CMPSCI 187 - Spring 2014
Assignment 2

The goal of this assignment is to exercise your understanding of arrays, linked lists, interfaces, classes, inheritance, and more generally the fundamental concepts related to the use, design, and implementation of basic abstract data types. This problem will have you implement a simple game called Hangman using an interface defining the logical view of a game board and an implementation using arrays. In this assignment you will be starting from initial code provided by us and then adding and modifying it.

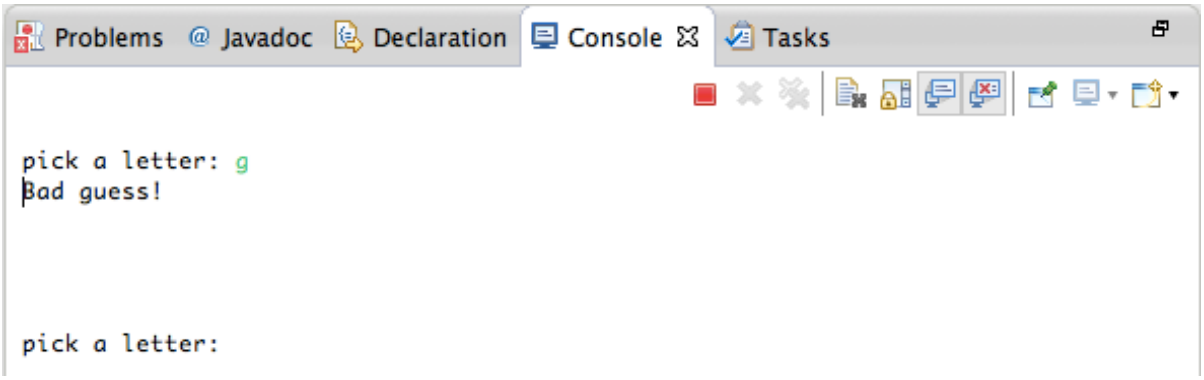
Eclipse and Import

First, you need to download and import the starter code provided on Moodle. The starter code is in the form of an Eclipse project in a zip archive. After you download the zip file from Moodle, import the project into Eclipse as you did in the previous project. Reference the documentation in Assignment 1 to do this properly.

Run the starter code and tests

First, explore the code as it is. Under src, you will see a number of files that you are provided. You should investigate each of these files to get a general understanding of how things are structured. A good starting point is HangManGameArray.java. This file includes the main method which is the entry point into the program. The next important files that you should look at are GameBoard.java and GameBoardArray.java. GameBoard.java is an interface describing methods that you will need to implement - you should read and understand the comments in this file. GameBoardArray.java is an empty implementation of that interface. More on these later.

The starter code you receive *is not a working implementation*. However, you can still run the program. It will allow you to enter guesses, but it will never terminate. You can terminate the running program by pressing on the red square button in the top left of the console buttons:



Hangman Game

The rules for Hangman are simple and you can read up on the details online. Typically two players play the game with pencil and paper. player A thinks of a word and writes down a sequence of underscores representing the word (the game board) that player B needs to guess. For example, if player A picks the word “HeLLo” then she would write:

_ _ _ _ _

An underscore for each letter that has not been guessed yet. Player B will then proceed to guess letters one-by-one that might be contained in the unknown word. If a letter that player B guesses is contained in the word, player A will replace the underscore in the appropriate location with the letter that player B guessed. For example, player B guesses ‘L’ then player

A would write:

```
_ _ L L _
```

Note that if the guessed letter appears in multiple places in the word, each letter is revealed on the game board. If player B guesses a letter that is **not** in the word then player A will draw a part of the hangman picture. For example, if player B guesses an ‘a’ then the result would be:

```
|--------|
[L]

_ _ L L _
```

Where the `|--------|` represents the start of the hangman platform. The `[L]` represents the list of letters that the player has guessed previously. If player B continues to guess incorrectly she will lose the game with the following being written down by player A:

```
-----|
|    |
|    O
|   /|\
|    |
|   / \
|
|--------|
[L, a, b, C, x, g, P, k, z, m, N, q, w, i]
The word was hello!
You lose!
```

Our version of hangman

In our version of the hangman game there are 11 states where the first state (state 0) is empty and the last state (state 10) is the complete hangman platform as shown above. You can see each state diagrammatically in HangManConsole.java. Our version of hangman also considers the difference between uppercase and lowercase letters. That is, the letter ‘A’ is different then the letter ‘a’. Make sure your implementation takes this under consideration. Our version of hangman also assumes that the letters are all the uppercase and lowercase alphabetic characters - not other characters that could possibly be entered in from the keyboard (e.g., “, ‘, <, ?, !).

Part A: Provide an array implementation of the GameBoard interface

In Part A your job is to complete the hangman game by providing an implementation of the GameBoardArray.java file. We have provided a minimal start to your implementation, but the rest is up to you! The GameBoardArray class *implements* the GameBoard interface. A “game board” represents the current state of the game (0 - 10), the word currently being guessed (`_ _ L L _`), and the previous guesses (`[L, a, b]`). It must also know what the word is that is being guessed (e.g., “HeLLo”). Again, you should review the documentation in the GameBoard interface file to gain a better understanding of each method you need to implement. Here is a brief overview of the methods that require your implementation (more details are provided in the comments in GameBoard.java):

boolean isPriorGuess(char guess)
This method returns true if the character ‘guess’ has been guessed previously.

int numberOfGuesses();
This method returns the number of guesses already guessed (excluding repeated guesses)

boolean isCorrectGuess(char guess);
This method returns true if the character 'guess' is a guess that has not been guessed before and is a character that is in the word to be guessed.

boolean doMove(char guess);
This method will play the character 'guess' on the game board.

boolean inWinningState();
This method returns true if the game board is in a winning state.

boolean inLosingState();
This method returns true if the game board is in a losing state.

int currentHungState();
This method returns the current "hung state" (I.e., from 0 - 10).

String toString();
This method returns a string representation of the game board. For example:

```
_ _ L L _
```

String previousGuessString();
This method returns a string representation of the previous guesses. For example:

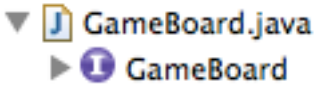
```
[L, a, b, C, x, g, P, k, z, m, N, q, w, i]
```

You will notice that there are 8 TODO comments in the provided GameBoardArray.java file. You must implement each of these TODOs in order to satisfy the public JUnit tests that we provide. Your implementation must use Java arrays internally to represent the game board and the previous guesses. **Do not use** any List abstractions provided by the Java libraries such as ArrayList - this will not be considered a solution. We have also provided the start of a constructor in the starter code that you should use to initialize the state of the game. The constructor takes a single String argument representing the word to be guessed:

```
public GameBoardArray(String guessWord) {
      // TODO (1)
      state     = STARTING_STATE;
}
```

Part B: Provide a linked list implementation of the GameBoard interface

In Part B your job is to complete the hangman game by providing an implementation using a linked list for the hangman game. You need to create a new class that *implements* the GameBoard interface. To do this in Eclipse you select the arrow to the right side of the GameBoard.java file in the Package Explorer in Eclipse to reveal the interface icon:



Right-click on the interface icon and select "New" > "Class". This will bring up a "Create a new Java class" wizard. Give your new class the name "GameBoardLinkedList". This will create a new Java file in the package explorer called "GameBoardLinkedList.java" with all of the methods that you must implement to satisfy the interface in that file. You will need to add a *constructor* for this new class that takes the same arguments as the GameBoardArray class. In order to implement the GameBoardLinkedList you will need to provide a class to implement a linked list of characters - in the same spirit as the LLStringNode class shown in class and covered in detail in the book. You should name this new class LLCharacterNode.

Use the LLCharacterNode in your implementation of the GameBoardLinkedList class. The GameBoardLinkedList class *implements* the GameBoard interface. A “game board” represents the current state of the game (0 - 10), the word currently being guessed (`_ _ L L _`), and the previous guesses (`[L, a, b]`). It must also know what the word is that is being guessed (e.g., “HeLLo”). Again, you should review the documentation in the GameBoard interface file to gain a better understanding of each method you need to implement.

You can test the execution of your implementation by running the HangManGameLinkedList.java file using the *play* button in Eclipse.

Testing

We provide two public JUnit tests you should use to test your implementation:

- GameBoardArrayPublicTest.java
- GameBoardLinkedListPublicTest.java
- LLCharacterNodePublicTest.java

You should run these tests to test your implementation of both classes. You should note that the GameBoardLinkedListPublicTest and LLCharacterNodePublicTest will both contain errors until **you have created** the required classes for Part B. Your grade for this assignment will depend on the results of these tests as well as private tests (that are not visible to you) that we have constructed to ensure that your implementation has not been tailored to the public tests.

Export your completed code

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, right-click on the Java project in the Eclipse package explorer and choose **`File -> Export`** from the menu. In the window that appears, under “General” choose “Archive File”. Then choose “Next” and enter a destination for the output file. The name of the output file is not important.

Submit your code using Moodle

Login to Moodle and upload your exported project to the Assignment 2 submission activity. To make sure that you have uploaded your assignment properly you should try to download it from Moodle - if you can, you know it has been properly received.

Example Solution Output

Here is an example of the output generated from our solution to give you an idea of what the completed program should produce when implemented correctly.

```
_ _ _ _ _ _
pick a letter: i
Good guess!

[i]

_ i _ _ _ _
pick a letter: i
You guessed i already!
guess: _ i _ _ _ _

[i]

_ i _ _ _ _
pick a letter: n
Good guess!

[i, n]
```

```
_ i n n _ _
pick a letter: m
Bad guess!

|--------|
[i, n, m]

_ i n n _ _
pick a letter: w
Good guess!

|--------|
[i, n, m, w]

w i n n _ _
pick a letter: q
Bad guess!

|
|
|
|
|
|
|--------|
[i, n, m, w, q]

w i n n _ _
pick a letter: x
Bad guess!

-----
|
|
|
|
|
|
|--------|
[i, n, m, w, q, x]

w i n n _ _
pick a letter: k
Bad guess!

-----|
|    |
|
|
|
|
|
|--------|
[i, n, m, w, q, x, k]

w i n n _ _
pick a letter: g
Bad guess!

-----|
|    |
|    O
|
|
|
|
|--------|
[i, n, m, w, q, x, k, g]
```

```
w i n n _ _
pick a letter: h
Bad guess!

-----|
|    |
|    O
|    |
|    |
|
|
|--------|
[i, n, m, w, q, x, k, g, h]

w i n n _ _
pick a letter: o
Bad guess!

-----|
|    |
|    O
|    |
|    |
|   /
|
|--------|
[i, n, m, w, q, x, k, g, h, o]

w i n n _ _
pick a letter: e
Good guess!

-----|
|    |
|    O
|    |
|    |
|   /
|
|--------|
[i, n, m, w, q, x, k, g, h, o, e]

w i n n e _
pick a letter: r
Good guess!

-----|
|    |
|    O
|    |
|    |
|   /
|
|--------|
[i, n, m, w, q, x, k, g, h, o, e, r]
You won!
The word was exactly!
Number of guesses: 12
```