Pig

Using Object Oriented Programming techniques, create a text-based (console) version of the game of "Pig". The rules are simple: two players roll dice and try to reach a certain number of points first (traditionally 100).

Each player rolls a die until they either decide to "hold", adding the total of the die rolls to their score, or a 1 is rolled - ending their turn and adding zero points to their total. **Note – rolling a 1 does not cause the player to lose already accumulated points, only *that turn's* points**. Click here to see a sample run of the program.

WARNING!!!!!

Method and instance variable names must match or the Gradle tests WILL NOT WORK!!!

This program will require the following classes:

The **Die.java** class is responsible for modeling dice; a Die object is what will be "rolled" in our game (by a Player). The Die class will produce random numbers to simulate the rolling of a die.

Field(s)

- `private int numFaces` – Stores the number of faces on the Die object. Used for "rolling" (returning a random number from 1 to `numFaces`).

Methods

- `public Die()` – A "default" constructor where `numFaces` is initialized to 6 by default.
- `public Die(int numFaces)` – A secondary constructor where the `numFaces` instance variable is initialized to the value of the parameter.
- `public int roll()` – Returns a random integer, from 1 to `numFaces`.
- `public int getNumFaces()` – Returns the value of `numFaces`.

Save everything (cmd + shift + s). Run the Gradle tests for Die_Tests.java
(Right click Die_Tests.java -> Run As -> Gradle Test)

If everything passes, go to GitHub Desktop, commit changes and click "Push to Origin".

The **Player.java** class is responsible for modeling a player (in our game) and creating Player objects, which have a `name` and a current `score`. A Player (object) takes a turn, rolling until they get a 1 or decide to hold. A game of Pig will require two Player objects.

Field(s)

- `private int score` – The player's current score (points).
- `private String name` – Stores the player's name, such that your program can call each player by name when announcing whose turn it is (when a Player object takes a turn) and displaying scores.

Methods

- `public Player(String name)` – Constructor, responsible for initializing Player's fields (the variables declared below the class header) and making Player objects. The Player's `name` should be initialized to the value of the parameter. The `score` instance variable doesn't require a parameter, but should definitely still be initialized.
- `public String getName()` and `public int getScore()` – Getter methods, return the value of `name` and `score`.

  Save everything (cmd + shift + s). Run the Gradle tests for Player_Tests.java
  (Right click Player_Tests.java -> Run As -> Gradle Test). Note: takeTurn() will fail because you haven't done it yet. That's ok.

- `public void takeTurn(Die die)` – Contains most of the logic for the players' turn (printing scores, prompting the player for their choice (rolling / holding), etc.). Every "round", a player will take a turn, adding to the player's `score` only if they "hold" without rolling a Pig. This method should use the `die` object parameter that will be supplied by the PigGame class to "roll". Don't forget to prompt the user with their roll and round total throughout their turn.

  ```
  /* Every time the Player "rolls" (calls the roll() method on the die object), the
  method returns a new random integer.  Multiple calls will yield multiple random
  numbers.  Save what it returns into a variable! */
  ```

  Note - a Player simply takes a turn. The Player class does not worry about printing the rules of the game, or whether the Player has reached the game's `goal` amount of points - that is PigGame's job. A Player *could* continue rolling after reaching the game's goal, but that obviously wouldn't be too smart.

  Save everything (cmd + shift + s). Run the Gradle tests for Player_Tests.java
  (Right click Player_Tests.java -> Run As -> Gradle Test). Note: if the takeTurn() fails, it will show what you printed and why it thinks it failed.

The **PigGame.java** class represents an actual game of Pig; its job is to create Player objects and have them continue playing until one of them wins. A PigGame object is an *actual game instance* - this class will contain the players, the die, and the main "game loop".

Field(s)

- `private Player player1` – A Player object, the first player in the Pig game.
- `private Player player2` – A Player object, the second player in the Pig game.
- `private Die die` – A Die object, used by the players to roll (during the player's turn)
- `private int goal` – The value of `goal` will determine how many points it takes to win the game. When a Player's `score` reaches `goal` points, the game will be over.

Methods

- `public PigGame(int goal)` – Constructor, responsible for initializing PigGame's instance variables. Initialize the `die` object using the Die class' default constructor (unless you wish to play with a different number of faces). This method should also get the players' names from the user, to initialize the `player1` and `player2` instance variables. Yes… that mean's you'll need a Scanner in your constructor to get this info from the user. You can also ask the user if they want to use a die with something other than 6 sides (If you want).

  Basically what it does
  - Initializes `goal` to the parameter
  - Asks the user for the player's names and initializes the `player1` and `player2` objects

  This method will NOT prompt the user for a goal number of points; that value is supplied when a PigGame instance is created (to the constructor). If you want to prompt for a goal, do it in the GameRunner class.

- `public void play()` – This method should contain the main "game loop" that will continue to run until a player has won the game (reached `goal` number of points after a player finishes their turn). Make sure to supply the PigGame's `die` object to the Players when they take a turn.

  Basically what it does
  - call `printInstuctions()`
  - loop throguh player 1's turn, player 2's turn, and a check to see if someone won.
  - print the winner

- `public void printInstructions()` – Should print the rules / instructions for the game. This method should be invoked (called) at the beginning of every game.

GameRunner.java will create a PigGame instance (object) and begin playing. This is the "runner" class, it contains one method:

- `public static void main(String[] args)` – This method will create a PigGame object (e.g. `PigGame game = new PigGame(100)`), passing in the `goal` number of points (prompt the user for this value, such that they can play a game to whatever score they want). To begin the game, call the `play()` method on the PigGame object.

  - The value passed in to PigGame's constructor determines how many points it takes to win the game.

Note how simple the PigGame class is – it uses Player objects to avoid having to repeat code! After you finish all classes, test your code by running the `main()` method in the Runner class. A PigGame object is an actual game of Pig (with Players that have score and name, etc.)!

Had we not broken the code up into "units" (classes / objects), a single class containing the entire program would be hundreds of lines long, and very hard to debug! By using classes, if you have a problem with a particular area (e.g. the Player's `takeTurn()` method isn't printing something it should), you immediately know *exactly* where you need to make a change. Easier debugging and avoiding code duplication are just a couple advantages of OOP (object oriented programming).

(Advanced) Adding AI

Add a class called AI (i.e. artificial intelligence) that will represent a computer player. Give the player an option to play against the computer. Good AI will have multiple difficulty levels (in other words, other methods that get called when the AI takes a turn). Some options:

- The AI will stop rolling after `n` number of scoring turns.
- The AI will stop rolling after scoring `n` number of points.
- (Harder) The AI will roll more aggressively if it's behind.