**Chapter 6 Book Work**

**Logan Yeubanks**

6.2: It shows method, constructor, and class descriptions. All seem to have these features but to different lengths.

6.3: The difference between the two are the amount of parameters found in them. One asks for a string and int while the other is just a string.

6.4: The endsWith method searches for if something ends with the parameter inputted. There is one parameter of the string type and then whatever suffix wanting to be searched for.

6.5: The length() method will return how many characters were in the string passed into the parameter.

6.6: I found it by going to the string section and scrolling. You could even do ctrl f and type in the method if you know the name of it.

6.7: trim gets rid of white space found in the inputted string.

text.trim();

6.8:

While (!finished)

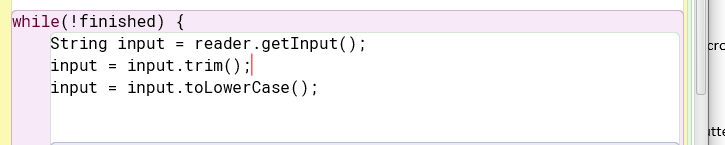
{

String input = reader.getInput();

Input = input.trim();

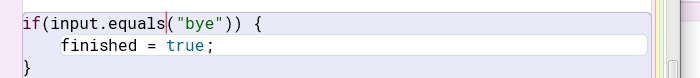
}

6.9:



6.10: The equals method returns true if the 2 strings equal each other, and false if they do not.

6.11:



6.12: Random is in the java.util package and can also be used to generate a random number.

It assumes the lower boundary starts at 0 and accepts a upper boundary, and then creates a random number between the 2.

6.13:

Random rand = new Random();

upperBound = 5

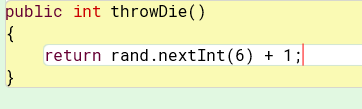
Int number = rand.nextInt(upperBound

6.14:

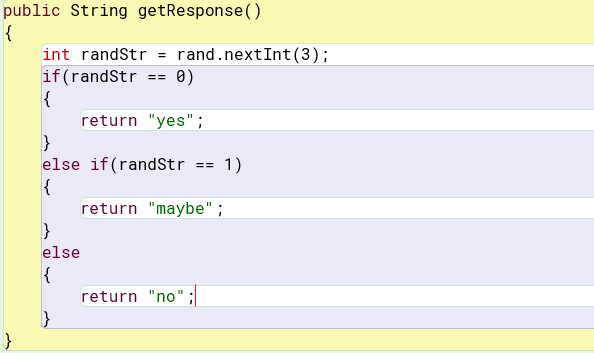


6.15: If 100 is passed in as the parameter in nextInt() then a random number is made between 0 and 100. This is because the lower bound is 0 and 100 is passed in as the upper bound.

6.16:



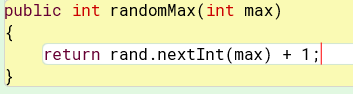
6.17:



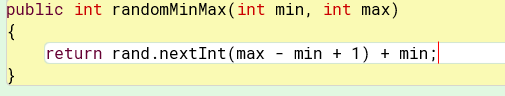
6.18:



6.19:



6.20:



6.21: I think if things are random it is much easier to hack into. It would be overkill but possible I think to use in this project.

6.23: If you add more responses it will still work and generate the responses added randomly as well.

6.24: A Hashmap is similar to an ArrayList but each entry is mapped to a key instead of a index number.

6.25:

public Set<Map.Entry<K,V>> entrySet()

public Collection<V> values()

public V remove(Object key)

public V put(k key, V value)

I think the same type could be used for both parameters.

6.26: The int size() method returns how many keys are found in a hashmap.

6.28. It will replace the previous entry and keep the same key.

6.29: They will both be added but accessed with different keys.

6.30: the containsKey method checks to see if a key is already in a map.

If a hashmap was called “map” the code would look like:

boolean contained = map. containsKey(key)

6.31: Nothing is returned when a non-existent key is entered.

6.32:

Iterator I = key.iterator();

While(i.hasNext))

{

System.out.println(i.next));

}

6.34: A HashSet and ArrayList both store things but the HashMap cannot have duplicates and does not have a order while ArrayList does.

6.35: You could split things based on the character sequence or a character sequence and a limit.

6.36:

For a space: String []stringSplit = str.split(“ ”);

For a colon: String []stringSplit = str.split(“:”);

6.37: The difference between returning words in a HashSet vs an ArrayList is that a HashSet will not return them in a certain order while a ArrayList will. There will also be no duplicate words returned in a HashSet.

6.38: It looks like it counts the second space as a empty string but doesn’t cause a issue.

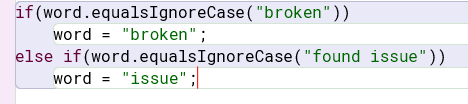
6.39: It looks like it converts a array to a list but makes sure it is a fixed size.

6.40: java.Arrays.toString(int[], i) will return a string from a element in a array.

Java.Arrays.equals(int[] first, int[] second) will return true if the array elements are equal and false otherwise.

6.41- 6.44: Added lines from text into the different classes.

6.45:



6.46:

Else if(word.equalsIgnoreCase(“already seen”))

Word = “already seen”;

6.47:

else if(word.equalsIgnoreCase(“unknown”))

word = “why”;

6.48: The putIfAbsent() method puts a new key value mapping in if it is not already in the map.

6.49:

If(count == 0)

{

System.out.println(“Word not found”);

}

Else

{

System.out.println(“There are “ + count + “words after goodbye”);

}

6.51: All parts generated are accurate and useful. It includes method, constructor details.

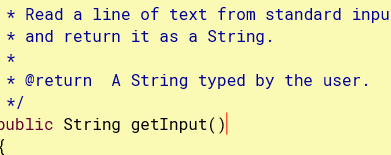
6.52: @param shows the details of the parameters in the method

@returns shows the details of what is returned from a method

@see allows for a picture to be implemented.

6.53: Some examples of others are @author which shows the author of the program and @link which shows links.

6.54:



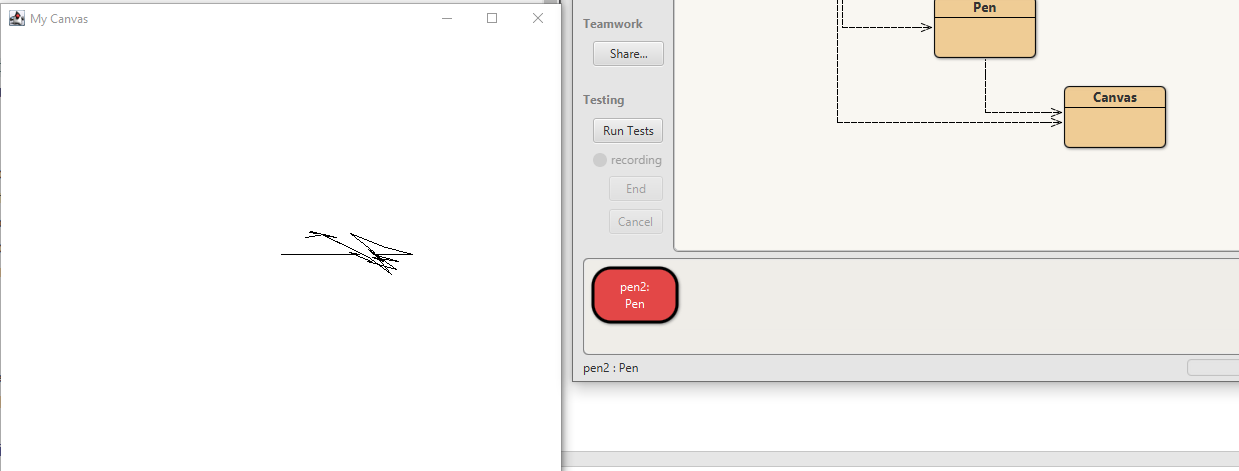
6.55:

colorScribble() creates a bunch of scribbles with different colors

drawSquare() makes a blue square

drawWheel() makes a red wheel with a design attached to it.

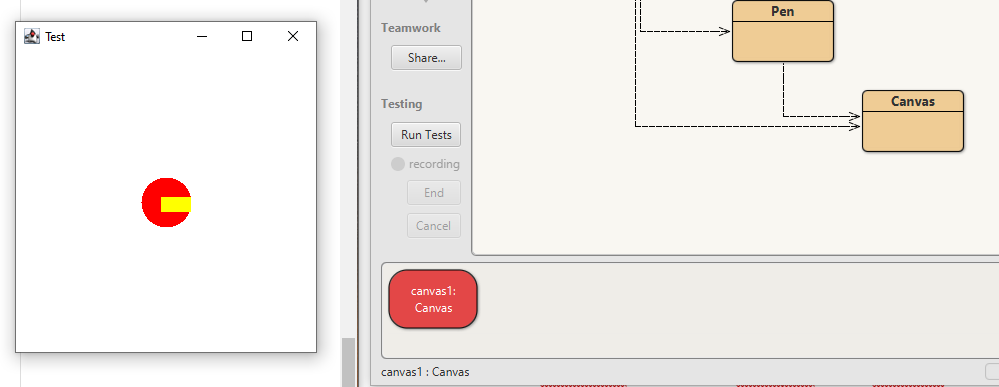
6.56:



6.58: Color constant for colorScribble is random while drawSquare is set to Color.BLUE

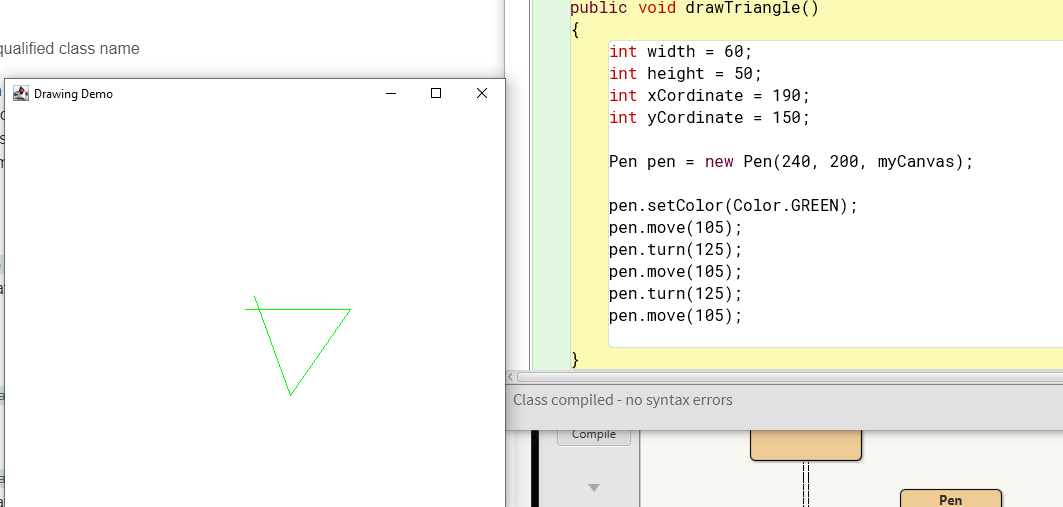
6.59: color.GREE color.YELLOW, Color.CYAN, and Color.ORANGE

6.60:

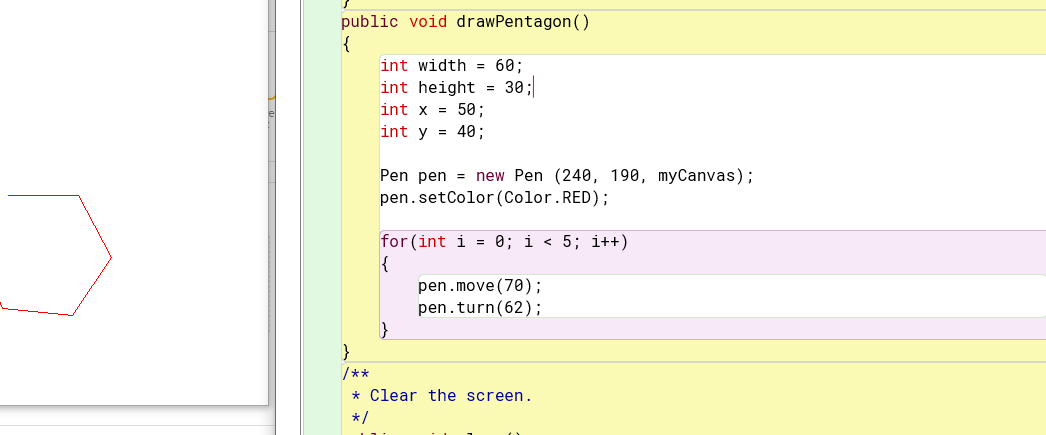


6.61: The erase() method erases the canvas

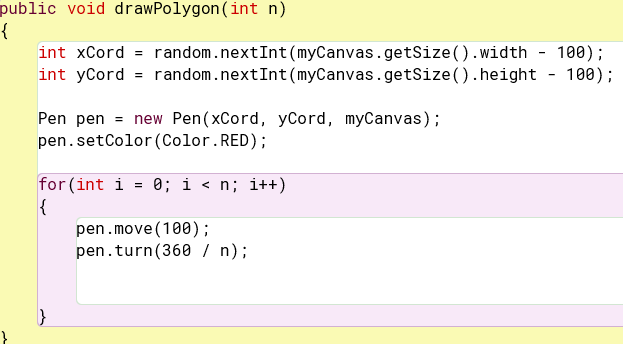
6.62:



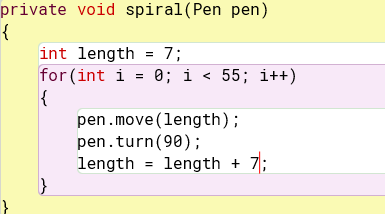
6.63:



6.64:

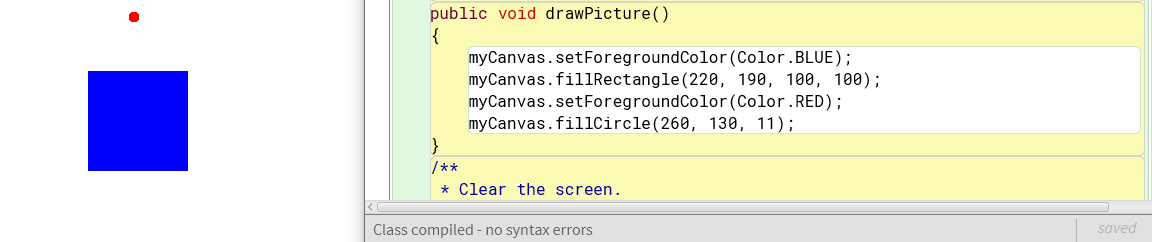


6.65:



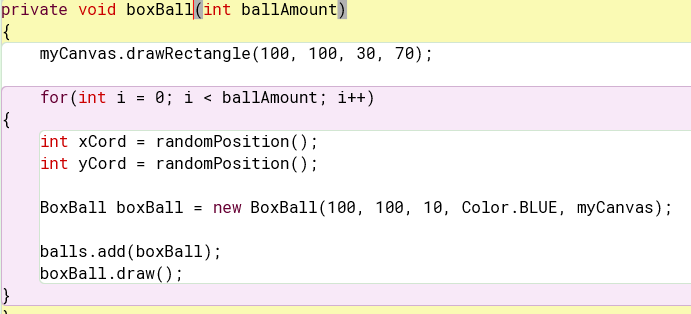
6.66: 16 methods are shown when I push ctrl-space.

6.67:



6.69: The HashSet is the best choice because each ball bounce is unique.

6.71:



6.73: The speed in which the balls fall changes when the gravity value is changed.

6.74:

public final double tolerance = 0.001;

private final int passMark = 40;

public final char command = ‘h’;

6.75:

Integer.MAX

Integer.MIN

Double.MAX VALUE

Double.MIN VALUE

Math.PI

Math.E

6.76: It would make sense in order to define what that number is with a good variable name as well as stop if from being changed later on because it is the same throughout the program. A variable will also allow for easier access later on.

6.77: Math.max() returns the max of 2 numbers

6.78: They could not be wrote as instance methods and this is okay because it saves memory and gives every class the same method due to it being static. There are no state of the method locally.

6.80: Static methods are able to get called from instance methods without a object but instance methods can’t be called from static methods.

Static methods can use another static method without a object being needed.

6.82:

public static void main(String [] args)

6.84:

