PokerHand.docx

Introduction

This is the culmination of your exploration of card and deck objects. What's next? Poker! With a few additional classes you can play a miniature version of poker. Just deal five cards to one player, five cards to another and then "showdown." If you're not familiar with the ranking of poker hands you can find lots of useful information in this Wikipedia article.

Hint: Start by importing the PokerHand.zip project. The only class that needs modification is PokerRanker.java.

Simplifications

There are two aspects of poker that make this program extremely tricky, so you don't have to do them. The first is whether the ace is to be played "high" or "low." Let's assume that the ace can only played "high" (this means it's always higher than a king). The other tricky part is how to break ties. For example, if two players both have a pair then you compare the value of each pair, or if they both have the same pair then you compare the next high card, etc. But in our case when there's a tie just let the 'compareTo' method in the PokerHand class just return zero (a tie).

Start by Browsing

Use Eclipse to browse the pre-written code supplied by your instructor. Browse them in the following order.

PokerRank.java

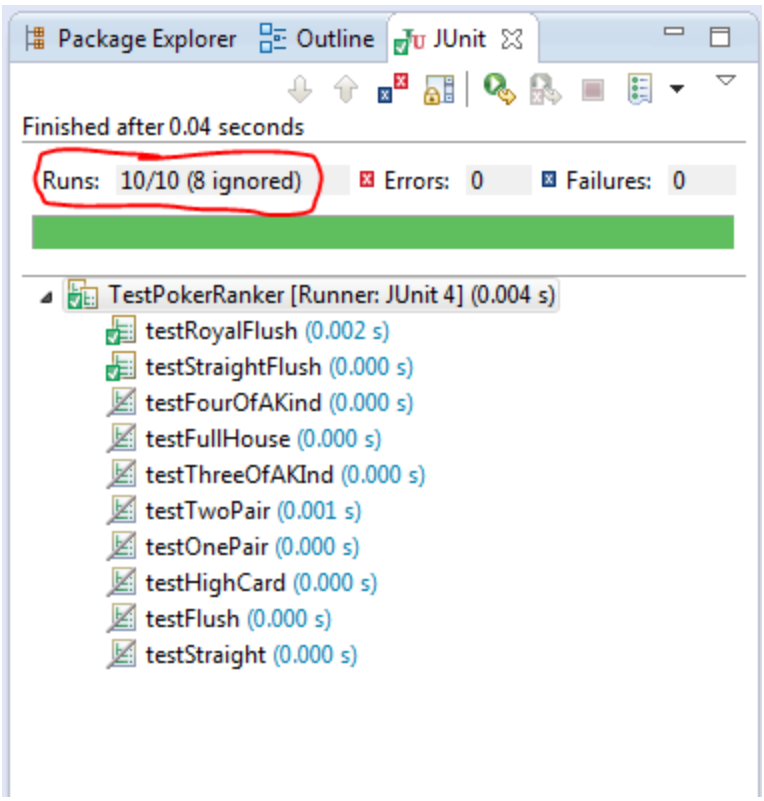
This is a simple enum class that lists the ranks of poker hands from low to high. The order is important! An enum type has a built-in compareTo() method that will be used to compare one rank to another. It is used in PokerHand.compareTo().

PokerHand.java

This class is a subclass of Hand and it implements the Comparable interface. Doesn't that make sense? It is more-specific than a Hand in that it only allows five cards in the hand. It has methods to get the rank of the hand (getRank()) and compare two poker hands (compareTo())

TestPokerRanker.java

This is a JUnit test class. You have run these before, just right click on the test file and select Run As => JUnit Test. Please do this and see the initial results:



Notice that 8 of the 10 test cases have been ignored. That's because you have to implement the logic before it can be tested. But the good news is that testRoyalFlush and testStraightFlush are working (because I wrote the logic in order to get you started).

As you implement the logic in PokerRanker.java you must activate each test case one by one. You do this by commenting out the @Ignore annotation as shown below:

BEFORE

```
        @Ignore
        @Test
        public void testStraight() {
```

AFTER

```
//      @Ignore
        @Test
        public void testStraight() {
```

How do the tests work? You don't want to randomly generate hands and then test one-by-one for thousands of hands as it may take many random deals to generate interesting results like four-of-a-kind or a flush. You take advantage of the fact that the deck is an array of cards, and you can construct specific hands by just pulling specific cards. Here's a table I made to guide me in this process:

Index	Card	Index	Card	Index	Card	Index	Card
0	2:SPADE	13	2:HEART	26	2:DIAMOND	39	2:CLUB

1	3:SPADE	14	3:HEART	27	3:DIAMOND	40	3:CLUB
2	4:SPADE	15	4:HEART	28	4:DIAMOND	41	4:CLUB
3	5:SPADE	16	5:HEART	29	5:DIAMOND	42	5:CLUB
4	6:SPADE	17	6:HEART	30	6:DIAMOND	43	6:CLUB
5	7:SPADE	18	7:HEART	31	7:DIAMOND	44	7:CLUB
6	8:SPADE	19	8:HEART	32	8:DIAMOND	45	8:CLUB
7	9:SPADE	20	9:HEART	33	9:DIAMOND	46	9:CLUB
8	10:SPADE	21	10:HEART	34	10:DIAMOND	47	10:CLUB
9	J:SPADE	22	J:HEART	35	J:DIAMOND	48	J:CLUB
10	Q:SPADE	23	Q:HEART	36	Q:DIAMOND	49	Q:CLUB
11	K:SPADE	24	K:HEART	37	K:DIAMOND	50	K:CLUB
12	A:SPADE	25	A:HEART	38	A:DIAMOND	51	A:CLUB

Good test cases are hard to construct. It is even possible to have a bug in them that leads to a false result.

MainClass

This class has been written to play poker. After all of the tests pass, you can right click on it and Run As => Java Application. It's fun to see the ranks of some of the hands. Look closely at the cards in each hand and verify that the rank of the hand is correct.

Get Down to Work

You must complete the code in PokerRanker.java by eliminating the TODO comments with the correct logic.

The main method is PokerRanker.rank() which is already complete. Do not change the code here. Look at the sequence of the if tests. Does this order of testing make sense? Why are nested if/else statements used?

Note: The word "rank" is used as both a noun and a verb here. Don't let that confuse you.

There are helper methods provided named isValueEqual() and isSuitEqual(). If you understand and use these methods it will save you a LOT of work.

The implementations of isStraight() and isFlush are complete. That's why two of unit tests pass.

Now implement isFourOfAKind() method. Once you think you have it, activate the test case testFourOfAKind(). If the test doesn't pass you have a bug that must be fixed. Otherwise, move onto isFullHouse().

When you have completed all of the TODOs and all the tests pass, you are done. Take a break and watch the computer play poker.

Assignment

Submit the PokerRanker.java file via Canvas. In a comment write your reflections on "what I learned" when doing this assignment.