ICT-4361 Homework 3b

Purpose

This exercise to to provide you an opportunity to extend the previous exercise, and generalize a new client class to reuse your existing code, and generate statistically interesting information. Extending our dice exercise will show how to create a new class which uses arbitrary collections of the Dice class. A DiceStatistics class will run combinations of Dice, collect sample runs, and print the resulting statistics.

Sum of two dice, one with 6 sides and one with 8 sides

What to Hand In

Please hand in a listing for each program requested, formatted in an easy-to-read style, and following our Java conventions.

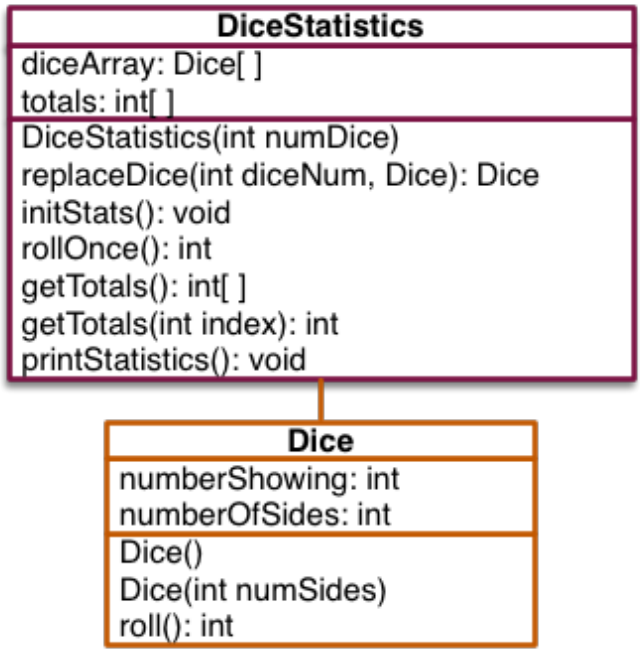
Ensure your name, and the name of the file is available in a comment at the top of each submitted file (excepting screenshots).

Also, ensure that you have a sample of the output from the program.

If your program fails to compile, hand in your error listing as your output.

For each question asked, provide one or two sentences summarizing your answer. Please be both complete and succinct.

Submit the files comprising your submission in a "zip" or similar archive through the assignment's submission button. Preferred file types in the archive are .java, .txt, .jpg, and .png

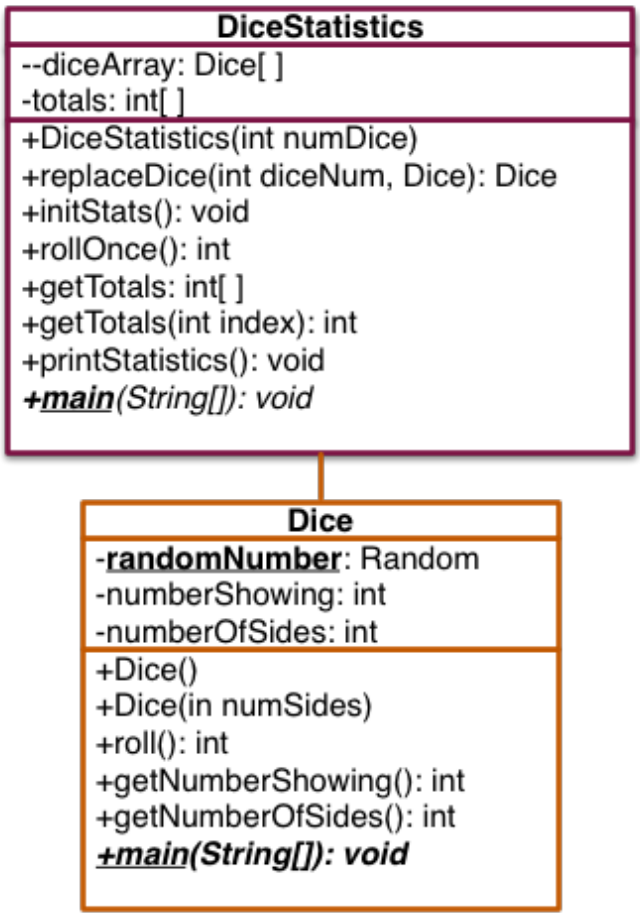


Summary class diagram for DiceStatistics

Problems

I. Create a class for the collection of statistics about our Dice.
   1. Create (or modify) a DiceStatistics class to allow the programmer to:
      - create a DiceStatistics object. Method: `DiceStatistics(int numDice)`.
        You may *optionally* **also** create a zero parameter constructor for DiceStatistics (Method: `DiceStatistics()`) which creates a DiceStatistics object with 2 (two) Dice.

- create the number of dice from a constructor parameter.
  Method: `DiceStatistics(int numDice)`
- store a provided Dice into a slot (index) in an array of Dice in the DiceStatistics object.
  Method: `replaceDice(int diceNum, Dice)`
  The method should return the previous Dice in the array slot (if any); if there was none, it will return null.
  The method should ensure that the requested slot is legal (index greater than or equal to zero, and less than the length of the aray).
  Also note that each time a Dice is changed in the DiceStatistics, the `totals` array must be recreated/reset.
- roll each of the Dice by invoking the roll method on each Dice in the array, and returning the sum total of all the Dice.
  Method: `rollOnce()`
  This method shows the power of delegation, used frequently in Java.
- print the statistics to the standard output (System.out).
  Method: `printStatistics()`
- reset the statistics (e.g., before making a separate run). This needs to set all the totals to zero.
  Method: `initStats()`
- Keep track of the number of times each totals (sum of all the Dice in the DiceStatistics array) occurs, based on the return value from rollOnce, in the same way we did last week.
  Manage array `totals`



Detailed class diagram DiceStatistics

2. Create or modify the `main` of the `DiceStatistics` class to:
   - create a DiceStatistics object
   - create different kinds of dice (using the new `Dice` constructor in the process), and place them in the `DiceStatistics` object created in the previous step.
   - call `rollOnce()` some large number of times to accumulate statistics.
   - print the statistics (`printStatistics()` to show the results of the current run.
3. Capture the output of printing the totals statistics, using DiceStatistics, for a pair of default dice, with at least 100,000 rolls.
4. Capture the output of printing the totals statistics, using DiceStatistics, for three Dice of 3, 4, and 5 sides, respectively, with at least 100,000 rolls.

II. In a short text document, or in comments in your code, describe the most popular total for each of combinations of Dice, and its percentage.

Notes

- You will need to copy (or otherwise ensure access for) last week's Dice class into this assignment. This might require you to change the package name to preserve the previous assignment's Dice class.
  Except for the package name, the Dice class should **not** be changed **in any way** for this exercise.
  You are permitted to use last week's sample solution Dice class with this week's assignment, with proper attribution.
- You may also want to adapt TwoDice.java to generalize it for the DiceStatistics class
- Our DiceStatistics constructor may be invoked as follows in the DiceStatistics main:
  `DiceStatistics myDiceStatistics = new DiceStatistics(2);`.
- To create the array of dice from the constructor `DiceStatistics(int numDice)`, and store it in your data member `diceArray`, use a statement like:
  `diceArray = new Dice[diceCount];`
  Of course, this doesn't create the Dice objects—it just creates an array big enough to hold two (diceCount) of them.
- To create a "default" (6-sided) die into slot "1", use code like:
  ```
  Dice myDice = new Dice();
  myDiceStatistics.replaceDice(1,myDice);
  ```
  Whereas, to create a 20-sided die, use a statement like:
  `Dice myDice = new Dice(20);`
- The possible totals can be computed from the number of sides of each of the dice.
- Remember to initialize your instance of `Random`!
- Remember that your code needs to be in a non-default package. Also, remember to include your name and the file name comments in all submitted files.

Evaluation

Criteria	Weight
DiceStatistics class, with an appropriate constructor	20%
DiceStatistics class data members implementated and properly initialized	20%
DiceStatistcs class member functions initStats, rollOnce, and printStatistics	15%
DiceStatistcs class member function replaceDice	15%
DiceStatistics main program with 2 default Dice and output	10%
DiceStatistics main program with 3 Dice of sizes 3, 4, and 5 and output	10%
Short textual answer to question II	10%