dice.getCombinations()  which returns Integer[] Combinations containing:  (4,5,6,7,8,9,10.11,12,13,14,15,16) | Valid | Roll the dice 10000 times & record the frequencies of each combination/card in frequencies[] Integer array. |

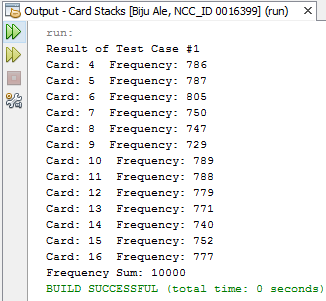
### Test Execution

#### Source Code

**[PLEASE TURN OVER]**



#### Output



### Test Summary

As from the above output, all statements, conditions and branches were executed at least once, and no errors were discovered during the execution.

## Test Suite No.11

**Testing** **class**: cardstacks.Dice  
**Testing** **type**: Integration testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** |
| 1 | Dice(NotitficationReader nreader) | nreader | Valid object | Constructor should set the dice name |
| 2 | void setMinMax(int minCombination, int maxCombination) | (4, 16) | Valid | Sets values of instance variables as following:   * minCombination = 4 * maxCombination = 16 |
| 3 | void populateCombinations() | N/A (parameter-less) | Valid | Populates Integer array with all correct possible combinations from the given dice roll (Sample used – 4d4). |
| 4 | void roll(Integer[] Combinations) | dice.getCombinations()  which returns Integer[] Combinations containing:  (4,5,6,7,8,9,10.11,12,13,14,15,16) | Valid | Roll the dice 10000 times & record the frequencies of each combination/card in frequencies[] Integer array. |

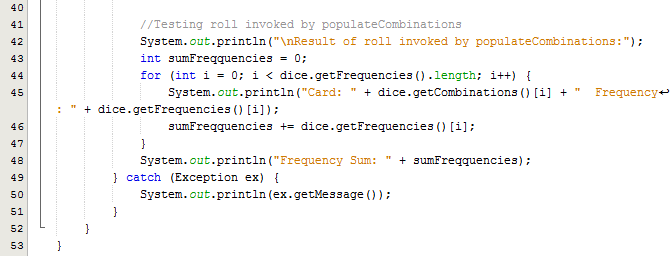
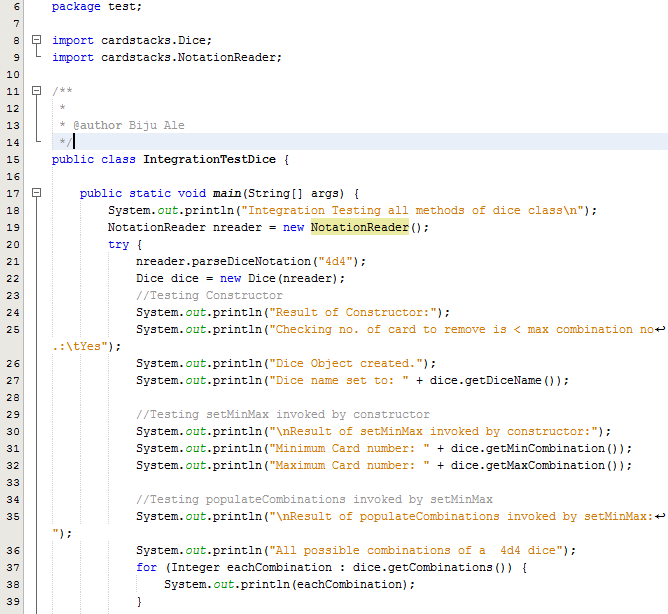
**Method of test data selection**

* Since constructor invokes setMinMax, and setMinMax invokes populateCombinations and so on, the test data are passed by value as the methods communicate.
* To initiate the communication between all method, constructor is passed with object nreader containing data-member – 4d4.

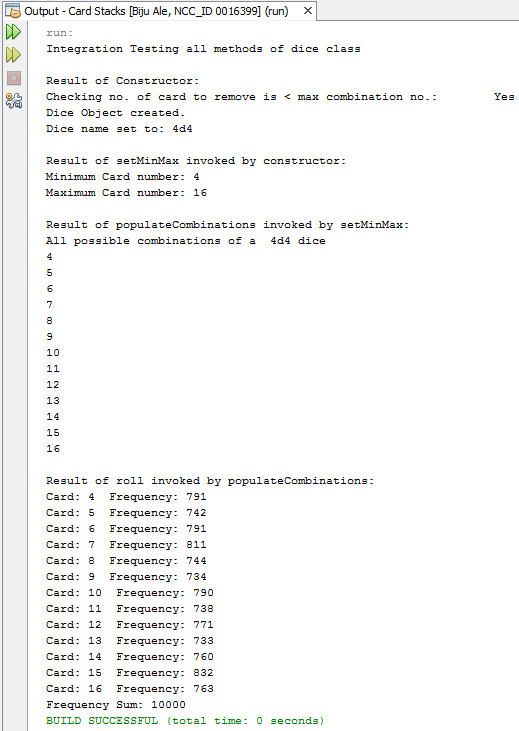
### Test Execution

#### Source code

**[PLEASE TURN OVER]**



#### Output



### Test Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Method** | **Test Data** | **Input type** | **Expected Outcome** | **Actual outcome as expected?** |
| 1 | Dice(NotitficationReader nreader) | nreader | Valid object | Constructor should set the dice name | Yes |
| 2 | void setMinMax(int minCombination, int maxCombination) | (4, 16) | Valid | Sets values of instance variables as following:   * minCombination = 4 * maxCombination = 16 | Yes |
| 3 | void populateCombinations() | N/A (parameter-less) | Valid | Populates Integer array with all correct possible combinations from the given dice roll (Sample used – 4d4). | Yes |
| 4 | void roll(Integer[] Combinations) | dice.getCombinations()  which returns Integer[] Combinations containing:  (4,5,6,7,8,9,10.11,12,13,14,15,16) | Valid | Roll the dice 10000 times & record the frequencies of each combination/card in frequencies[] Integer array. | Yes |

### Test Summary

Above result shows that communication between methods were executed without any errors. Integration testing of methods in dice class was executed as expected.