Farkle.java, scaffolded

Farkle is a dice game. The goal is to earn points based on dice rolls, and NOT Farkle (which means you didn't earn any points, and your turn is over).

We'll complete Farkle iteratively. You will be graded based on:

- Ability to code with one-dimensional arrays, evidenced by code and pseudocode (you can use a lot of the directions to help with that).
- Functionality. Ideally the Farkle game will work across the duration of the project. It may break as you incrementally change it, with each step. For this reason, please make Farkle1 (for step 1), then use Save-As to resave Farkle1 as Farkle2 (for step 2, including code left from step 1), and so on. These are all Java classes.

1. To play, six dice are rolled. This means, you will need to generate random numbers from 1-6. We'll store all of the dice rolls in an array that is 6 long.
   a. Write a class called Farkle. It should have a constructor, main(), and a run() method called from main() using an instance of Farkle. Create a field variable for the integer array, called roll. Instantiate roll, giving it size, in the constructor.
   b. Write a method called rollDice(). In this, first use a loop to assign each cell in the array the value zero. This may seem extraneous now, but you will need this to reset your roll array as you continue to play.
   c. Also in rollDice(), use a second loop to roll the dice to fill your array with random numbers. This loop in rollDice() should have the header:
      for(int times = 0; times<6-limit; times++)
      limit is a parameter of rollDice(), meaning it is a local variable. Inside this loop you will use Math.random to make a random number, assign it to the array, and print the value in that slot of the array.
   d. From run(), add the following code to the body, and see if rollDice() works:
      rollDice(0); //note zero sets limit to zero

Don't forget to plan (pseudocode) first, then write code into a program called Farkle1.java. Test it thoroughly before saving it as Farkle2.java and editing it to change/add more according to step 2.

2. Points and Farkling. *Now you are coding Farkle2.java.*
   a. Let's start with the simplest way to earn points - by rolling 1 or 5. 1 is worth 100 points, and 5 is worth 50. Write a method called calcPointsFromRoll() that uses a loop to look at what is in the roll array, and calculate points by counting up the number of 1s and 5s, then multiplying by their values.
   b. Make an array called table, which is an int array with size 6. Just let it have default values for now.
   c. In calcPointsFromRoll(), if the roll array has some point-generating value(s), also look at the table array and add the points from here (you'll need another loop, which should look nearly identical to the one you wrote for (b)) to generate the total points earned.
   d. In calcPointsFromRoll(), if the roll generates no points, you should not look at table, and the point total should be zero.
   e. Pass these points that you calculated from calcPointsFromRoll() to a new method, checkForFarkle().
   f. Farkling happens when a roll doesn't generate any points. Write checkForFarkle() which uses a parameter from calcPointsFromRoll to determine if you Farkled or not. If you Farkle, print a statement to indicate this, then return the boolean value *true* from checkForFarkle(). If you do not Farkle, print out the points earned and return false from chcekForFarkle(). Later, this *true* boolean will be used to stop the game, or *false* to continue.
   g. Think about the game play and what these methods do, then determine where you want to call these methods from, and in what order, so you can test this.

Run Farkle2.java over and over to make sure that eventually you can Farkle. Don't worry about when the game ends – for now, it should just roll once and tell you if you Farkled or what your score is.

3. It is quite likely that a player will not Farkle. Here's where the game gets interesting! A player can risk continuing to play. The player can opt to keep some of the die and roll again. You will ask the player what to keep, then use the numbers entered to fill a second array called table, which you set up already. You are now writing Farkle3.java
   a. Write a method called decide(). Ask the user which numbers (s)he would like to keep. For now, you can assume that what the user enters is true for what they rolled (i.e. (s)he will not lie). As soon as you have input from the user, use it to populate the table array. You will need to prompt the user specifically and may need to process what was typed so that the input can be interpreted as integers and stored in the table array.
   b. In decide(), print the table array that is in use (not zero values), with a little String context (you'll be printing several things in the program). In the same loop that can print, generate a count of how much of table is in use, then store this count as the local integer variable called inTable. Pass this variable to rollDice() to generate a new roll, then check to see that the appropriate number of dice have been rolled.
   c. Decide where you should call decide(), based on when this happens in game play.
   d. Modify decide() to handle edge cases: If the user types six values, then you will not need to roll again; the user is indicating that the score is good enough and the turn is over. If the user types no values (you can tell the user to type "done" or something), then the turn is also over. In both cases, table should only have zero values in it.
   e. Because you call rollDice() after running decide(), your software is now iterative. You should be able to keep playing, meaning:
      Roll – Determine score or farkle – Decide
      over and over. Go back to earlier steps and integrate needed changes to make the game stop as indicated previously. You can use the list below to help.
   f. Be sure you have a testing plan and have noted what you tested with and what works/doesn't work. This is the first version that requires a testing plan, because it is the first version with user input.

Test Farkle3.java. It should work now work iteratively. You can now test to see if Farkling, typing six values, or typing "done" (or equivalent) causes the game to end.

Test your code thoroughly for this step and the whole game play. Check on:

__ Dice rolls can be 1-6; each number can show up
__ Table is being populated and changes the number of dice rolled
__ Table can grow bigger and continue to reduce the number of dice rolled
__ Count in roll and number in table (except zero values) sum to 6 every time
__ A player can keep dice and re-roll over and over
__ A player can choose not to keep any dice, then the game ends
__ A player can keep all of the dice, then the game ends
__ Scores accumulate over time and are still accurate
__ No runtime errors occur, for instance Null Pointer Exceptions or Index Out of Bounds

Be sure also to re-read the directions. Your code should include pseudocode alongside so parts can be found easily. Be sure you are using field and local variables as described in the directions. Be sure all methods are named appropriately.