# CS4233 Programming Assignment
# The Pig Game

Gary F. Pollice

October 24, 2020

## Introduction

*Pig* is a simple dice game dice game invented by John Scarne in 1945. [1]
This assignment has these objectives:

- let you practice TDD and the process described for this course
- learn or improve your knowledge and application of interfaces in Java
- improve your understanding of abstraction and separation of concerns
- learn or improve your knowledge of lambdas in Java (depending upon your implementation)

You should read this document carefully and ask any questions about it in the discussion thread for this assignment that you will find on Canvas. We look at these several times a day and we expect students to do the same. If you have an answer to a post in a discussion, feel free to provide the answer unless the discussion explicitly asks you not to. You have the responsibility to keep up with the discussion board.

Download the starting code for this assignment and use it as your base for starting work. Try to follow the code style in the base code or format the code to your style standards. You can set formatting standards in Eclipse and you can use `SHIFT-CMD-F` to quickly format all or some of your code. The course Canvas site has the formatting conventions that we use, feel free to install them in your Eclipse. Take time to look over the code and the initial design described in this document before starting to modify it. Spending a few minutes to understand the organization and purpose of each class and method will be worth the time spent.

You should follow the TDD approach described in the course as strictly as possible in order to ensure that you're doing it correctly and also to let you evaluate the effectiveness of using this approach as one of your personal process practices.

## Pig Game rules

You will implement software to manage a single game of *Pig* with three variations, standard, two dice, one die with different turn conditions. All of these games are played with two players. Player 1 always goes first.

### Standard Pig

The standard game of Pig has one die one or more times by a player in that player's turn. Both players alternate turns. At the beginning of the game each player has a current score of 0.

A player starts a turn with some current score of $x$. The following steps occur during the player's turn:

1. The turn score is set to 0.

[1]See Wikipedia article *Pig (dice game)*.

2. Player rolls the die and it shows a value $d$. If $d = 1$, the turn is over and the player's current score remains at $x$. The next player starts a new turn
3. If the $d \neq 1$, then the turn score is incremented by $d$.
4. If the turn score plus the current score $\geq$ the score needed to win, the player wins and the game is over.
5. If the player decides to continue rolling the die, go to step 2.
6. The player decides to *hold* and the turn score is added to the player's current score. The turn is over and the next player starts a new turn.

### Two dice

For this version of the game there is one change. A player rolls two dice for each roll (step 2 above). If the total of the dice is 7, then the turn is over and the current score does not change. All other steps from the standard game apply.

### One die, duplicate roll

This version is similar to the standard pig game, with one difference. The first roll of a turn counts; that is it is the turn score after that first roll, even if it is a 1. The player may hold at this point, just as in the standard game and the turn score is added to the current score (possibly winning the game as in step 4 of the standard game). On subsequent rolls, if the die roll has the same result as the immediately previous roll, the turn ends and the current score does not change. This is the same thing as if the player had rolled a 1 in the standard game.

## The starting code base

This section describes the starting code base. We use UML diagrams to describe the structure and elaborate with text as needed. For this assignment, there is just one package, pig, that contains all of the source files. The relationship is shown in Figure 1.

Table 1 describes the purpose of the classes. You should be able to look at the code and understand it with little difficulty. Pay attention to the comments on the methods since they describe the inputs and outputs.

Table 1: Class descriptions.

| **Class** | **Purpose** |
|---|---|
| PigGame | This is the main class for the game. It has a single constructor that takes in everything for a specific game. Once instantiated, it will leet the client (users) play a single game. |
| PigGameVersion | This is simply an enumeration with constants for the three versions we will be using. |
| die | This interface has a single method, `roll()` that must be implemented by every implementation of a die. While we're only using six-sided dice in this game, you will probably need different implementations for testing |
| PigGameException | A runtime exception that you throw when any error occurs. |

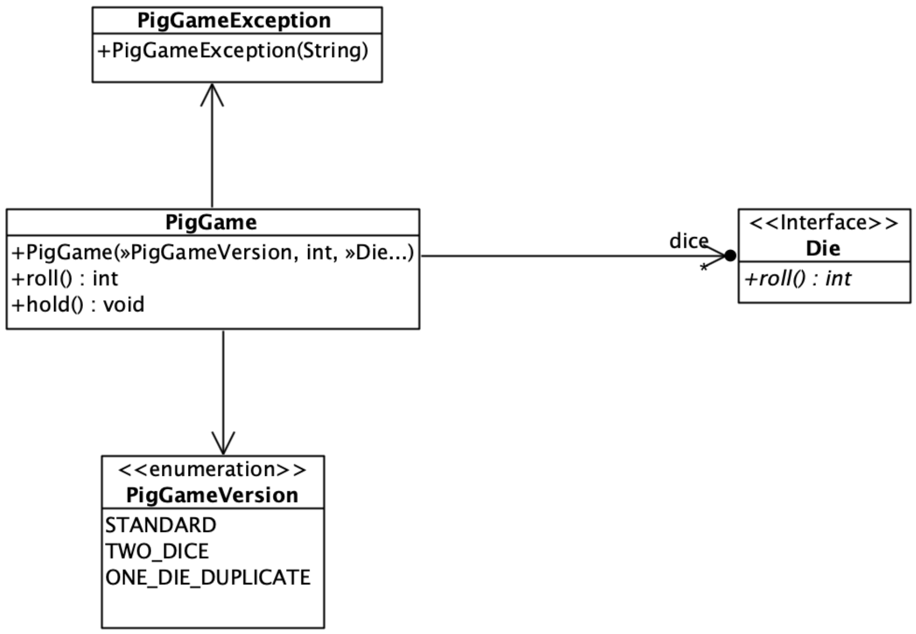

Figure 1: Starting code.

I have provided a starting test file in the `test` source folder with one test in it. This test expects a `PigGameException` when creating a game with no dice. You can use this a s a starting point for your Test-Driven-Development.

## What you will do

You will write code that implements the three versions of the *Pig* game. The version, score needed, and the dice are supplied as arguments to the constructor of the game. Think about your design and refactor as you go so that if you had to implement different versions of the game it would require a minimal amount of work. All of the tests will create an instance of your `PigGame` class and call the `roll()` and `hold()` methods. You can add as many other helper methods and classes as you need. You can create new sub-packages, but the files that are in the `pig` package may not be moved.

You will use Test-Driven Development (TDD) for this assignment. Keep a TODO task list as a text file in your project. Number the tasks in the order you implement them. Also, number the tests in your test code as you create them.

### Submitting your work

Submitting your work should be easy, but it's amazing how many people do not seem to know how to export a project in Eclipse and make sure that it can be imported by the graders. Here are the steps you should follow to ensure that there are no problems that will cost you points on the grade.

1. Export your project to a zipped archive. You can right-click on the project and select the **Export...** context menu item. Export to an archive file under the **General** menu. The project should have a check in the box next to its name and the project and classpath should be checked on the right side of the pane.

2. Save the project to a file called **PigGame-*username*.zip** where the username is yours.

3. Change the name of your project to some other name. Now try to import the project from your zipped archive that you saved in the previous step.
4. Run the tests to make sure everything is still working.
5. Submit the archive to the assignment on the Canvas page for the assignment.

My estimate of the time it will take you to complete this assignment is 2-4 hours. *Make sure that you allow enough time to get the assignment completed, cleaned up, checked, and submitted.*

## Grading

This assignment is worth 80 points. The grade is calculated as follows. Note that some of the categories might yield negative points.

Table 2: Grading rubric

| **Criterion** | **Max Points** |
|---|---|
| **Code correctness**. You begin with 50 points and lose three points for each test of mine that fails | 50 |
| **Intentional code**. When looking at your code, is it clear what the purpose of each class is? Are the methods named such that they are clear as to their purpose? Are there comments that follow the class guidelines. Is your code consistently formatted? | 10 |
| **Code coverage/waste**. Do you expose only what a client needs? Do you have code with no purpose and is not used? We will use the EclEmma code coverage too that is in Eclipse to see how much of your code is executed when running **your tests**. You begin with 15 points and lose one point for each percent or fraction of a percent less than 92%. We do not look at coverage of enumerations, interfaces, and your test code. | 15 |
| **TDD**. Does your TODO task list clearly indicate the use of TDD? The grader will assess 0 or 5 points on their assessment of your tasks, the order of the tasks, and your tests. | 5 |