CS 152 Computer Programming Fundamentals
Lab 8: Klondike Solitaire*

Brooke Chenoweth

Fall 2019

1 Game Rules

You are likely familiar with this solitaire card game. An implementation has been included with Microsoft Windows for years.

1.1 Set up

Taking a shuffled standard 52-card deck of playing cards (without Jokers), one upturned card is dealt on the left of the playing area, then six downturned cards (from left to right). On top of the downturned cards, an upturned card is dealt on the left-most downturned pile, and downturned cards on the rest until all piles have an upturned card. The piles should look like the figure.

1.2 Game Play

The four foundations (light rectangles in the upper right of the figure) are built up by suit from Ace (low in this game) to King, and the tableau piles can be built down by alternate colors, and partial or complete piles can be moved if they are built down by alternate colors also. Any empty piles can be filled with a King or a pile of cards with a King.

The aim of the game is to build up a stack of cards starting with 2 and ending with King, all of the same suit. Once this is accomplished, the goal is to move this to a foundation, where the player has previously placed the Ace of that suit. Once the player has done this, they will have "finished" that suit, the goal being, of course, to finish all suits, at which time the player would have won.

There are different ways of dealing the remainder of the deck. In our game, we are using a variant where cards are turned over one at a time and there are unlimited passes through the deck.

*This lab is based on the project at `http://ljing.org/games/klondike/`

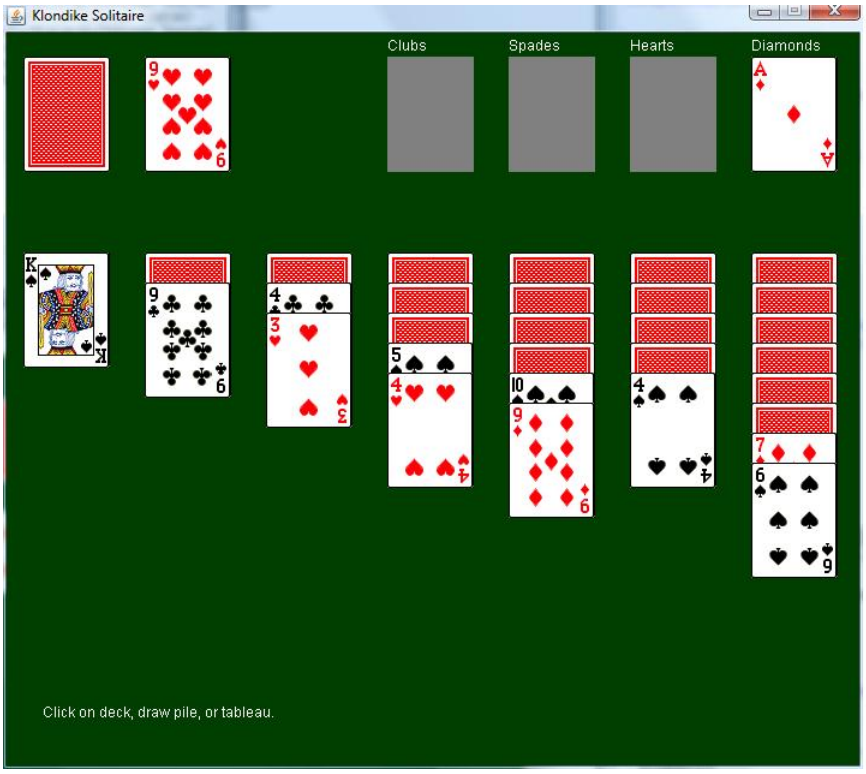

Figure 1: Example of Klondike game

2 Program description

You are going to write two classes for this assignment: `Card` and `Deck`.

I have provided you with five files of code. Two of them contain the enums used for a card's rank and suit. The other files contain the solitare game logic, GUI, and testing code. You will not be changing them.

- `Suit` is an enum containing constants for all the suits in a standard deck of cards.
- `Rank` is an enum containing constants for all the card ranks.
- `KlondikeModel` contains the logic for the rules of the game.
- `Klondike` starts the GUI and plays the game. This code will not run until you have implemented the methods in Card and Deck.
- `KlondikeTester` contains code to test the various methods. If you run this class, it will report how many tests you passed. You may find this especially useful when you have yet to complete enough methods to be able to play the game.

