Wordle!

This project is designed to help you practice working with strings in the context of an engaging application: the Wordle game initially developed by Josh Wardle, now available on the *New York Times* web site. Given Wordle's enormous popularity, we thought it would be fun to give you the chance to implement the game.

The starter folder

The good news is that you don't have to implement the Wordle project entirely from scratch. The Project1Starter.zip file includes the file Wordle.java, shown in Figure 1, which you should use as the starter file for all milestones in this project.

Figure 1. Starter file for Wordle

```java
/*
 * File: Wordle.java
 * ----------------
 * This module is the starter file for the Wordle assignment.
 * BE SURE TO UPDATE THIS COMMENT WHEN YOU COMPLETE THE CODE.
 */



public class Wordle {

/* Private instance variables */

   private WordleGWindow gw;


   public void run() {
      gw = new WordleGWindow();
      gw.addEnterListener((s) -> enterAction(s));
   }

/*
 * Called when the user hits the RETURN key or clicks ENTER,
 * passing in the string of characters on the current row.
 */

   public void enterAction(String s) {
      gw.showMessage("You have to implement this method.");
   }

/* Startup code */

   public static void main(String[] args) {
      new Wordle().run();
   }

}
```

In addition to the Wordle.java file, the Project1Starter.zip file includes the files WordleGWindow.java and WordleDictionary.java.

When you download the starter folder, a lot of the code is already running because we've implemented the graphics for you. Running the starter program creates a window, draws the letter boxes, and creates the keyboard at the bottom of the window. You can even type in letters either by hitting keys on the keyboard or clicking the keys on the screen, just as you can when you are playing the online version. Figure 2, for example, shows both the initial screen and the screen you get after typing in the five letters in the useful starting word RATES, which includes five of the most common letters.

Unfortunately, that's all the program does at this point. It doesn't actually let you play the Wordle game. That's your job. But first, it is worth spending a bit of time reviewing the rules for Wordle, in case you've somehow managed to miss this craze.

Figure 2. Running the starter program
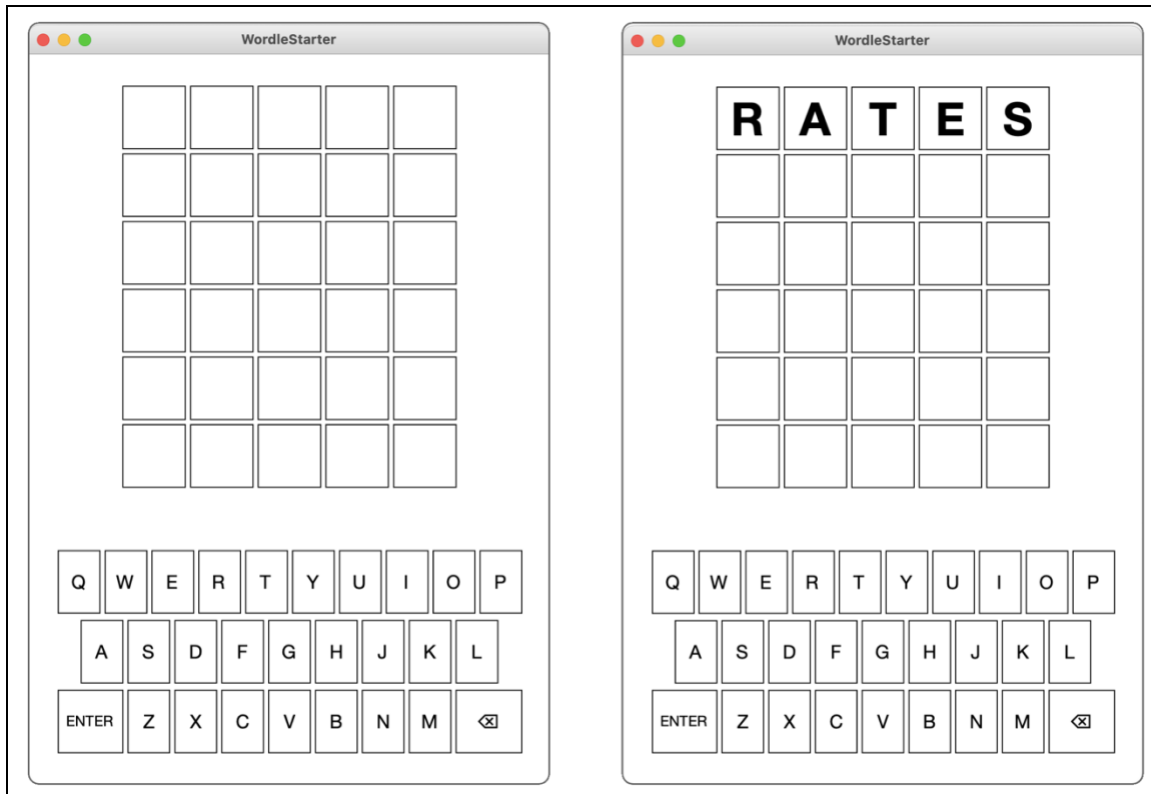
Playing Wordle

