Hangman YEAH Hours

Tuesday, May 8, 6:00 – 7:00PM
Andrew Tierno and Milan Mossé
(atierno@stanford.edu; mmosse19@stanford.edu)
Slides by Julia Daniel & Ben Barnett

Overview

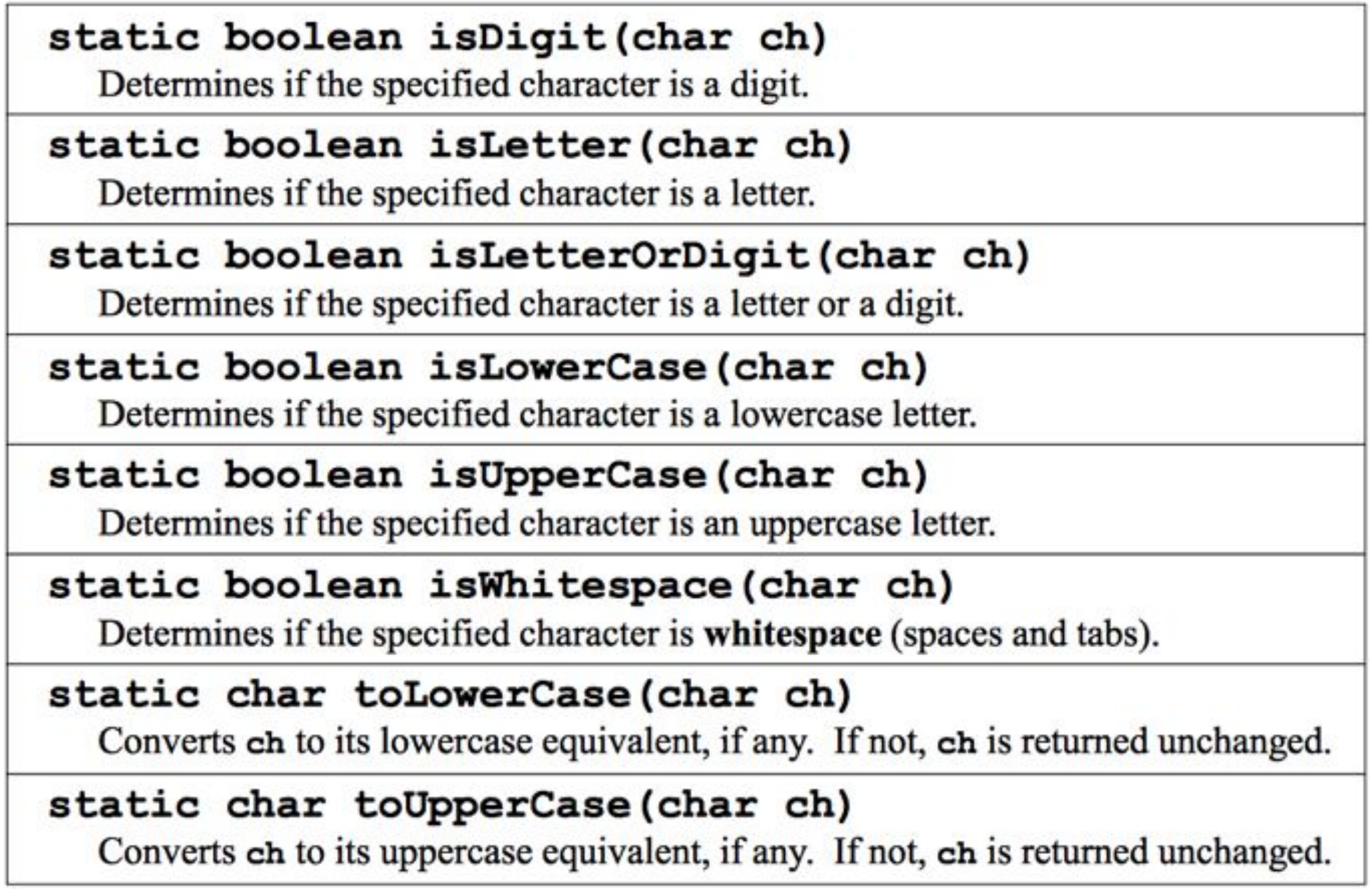
- Review Lecture Material
  - Characters
  - Strings
- Assignment Overview
  - Milestones/breakdown of tasks
  - General suggestions and reminders
- Q&A

Lecture Review

Characters

```
char ch = ’a’;
ch = Character.toUpperCase(ch);   // need to store return value
String str = “” + ch;             // converting a char to a string
```

Useful methods in the `Character` Class



`static boolean isDigit(char ch)` Determines if the specified character is a digit.
`static boolean isLetter(char ch)` Determines if the specified character is a letter.
`static boolean isLetterOrDigit(char ch)` Determines if the specified character is a letter or a digit.
`static boolean isLowerCase(char ch)` Determines if the specified character is a lowercase letter.
`static boolean isUpperCase(char ch)` Determines if the specified character is an uppercase letter.
`static boolean isWhitespace(char ch)` Determines if the specified character is **whitespace** (spaces and tabs).
`static char toLowerCase(char ch)` Converts **`ch`** to its lowercase equivalent, if any. If not, **`ch`** is returned unchanged.
`static char toUpperCase(char ch)` Converts **`ch`** to its uppercase equivalent, if any. If not, **`ch`** is returned unchanged.

