YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 427a: Object-Oriented Programming *Handout #4*
Professor M. J. Fischer *September 19, 2010*

Problem Set 2

Due before midnight on Monday, September 27, 2010.

1 Assignment Goals

1. To write an application using more than one class.
2. To learn how to use random numbers for simulations.

2 Problem

A die is a physical object, often cubical, for generating random numbers for use in various kinds of games. (See `http://en.wikipedia.org/wiki/Dice`.)

One widely known betting game played with dice is called *craps*. It uses two standard six-sided dice, each bearing the numbers 1, . . . , 6 on its six sides. The *shooter* rolls two dice one or more times until the round is either won or lost. On each roll, the *number rolled* is the sum of the numbers showing on the two dice and hence is in the range $\{2, \ldots, 12\}$.

On the first roll, if the number rolled is 7 or 11 (called a "natural"), the shooter wins immediately. If the number rolled is 2, 3, or 12 (called "craps"), the shooter loses immediately. Otherwise, the number rolled becomes the *point*. From then on, the player keeps rolling until either the number rolled equals the point, in which case the round is won, or the number rolled is 7, in which case the round is lost.

In this assignment, you are to determine experimentally the probability of winning in craps (as I've described it here) and also the expected number of rolls until the round ends. To do this, you will simulate many rounds of craps. For each round, you will compute the outcome (win or lose) and the number of rolls used. The probability of winning is estimated to be the total number of wins divided by the total number of rounds. The expected number of rolls per round is estimated to be the average over all rounds played of the number of rolls until the round ends.

Your program should take two command line arguments: the number of rounds to simulate and a seed for the random number generator. If the second argument is omitted, the seed should be taken to be the result of a call on `time(NULL)`. In either case, your program should print out the number of rounds simulated and the seed used for the random number generator. This way, your experiments are repeatable, and the TA should be able to get the exact same answer to each experiment as you do by running it with the same seed.

The program should then perform the simulation and print its output: the estimated win probability and the expected number of rolls.

The program should consist of three (or more) classes and a main program.

- Class `Dice` models a set of k dice, each with n sides. Here k and n are fixed when the class is constructed. An instance of `Dice` stores the number showing on each of the k dice. It supports a public function `roll()` which randomly rolls each of the dice, and a function `outcome()` which returns the sum of the numbers showing on the dice.

- Class `Craps` models a round of craps. It should have private data members that keeps track of the game state: is the game won, lost, or continuing, how many rolls have taken place so far, has the point been determined yet, and if so what is it? It should have a public function `void play()` that keeps rolling the dice and updating the game state until the round is over. It should also have public functions that return the win/loss status of the round and the number of dice rolls. Internally, it should have a function `oneRoll()` that performs one roll of the dice and updates the state variables appropriately.

- Class `Simulator` controls an experiment. It should have private data members that specify the total number of rounds to run in the experiment and the number of rounds run so far. It will also need private data members to keep track of the number of winning rounds and the total number of dice rolls used. It has a public function `run()` that carries out an experiment and a function `print()` to print the results of the experiment (fraction of wins and average number of rolls per round).

All three classes should have a print function for debugging that prints out the information stored in the data members in a readable way.

Corresponding to each class should be a pair of files: an interface file and an implementation file. The main program should be in its own file. It should have nothing in it except for code to print the banner, process command line arguments, create and run the simulator, print the results of the simulation, and perform any necessary cleanup before termination.

3 Program Notes

To generate a random roll of a single die, you can use the function `RandomUniform()` below:

```
// A method to generate a random number uniformly over the range
// {0,...,n-1} using the library function rand().
int RandomUniform(int n) {
  int top = ((((RAND_MAX - n) + 1) / n) * n - 1) + n;
  int r;
  do {
    r = rand();
  } while (r > top);
  return (r % n);
}
```

The purpose of this code is to make all numbers in the given range equally likely. Note that it is not sufficient to just take `rand()%n` since if n does not divide `RAND_MAX + 1`, some numbers will have a greater probability of being output than others. For example, if `rand()` were to produce numbers in the range $\{0, \ldots, 9\}$ and we wanted numbers in the range $\{0, \ldots 3\}$, then reducing each of the numbers in the range $\{0, \ldots, 9\}$ mod 4 gives the sequence $0, 1, 2, 3, 0, 1, 2, 3, 0, 1$. We see that 0 and 1 each occur 3 times, whereas 2 and 3 each occur only twice. Thus, 0 and 1 are each generated with probability 0.3 and 2 and 3 are each generated with probability only 0.2. To be uniformly distributed, all probabilities should be 0.25.

4 Deliverables

You should submit this assignment using the `submit` script that you will find in `/c/cs427/bin/` on the Zoo. Remember to submit the following items:

1. Source code and header files.
2. A `makefile` and any other files necessary to build your project. (If you're using Eclipse with the default settings, the makefiles will be found in the directory `Debug`.)
3. The output of several runs of your program using different seeds and different numbers of experiments.
4. Any test programs along with test input and output that you used to verify that your program was operating correctly.
5. A brief report named `report.txt` (or any other common format such as `.pdf`) describing the design choices you made in implementing the required code extensions. You can also put any other information here that the TA should know about your program.