Homework 5 (due Thursday March 9th, 11:00 AM)

In this assignment, you will update your code from Homeworks 3&4.

First, you may have noticed when writing the **`getSymbol(piece)`** method that MATLAB produced a warning: "Argument 'piece' is unused – should this method be Static?" Here 'piece' is the object the method is being called on (that is, the particular instance of the class), but it is not used in the method. What this means is that the method has nothing to do with any particular instance of the class. Instead, the symbol is associated with the *class*. Therefore, change the **`getSymbol`** method to be static (but still abstract in **`ChessPiece`**). This includes changing the definition in all the relevant files, and may include changing the way it is called in **`ChessGame`** and in the display method of **`ChessBoard`**. You don't need to understand how **`ChessGame`** or **`display`** works, only how a call like "**`piece.getSymbol()`**" might change with a static method call.

Second, you will implement a doubly-linked list. Create a handle class called **`LinkedList`**. This class will receive and return node data stored in its list. Use the **`dlnode`** class provided for the nodes of your linked list. This class must support the following methods:

- **`add(list,NodeData,index)`** – creates a dlnode instance with **`NodeData`** stored in the Data property and inserts the node in the LinkedList instance **`list`** at **`index`**. The node currently at **`index`** will come *after* the new node. This method must take advantage of the fact that this is a doubly-linked list (that is, if the index is close to the end, it should start with the last node instead of the first).

- **`add(list,NodeData)`** – creates a dlnode instance with **NodeData** stored in the Data property and appends the node to the LinkedList **`list`**.

- **`addAll(list,otherList)`** – for each node in **`otherList`,** the data stored in the Data property is used to append a node to **`list`.** Adding or subtracting nodes from either list should not effect the other.

- **`clear(list)`** – removes all nodes from the list.

- **`out = contains(list,NodeData)`** – returns true if the **`NodeData`** is contained in the list.

- **`NodeData = get(list,index)`** – returns the data at the given index. This method must take advantage of the fact that this is a doubly-linked list.

- **`index = indexOf(list,NodeData)`** – returns the index of node for the first occurrence of **`NodeData`** in the list or -1 if **`NodeData`** is not in the list.

- **`NodeData = remove(list,index)`** – removes the node at a given index from the list, and returns its data. This method must take advantage of the fact that this is a doubly-linked list.

- **`set(list,NodeData,index)`** – sets the Data property of the node at the given index to **`NodeData`**, overwriting the data already there. This method must take advantage of the fact that this is a doubly-linked list.

- **`out = subList(list,indexFrom,indexTo)`** – returns a *new* **`LinkedList`** object with data from the nodes starting at **`indexFrom`** and ending at **`indexTo`**. It should include the data at **`indexFrom`** and **`indexTo`**. Adding or subtracting nodes from either list should not effect the other.

The class must have the following read-only properties:

- **`Size`** – the size of the array.

- **`CellArray`** – this is a dependent property that compiles the data of the list into a 1-D cell array where the data stored in the cell array of a given index matches the data stored in the node of the linked list with the same index.

Other requirements: Any method that takes an index should give an error if the index is not valid. The index of the first element should be 1. All properties you add to the class other than the two given above should be private. There cannot be any public methods other than the ones listed here.

Note about the two **`add`** methods: MATLAB does not allow two methods with the same name. Therefore, you will need to create one method and use **`nargin`** to differentiate between the two different usages.

Third, you should modify your **`ChessBoard`** code to use your **`LinkedList`** class to store the active pieces. You must not use the **`CellArray`** property anywhere in your code (it will be of use later in the course, when you will use your **`LinkedList`** class again). Note that the property you store your **`LinkedList`** in should be private.

Finally, create a UML class diagram for all the classes involved in the chess game, including the files you were given.

Submit the homework on bCourses. You should create a folder named **`lastname_firstname_hw5`**. Place all of your m-files and UML diagram in this folder and zip it. Please upload this single zip file.