Using portions of slides by Eric Roberts

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

```
String first = readLine(“Enter a word: “);
String second = readLine(“Enter a word: “);
```

Comparing Characters - Solution

```
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.");
}
```

Comparing Characters - Solution

```
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?

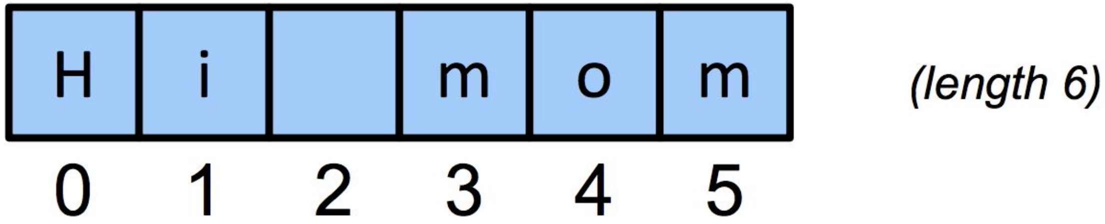
Comparing Characters - Solution

```
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 = “Hi mom”;   // ordered characters
```

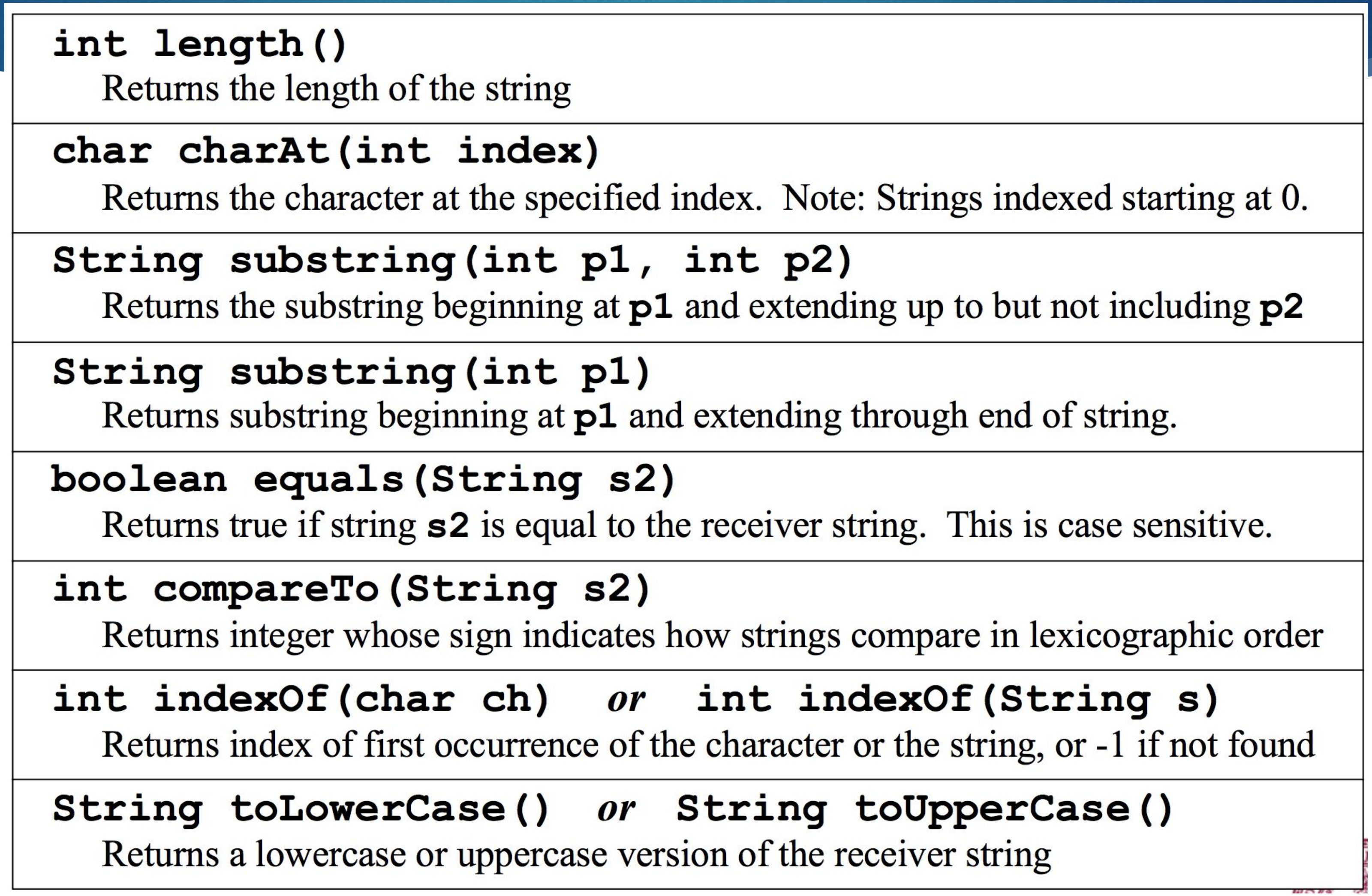


H	i		m	o	m
0	1	2	3	4	5

(length 6)

```
// need to store value of s.toUpperCase()
s = s.toUpperCase();
println(s);            // prints “HI MOM”
```

Useful methods in the `String` Class



`int length()`

Returns the length of the string

`char charAt(int index)`

Returns the character at the specified index. Note: Strings indexed starting at 0.

`String substring(int p1, int p2)`

Returns the substring beginning at **`p1`** and extending up to but not including **`p2`**

`String substring(int p1)`

Returns substring beginning at **`p1`** and extending through end of string.

`boolean equals(String s2)`

Returns true if string **`s2`** is equal to the receiver string. This is case sensitive.

`int compareTo(String s2)`

Returns integer whose sign indicates how strings compare in lexicographic order

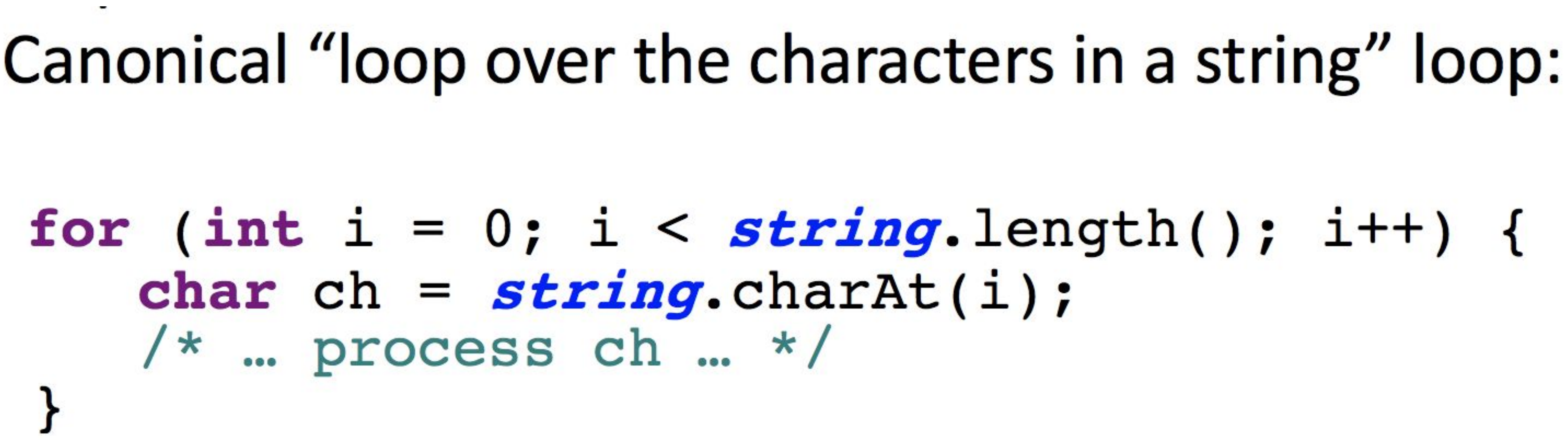
`int indexOf(char ch)` *or* **`int indexOf(String s)`**

Returns index of first occurrence of the character or the string, or -1 if not found

`String toLowerCase()` *or* **`String toUpperCase()`**

Returns a lowercase or uppercase version of the receiver string

Looping over a String



Canonical “loop over the characters in a string” loop:

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

Comparing Strings

