The University of Melbourne
School of Computing and Information Systems
COMP90041
Programming and Software Development
Second Semester, 2018

Project
Due Saturday, 13 October 2018, 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 Cribbage

Cribbage is a very old card game, dating to early 17^{th} century England. The game can be played by two to six (and possibly more) players. The object of the game is to be the first player to reach 121 points. Game play begins with the dealer dealing each player 4 to 6 cards (depending on how many are playing). Each player then selects four cards to keep and discards the rest. The cards discarded for all the players form an extra hand, called the *crib* or *box*, which the dealer gets to count as a bonus. Next the player preceding the dealer cuts the deck to select an extra card, called the *start* card.

The hand then proceeds to the *play*, which is outside the scope of this project, followed by the *show*, where each player counts the points in their hand according to the rules below, and adds the total to their running score. For this phase, the start card is usually considered as part of *each* player's hand, so each player establishes the value of a 5 card hand. Points are scored for certain combinations of cards according to the following rules:

- (**15s**) 2 points are scored for each distinct combinations of cards that add to 15. For this purpose, an ace is counted as 1, a jack, queen or king is counted as 10, and other cards are counted as their face value. For example, a hand with a 2, 3, 5, 10, and king scores 8 points: 2 points each for 2+3+10, 2+3+king, 5+10, and 5+king.

- (**Pairs**) 2 points are scored for each pair. With 3 of a kind, there are 3 different ways to make a pair, so 3 of a kind scores 6 points. Similarly, 4 of a kind scores 12 points for the 6 possible pairs.

- (**Runs**) 1 point is scored for each card in a *run* of 3 or more consecutive cards (the suits of these cards need not be the same). Aces are low in cribbage, so Ace, 2, 3 is a run, but Queen, King, Ace is not. For example, a 2, 6, 8, and 9, with a 7 as the start card scores 4 points for a 4 card run. It also scores a further 6 points for 15s (6+9, 7+8, and 2+6+7).

- (**Flushes**) 4 points is scored if all the cards in the hand are of the same suit. 1 further point is scored if the start card is also the same suit. Note that no points are scored if 3 of the hand cards, plus the start card, are the same suit.
- (**"One for his nob"**) 1 point is scored if the hand contains the jack of the same suit as the start card.

All of these points are totalled to find the value of a hand. For example (using **A** for ace, **T** for 10, **J** for jack, **Q** for queen, and **K** for king, and ♣, ♢, ♡, and ♠ as clubs, diamonds, hearts, and spades):

Hand	Start card	Value	Explanation
7♣, Q♡, 2♣, J♣	9♡	0	
A♠, 3♡, K♡, 7♡	K♠	2	1 pair
A♠, 3♡, K♡, 7♡	2♢	5	1 run, 1 15
2♠, 3♡, K♡, 3♠	4♡	12	2 runs, 1 pair, 2 15s
6♣, 7♣, 8♣, 9♣	8♠	20	2 runs, 1 pair, 3 15s, flush
7♡, 9♠, 8♣, 7♣	8♡	24	4 runs, 2 pair, 4 15s
5♡, 5♠, 5♣, J♢	5♢	29	8 15s, 6 pairs, 1 for his nob

Following the show, the cards are all collected and shuffled, and the person to the left of the dealer becomes the dealer for the next hand. The game continues this way until someone scores 121 points. Please see `http://www.pagat.com/adders/crib6.html` for a more complete description of the rules of cribbage. However, what is presented above will be enough to complete the project.

The Program

For this project, you may select *either* of the below programs to develop. The first is the easier project; students looking for a bigger challenge may want to select the second. Remember: **you only need to do one of these.**

Evaluate a hand. You will write a Java main program that receives 5 cards on the command line and will print out only the number of points the hand comprising the first four of those cards would score if the fifth card were the start card. The table above shows some examples of inputs and the score your program should print.

For this project, you should submit a source file called `HandValue.java`, plus any other Java source files needed for the application. The correctness of your program will be assessed based on the correctness of the output produced for a number of input hands.

