You will write a complete memory cards program that presents the user a rectangular pattern of cards, face down. Each card will have a number to identify it. The user will choose 2 cards at a time, by entering a number for each card. If the 2 cards match, they are removed from the playing area. When all matches are found, the program outputs the number of attempts required to complete the board.

This first project will be a text-based application, run using the java interpreter on acad’s command line (or a PC’s). When started, the game layout is 30 memory cards, randomly placed, in a 6 x 5 grid. Each memory card is an object that contains information about itself.

You will define two object types for this program, one for a card, and one for the layout of 30 cards. Of course, there will be a main class that controls these objects.

The class for a single card has a symbol as its primary attribute. You may also want an attribute that indicates whether the card is face up or face down. Other attributes are possible. You decide what your cards’ symbols will be. You can use letters, words, or numbers. If you work on a PC, you can use higher ASCII characters to draw box and other shapes (there’s one on this page). Make sure your readme states if you use characters that don’t show up on acad. You should pass in the symbol when the card is constructed. For example, if your cards use words, you could construct a card with new MemoryCard(“Parson”).

For the layout’s class, you need to keep track of the 30 cards in known locations (by row and column? how to work with numbering 1 to 30?), which cards have and haven’t yet been matched, how many matches so far, how many attempts so far, and possibly other data. Therefore, you need to declare attributes for references to the cards (do you construct them in this object?), and the other values. There are several design decisions here (where do you keep track of whether a card has been matched? etc.).

The main class, to be named Memory, has a layout for the cards declared in its main. There are likely other attributes. You must write methods to perform the following tasks:

- set up the layout; For example, you need method addCard() that passes a reference to a memory card to the Layout.
- input a pair of memory cards; you must use methods from the Input package (make sure /export/home/public/spiegel/cis421/java is in your CLASSPATH variable)
- determine the result of the present attempt; state whether a match was made, how many attempts so far, and how many matches have been made. This info must be obtained via calls to get functions of a memory card layout object and be printed below the board, which is printed in its updated status.

Notes:
- The memory card layout may be global in the main class, as it is likely that this class will later morph into a GUI.
  - Note: CardLayout is a pre-defined layout type in the awt. Avoid using this identifier to name the memory card layout
- You must use printf to line up numbers properly.
- To read the user’s input, you must employ a method from classes in the Input package.
- Show cards that have been matched face up, or show the matches off to the side, with the place in the layout blank.
- All object-type classes MUST have appropriate set and get methods. This may save work later.
- Keep in mind that this project will be repeated several times this term, with all subsequent versions windows based. You should be cognizant of this fact while making design decisions for this project.
- Turn in a file, named readme.txt, (all lower case, this exact name) that details any and all design decisions you made, along with any other information you would provide to someone who purchased your product (including how to play)
- To try the game out, one implementation is at http://www.gamesgames.com/game/card-match
- This is a senior/graduate level course. Proper style and modular design is a must. Substantial penalties, up to and including your program not being graded, will be levied for lazy, incomplete, or chintzy style, and/or not employing a proper modular design.
- Graduate Students Only: Log the results of each game play in a file named log.txt, that contains a list of integers representing how many turns were required in each game played so far. When the user starts a game, print how many games have been played, and the maximum, minimum, and average number of turns required to find all matches, as stored within this history, when the initial board is printed. It is not retained after subsequent attempts.

Turnin: All .java files. Do not turnin the Input pkg (but you must properly access at least one input routine within the package).