Project 6: Refactoring a Video Game

Due: Monday, Oct. 12 at 11:59 PM

Description: In this project, we will work with an existing program that runs a word scramble video game! When the game is played, the computer selects a random word from a text file of possible words. The word is scrambled and printed to the console, and the user attempts to guess the unscrambled word.

The user is given one fewer guesses than there are letters in the word. For every incorrect guess, part of the unscrambled word is shown. If the first guess is wrong, the first letter of the word is shown. If the second guess is wrong, the first and second letters are shown, and so on. The user earns 1 point for each letter not shown when the word is guessed correctly. No point is awarded for the last letter, however, since it can be determined by process of elimination.

All the code that runs the game is written in the main method. This makes the program difficult to understand. Your goal is to improve the organization by transferring four sections of code (specified below) into methods and then replacing the sections with calls to these methods. Restructuring code like this is a common task for programmers and is known as "refactoring."

Objectives: Your program will be graded according to the rubric below. Please review each objective before submitting your work so you don't lose points.

1. (15 points) Transfer lines 42 to 64 into the body of a method described in the next section. Then replace these lines with code that calls the method. Follow these guidelines:
   - Define the method in the class but not inside the main method.
   - Choose an appropriate signature that includes the correct return type and all necessary parameters (but no extraneous parameters).
   - If the return type of the method is not void, make sure the body of the method has a return statement. Think carefully about which variable should be returned.
   - Replace only the specified lines in the main method with a call to the new method.
   - Pass the method the correct arguments in the main method.
   - If the method returns a value, assign it to an appropriate variable in the main method.
2. (25 points) Transfer lines 66 to 83 into the body of a method described in the next section. Replace these lines with code that calls the method. Follow the objective 1 guidelines.
3. (25 points) Transfer lines 85 to 107 into the body of a method described in the next section. Replace these lines with code that calls the method. Follow the objective 1 guidelines.
4. (25 points) Transfer lines 114 to 124 into the body of a method described in the next section. Replace these lines with code that calls the method. Follow the objective 1 guidelines.
5. (10 points) Your program should compile successfully after all of the methods are written.

Method Descriptions: Below is a list of the four sections of code that need to be transferred into methods. Each section is specified by a range of line numbers.

Below each range of line numbers is a list of information about the corresponding method. This information includes a description of what the method does, a list of parameters, the return type, the method signature, and a list of local variables. All this information is included for the first method, but you must figure out the parameters, return type, signature, and local variables for the other three methods. Consider writing this information on a piece of paper before trying to implement the methods in Eclipse.

1. Lines 42 to 64
   - Description: The method returns a random word from the dictionary text file.
   - Parameters: none
   - Return type: String
   - Method signature: public static String chooseRandomWord()
   - Local variables: Scanner dictionary, int numWords, int numSkip, int i, String word
   - Additional notes: The statement "throws FileNotFoundException" must appear after the signature because the method reads a file. See below for more information.
2. Lines 66 to 83
   - Description: The method takes a String and returns a copy with the letters scrambled (i.e., in random order).
   - Parameters:
   - Return type:
   - Method signature:
   - Local variables:
3. Lines 85 to 107
   - Description: The method reads input from the keyboard and returns the number of guesses the user required to correctly guess the unscrambled word. After each incorrect guess, the method prints one additional letter of the unscrambled word.
   - Parameters:
   - Return type:
   - Method signature:
   - Local variables:
4. Lines 114 to 124
   - Description: The method prints information to the console to inform the user how the game ended. If the user failed to guess the word, the unscrambled word is shown. If the user guessed correctly, the number of points earned is shown.
   - Parameters:
   - Return type:
   - Method signature:
   - Local variables:

While figuring out this information for methods 2 through 4, keep the following in mind:

- Parameters store the input that each method gets from the main method.
- Local variables are the details hidden inside the method. You want to hide as many details as possible, so anything that can be a local variable should be.
- Return type is the data type (e.g., int or boolean) of the value returned by the method.
- Name each method whatever you like, but choose something descriptive.

