

CS 326-01-08W
OBJECT ORIENTED DESIGN
WINTER 2008
Due: Friday 14 March 2008 11:00 A.M. EST

Program 4

Create a folder "PG4" 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 Board class. Your Board class should solve the exact same problems that it did before:

```
public class NDimBoard extends Board;
```

You should also write public class NDimPosition extends Position;

Now, an NDimBoard should have two components: a dimension array of positive integers and a data array of booleans. You can call these arrays anything you like, but for the purpose of this document they are dim and data. dim contains the size information of the maze. For example, if I have a 3-dimensional maze 4x2x3, dim should contain three integers: dim[0]=4; dim[1]=2; dim[2]=3; dim.length will equal the number of dimensions in the maze. data is a single dimension character array that stores all the '1's and '0's in *row-major* order, and we will talk about what that means. But as in illustrative example, suppose my board looks like:

```
TFT  
FTF
```

```
FFF  
TTT
```

```
FTT  
FFT
```

```
TTF  
FFT
```

The data array will be stored as "TFTFTFFFFTTTFTTFFTTTFFFT"

NDimBoard should contain a new loadBoard (int[] dim, boolean[] data); it should throw a MalformedBoardException (as usual) if there's a problem.

NDimBoard should NOT contain a new loadStartPosition. The old ones should still work on the old positions as well as on the new positions. You will probably need to write (private protected) auxiliary methods to make this work. You will probably have to rewrite the old methods as well.

You may write a new solve() method if you like, but you do not have to; in fact, it is better design not to.

A knight moves the same way in n-dimensional space as it does in two-dimensional space: it moves two squares forward or backward in one dimension, one square forward or backward in a different dimension, and no squares along any other dimension. In n-dimensional space, a knight has $4n(n-1)$ possible squares (maximum) to which it can move.

Since NDimPosition inherits from Position, it will contain the row and column information from Position. However, the inherited class might not use it too much. The NDimPosition class should contain an integer array (like dim) which can be accessed and modified with getposn() and setposn(). For example, if posn is {3,1,1}, that will correspond to an 'F' in the above data array. ("matrix" 3, row, 1, column 1, understanding that counting begins at 0.)

Your stack, linked list, and linked list node classes should not be extended. They should just work. If they don't, you probably made a design error in an earlier project.

It is perfectly OK to go back and mess with your Board code so that the inherited classes work. In fact, you will probably HAVE to do this! However, when you're done fiddling with it, Board should work exactly as it did before, and NDimBoard should inherit seamlessly from it.

I will grade this program not just on functionality but also on design. It DOES have to work! But that is not the ONLY thing you need to do. You must write your program well. Focus particularly on these issues:

Try to design your classes and methods so that somebody down the road will be able to inherit from them later relatively easily. They might not be in a position (position? get it?) to go back and mess with the original code to make their inheritance work.

Your inherited classes should contain code that is as small and tight as possible. For example, if you have a two-page solve method and your inherited class also contains a two-page solve method, and 90% of these methods are identical, this is BAD design. Code that is essentially the same should NOT be replicated. Farm it out to a shared method instead.

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