PROJECT 2

Assigned: March 29th, 2021

Due: Sunday, April 26th, 2021 before midnight

Intermediate Deadline: Sunday, April 14th

Updates: April 7, 2021

Objectives

- To implement a stack-like ADT from scratch
- To create and extend ADTs to serve a particular purpose
- To explore control flow choices in an interactive application
- To use and potentially extend a GUI
- To recognize design patterns in the implementation
- To add two more design patterns to your implementation

Overview

In this project, you will complete an interactive version of the Rummy card game. We will give you basic UI code and a Card class, as well as specify an interface that you must implement for the Hand, Deck, and Set classes. Other class design choices are yours. There are many variations of Rummy. You are required to follow the rules listed below.

Rules of Play

These rules are a slight adaptation of standard Rummy rules. The objective of this game is to make melds (sets and runs) of similar cards and lay them on the table until all cards in your hand have been disposed of. A set is three or more cards of the same rank, such as three aces or four sevens. A run is three or more sequential cards of the same suit, such as the three-four-five of clubs or the jack-queen-king of diamonds. The ace is always low.

Each player is dealt nine cards from the stock pile (Deck). The next card is turned face-up to initiate the discard pile (Stack). The deck is placed face-down on the table. In each turn, a player draws one card from either the Deck or Stack. They may optionally lay sets, runs on the table or place the card on Stack. They may also optionally lay cards that fit in sets or runs already on the table. The turn ends with a discard. If a player is able to lay all remaining cards on the table at the end of a turn, the discard is optional.

The game is over when one player is out of cards or the stock is exhausted. At this point, players count the points remaining in their hands. Aces are worth one, face cards are worth ten, and all other cards are worth their face value. Lowest value hand wins. Ties are possible.

Project Specification

In this project you will be implementing a stack class, named MyStack, from scratch. It should provide the standard stack operations of push, pop, top, and isEmpty. You should use generics for this class.

You are also expected to write the Hand, Deck, and Pile classes that implement the HandInterface, DeckInterface, and PileInterface interfaces. You may not change the methods given in these interfaces, but may add methods to these classes. Hand and Deck may be derived from other Java classes, such as those in the Collection framework. Pile should be derived from your MyStack class.

Your Deck class should provide a constructor which creates a 52-card deck of shuffled cards.

Your Hand class should maintain the cards in sorted order. Your Hand class should provide a play() function to implement a computer opponent. This automated player can be really stupid, but must follow the rules. Specifically, for each choice the automated player must make (for example, whether to draw from stock or discards), a random choice is fine. A more intelligent play routine would be extra credit. The base project need look only for sets in the hand. Runs are left as extra credit. We will be providing you with the code for the UI, interactive game control, and the Card class. The program takes two kinds of command line arguments (processed in the provided code). The -h switch enables logging of actions (by default there is no logging). ~~The player number switch (-0, -1, or -2) gives the number of interactive players in the game (by default this is 0, indicating two automated players).~~

The files for this project can be found on Piazza.

Sample Results

```
Initial Player 1: 7C, 4D, 9D, QD, 2H, 4H, 9H, JH, AS,
Initial Player 2: AD, 7D, JD, 5H, 8H, 4S, 7S, TS, JS,
Player 1
    Added: 9C
    Discarded: 7C
    Hand now: 9C, 4D, 9D, QD, 2H, 4H, 9H, JH, AS,
Player 2
    Added: QC
    Discarded: QC
    Hand now: AD, 7D, JD, 5H, 8H, 4S, 7S, TS, JS,
Player 1
    Added: QC
    Discarded: 9C
    Hand now: QC, 4D, 9D, QD, 2H, 4H, 9H, JH, AS,
Player 2
    Added: 9C
    Discarded: 9C
```

```
    Hand now: AD, 7D, JD, 5H, 8H, 4S, 7S, TS, JS,
Player 1
    Added: 9C
    Discarded: 9C
    Hand now: QC, 4D, 9D, QD, 2H, 4H, 9H, JH, AS,
Player 2
    Added: 7H
    Discarded: AD
    Hand now: 7D, JD, 5H, 7H, 8H, 4S, 7S, TS, JS,
Player 1
    Added: AD
    Discarded: QC
    Hand now: AD, 4D, 9D, QD, 2H, 4H, 9H, JH, AS,

(and so on)

Player 2
    Added: TH
    Discarded: JS QS KS
    Hand now: 8H, TH, JH, QH, 5S, 8S,
Player 1
    Added: 7C
    Discarded: 7C
    Hand now: KH, AS, 2S, 3S, 4S, 6S, 7S, 9S, TS,
Points: 52 to 51
Player 2 Wins!
```

