Hangman

Thursday, February 14, 7:30 – 9:00PM
Andrew Tierno
Overview

► Review Lecture Material
  ► Characters
  ► Strings

► Assignment Overview
  ► Milestones/breakdown of tasks
  ► Some useful upcoming topics
  ► General suggestions and reminders

► Q&A
Lecture Review
Characters

```java
char ch = 'a';
// need to store return value
ch = Character.toUpperCase(ch);
// converting a char to a string
String str = "" + ch;
```
### Useful methods in the Character Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static boolean isDigit(char ch)</code></td>
<td>Determines if the specified character is a digit.</td>
</tr>
<tr>
<td><code>static boolean isLetter(char ch)</code></td>
<td>Determines if the specified character is a letter.</td>
</tr>
<tr>
<td><code>static boolean isLetterOrDigit(char ch)</code></td>
<td>Determines if the specified character is a letter or a digit.</td>
</tr>
<tr>
<td><code>static boolean isLowerCase(char ch)</code></td>
<td>Determines if the specified character is a lowercase letter.</td>
</tr>
<tr>
<td><code>static boolean isUpperCase(char ch)</code></td>
<td>Determines if the specified character is an uppercase letter.</td>
</tr>
<tr>
<td><code>static boolean isWhitespace(char ch)</code></td>
<td>Determines if the specified character is whitespace (spaces and tabs).</td>
</tr>
<tr>
<td><code>static char toLowerCase(char ch)</code></td>
<td>Converts <code>ch</code> to its lowercase equivalent, if any. If not, <code>ch</code> is returned unchanged.</td>
</tr>
<tr>
<td><code>static char toUpperCase(char ch)</code></td>
<td>Converts <code>ch</code> to its uppercase equivalent, if any. If not, <code>ch</code> is returned unchanged.</td>
</tr>
</tbody>
</table>
Comparing Characters

- Write a program that...
  - ...prompts the user for 2 words
  - ...prints out “The first letters match!” if the first letters of the two words are the same and “The first letters differ” if the first letters are not the same
- Case-insensitive (so “CS106A” and “cs106a” should match)
Comparing Characters - Solution

```java
String first = readLine("Enter a word: ");
String second = readLine("Enter a word: ");
```
Comparing Characters - Solution

```java
String first = readLine("Enter a word: ");
String second = readLine("Enter a word: ");
if (Character.toLowerCase(first.charAt(0)) ==
    Character.toLowerCase(second.charAt(0))) {
    println("The first letters match!");
} else {
    println("The first letters differ.");
}
```
String first = readLine("Enter a word: ");
String second = readLine("Enter a word: ");
if (Character.toLowerCase(first.charAt(0)) == Character.toLowerCase(second.charAt(0))) {
    println("The first letters match!");
} else {
    println("The first letters differ.");
}

What if the user enters an empty string?
Comparing Characters - Solution

```java
String first = readLine("Enter a word: ");
String second = readLine("Enter a word: ");
if (first.length() == 0 || second.length() == 0) {
    println("Empty string");
} else if (Character.toLowerCase(first.charAt(0)) ==
    Character.toLowerCase(second.charAt(0))) {
    println("The first letters match!");
} else {
    println("The first letters differ.");
}
```
Strings

String \( s = \text{“Hi mom”}; \) // ordered characters

\[
\begin{array}{cccccc}
\text{0} & \text{1} & \text{2} & \text{3} & \text{4} & \text{5} \\
H & i & m & o & m & \text{(length 6)}
\end{array}
\]

