The University of Melbourne
Department of Computing and Information Systems
COMP90041
Programming and Software Development
Second Semester, 2015

Project
Due Monday, 12 October 2015, 5:00PM

Objective

The objective of this project is to practice your Java programming skills. This project is a departure from the earlier assessed labs, in that the assessment will focus on *code quality*, as well as correctness. You should allocate a few hours to tidying up your code after your implementation is complete. You will make comments on three of your peers' projects' coding and documentation style, and will receive feedback from your peers on your submission.

The Game of Poker

Poker is a gambling card game dating back to the early 19$^{\text{th}}$ century. There are a great many variations of the game, but for this project we will just consider the simple 5-card game, leaving out much of the subtlety of the game involving betting and bluffing.

A poker hand consists of five cards from a standard 52-card deck (without jokers). The ranks, in order of increasing value, are 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, and Ace. There are 9 classifications of hands, as follows, in order of decreasing value:

Straight flush consists of 5 cards of the same suit whose ranks form a sequence, such as 7♡, 8♡, 9♡, 10♡, J♡. If two players both have a straight flush, the one with the highest card wins. If both players have the same ranks of cards (but different suits), the players draw. Suits are never used to decide the winner in poker.

Four of a kind consists of a set of four cards of the same rank, and any other card, such as 3♡, 3♠, 3♢, 3♣, 7♡. If two players both have four of a kind, the one with the highest rank of their set of four wins. In some varieties of poker it is possible for two players to both have 4 of the same rank; in this case the one with the higher rank of their fifth card wins. If the fifth cards are also the same rank, it is a draw.

Full house consists of three cards of one rank and two of another; that is three of a kind plus a pair, for example Q♡, Q♣, Q♢, 7♠, 7♣. If two player both have a full house, the one with the higher rank of their set of three wins. If two players to each have three cards of the same rank, then the rank of the pair is used to decide.

Flush consists of any 5 cards of the same suit, regardless of their rank, for example K♣, 9♣, 3♣, J♣, 4♣. If two players both have a flush, then their highest ranking card is used to decide the winner. If their highest ranking cards have the same rank, then their second highest ranking cards, followed by their, third, fourth, and fifth highest ranking cards are used in turn.

Straight consists of 5 cards whose ranks form a sequence, regardless of their suits, for example, 6♠, 7♣, 8♢, 9♠, 10♡. If two players both have a straight, the one with the highest ranking card wins. If two hands have the same highest ranking card, they draw.

Three of a kind consists of 3 cards of the same rank, and any other 2 cards, for example, 6♢, 6♠, 6♡, 9♡, J♢. If two players both have three of a kind, the player whose three of a kind have the highest rank wins. If this is a tie, then the higher ranking other card is used, and if this, too, is a tie, then the rank of last card decides.

Two pair consists of 2 cards of one rank, 2 cards of another rank, and any other card, such as 5♢, 5♠, 9♣, 9♠, A♡. If two players both have two pair, the ranks of their highest ranking pairs decide the winner; if their highest ranking pairs are the same rank, the second ranking pair decides; if they are also the same, the rank of the final cards decides.

One pair consists of 2 cards of one rank, and any other 3 cards, for example, A♠, A♣, 6♠, J♡, 2♢. If two players both have one pair, the rank of the pair decides, with the ranks of their other cards, from high to low, used if their pairs are of the same rank.

High card consists of any 5 cards. If two players both have a high card hand, the ranks of the cards, from high to low, are used to decide the winner.

Note that the best (earliest in the list) description of a hand should be used. For example, a hand with three of a kind and a pair must always be classified as a full house, even though it could also be called three of a kind or a pair (or a high card hand).

There are innumerable sources of information about poker available online. One that may be helpful for this project is `http://en.wikipedia.org/wiki/List_of_poker_hands`, but the information in this description should be enough the complete the project.

The Program

There are two difficulty levels for this project. All students must complete the simpler version. Students wishing for a greater challenge may also undertake the second part of the assignment. Each student will be assessed on the part or parts they have undertaken.

For this project, you will write a program that will correctly characterise a poker hand according to the list of hands above. For students undertaking the more advanced project, if more than one hand is given as input, the program will also decide which hand wins.

In either case, your main program should be called `Poker.java`, and should expect some multiple of 5 cards on the command line (at least 5 cards). Cards should be entered on the command line as two-character strings, the first being an `A` for Ace, `K` for King, `Q` for Queen, `J` for Jack, `T` for Ten, or digit between `2` and `9` for ranks 2–9. The second character should be a `C` for Clubs (♣), `D` for Diamonds (♢), `H` for Hearts (♡), or `S` for Spades (♠). Your program should handle both upper and lower case characters (or even a mixture).

