Assignment 2

Object Orientation

Spring 2020

1 Learning Objectives

After making this assignment, you should be able to

- Design and use Java classes and objects properly, especially using encapsulation and model-view separation
- Split a program into a model and a view
- Work with the classes `String` and `StringBuilder` and know their difference

2 Hangman With Classes and Objects

In this exercise you implement the popular game Hangman. Your task is to design classes and methods that implement the different aspects of the game.

2.1 The Game

The goal of Hangman is to guess a word that the computer has in mind. The user sees a line of dots, where each dot represents a letter of the word to be guessed. The user then inputs a letter. If the letter is in the word, all dots at the positions of that letter are replaced by the letter. If the letter is not in the word, a part of a virtual gallows is constructed. This continues until either the player has made enough mistakes to complete the gallows or the player mentioned all letters of the word. The amount of mistakes that can be made differ per version of the game, but a maximum of 10 is a commonly used value.

2.2 Producing Words for the Game

When the game starts, the computer has to pick a word for the player to guess. For this exercise, we provide a class `WordReader` that can do this. Later in the course you will be able to write such a class yourself, but for the moment it contains too many nasty Java details.

When you create an object of class `WordReader`, it reads all words from a given file. The word file must be specified as constructor argument. Once the file has been processed, the method `WordReader.getWord` returns a randomly chosen word. The file `words.txt` contains several example words. The file containing the words should be in the root folder of your Java package. You can add or delete words from the word file as you wish.

2.3 String and StringBuilder

The word that needs to be guessed and the word as it has been guessed so far need to be remembered. In Java, text is stored in objects of type `String`. Strings are immutable, which means there is no way to change the content of such an object.

When you want to modify an existing string, you have to make a copy that has the desired modifications. This is often inconvenient. Java has a class called `StringBuilder` for mutable strings. See `http://docs.oracle.com/javase/8/docs/api/` or the help of NetBeans for a description of the available methods.

2.4 Encapsulation

One of the key design principles in object orientation is *encapsulation*, which means hiding data inside an object and providing methods that manipulate this data. Make sure that all mutable attributes in your classes are `private`. Define getters to make this data available and setters for the attributes that should be mutable.

2.5 Model-View

Another key design principle in programming is *separation of concerns*. This principle is often applied to separate data, logic, and user interface. The user interface, often called *view*, should be separated as much as possible from logic and data, the *model*. In this assignment it should be possible to design a completely different user interface without changing the rest of the program.

2.6 Different Ways to Use The Model

To make the game a little more interesting, you should provide two ways to start a new game. One way is to let `WordFinder` randomly select a word, the other is to let the user enter a word. To accommodate this, you need at least two constructors of the model class. If the model class is called `Gallows`, it has the constructors `Gallows(String s)` and `Gallows()`. The first one uses a word that was provided by the user and the second one goes to the `WordReader` class to get a word there.

2.7 Example

Here is an example session with a console-based user interface of the game.

```
Welcome to Hangman!
Please enter a word or press Enter to randomly pick one
>
Randomly picking a word.
Remaining mistakes: 10
Guessed letters: []
Word: ......
> a
a is in the word
Remaining mistakes: 10
Guessed letters: [a]
Word: .a..a.
> e
```

```
e is not in the word
Remaining mistakes: 9
Guessed letters: [ae]
Word: .a..a.
>
```

3 Your Tasks

- Create a class `Gallows` that stores the state of the game. The state includes the word to be guessed, the number of tries, and the list of guessed letters, and whatever else you need.
- `Gallows` should have the following methods.
  - Guess a new letter
  - Get the word as dots with only correctly guessed letters shown
  - Get the guessed letters as String
  - Determine if the word has been correctly guessed
  - Other helper methods if you need them
- Create a class `GallowsUI` for the command line user interaction.
- The Main class should only start the UI.

4 Submit Your Project

To submit your project, follow these steps.

1. Use the **project export** feature of NetBeans to create a zip file of your entire project: File → Export Project → To ZIP.

2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Do not submit any other archive format. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.