

CMPSCI 187 / Spring 2015

Hangman

Due on February 12, 2015, 8:30 a.m.

Marc Liberatore and John Ridgway

Morrill I N375

Section 01 @ 10:00

Section 02 @ 08:30

Contents

Overview	3
Learning Goals	3
General Information	3
Policies	3
Test Files	4
Problem 1	4
Import Project into Eclipse	4
Starter Code and Tests	5
Hangman Game	5
Our Version of Hangman	6
Array Implementation	6
Linked List Implementation	7
Testing	8
Export and Submit	9
Sample Output	9

Overview

This problem will have you implement a game called Hangman, using an interface defining the logical view of a game board. You will write two implementations, the first using arrays, and the second using linked lists. In this assignment you will be starting from initial code provided by us, adding to and modifying it, and creating new classes similar to existing classes.

Learning Goals

- Continue to exercise your understanding of arrays, interfaces, and classes.
- Develop your understanding of the linked list abstraction and its implementation.
- Show ability to break a larger problem down into manageable pieces.
- Read a specification in text and as a Java interface, and demonstrate understanding by implementing it.
- Use existing JUnit test cases.
- Write new JUnit test cases to test specified behaviors.

General Information

Read this entire document. If, after a careful reading, something seems ambiguous or unclear to you, then email cs187help@cs.umass.edu immediately.

Start this assignment as soon as possible. Do not wait until 5pm the night before the assignment is due to tell us you don't understand something, as our ability to help you will be minimal.

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code.

You are responsible for submitting project assignments that compile and are configured correctly. If your project submission does not follow these policies exactly you may receive a grade of zero for this assignment.

Policies

- For some assignments, it will be useful for you to write additional class files. Any class file you write that is used by your solution **MUST** be in the provided `src` directory you export.
- The TAs and instructors are here to help you figure out errors, but we won't do so for you after you submit your solution. When you submit your solution, **be sure to remove all compilation errors from your project**. Any compilation errors in your project will cause the autograder to fail, and you will receive a zero for your submission.

Test Files

In the `test` directory, we provide several JUnit test cases that will help you keep on track while completing the assignment. We recommend you run the tests often and use them to help create a checklist of things to do next. But you should be aware that we deliberately don't provide you the full test suite we use when grading.

We recommend that you think about possible cases and add new `@Test` cases to these files as part of your programming discipline. Simple tests to add will consider questions such as:

- Do your methods taking integers as arguments handle positives, negatives, and zeroes (when those values are valid as input)?
- Does your code handle unusual cases, such as empty or maximally-sized data structures?

More complex tests will be assignment-specific. To build good test cases, think about ways to exercise methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case. Note that we will not be looking at your test cases, they are just for your use.

Before submitting, make sure that your program compiles with and passes all of the original tests. If you have errors in these files, it means the structure of the files found in the `src` directory have been altered in a way that will cause your submission to lose some (or all) points.

Problem 1

Note: You may not use any of the classes from the Java Platform API that implement the `Collection` interface (or its sub-interfaces, etc.). The list of these classes is included in the API documentation, available at: <http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html>. Doing so will be viewed as an attempt to cheat.

Import Project into Eclipse

Begin by downloading the starter project and importing it into your workspace. It is very important that you **do not rename** this project as its name is used during the autograding process. If the project is renamed, your assignment will not be graded, and you will receive a zero.

The imported project may have some errors, but these should not prevent you from getting started. Specifically, we may provide JUnit tests for classes that do not yet exist in your code. You can still run the other JUnit tests.

The project should normally contain the following root items:

src This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified in the problem description and in the code we provide), you can add new files, etc.

support This folder contains support code that we encourage you to use (and must be used to pass certain tests). You must not change or add anything in this folder. To help ensure that, we suggest that you set the support folder to be read-only. You can do this by right-clicking on it in the package explorer, choosing Properties from the menu, choosing Resource from the list on the left of the pop-up Properties window, unchecking the Permissions check-box for Owner-Write, and clicking the OK button. A dialog box will show with the title “Confirm recursive changes”, and you should click on the “Yes” button.

test The test folder where all of the public unit tests are available.

JUnit 4 A library that is used to run the test programs.

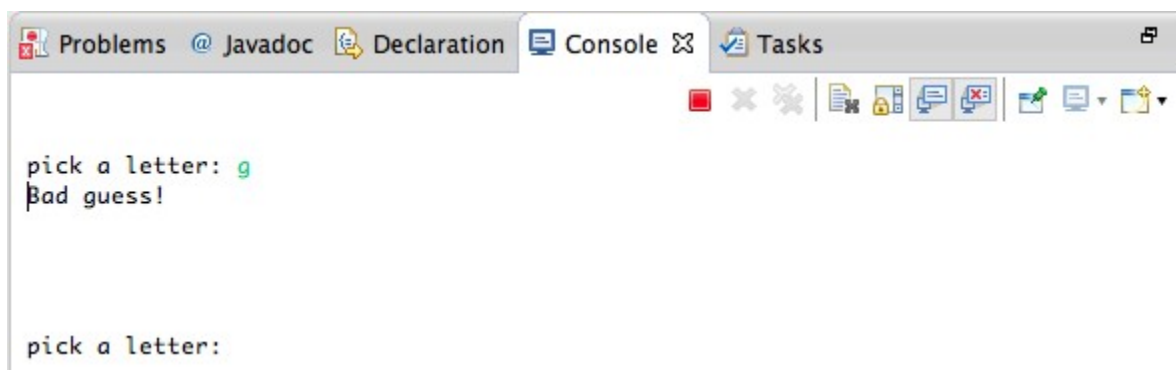
JRE System Library This is what allows Java to run; it is the location of the Java System Libraries.

If you are missing any of the above or if errors are present in the project (other than in JUnit tests), seek help immediately so you can get started on the project right away.

Starter Code and Tests

First, explore the code as it is. In the `src/hangman` directory you will find `ArrayGame.java`. This file is just a driver, and contains only a `main()` method. It picks a word and allocates instances of `ConsoleGameIO`, `ArrayGameModel`, and `GameController`. (This driver is for the array-based version, so it uses `ArrayGameModel`.) These classes (except `ArrayGameModel`) are defined in `support/hangman`, and *you must not modify them*. The `GameController` class actually runs the game, getting input from and sending output to the `GameIO` instance. The state of the game is maintained in the `GameModel` instance. The `GameModel` interface is also defined in `support/hangman` and *must also not be modified*. Read `GameModel` carefully to see what your implementations of it must do. `ArrayGameModel` is an empty implementation of `GameModel`.

The starter code you receive is *not a working implementation*. However, you can still run `ArrayGame`. 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:



```
pick a letter: g
Bad guess!

pick a letter:
```

Hangman Game

The rules for Hangman are simple and you can read the details online, for example, at http://en.wikipedia.org/wiki/Hangman_%28game%29. 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 case-sensitive word `heLLo` then she would write: `_ _ _ _ _`. An underscore is used for each letter that has not been guessed yet.

Player B will then proceed to guess letters one at a time 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, if 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, a] _ _ L L _`, where the `|-----|` represents the start of the hangman platform. The `[L, a]` 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. 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 `ConsoleGameIO.java`. Note that, as demonstrated above, our version of hangman also considers the difference between uppercase and lowercase letters. That is, the letter ‘A’ is not the same as ‘a’. Make sure your implementation takes this distinction into consideration. Finally, our version of hangman assumes that the set of possible letters are drawn from the standard English alphabet — not other characters that could possibly be entered (such as numbers, symbols, or characters from other languages such as `é`). Your implementation must accept and correctly handle all fifty-two of these characters, and reject all other characters.

Array Implementation

First, you will complete the hangman game by providing an array-based implementation of `GameModel` in the `ArrayGameModel.java` file. We have provided a minimal start to your implementation, but the rest is up to you! The `ArrayGameModel` class implements the `GameModel` interface. A “game model” 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 `GameModel` interface to gain a better understanding of each method you need to implement. Here is a brief overview of the methods that are required in your implementation (more details are provided in the comments in `GameModel.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 is in a winning state.
- **boolean** `inLosingState();`
This method returns true if the game is in a losing state.
- **int** `getState();`
This method returns the current hung state (in the interval 0–10).
- **String** `toString();`
This method returns a string representation of the game. 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]`
- **String** `getWord();`
This method returns the word that the player is trying to guess.

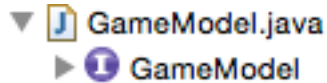
You will notice that there are 10 TODO comments in the provided `ArrayGameModel.java` file. You must implement each of these TODOs in order to satisfy the public JUnit tests that we provide. Your implementation must use Java arrays internally to represent the game board and the previous guesses. We have 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:

```
1 public ArrayGameModel(String guessWord) {  
2     // TODO (1)  
3     state = STARTING_STATE;  
4 }
```

Remember, you must not use any `Collection` abstractions (such as `ArrayList`) provided by the Java Platform API.

Linked List Implementation

In this part, 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 `GameModel` interface. To do this in Eclipse you select the arrow to the right side of the `GameModel.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 **New Java Class** wizard. Modify the **Source folder**: to create the file in the `src` directory instead of the `support` directory. Give your new class the name `LinkedListGameModel` (do not use a different name, or your program will not pass the autograder). This will create a new Java file in the **Package Explorer** called `LinkedListGameModel.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 argument as the `ArrayGameModel` class (a `String` representing the word to be guessed).

In order to implement the `LinkedListGameModel` interface you will need to provide a class to implement a linked list of characters — in the same spirit as the `LLStringNode` class covered in detail in the book. You must name this new class `LLCharacterNode`. It must have a constructor that takes a single `char` argument, a getter named `getInfo` that returns the stored `char`, and a pair of getters/setters named `getLink` and `setLink` that allow creation and traversal of the list. Use the `LLCharacterNode` in your implementation of the `LinkedListGameModel` class.

The `LinkedListGameModel` class implements the `GameModel` interface. A “game model” 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 `GameModel` interface file to gain a better understanding of each method you need to implement.

You will also need to create a `LinkedListGame` class similar to the `ArrayGame` class to test the execution of your implementation.

Remember, you must not use any `Collection` abstractions (such as `ArrayList`) provided by the Java Platform API.

Testing

We provide three public JUnit test classes you should use to test your implementation:

- `ArrayGameModelPublicTest.java`
- `LinkedListGameModelPublicTest.java`
- `LLCharacterNodePublicTest.java`

You should run these tests to test your implementation of each classes. You should note that the `LinkedListGameModelPublicTest` and `LLCharacterNodePublicTest` classes will contain errors until you have created the required classes for the linked list implementation. 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 and Submit

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, click on the `hangman-student` project in the package explorer. Then 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. Be sure that the project is named **hangman-student**. Save the exported file with the `zip` extension (any name is fine). Log into Moodle and submit the exported zip file.

Sample 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 winner!
Number of guesses: 12