We make one small change to the rules of poker to make the project easier: we always take Aces to be the highest rank. Most versions of the rules of poker would consider Ace, 2, 3, 4, 5 to be a straight (or straight flush), but we do not.

The output generated by your program should include, for each hand specified on the command line, one line of the form:

`Player␣`*n*`:␣`*description of hand*

where n is `1` when describing the first 5 cards, `2` for the next 5, and so on, and space is shown as ␣. The *description of hand* gives both the hand category and some indication of how to decide ties using standard Poker lingo as follows:

Category	**Description**	**Detail**
Straight flush	r`-high␣straight␣flush`	r is highest rank in hand
Four of a kind	`Four␣`r`s`	r is rank of 4 cards
Full house	r_1`s␣full␣of␣`r_2`s`	r_1 is rank of 3 cards; r_2 is rank of 2
Flush	r`-high␣flush`	r is highest rank in hand
Straight	r`-high␣straight`	r is highest rank in hand
Three of a kind	`Three␣`r`s`	r is rank of 3 cards
Two pair	r_1`s␣over␣`r_2`s`	r_1 is rank of higher pair; r_2 is rank of lower pair
One pair	`Pair␣of␣`r`s`	r is rank of pair
High card	r`-high`	r is highest rank in hand

In all of the above, ranks (r, r_1 and r_2 above) should be shown as numerals, or full capitalised rank names for face cards: `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `Jack`, `Queen`, `King`, or `Ace`. Note that this is different from the way ranks are specified on the command line. Also note that the information conveyed in these hand descriptions is not always enough to decide the winner; for example, in deciding the winner between two Queen-high hands, the second-highest ranked cards in the two hands must be consulted, but these are not shown in the hand descriptions. Nevertheless, you must correctly decide the winner between two such hands, if you undertake the more challenging project.

For students choosing to complete the simpler version of the project, if more than 5 command line arguments are supplied, the program should output only `NOT␣UNDERTAKEN` as a single line, and then exit. They will not be assessed on the tests for more than one hand; however, it is important that they output exactly the line `NOT␣UNDERTAKEN`, so that the testing script does not take the output to be an erroneous attempt to handle the harder version of the project.

For students choosing to complete the more challenging assignment, if more than one hand is specified on the command line, the lines indicating each hand's description must be followed by a single line sentence indicating the winner. When player number n wins, this should have the form "`Player␣`n`␣wins.`" When two players draw, the sentence should have the form "`Players␣`n_1`␣and␣`n_2`␣draw.`" where n_1 and n_2 are the player numbers of the tied winning players, in order of ascending player number. If three or more players draw, the sentence should have the form "`Players␣`n_1`,␣`n_2 ...`␣and␣`n_k`␣draw.`" where n_1–n_k are the winning player numbers in ascending order. In all of this, `Player␣1` corresponds to the first 5 cards, `Player␣2` to the next 5, and so on.

If the number of command line arguments is not greater than zero or not a multiple of five, your program should print out

```
Error:␣wrong␣number␣of␣arguments;␣must␣be␣a␣multiple␣of␣5
```

and exit. If any of the command line arguments is not a valid card name, as specified above, the program should print out

```
Error:␣invalid␣card␣name␣'c'
```

and exit, where *c* is the (first) invalid card entered on the command line.

Note well: Since some forms of poker allow the same card to be considered to be part of many players' hands, you should not assume that no cards appear in more than one hand. If it is useful to you, you may assume that no cards are repeated in a single hand, but you do not need to check for this.

Examples

For example, this input:

```
java Poker 2H TH AS AD TC
```

should produce this output:

```
Player␣1:␣Aces␣over␣10s
```

This input:

```
java Poker 2H TH 1S 1D TC
```

Should produce this output:

```
Error:␣invalid␣card␣name␣'1S'
```

This input:

```
java Poker KS 9S QS AS JS  3D 7C 3S 3H 7S
```

Should produce this output:

```
Player␣1:␣Ace-high␣flush
Player␣2:␣3s␣full␣of␣7s
Player␣2␣wins.
```

This input:

```
java Poker qc jc 2h 7s 9h  qd jd 2s 7c 9s  9c 7d 2c jh qh  9d 7h 2d js qs
```

Should produce this output:

```
Player␣1:␣Queen-high
Player␣2:␣Queen-high
Player␣3:␣Queen-high
Player␣4:␣Queen-high
Players␣1,␣2,␣3␣and␣4␣draw.
```

Submission

You will submit your work twice almost identically both times. The first submission will be for assessing correctness and code quality, and should include your name and login ID (but not your student ID) on all files. The second submission will be for anonymous peer assessment, and so should not include your name or login ID in any of the files.

