ICT-4361 Homework 3

Purpose

This exercise to to provide you an opportunity to extend the previous exercise, and to understand static and instance methods and attributes. Extending our dice exercise will show how to apply another layer of abstraction on top of our class. We will be moving our dice class to handle configurable sides, and allowing different dice to have different numbers of 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.

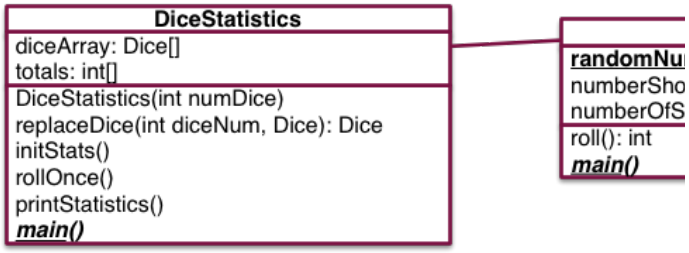
Problems

I. Create different kinds of dice

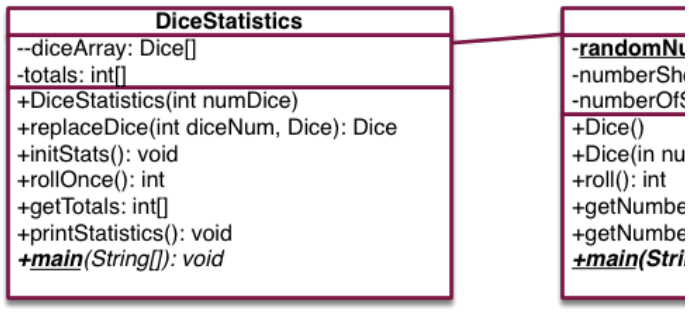
Note: You should not need to add any parameters to Dice roll(). This exercise makes only a small change to the Dice class we wrote last week.

1. Modify your `Dice` class to add a new private int field, `numberOfSides`.
2. Create (or modify) a default constructor for the Dice class that sets the number of sides to 6. It should set the numberOfSides field for that instance of dice to 6.
3. Add a new constructor that takes an integer argument that allows you to create a different number of sides for a dice. It should set the numberOfSides field for that instance of dice to the argument.
4. Make the `Random` instance a *static attribute*. That is, you will have one instance of the `Random` class that services all the Dice objects.
5. Modify your `roll()` method to take into account the `numberOfSides` on the Dice object.
6. Modify the `main` method of the `Dice` class, ensuring you account for the new range of totals, and test the new constructor.
7. Create or modify the DiceStatistics class to allow the programmer to:
   - create a DiceStatistics object. `DiceStatistics(int numDice)`. You may *optionally* create a zero parameter constructor for DiceStatistics (`DiceStatistics()`) which creates a DiceStatitics object with 2 (two) Dice.
   - create the number of dice from a constructor parameter. `DiceStatistics(int numDice)`
   - store a provided Dice into a slot (index) in an array of Dice in the DiceStatistics object. `replaceDice(int diceNum, Dice)` The method should return the previous Dice in the array slot (if any); if 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).
   - 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. `rollOnce()` This method shows the power of delegation, used frequently in Java.

Dice with different numbers of sides



Summary class diagram



Detailed class diagram

- print the statistics to the standard output (`System.out`). `printStatistics()`
- reset the statistics (e.g., before making a separate run). This needs to set all the totals to zero. `initStats()`
- Keep track of the number of times each totals (sum of all the Dice in the DiceStatistics array) in the same way we did last week using DiceStatistics.

8. 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.

II. What needed to change in order to modify your class this way? Were any of the changes unexpected?

III. Imagine that we wished to change the class for dice not numbered from one to the number of faces (say, a backgammon doubling cube)? What would need to change to support this approach?

Backgammon doubling cube: 6-sides with values 2, 4, 8, 16, 32, 64

Notes

- The new attribute in Dice should have `private` visibility.
- Our DiceStatistics constructor will be invoked as follows in the DiceStatistics main:
  `DiceStatistics myDiceStatistics = new DiceStatistics(2);`.
- To create an array of dice, use a statement like:
  ```
  int diceCount = 2;
  Dice[] diceArray = new Dice[diceCount];
  ```
  Of course, this doesn't create the dice—just an array big enough to hold two (diceCount) of them. To create a "default" (6-sided) die, use a statement like:
  `diceArray[0] = new Dice();`
  Whereas, to create a 20-sided die, use a statement like:
  `diceArray[1] = new Dice(20);`
- Keep track of the totals of the rolls of all the dice in your array, and print them at the end of the program (like DiceStatistics last week). The possible totals can be computed from the number of sides of each of the dice.
- Remember to initialize your instance of `Random`!

Evaluation

Criteria	Weight
New instance attribute	15
Modified constructor	15
New constructor	15
New static attribute	15
Main program and output	20
Short textual answer to question II	10
Short textual answer to question III	10