1 Exercises

For the following exercises, pair up with someone else in the class and create a project for each exercise. In your comment section at the top of each of the programs make sure that both of your names are listed in the Author line in the comments at the top of each program.

For each of the following create a new project with an appropriate name and then write a program that solves the given problem. Remember to do Shift+Ctrl+F to format the program, or Shift+Command+F on the Mac to format the code. Also remember the standard comments of at the top, Authors, Date, and Description.

As usual, you will be submitting through MyClasses the java code files and a Microsoft Word docx file, LibreOffice Writer odt, or a text file (which you can create with NotePad++) which contains the output of at least three runs of each program on different data inputs. Remember that you can copy and paste output from the Eclipse console area to the word or text program. Only one of you will need to submit the files.

1. In this program you are going to empirically find the probability of rolling different values with two 6-sided die.

   Let the user input the number of rolls as before (check it for errors, that is you cannot have 0 or fewer rolls), then have the program roll the dice that many times looking for a 2, then repeat the number of rolls and look for a 3, then repeat the number of rolls again and look for a 4, and so on up to 12. Have the program print out the probability of each. A sample run is below.

   ```
   Input number of rolls: 1000000
   Target =  2  Probability = 0.027861
   Target =  3  Probability = 0.055617
   Target =  4  Probability = 0.083493
   Target =  5  Probability = 0.11097
   Target =  6  Probability = 0.138933
   Target =  7  Probability = 0.16713
   Target =  8  Probability = 0.138379
   Target =  9  Probability = 0.111412
   Target = 10  Probability = 0.083153
   Target = 11  Probability = 0.055637
   Target = 12  Probability = 0.027935
   ```

2. Most computer games use physics engines to take care of many aspects of game play. The most work that a physics engine usually does in a game is collision detection. Determining when the user (or object) hits a wall, when the laser blast or bullet in a FPS hits a target, etc. Collision detection is a fairly difficult problem to solve and is beyond the scope of this class, but another, more accessible, use of a physics engine in a game is to apply gravity to an object.

   You have probably seen this in high school physics or possibly as an example in a mathematics class like Calculus. If not, we will discuss the equation fully. Say you take an object, like a tennis ball or small rock, and through it up into the air. The object will have a particular height off the ground at any time after you toss it until it eventually hits the ground. The object will increase in height, until gravity overcomes its initial velocity, and then it will start to fall to the ground. To find the height of the object at any time t in seconds after you toss the object is given by the equation,

   $$d = -\frac{1}{2}gt^2 + v_0 t + d_0$$

   where d is the current height of the object, g is the acceleration of gravity, v_0 is the initial velocity and d_0 is the initial height. The acceleration of gravity on Earth in English units is 32.17405 feet/sec^2, so velocity is in feet/sec, distance is in feet and time is in seconds. Also note that we usually consider the positive direction as up. So a positive d_0 would indicate that the initial position of the object is above the ground and a positive v_0 would indicate that the initial velocity is in the upward direction. Along these lines, gravity would be pulling the object down and hence the negative sign on that term.

Write a program that asks the user for an initial velocity and initial distance. Have the program print out a chart of heights of the object every 0.1 seconds until the object hits the ground. When the object does hit the ground have the output state that. Also, have the time to one decimal place and the height to three, decimal points lined up. Use whichever type of loop structure will work best for this problem. A sample run of the program is below.

```
Input the Initial Height (in feet): 10
Input the Initial Velocity (in feet/sec.): 25

 Time          Height
 ----          ------
  0.0          10.000
  0.1          12.339
  0.2          14.356
  0.3          16.051
  0.4          17.424
  0.5          18.475
  0.6          19.204
  0.7          19.611
  0.8          19.696
  0.9          19.459
  1.0          18.900
  1.1          18.019
  1.2          16.816
  1.3          15.291
  1.4          13.444
  1.5          11.275
  1.6           8.784
  1.7           5.971
  1.8           2.836
  1.9      Hit the ground.
```