Computing and Algorithms I: Programming Assignment 6

CS-101 **Winter 2023**

In this final programming assignment (huzzah!), you will demonstrate your knowledge of all the material in CS-101 by writing a program for playing a dice game called *Bulldog*.

Background

The game of *Bulldog* is played by several players, who each take turns rolling a single six-sided die as many times as they wish, adding all of their roll results to a running total. However, if the player rolls a 6, the player loses their score for that turn.

Game Play

On each turn, a player repeatedly rolls a six-sided die until either a 6 is rolled or the player decides to end their turn.

- If the player rolls a 6, their turn is over; they score nothing, and control passes to the next player.
- If the player rolls any number other than a 6, the number is added to their score for that turn.
  - If the player chooses to continue, they roll the die again, applying the rules above.
  - If the player chooses to end their turn, they add their accumulated turn score to their overall score, and control passes to the next player.

The first player to score at least 104 points wins the game.

The most interesting part of the game, of course, is the strategy that each player uses to decide when to end their turn. A conservative strategy ends turns early, taking guaranteed small scores. An aggressive strategy favors longer turns, balancing the risk of losing all accumulated points against the reward of large scores.

Design Requirements

Your program must satisfy the following design requirements.

The Player Class

You have been provided with an *abstract* `Player` class. The `Player` class contains the following instance variables:

- `private String name;`

  This instance variable contains the name of the `Player`.

- `private int score;`

  This instance variable contains the score earned by this `Player` during this game.

The `Player` class contains the following public methods:

- `public Player (String name)`

  This is the constructor for the `Player` class. It initializes the name of this `Player` to the given value, and initializes the score of this `Player` to 0.

- `public String getName()`

  This is the public accessor method for the instance variable `name`.

- `public int getScore()`

  This is the public accessor method for the instance variable `score`.

- `public void setScore (int score)`

  This is the public mutator method for the instance variable `score`.

- `public abstract int play();`

  This is the *abstract* method that represents one turn of the `Player`. The method plays one turn the game for the `Player`, using the strategy of the `Player`. When the turn of the `Player` is over, the method returns the score earned by the `Player` during that turn: 0 if a 6 was rolled, or the sum of the rolls if a 6 was not rolled.

  This method is *abstract*; it will be overridden by subclasses which you will be responsible for writing.

Subclasses of the Player Class

You will write *four* subclasses of the `Player` class, as described below.

Each class will provide a constructor, that takes a `String` as an argument, and invokes the parent constructor (using `super()`) with that `String`. Your constructor may accept additional arguments as desired.

Each class will override the `play()` method. The basic structure of the `play()` method will be the same. The method will generate a random number in the range [1−6], representing the result of the roll of a six-sided die. If the roll is 6, the method will end and return 0 as the score for that turn. If the roll is not 6, the method will decide whether or not to continue rolling, based on the strategies outlined below.

The four subclasses of the `Player` class you will write are:

- `HumanPlayer`

  Objects in the `HumanPlayer` class will print the cumulative score earned so far in this turn, and ask the user to choose whether or not to end the turn. The object will use `System.in` to read the user's response from the keyboard.

- `RandomPlayer`

  Objects in the `RandomPlayer` class will choose to continue or end the current turn randomly, with a 50% probability for either choice.

- `FifteenPlayer`

  Objects in the `FifteenPlayer` class will always choose to continue the turn until a score of at least 15 is reached, at which point the object will choose to end the turn. (Note that, of course, if a 6 is rolled, the turn ends with a score of 0.)

- `UniquePlayer`

  You will design a unique strategy for the `UniquePlayer` class that differs from any strategy described here.

In all cases, the `play()` method should print relevant information to `System.out` at each step: the roll of the die, the strategy decision made, the cumulative score earned after each roll, and the final score at the end of the turn.

Sample Player Class

As an example, you have been provided with a `WimpPlayer` class, which is a subclass of the `Player` class. Objects of the `WimpPlayer` class always choose to end their turn after the first roll.

(You may not use `WimpPlayer`'s strategy as the strategy for your `UniquePlayer` class.)

Program Requirements

Your program will, as usual, contain a `Prog6` class with a `main()` method. (You may write other methods in the `Prog6` class to assist your `main()` method.)

Your program will begin by asking the user for the number of players who will play the game. Your program will then create that number of players, storing them in a variable of type `ArrayList<Player>`. The players stored in that `ArrayList` will be chosen by the user; for each player, your program will ask the user which class to instantiate. Multiple instances of a given class can be used. For example, the user could choose to create two `HumanPlayer` objects and two `RandomPlayer` objects. (Note that the user could choose to play the game solely with automated players.)

Your program will then play the game of *Bulldog* with the players created in the previous phase. Your program will repeatedly iterate through the list of `Player` objects, calling `play()` on each object to obtain the player's score, then calling `setScore()` to update the player's score appropriately. When any player's score reaches at least 104, the program will immediately end the game for all players.

Your program will print appropriate information at each step, including: which player is taking a turn, the result of that turn, and every player's score after each turn (including the last turn).

Style Requirements

See the *Style Guide* for information on style requirements for multi-class programs.

Submitting Your Program

Before 11:59:59 p.m., Tuesday, 21 March 2022 (11th Tuesday), you must upload a zip archive to the course Blackboard assignment for Programming Assignment 6. This zip archive should contain all of the Java source code files for your program. (Please include the `Player.java` file provided to you.)

Notes

1. ***Start Early!*** This project requires you to pull together everything you have learned this term. Starting early also allows you to consult with the instructor during scheduled work days and student office hours.

2. The due date for this program is the last day of regular classes; late submissions become hard to handle, and also interfere with your ability to study for final exams.

Bulldogs, Pigs, and Academic Integrity

The game of *Bulldog* is based upon the game of *Pig*, first described in print by John Scarne in 1945. The differences between *Bulldog* and *Pig* are designed to honor Kettering University's heritage:

- In *Pig*, a roll of 1 ends the turn with a loss of all accumulated points. Since Kettering University has had five different names during its history, it has no sixth name, and 6 is the roll that ends the turn.
- In *Pig*, the game is played to 100 points. In *Bulldog*, the game ends at 104 points, in honor of the founding of Kettering University 104 years ago (in 1919).
- The `FifteenPlayer` strategy in *Bulldog* honors the fifteen different minors currently available to Kettering students.

The game of *Pig* is a favorite of computer science instructors worldwide to use as a programming project. Consequently, there are dozens of solutions for *Pig* available online, written in a variety of programming languages (including Java). *Bulldog* has been designed to differ from these solutions in significant ways, in order to encourage you to write your own solution.

The course syllabus for CS-101 states:

> Unless otherwise stated in the assignment, programming assignments should be *entirely your own work*. You are permitted to discuss homework assignments with fellow classmates in a general nature; you are *not* permitted to jointly develop algorithms and/or code. You are not permitted to copy algorithms and/or code from other students or from outside sources, or to give or loan code to other students. You are not permitted to use code-generating programs to assist you in writing your code.

In case this is not sufficiently clear: using online solutions to *Pig* to write your solution to *Bulldog* is ***explicitly forbidden***. Violations will be treated extremely severely, up to and including course failure.

Acknowledgements

The following webpages were used in creating this assignment:

- `https://en.wikipedia.org/wiki/Pig_(dice_game)`
- `https://en.wikipedia.org/wiki/Kettering_University`
- `https://www.kettering.edu/academics/program-finder`