716180 Data Structures and Algorithms

Assignment Two

Due 6pm Wednesday 7 May 2014

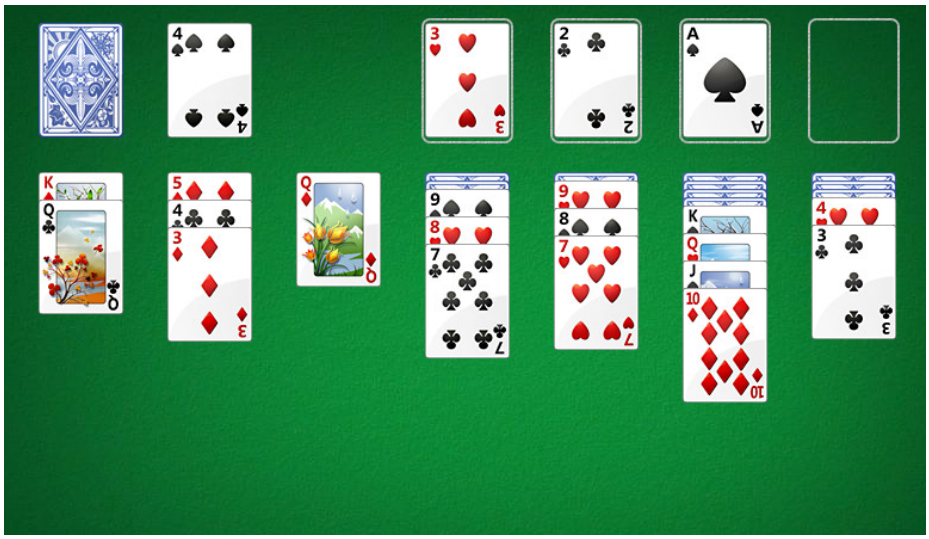

Fig. 1. A screenshot of the solitaire game

Instructions

- You may work in **pairs** to complete this assignment.
- Please submit a `assignment2.zip` package containing only the source code (.java) and other relevant files such as pictures (if any) to the correct assignment folder on AUTOnline by 5pm on the due date.
- On top of the main source code file (`Solitaire.java`), please write as comments the following information:

  - Your two names and student IDs
  - Contribution of each member of the pair (if equal contribution, just write "Equal contribution")
  - If you're submitting an extension, a description of that extension (see the Extension section below)

- You are only allowed to use classes from the Java standard library.
- Marks are allocated for program design, readability as well as correctness of the code

1 Your Tasks

In this assignment, you will make a solitaire game, see Fig. 1. The rules of the game are as follows:

- The game has a card deck, four card stacks, and seven card lists.
- The card deck starts off without any cards revealed, and the card stacks start off empty.
- The first lists starts with one card, the second with two, and so on, up to the seventh list with seven.
- The tail card of any list is always revealed (unless the list is empty).
- You can reveal a card from the deck (one card at a time).
- In a list, you can only put a red card (hearts, diamonds) on top of a black card (spades, clubs), or vice versa.
- The Ranks of the cards go king – queen – jack – 10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – 2 – ace, from highest to lowest.
  The revealed cards in a card list must have consecutive ranks in decreasing order.
  The cards on a card stack must start from ace and have consecutive ranks in increasing order.
- Each card stack corresponds to a specific suit (Hearts, Clubs, Spades, Diamonds). You can only place cards to the stack that corresponds to their suits.
- You can *cut* a card list into two lists, where the lower list contains only revealed cards. And then link the lower list to the end of another list
- Once all four card stacks have 13 cards in them, you win.

Extension

If you want to, you can add something to the Solitaire game in addition to implementing all the specifications. This can be absolutely anything you want – as long as what you have at the end is (basically) a Solitaire game, anything else goes. Some suggestions include: implement an improved GUI which includes better art works on each card, sound effects, more sophisticated game rules, and winning animation on the screen.

2 Classes

You should implement the following classes, along with these fields and methods. You can add additional classes, methods or fields as you see fit.

1. `Card` (See Slides of Lecture 7)
   - **Fields**
     - `cardIndex:` The card's index, from 1 to 52.
   - **Methods**
     - `getSuit:` Returns the card's suit.
     - `getValue:` Returns the card's value (such as 10, king, etc).
     - `colour:` The colour of the card is 'red' if this card is a heart or diamond, and 'black' otherwise.

- `toString`: Returns a string representation of this card, including its suit and rank. *Example:* Ace of clubs would be `ClubA`, ten of diamonds would be `Diamond10`, and queen of spades would be `SpadeQ`.
- `paintThis(Graphics g)` Draws the card. In the simple GUI, this should draw a `Rectangle`, with the string representation of the card written in the colour corresponding to the colour of the card (either black or red). (You don't have to make it pretty)

2. `CardDeck`
   - **Fields**
     - `cards`: A *circularly-linked list* storing all the cards in the deck.
     - `currentCard`: The current card.
   - **Methods**
     - `drawCard`: Open the next card, if this is the tail card, return `null`.
     - `takeCard`: Delete and return the current card (so we can place it in a list or a stack).
3. `CardList`
   - **Fields**
     - `cards`: A *linked list* to store the cards in this list.
     - `openedIndex`: The index of the first opened card.
     - `tailCard`: The tail card.
   - **Methods**
     - `cut(int index)`: Separate the list into two: `[0 .. (i-1)]` and `[i .. count]`. Open the card at `(i-1)` if necessary. Return the second list.
     - `link(CardList other)`: Join this list to the tail of the other list, if the rules allow this.
     - `add(Card c)`: Add `c` as the new tail card, if the rules allow this.
     - `moveTail`: Delete and return the tail card. Set the card beneath it as the new tail card. Open the new tail card if necessary.
4. `CardStack`
   - **Fields**
     - `stack`: A *stack* to store the cards.
   - **Methods**
     - `add(Card c)`: Adds `c` to the top of the stack, if the rules allow this.
5. `PaintingPanel`
   - **Methods**
     - `paintComponent(Graphics g)`: Draw the game stored in `game`.
6. `Solitaire`
   This is the main class of the game.

– **Fields**
  - `deck`: A `CardDeck` for the current game's deck.
  - `stacks`: An array of `CardStack`s (of length 4).
  - `lists`: An array of `CardList`s (of length 7).

– **Methods**
  - `main(String[] args)`: Should create a new `Solitaire`, then call `showGUI`, passing the newly-created `Solitaire` object to it. Lastly, should call the `Solitaire` object's `startGame` method.
  - `showGUI(Solitaire game)`: Should create a GUI and display the game.
  - `executeCommand(String commmand)`: Perform whatever `command` indicates if the rules allow it and return a success message. If the command is invalid, return a warning instead.
  - `startGame`: Runs a loop that accepts commands until either a quit command is given or the player wins. Should attempt to perform any commands given, and prints all messages back to the user.

Note 1: Data Structures Implementations

For this assignment you will need to use several data structures taught in class such as (circularly) linked lists and stacks. For this you will need to use your own implementations together with the appropriate interface and abstract class files.

Note 2: Shuffling the Cards

As part of creating a new `CardDeck` for solitaire, you will need to shuffle it. You can do this using the Fisher-Yates algorithm, which is described as follows:

1. Create an array of length 52, filled with each number from 1 to 52 in order from smallest to biggest.
2. Iterate over the array backwards (i.e. starting from the end and going back). At each entry, do the following:
   (a) Pick a random integer j between 0 and the current index i.
   (b) Switch the elements at position i and position j in the array.

After you have finished this, the array will be in a random order, and you can use this to generate your shuffled `CardDeck`. For code of this algorithm, Google "Fisher-Yates Java".

3 User Interface

You need to implement two user interfaces:

(i) **Command-line User Interface (CUI)**:
  - When the game starts, the CUI should print a *welcome message*.

```
-------------Current Step--------------
Card Deck: Not Empty     Open Card: Spade4
Card Stacks : Heart3 Club2 SpadeA Empty
Card Lists :
1: DiamondJ-ClubQ
2: Diamond5-Club4-Diamond3
3: DiamondQ
4: Back-Back-Back-Spade9-Heart8-Club7
5: Back-Back-Heart9-Spade8-Heart7
6: Back-Back-Back-SpadeK-HeartQ-SpadeJ-Diamond10
7: Back-Back-Back-Heart4-Club3
Your next move:  DrawCard
```

- At each stage of the game, the CUI should print out a current description of the game, which include information about the card deck, card stacks and card lists. See below for a screenshot of the CUI which corresponds to the game shown in Fig. 1.
- Then the CUI should ask for the player to type in a command by printing "`Your next move:`". If the move is not valid, print out "`Invalide move`" and ask the player to give another command. The possible commands are:

  - `DrawCard`: Open the next card on the card deck.

  - `DeckTo x` : Move one card from the deck to the `x`th list. For example the command `DeckTo 3` moves the card that is currently open in the card deck to the third list.

  - `Link c x`: Suppose `c` is a revealed card in a card list, and $1 \leq x \leq 7$. This command moves all cards below and including `c` in the same list to the `x`th list. For example the command `Link Spade9 6` moves all card below and including `Spade6` to the 6th card list; see the screenshot below.

  - `Send c`: Suppose `c` is a tail card of a card list. This command moves the card `c` to the stack that corresponds to its suit.

  - `Restart`: Restart the game.

  - `Quit`: Stop the game.

- If the game is won, the CUI should print out a line congratulating the player.

(ii) **Graphical User Interface (GUI)**: The GUI includes the following:

- A panel which paints the current state of the game (the card deck, the card lists, and the card stacks)
- A menu bar with a single item called "Game" that allows the player to restart a new game
- You can implement a mouse listener which allows the players to drag and place the card as in a typical Windows solitaire game. BUT THIS IS NOT COMPULSARY.

  If you don't want to use a mouse listener, you can allow the user to play the game by using a *TextField* and a *Button*. Within the TextField, the player can type in commands as in the CUI. Then the player can execute the command by pressing the Button.

4 Marking schedule

There are **40** total marks available for this assignment. There are also **10** extra marks available for teams that complete an extension.

Question 1 (5 marks). Create the `Card` class as described above.

Question 2 (7 marks). Implement the linked list and stack data structures that you need for the game. You need make your classes implement the appropriate interfaces of these data structures.

Question 3 (9 marks). Implement the `CardDeck`, `CardList`, `CardStack` classes.

Question 4 (11 marks). Implement the `Solitaire` class, and make a CUI as described above.

- Implement a card shuffle method for shuffling the cards **(3 marks)**
- Implement the CUI **(3 marks)**
- Implement other parts of the `Solitaire` class that makes the game playable with the CUI **(5 marks)**

Question 5 (8 marks). Implement a simple GUI for your solitaire game.

Question 6 (10 marks, extra marks). Implement some non-trivial extensions of the game (e.g. improve the GUI, more sophisticated game rules, etc.) Marks will be allocated to the technicality and improvement on gameplay of your extension.