CS/MCS/CSP Object-Oriented Programming MT2014 Assessed Practical Assignment — draft 3

1 Solitaire

Your assignment is to complete an implementation of the card-game, *Solitaire*. Solitaire is a game for one player, using a standard set of 52 playing cards. There are many variations of the game, and the aim of this practical is to implement a version of *Klondike* as a Java program. The emphasis of the practical though is to demonstrate your skills of object-oriented design, rather than your skills of graphical design, and for this reason you are provided with a rudimentary framework that can

- draw a pack of cards,
- respond to mouse events and button presses, and
- shuffle a collection (i.e., Java `List`) of cards.

The code that is provided is deliberately *not* well-structured as an OO program, in order to leave you with considerable freedom in how you construct your solution. As mentioned again below, you will be expected to provide with your solution code documentation for your design decisions in the form of UML diagrams and supporting text.

Rules of the Game

Three-deal Klondike is played with a single standard pack of 52 playing cards, namely 4 suits (Clubs ♣, Diamonds ♢, Hearts ♡ and Spades ♠) each of 13 cards (in the order A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K). A shuffled pack is taken and 28 cards are laid out in 7 piles; these 7 piles are laid out vertically in the lower part of figure 1. Initially the first pile on the left contains 1 card, the second pile 2 cards, etc., and the cards that come from the shuffled pack are firstly laid out along the top row from left to right, then from left to right along each subsequent row, with only the left-most card in each row visible to the player. The remaining 24 cards from the shuffled pack are left face-down in the *Talon*, which is the pile shown on the top-left of figure 1. Cards may be taken 3 at-a-time from the Talon (by mousing on it or by using a button), and placed face-up on the *waste pile* just to the right. All 3 cards that are dealt should be visible, although only the top-most card can be moved directly, and any cards that are visible will be covered by any more dealt from the Talon. As the bottom of the Talon is reached fewer than 3 cards may remain, in which case these are visible, and at the next go the remaining waste pile is returned back to the Talon.

The purpose of the game is to end up with the entire pack on the four *Foundations*, which are the piles shown on the top-right of figure 1. An ace (A) may be moved to an empty Foundation (face-up), after which that suit can be built up on that Foundation. We will also allow the top-most card on a Foundation to be moved back into the lower part of the playing area (the *Tableau*). The piles in the Tableau can be built up by moving a card onto the bottom of a pile, so that the card moved is placed on the next card in sequence, and of opposite colour.

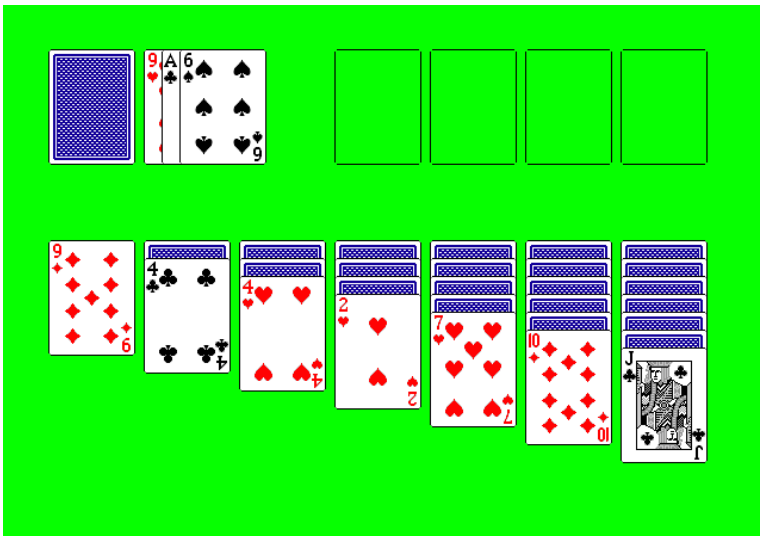

Figure 1: Initial View of Klondike Solitaire

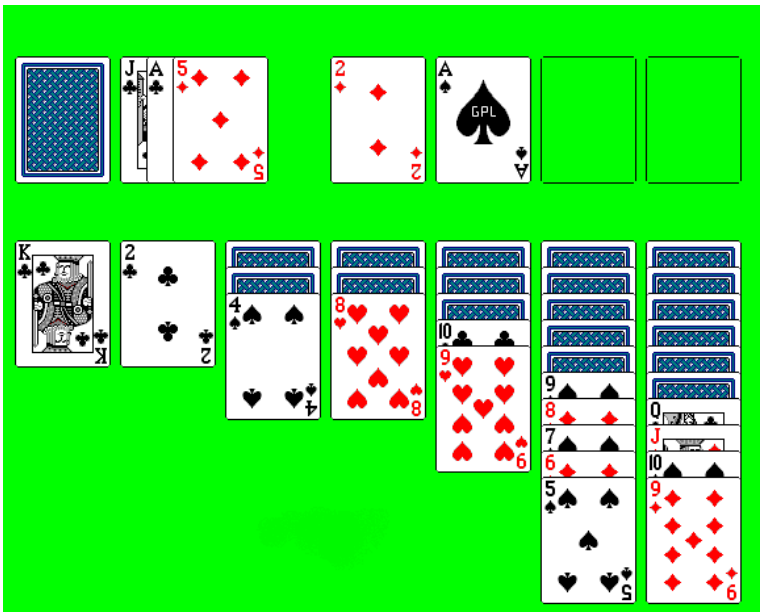

Figure 2: A game after a few moves

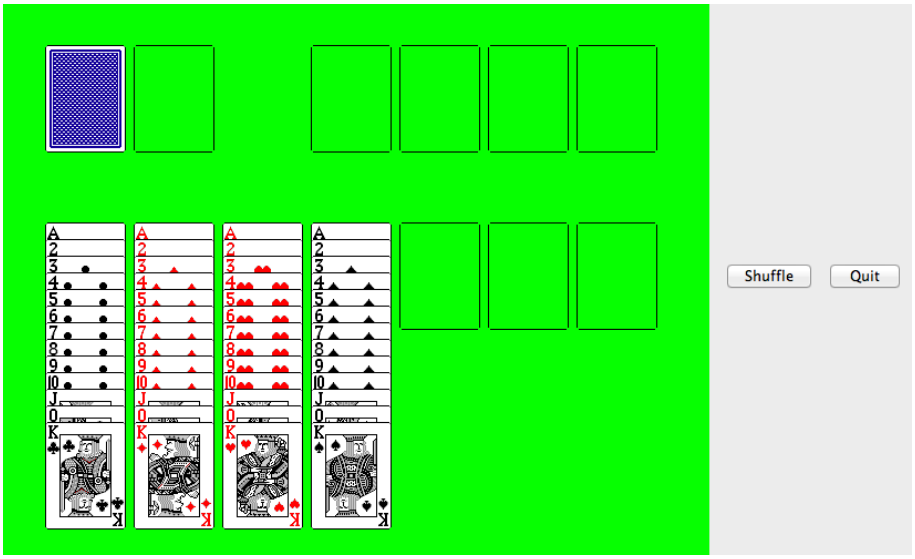

Figure 3: The Starting Point

(Thus, for example, a 5 of diamonds can only be placed onto a pile whose bottom card is a 6 of spades or a 6 of clubs.) A single card can be moved from the top of the waste pile, or from a Foundation, or an entire column of visible cards can be moved from one pile to another (for example, in figure 2, the 7 of spades could be moved with the 6 of diamonds and 5 of spades as a group onto the 8 of hearts). Finally, if a Tableau pile is emptied, a king (or an entire column below a king) could be moved onto that pile (but not back again!).

Single cards (or columns of cards) should be moved by mousing down over the visible part of the card (or the top-card of a column), and moving the mouse over to the destination before releasing the mouse button. (Fancy short-cuts are permitted, but may not gain credit when the practical is marked!)

Existing code

The code that you are provided with implements a Java program (`Solitaire.java`), together with 3 other classes in `Pack.java`, `Card.java` and `CardTable.java` and an enumerated type in `Suit.java`. (The latter is simply used to provide distinct identifiers for the 4 suits). `Card.java` is a prototype for a class for a physical playing card; cards are simply indexed in the range 1–13, with 1 for an Ace, 11 for a Jack, 12 for a Queen, and 13 for a King. There are also two special 'cards' to provide a card outline and a card back; all these images are found in the `cards` directory. The main contribution of `Pack.java` is the provision of operation `Shuffle()`, that takes a `List` of objects and shuffles its contents into random order. Both `Card.java` and `Pack.java` will require extensive modification and re-design, although you may (of course) use the code found there for your own classes.

The `JApplet Solitaire` as given to you just provides code for dealing out the current pack of cards into 4 columns of 13 cards, plus a button that shuffles the pack (figure 3). There is also a simple set of handlers for 3 mouse events, namely button-down, drag, and button-up; they are set-up to provide a rubber-band line, with the idea that these actions can also be used to play the game. All these

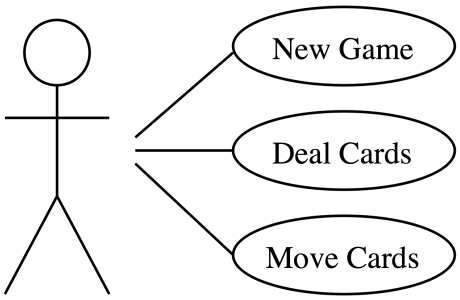

Figure 4: Top-level Use Cases for Solitaire

classes are provided so that you shouldn't have to worry at all about how to work the Java Swing methods, and can concentrate more on the classes and the events that will make up your design and its implementation.

Use Cases

There are just 3 top-level use-cases for this problem, as diagrammed in figure 4. These can be described in outline as follows:

- *New Game:* The pack is shuffled and the cards laid out for a new game.
- *Deal Cards:* Clicking the mouse on the Talon causes (up to) 3 cards to be dealt out.
- *Move Cards:* An attempt is made to move an exposed card from the waste pile or one of the Foundations, or a sequence of cards from the Tableau piles.

The practical

Your goal is a design and implementation of this basic Solitaire game.

> *Attention to the design (as judged by your UML diagrams and report) will count as much as whether your program actually works!*

You should start by copying the source files from the course web-page: `Solitaire.java, Suit.java, Card.java, CardTable.java, Pack.java`. You will also need the directory `cards` (which is provided as a ZIP file), which holds GIF files for each of the cards; `cards` also contains GIFs for a card backs and outlines. (There is also an alternate single ZIP file on the web-page containing the Java files and cards in a form suitable for loading into Eclipse.)

Once you have the source files, you should be able to compile it and run it in the usual way; either by loading all the source files into a new Eclipse project (recommended), or with `javac Solitaire/Solitaire.java` and then running it with `java Solitaire/Solitaire`. See how the button and mouse are used, and then criticise the code you have been given.

Then use UML to design a better, working version of Solitaire, and finally implement it! (Hint: As is common with this sort of design, the best place to start is by detailing the use cases. You may like to know that my model solution contains just over a dozen classes, other than those found in Java itself.)

Note: this practical counts towards your final mark for this course, and so the usual rules about plagarism and examinations *do* apply!

In particular, this means that you must not confer to complete this practical; and if you use published materials you must list them.

Marking

Please take your time on the design phase of this practical; it is expected that this practical will take most people around 2–3 full days to complete.

You are required to produce the following deliverables, all of which will be evaluated:

- An initial evaluation and criticism of the code as provided to you.
- A list of your reasoned design decisions as to how to implement your final program according to object-oriented design criteria.
- A description of the response of your program according to the use cases given.
- A UML diagram (or diagrams) outlining your solution. (Hand-drawn diagrams are certainly allowed, but please make sure that they are legible. You may type the details — such as lists of operations for a class — onto a separate sheet, or sheets, if that is easier.)
- Your complete, commented program code. This should be submitted HOW????

Hints

- Two classical design patterns that you should certainly consider are *Model-View-Controller* and *State*.
- The different piles of cards have similar properties that do differ.
- Your solution should focus on *clarity* and the principles of OO-design; efficiency should definitely be a secondary issue, especially when it comes to relating the mouse positions to cards.
- You should probably separate the question of whether a movement of cards can physically take place from whether that move obeys the other rules of the game.
- I am expecting about a couple of sides of text in your report (in *addition* to any diagrams).