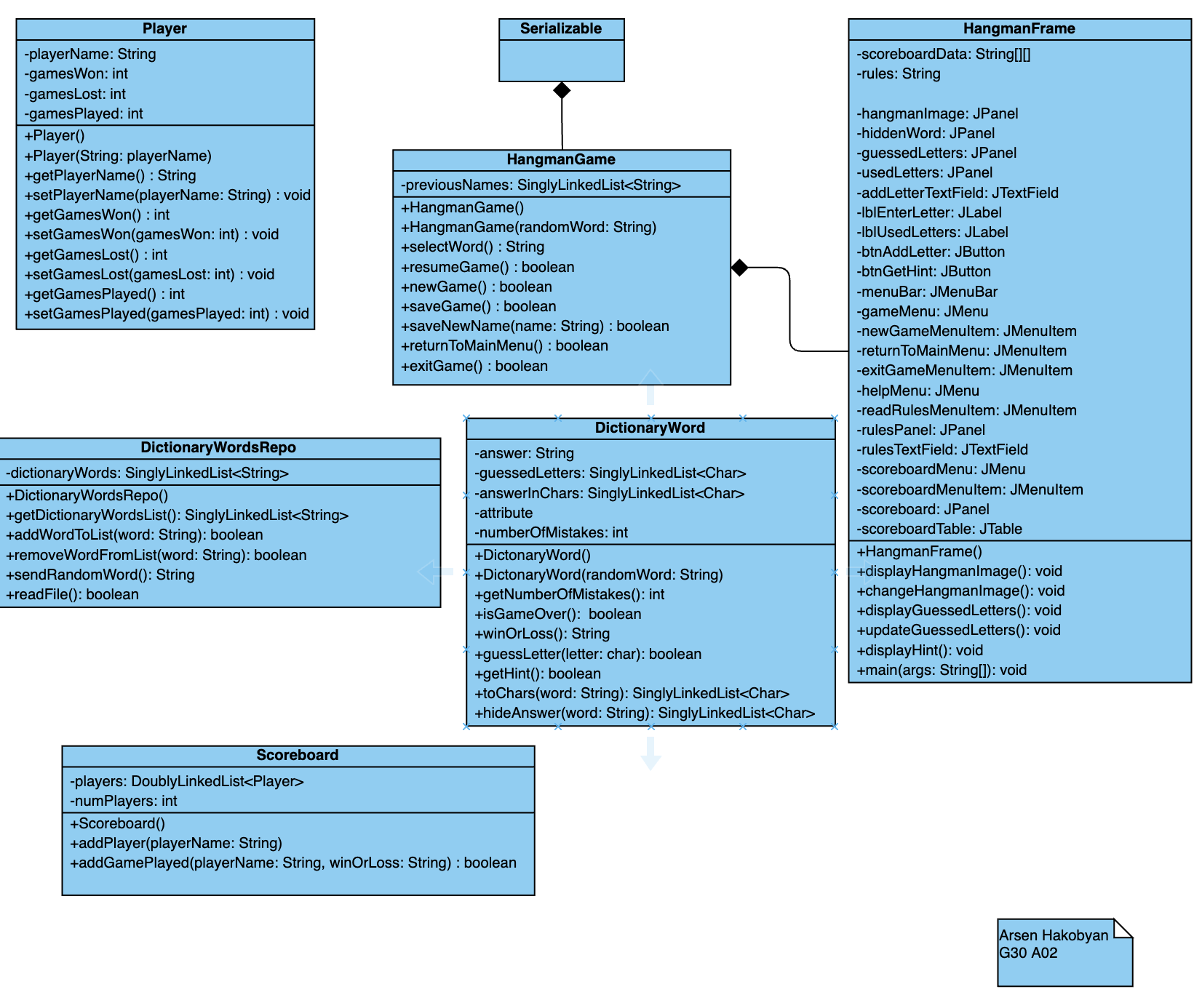
**CLASS DIAGRAM:**

****

**Test Cases for DictionaryWordsRepo class**

**Test Case 1: addWordToList(word: String) : boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * word should have valid characters | 1. A-Z and a-z 2. Contains "-" | 1. numbers 2. Symbols/Special characters 3. "" |
| * Must have more than one 2 letters | 1. >2 | 1. <2 2. =2 |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | "greenhouse" |
| 2 | "jack-o-lantern" |
| 3 | "Exhibit1" |
| 4 | "Mike&John" |
| 5 | "" |
| 6 | "Egg" |
| 7 | "O" |
| 8 | "if" |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. The word from the dictionary is a has valid characters | Check if the word is valid | word="greenhouse" | The word is added to the list.  The method returns true. |
| 1. The word from the dictionary is a has valid characters/characters | Check if the word is valid | word="jack-o-lantern" | The word is added to the list.  The method returns true. |
| 1. The word contains number(s) | Check if the word is valid | word="Exhibit1" | The operation fails.  The method returns false. |
| 1. The word has symbols/special characters | Check if the word is valid | word="Mike&John" | The operation fails.  The method returns false. |
| 1. The word is an empty string | Check if the word is valid | word="" | The operation fails.  The method returns false. |
| 1. The word has more than 2 letters | Check If the word is longer than 2 letters | word="Egg" | The word is added to the list.  The method returns true. |
| 1. The word has less than 2 letters | Check If the word is longer than 2 letters | word="O" | The operation fails.  The method returns false. |
| 1. The word has 2 letters | Check If the word is longer than 2 letters | word="if" | The operation fails.  The method returns false. |

**Test Case 2: removeWordFromList(word: String) : boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid**  **Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The word must be in the list | 1. The word is in the list | 1. The word is not in the list |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | word = "sword"  dictionaryWords: "sword" -> "horse" ->"gunman" |
| 2 | word = "egg"  dictionaryWords: "sword" -> "horse" ->"gunman" |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. The word is in the list | Check if the word is in the list | word = "sword"  dictionaryWords: "sword" -> "horse" ->"gunman" | **Deletes** the word and returns **true** |
| 1. The word is not in the list | Check if the word is in the list | word = "egg"  dictionaryWords: "sword" -> "horse" ->"gunman" | The program **fails** and returns **false**. |

**Test Case 3: sendRandomWord(): String**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * Selects a random word from the list | 1. The list returns a random word | 1. The list is empty and there are no more words left |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | dictionaryWords: "sword" -> "horse" ->"gunman" |
| 2 | dictionaryWords: null |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. The list returns a random word from the list | Select a random word from the list | dictionaryWords: "sword" -> "horse" ->"gunman" | returns a random word |
| 1. The word list is empty | Select a random word from the list | dictionaryWords: null | The program **returns null** |

**Test Case 4: readFile(): boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * "dictionary.txt" must exist | 1. The file exists | 1. The file doesn't exist |
| * The file shouldn't be empty | 1. The file has some empty lines 2. Has no empty lines | 1. The file is empty |
| * Processes the return from addWordToList | 1. addWordToList returns true | 1. addWordToList returns false |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | filename="dictionary.txt" |
| 2 | filename="words.txt" |
| 3 | sword  horse  gunman |
| 4 | sword  horse  gunman |
| 5 |  |
| 6 | "sword" |
| 7 | "excibit1" |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. The file exists | Check if the file exists | filename="dictionary.txt" | Proceeds to reading the file |
| 1. The file doesn't exist | Check if the file exists | filename="words.txt" | returns a FileNotFoundException and shows a message to the user |
| 1. The file has empty lines in-between words | Check inside the file | filename="dictionary.txt"  File Contents:  sword  horse  gunman | The program reads in the words that are not empty, passes them to addWordToList for validation. |
| 1. The file has no empty lines | Check inside the file | filename="dictionary.txt"  File Contents:  sword  horse  gunman | The program reads in all the words, passes them to addWordToList for validation |
| 1. The file is empty | Check inside the file | filename="dictionary.txt"  File Contents: | The program throws an EmptyFileException and shows a message to the user |
| 1. addWordToList returns true | Process the return from addWordToList | word = "sword" | The program goes into the next word |
| 1. addWordToList returns false | Process the return from addWordToList | word = "excibit1" | The program ignores the word and goes into the next one |

**Test Cases for DictionaryWord class**

**Test Case 1: isGameOver(): boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * guessedLetters should have no '\_' left | 1. There's no single '\_' in the list | 1. There's at least one '\_' in the list |
| * numberOfMistakes can't be 6 | 1. 0-5 | 1. 6 |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1,3 | guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes: 5 |
| 2,4 | guessedLetters: '\_'->'w'->'o' ->'r'->'d'  numberOfMistakes: 6 |
| 2,3 | guessedLetters: '\_'->'w'->'o' ->'r'->'d'  numberOfMistakes: 5 |
| 1,4 | guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes: 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. There's no single '\_' in the list and the numberOfMistakes is less than 6 | Check if game is over | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 5  guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd' | calls winOrLoss() and  returns true |
| 1. There's at least one '\_' in the list and the numberOfMistakes is 6 | Check if game is over | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 6  guessedLetters: '\_' -> 'w' -> 'o' -> 'r'-> 'd' | calls winOrLoss() and  returns true |
| 1. There's at least one '\_' in the list and the numberOfMistakes is less than 6 | Check if game is over | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 5  guessedLetters: '\_' -> 'w' -> 'o' -> 'r'-> 'd' | return false |
| 1. There's no single '\_' in the list and the numberOfMistakes is 6 | Check if game is over | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 6  guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd' | calls winOrLoss() and  returns true |