// need to store value of \( s.toUpperCase() \)
\( s = s.toUpperCase(); \)
println(s); // prints “HI MOM”
### Useful methods in the String Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int length()</code></td>
<td>Returns the length of the string</td>
</tr>
<tr>
<td><code>char charAt(int index)</code></td>
<td>Returns the character at the specified index. Note: Strings indexed starting at 0.</td>
</tr>
<tr>
<td><code>String substring(int p1, int p2)</code></td>
<td>Returns the substring beginning at <code>p1</code> and extending up to but not including <code>p2</code></td>
</tr>
<tr>
<td><code>String substring(int p1)</code></td>
<td>Returns substring beginning at <code>p1</code> and extending through end of string.</td>
</tr>
<tr>
<td><code>boolean equals(String s2)</code></td>
<td>Returns true if string <code>s2</code> is equal to the receiver string. This is case sensitive.</td>
</tr>
<tr>
<td><code>int compareTo(String s2)</code></td>
<td>Returns integer whose sign indicates how strings compare in lexicographic order</td>
</tr>
<tr>
<td><code>int indexOf(char ch) or int indexOf(String s)</code></td>
<td>Returns index of first occurrence of the character or the string, or -1 if not found</td>
</tr>
<tr>
<td><code>String toLowerCase() or String toUpperCase()</code></td>
<td>Returns a lowercase or uppercase version of the receiver string</td>
</tr>
</tbody>
</table>
Looping over a String

Canonical “loop over the characters in a string” loop

```java
for (int i = 0; i < string.length(); i++) {
    char ch = string.charAt(i);
    /* ... process ch ... */
}
```

*string.length() returns length not final index*
Comparing Strings

```java
String s1 = "racecar";
String s2 = reverseString(s1);
```

// How do we check equality?
Comparing Strings

String s1 = "racecar";
String s2 = reverseString(s1);

// How do we check equality?
if (s1 == s2) {
    ...
} OR
if (s2.equals(s1)) {
    ...
}
Comparing Strings

```java
String s1 = "racecar";
String s2 = reverseString(s1);

// How do we check equality?
if (s1 == s2) {
    ...
}
```

DON’T DO THIS
String References vs. Literals

```java
String s1 = "racecar";
String s2 = reverseString(s1);

// How do we check equality?
if (s1 == s2) {
    if (s2.equals(s1)) {
        ... // Compares reference address
    }
}
```

<table>
<thead>
<tr>
<th>Stack</th>
<th>reference address</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>memory.com/42</td>
</tr>
<tr>
<td>s2</td>
<td>memory.com/79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heap</th>
<th>string literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory.com/49</td>
<td>&quot;racecar&quot;</td>
</tr>
<tr>
<td>memory.com/79</td>
<td>&quot;racecar&quot;</td>
</tr>
</tbody>
</table>
String References vs. Literals

```java
String school = "Harvard";
println(school + " is a top university..");
fix(school);
println(school + " is not as good as Stanford!");
```

// What is printed?
```java
private void fix(String str) {
    str = "Stanford of the East";
}
```
String References vs. Literals

```java
String school = "Harvard";
println(school + " is a top university.."); // Harvard is a top university..
fix(school);
println(school + " is not as good as Stanford!"); // Harvard is not as good as Stanford!

// Because strings are immutable they behave like primitives in methods!
private void fix(String str) {
    str = "Stanford of the East";
}
```

<table>
<thead>
<tr>
<th>Stack (run)</th>
<th>reference address</th>
</tr>
</thead>
<tbody>
<tr>
<td>school</td>
<td>memory.com/36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heap</th>
<th>string literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory.com/36</td>
<td>&quot;Harvard&quot;</td>
</tr>
<tr>
<td>memory.com/99</td>
<td>&quot;Stanford of the East&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack (fix)</th>
<th>reference address</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>memory.com/99</td>
</tr>
</tbody>
</table>
String References vs. Literals

```java
String school = "Harvard";
println(school + " is a top university..") // Harvard is a top university..

school = fix(school);
println(school + " is not as good as Stanford!") // Stanford of the East is not as good as Stanford!

// Because strings are immutable they behave like primitives in methods!
```

```java
private String fix(String str) {
    str = "Stanford of the East";
    return str;
}
```
Searching Strings

- You can use the `indexOf` method to search a string:

  ```java
  int index = str.indexOf(pattern);
  ```

- `indexOf` returns the start index of the first occurrence of the pattern if the pattern exists in the string. Otherwise, it returns -1.

- Overloaded so that `pattern` can be either a `String` or a `char`!

```java
int index = "hello".indexOf("el"); // 1
int index = "hello".indexOf('e'); // 1 (overloaded to work with chars!)
int notFound = "cs106a".indexOf("b"); // -1
```
Building Strings

- 1. Use substrings – smaller pieces of strings
   OR
- 2. Make new string and build over time
1. Substrings

- To get all of the characters in the range \([\text{start}, \text{stop})\), use

  ```java
  str.substring(start, stop);
  ```

- To get all of the characters from some specified point forward, use

  ```java
  str.substring(start);
  ```

```java
str.substring(0, 2);
str.substring(6);
```
Building a New String

- Start with an empty string and build up a new string
- Iterate through the old string
- Use Character methods at each position to decide what to concatenate to the new string
- See this week’s section handout for examples
String Summary: Strings are...

- objects that have methods (`length()`, `charAt()`, `equals()`, `indexOf()`...)
- zero-indexed list of `char`s
- immutable!
  - but you can concatenate them, get substrings from them, search them, compare them
  - ...using `methods` and the canonical `new string + reassignment to old` pattern
Assignment 4 - Hangman!
Assignment 4 - Hangman

- Due Wednesday, February 20th
- String processing
- Pair assignment (optional)
- We suggest approaching this assignment in stages
Part I: Playing a Console-Based Game
Part I: Console Game

- Choose a random word (using HangmanLexicon)
- Display a “hint” (initially “- - - - - - - - - ”)
- Get guesses from the user
- Figure out if a guess is correct (letter in the secret word) or incorrect (not in secret word)
- Update hint
- Keep track of the number of guesses the user has left
- Determine when the game has ended (no guesses left or they guessed the word)
- ...Repeat
Game Flow

String secretWord

String wordState

char guess

String newWordState
Hangman Lexicon

- Stub class you can use until you are able to implement part 3
- Beginning of run: create a new `HangmanLexicon` and store it in an instance variable
- If you extend the program to allow the user to play multiple games, create `HangmanLexicon` outside the loop that plays the game repeatedly
Part I: Console Game - Tips

- Keep track of the user’s partially-guessed word (dashes and letters)
- Your program should be case-insensitive (R and r should be the same guess)
  - Guessed letters string should be all upper-case, even when a guess is lower case
- You will have some fencepost issues – look at lecture slides for techniques to deal with this
- Watch out for edge case input! (single letter, empty string, etc)
Part I: Console Game - Error Checking

- You’ll need to prompt the user to enter guesses.
- The user may enter a letter in upper or lower case (hint: the secret words are all upper-case).
- If the user guesses anything other than a single letter, print out an error message and reprompt.
- If the user enters the same correct letter more than once, do nothing.
- If the user enters the same *incorrect* letter more than once, it’s incorrect again.
Part I: Console Game – Sample Output

Follow the screenshots to know what your output should look like!
Part II: Adding Graphics
Part II: Hangman Graphics
Drawing

- Add the canvas instance variable to the window using `init()`
  - Call graphics methods on the canvas object, since console programs don't know how to do graphics tasks! i.e.:
    ```javascript
    canvas.add(object, x, y);
    ```
  - The canvas will behave exactly like your `GraphicsPrograms`, besides the fact that you have to put `canvas.` at the start
- If the user inputs a wrong guess, remove one of the strings holding Karel's parachute
- If all the guesses are up, display flipped Karel :'("
Drawing the Guess

- Should update any time the user enters a valid guess (i.e. one character)
- If it is correct, update the partially guessed dashed word
- If it is incorrect, update the set of incorrect guesses

- Setting the font of a GLabel (e.g. label): label.setFont("MyCoolFont-42");
Drawing an Image

GImage img = new GImage("mycoolpicture.png");
img.setSize(400,300); //optional, if you want to resize
canvas.add(img, 20, 50); //Just like GRects!
Drawing the strings (strings, not Strings)

- Should initially draw as many strings as N_GUESSES constant that are equally spaced across the base of the parachute.
- Special order for removing the strings! Basically outside in.
- What’s the best way to represent these strings in your code?
Part III: Reading The Lexicon From a Data File
Part III: Random Word from File

```java
private String getRandomWord()
```

- Before starting this milestone, just use the provided “stub” implementation to get one of 10 random words.
- 1. Open the data file `HangmanLexicon.txt` using a Scanner (at start of program)
- 2. Read the lines from the file into an `ArrayList` (at start of program)
- 3. Reimplement `getRandomWord` so it uses this ArrayList as the source of the words.

