CSCM41J Introduction to Programming with Java — February 18th 2022
Leigh Clark, Cécilia Pradic

Coursework: due Monday 14th March 2022

1. Submissions will be individual
2. Please submit exactly one file that is a compressed archive (.zip or .tar.gz) containing your code, even if it is a single file. The filename should be `<your student number>.{zip,tar.gz}`.
3. If you have several files, please have a file named ReadMe.md indicating what is the class containing the main function (you can call it e.g. `Wordle`) and other pertinent details; otherwise I will assume that I can run your code by compiling every .java file in your project and running the `java Wordle`.
4. You can define as many additional auxiliary methods and classes as necessary to complete the coursework, although you should be able to manage without having to use any of the object-oriented features discussed in week 4.
5. You may only use classes that are part of the java development kit version 17 (I will be running your code with `openjdk 17.0.1`). In particular, no external libraries or platform-dependent code should be in there.
6. Marks will be awarded for both correct functionality and good code quality as well as for complying with the instructions. "Good code quality" will mean here essentially readability; if you find yourself copying and pasting your own code to complete the assignment, please try to think about making things better by declaring auxiliary methods. Do not hesitate to make use of comments can help make the code readable.
7. **The university takes plagiarism very seriously. With the submission you take responsibility that this is your own solution, and that that you written the code yourself.** We will run an automated tool to help with detecting plagiarism and if we have any doubts, will signal our academic misconduct officier. (We won't penalize if you use some piece of code included in this PDF to handle coloring or file reading of course)
8. Don't hesitate to use next week's lab slots to work on the coursework and ask for help.
9. Feel free to indicate any further relevant information in your submission.

1 The game

Your main goal for this coursework is going to implement a game similar to *Wordle*, which is essentially a variation of the hangman game which has allegedly become very popular during the pandemic[1]

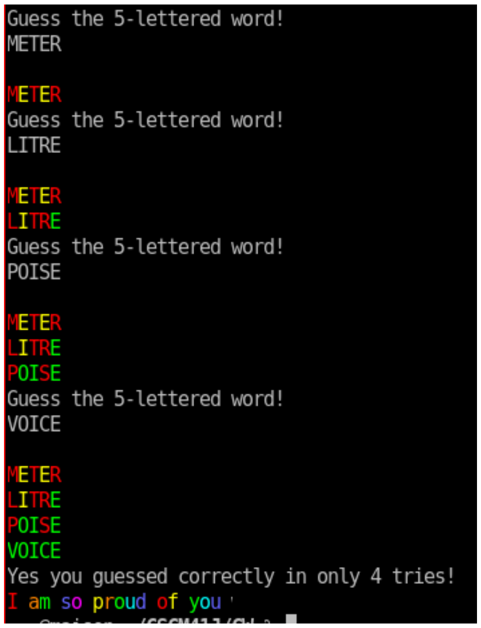
The principle is relatively simple: your program will select randomly a word from a dictionary of a given length and you will ask the user to guess it. The user is only allowed to guess words that have the exact same length. If the guess is correct, the user wins; if not, the computer colors each letter of the guess as follows:

- if the letter also occurs in the same position in the word to be guessed, then it is colored green
- if the letter also occurs in the word the user is trying to guess, but not in the same position, it is colored yellow
- otherwise, we color it red

This guessing process is then reiterated only a finite number of time in some version of the game, but let us be lenient and allow arbitrarily many guesses. The goal then for the user is to guess the word in as few tries as possible.

I coded a version of the game and took a screenshot of a run that you can check out in fig. 1

[1] I mostly know it through one of the most boomeresque french tv show; this causes me no small amount of cognitive dissonance.



```
Guess the 5-lettered word!
METER

METER
Guess the 5-lettered word!
LITRE

METER
LITRE
Guess the 5-lettered word!
POISE

METER
LITRE
POISE
Guess the 5-lettered word!
VOICE

METER
LITRE
POISE
VOICE
Yes you guessed correctly in only 4 tries!
I am so proud of you '
```

Figure 1: A screenshot of a play of a command-line version of Wordle, soon to be implemented by you! You can probably figure out what was inputted by me and what was outputted by the program, which recalls all guesses made so far after each guess.

2 Your task

Your task is imply to implement in java a version of Wordle that behaves in a similar way as the one in fig. 1: at each guess, it should display all previous guesses, colored as described above.

Your first goal is to have something functional that can at the very least reproduce the input-output behavior as in fig. 1. I then expect you to try to improve that program so that

1. it is resilient against bad input; if the user enters a string with numbers or non-letter characters, then I would like the program to keep asking for a valid input.
2. it is case-insensitive (i.e., if the user entered "METER", "meter", or "mEteR" in the first step above, the result should remain the same)
3. it can be run using a custom dictionary provided as a separated `.txt` file containing one word on each line
4. if using a large custom dictionary, you should can also restrict the user inputs so that it needs to match at least one word from that dictionary.

To get marks above 90, I would suggest also pursuing one of the following stretch goals:

- Allow the use to enter, in addition to guesses one or two commands like `:help`, `:giveup`, `:info` or `:hint` so that
  - typing `:help`, a message explaining how the game is played and what are the available commands is displayed before resuming the game
  - typing `:giveup` prints out the word that was to be guessed and closes the program
  - typing `:info` rephrases the information obtained with guesses so far in a human-readable way; for instance, after the guess "LITRE" of fig. 1, it might print out "The word to be guessed has shape —-E, contain the letter(s) I as well and does not contain the letter(s) M, T, R and L."
  - typing `:hint` suggests to the player a number of (valid) guesses for the next step

- Design a scoring system a bit more elaborated than the number of guesses, based on the hardness of the word to be guessed.
- Try to design a more adversarial version of the game where the goal is not determined in advance, a bit in the spirit of e.g. `https://qntm.org/absurdle`.

If you go for one of these stretch goals, please try to detail what you have done in the readme or in comments, especially when non-trivial design decisions are involved.

3 Technical matters

For the most part, you should be able to get started (as long as you diligently check out the documentation for the `String` class and useful methods therein) except for two things: colors in the terminal and having an easily modifiable custom list of words.

I will attempt to let you know how to handle these in week 5.

3.1 Custom dictionary and reading files

For the first versions, you can simply use a small list of words defined in your source file. For instance, you could have a declaration

```java
final static String[] DICTIONARY = { "METER", "LITER", "SPEED", "COLOR", "PHONE" };
```

somewhere at the beginning of your file and just work with that to code the first version of your program. But ultimately you'd like to work with a larger set of words that is easily customizable. For this we can use text files containing one word per line, as provided on canvas, and read that file from your Wordle program.

There are a few ways for reading files in java; I include one of them in fig. 2 in a program that computes the number of words in a file with commentary.

3.2 Colors

One of the easiest way of playing with colors in the terminal is to use special string that tells your terminal stuff like "from now on, print everything in red". For some context, you may consult for instance `https://en.wikipedia.org/wiki/ANSI_escape_code`. I provide a small example of a program using those in fig. 3.

I suggest you try to run it *before trying to put colors in your submission*, **especially if you are running Windows**. Microsoft only included support for ANSI escape codes to their terminal in the late 2010s and it is often not turned on by default in their terminal emulator. On my home machine, I had to create a new registry entry: at `HKEY_CURRENT_USER\Console`, I added the `DWORD` value `VirtualTerminalLevel` and set it to 1 for the colouring via ANSI code to work.

If you cannot colors to work in your terminal, please feel free to find an alternative way of displaying the accuracy of one's guess using only characters. For instance, you could display `+-**-` below `LITRE` to mean that the first letter is indeed L, that I and E occur somewhere else in the word and and that T and R do not occur there.

Of course you wil not be penalized if you do not use colors in program, but I thought it'd be nice to give you the opportunity to play with that!

```java
import java.util.Scanner;
import java.io.File; // Note this new class!

/* This program will take a path to a file as an argument on the command line
 * and count the number of words that this file contains. */

public class CountWords
{

    public static void main(String[] args) throws Exception
      // This "throws Exception" is a required annotation because we create
      // a Scanner from a File object. This might raise an error. Those are
      // called exceptions in java and you might see later how to handle them
      // properly. For now we can be optimistic and assume you will only try to
      // open files that exist :)
    {
      // args is an array containing all the arguments we feed to the program on
      // the command line
      //
      // If there are no arguments, we simply exit
      if (args.length == 0)
        return;
      // Otherwise, we create a file object with the path given as first argument
      // of the program
      File file = new File(args[0]);
      // We can now create a new Scanner that will read that file using a constructor
      Scanner in = new Scanner(file);

      int counterWords = 0;
      //then the Scanner can be used with the same methods we have been using so
      //far for the standard input
      while(in.hasNext())
      {
        in.next();
        counterWords++;
      }
      System.out.println(args[0] + " contains " + counterWords + " words");
      //closing Scanners built from File objects can be important if you want
      //to reopen the file in the same program.
      in.close();
    }
}
```

Figure 2: Example of a program that reads a file in Java.

```
public class Colors
{

    public static final String RESET  = "\u001B[0m";
    public static final String RED_BRIGHT = "\u001B[091m";
    public static final String YELLOW_BRIGHT = "\u001B[093m";
    public static final String GREEN_BRIGHT = "\u001B[092m";
    public static final String CYAN_BRIGHT = "\u001B[096m";
    public static final String BLUE_BRIGHT = "\u001B[094m";
    public static final String MAGENTA_BRIGHT = "\u001B[095m";
    public static final String ORANGE = "\u001B[38;5;208m";

    public static String colorFromInt(int i)
    {
      switch(i % 7)
      {
        case 0:
          return RED_BRIGHT;
        case 1:
          return YELLOW_BRIGHT;
        case 2:
          return ORANGE;
        case 3:
          return GREEN_BRIGHT;
        case 4:
          return CYAN_BRIGHT;
        case 5:
          return BLUE_BRIGHT;
        default:
          return MAGENTA_BRIGHT;
      }
    }

    public static String rainbow(String input)
    {
      String res = "";
      for(int i = 0; i < input.length(); i++)
      {
        if(input.charAt(i) != ' ')
          res += colorFromInt(i);
        res += input.charAt(i);
      }
      return res + RESET;
    }

    public static void main(String[] args) throws Exception
    {
      System.out.println("Hello" + RED_BRIGHT + " mortal" + RESET + "!");
      System.out.println(rainbow("Life is colorful!!!"));
    }
}
```

Figure 3: A program using ANSI escape code to produce a colorful output