CSCI 2110
Data Structures and Algorithms
ASSIGNMENT NO. 1

Date Given: Tuesday, September 18, 2018
Date Due: Tuesday, October 2, 2018 at 11.55 p.m. (5 minutes to midnight)
Submission on dal.ca/brightspace

Welcome to your first assignment. The problem statement in the assignment is long, but that's because the question specifies the requirements in detail. Develop the program class-by-class, and module-by-module in a step-wise fashion, including tests at every stage.

Most board games have pieces that move across the board with certain set rules about how far and in which direction they may move. In this assignment, you will set up the infrastructure for a board game and demonstrate its operation.

Start by defining ~~an abstract~~ a class called **Piece** with the following fields that describe the piece.

- Name: A String (such as Jedi, Droid, etc.)
- Colour: Another String (such as Black, Red, Green, Blue, etc.)
- Position: Assume that the games for which a piece may be used are played on 8 X 8 boards – the position is a coordinate (i, j) on such a board, with each coordinate taking on a value between 0 and 7. (The top left corner of the board is taken to be the origin (0,0).) You can define (x,y) position in a separate class.

Implement:

- A constructor that creates a Piece object with the given name, colour and position.
- Appropriate get and set methods
- toString method to display the object's attributes.

A **SlowPiece** is a piece that moves left or right exactly one step in every move. Define a class called SlowPiece that extends Piece, implementing a constructor and redefining the inherited toString method. In addition, implement

- A method move (String direction) to move the piece left or right by one space. This method will accept a direction parameter. If the move is requested that would take the piece off the edge of the board, simply return without doing anything.

A **FastPiece** is a piece that can move left or right as many steps as specified. Define a class called FastPiece that extends Piece, implementing a constructor and redefining the inherited toString method. In addition, implement

- A method move(String direction, int n) to move the piece left or right by as many spaces as needed. This method will accept a direction parameter and the number of spaces. As before, if a move is requested that would take the piece off the edge of the board, simply return without doing anything.

A **FlexiblePiece** is one that can move right or left as well as up or down – this leads to **slow flexible** and **fast flexible** pieces. Extend the class hierarchy developed so far to include these two new kinds of pieces. Class **SlowFlexible** extends **SlowPiece** and class **FastFlexible** extends **FastPiece**. Define methods in these classes as needed.

Next define a class called **Board** that would hold an 8 X 8 game board. Each position of the board either contains a piece or is empty. Since the pieces move around on the board, any location may contain any type of piece at any given time. The board should be able to accept messages to do the following:

- Add a new piece to the board (note: a new piece can be added to a location only if none exists at that location; display an error message if the piece cannot be added).
- Move a piece at a given location in a given direction by a given number of spaces (display an error message if the piece cannot be moved).
- Display the board, showing the name, color and type of each piece on the board at its current location. You can use a simple text-based display to show the board and the pieces on it. For example, your display might look like this:

```
-           -   -   JediBluS CloneRedF   -   -   -
DroidBluFF  -   -   -        -           -   -   -
-           -   -   -        -           -   -   -
-           -   -   -        CloneBluSF  -   -   -
-           -   -   -        -           -   -   -
-           -   -   -        -           -   -   -
-           -   -   -        -           -   -   TroigRedF
```

In the above display, there is a Jedi (blue colour, slow piece) and a Clone(red colour, fast piece) in the first row, there is a Droid (blue colour, fast flexible piece) in the second row, and so on.
Don't worry too much about getting all the rows aligned properly. As long as it is displayed as a decent 8X8 readable board, it should be OK.

Write an application called **PracticeMoves** that would accept commands from the user to create pieces and move them on an 8 X 8 game board, and print the board. You may use the following command syntax, with each command appearing on a new line. A non-emphasized word implies that it appears verbatim in the command; an emphasized word stands for several possibilities that are explained. A word in square brackets implies it is optional.

- create *location* [fast][flexible]
  Create a piece and place it in a given initial *location* of the form x y. By default a piece is slow and nonflexible. As examples,
  create 1 1 → would create a slow nonflexible piece at location [1, 1]
  create 1 1 fast → would create a fast nonflexible piece at location [1,1]
  create 1 1 flexible → would create a slow flexible piece at location [1, 1]
  create 1 1 fast flexible → would create a fast flexible piece at location [1, 1]

  Obviously, this command should be followed by a prompt asking the user to enter the name and the colour of the piece.

- move *location direction* [*spaces*]
  Move a piece, where direction is one of "left", "right", "up" or "down", of which the last two only apply to flexible pieces. The optional [spaces], which is only applicable to fast pieces, specifies how many places to move the piece.
As examples,

move 5 5 left 3 → moves the piece at (5,5) by 3 spaces to the left. In order for this to work, the piece at (5,5) must be a fast piece.

move 5 5 up 2 → moves the piece at (5,5) two spaces up. In order for this to work, the piece at (5,5) must be a fast flexible piece.

move 5 5 right → moves the piece at (5,5) one space to the right. In order for this to work, the piece can be a slow piece or a slow flexible piece or a fast piece or a fast flexible piece. If it is a fast piece or a fast flexible piece, you would invoke the method move(right, 1) indicating that it is to be moved 1 space to the right.

move 5 5 up → moves the piece at (5,5) one space up. In order for this to work, the piece must be a slow flexible piece or a fast flexible piece. If it is a fast flexible piece, you would invoke the method move(up, 1) indicating that it is to be moved 1 space up.

move 5 5 left 1 → moves the piece at (5,5) one space to the left. In order for this to work, it can be a slow piece or a slow flexible piece or a fast piece or a fast flexible piece. If it is a slow piece or a slow flexible piece, you would invoke the method move(left) since the piece can only move by one space.

- print
  Display the board
- help
  List all the commands that can be used.
- exit
  Exit the game.

Your program must include proper error checks to ensure that the parameters are legal.

Note that the PracticeMoves application does not play a game. It is only a template to be used by others to develop games.

Here's a sample portion of the run of the program (not all methods and commands are shown here):

```
Enter a command (type help for details):
help
Possible commands are as follows:
create location [fast][flexible]: Creates a new piece.
move location direction [spaces]: Moves a piece.
Print: Displays the board.
Help: Displays help.
Exit: Exits the program.


Enter a command (type help for details):
print

-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
-      -      -      -      -      -      -      -
```

```
Enter a command (type help for details):
create 3 3 fast flexible
Input a name for the new piece:
Jedi
Input a colour for the new piece:
Red
Enter a command (type help for details):
print

-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      JediRedFF     -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -

Enter a command (type help for details):
move 7 7
Error: no piece at (7,7)
Enter a command (type help for details):
move 3 3 up 2
Piece at (3,3) moved up by 2 spaces

Enter a command (type help for details):
print
-      -      -      -             -      -      -      -
-      -      -      JediRedFF     -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -
-      -      -      -             -      -      -      -

Enter a command (type help for details):
exit
Done.
```

Submit a zip file containing source codes (.java files), nicely commented and formatted, plus sample outputs.