There is also a `ShorterLexicon.txt` file you can use for testing/debugging.
Useful Topics
Scanner

Use a Scanner to read from a file

Yesterday, upon the stair,
I met a man who wasn’t there
He wasn’t there again today
I wish, I wish he’d go away...
- Hughes Mearns, "Antagonish"

Scanner input = new Scanner(new File("mydata.txt"));

// Yesterday, upon the stair
String line1 = input.nextLine();
Scanner input = new Scanner(new File("poem.txt"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    //more awesome code here
}
input.close();
Try/Catch

try {
    Scanner input = new Scanner(new File("poem.txt"));
    while (input.hasNextLine()) {
        String line = input.nextLine();
        //more awesome code here
    }
}

} catch (IOException ex) {
    println("I couldn’t open poem.txt :(“);
}
Why try/catch?

- Opening a file is “dangerous” (what if it doesn’t exist?)
- By using a try/catch, we’re promising Java that we have a plan if things go horrifically wrong
- Things often go horrifically wrong
ArrayLists

- Class representing an ordered, variable size list of data
- Homogeneous (each entry is of the same data type)
- Can store any object!
ArrayList Example

```java
ArrayList<String> ourFirstArrayList = new ArrayList<String>();

// Add elements to the right (the 'back')
ourFirstArrayList.add("hello");
ourFirstArrayList.add("world");

// Elements 0-indexed left ('front') to right ('back')
println(ourFirstArrayList.get(0) + " " + ourFirstArrayList.get(1));

// Will get an IndexOutOfBoundsException since size is 2, but we 0-index!
String tmp = ourFirstArrayList.get(ourFirstArrayList.size());
```
Helpful ArrayList Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(&lt;T&gt; element)</td>
<td>Adds a new element to the end of the ArrayList; the return value is always true.</td>
</tr>
<tr>
<td>void add(int index, &lt;T&gt; element)</td>
<td>Inserts a new element into the ArrayList before the position specified by index.</td>
</tr>
<tr>
<td>&lt;T&gt; remove(int index)</td>
<td>Removes the element at the specified position and returns that value.</td>
</tr>
<tr>
<td>boolean remove(&lt;T&gt; element)</td>
<td>Removes the first instance of element, if it appears; returns true if a match is found.</td>
</tr>
<tr>
<td>void clear()</td>
<td>Removes all elements from the ArrayList.</td>
</tr>
<tr>
<td>int size()</td>
<td>Returns the number of elements in the ArrayList.</td>
</tr>
<tr>
<td>&lt;T&gt; get(int index)</td>
<td>Returns the object at the specified index.</td>
</tr>
<tr>
<td>&lt;T&gt; set(int index, &lt;T&gt; value)</td>
<td>Sets the element at the specified index to the new value and returns the old value.</td>
</tr>
<tr>
<td>int indexOf(&lt;T&gt; value)</td>
<td>Returns the index of the first occurrence of the specified value, or -1 if it does not appear.</td>
</tr>
<tr>
<td>boolean contains(&lt;T&gt; value)</td>
<td>Returns true if the ArrayList contains the specified value.</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>Returns true if the ArrayList contains no elements.</td>
</tr>
</tbody>
</table>
Wrapping Up
Extensions

- Extensions are optional, and you will get a small amount of extra credit if you do them
  - Focus on the main program first, though – extensions won’t make up for a broken Hangman!
- If you do extensions, submit two different .java files for the assignment
  - The basic Hangman.java that meets all of the assignment requirements
  - HangmanExtra.java that has your extensions. In Eclipse, right click on Hangman.java, click Copy, then ctrl+v (paste). In the Name Conflict window that appears, write HangmanExtra and click OK, then make extension edits in the new file. Both files will submit together.
- In HangmanExtra.java, be sure to comment all of your extensions in the header comment so your SL knows what to look for.
- See the spec for ideas or come up with your own!
Final Tips

- Make sure your program compiles without any errors or warnings
- Follow the spec carefully and make sure your output matches the spec and expected output
- Make sure you properly handle all user input, including faulty/unexpected input
- Use instance variables only where absolutely necessary
- Don’t have a method that calls itself
- Go to the LaIR if you get stuck, and incorporate IG feedback!
questions?