The first submission will be done similarly to the assessed labs. You must submit your project from any one of the student unix servers. Make sure the version of your program source files you wish to submit is on these machines (your files are shared between all of them, so any one will do), then `cd` to the directory holding your source code and issue the command:

```
submit COMP90041 proj  all your source files
```

where *all your source files* is the names of all your `.java` files, including `Poker.java`, in any order, separated by spaces.
Important: you must wait a minute or two (or more if the servers are busy) after submitting, and then issue the command

```
verify COMP90041 proj | less
```

This will show you the test results and the marks from your submission, as well as the file(s) you submitted. If the test results show any problems, correct them and submit again. You may submit as often as you like; only your final submission will be assessed.

If you wish to (re-)submit after the project deadline, you may do so by adding "`.late`" to the end of the project name (*i.e.,* `proj.late`) in the `submit` and `verify` commands. But note that a penalty, described below, will apply to late submissions, so you should weigh the points you will lose for a late submission against the points you expect to gain by revising your program and submitting again. **It is your responsibility to verify your submission.**

Instructions for the second, anonymised, submission will be posted closer to the due date.

Assessment

Your project will be assessed on the following criteria:

35% The quality of your code and documentation;

35% The correctness of your program based on the output produced;

30% The quality of the anonymous feedback you give the peer projects you are assigned.

Note that timeouts will be imposed on all tests. You will have at least 1 second per test case, which should be ample. At least half of the correctness tests will involve only one hand. This means that a well-written program that can only evaluate hands, but not compare them, can still pass.

Hints

1. Start by implementing a class to represent cards. I suggest using an `enum` to represent the ranks and suits.

2. For the simpler project, you will need to classify a hand. For the more challenging project, you also need to compare two hands to see which wins. I suggest writing a class to represent a poker hand description, including the classification and extra information that needs to be displayed (such as the rank of a pair or the ranks of the triple and pair in a full house). For the more challenging project, this will need to contain more information, such as the ranks of all the cards in a high-card hand, to decide which of two hands of the same category wins. Write a static method for this class that takes an array of five instances of your card class and constructs and returns the appropriate description object.

   Again, an `enum` is good way to handle the classifications. As for the extra information, it always comes down to comparing ranks (suits never matter), so I suggest using an array of ranks to provide the extra information. If two hands have different classifications, then the extra information is not needed to determine the winner. When the classifications are the same, the extra information needed depends on the actual classification. For example, for a straight or straight flush, only the highest ranking card is needed, so only that rank needs to be in the array. For a full house, the ranks of the triple and pair are considered, in that order, so those ranks should be in the array. And so on for the other classifications. Implemented this way, you can compare two hand descriptions to choose the winner by first comparing the classifications; if one is higher, that one wins. If they are the same, then compare the first rank in the extra information arrays. If one is greater, that hand wins; if not, consider the two second ranks in the array, and so on. If you get to the end of the array without finding a difference, the two hands draw.

3. Despite there being nine possible classifications ranging from high card up to straight flush, there are really only three things you need to look for in a hand: straights, flushes, and n-of-a-kind. I suggest writing a method for each of these.

4. Only for flushes do the suits of the cards matter; when looking for runs and n-of-a-kinds it is only necessary to consider the ranks of the cards. Also notice that both of these are much easier to check for if the ranks are sorted first. To sort an array, you can use the standard library method `java.util.Arrays.sort` that takes an array of objects as its only argument, and leaves the array sorted. Documentation is available at `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Arrays.html`. To use this method, your class needs to implement a `compareTo` method, as we will discuss in lecture. Note, however, that java `enum`s automatically implement a `compareTo` method that will do what you want (as long as you enumerate the ranks in order), so if you define your rank type as an enum, you will be able to sort an array of ranks.

   Once the ranks are sorted, you can check for a run by just checking that each card but the first is the next greater rank than the previous rank. To look for n-of-a-kind, just look for runs of adjacent identical ranks.

   To handle full houses and two pairs, you will need to find two n-of-a-kinds in a single hand. This means your n-of-a-kind code needs to find the start and end position of the

n-of-a-kind so you can look for another one after the first one. Note that there cannot be more than two n-of-a-kinds in a single hand (since n must be at least 2).

Late Penalties

Late submissions will incur a penalty of 0.5% of the possible value of that submission per hour late, including evening and weekend hours. This means that a perfect project that is a bit more than 4 days late will lose half the marks. If you have a medical or similar compelling reason for being late, you should contact the head tutor, Matt DeBono (`mdebono AT student.unimelb.edu.au`) as early as possible to ask for an extension.

Academic Honesty

This project is part of your final assessment, so cheating is not acceptable. Any form of material exchange between students, whether written, electronic or any other medium, is considered cheating, and so is the soliciting of help from electronic newsgroups. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties.