## **Description**

Have you ever encountered an innovative and exciting product that is surprisingly simple? Have you wondered how complicated it would be to create something similar yourself?

For this Project, you will be writing five classes: InvalidWordException, InvalidGuessException, WordLibrary, WordGuesser, and WordGame. These classes will allow a user to play a word guessing puzzle game using the command line interface.

## **Instructions**

The implementation instructions are documented below. Functionality and naming descriptions are exact. That is, if a method performs some function, your program must implement that functionality within the method. If you implement it elsewhere, such as in the main method or a different class, you will not receive credit. All names must match the handout to be scored.

Note: This assignment uses arrays for several requirements. You cannot substitute an ArrayList for an array.

Note: You are not permitted to use the static modifier on any field unless you are directed to do so in the assignment instructions. This rule will be consistent for this and every other CS 18000 assignment. If you do make a field static, it will lead to unexpected behavior in your solution and will lead to failed test cases.

Note: Unless we direct you to throw an Exception (in which case, you should throw it to the calling method for handling), handle all other Exceptions in the methods within which they might occur.

### 

### **InvalidWordException**

Note: InvalidWordException extends Exception.

#### **Constructor**

| **Parameters** | **Modifier** | **Description** |
| --- | --- | --- |
| String message | public | Calls super() with the message parameter. |

### **InvalidGuessException**

Note: InvalidGuessException extends Exception.

#### **Constructor**

| **Parameters** | **Modifier** | **Description** |
| --- | --- | --- |
| String message | public | Calls super() with the message parameter. |

### **WordLibrary**

This class contains the library of words used in the puzzle. We provide fields, constructor, and several methods as starter code. Modifications to provided code may result in unexpected behavior and a reduced score.

#### **Fields**

| **Name** | **Type** | **Modifiers** | **Description** |
| --- | --- | --- | --- |
| library | String[] | private | A String array containing individual words for the library. |
| seed | int | private | The seed value used by the random number generator. |
| random | Random | private | The random number generator used to select a word during gameplay. |
| fileName | String | private | The name of the input file used to generate the library. |

#### **Constructor**

| **Parameters** | **Modifier** | **Description** |
| --- | --- | --- |
| String fileName | public | Read the input file to set the values for the library and seed. The input file format is specified in the next section.  Instantiate the random field using the seed value.  Call the processLibrary method to review the words in the library.  Note: The constructor should handle an exception thrown from processLibrary by printing the stack trace. |

#### 

#### 

#### 

#### 

#### 

#### **Methods**

| **Name** | **Return Type** | **Parameters** | **Modifier** | **Description** |
| --- | --- | --- | --- | --- |
| getLibrary | String[] | None | public | Returns the library. |
| getSeed | int | None | public | Returns the seed. |
| setLibrary | void | String[] library | public | Sets the library. |
| setSeed | void | int seed | public | Sets the seed. |
| verifyWord | void | String word | public | Review the word argument to verify that it has exactly 5 letters.  If a word does not meet requirements, throw a new InvalidWordException with the message "Invalid word!" |
| chooseWord | String | None | public | Returns the library word at an index randomly chosen using the random field with the bound set to the length of the library. |
| processLibrary | void | None | public | Reviews the library field, calling verifyWord on each word present. If no Exception is thrown, keep the word in the library. If an Exception is thrown, remove that word from the library and print the message to the terminal.  The updated array should contain only the remaining words in the library, in the same order as listed previously, without any empty indexes. |

The only methods you must complete in this class are verifyWordand processLibrary. The rest are included as Starter Code. Be sure to add documentation as needed.

#### **Input File Format**

The input file will have the following format:

Seed: [Any Positive Integer]

First\_Word

Second\_Word

Third\_Word

...

Last\_Word

For example:

Seed: 32

apple

anger

break

while

false

union

### **WordGuesser**

This class contains the fields and methods that will support playing the game.

