Poker Dice

Your name:

By the time you are done with this activity, you should be able to:

- devise and implement linear-time algorithms.
- write JUnit tests.

After you complete this activity, please fill out the short survey at

http://goo.gl/forms/HXjyuUb2ou

to improve this project for future users.

Playing the game

Included in the Poker Dice project is a file Poker Dice.jar. This is a compiled version of the working game. To play it, use a terminal to navigate to the directory containing the file and type this on the command line:

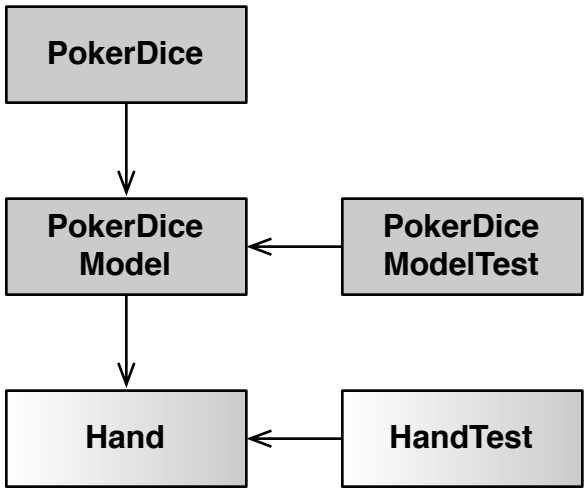
```
java -jar Poker Dice.jar
```

This game works for 2-5 players. You and other students should play a few times to get a sense of how the game works. If you are not familiar with the meanings of the various *Poker* hands, look them up on the internet.

Overview

The structure of the program is fairly straightforward: Hand represents a hand of dice, PokerDiceModel represents the logical model of the game, and PokerDice is the graphic user interface.

The UML class diagram below shows the relationships between classes. The shaded classes have been provided for you; you should not alter them. Hand and HandTest are half-shaded because you have been given incomplete skeletons of them.



Take notes as you work. What bugs and conceptual difficulties did you encounter? How did you overcome them? What did you learn?

There are two non-obvious parts of the implementation of the game.

The first is that, except in the GUI, the sides on the dice are numbered 0 through 5. This makes detecting things like straights easier. In the GUI, 0 is displayed as 9, 1 as 10, 2 as J, and so on.

The second is that the various scoring methods each return a large decimal number indicating the exact score for the hand. For example, given the hand Q J J Q J, the `fullHouseScore` method should return 523000. The first digit (5) indicates that this is indeed a full house, the second (2) indicates that the three identical dice are jacks, and the third (3) indicates that the remaining pair are queens. The information about the ranks is necessary because this hand should beat (i.e., receive a higher score than) 10 10 10 K K, which would receive a score of only 514000. Further details can be found in the Javadoc comments.

Five of a kind and four of a kind

Examine `fiveOfAKindScore` in Hand.java and `scoresFiveOfAKind` in HandTest.java. Notice that:

- `fiveOfAKindScore` runs in linear time.
- While not testing every possibility (which would be infeasible), `scoresFiveOfAKind` does include one case that *is* five of a kind and one case that *is not quite* five of a kind.

Using `scoresFiveOfAKind` as a template, complete the test `scoresFourOfAKind`. This test should pass since `fourOfAKindScore` is correct.

Remaining methods

For each of the methods listed below, complete the corresponding test, fail the test, and then write code that passes the test.

- `fullHouseScore`
- `straightScore`
- `threeOfAKindScore`
- `twoPairScore`
- `onePairScore`
- `highCardScore`

The last test is slightly easier because you only need one case; a hand can't fail to have a high card.

Each of your methods must run in time linear in the length of the array `dice`. This means that:

- You cannot sort the array, as the fastest sorting algorithm you know (mergesort) takes time in $\Theta(n \log n)$. (If you take an algorithms course, you will learn that sorting takes time in $\Omega(n \log n)$ *in general*; it is possible to do better in some special cases, including this one, but it's beyond the scope of this assignment.)

- It's okay to iterate through `dice` a fixed number of times, but not (for example) to use a nested loop that considers each pair of indices.

Be sure the play the game (by running PokerDice.java) a few times after you're done to make sure the entire system is working correctly.

Please fill out the survey at http://goo.gl/forms/HXjyuUb2ou.

This work was supported by the Google Education and Relations Fund's CS Engagement Small Grant Program grant #TFR15-00411.

The principal investigator, Peter Drake, wishes to thank the following for their useful comments: the members of the CS-POGIL project, specifically Clif Kussmaul and Helen Hu; Maggie Dreyer; and various anonymous reviewers.