Project Notes, Hints, and Requirements

1. This project is in Java 13 SE. It produces an error that it uses some deprecated APIs that you can ignore. I coded it by hand with TextPad and ran it on the commandline, but you are welcome to use an IDE such as Eclipse which can be found at www.eclipse.org or Netbeans which can be found at http://netbeans.apache.org/.
2. Currently, Table.java runs the program. You are to separate the main method into a file Proj2.java and wrap the entire project in a package named proj2.
3. By the intermediate deadline, hand in a description of the classes you intend to create, along with a specification of their data items and methods. You may change these choices later, but you must submit the initial design by the intermediate deadline.
4. You must implement your Stack class from scratch, with nothing but Object or primitive types as a superclass of Stack or any data elements of Stack.
5. The Deck and Hand classes may be subclasses of standard Java classes, such as those in the Collections framework. The Set class should be a subclass of your MyStack class, which is implemented from scratch.
6. The last line is either: **Player 1 Wins!**; **Player 2 Wins!**; or **It's a tie!**.
7. You may not change the HandInterface, SetInterface, or DeckInterface.
8. To create the file containing your output, use unix redirection on the command line to redirect standard output to a file. Thus, once you have the output on the screen as shown in the sample results above, then you can redirect your output to file by using the command: **`java proj2.Proj2 > p2-output.txt`**.
9. This project is an OPEN assignment (as was Project 1). You are still expected to write your own code, but you may continue to help each other debug. Specifically, you:

- should try as much as possible to complete the project by yourself.
- may get assistance from anyone -- TAs, instructors, fellow students, relatives, friends, etc.
- must document all outside help you get as part of your file header comment.

You MAY NOT:

- copy anyone else's code
- have someone else write your code for you
- submit someone else's code as your own

Any help you receive must be documented, including discussion, books, papers, and web resources. With each assignment, you will turn in a README text file indicating the sources you used while working on it and the type of help you received from each. If you received no help, say so. If you helped someone else, say so. Failure to include this README file will result in your program being returned ungraded.

10. Items 1-9 are worth 10 points for a total of 90 points. The final 10 points are for collaboration and adhering to coding standards are based on the comments of your team members and a rubric from the department respectively.

Extra Credit

There are MANY interesting ways to extend your base project. Be sure you finish all requirements of the base project before you begin working on extra credit. Indicate any items of extra credit that you did in your README file. This will allow me to look for that additional functionality when grading your project.

1. The card display in the UI is very minimalist. There are card images available in the folder named Iscards. Use these images to spruce up the appearance of your game. Worth five points.
2. Modify the Table so that the game begins with the first card being on the Stack. Worth five points.
3. The base project requires only sets be discovered and laid. Extend your program to include runs using an interface named RunInterface.py, and file named Run.py. Worth 5 points.
4. If you do #3, create a RunInterface that is a super interface of the SetInterface and RunInterface to deal with both the sets and the runs. Worth five points.
5. The random decisions in automated play make for a really dumb player. Add some heuristics to improve performance in automated play. Discuss each rule you add and why you expect it to improve performance in your README file. Worth two points for each rule (for a max of four rules).
6. The player number switch (-0, -1, or -2) gives the number of interactive players in the game (by default this is 2, indicating two interactive players). Worth ten points.
7. Programming effective game playing has drawn the attention of many computer scientists, mainly in the area of Artificial Intelligence. Read "A Gamut of Games" by Jonathan Schaeffer and "Game Playing: The Next Moves" by Susan Epstein. Write a two-page paper very briefly summarizing the research challenges of intelligent game play and discussing how AI research might be used to improve your Rummy program. Your paper should be grammatically correct, include appropriate bibliographic references, and be submitted in pdf format. Worth ten points.

Files To Be Submitted

Submit the following files:

1. *.java (including Proj2.java and Card.java),
2. p2-output.txt,
3. README, including instructions for how to run your GUI if you implement one for extra credit,
4. any items required for extra credit options