#### **Fields**

| **Name** | **Type** | **Modifiers** | **Description** |
| --- | --- | --- | --- |
| playingField | String[][] | private | A 5x5 String grid that stores the game field. |
| round | int | private | A counter variable that tracks the current round. |
| solution | String | private | The word the user tries to guess. |

#### **Constructor**

| **Parameters** | **Modifier** | **Description** |
| --- | --- | --- |
| String solution | public | Instantiate the class field to the associated parameter.  Initialize the round field to 1.  Instantiate the playingField field to a new String[5][5] array and initialize each position to an empty space " ". |

#### **Methods**

| **Name** | **Return Type** | **Parameters** | **Modifier** | **Description** |
| --- | --- | --- | --- | --- |
| getPlayingField | String[][] | None | public | Returns the playing field. |
| getRound | int | None | public | Returns the round. |
| getSolution | String | None | public | Returns the solution. |
| setPlayingField | void | String[][] playingField | public | Sets the playing field. |
| setRound | void | int round | public | Sets the round. |
| setSolution | void | String solution |  | Sets the solution. |
| checkGuess | boolean | String guess | public | Determines if the user's guess is the same as the solution and updates the playingField.  Returns true if the guess is the same as the solution, false otherwise.  Updates the playingField by adding each character of the guess to the row corresponding to the current round. Use the following characters to tell the user how they did:   * '' surrounding a character that is in the solution and is in the correct position. Note: This is a single quote on either side. * \*\* surrounding a character that is in the solution but is not in the correct position. * {} surrounding a character that is not in the solution.   For example, if the guess is "anger" and the solution is "apple", the word would be added to the playing field as the following:  'a' | {n} | {g} | \*e\* | {r}  Additionally, if the user's guess is not exactly five characters, throw a new InvalidGuessException with the message "Invalid Guess!".  Note: The method that calls checkGuess should handle the exception by printing the error message to the terminal. The game should continue afterwards.  Note: Remember, arrays are 0-indexed. For round 1, you'll be modifying the row at index 0. Moreover, be sure to add each character by index to the correct location in the playingField. The first character in the first guess would be in [0][0], the second character in the first guess would go to [0][1], and so on. |
| printField | void | None | public | Print the playingField in a grid, with each character separated by the following pattern: [space][vertical bar][space], as depicted by " | ".  Do not print a vertical bar after the last character in the string. Examples are included in the checkGuess method description and sample output section. |

### **WordGame**

This class includes a main method and menu interface that lets the user play the game.

**Read this first**

Starter code is provided for this class. You will notice that we declare and initialize a Scanner in the main method. This is the only Scanner you are permitted to use for the assignment. If you create additional methods, you must pass this scanner as an argument. If you make a global scanner, declare a new scanner in a different method, or take some other action to change how Scanners are declared or instantiated in your program, you will fail the test cases. This format is required for every assignment in CS 18000.

#### **Menu**

The menu allows the user to play the game. The user will provide a word library and seed value for the random number generator via an input file. You must use this information to instantiate a WordLibrary object for the game.

When the user selects to play a new game, create a WordGuesser object with a word parameter from the library's chooseWord() method. You must select the word in this manner to ensure your output will match the test case expectations.

After everything is initialized, the user will play the game. They have five guesses to identify the correct word. The playingField should print updated results after every round. The user can decide to quit at any time, or play through to the end. They may get the correct answer or fail to find the word, with the game printing messages accordingly.

After a game is completed, the user can decide to play again.

Each output prompt is included in the starter code! You are free to implement your solution in any manner you like, provided you follow the guidelines described above and match the output below.

In addition to the requirements described above, one method is required:

| **Name** | **Return Type** | **Parameters** | **Modifier** | **Description** |
| --- | --- | --- | --- | --- |
| updateGameLog | void | String solution, String[] guesses, boolean solved | public static | This method generates a game log file to track the user's performance in the game between application runs. That is, a user will be able to create a long-term record of their gameplay that persists even when the application is closed.  The log is always named "gamelog.txt". **Games Completed** The file's first line is always "Games Completed: " followed by the total number of rounds the player has completed. The game will need to read this line before the first round so it can be updated to the new count when the player is finished.  For example, if the first line is "Games Completed: 15" and the player completes another five games, the new file will have "Games Completed: 20". **Game Data** The remaining lines in the file are logs of each round. The following data is required:  Game [Game Number]  - Solution: [Solution]  - Guesses: [Comma-separated list of guesses]  - Solved: [Yes/No]  For example:  Game 3  - Solution: break  - Guesses: apple,while,false,email,error  - Solved: No  New game data will be appended to the existing file. If a file does not initially exist, create one.  The game log file will be generated automatically with no input from the user. Update it after every completed game. |

We include several sample games in the output examples below to illustrate how the game will run.

**Before Sample 1 - No gamelog.txt file exists**

##### **Sample 1**

Please enter a filename

[input.txt]

Ready to play?