  Note: Most of the tests for KlondikeModel and Klondike will not pass until you have working Card and Rank classes. Don't worry about them until you have done that.

3 Card Class

`Card` is one of the first class you must complete. A card has a suit, a rank, and a boolean indicating if it is face up.

3.1 Constructors

The `Card` class has one constructor:

- `public Card(Rank rank, Suit suit)`

  Constructs a new Card with the given rank and suit. Newly constructed cards are face-up by default.

3.2 Methods

The `Card` class has at least the following methods: (You can make more if you feel you need to, for helping your other methods, but I will be using these.)

- `public Rank getRank()`

  Returns the rank of this card: one of ACE through KING.

- `public Suit getSuit()`

  Returns the suit of this card, one of CLUBS, SPADES, HEARTS, or DIAMONDS.

- `public boolean isFaceUp()`

  Returns true if the card is face-up, false if not.

- `public boolean isRed()`

  Returns true if the card is red, false if not.

  Note: clubs and spades are black, hearts and diamonds are red.

- `public void flipOver()`

  Toggles the face-up status of the card. If the card was face-up, it is now face-down, and vice-versa.

4 Deck Class

`Deck` is the other class you must complete. The methods in this class are a bit trickier, manipulating the cards in the deck, transfering cards between decks, etc. I recommend you get the other methods working before you implement the shuffle method, since it is possible to play a (rather predictable) game of solitaire with an unshuffled deck.

A Deck object contains an array of Cards that is large enough to hold all the cards in a standard deck and an int representing how many cards are currently in the deck. The bottom card in the deck wil be stored at index zero in the array.

The Klondike game uses Deck instance for each pile of cards in the game.

4.1 Constructors

The `Deck` class has one constructor:

- `public Deck()`

  A new deck is empty (containing no cards), but has the capacity to hold all the cards in a standard deck.

  Note: there are four suits and thirteen ranks, so a standard deck has fifty-two cards.[1]

4.2 Methods

The `Deck` class has at least the following methods: (You can make more if you feel you need to, for helping your other methods, but I will be using these.)

- `public void add(Card card)`

  Adds card to the top of this deck.

- `public void fill()`

  Fills the deck with all the cards in a standard deck of cards. The specific order does not matter as long as all the cards are there.

  You should not explicitly construct and add all 52 possible cards. Use methods provided in the RANK and SUIT enums to examine all the possible combinations.

- `public Card getCardAt(int n)`

  Returns the nth card in this deck, where card 0 is the one on the bottom. Assumes the deck is not empty. Does not modify the deck.

- `public void moveTo(Deck other)`

  Moves one card from the top of this deck to the top of the other deck.

- `public void moveTo(Deck other, int n)`

  Moves the top n cards from the top of this deck to the top of the other deck, maintaining their order so that the card that was on top of this deck becomes the top card of the other deck.

- `public int size()`

  Returns the number of cards in this deck.

- `public Card getTopCard()`

  Returns the top card on this deck. Returns null if the deck is empty. Does not modify the deck.

[1] Our deck does not have jokers.

- `public void shuffle()`

  Randomly reorders the cards in this deck. You may *not* assume that this method will only be called on a full deck of 52 cards (though that is how it is used in the actual game), but instead you must be able to correctly shuffle any non-empty deck.[2]

  You may want to initially create this method with an empty body while you work on the rest of the program.

  The instructors will not help you on this method. You must research shuffling on your own.[3] Even if you think you have a good idea how to do this on your own, I recommend you at least do a quick Google search to see how other folks have tackled the problem of shuffling an array.

  Mention in a comment in the method what source(s) you used when researching shuffling algorithms. Include URLs for web pages, title and author for books, names for people, etc.

4.3 Card Images

I have provided you card image files[4] to use in the game. In order for the GUI code to be able to find them, they must all be placed in a folder named `card-images` inside your project (but not inside the src folder).

5 Turning in your assignment

Submit your `Card.java` and `Deck.java` files to the UNM Learn. Do not attach `.class` files or any other files.

[2] Shuffling an empty deck is trivial.

[3] At this point, you know enough programming terminology to understand much of the jargon out there.

[4] from `http://www.jfitz.com/cards/`