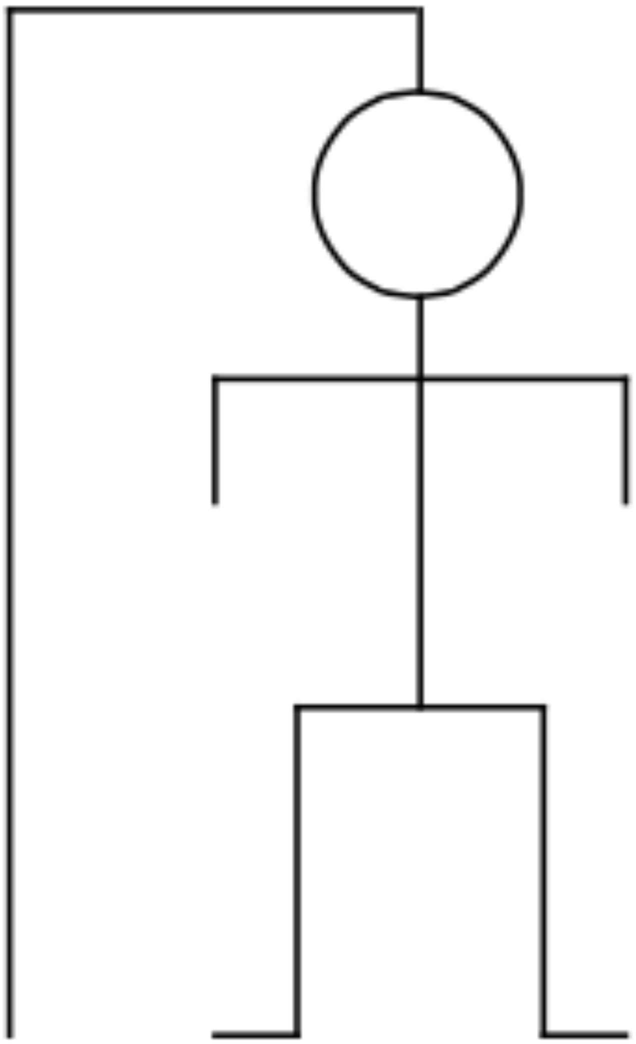


YEAH Session #4

October 24 2014, 4:15-5:15 PM
Nick Troccoli

YEAH Hours Schedule

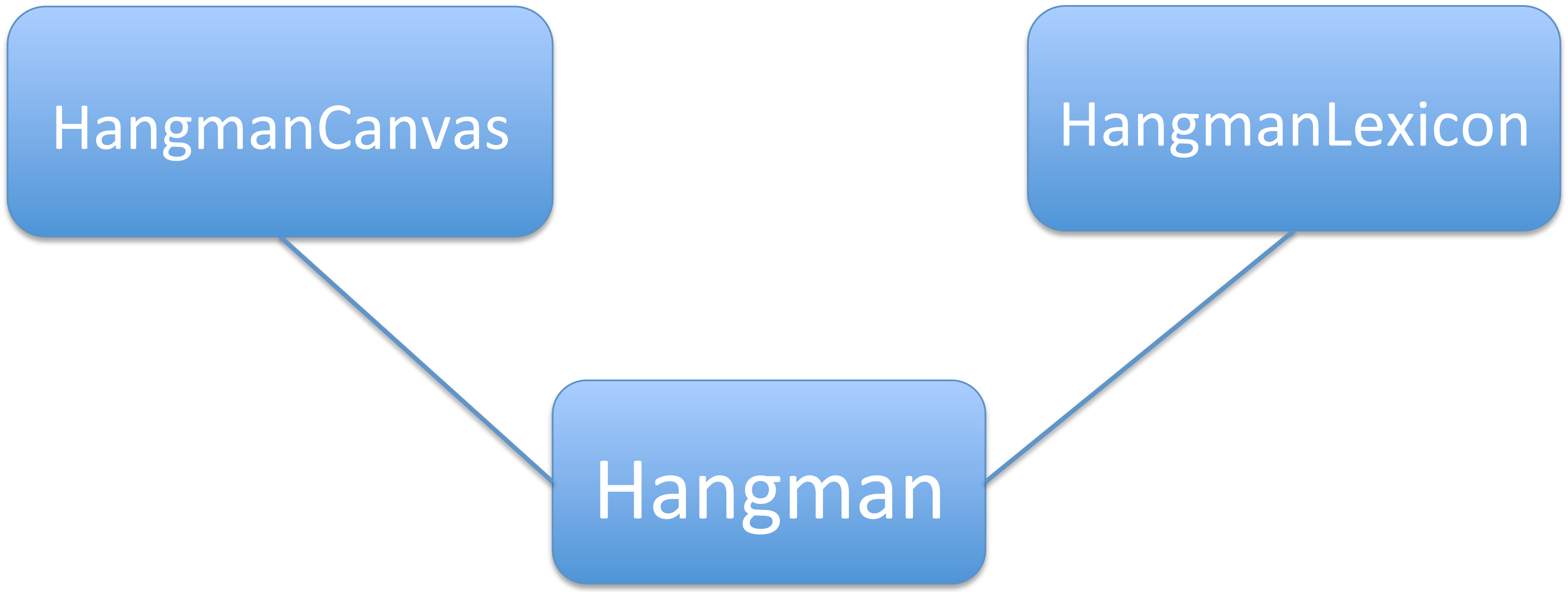
Topic	Date	Time	Location
Assignment 4	Today!	Now!	Here!
Midterm	10/29 (Wed)	7-9PM	MemAud
Assignment 5	11/4 (Tues)	6-7PM	BraunAud
Assignment 6	11/13 (Thurs)	6-7PM	Hewlett 200
Assignment 7	11/21 (Fri)	4:15-5:05PM	Hewlett 200
Final Exam	12/10 (Wed)	12:15-3:15PM	TBD

