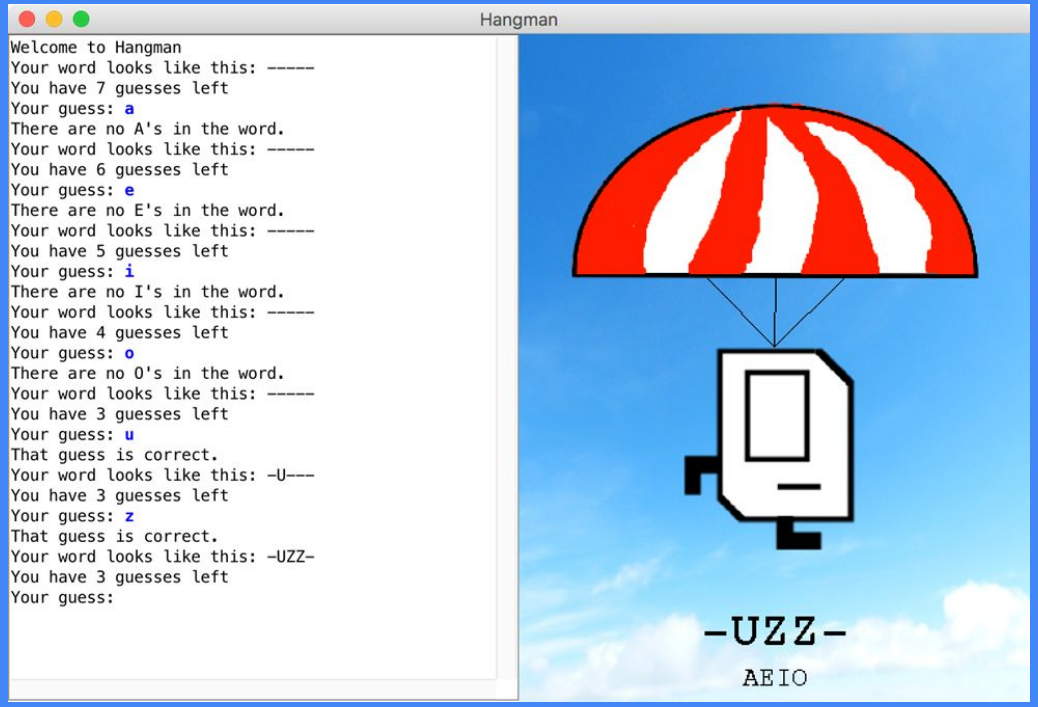


YEAH Hours Hangman

09 February 2017, George Horrell and Armin Namavari

YEAH Hours Schedule

Hangman	**Here, now!**
Yahtzee	Tuesday February 21st, 7:30 - 9:00 P.M.
Namesurfer	Wednesday March 1st, 7:30 - 9:00 P.M.
FacePamphlet	Thursday March 9th, 7:30 - 9:00 P.M.

Today’s YEAH Hours Plan

1. REVIEW STRINGS

a. How do we store text?
b. Characters v.s. Strings
c. Useful String methods
d. Useful Character methods
e. Solving String problems

2. HANGMAN

a. Game flow
b. Console based game
c. Graphics
d. File Reading
e. Debugging
f. Common pitfalls

How do we store text?

Text is stored using the variable type **String**.
A **String** is a sequence of **characters**.

```
public void run() {
    String text = “Hello World!”;
    println(text);
}
```

Characters

```
char ch = ‘a’;
ch = Character.toUpperCase(ch);
String str = “” + ch; // char -> String
println(str); // A
```

Can’t just write (for line 2):

```
Character.toUpperCase(ch);
```

Useful Character methods

`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.

EXAMPLE

Comparing chars

Let’s write a program that:

- prompts the user for 2 words
- print out “they match” if the first letters of the two words are the same

Solution: Comparing chars

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

if(Character.toLowerCase(first.charAt(0)) ==
  Character.toLowerCase(second.charAt(0))) {
    println("The first letters match!");
} else {
    println("The first letters differ.");
}
```

Edge case: no word entered!

EXAMPLE

Comparing chars (irrespective of case)

Chars are case sensitive, so we will not get a match from “A” and “a”.

What if we wanted to match regardless of case?

Comparing chars (again)

```
String first = readLine("Enter a word: ").toLowerCase();
String second = readLine("Enter another: ").toLowerCase();

If(first.charAt(0) == second.charAt(0)) {
    println("The first letters match!");
} else {
    println("The first letters differ.");
}
```

Same edge case: the empty String!

Strings

```
String s = “Hello!”;
s = s.toUpperCase();
println(s); // prints HELLO!

// Can’t just write (for line 2)
s.toUpperCase();
```

H 0	e 1	l 2	l 3	o 4	! 5

Immutability

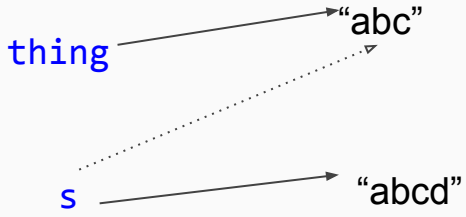
```
public void changeIt(String s) { // deceptive method name
    s += "a";
}

public void run() {
    String thing = "abc";
    println("Before: " + thing);
    changeIt(thing);
    println("After: " + thing);
}
```

Any guesses as to what will be printed out?

Immutability Illustrated

```
public void changeIt(String s) { // deceptive method name
       s += "d"; // s = s + “d”
}
 public void run() {
       String thing = "abc";
       println("Before: " + thing);
       changeIt(thing);
       println("After: " + thing);
 }
```



s is scoped to changeIt!

Output

```
Before: abc
After: abc
```

Strings never change! (but references variables hold can!)

compare to rect.setColor(...)

Useful String methods

`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

Testing equality with Strings

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

// how do we test if they are equal?

if(s1.equals(s2)) {
     . . .
}
-----OR------
if(s2.equals(s1)) {
     . . .
}
```

Don’t do this!

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

// This is wrong!

if(s1 == s2) {
     . . .
}
```

Searching Strings

- Search using the indexOf method: String.**indexOf**(pattern)
- **indexOf** returns the start index of the first occurrence of pattern, if one exists.
- Otherwise, it returns **-1**.

```
int index = “hello”.indexOf(“el”); // 1
```

Building new Strings

1) Use subStrings (or other string methods) – smaller pieces of Strings

AND/OR

2) Make new String, build up over time

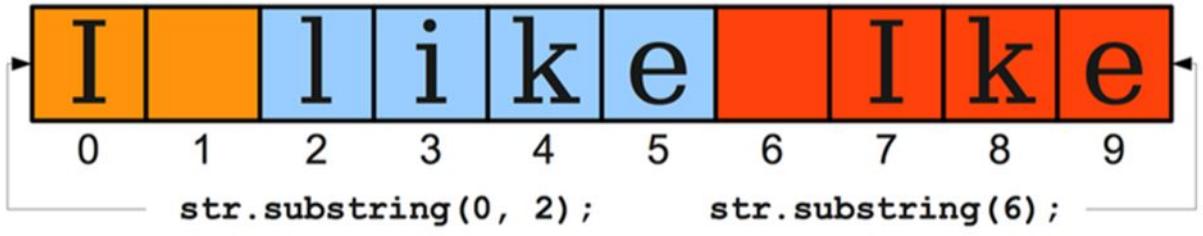
Getting substrings

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

```
string.substring(start, stop)
```

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

```
string.substring(start)
```



EXAMPLE

getSUNet

Let’s write a method that takes a Stanford email address, and returns the SUNet.

Solution: getSUNet