```
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.equals(s2)) {
    …
}
```

OR

```
if (s2.equals(s1)) {
    …
}
```

Comparing Strings

```
String s1 = “racecar”;
String s2 = reverseString(s1);


// How do we check equality?
```

DON’T DO THIS

```
if (s1 == s2) {
    …
}
```

Searching Strings

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

```
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, if returns -1

```
int index = “hello”.indexOf(“el”);          // 1
int notFound = “cs106a”.indexOf(“b”);       // -1
```

Building Strings

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

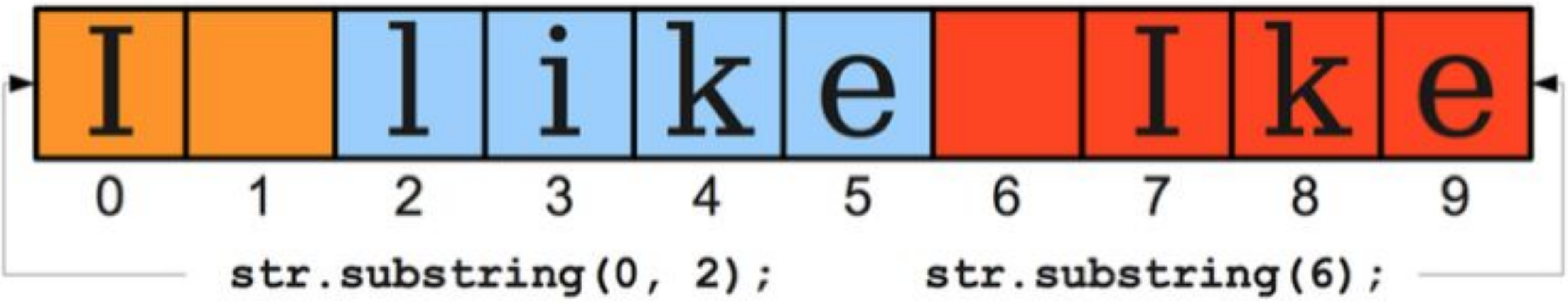
1. Substrings

- To get all of the characters in the range [start, stop), use

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

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

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



```
str.substring(0, 2);
```

```
str.substring(6);
```

2. 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…

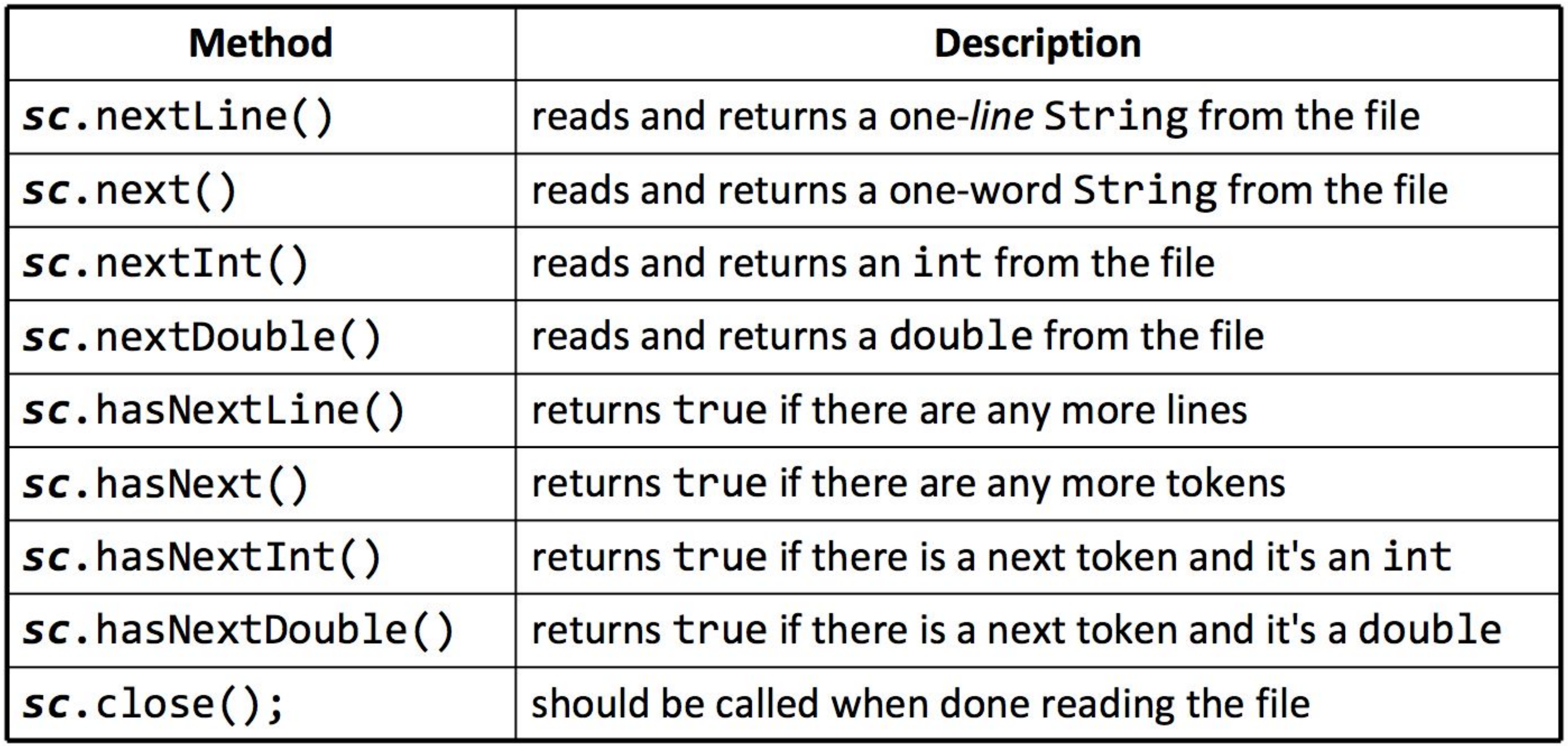
- objects that have methods (`length(), charAt(), equals(), indexOf()...`)
- zero-indexed lists of **`char`**s
- **immutable**!
  - but you can concatenate them, get substrings from them, search them, compare them…
  - …using **methods** and the canonical **new string + reassignment to old variable**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

