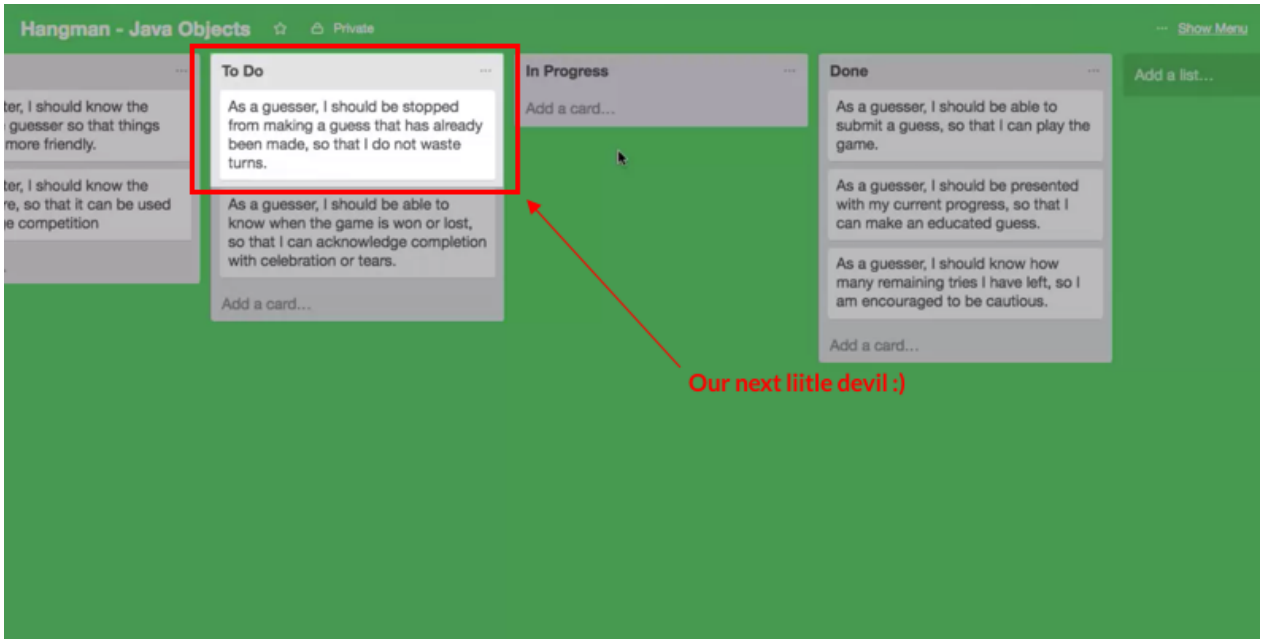
Java Objects Part 4 Notes

Team Treehouse

May 26, 2020

1 Exceptions



- *throw new EXCEPTION_NAME:* raises exception *EXCPETION_NAME*
- *try* and *catch:* handles expections

```
1    public class Game {
2        ...
3        public boolean applyGuess(char letter) {
4            if (misses.indexOf(letter) != -1 || hits.indexOf(letter)
   != -1) {
5                throw new IllegalArgumentException(letter + " has
   already been guessed"); // <- this little guy here :)
6            }
7
8            ...
9        }
10
11       ...
```

```
12      }
13
```

Listing 1: lesson_01/Game.java

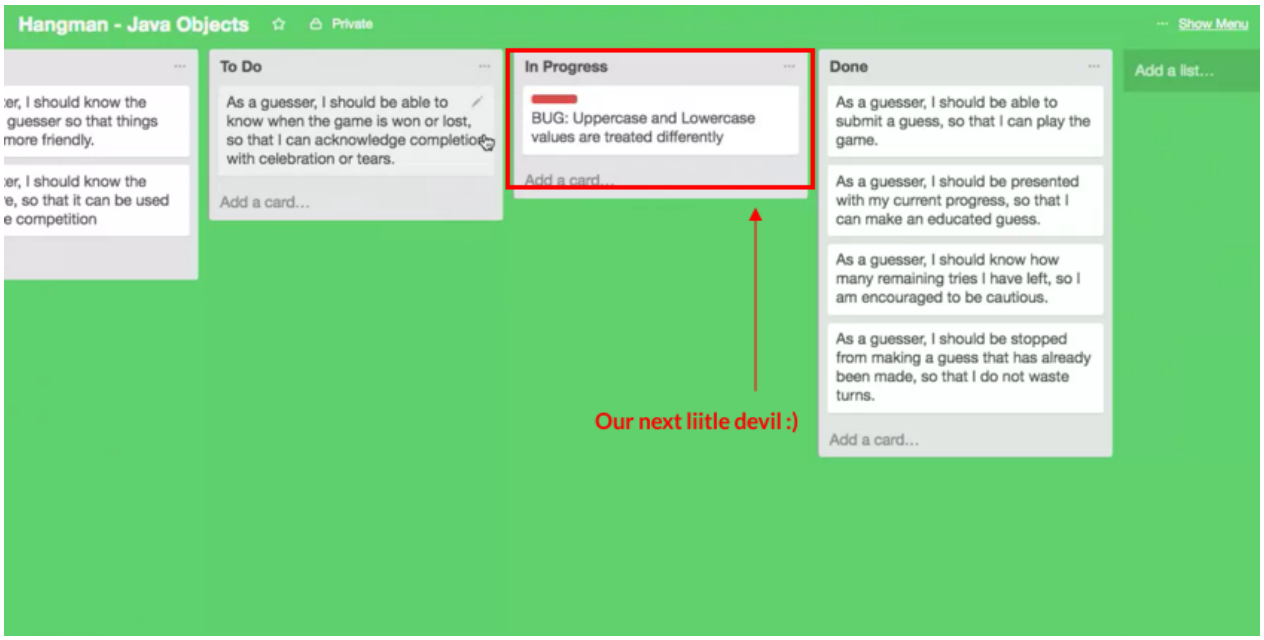
```
1   import java.util.Scanner;
2
3   public class Prompter {
4       ...
5       public boolean promptForGuess() {
6           ...
7           boolean isHit = false;
8           try { // <- And this little guy here :)
9               isHit = game.applyGuess(guess);
10          } catch (IllegalArgumentException iae) {
11              System.out.println(iae.getMessage());
12          }
13
14          return isHit;
15      }
16  }
17
```

Listing 2: lesson_01/Prompter.java

Notes:

- Files can be compiled and displayed by typing *javac Hangman.java && java Hangman* in terminal

2 Validating and Normalizing User Input



- *Character.toLowercase(CHAR_VAR)*: turns value in *CHAR_VAR* to a lowercase character

```
1    public class Game {
2        ...
3        private char normalizeGuess(char letter) {
4            ...
5            letter = Character.toLowercase(letter); // <- This little
     guy here :)
6            ...
7        }
8    }
9
```

Listing 3: lesson_02/Game.java

Notes:

– Files can be compiled and displayed by typing *javac Hangman.java && java Hangman* in terminal

3 Exercise 2

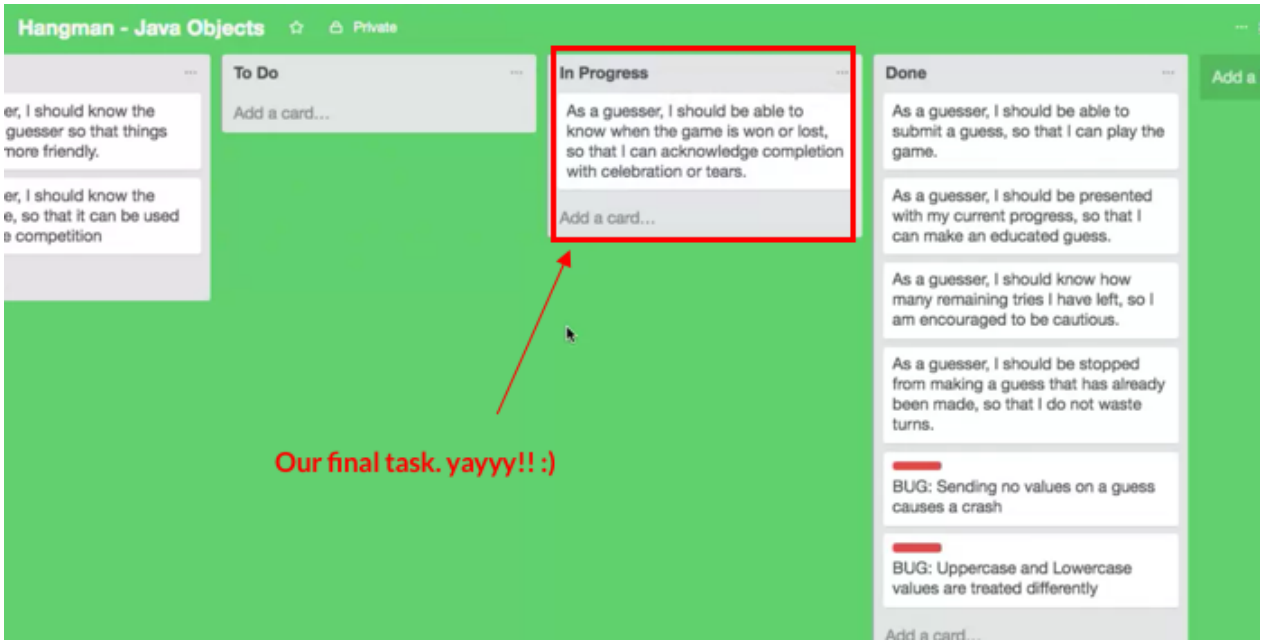
- Solution included in *exercise_2.java*

4 Using Method Overloading

Notes:

- Files can be compiled and displayed by typing *javac Hangman.java && java Hangman* in terminal

5 Determining if the Game is Won



Notes:

- Files can be compiled and displayed by typing *javac Hangman.java && java Hangman* in terminal

6 Quiz 1

1. Below is some code for a carnival based Dunk Tank. As long as there are tries remaining and the person has not been dunked keep throwing the ball.

   Please choose the answer that properly fills in the blanks with the proper code for the while loop

```
1    int remainingTries = 3;
2    while (remainingTries __ 0 __ __ dunkTank.isDunked()) {
3        throwBall();
4    }
5
```

Listing 4: lesson_02/Game.java

A. >,&&,!

B. =,——,!

C. !=,——,@

Answer: A

2. Due to the separation we have chosen, what outcome can we expect?

A. We will be able to run this in other languages like JavaScript or Python.

B. We can more easily generate code using external tools.

C. We will be able to use the same game logic in other applications, such as console applications, web sites and apps.

Answer: C

7 Arrays and Command Line Arguments

- *string[] args:* represent arguments passed in commandline

```
>>> java Hangman corgi # <- 'corgi' is stored in args[0]

```

Example:

```
public class Hangman {

    public static void main(String[] args) { // <- this guy here :)

        if (args.length == 0) {
            System.out.println("Usage: java Hangman <answer>");
            System.err.println("Answer is required");
            System.exit(1);
        }
        Game game = new Game(args[0]);
        ...
    }

}

```

Listing 5: lesson_07/Hangman.java

Notes:

- Files can be compiled and displayed by typing *javac Hangman.java && java Hangman <answer>* in terminal
  - i.e. javac Hangman.java && java Hangman corgi

8 Wrapping Up