PegSolitaire

Objective: To create a game that uses conditionals and iterative statements.

Background:
Peg Solitaire is a board game for one player that uses either pegs or marbles as pieces. The objective is to remove all but one of the pieces from the board to win. The board is a 7x7 grid of holes with the corners removed (picture above). To start, pieces are placed in all but the center hole. Pieces may jump adjacent pieces horizontally or vertically if there is an empty hole two spaces away. No diagonal move is allowed. The jumped piece is removed and the game continues until there are no valid moves available. The less pieces remaining the better.

Discussion:
Peg Solitaire requires at least two classes. One class description will be for the board (**PegBoard**) and a second for playing the game (**PegSolitaire**). Mr Greenstein will provide the **PegBoard** class. Your job will be to construct the **PegSolitaire** class.

The **PegBoard** class constructs the initial board with pieces in all the locations excluding the middle (see right). Pieces are identified by their row and column indices. **PegBoard** contains the following methods for accessing and modifying the board.

```
 col 0   1   2   3   4   5   6
row
 0          | P | P | P |
            -------------
 1          | P | P | P |
  -------------------------------
 2 | P | P | P | P | P | P | P |
  -------------------------------
 3 | P | P | P |   | P | P | P |
  -------------------------------
 4 | P | P | P | P | P | P | P |
  -------------------------------
 5          | P | P | P |
            -------------
 6          | P | P | P |
            -------------
```

- **void printBoard()** - Prints the board as shown on the right.
- **boolean isValidLocation(row, col)** - Returns true if the row and column values are on the board. Example: (2, 2) and (4, 6) return true. (1, 5) and (-21, 3) return false.
- **boolean isPeg(row, col)** - Returns true if a peg is in location (row, col); false otherwise. Precondition: (row, col) must be a valid location.
- **void putPeg(row, col)** - Places a peg into the (row, col) location even if a peg is already there. Precondition: (row, col) must be a valid location.
- **void removePeg(row, col)** - Removes the peg from the (row, col) location even if the location is empty. Precondition: (row, col) must be a valid location.
- **int pegCount()** - Returns the number of pegs remaining on the board.
- **int getBoardSize()** - Returns the side length of the board (7).

The **PegSolitaire** class has the **main** method. It provides a user interface, decides which moves are valid, manages the pieces in the **PegBoard**, and determines when the game is over.

The user interface is through the terminal window. The interface will ask for the "jumper peg" location. The interface should be forgiving of bad input and ask again. For example, the first three inputs below are rejected and the last one accepted:

```
Jumper peg - row col (ex. 3 5, q=quit) -> 2
Jumper peg - row col (ex. 3 5, q=quit) -> hello
Jumper peg - row col (ex. 3 5, q=quit) -> 5 0
Invalid jumper peg -> 5 0
Jumper peg - row col (ex. 3 5, q=quit) -> 3 1
```

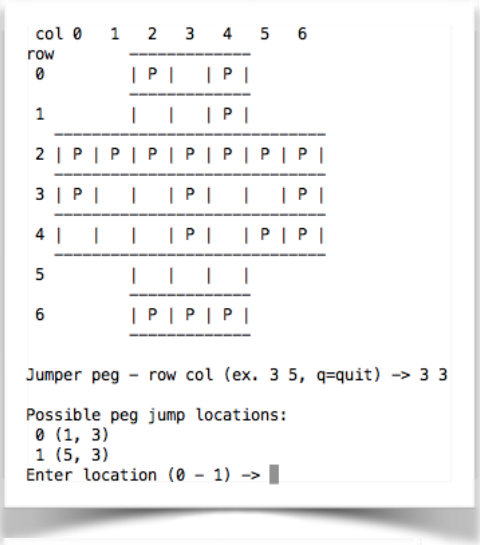
The user can also quit out of the game at any time by entering a "q".

To handle the input, the **String** class has a **split** method that you can use to parse the input line. The following code takes a **String**, separates the values delimited by spaces, and returns an array of **String**s.

```
String str = "1 4";
String[] values = str.split(" +");
// values[0] = "1" and values[1] = "4"
```

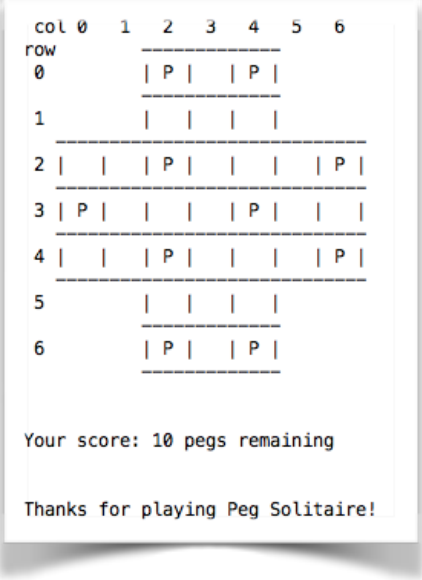
Use the values in the **String** array to determine if the input is valid, then perform the proper action.

The **PegSolitaire** class should recognize when the selected peg has no moves, one move, or more than one move. If there are no moves, then the program should announce "invalid jumper peg" and prompt for another input. If there is only one possible move, the move should be executed immediately. If there is more than one move, the interface should give the user the choice. For example, the screen on the right shows the user selected peg (3, 3) and it has two possible jump locations, (1, 3) and (5, 3). The program displays both moves and prompts the user to choose.



To store a location, a **Location** class has been provided in your zip file. A list of possible **Location**s can be stored in either an array or **ArrayList**.

After each move, **PegSolitaire** should determine whether there are valid moves remaining. If not, it should announce the total number of pegs remaining and end the game (example in figure below right).



Assignment:
Download the **PegSolitaire.zip** file from Mr Greenstein's web site and unzip. It will create a **PegSolitaire** directory containing the files **PegSolitaire.java**, **PegBoard.java**, **Location.java**, and **PegSolitaire.jar**. You will need to copy over your **Prompt.java** file to this directory and use it. Do all of your work in the **PegSolitaire** directory and construct your game inside the **PegSolitaire.java** file.

The **PegSolitaire.jar** file contains a working game from Mr Greenstein. Play the game several times to get a feel for how your game should look. To run Mr Greenstein's game, enter:

```
java -cp PegSolitaire.jar PegSolitaire
```

Be sure not to include "`-cp PegSolitaire.jar`" when compiling or running your version of the game.