Scanners

- Use a `Scanner` to read from a file
- Remember to use a try/catch
- Remember to close your scanner when you’re done! (like housekeeping)

```
try {
    Scanner input = new Scanner(new File(“filename.txt”));
    while (input.hasNextLine()) {
        String line = input.nextLine();
        // do something with line
    }
    input.close();
} catch (IOException e) {
    // put some descriptive error message here
}
```

Scanners



Method	Description
`sc.nextLine()`	reads and returns a one-*line* `String` from the file
`sc.next()`	reads and returns a one-word `String` from the file
`sc.nextInt()`	reads and returns an `int` from the file
`sc.nextDouble()`	reads and returns a `double` from the file
`sc.hasNextLine()`	returns `true` if there are any more lines
`sc.hasNext()`	returns `true` if there are any more tokens
`sc.hasNextInt()`	returns `true` if there is a next token and it's an `int`
`sc.hasNextDouble()`	returns `true` if there is a next token and it's a `double`
`sc.close();`	should be called when done reading the file

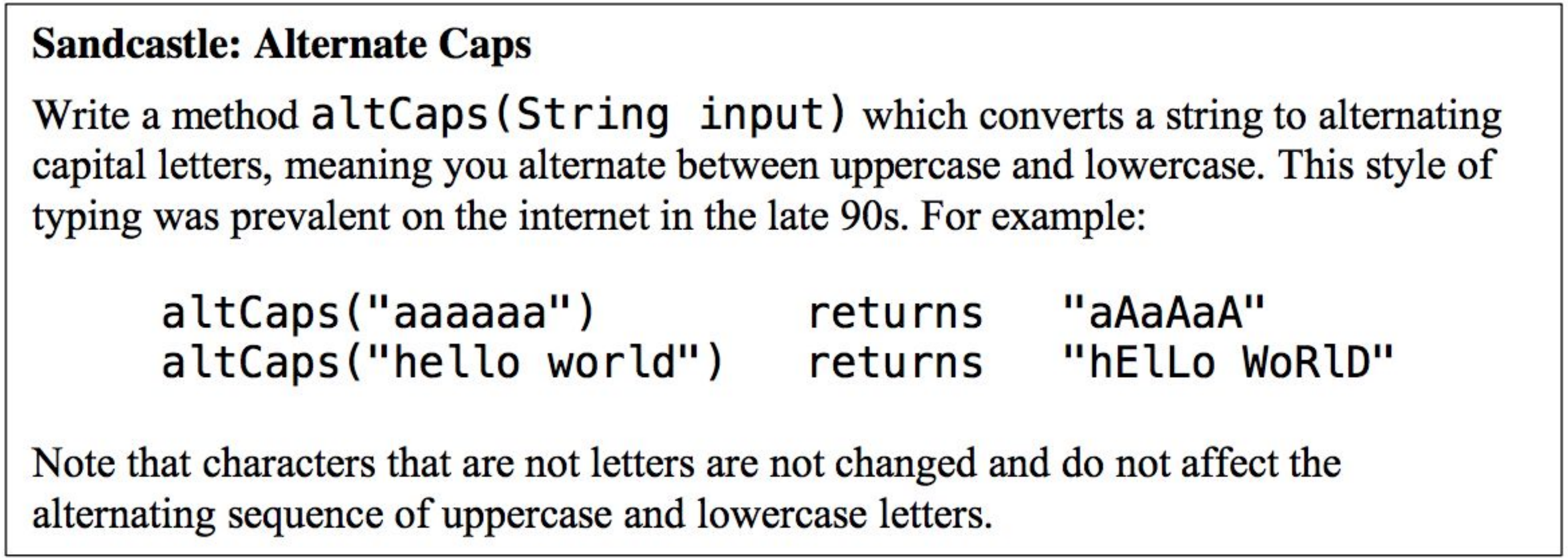
Assignment 4

Assignment 4 - Hangman

- Due Monday, May 14 at 11:00am
- String processing
- Pair assignment (optional)
  - Notes on pair programing (read these)
- We suggest approaching this assignment in stages

Task 0: Sandcastle

- Start with this to warm up!

