CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 2

1. Introduction

You will implement a program that represents and plays the game of Hangman. This homework helps students to gain experience with fundamental concepts of object-oriented programming, including:

- Classes vs Objects
- State vs Behavior
- Encapsulation
- Constructor and Instantiation
- Accessors
- Overriding method `Object.toString`

We will implement the Hangman game for just one player. The program will choose the word from a dictionary file and the user must guess the word within a limited number of guesses.

Reading Dictionary Files

The dictionary of words that the program will use to select the secret word from will be stored in a .txt file extension. Each line of the file will be a word.

To read the file use the `java.util.Scanner` class.

2. Provided Files

- dict.txt is a file that contains the Java keywords. We will use this file as our dictionary for selecting the word to guess.
- dict_hard.txt, another dictionary file.
- Example of Javadoc documentation here.

3. Design

Activity 1: UML diagram (3 points)

For this homework, the design of the program has been provided. You are encouraged to take the time to examine the design for each class until you understand how everything works (the javadoc provided will help you to have a better understanding of each class).

As part of your final submission, you will need to provide a complete UML diagram that includes all of the classes in your solution. This diagram may be hand drawn (i.e. on paper or a whiteboard) or generated with a drawing tool (e.g. LucidChart, Visio). You must add lines

connecting classes that are *related* to each other in some way (e.g. one class uses the other). An arrow points from the user to the class being used.

You can find more information about UML diagrams in the Resources.pdf document at myCourses.

4. Implementation

Activity 2: The Hangman Game (20 points)

Design

You should closely follow the supplied design for this homework.

Class name	Fields	Methods and Constructors
Hangman The main program for the pen and paper game.	Gameboard gameBoard	Hangman(LevelMode mode) main(String[] args) void play() boolean enterLetter(Scanner in)
GameBoard This class represents and plays the Hangman game. It is called on by the Hangman class and it uses the WordReader class.	WordReader wordReader String hiddenWord char[] maskWord Set<String> missedLetters int maxNumTries	GameBoard(LevelMode mode) void enterLetter(char guessesLetter) boolean gameOver() String toString()
WordReader This class chooses the hidden word from a dictionary file.	List<String> dict	WordReader(String filename) String pickHiddenWord() void readFile(String filename)
LevelMode This class is an enumeration for representing the difficulty level modes in the Hangman game	String dictFile int maxNumTries Properties properties static final String CONFIG static final String DEFAULT_MAX_NUM_TRIES static final String DEFAULT_DICT_FILE	void init() String getDictFile() getMaxNumTries()

For more information about these classes, their methods and fields, see the javadoc provided.

Input At Runtime

All input comes from standard input. Use the `java.util.Scanner` class to read the user's input. The user is expected to enter a letter once at a time. However, the program must ensure that the input is a valid letter. For example, if the user enters something that is not a letter, it should display a message error.

```
Word: ******

Misses:

Num. of Remaining Tries: 6

Enter a letter: 5

Invalid user input!
```

Command-line arguments

Users will be able to select the difficulty level mode when running the program from command line. Our program will offer three modes: easy (e), medium (m) and hard (h).

The main program Hangman runs on the command line as:

```
$ java Hangman mode
```

If the number of arguments is correct, the value is guarantee to be one of the three options available: 'e', 'm', or 'h'.
If the number of arguments is incorrect, display the following usage message and exit the program:

```
Usage: java Hangman mode (e – easy, m – medium, h - hard)
```

Difficulty level configuration:

- Easy
    - Number of tries: 8
    - Dictionary file: dict.txt
- Medium
    - Number of tries: 6
    - Dictionary file: dict.txt
- Hard
    - Number of tries: 6
    - Dictionary file: dict_hard.txt

Standalone Output

When executed as a standalone program, Hangman should produce output following the example pattern below.

```
$ java Hangman e

Welcome to the Hangman Game!!

Word: *****
Misses:
Num. of Remaining Tries: 8
Enter a letter: I

Secret word: *****
Misses: i
Num. of Remaining Tries: 7
Enter a Letter: A

Secret word: ***A*
Misses: i
Num. of Remaining Tries: 7
Enter a Letter: c

Secret word: ***A*
Misses: i, c
Num. of Remaining Tries: 6
Enter a Letter:
```

If the user enters a '0', the program should terminate with no further output. If the user guesses the secret word, prints "You guessed the secret word!" and the program terminates.

```
Secret word: B R E A _
Misses: i, c, f
Num. of Remaining Tries: 3
Enter a Letter: k

Secret word: B R E A K
Misses: i, c, f
Num. of Remaining Tries: 3
You guessed the secret word!
```

If the number of remaining tries reaches zero, prints "Game Over! The secret word was %s", where %s will be the secret word.

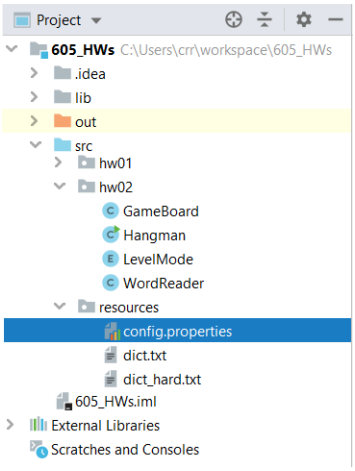
```
Secret word: B R E A _
Misses: i, c, f, l, p, m, t
Num. of Remaining Tries: 7
Enter a Letter: d
Game Over! The secret word was BREAK.
```

Activity 3: Java Properties file (7 points)

Once you have finished implementing your Hangman game, it is time to remove those constant fixed values from your program and put them into a Java properties file. A Java properties file is mainly used to store configurable parameter of a program or application. It is a file with .properties extension that store key-value pairs. See below an example of the content for this file.

```
easy.dict=/resources/dict.txt
easy.tries=8
```

Create a config.properties file and put there all the program configurable parameters. Store that properties file into a resources folder like in the image below:



For loading the file and retrieve the properties values, take a look here.

5. Grading

The breakdown for this homework is as follows:

- Explanation: 80%
- UML Class diagram: 10%
- Game functionality: 65%
- Testing: 15%
- Style: 10%

6. Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your hw2 folder and your UML class diagram as a ZIP archive named “hw2.zip” (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework).