**Assignment 1 – Die Object**

**Dean Zeller 20 points**

### CS200 Due: Friday, August 31st

**Fall 2018**

***Objective*** The students will design a simple die object for use in game- and statistics-related assignments, given later in the course.

***Background***

Many board games employ some sort of random determination mechanic. The most common version for gameplay is a set of six-sided dice (singular, die). *Yahtzee* uses five dice, whereas *Monopoly*, *Parcheesi* and *Backgammon* use two. Other games such as *Dungeons and Dragons* use dice of different sizes, denoted by a D. D4 refers to a 4-sided die, with an equal chance of any number between 1 and 4, whereas a D20 would have a range between 1 and 20. The object you create within this assignment will be used for future assignments. This is more of an assignment in understanding how objects are set up in Java.

***Layout:***

Object: Die (Note: do not call it dice, that is plural and denotes multiple die in a set)

Attributes:

name – name of the die, such as *D4*, *Sword Damage*, or *Monopoly Turn*

numSides – number of sides on the die

currentValue – the current value of the die roll

Methods:

|  |  |
| --- | --- |
| *Method* | *Purpose* |
| Constructor (no parameters) | Create a die object, with 6 sides. |
| Constructor(numSides) | Create a die object, with a parameter for the number of sides. Use this to call the other constructor. |
| getNumSides | Return the number of sides. |
| getCurrentValue | Return the current value of the die. |
| toString | Return a formatted table listing the values of the attributes |
| roll | Set the current value to a generated random value between 1 and the number of sides, inclusive. |
| cheat | Set the current value to the specified parameter, with error checking to ensure the new value is not greater than the number of sides. |
| reallyCheat | Set the current value to the specified parameter. Do not provide any error checking, and allow the die roll to be any integer value, regardless of common sense. |

***Testing:***

Use your program tester to put your die object through several tests. Run the five tests shown on the test run, and create three other tests. Emulate games as necessary, to identify use.

***Test Run:***

Test 1: Six-sided die (D6)

Range 1 to 6

Single number test: 5

Multiple number test: 3 1 6 4 5 1 2 3 1 1

Test 2: Percentile die (D100)

Range 1 to 100

Single number test: 17

Multiple number test: 31 12 63 41 15 17 28 39 81 71

Test 3: Random month (D12)

Range 1 to 12

Single number test: 4

Multiple number test: 8 2 7 4 5 7 2 12 8 11

Test 4: Monopoly turn (2D6 added together)

Range 2 to 12

Single number test: 7

Multiple number test: 7 9 8 5 10 6 7 9 11 8

Test 5: Yahtzee (Chance field) (5D6 added together)

Range 5 to 30

Single number test: 17

Multiple number test: 13 12 16 14 15 11 12 23 14 16

***Other Tests:***

6th level Mage Fireball (5D6)

6th level Mage Magic Missile (3D4 + 3)

Cure light wounds (D8)

Dagger damage on a large creature (D3)

Game of life spinner (D10)

Flipping a coin (D2)

Magic 8-Ball (D20)

Trivial pursuit and many other games (D6)

***Instructions***

1. Create two Java program files named Die.java and DieTester.java.
2. Enter the appropriate file documentation to identify you as the primary programmer. See the *Java Programming Documentation and Style Guidelines* for more information.
3. In Die.java, implement a class for the die object described above. Use the DieTester.java file to test your object in development. You do not need user input; simply hardcode the values using parameters.
4. Upon completion, your program should execute exactly like the program test run. Test your program several times with different integer values, to make sure your program works correctly.
5. Use screencastomatic.com to record a two-minute screen capture video of your program demo, with the following requirements.
   1. Introduce yourself, and greet the teachers, TA, and students.
   2. Scroll through your code, making note of the assignment requirements and how you solved them.
   3. Run the test data used on the assignment write-up.
   4. Run the program two more times with other test data, and ensure it works correctly.
   5. Thank your audience and indicate that the video is complete. You may have some sort of fun ending to entertain the viewer.
6. Complete the self-evaluation of your assignment. This will be the same rubric used to evaluate your work. The self-evaluation may be turned into the instructor the following week.
7. Create a zip file with your program file and screen capture video. Make sure all necessary files are in the submission, and no unnecessary files. Submit the zip file to the appropriate dropbox on canvas by the due date.

***Grading***

You will be graded on the following criteria:

*Effort* Creating the class and tester for the Die object.

*Code Formatting* Entering correctly formatted file and method documentation

*Output Readability* Organization and flow of the output, follows test-run perfectly

***Evaluation Instructions (Summary)***

On Monday after the due date, the solution and all student submissions will be made available for download. Find three students in class to evaluate your work, and you are to evaluate their work. You will be given the rubrics similar to the following page to evaluate their work.

***Assignment 1 Grading Rubric Programmer Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***CS200 -------------SELF EVALUATION-------------***

Program Testing

\_\_\_\_\_\_ Execution

Does the program execute correctly?

-20 for inexecutable code

\_\_\_\_\_\_ Output

Does the video correctly demonstrate the execution of the program?

-20 for altered or fake output

\_\_\_\_\_\_ Test input/output

Does the program work correctly for the given test input from the assignment write-up?

-2 to -10 for errors in output

\_\_\_\_\_\_ Other input/output

Test the code for other combinations of data, does it work correctly?

-2 to -10 for insufficient testing

\_\_\_\_\_\_ Is the output formatted exactly like the test run?

-2 to -5 for incorrect formatting

-2 to -10 for elements missing

Submission Requirements

\_\_\_\_\_\_ Program

Does the submission have all required files to execute?

-10 for missing any necessary program file(s)

\_\_\_\_\_\_ Video

Is there a video demonstration included in the submission?

-10 for missing video

Does the video include the five requirements (intro, code, given test, other test, thanks)

-2 to -9 for insufficient video or missing elements

\_\_\_\_\_\_ Program Style

Does the program follow the specified style guidelines including indentation, variable names, and blank spaces?

-2 to -5 for insufficient attention to program style

Documentation Requirements

\_\_\_\_\_\_ File documentation

Is there correct documentation at the top of the file following the documentation guidelines in the writeup.

-10 for no documentation

-2 to -9 for insufficient or incorrect documentation

\_\_\_\_\_\_ Spelling/grammar

Is the documentation properly written and free of spelling and grammar errors?

-1 to -5 for spelling and grammar errors