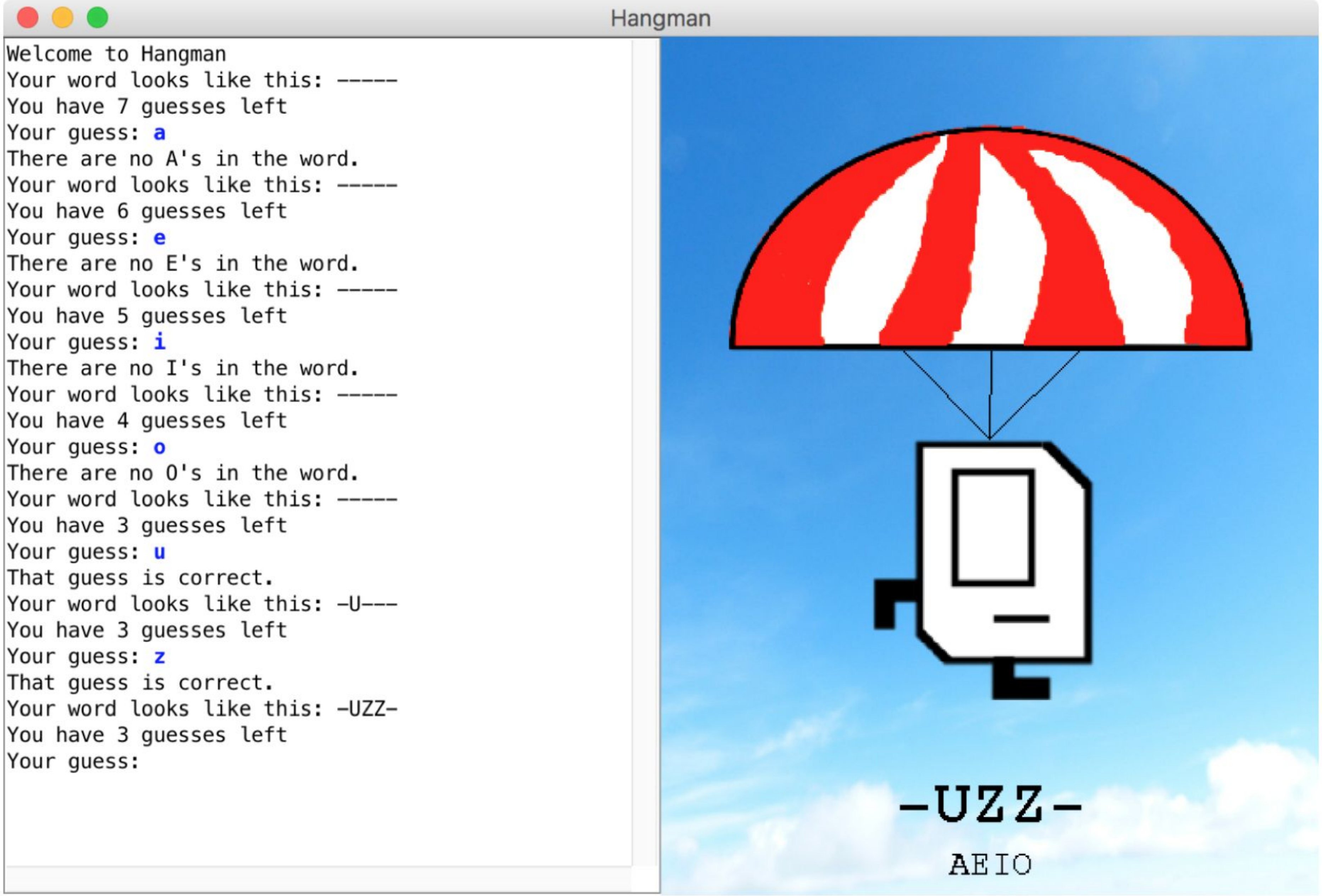


Sandcastle: Alternate Caps

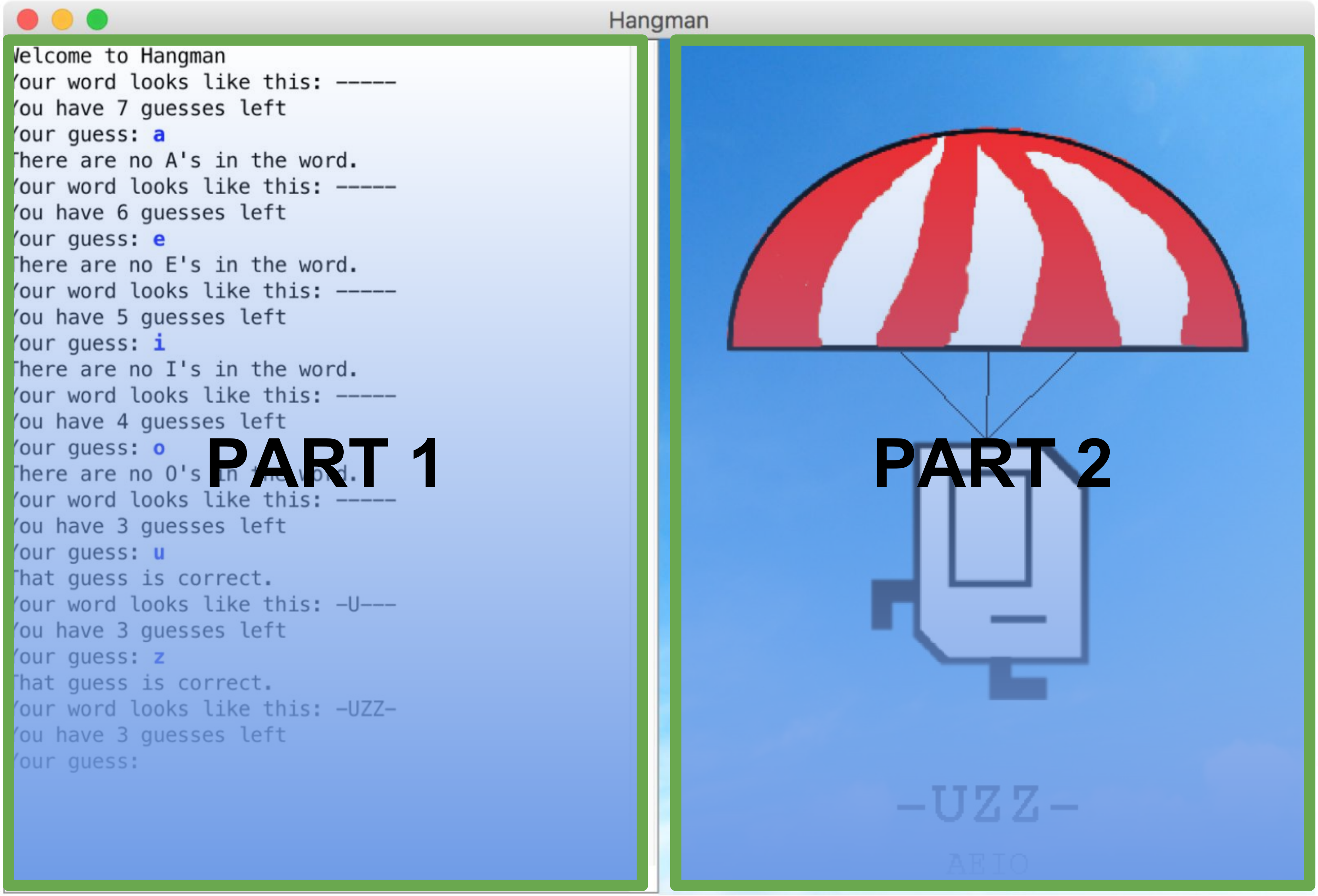
Write a method `altCaps(String input)` which converts a string to alternating capital letters, meaning you alternate between uppercase and lowercase. This style of typing was prevalent on the internet in the late 90s. For example:

```
altCaps("aaaaaa")        returns   "aAaAaA"
altCaps("hello world")   returns   "hElLo WoRlD"
```

Note that characters that are not letters are not changed and do not affect the alternating sequence of uppercase and lowercase letters.



```
Welcome to Hangman
Your word looks like this: -----
You have 7 guesses left
Your guess: a
There are no A's in the word.
Your word looks like this: -----
You have 6 guesses left
Your guess: e
There are no E's in the word.
Your word looks like this: -----
You have 5 guesses left
Your guess: i
There are no I's in the word.
Your word looks like this: -----
You have 4 guesses left
Your guess: o
There are no O's in the word.
Your word looks like this: -----
You have 3 guesses left
Your guess: u
That guess is correct.
Your word looks like this: -U---
You have 3 guesses left
Your guess: z
That guess is correct.
Your word looks like this: -UZZ-
You have 3 guesses left
Your guess:
```



```
Welcome to Hangman
Your word looks like this: -----
You have 7 guesses left
Your guess: a
There are no A's in the word.
Your word looks like this: -----
You have 6 guesses left
Your guess: e
There are no E's in the word.
Your word looks like this: -----
You have 5 guesses left
Your guess: i
There are no I's in the word.
Your word looks like this: -----
You have 4 guesses left
Your guess: o
There are no O's in the word.
Your word looks like this: -----
You have 3 guesses left
Your guess: u
That guess is correct.
Your word looks like this: -U---
You have 3 guesses left
Your guess: z
That guess is correct.
Your word looks like this: -UZZ-
You have 3 guesses left
Your guess:
```

PART 1

PART 2

PART 3

Task 1: Console Game

- 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	PROGRAMMER
String wordState	- - - - - - - - - -
char guess	r

String newWordState	-R--R----R

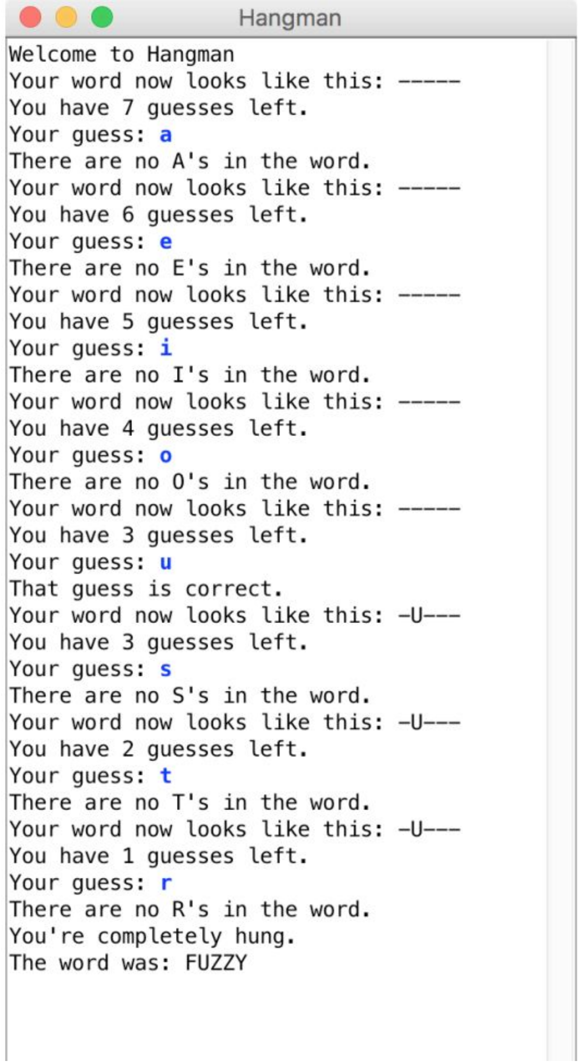
Task 1: Console Game - Tips

- 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

Task 1: 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.

