CS100J Spring 2003 Assignment 5 Maintaining Duckpin Bowling Scores I (maintaining a frame)

Submit your solution on the course management system by the deadline: Saturday, 18 October, 23:59

You may work in groups of 2. But PLEASE form your group well before you submit!!!!

Before we discuss this assignment, let us discuss the timetable over the next few weeks:

09 October (Thursday). Lecture on loops.
14 October (Tuesday). Fall break --no lecture.
14-15 October (Tuesday-Wednesday). no lab, because of fall break.
15 October (Thursday). Lecture on loops. A handout will describe Prelim II.
18 October (Saturday) Assignment A5 due by midnight.
19 October (Sunday) Review session for prelim II, time and place to be announced.
21 October (Tuesday) Lecture on loops.
21 October (Tuesday). 7:30PM. Prelim II.
21-22 October (Tuesday-Wednesday) labs, as usual, in the ACCEL lab.

The official lab is canceled for Wednesday, 15 October. However, some TAs and consultants will be there during the whole time (12:20 to 4:25) in order to help individuals who want to work on assignment A5. Also, the consultant will be in the Carpenter basement. Finally, Gries will be in his office from 01:00 to 16:00, and people with laptops can visit him get some help in writing the assignment.

Thus, if you have studied the assignment and done some initial work on it, you should be able to get it done on Wednesday. If you do your programming, compiling, and testing incrementally, as we suggest below, you should have no trouble with the assignment.

Introduction

We will have a series of assignments that end up in a program that maintains statistics on bowling games for any number of people. The group of assignments will exercise your knowledge and skill in a number of topics. (Some of the topics you don't know about yet! Don't be scared. This is an overview of what is to come, not what you need for assignment A5.)

- Keeping score in bowling and formatting the data requires attention to detail. The logic is just messy enough to cause you to think carefully.
- You will have to write code that reads from a file --the data for a bunch of games played by different people. This also requires the ability to break a String up into its various pieces.
- The games are kept in a table that is implemented as an array, so you will have practice with this aspect of arrays.
- Your program will have to sort the set of games on several keys --at least the total scores, names of people.
- Your program will print statistics on the games as a whole and for a particular person as well, e.g. the mean and median scores, the standard deviation, etc. So, you get more practice with arrays.
- You will use GUI JLiveWindow for some input.

Duckpin Bowling (a.k.a. Candlepin Bowling)

Duckpin bowling is a variant of ten-pin bowling, except that the ball and pins are significantly smaller. The size is such that it is very difficult to knock down all ten pins withonly two balls. Hence, in duckpin bowling you are given three balls per frame. Those of you familiar with Candlepin Bowling will find the rules familiar --they are in fact the same! The main difference between Candlepin and Duckpin bowling is the size of the pin used.

A game consists of ten *frames*. In each of the first 9 frames, you roll one, two, or three balls:

- A frame starts with ten pins up.
- If the first ball knocks down all ten pins, it is called a *strike* (written "**X**"), and you don't get a second or third ball. As in golf, the better you are, the less you get to play.
- If the first ball knocks down less than ten pins, you throw a second ball. If together the two balls knock down ten pins, it is called a *spare* (written"/"). You don't get a third ball.

- If there are still pins up after the second ball, you throw a third ball. You do *not* get a strike or spare for knocking down ten pins using all three balls.

Here is how the information is recorded for a frame (the tenth frame can have more in it):

strike:	**X**
spare with 5 on the first ball:	5/
Three balls, say 4, 4 and 1	445

In the tenth frame, you *always* roll three balls. Whenever all 10 pins are knocked down, they are set up again. For example, you can throw three *strikes* in the tenth frame: **XXX**; or a *spare* and a *strike*, e.g. 5/**X**; or a *strike* and no *spare*, e.g. **X**45; or no strike or spare at all, e.g. 711, or 0010 --i.e. two 0's and a 10. These examples don't include all possibilities.

The scoring is complicated:

- In any frame, if you use all three throws, the score is the number of pins knocked down.
- In any frame but the tenth, if you get a spare, the score is 10 + what you get on the next ball (in the next frame).
- In any frame but the tenth, if you get a strike, the score is 10 + what you get on the next two balls.

Below, we show a game for Mary and one for Harry. The first line of each has the person's name together with what they rolled in each of the ten frames. The second line contains the running total score after each frame --this is how you keep a score.

Mary got a strike in every frame and three strikes in the last frame, for a perfect score of 300. Since Mary got a strike in her first frame, her score there is 10 + what she rolled on the next two balls, which were also strikes. Harry got just 9 pins in the first frame. In the second frame, he got a spare, so the score is 10 plus the next ball, which was a strike in the next frame, so his score for the second frame is 20. And so it goes.

Mary	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**XXX**
	30	60	90	120	150	180	210	240	270	300
Harry	450	4/	**X**	6/	622	**X**	**X**	**X**	**X**	5/2
	9	29	49	65	75	105	135	160	180	192

Class Frame

We give you a skeleton of class Frame, which is the only class you have to write. It compiles, but it contains only what the user would see --specifications of methods and their headers-- together with method bodies that are either empty or contain a simple return statement. We provide four well-specified fields. You have to fill in the bodies of all the methods. You may add extra private methods if they help --make them private because they should not be seen/usable outside the class.

We discuss the various pieces of class Frame. We urge you to write them and test them thoroughly, in the steps provided below. Don't go on to the next step until you are absolutely sure the first is correct. **Before you begin, read the specs of all the methods so that you know what you have to do.**

Step 0. Constructors and function toString

First, fill in the bodies of the three constructors.

Second, fill in method toString.

Third, test the four methods thoroughly, by creating instances of the class (in the Interactions pane) and calling toString.

Step 1. Write function card, which produces the information about balls rolled in the frame. The result MUST be FOUR characters long, as specified. Append blanks, if necessary, to get it four characters long.

This can be a messy function. We suggest that you write a private function to handle the case of the tenth frame, which is totally different. Then, in function card, test whether it is the tenth frame and return its value immediately if it is (by calling the private function).

Suppose it is not the tenth frame. We suggest that you handle one case at a time, returning its value immediately. For example, start with the case that the player uses all three balls: if the three balls are, say 5, 1, and 4, then return "514 ". Next, handle the case of a spare, and finally handle the case of a strike. AFTER EACH CASE, TEST IT. Do incremental programming, compiling, and testing. That way, you remain under control and you have confidence in what your are doing.

Write and test the case of the tenth frame in the same way.

Step 2. Write function score, which returns the score for the frame. This one is weird because the score in the frame may depend on what happens in the next one or two frames! Therefore, the function has two parameters, which are the next one or two frames. Here is how you can determine which frame it is. If both parameters p1 and p2 are **null**, it is the tenth frame. If parameter p1 is not **null** and parameter p2 is **null**, then it is the ninth frame. Otherwise, the frame is in the range 1..8.

Program, compile, and test incrementally --as done in step 1.

Step 3. Submit your file Frame.java, by the deadline.