Classes and Instances

- A **class** is like a dictionary entry for something – it defines what something is supposed to do

- An **instance** is an *actual copy* of what that entry describes

Hangman Blueprint



```
/*
 * File: HangmanLexicon.java
 * -------------------------
 * This file contains a stub implementation of the HangmanLexicon
 * class that you will reimplement for Part III of the assignment.
 */

import acm.util.*;

public class HangmanLexicon {

/** Returns the number of words in the lexicon. */
   public int getWordCount() {
      return 10;
   }

/** Returns the word at the specified index. */
   public String getWord(int index) {
      switch (index) {
         case 0: return "BUOY";
         case 1: return "COMPUTER";
         case 2: return "CONNOISSEUR";
         case 3: return "DEHYDRATE";
         case 4: return "FUZZY";
         case 5: return "HUBBUB";
         case 6: return "KEYHOLE";
         case 7: return "QUAGMIRE";
         case 8: return "SLITHER";
         case 9: return "ZIRCON";
         default: throw new ErrorException("getWord: Illegal index");
      }
   };
}
```

Creating An Instance

```
HangmanLexicon lexicon =
   new HangmanLexicon();

int wordCount =
   lexicon.getWordCount(); // 10

String word =
   lexicon.getWord(0); // BUOY
```

Assignment 4: Hangman

- Due Monday, November 3rd at 3:15PM
- Good practice with multiple classes and strings
- Do it in parts!

Part I: Console Game

- Choose a random word
- Keep track of partially-guessed word
- Game structure – guess, guesses remaining, messages, game end, etc.

chars and Strings

Characters

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

Can’t just write (for line 2):

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

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.

Comparing chars

Lets 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

```
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.”);
}
```

Still 1 edge case to cover here! – EMPTY STRING

Strings

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

Can’t just write (for line 2):

```
s.toUpperCase();
```

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

Using portions of slides by Eric Roberts

Take 2

```
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.”);
}
```

Still 1 edge case to cover here! – EMPTY STRING

Comparing Strings

```
String s1 = “racecar”;
String s2 = reverseString(s1);
// How do we check equality?
if(s1.equals(s2)) {
   ….
}
---------OR----------
if(s2.equals(s1)) {
   ….
}
```

Don’t do this!

```
String s1 = “racecar”;
String s2 = reverseString(s1);
// How do we check equality?

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 Strings

- 1) Use substrings – smaller pieces of strings

OR

- 2) Make new string, build up over time

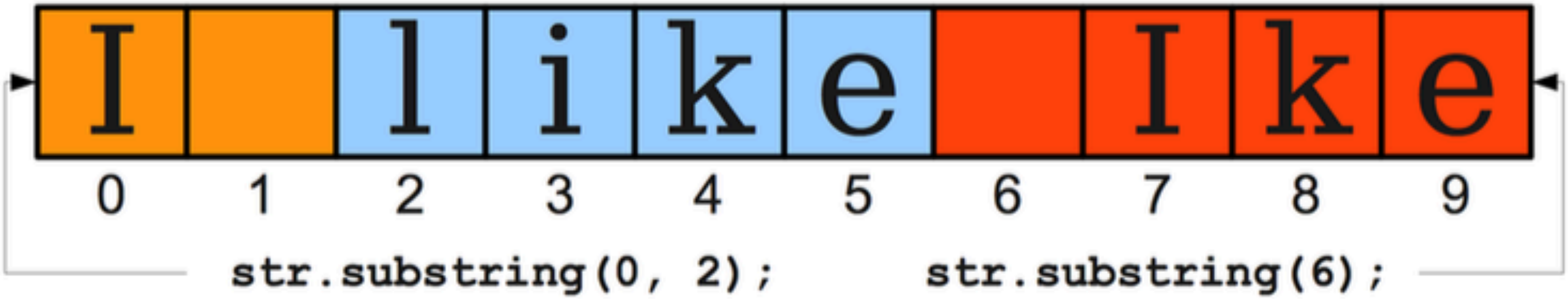
1) Obtaining Substrings

- 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*)**



2) Building a New String

- Start with nothing and build up a new string
- Iterate through the string
- Use Character methods at each position to decide what to concatenate to the new string
- See this week's section handout for examples

Game Flow

`String secretWord` S E C R E T

`String wordState` _ _ _ _ _ _

`char guess` e

`String newWordState` - E - - E -

(most important slide!)

Guess 2

`String secretWord` **S E C R E T**

`String wordState` **- E - - E -**

`char guess` note: guesses are case-insensitive R

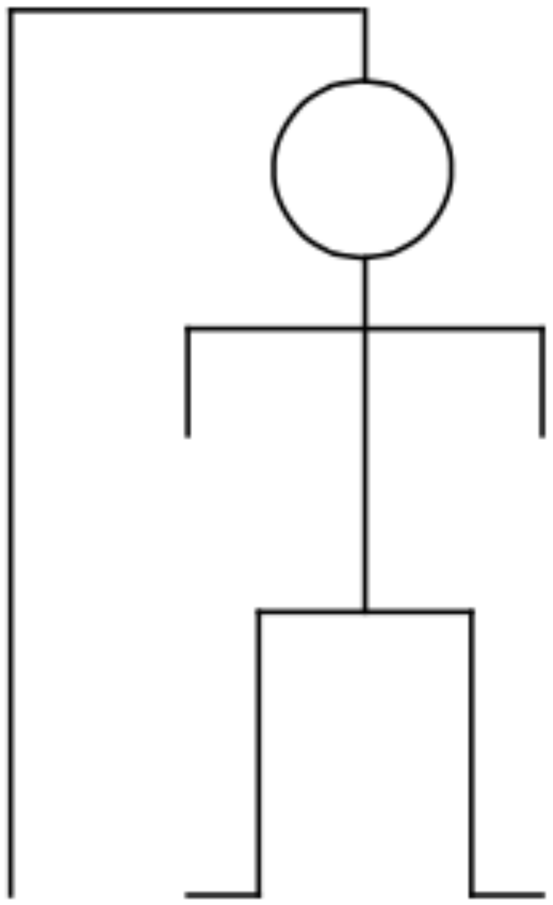
`String newWordState` **- E - R E -**

¿¿ `int guessesLeft` ??

User Guesses

- Case-insensitive
- Only 1 letter guesses allowed
- Re-guess correct guess – do nothing
- Re-guess incorrect guess – another wrong guess!

Part II: Graphics



HangmanCanvas.java

```
public void reset() {
...
}

public void displayWord(String word) {
...
}

public void noteIncorrectGuess(char guess) {
...
}
```

HangmanCanvas Usage

In hangman.java:

```
private HangmanCanvas canvas;
...

public void init() {
  canvas = new HangmanCanvas();
  add(canvas);
}
```

Part III: Files

- BufferedReader – open, read, close
- try/catch
- Read in line by line and store all lines in an ArrayList
- “catch” an error if there is one
- close your BufferedReader!
    - .close()

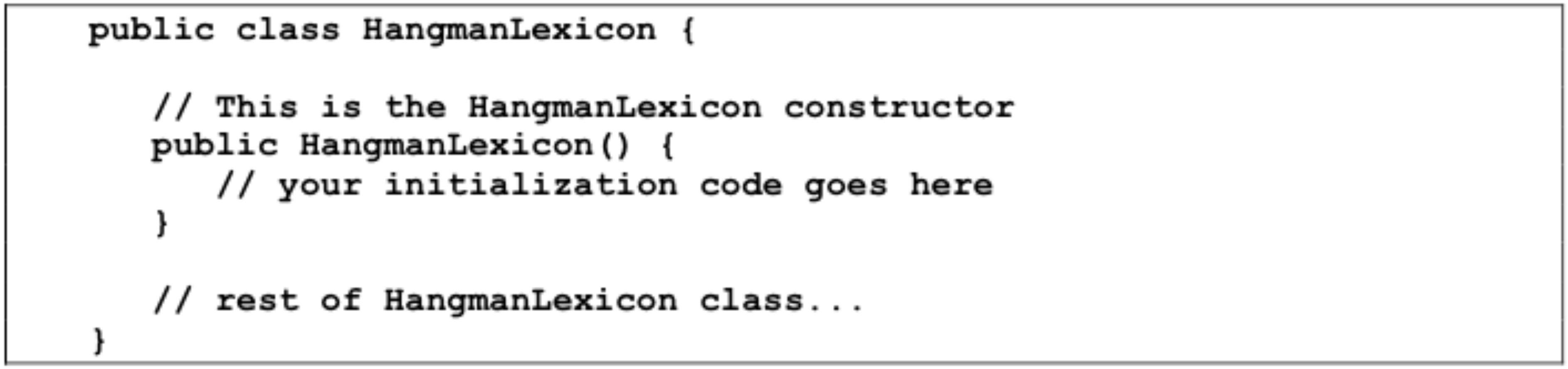
```
try {
   BufferedReader rd = new BufferedReader(new
      FileReader(“test.txt”));

   while(true) {
      String line = rd.readline();
      if(line == null) break;
      println(line); // do something with line
   }
   rd.close(); // close when you’re done!
} catch (IOException ex) {
   // do something in response to exception
   throw new ErrorException(ex);
}
```

Careful with readline (BufferedReaders) vs. readLine (getting input from the user)!

ARE YOU AN EXCEPTION?

BECAUSE I CAN'T WAIT TO CATCH YOU.

Constructors



```
public class HangmanLexicon {

    // This is the HangmanLexicon constructor
    public HangmanLexicon() {
        // your initialization code goes here
    }

    // rest of HangmanLexicon class...
}
```

```
HangmanLexicon lexicon =
  new HangmanLexicon(); // triggers
HangmanLexicon constructor above
```

Constructors

- A **constructor** is a special method defined in a class that is responsible for setting up class's instance variables to appropriate values.
- Syntax:

```
public NameOfClass(parameters) {
  /* … body of constructor … */
}
```

- Inside a constructor:
  - Give initial values to instance variables.
  - Set up instance variables based on values specified in the parameters.
- Constructor called when instance created with `new`.

Testing/Coding Tips

- Manually set the word to guess so you know what it is each game
- Watch the cases of your strings/chars!
- Add extra printlns along the way if you want to know what your string or char variables are

Final Tips

- Follow the specifications carefully
- Extensions! Graphics, etc.
- Comment!
- Go to the LaIR if you get stuck
- **Incorporate IG feedback!**

- Have fun!