The object of the Wordle puzzle is to figure out the hidden word for the day using no more than six guesses. When you type in a word and then hit the RETURN or ENTER key, the website gives you information about how close your guess is by coloring the background of the letters. For every letter in your guess that is in its correct position, Wordle colors the background a light shade of green. For every letter that appears in the word but is not in the correct position, Wordle colors the background a brownish yellow. All letters in the guess that don't appear in the word are colored a medium gray. These colors are defined in the WordleGWindow class, which means that you can refer to them in your program using the following constants:

```
WordleGWindow.CORRECT_COLOR
WordleGWindow.PRESENT_COLOR
WordleGWindow.MISSING_COLOR
```

For example, suppose that the hidden word for the day was RELIC, and your first guess was RATES as in the Figure 2 example. The R is in the correct position, and the word contains an E, but not in the position you guessed. The hidden word does not contain any

of the letters T, E, and S. Wordle reports that information by changing the background colors of the squares like this:

Even though you know the position of the R, it doesn't make sense to guess more words beginning with R at this point because doing so gives you no new information. Suppose that you tried guessing the word LINGO, which contains five new letters, two of which appear in the word, but none of which are correctly positioned. Wordle responds by coloring the letter squares in your second guess as follows:

Putting these two clues together means that you know that the word begins with an R, contains the letters E, L, and I in some order other than the one you guessed, and that the letters A, T, S, N, G, and O do not appear anywhere in the word. These answers give you an enormous amount of information. If you think carefully about it, you might find the word RELIC, which is in fact the only English word that meets these conditions:

Done in three!

It is worth noting a few other rules and special cases. The hidden word and each of your guesses must be a real English word that is five letters long. The WordleDictionary class included with the starter package exports the constant FIVE_LETTER_WORDS as

```
public static final String[] FIVE_LETTER_WORDS = [

    "aahed", "aalii",. . . , "zoril", "zowie"

];
```

where the three dots are placeholders for more than 5000 other five-letter words. If you guess a word that it not in the word list, Wordle displays a message to that effect, at which point you can delete the letters you've entered and try again. Another rule is that you only get six guesses. If all the letters don't match by then, Wordle gives up on you and tells you what the hidden word was.

The most interesting special cases arise when the hidden word and the guesses contain multiple copies of the same letter. Suppose, for example, that the hidden word is GLASS and you for some reason guess SASSY. Wordle responds with the following colors:

The green S shows that there is an S in the fourth position, and the yellow S shows that a second S appears somewhere else in the hidden word. The S in the middle of SASSY, however, remains gray because the hidden word does not contain three instances of the letter S.

The WordleGWindow class

Even though you don't have to make any changes to it or understand the details of its operation, you need to know what capabilities the WordleGWindow class has on offer so that you can use those facilities in your code. The most important thing to know is that this library module exports a class called WordleGWindow, which implements all the graphical capabilities. The methods exported by the WordleGWindow class are outlined in Figure 3 at the top of the next page. The right column of the table gives only a brief description of what these methods do. More complete descriptions appear later in this handout in the description of the milestone that requires them.

Planning the implementation as a sequence of milestones

Whenever you are working on a programming project of any significant size, you should never try to get the entire project running all at once. A much more effective strategy is to define a series of milestones that allow you to complete the project in stages. Ideally, each milestone you choose should be a program that you can test and debug independently. Similarly, it often makes sense to defer more the more complex aspects of a project until after you have gotten the basic foundation working. The next few sections outline four milestones for the Wordle project that walk you through different stages of the implementation. You should get each one working before moving on to the next one.