```
private String getSUNet(String email) {
        int atSign = email.indexOf("@");
        return email.subString(0, atSign);
}
```

Building new Strings

Start with nothing 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

EXAMPLE

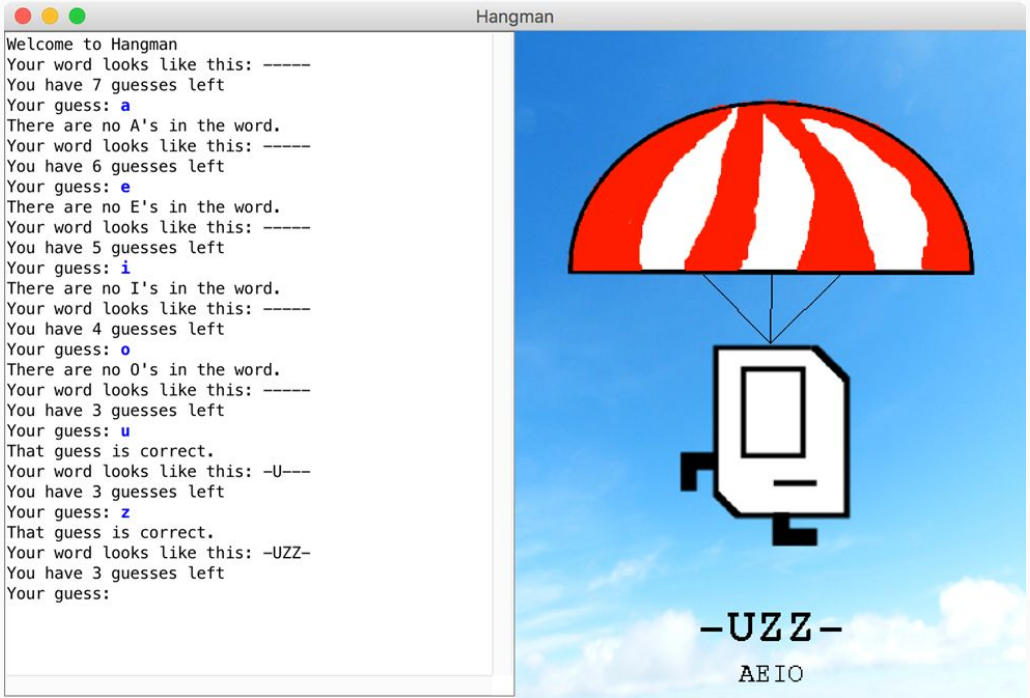
repeatLetters

Given a word, repeat each letter in sequence a given number of times

Solution: repeatLetters

```
private String repeatLetters(String input, int numTimes) {
     String result = "";
     for (int i = 0; i < input.length(); i++) {
         for (int j = 0; j < numTimes; j++) {
             result += input.charAt(i);
         }
     }
     return result;
 }
```

Hangman Game Flow



Console Based Game

- Choose random word (use stub implementation first!)
- Keep track of partially guessed word (building hint: decompose!)
- Need to also keep track of guesses/give user appropriate prompts/messages (what control structures can help us check/reprompt?)

Updating the hint string

Graphics

```
private void drawBackground() { // from handout
    GImage bg = new GImage("background.jpg");
    bg.setSize(canvas.getWidth(), canvas.getHeight());
    canvas.add(bg, 0, 0);
}

// note how we call all the graphics methods on the canvas
```

Split canvas

Make sure to have a private instance variable for the canvas:

```
private GCanvas canvas = new GCanvas();
```

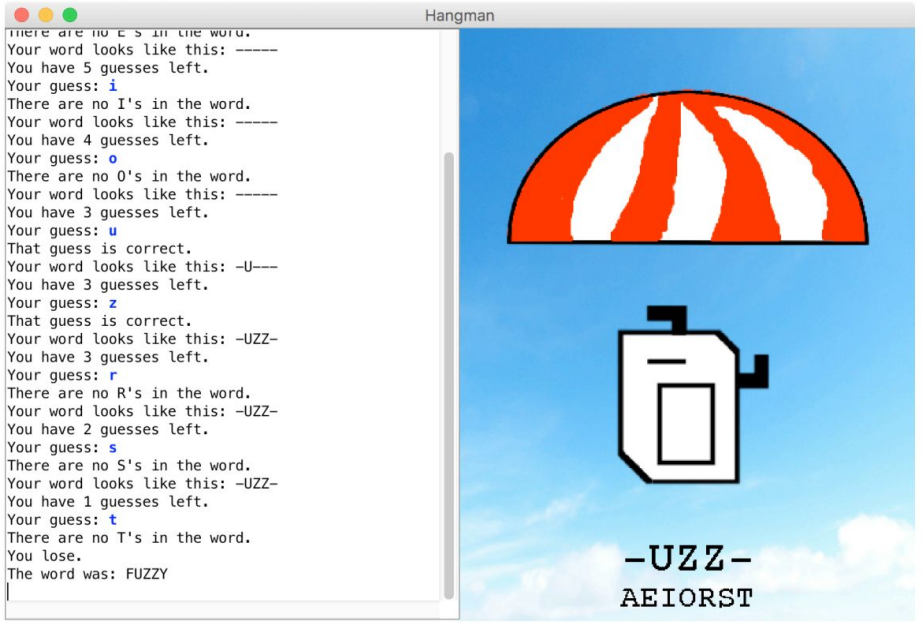
And also declare an init method that adds the canvas:

```
public void init() {
    add(canvas);
}
```

Graphics con’t

- objects should be centered
- break cords from outside in
- (first break furthest right, then left, etc.)
- Karel turns upside down when out of cords

Figure 5. When you run out of guesses, Karel runs out of hope



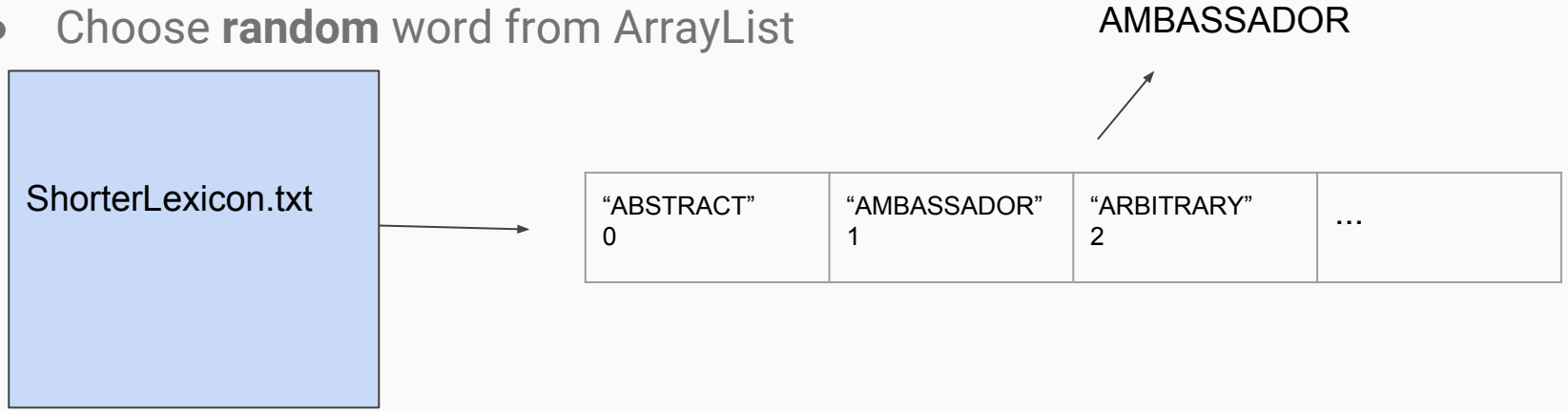
File Reading (from lecture)

```
try {
    BufferedReader br = /*…open the file… */
    while (true) {
        String line = br.readLine();
        if (line == null) break;

        /* … process current line … */
    }
    br.close();
} catch (IOException e) {
    throw new RuntimeException(e);
}
```

File Reading in Hangman

- Read words into ArrayList from file (HangmanLexicon.txt/ShorterLexicon.txt)
- Choose **random** word from ArrayList



Debugging tips

- Use print statements/the debugger
- Pick smaller/simpler test cases in the beginning
- Debug in small steps along the way

Things to Watch out For

- accept guess in either upper or lower case
- reprompt w/message if user guesses something other than single letter
- guessing incorrect letter a second time is another wrong guess
- do nothing if a correct letter is guessed more than once (you can still print out something indicating the guess is correct -- nothing else in the state changes)

Thanks!

Any questions?