Programming Project 9

Assignment Overview
This assignment will give you more experience on the use of classes.

This assignment is worth 50 points (5.0% of the course grade) and must be **completed and turned in before 11:59 on Monday, April 6th , 2009.**

Background
You will implement the Solitaire card game called Easthaven. It is a fairly simple game and does not have much strategy, but our goal is to simply enforce the rules for players. To see a copy of the rules, goto http://www.worldofsolitaire.com , click Solitaire at the left-upper corner, click "select game", and choose "Easthaven".

More important than anything, familiarize yourself with the game by playing the online version before considering programming. It is much easier to understand the game by playing it rather than reading the rules. A sample implementation, Easthaven, is provided.

The game play proceeds as follows:

1. The Start
    a. There is one deck which is called "stock"
    b. There is one tableau. It begins with 7 columns with 3 cards in each column. In each column, initially, only the top card is visible.
    c. There is one foundation. Cards are moved from a row of the tableau to the Foundation. Foundation starts with four empty spots. (each for one suit)
    d. You can deal one card to each row of the tableau from stock, until there are no cards left in stock.
2. The Goal
    a. The goal is to move all the 52 cards to the foundation
3. Rules of Play
    a. Foundation
        i. Built up by rank and by suit from Ace to King. Ace is low.
    b. Tableau
        i. Built down by rank and by alternating color. For example, you can play a Two of Hearts on a Three of Spades (the column goes down by rank, and alternates colors)
        ii. Either the top card may be moved or complete or partial correctly ranked piles may be moved as a pile. If a pyle is moved, the top card of the pyle must follow the rules (down rank and alternating color) and all the rest of cards in the pile must also follow the rules.
        iii. An empty spot may be filled with any card or correctly ranked pile
    c. Stock
        i. If you cannot make any moves on the Tableau, or any moves from the Tableau to the Foundation, you can take cards from the stock
        ii. **Different**! When the stock is used, a card is added to the end of each Tableau column.
    d. The goal is to get all the cards to the foundation.
    e. To make the project a little simpler, once a card is moved to foundation, it is not allowed to be moved.

Your program allows a user to play the game, ensuring that they follow the rules.

Requirements
Implement the game in Python. You can also try the example game *easthaven.py* to get a feel for the game. Copy easthaven.py and either implementation25.pyc or implementation26.pyc (depending on whether you are running python 2.5 or python 2.6) to your computer if you use IDLE. A *proj09Skel.py* is provided for a possible approach to write your solution. Up to you if you use it or not.

Requirements are:

1. Use the provided Card and Deck Class, found in the cards.py file in the project directory. **Do not modify the cards.py program** as you will only turn in proj09.py
2. You must use functions in this game. Implementations without functions will not be graded.
3. Create one function to be called play(), which starts the play of the game.
4. If the user makes a move that is illegal, you must inform them of the error and let user choose another move.
5. You must determine if a winning position is achieved. If so, report it and stop the game, printing out a "Winning" message.
6. The provided demo uses the following prompts, **which you are also required to use**:
    a. f [row A] [foundation F]' to move the top card of row A to foundation F"
    b. m [number of cards] [row A] [row B]' to move a certain number of cards from row A to row B"
    c. "d" to deal cards
    d. "h" gives help to the user (see the demo program)
    e. "q" means quit
7. The demo program checks for the following errors, **which you are also required to check for**:
    a) trying to move a card off to the foundation that violates the rules
    b) checking on user input to capture incorrect commands
    c) whether the move command is in correct format, and whether it is a valid move trying to deal more cards when there are no cards left in the stock
8. The demo program provides a template for the output format of your program. **You are required to use the output format of the demo program for your program**

Card and Deck Classes plus Display function
We provide a module named ***cards*** that contains a card class and a deck class. We also provide a sample piece of code that demonstrates how to use the cards module. The card and deck classes are general purpose for developing card games so they contain many methods that may not be used in any particular implementation. You are welcome to use all of them, but do not be surprised if there are many that you do not need for this project. For example, in my implementation I used *set_hidden*, *show_card*, *get_rank*, and *has_same_color* functions of Card and *deal, cards_left, shuffle* and constructor method of Deck. That's about it.

The get_rank() method returns the rank of the card: 1 for ace, 2-10 for number cards 2-10 (respectively), 11 for Jack, 12 for Queen, 13 for King.

Deliverables
You must use handin to turn in the file **proj09.py** – this is your source code solution; be sure to include your section, the date, the project number and comments describing your code. Please be sure to use the specified file name, and save a copy of your proj09.py file to your H drive as a backup.

Other good information

Notes:

1. Play with the provided demo program and get a feel for the game
2. Look carefully at the example cardsDemo.py program. It imports the cards module and uses the two classes and gives you a better idea how you can use them. These classes provide methods you may not need, but they should provide almost any method you do need.
3. When using class methods remember the parenthesis—no error is generated for missing parenthesis, but results will not be what you expect.
4. There are multiple parts to the game (setup, printing, game play, starting). Address each one individually and then put them together.
5. You should avoid dealing a card from an empty stock.
6. For playing the game, begin by assuming perfect input. Get that working and add error checking later.
7. Add as much error checking as you can. Error conditions could occupy quite a bit of code!