Milestone #1: Pick a random word and display it in the first row of boxes

For your first milestone, all you have to do is choose a random word from the list provided in the FIVE_LETTER_WORDS constant and then have that word appear in the five boxes across the first row of the window. This milestone requires only a few lines of code, but requires you to understand what tools you have and start putting them to use. For example, the WordleGWindow class does not export any method for displaying an entire word. All you have is a method setSquareLetter that puts one letter in a box identified by its row and column numbers.

Figure 3. Methods exported by WordleGWindow class

new WordleGWindow()	Creates and displays the graphics window.
setSquareLetter(*row*, *col*, *letter*)	Sets the letter in the specified row and column.
getSquareLetter(*row*, *col*)	Returns the letter in the specified row and column.
setSquareColor(*row*, *col*, *color*)	Sets the color of the specified square.

getSquareColor(*row, col*)	Returns the color of the specified square.
setCurrentRow(*row*)	Sets the row in which typed characters appear.
getCurrentRow()	Returns the current row.
setKeyColor(*letter, color*)	Sets the color of the specified key letter.
getKeyColor(*letter*)	Returns the color of the specified key letter.
addEnterListener(*listener*)	Specifies a listener function for the ENTER key.
showMessage(*msg*)	Shows a message below the squares.

As with everything in Java, rows and columns are numbered beginning with 0, so that the top row of the window is row 0, and its column number range from 0 to 4. To avoid cluttering up your code with numbers that don't tell you much about what they mean (where does 4 come from in the previous sentence, for example?), it is best to use the constants WordleGWindow.N_ROWS and WordleGWindow.N_COLS whenever your code needs to know how many rows and columns exist. Not only does WordleGWindow.N_COLS - 1 provide more insight than the number 4, but this strategy also makes it easier to implement a SuperWordle program with longer words or a different number of guesses.

Once you finish Milestone #1, comment most of it out, because you obvioulsy won't need it for your Wordle game (with the exception of the chooseWord method). It's just to get you comfortable with the starter code.

Milestone #2: Check whether the letters entered by the user form a word

Although the starter program lets the user type letters into the Wordle game, hitting the RETURN key simply generates a message telling you that you have more to implement. The linkage between the WordleGWindow class and the main program occurs through a ***callback function,*** which is a function supplied by a client to a library, which can later call that function to execute that operation on the client's behalf. In this case, the main program makes the following call to register its interest in being notified whenever the user hits the RETURN key or clicks the ENTER button:

```
gw.addEnterListener((s) -> enterAction(s));
```

The argument in this call is an example of a Java ***arrow function,*** which is a convenient bit of syntax for a function definition in which the argument list appears to the left of the two-character arrow (->) and the body of the function appears to the right. Note that no type declarations are required here. The Java compiler simply looks at the definition of addEnterListener to determine the type of function it expects. In this case, that definition tells the compiler that addEnterListener requires a function that takes a string and returns no value. The argument (s) -> enterAction(s) matches that definition and produces a function that takes a string as its argument and then calls the enterAction method, passing along the string s. The effect of this call—mysterious as its syntax may seem at first—is to trigger a call to enterAction whenever the user hits RETURN or clicks ENTER, passing the five letters on the current row as a string.

For Milestone #2, your job is to write an implementation of enterAction that checks to see whether the word passed from WordleGWindow is a legitimate English word. If it isn't, your implementation of enterAction should call the showMessage method with the string "Not in word list", which is what the *Times* website says. If it is a word, you should display some more positive message that shows that you got this milestone running.

Milestone #3: Color the boxes

For this milestone, you need to add code to enterAction that, after checking to make sure it is a legal word, goes through and colors the boxes to show the user which letters in the guess match the word. The method you need to accomplish this task is

gw.setSquareColor(*row, col, color*)

The row and column arguments are the same as the ones you used to set or get the letters from the boxes, and *color* is the color you want to use for the background, which will typically be one of the constants CORRECT_COLOR, PRESENT_COLOR, and MISSING_COLOR imported from WordleGWindow, which have the following values, each of which is defined in terms of a six-digit hexadecimal number that gives the red, green, and blue intensities.

