**Assignment 2**

**Due, Sunday, June 4, 2017 for maximum 100**

**Monday, June 5, 2017 for maximum 90**

**Tuesday, June 6, 2017 for maximum 80**

**Wednesday, June 7, 2017 for maximum 70**

**Assignment Scope**

1. Adding member variables or instance variable in a class
2. Generating getters/setters for member variables
3. Instantiating an instance of a class
4. Calling a method of a class using the reference object
5. Using Java API class **ArrayList**
6. Reading a data file
   1. Using a try/catch/finally block to handle exceptions and close resource files
      1. IOException
      2. URISyntaxException
   2. Using Java API class **Scanner**
   3. Using Java API class **URL**
   4. Using Java API class **File**
7. Saving data from a data file in an **ArrayList**
8. Adding method signatures to an interface
9. Adding a constant to an interface
10. Implementing an interface
11. Using the enhanced **for** loop

**Deliverables**

To complete this assignment you must submit your **compressed Netbeans project** to Webcourses.

**Tasks and Rubric**

|  |  |
| --- | --- |
| Activity | |
| Boggle project |  |
| Boggle class | 1. Create member variables of type:    1. **ArrayList** of class **String** // stores data from data file with dice data    2. **String** // set to the name of input file “BoggleData.txt”    3. **ArrayList** of class **String** // stores data from data file with dictionary data    4. **String** // set to the name of input file “Dictionary.txt” |
|  | 1. Method main() should:    1. Instantiate an instance of class **ReadDataFile** passing the file name variable for file BoggleData.txt as an argument    2. Call method **populateData** on the above object    3. Instantiate an instance of class **ReadDataFile** passing the file name variable for file Dictionary.txt as an argument    4. Call method **populateData** on the above object    5. Instantiate an instance of class **Board** passing as arguments method call **getData** on each of the two **ReadDataFile** object references from above    6. Call method **populateDice** on object reference of class **Board**    7. Output to the IDE output window the number of objects in the **ArrayList** of type **String** that stores the dictionary data |
| core package |  |
| IBoard interface | 1. Add constant fields:    1. **NUMBER\_OF\_DICE** equal to value 16    2. **GRID** equal to value 4 2. Add method signatures:    1. **populateDice** with return type void and an empty parameter list    2. **shakeDice** with return type **ArrayList** and an empty parameter list |
| Board class | 1. Update the class so that it implements interface **IBoard**    1. TIP: Use Netbeans right click menu, Insert Code, Implement Method to have the IDE generate the methods for you; you will replace the throw exception statements with the source code you write 2. Add member variable of type:    1. **ArrayList** of class **String** // stores dice data    2. **ArrayList** of class **String** // stores dictionary data    3. **ArrayList** of class **Die** // stores 16 game dice 3. Add a custom constructor with two parameters of type **ArrayList** of class **String**; it should do the following    1. Set the member variable of type **ArrayList** of class **String** that stores the Boggle data equal to the associated parameter in the method signature    2. Set the member variable of type **ArrayList** of class **String** that stores the dictionary data equal to the associated parameter in the method signature    3. Instantiate the member variable of type **ArrayList** of class **Die**. 4. Implement method **populateDice**; the method should do the following:    1. Declare a variable of type class **Die**    2. Declare and initialize a variable of type **int** to serve as a counter to access the data in the member variable of type **ArrayList** of type **String** storing the Boggle data    3. Loop through the 16 dice (use the constant **NUMBER\_OF\_DICE** as your terminating condition):       1. Instantiate the instance of class **Die** using the default no-argument constructor       2. For each **Die** instance, loop through the six sides of the die (use the constant **NUMBER\_OF\_SIDES** as your terminating condition):          1. Add each of the 6 letters to the **Die** **ArrayList** representing the die letters by calling method **addLetter** in class **Die**       3. Display the letters of each die by calling method **displayLetters**() in class **Die** on a separate row 5. Add each die instance to the **ArrayList** declared specifically for class **Die** |
| IDie interface | 1. Add constant field:    1. **NUMBER\_OF\_SIDES** equal to value 6 2. Add method signatures:    1. **rollDie** with return type String and an empty parameter list    2. **addLetter** with a return type of void and one parameter of type String representing one of the six letters on the die    3. **displayLetters** with a return type of void and an empty parameter list |
| Die class | 1. Update the class so that it implements interface **IDie**    1. TIP: Use Netbeans right click menu, Insert Code, Implement Method to have the IDE generate the methods for you; you will replace the throw exception statements with the source code you write 2. Add member variable of type:    1. **ArrayList** // stores dice data for the sides of the die    2. TIP: member variables should have an access level modifier of **private** to protect the data 3. Implement method **addLetter;** the method should:    1. Add the parameter to the **ArrayList** representing the letters on the die 4. Implement method **displayAllLetters**; the method should:    1. Use an enhanced **for** loop to output the letter on each of the six sides of the die |
| inputOutput package |  |
| IReadDataFile interface | Create interface IReadDataFile |
|  | 1. Add method signature:    1. **populateData** with no return type and an empty parameter list |
| ReadDataFile class | 1. Update the class so that it implements interface **IReadDataFile**    1. TIP: Use Netbeans right click menu, Insert Code, Implement Method to have the IDE generate the methods for you; you will replace the throw exception statements with the source code you write 2. Add member variables using the specified data types:    1. **Scanner** // for reading the file    2. **String** // for storing the file name    3. **ArrayList** of class **String** // for storing the data from the file    4. TIP: member variables should have an access level modifier of **private** to protect the data 3. Add a custom constructor that receives one parameter of type **String** representing the name of the data file to read; it should do the following:    1. Set the member variable of type **String** for storing the file name equal to the parameter    2. Instantiate the member variable of type **ArrayList** 4. Add a getter for the **ArrayList** member variable that stores the data read from the data file    1. TIP: Use the IDE right click menu, Refactor, Encapsulate Fields and select just getter for the member variable of focus 5. Implement method **populateData**; it should do the following:    1. Instantiate an instance of Java API class **URL** passing as an argument member variable representing the file name of the data file       1. TIP: this is a unique implementation, to instantiate the instance set the **URL** variable equal to static method call **getClass().getResource()**    2. Instantiate an instance of class **File** using the **URL** created above    3. Initialize member variable of type **Scanner** based on the **File** instance created above       1. TIP: pass as an argument to the constructor the reference object of the **URL** instance with method call **.toURI()**    4. Loop through the data file until the end       1. Add to the **ArrayList** representing the data in the file each value read from the data file |
| userInterface package |  |
| Boggle application |  |
| Test Case 1 | Test Case 1 passes |
| Test Case 2 | Test Case 2 passes |
| Test Case 3 | Test Case 3 passes |
|  | Source compiles with no errors |
|  | Source runs with no errors |
|  | Source includes comments |
| Total |  |

**Perform the following test cases**

|  |  |  |
| --- | --- | --- |
| Test Cases | | |
|  | **Action** | **Expected outcome** |
| Test Case 1 | **Project view** | Completed project view should look like figure 1 |
| Test case 2 | **Run application** | **displayLetters** should look like figure 2 |
| Test case 3 | **Run application** | Size of **ArrayList** storing dictionary data should be similar to figure 3 |

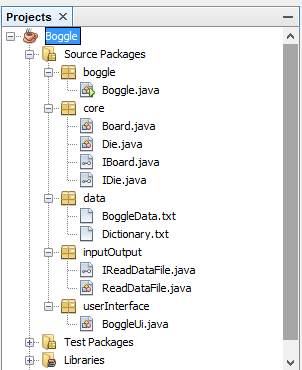
****

Figure 1 Project View

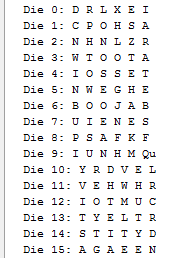


Figure 2 Output from method displayLetters



Figure 3 Output for number of entries in the Dictionary.txt file