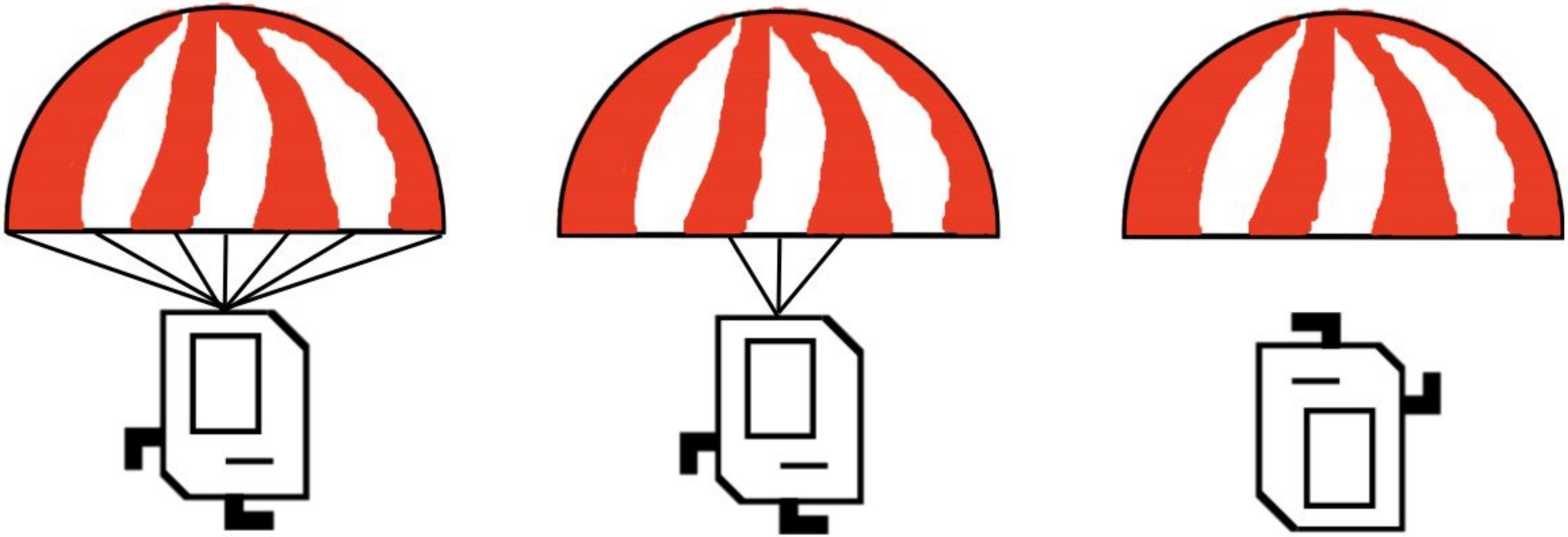
Task 1: Console Game – Sample Output



```
Welcome to Hangman
Your word now looks like this: -----
You have 7 guesses left.
Your guess: a
There are no A's in the word.
Your word now looks like this: -----
You have 6 guesses left.
Your guess: e
There are no E's in the word.
Your word now looks like this: -----
You have 5 guesses left.
Your guess: i
There are no I's in the word.
Your word now looks like this: -----
You have 4 guesses left.
Your guess: o
There are no O's in the word.
Your word now looks like this: -----
You have 3 guesses left.
Your guess: u
That guess is correct.
Your word now looks like this: -U---
You have 3 guesses left.
Your guess: s
There are no S's in the word.
Your word now looks like this: -U---
You have 2 guesses left.
Your guess: t
There are no T's in the word.
Your word now looks like this: -U---
You have 1 guesses left.
Your guess: r
There are no R's in the word.
You're completely hung.
The word was: FUZZY
```

Follow the screenshots to know what your output should look like!

Task 2: Hangman Graphics

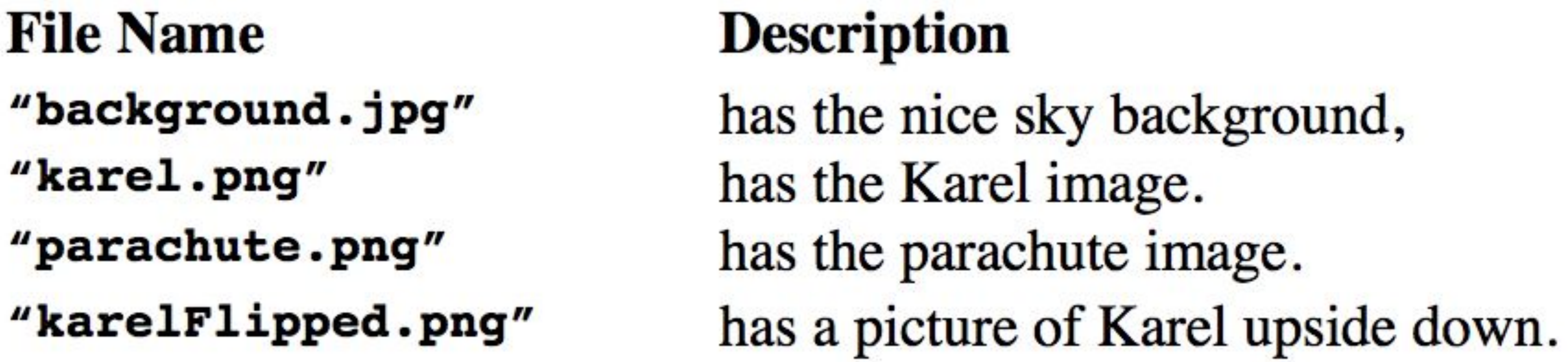


Task 2: Hangman Graphics

- Add the canvas instance variable to the window using `init()`
  - This is a console (not graphics!) program—call graphics methods on the canvas object. i.e.: `canvas.add(`***`object`***`, x, y);`
- Add the main objects (background, Karel, and parachute) to the canvas
- Add, and remove one-by-one, the parachute cords
  - Use the exact order specified in the handout: alternating from outside in, start on right
- Add current word state and incorrectly guessed letters to the canvas
- Flip Karel if user loses game

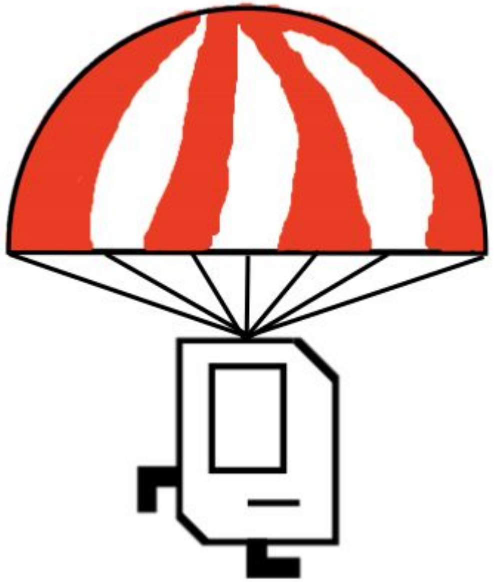
Task 2: Add main graphics

- All images are in files included in the project
- Sizes and y-locations are constants
- Make sure objects are centered!

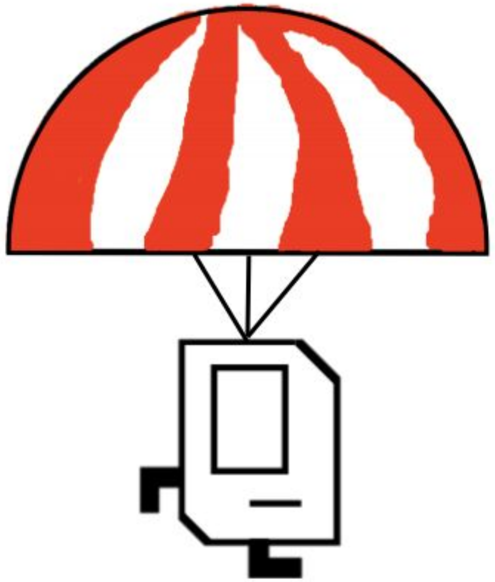


File Name	Description
`“background.jpg”`	has the nice sky background,
`“karel.png”`	has the Karel image.
`“parachute.png”`	has the parachute image.
`“karelFlipped.png”`	has a picture of Karel upside down.