1.Yes

2.No

[1]

Current Round: 1

| | | |

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[break]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 2

{b} | {r} | {e} | {a} | {k}

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[float]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 3

{b} | {r} | {e} | {a} | {k}

{f} | {l} | \*o\* | {a} | {t}

| | | |

| | | |

| | | |

Please enter a guess!

[short]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 4

{b} | {r} | {e} | {a} | {k}

{f} | {l} | \*o\* | {a} | {t}

{s} | {h} | \*o\* | {r} | {t}

| | | |

| | | |

Please enter a guess!

[throw]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 5

{b} | {r} | {e} | {a} | {k}

{f} | {l} | \*o\* | {a} | {t}

{s} | {h} | \*o\* | {r} | {t}

{t} | {h} | {r} | 'o' | {w}

| | | |

Please enter a guess!

[union]

You got the answer!

{b} | {r} | {e} | {a} | {k}

{f} | {l} | \*o\* | {a} | {t}

{s} | {h} | \*o\* | {r} | {t}

{t} | {h} | {r} | 'o' | {w}

'u' | 'n' | 'i' | 'o' | 'n'

Ready to play?

1.Yes

2.No

[2]

Maybe next time!

**After Sample 1, but before Sample 2 gamelog.txt file contents:**

Games Completed: 1

Game 1

- Solution: union

- Guesses: break,float,short,throw,union

- Solved: Yes

##### **Sample 2**

Note: In this sample, we use input2.txt. This file has two invalid words.

Please enter a filename

[input2.txt]

Invalid word!

Invalid word!

Ready to play?

1.Yes

2.No

[1]

Current Round: 1

| | | |

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[float]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 2

{f} | 'l' | {o} | \*a\* | {t}

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[class]

You got the answer!

{f} | 'l' | {o} | \*a\* | {t}

'c' | 'l' | 'a' | 's' | 's'

| | | |

| | | |

| | | |

Ready to play?

1.Yes

2.No

[1]

Current Round: 1

| | | |

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[break]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 2

{b} | \*r\* | \*e\* | {a} | {k}

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[super]

You got the answer!

{b} | \*r\* | \*e\* | {a} | {k}

's' | 'u' | 'p' | 'e' | 'r'

| | | |

| | | |

| | | |

Ready to play?

1.Yes

2.No

[2]

Maybe next time!

**After Sample 2 gamelog.txt file contents:**

Games Completed: 3

Game 1

- Solution: union

- Guesses: break,float,short,throw,union

- Solved: Yes

Game 2

- Solution: class

- Guesses: float,class

- Solved: Yes

Game 3

- Solution: super

- Guesses: break,super

- Solved: Yes

The remaining samples omit the gamelog contents for brevity, you may assume that they follow the same format described in the first two scenarios.

##### **Sample 3**

Please enter a filename

[input.txt]

Ready to play?

1.Yes

2.No

[1]

Current Round: 1

| | | |

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[catch]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[2]

{c} | {a} | {t} | {c} | {h}

| | | |

| | | |

| | | |

| | | |

Ready to play?

1.Yes

2.No

[2]

Maybe next time!

##### **Sample 4**

Please enter a filename

[input.txt]

Ready to play?

1.Yes

2.No

[2]

Maybe next time!

##### **Sample 5**

Please enter a filename

[input.txt]

Ready to play?

1.Yes

2.No

[1]

Current Round: 1

| | | |

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[break]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 2

{b} | {r} | {e} | {a} | {k}

| | | |

| | | |

| | | |

| | | |

Please enter a guess!

[catch]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 3

{b} | {r} | {e} | {a} | {k}

{c} | {a} | {t} | {c} | {h}

| | | |

| | | |

| | | |

Please enter a guess!

[class]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 4

{b} | {r} | {e} | {a} | {k}

{c} | {a} | {t} | {c} | {h}

{c} | {l} | {a} | {s} | {s}

| | | |

| | | |

Please enter a guess!

[super]

That's not it!

Would you like to make another guess?

1.Yes

2.No

[1]

Current Round: 5

{b} | {r} | {e} | {a} | {k}

{c} | {a} | {t} | {c} | {h}

{c} | {l} | {a} | {s} | {s}

{s} | \*u\* | {p} | {e} | {r}

| | | |

Please enter a guess!

[short]

You ran out of guesses!

Solution: union

{b} | {r} | {e} | {a} | {k}

{c} | {a} | {t} | {c} | {h}

{c} | {l} | {a} | {s} | {s}

{s} | \*u\* | {p} | {e} | {r}

{s} | {h} | \*o\* | {r} | {t}

Ready to play?

1.Yes

2.No

[2]

Maybe next time!

##### **Sample 6**

Note: In this sample, we use input2.txt. This file has two invalid words.

Please enter a filename

[input2.txt]

Invalid word!

Invalid word!

Ready to play?

1.Yes

2.No

[2]

Maybe next time!

Note: Brackets [] indicate input.

## **Testing**

We have included a program that will allow you to create and run output tests automatically in the Starter Code. This will make it easier for you to verify that each possible progression through your solution is correct. Take a look at **RunLocalTest.java**. There are many utility features and tools that you do not need to worry about at the moment, instead, focus on the test case. It is included in the **Starter Code**. Read through it.

### **Public Test Cases Note**

You should think of other test cases to use that will fully test every aspect of every feature of your program.

**RunLocalTest:**

import org.junit.Test;

import org.junit.After;

import java.lang.reflect.Field;

import java.lang.reflect.Constructor;

import java.lang.reflect.Method;

import java.lang.reflect.Modifier;

import org.junit.Assert;

import org.junit.Before;

import org.junit.rules.Timeout;

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

import javax.swing.\*;

import java.io.\*;

import java.util.Random;

import java.lang.reflect.\*;

import java.util.ArrayList;

import java.util.concurrent.ThreadLocalRandom;

import java.lang.reflect.InvocationTargetException;

import java.util.UUID;

import java.math.BigInteger;

import static org.junit.Assert.\*;

public class RunLocalTest {

public static void main(String[] args) {

Result result = JUnitCore.runClasses(TestCase.class);

if (result.wasSuccessful()) {

System.out.println("Excellent - Test ran successfully");

} else {

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

}

}

public static class TestCase {

private final PrintStream originalOutput = System.out;

private final InputStream originalSysin = System.in;

@SuppressWarnings("FieldCanBeLocal")

private ByteArrayInputStream testIn;

@SuppressWarnings("FieldCanBeLocal")

private ByteArrayOutputStream testOut;

@Before

public void outputStart() {

testOut = new ByteArrayOutputStream();

System.setOut(new PrintStream(testOut));

}

@After

public void restoreInputAndOutput() {

System.setIn(originalSysin);

System.setOut(originalOutput);

}

private String getOutput() {

return testOut.toString();

}

@SuppressWarnings("SameParameterValue")

private void receiveInput(String str) {

testIn = new ByteArrayInputStream(str.getBytes());

System.setIn(testIn);

}

@Test(timeout = 1000)

public void invalidGuessExceptionDeclarationTest() {

Class<?> clazz;

int modifiers;

Class<?> superclass;

Class<?>[] superinterfaces;

clazz = InvalidGuessException.class;

modifiers = clazz.getModifiers();

superclass = clazz.getSuperclass();

superinterfaces = clazz.getInterfaces();

Assert.assertTrue("Ensure that `InvalidGuessException` is `public`!", Modifier.isPublic(modifiers));

Assert.assertFalse("Ensure that `InvalidGuessException` is NOT `abstract`!", Modifier.isAbstract(modifiers));

Assert.assertEquals("Ensure that `InvalidGuessException` extends `Exception`!", Exception.class, superclass);

Assert.assertEquals("Ensure that `InvalidGuessException` implements no interfaces!", 0, superinterfaces.length);

}

@Test(timeout = 1000)

public void invalidWordExceptionDeclarationTest() {

Class<?> clazz;

int modifiers;

Class<?> superclass;

Class<?>[] superinterfaces;

clazz = InvalidWordException.class;

modifiers = clazz.getModifiers();

superclass = clazz.getSuperclass();

superinterfaces = clazz.getInterfaces();

Assert.assertTrue("Ensure that `InvalidWordException` is `public`!", Modifier.isPublic(modifiers));

Assert.assertFalse("Ensure that `InvalidWordException` is NOT `abstract`!", Modifier.isAbstract(modifiers));

Assert.assertEquals("Ensure that `InvalidWordException` extends `Exception`!", Exception.class, superclass);

Assert.assertEquals("Ensure that `InvalidWordException` implements no interfaces!", 0, superinterfaces.length);

}

@Test(timeout = 1000)

public void wordGameClassDeclarationTest() {

Class<?> clazz;

int modifiers;

Class<?> superclass;

Class<?>[] superinterfaces;

clazz = WordGame.class;

modifiers = clazz.getModifiers();

superclass = clazz.getSuperclass();

superinterfaces = clazz.getInterfaces();

Assert.assertTrue("Ensure that `WordGame` is `public`!", Modifier.isPublic(modifiers));

Assert.assertFalse("Ensure that `WordGame` is NOT `abstract`!", Modifier.isAbstract(modifiers));

Assert.assertEquals("Ensure that `WordGame` extends `Object`!", Object.class, superclass);

Assert.assertEquals("Ensure that `WordGame` implements no interfaces!", 0, superinterfaces.length);

}

@Test(timeout = 1000)

public void wordGuesserClassDeclarationTest() {

Class<?> clazz;

int modifiers;

Class<?> superclass;

Class<?>[] superinterfaces;

clazz = WordGuesser.class;

modifiers = clazz.getModifiers();

superclass = clazz.getSuperclass();

superinterfaces = clazz.getInterfaces();

Assert.assertTrue("Ensure that `WordGuesser` is `public`!", Modifier.isPublic(modifiers));

Assert.assertFalse("Ensure that `WordGuesser` is NOT `abstract`!", Modifier.isAbstract(modifiers));

Assert.assertEquals("Ensure that `WordGuesser` extends `Object`!", Object.class, superclass);

Assert.assertEquals("Ensure that `WordGuesser` implements no interfaces!", 0, superinterfaces.length);

}

@Test(timeout = 1000)

public void profileClassDeclarationTest() {

Class<?> clazz;

int modifiers;

Class<?> superclass;

Class<?>[] superinterfaces;

clazz = WordLibrary.class;

modifiers = clazz.getModifiers();

superclass = clazz.getSuperclass();

superinterfaces = clazz.getInterfaces();

Assert.assertTrue("Ensure that `WordLibrary` is `public`!", Modifier.isPublic(modifiers));

Assert.assertFalse("Ensure that `WordLibrary` is NOT `abstract`!", Modifier.isAbstract(modifiers));

Assert.assertEquals("Ensure that `WordLibrary` extends `Object`!", Object.class, superclass);

Assert.assertEquals("Ensure that `WordLibrary` implements no interfaces!", 0, superinterfaces.length);

}

@Test(timeout = 1000)

public void testExpectedOutput() {

// Set the input

String input = "input.txt" + System.lineSeparator() +

"2" + System.lineSeparator();

// Pair the input with the expected result

String expected =

"Please enter a filename" +

System.lineSeparator() +

"Ready to play?" +

System.lineSeparator() +

"1.Yes" +

System.lineSeparator() +

"2.No" +

System.lineSeparator() +

"Maybe next time!" +

System.lineSeparator();

// Runs the program with the input values

receiveInput(input);

WordGame.main(new String[0]);

// Retrieves the output from the program

String output = getOutput();

// Trims the output and verifies it is correct.

output = output.replace("\r\n", "\n");

assertEquals("Make sure your output matches the expected format",

expected.trim(), output.trim());

}

}

}

**WordGame.java:**

import java.util.Scanner;

public class WordGame {

public static String welcome = "Ready to play?";

public static String yesNo = "1.Yes\n2.No";

public static String noPlay = "Maybe next time!";

public static String currentRoundLabel = "Current Round: ";

public static String enterGuess = "Please enter a guess!";

public static String winner = "You got the answer!";

public static String outOfGuesses = "You ran out of guesses!";

public static String solutionLabel = "Solution: ";

public static String incorrect = "That's not it!";

public static String keepPlaying = "Would you like to make another guess?";

public static String fileNameInput = "Please enter a filename";

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

//TODO: Implement your solution!

}

}

**WordLibrary.java:**

import java.util.Random;

public class WordLibrary {

private String[] library;

private int seed;

private Random random;

public WordLibrary(String fileName) {

//TODO: Implement this constructor!

}

public void verifyWord(String word) {

//TODO: Implement this method!

}

public void processLibrary() {

//TODO: Implement this method!

}

public String chooseWord() {

return library[random.nextInt(library.length)];

}

public String[] getLibrary() {

return library;

}

public void setLibrary(String[] library) {

this.library = library;

}

public int getSeed() {

return seed;

}

public void setSeed(int seed) {

this.seed = seed;

}

}