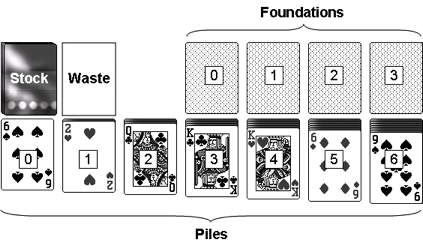
**Solitaire**

In this lab, you will implement the classic game of Klondike Solitaire (you'll probably just recognize this game as "solitaire", however "Klondike Solitaire" is the real name of the game).  We'll use the **Card.java** class (provided, though you may add methods to it) to represent each playing card in the game; open and familiarize yourself with it.  Note that rank is represented as an integer from 1 (ace) to 13 (king), and that suit is represented as a String ("c", "d", "h", or "s").

Our solitaire game will be arranged as shown below.  Cards from the ***stock*** are turned face-up onto the ***waste*** (three at a time), which may then be moved onto ***piles*** numbered 0 - 6.  The goal is to get all the cards up onto the ***foundations*** numbered 0 - 3.  We will use this terminology throughout the lab.



**The Rules**

At the start of the game, a complete deck of 52 cards is shuffled.  Cards are dealt onto the seven piles, with one card on pile 0, two cards on pile 1, and so on.  The top card on each pile is turned face up.  The remaining cards are placed in the stock.

1. The user may transfer the top three cards from the stock to the waste, where they will now be face up.  (If fewer than three cards remain in the stock, then any remaining cards are transferred onto the waste.)  If the stock is empty, the user may transfer all cards back from the waste to the stock, where they will be face down again.
2. The user may transfer the top card from the waste onto any pile.  Likewise, they may transfer all face up cards from one pile onto the top of another.  When all face up cards are removed from a pile, the top remaining card may be turned up.

Cards on piles are subject to the following restrictions:  Any cards placed face up on the piles must alternate in color, with the rank of each card being one more than the card on top of it.  For example, only the 5 of diamonds or 5 of hearts may be played on top of the 6 of spades.  Only a king may be placed on an empty pile. If you've never played Solitaire before, Google "solitaire" and play the doodle above the search results.

1. The user may transfer the top card from the waste or the top card from any pile onto a foundation.  Cards on foundations are subject to the following restrictions: only an ace may be placed on an empty foundation.  Cards on a foundation must be of a single suit, with each card being one more than the card below it.  For example, only the 7 of spades may be placed on top of the 6 of spades.

**The User Interface**

The user will click on the stock to deal three more cards, or, if empty, to reset the stock.  To move a card (or group of face up cards), the user will first click to select the source of the move, and then click to select the destination.  If the move is valid, the selected card will be moved.  Here are some further suggestions:

1. When a card is moved successfully, the original selection is cleared.
2. When a move is denied, the source remains selected.
3. When a card is selected, no cards may be dealt from the stock.
4. A selection may be cancelled by clicking on the selected card.
5. The user must explicitly click on a face down card to turn it up.

**Whatever you do, your user interface must be consistent, and must never crash.**

**The Code**

All the code you'll be writing in this part will be in **Solitaire.java**.  Open this file.  Notice that the stock, waste, foundations, and piles are stored in instance variables, and each is represented in terms of java.util.Stacks of Cards (or you can use your MyStack class, if you've done that lab already).  The SolitaireDisplay class provides a graphical user interface (GUI).  Notice that the Solitaire class maintains a reference to the SolitaireDisplay, and that the display is initialized in the last line of the Solitaire class's constructor (the SolitaireDisplay also maintains a reference back to the Solitaire object).  
  
To show the game state in the user interface, the SolitaireDisplay calls the following methods in the Solitaire class:  
  
 **class Solitaire.java**

|  |  |
| --- | --- |
| Card  **getStockCard**() | Returns the top card, or null if empty |
| Card  **getWasteCard**() | Returns the top card, or null if empty |
| Card  **getFoundationCard**() | Returns the top card, or null if empty |
| Stack<Card> **getPile**(int index) | Returns the entire pile at index |

When the player clicks on the user interface, SolitaireDisplay calls one of the following methods:

|  |  |
| --- | --- |
| void **stockClicked**() |  |
| void  **wasteClicked**() |  |
| void  **foundationClicked**(int index) |  |
| void **pileClicked**(int index) |  |
|  |  |

Finally, SolitaireDisplay provides methods you may call to determine which card is selected in the user interface, and to select and de-select cards:  
  
 **class SolitaireDisplay.java**

|  |  |
| --- | --- |
| boolean **isWasteSelected**() |  |
| int  **selectedPile**() | Returns the index of the selected pile (0 - 6), or -1 |
| void  **selectWaste**() |  |
| void  **selectPile**(int index) |  |
| void  **unselect**() | Clears any selection |

**Exercises**

1. In Solitaire.java, complete the constructor, which should initialize all instance variables.  You will need to create a standard deck of 52 cards, then shuffle it, and then deal out these cards to the piles, leaving the remaining cards in the stock.  You will also need to implement getStockCard, getWasteCard, getFoundationCard, and getPile.  When you have completed this, run the main method to verify that you see a valid and random initial arrangement of cards.  
     
   **Note:**  The algorithm you use to shuffle the deck must be equally likely to choose from any of the 52! arrangements of cards.  One easy way to implement this is to first create the deck of cards in an ArrayList.  Then remove a random card from the deck and move it to the stock.  Keep doing this until there are no cards remaining in the deck.  
     
   **Food for thought:**  For an ArrayList of n elements, what is the running time of this shuffling algorithm?  What simple modification to this algorithm will dramatically improve its running time?
2. Modify the code so that the user can click on the stock to transfer the top three cards from the stock to the waste, where they will now be face up (if fewer than three cards remain in the stock, then any remaining cards are transferred onto the waste).  If the stock is empty, then clicking on the stock should transfer all cards back from the waste to the stock, where they will be face down again.  Test that you can now cycle through the stock (you should see eight cards in each cycle).
3. Modify the code so that the user may transfer the top card from the waste onto any pile if the move is legal (this will involve modifying wasteClicked to select the waste and pileClicked to perform the move and clear the selection, if legal).  
     
   Your code should enforce that any cards placed face up on the piles must alternate in color, with the rank of each card being one less than the card below it.  For example, only the 5 of diamonds or 5 of hearts may be played on top of the 6 of spades.  Only a king may be placed on an empty pile.  
     
   **Hint:**  Consider writing a method like the following:  
     
   /\*\* returns true if *card*can be added to the pile at *index*, else false \*/  
   private boolean canAddToPile(Card card, int index)  
     
   Test that you can now move cards from the waste onto piles, when legal.

**Note:**  public static final String constants representing the different suits exist in the Card class; you should not be comparing a Card's suit to String literals "d", "h", "c", or "s" due to the possibility of confusion (e.g. "s" vs. "spades", spelling errors, etc.). Similar constants for rank could also be included, but haven't been due to how easy it is to compare integers.

1. Modify the code so that the user may transfer all face up cards from one pile onto the top of another, if it is legal to do so.  
     
   **Hint:**  Consider writing methods like the following:  
     
   /\*\* removes all face-up cards from the top of the pile at index, and returns a stack containing those cards (in reverse order) \*/  
   private Stack<Card> removeFaceUpCards(int index)  
     
   /\*\* removes all cards from *cards*and adds them to the top of the pile at *index*(in reverse order) \*/  
   private void addToPile(Stack<Card> cards, int index)  
     
   Test that you can now move cards between piles, when legal.
2. Modify the code so that the user may click to turn up a card on the top of a pile (if you have not already done this programmatically).  Test that this works correctly.
3. Modify the code so that the user may transfer a card from the waste to a foundation, or from a pile to a foundation, if it is legal to do so.  Your code should enforce that cards on foundations are subject to the following restrictions: only an ace may be placed on an empty foundation.  Cards on a foundation must be of a single suit, with each card being one more than the card below it.  For example, only the 7 of spades may be placed on top of the 6 of spades.  
     
   **Hint:**  Consider writing a method like the following:  
     
   /\*\* returns true if *card*can be added to the foundation at *index*, and false otherwise \*/  
   private boolean canAddToFoundation(Card card, int index)  
     
   Now play your game until you win!  Be sure to test various error / edge conditions.

**(Advanced) Additional Challenges**

Have a nicely working game? Here are a few suggestions you might consider implementing:

•    Allow the user to undo a move, possibly all the way back to the beginning of the game.  
•    Allow the user to move a card from a foundation to a pile.  
•    Show all three cards moved from the stock to the waste.

•    Use ***enum****erations*, rather than public static final variables, for rank and suit.  
•    Provide a shortcut for sending all cards to the foundations that can legally be moved there.  
•    Have your program celebrate when they user wins.  
•    Allow the user to start a new game when they win or give up.  
•    Allow the user to move only some of the cards in a pile to a new pile.    
•    Allow the user to double-click on a card to move it to a foundation.  
•    Provide a scoring system.

•    Keep track of the time elapsed in the game.  
•    Have your game suggest a possible move.  
•    Automatically move cards to foundations when possible.

* Introduce an auto-play mode.

*This project based on the* ***Solitaire*** *lab by Dave Feinberg*

*https://sites.google.com/site/feinbergcompsci/home/hcs2/labs/solitairelab*