The hard part of this milestone is figuring out how to color the squares, which is not as easy as it might at first appear, particularly when the hidden word contains multiple copies of the same letter. To accomplish this, you will first complete the helper method

getHint(*guess, word*)

Fgure 4. Starter code for getHint()

```
/**
 * @param guess the user's guess
 * @param word the secret word to be guessed
 * @return a String version of the hint where a capital letter
 * represents a correct guess at the correct location, a lower
 * case letter represents a correct guess at the wrong location,
 * and a '*' represents an incorrect letter (neither in the
 * correct place nor a correct letter anywhere in the word)
 *
 * You will use this helper method when coloring the squares.
 * It's also the crucial method that is tested in codePost.
 *
 * Examples:
 * word      = "CLASS"
 * guess     = "SASSY"
 * returns:    "sa*S*"
 *
 * word      = "FLUFF"
 * guess     = "OFFER"
 * returns:    "*ff**"
 *
 * word      = "STACK"
 * guess     = "TASTE"
 * returns:    "tas**"
 *
 * word      = "MYTHS"
 * guess     = "HITCH"
 * returns:    "h*T**"
 *
 */




  public String getHint(String guess, String word) {
      // TODO
  }
```

Hint: It's probably best to first find the letters that are in the correct position, before looking for letters that are correct but in the wrong position.

Whenever the user enters a guess that appears in the word list, your program must do a few things. First, it must check to see whether the user has correctly guessed all five letters, in case you want to have your program display some properly congratulatory message. If not, your program must move on to the next row. This information is maintained inside the WordleGWindow class (which needs this information to know where typed letters should appear) using the setCurrentRow and getCurrentRow methods.

Important: For Milestone #3, you must complete the helper method getHint(). It's the key method used in codePost te

Milestone #4: Color the keys

Your last milestone implements a very helpful feature from the *New York Times* website in which it updates the colors of the keys on the virtual keyboard, making it easy to see what letters you've already positioned, found, or determined not to be there. The WordleGWindow class exports the methods setKeyColor and getKeyColor to accomplish this task. These methods use the same string codes as the corresponding methods for squares.

In solving this milestone, it is important to remember that once you have set the color of a key, it won't change back. If, for example, you've colored the S key green, it will never get set to yellow or gray even though you may end up using those colors for squares that contain an S.

Thoughts to keep in mind

- As with any large program, it is essential to get each milestone working before moving on to the next. It almost never works to write a large program all at once without testing the pieces as you go.
- You have to remember that uppercase and lowercase letters are different in Java. The letters displayed in the window are all uppercase but the FIVE_LETTER_WORDS constant is an array of lowercase words. At some point, your code will have to apply the necessary case conversions.

Possible extensions (optional)

There are many extensions you could add to the Wordle game. Here are a few that might be fun:

- *Create a more balanced dictionary*. If you simply choose a word at random from the dictionary, some letters will appear more frequently than others in specific positions. Josh Wardle's original implementation solved this problem by keeping two sets of words: a smaller one used to select the secret word in which frequencies are more balanced and a larger one for determining whether a guess is legal. Devise a strategy for implementing this two-tiered dictionary without having to choose words by hand.
- *Make more balanced choices from the dictionary.* There are other strategies you can use to improve the distribution of letters in the hidden words that don't require creating a separate dictionary. For example, you could make a significant improvement simply by choosing fewer hidden words that end with the letter "s", as almost 30% of the five-letter words do. To implement this strategy, you could define the constant

        FINAL_S_FRACTION = 1 / 3

  and then use that constant to accept words ending in "s" only ⅓ of the time, going back and choosing a different word the other ⅔ of the time.
- *Enhance the graphics when the user wins the game*. The setSquareColor method allows you to change the background color of a square to any color you choose. If you want to

make victories more exciting, you could animate the square colors so that the letters in the correct entry cycled through the colors of the rainbow before settling to all green.

- *Create an option that lists all possible words that are legal given the previous guesses.* Even though doing so is clearly cheating, some players would like to see a list of all the words remaining in the dictionary that would be acceptable given the previous set of guesses. You can trigger this option by having the user hit the RETURN key or click the ENTER button when the line is not yet finished, in which case some of the squares will contain the empty string. When this occurs, you can have your program go through all the words, check whether they conform to all the previous clues, and print those words out on the console.

- *Keep score in a file.* The *New York Times* Wordle site keeps track of the number of games you've played and presents a graph of the number of guesses you needed. To implement this feature, you would need to maintain a file that kept track of this information from game to game and then display it at the end. Although you could print this information to the console, you could also display the counts in the Wordle grid, so that each row shows the number of times you needed that many guesses. Thus, if you had solved four Wordle problems in three guesses, eleven in four guesses, and six in five guesses, your Wordle program might show the following display at the end: