

Programming Assignment 4

Due: 11:59pm, Saturday, January 30

Overview

The goals of this section are to:

1. Use methods
2. Break down a problem into small tasks to implement

Setup

This assignment requires one starter file. Set up your directory via:

```
$ mkdir HW4
$ cd HW4
$ cp /home/linux/ieng6/cs11wb/public/HW4/* .
```

Poker (40 pts)

By the end of this assignment you will have created a program that can compare two poker hands and declare a winner. If you've never played poker, review [wikipedia](https://en.wikipedia.org/wiki/List_of_poker_hands) for the list of winning hands (in order):

- straight flush
- four of a kind
- full house
- flush
- straight
- three of a kind
- two pair
- one pair
- high card

We will be storing both the card values and suits in arrays. Values will range from:

Number Representation	Card Value
1	A (Ace)
2	2
3	3
4	4
5	5
6	6
7	7
8	8

9	9
10	10
11	J (Jack)
12	Q (Queen)
13	K (King)

And suits will be stored with chars:

Char Representation	Suit Value
S	Spades
D	Diamonds
H	Hearts
C	Clubs

Step 1 (5 pts): enter a hand of cards

Allow the user to input the hand of a single player. You must store this into arrays. My solution used 2 arrays, one for the card value and one for the card suit. It should look **exactly** like this:

```
Enter Player 1's cards: 1 S 2 C 10 D 12 H 13 D
```

This hand represents:

Ace of spades, 2 of clubs, 10 of diamonds, Queen of hearts, and King of diamonds.

Note: it is safe to assume that we will only enter numbers between 1 – 13 and we will only enter characters S, D, H, and C. When we test your code, we will follow the rules of input, so it is not necessary to check for bad input.

Step 2 (5 pts): print player's cards

Print out the cards of the player. This should happen in its own method. Your code should now result in **exactly** the following:

```
Enter Player 1's cards: 1 S 2 C 10 D 12 H 13 D
Player 1's hand: AS 2C 10D QH KD
```

Step 3 (15 pts): Discover player's poker hand

Print out the hand of the player, e.g. FULL HOUSE, FOUR OF A KIND, etc. You should select the best hand. That is, if a player has 3 queens and 2 jacks, you would say they have a FULL HOUSE, **not** three of a kind. You should create methods to test for each possible hand. This will result in 8 methods (*high card* does not require a method). Don't forget the Ace can be high *or* low in a run, e.g. A, 2, 3, 4, 5 is a straight, and 10, J, Q, K, A also counts as a straight.

Critical at this point: cards will be entered in numerical order, e.g. 1, 2, 3, ... 11, 12, 13. This is **HUGE** and will be a key assumption that you can follow in your program, and will make this program much easier. This is my gift to you.

Here's some example runs. Make sure your code prints out these exact hands, this is how we'll be testing your code.

```
Enter Player 1's cards: 1 S 2 C 10 D 12 H 13 D
Player 1's hand: AS 2C 10D QH KD
Best hand: HIGH CARD
```

```
Enter Player 1's cards: 1 S 2 S 3 S 4 S 5 S
Player 1's hand: AS 2S 3S 4S 5S
Best hand: STRAIGHT FLUSH
```

```
Enter Player 1's cards: 1 D 10 C 11 S 12 H 13 H
Player 1's hand: AD 10C JS QH KH
Best hand: STRAIGHT
```

```
Enter Player 1's cards: 10 D 10 H 10 C 13 H 13 D
Player 1's hand: 10D 10H 10C KH KD
Best hand: FULL HOUSE
```

```
Enter Player 1's cards: 2 C 2 H 5 D 11 D 11 C
Player 1's hand: 2C 2H 5D JD JC
Best hand: TWO PAIR
```

Step 4 (10 pts): Allow a 2nd player to enter their cards, print cards and hand

This will happen in the same fashion as player 1. You should not need to duplicate code to do this, make sure you're using methods. If we see duplicated code, we will deduct half points for this section. Your code should now look like this:

```
Enter Player 1's cards: 1 S 2 S 3 S 4 S 5 S
Player 1's hand: AS 2S 3S 4S 5S
Best hand: STRAIGHT FLUSH
Enter Player 2's cards: 2 C 2 H 5 D 11 D 11 C
Player 2's hand: 2C 2H 5D JD JC
Best hand: TWO PAIR
```

Step 5 (5 pts): Print the winner or tie

Print the player that has the winning hand. If both players have the same hand, e.g. both players have a full house, print there's a tie. This isn't what happens in poker, but I'm making the decision to make this part of the assignment easier, so just print "Tie". Here's some example runs.

```
Enter Player 1's cards: 1 S 2 S 3 S 4 S 5 S
Player 1's hand: AS 2S 3S 4S 5S
Best hand: STRAIGHT FLUSH
Enter Player 2's cards: 2 C 2 H 5 D 11 D 11 C
Player 2's hand: 2C 2H 5D JD JC
Best hand: TWO PAIR
Player 1 wins!
```

```
Enter Player 1's cards: 1 S 2 S 3 S 4 S 5 S
Player 1's hand: AS 2S 3S 4S 5S
Best hand: STRAIGHT FLUSH
Enter Player 2's cards: 1 D 10 D 11 D 12 D 13 D
Player 2's hand: AD 10D JD QD KD
Best hand: STRAIGHT FLUSH
Tie!
```

Programmer of the week (optional)

If you'd like to compete in the programmer of the week challenge, you must complete the above part of the assignment as described. Once you get that to work, create another class, POWeek.java, and start to extend the poker game. If you know how to use javafx or java.swing, you could include graphics. You could implement betting, support multiple players, allow for cards being dealt instead of being input by the user, handle ties, change to Texas hold 'em, etc.

Your tutor will present your submission at our weekly meeting and we'll vote for the top 5 submissions from each class and show those to the students to vote for the programmer of the week. The winner will get an award, 1% extra credit added to their final grade, and be famous for a week. Note: we will only be looking at the running of the code, i.e. we will not count off for anything done in the POWeek.java submission.

Style Requirements (10 pts) – Different from other assignments

You will be graded for the style of programming on this assignment. **We are adding javadoc comments as a requirement to the assignment.**

- Use reasonable comments to make your code clear and readable.
- **(New)** All methods must have javadoc comments. We will be testing this by running "javadoc filename.java" and ensuring that the resulting documentation pages appear.
- Use reasonable variable names that are meaningful.
- Use static final constants to make your code as general as possible. No hardcoding constant values inline (no magic numbers).
 - **Note (new):** There will be many cases in this homework where you want to compare card 1 to card 2, card 2 to card 3, etc. I'm allowing you to hardcode numbers for this.
- Judicious use of blank spaces around logical chunks of code makes your code much easier to read and debug.
- Keep all lines less than 80 characters. Make sure each level of indentation lines up evenly.
- Every time you open a new block of code (use a '{'), indent farther by 2 spaces. Go back to the previous level of indenting when you close the block (use a '}').
- Always recompile and run your program right before turning it in, just in case you commented out some code by mistake.

Turnin Instructions

Remember the deadline to turn in your assignment is Saturday, January 30 by 11:59pm.

When you are ready to turn in your program in, type in the following command and answer the prompted questions:

```
$ cd ~/
$ bundleP4
Good; all required files are present:

    HW4

Do you want to go ahead and turnin these files? [y/n]y
OK.  Proceeding.

Performing turnin of approx. 6144 bytes (+/- 10%)
Copying to /home/linux/ieng6/cs11wb/turnin.dest/cs11wb.P4
...
Done.
Total bytes written: 31744
Please check to be sure that's reasonable.
Turnin successful.
```