Importing Code into Eclipse: Before you can edit the video game code, you need to import it into Eclipse. Here are instructions on how to manually do this.

1. Create a new project in Eclipse, just like you have done for previous projects.
2. Do not create a new class. Instead, open your file explorer (in Windows or macOS) and move the file Project6.java into the src folder located inside the new project folder.
3. Move the text file dictionary.txt into the project folder. This is the folder that contains the src and bin folders.
4. In Eclipse, right-click the new project in the Project Explorer and select "Refresh" from the menu. If you followed steps 2 and 3, the files dictionary.txt and Project6.java should appear. (You may need to click the arrows to show the contents of the project folders.)
5. Click the run button to build and execute the program.

Additional notes:

- The instructions above can be used to import any .java file into Eclipse. For instance, if you are a student in CS 1324, you can import a copy of code you wrote on your lab partner's laptop. We will use these instructions again in future projects.

Implementation Suggestions:

1. After you import the code, play the game a few times to get a feel for how it works. This will also ensure that you imported the files correctly.
2. After you play the game, read the code carefully. Reading someone else's code is different than reading your own code. Think about what part of the game is implemented by each segment of code.
3. As you transfer each section of code out of the main method, you will change the line numbers of the surrounding lines of code. In order to keep track of the remaining sections, it's helpful to keep an unedited copy of the code. If you are a student in CS 1324, a good idea would be to keep the unedited copy open on the laptop that you and your partner are not using to change the code. Another good idea is to put the original line numbers from this document into the code in comments.
4. After you transfer each section and replace it with a method call, run the program to check that you haven't introduced any errors.

Reading Files: This is the first program we have seen that reads information from a text file. Fortunately, files can be read with Scanner objects using the same methods we have used to read keyboard input. The only difference is that the Scanner must be constructed with a reference to the file, rather than System.in.

Whenever a file is opened, read, or modified in a program, there is a chance that an error could occur (e.g., if the file is in use by another program). For this reason, any method that opens a file with a Scanner (whether directly or by calling another method) needs the statement "throws FileNotFoundException" to appear after its signature. (For an example, see the signature of the main method of the program.) You will need to include this statement when you write the method that reads a random word from the file dictionary.txt. Note that this exception requires an import statement, which has already been added to the program.

StringBuilder Class: The String class is designed for words that do not change. Adding and deleting characters in words are changes. There is another class that is designed to handle words that change; it is called "StringBuilder."

You do not need to know anything about the StringBuilder class in order to complete this project. The code that is given uses it, but it will be hidden inside a method. You do not need to modify any code that uses a StringBuilder object.

Method Refactoring Example: If you are stuck trying to write the first method, you may find this example to be helpful. Consider the following main method, which contains code to reverse the first and last names in a string, where a space is used as a separator:

```
public static void main(String[] args)
{
    Scanner input = new Scanner(System.in);
    System.out.println("Enter a name in the format 'first last'");
    String name = input.nextLine();

    int space = name.indexOf(' ');
    String first = name.substring(0, space);
    String last = name.substring(space + 1);
    String result = last + " " + first;

    System.out.println("The reversed name is " + result);
}
```

Suppose we want to transfer the highlighted lines into a separate method. The method will take a string that contains a first and last name and output a second string with the order of the names reversed. Below is the method:

```
public static String reverseNames(String name)
{
    int space = name.indexOf(' ');
    String first = name.substring(0, space);
    String last = name.substring(space + 1);
    String result = last + " " + first;
```

```
        return result;
    }
```

Note the following:

1. The variables space, first, last, and result, which used to be local variables of the main method, are now local variables of the method reverseNames.
2. The variable name must be a parameter because it is assigned a value on a line that is not included in the method.
3. The variable result must be returned by the method because the value is used on a line that is not included in the method.

Replacing the highlighted lines with a call to the new method results in the following, simplified main method:

```
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a name in the format 'first last'");
        String name = input.nextLine();

        String result = reverseNames(name);

        System.out.println("The reversed name is " + result);
    }
```

Submission Instructions: Submit your source code to Zylabs.