Select a hand. You will write a Java main program that receives 4–6 cards on the command line and will print out which four should be kept to maximise your chances of having a good hand once the start card is selected. Of course, if 4 cards are specified on the command line, there is only one choice of hand; if 5 are specified, there are 5 possible choices; if 6 are specified, there are 15 possibilities.

To make the best choice, you should consider each of the possible selections of four cards to keep (the others being discarded), and select the one with the greatest *expected*

score. The expected score for a selection is the average hand value of the four cards to be kept taken with each of the possible start cards (every card in a full deck except the 4–6 specified on the command line, which cannot be the start card).

For this project you should submit a source file called `SelectHand.java`, plus any other Java source files needed for the application. The correctness of your program will be assessed based on the optimality of the output produced for a number of input hands. That is, if the hand your program selects is almost as good as the optimal choice, you will receive most of the marks for that test.

For both programs, cards should be entered on the command line as two-character strings, the first being an upper-case `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 a `S` for Spades (♠). For the `SelectHand` program, output should follow the same format, with selected cards separated by a single space. For both programs, the output should not contain any extra characters, except the single line should end with a newline character.

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 on all files. The second submission will be for anonymous peer assessment, and so should not include your name and 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 the files of your project...⟩
```

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 a little closer to the due date.

Assessment

Your project will be assessed on the following criteria:

40% The correctness of your `HandValue` or `SelectHand` implementation, as described above

35% The quality of your code and documentation

25% 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 for `HandValue` and 5 seconds per test for `SelectHand`. Executions taking longer than that may be unceremoniously terminated, causing failure of that test.

Hints

1. Start by implementing a class to represent cards. I am supplying a `CribbageRank` class. It's actually defined as an *enum*, and uses a feature of enums that was not covered in lecture or in the textbook: enums can define methods. You can read about this at `http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html`. You are welcome to use the `CribbageRank` class as is, or to adapt it to your needs, or to ignore it altogether. If you adapt it, you should leave my name on it as an author, and add your name as another author. If you do use it, you must submit it with your other files.

2. Whichever project you choose, start by writing code to compute the value of a hand. This will be needed whichever project you undertake, and if you implement this and have difficulty with the `SelectHand` project, it will be easy to complete the `HandValue` project.

3. Start out computing "one for his nob"; that's the easiest one to handle. Then handle flushes; they're not too much harder. Other than those two, nothing else needs to consider the suits of the cards, only the ranks, and nothing else needs to distinguish the start card from the other cards in the hand.

4. The other three ways of scoring points all need to consider combinations of the ranks of the cards in the hand including the start card. I am supplying a class `Combinations` with a single static method `combinations` that you are welcome to use (and if you do, you must submit it with your other files). This method takes an array of objects as input and returns an array of arrays of objects containing all the combinations of the input objects in the order they appear in the input array. For example, if the input is an array of the strings `"a"` and `"b"`, output will be an array whose first element is the empty array, second will be the array containing only `"a"`, third will contain only `"b"`, and fourth will contain both `"a"` and `"b"`. The supplied class has a `main` method that computes all combinations of the command line arguments and displays them one per line; you can experiment with this to understand this class, and examine the code of the `main` method to see how to use the `combinations` method.

5. To compute pairs, count all the combinations of two identical ranks, and multiply by two.

6. To compute 15s, add up the face values of each combination, count the ones that sum to 15, and multiply by two.

7. Computing runs is a little harder. Compute the length of each combination consisting only of cards that form a run. Then find the maximum length; if it is 3 or more, score one point for each card in the combination. However, if there is a multi-way tie for longest run (up to a 4-way tie is possible), you should multiply the length of the run by the number of combinations that give this length.

   It is easier to check if a combination forms a run if the array is sorted in rank order (Ace up to King). Once that is done you need only check that each element but the first is the successor of the previous one. 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 enums count as objects), and leaves the array sorted. Documentation is available at `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Arrays.html`.

Late Penalties

Late submissions will incur a penalty of 1% of the possible value of that submission per hour late, including evening and weekend hours. This means that a perfect project that is a little more than 2 days late will lose half the marks. If you have a medical or similar compelling reason for being late, you should contact me (`tchristy AT 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.