

CS 326-01-08W
OBJECT ORIENTED DESIGN
WINTER 2008

Program 5
Due: Friday 28 March 2008 11:00 A.M. EDT

Create a folder "PG5" in the root of your shared folder. Put all files pertaining to this project in this folder. Place a (possibly empty) file called "DONE" in this folder when you are ready to have it graded.

For this project, you will write a class that INHERITS from your NDimBoard class. Your Board and NDimBoard classes should solve the exact same problems that it did before:

```
public class NDimMoveBoard extends NDimBoard;
```

It should not be necessary to make a new position class for this assignment.

The difference between NDimMoveBoard and NDimBoard is that in NDimMoveBoard, a user can specify how a piece moves; the knight move is no longer hardcoded.

The move(s) will be specified as a two-dimensional array (array of array) of integer. Each row of the array will specify a different legal move that can be made by that piece. For example, a two-dimensional knight's move would be specified as:

```
1    2
2    1
-2   -1
-1   -2
-1    2
2    -1
1    -2
2    -1
```

There are eight rows indicating to which eight squares a knight can move. The integers in each row specify the offset in each dimension of the next square relative to the present square. A three-dimensional knight's move would be specified as:

```
1    2    0
2    1    0
-1   2    0
-2   1    0
1   -2    0
2   -1    0
```

-2	-1	0
-1	-2	0
1	0	2
2	0	1
-1	0	2
-2	0	1
1	0	-2
2	0	-1
-2	0	-1
-1	0	-2
0	1	2
0	2	1
0	-1	2
0	-2	1
0	1	-2
0	2	-1
0	-2	-1
0	-1	-1

Some pieces with very simple moves have a fairly large matrix. Here is how a two-dimensional queen's matrix would look, considering that a queen can move any number of squares in any direction.

1	0
2	0
3	0
4	0
5	0
6	0
7	0
-1	0
-2	0
-3	0
-4	0
-5	0
-6	0
-7	0
0	1
0	2
0	3
0	4
0	5
0	6
0	7
0	-1
0	-2

```
0    -3
0    -4
0    -5
0    -6
0    -7
1     1
2     2
3     3
4     4
5     5
6     6
7     7
-1    1
-2    2
-3    3
-4    4
-5    5
-6    6
-7    7
1    -1
2    -2
3    -3
4    -4
5    -5
6    -6
7    -7
-1   -1
-2   -2
-3   -3
-4   -4
-5   -5
-6   -6
-7   -7
```

However the moves, simple or complex, it is now the user's job to describe them, not yours. It is your job to emulate them.

`NDimMoveBoard` should contain a new `loadBoard` (`int[] dim, boolean[] data, int[][] move`); it should throw a `MalformedBoardException` (as usual) if there's a problem with any of the parameters. In particular, a row of all zeros in `move` should trigger the exception. Left unchecked, that row could cause your program to enter an infinite loop.

`NDimMoveBoard` should contain **NEITHER** a new `loadInitialPosition()` nor a new `solve()`. Your `Board`, `NDimBoard`, and `NDimMoveBoard` should contain no code that is essentially the same in more than one of these classes. If you have code in `Board`, for example, that looks an awful lot like code in `NDimBoard`, take it out of `NDimBoard` and inherit from `Board`. Code that is

structured the same way and does pretty much the same thing but contains different variables or minor code differences is considered “essentially the same.” Grades are based not only on functionality, but also on elegance of design. It is perfectly OK to go back and mess with your previous code so that the inherited classes work. In fact, you will probably HAVE to do this! However, when you're done fiddling with it, PG3 and PG4 should still work correctly under their own rules.

Methods that other people should be able to access should be public. Internal methods used to do a task should be private (or protected if it needs to be inherited). All data fields should be private or protected. Each class should be in its own file.