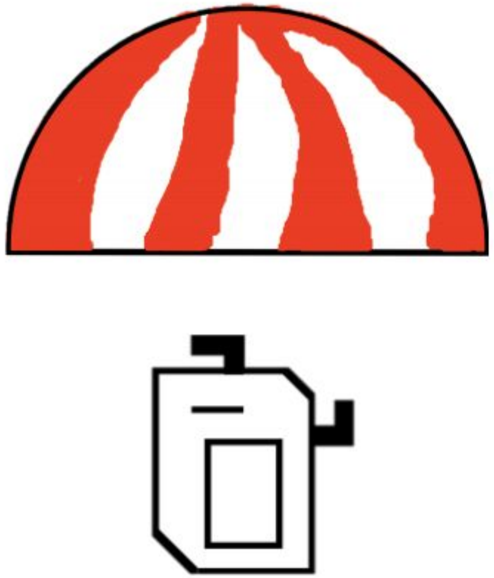
Task 2: Add & remove parachute cords



7 guesses left



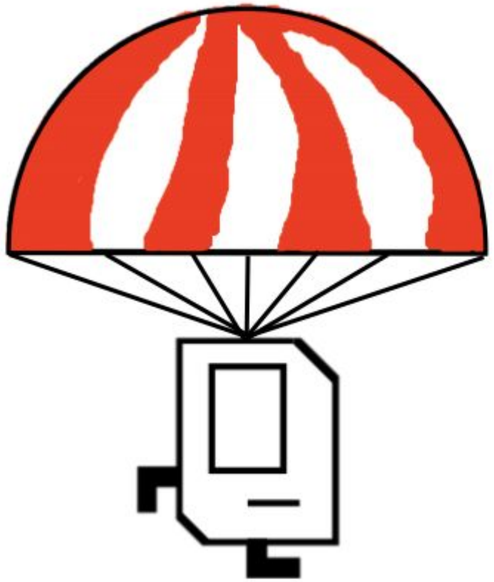
3 guesses left



game over

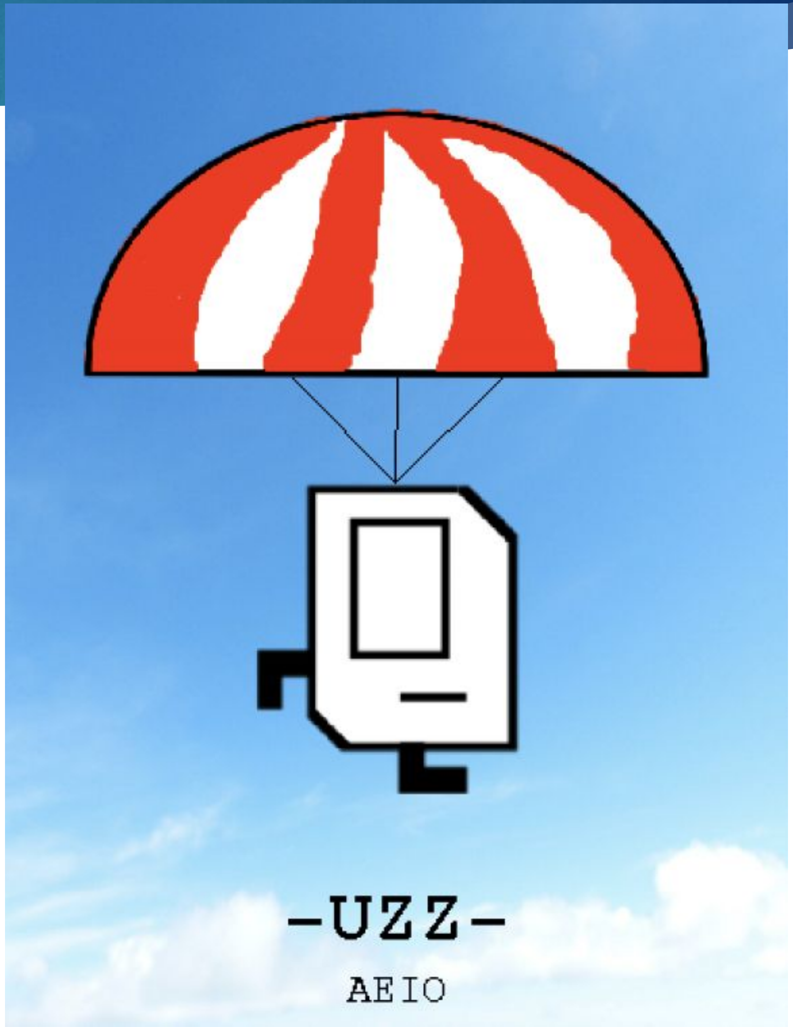
Task 2: Add & remove parachute cords

- N_GUESSES (default 7) lines
- Tops of lines are evenly spaced along the bottom of the parachute
- Bottoms of lines are all at centerpoint of top edge of Karel
- Removed one-by-one from outside to center, alternating starting on the right
- There are multiple reasonable ways to do this, and this part will be easier to think about once you've added main graphics



Task 2: Add labels for game state

- Use GLabels to represent current state of guessed word and incorrectly guessed letters
- Update these when the game state changes due to user input
- Center horizontally
- Size, y-location, font are constants



Task 2: Ending graphics

- Use karelFlipped.png if Karel runs out of cords
- Plenty of possibilities for extensions here!

Task 3: Random Word from File

```
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.

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
- Continuous decomposition
- Make sure you properly handle all user input, including faulty/unexpected input
  - Try to break your program :)
- Use instance variables only where absolutely necessary
- Don’t have a method that calls itself
- Once you’ve put some time into understanding the relevant bug or concept, come to the LaIR with any remaining questions
- Incorporate IG feedback!

fin.