**Test Case 2: winOrLoss()**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The guessed word must match the answer | 1. guessed word and answer match | 1. guessed word and answer do not match |
| * The number of mistakes have to be less than 6 | 1. number of mistakes is less than 6 | 1. number of mistakes is 6 |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1,3 | numberOfMistakes = 5  guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd' |
| 2,4 | numberOfMistakes = 6  guessedLetters: '\_' -> 'w' -> 'o' -> 'r'-> 'd' |
| 1,4 | numberOfMistakes = 6  guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd' |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. guessed word and answer match and the number of mistakes is less than 6 | check win or loss | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 5  guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd' | return "win" |
| 1. guessed word and answer do not match and the number of mistakes is 6 | check win or loss | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 6  guessedLetters: '\_' -> 'w' -> 'o' -> 'r'-> 'd' | return "loss" |
| 1. guessed word and answer match, but the number of mistakes is 6 | check win or loss | answerInChars: 's' -> 'w' -> 'o' -> 'r'-> 'd'  numberOfMistakes = 6  guessedLetters: 's' -> 'w' -> 'o' -> 'r'-> 'd' | return "loss" |

**Test Case 3: guessLetter(letter: char): boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The guessed letter must have at least one occurrence inside the answer char | 1. The guessed letter has one occurrence inside the answer char 2. The guessed letter has multiple occurrences inside the answer char | 1. The guessed letter has no occurrence(s) inside the answer char |
| * The input must be a letter | 1. A-Z and a-z | 1. symbols/special characters 2. numbers |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | guessedLetter = 's';  answerInChars: 's' -> 'w' -> 'o' -> 'r' -> 'd' |
| 2 | guessedLetter = 'A'  answerInChars: 'a' -> 'r' -> 'm' -> 'e' -> 'n' -> 'i' -> 'a' |
| 3 | guessedLetter = 'a'  answerInChars: 's' -> 'w' -> 'o' -> 'r' -> 'd' |
| 4 | guessedLetter = 'a' |
| 5 | guessedLetter = '-' |
| 6 | guessedLetter = '2' |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. The guessed letter has one occurrence inside the answer char | Check guess | guessedLetter = 's';  answerInChars: 's' -> 'w' -> 'o' -> 'r' -> 'd' | updates the guessedLetters List at that position  returns true |
| 1. The guessed letter has multiple occurrences inside the answer char | Check guess | guessedLetter = 'A'  answerInChars: 'a' -> 'r' -> 'm' -> 'e' -> 'n' -> 'i' -> 'a' | updates all the occurrences of the letter in the guessedLetters List at that position  returns true |
| 1. The guessed letter has no occurrence(s) inside the answer char | Check guess | guessedLetter = 'a'  answerInChars: 's' -> 'w' -> 'o' -> 'r' -> 'd' | adds the letter to the unsuccessfulGuesses list  return false |
| 1. A-Z and a-z | Ensure the input is a letter | guessedLetter = 'a' | Proceeds with checking the guess |
| 1. symbols/special characters | Ensure the input is a letter | guessedLetter = '-' | throws InvalidInputException and shows the user an error message |
| 1. numbers | Ensure the input is a letter | guessedLetter = '2' | throws InvalidInputException and shows the user an error message |

**Test Case 4: getHint(): boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * checks whether there is one least one empty slot left | 1. There is at least one | 1. There're no empty slots left |
| * reveals a random letter from the empty slots (random if there are multiple empty slots) | 1. Reveals a random letter with a single occurrence 2. Reveals a random letter with multiple occurrences |  |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | '\_' -> 'w' -> 'o' -> 'r'-> 'd' |
| 2 | 's' -> 'w' -> 'o' -> 'r' -> 'd' |
| 1,3 | '\_' -> 'w' -> '\_' -> 'r' -> '\_' |
| 1,4 | '\_' -> 'r' -> 'm' -> 'e' -> 'n' -> 'i' -> '\_' |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. There is at least one empty slot | Checks for empty slots | guessedLetters: '\_' -> 'w' -> 'o' -> 'r'-> 'd' | Proceeds with the method |
| 1. There're no empty slots left | Checks for empty slots | guessedLetters: 's' -> 'w' -> 'o' -> 'r' -> 'd' | no numberOfMistakes change  returns false |
| 1. reveals a random letter (single occurrence) | Reveal letters | guessedLetters: '\_' -> 'w' -> '\_' -> 'r' -> '\_' | guessedLetters: 's' -> 'w' -> '\_' -> 'r' -> '\_'  numberOfMistakes++  return true |
| 1. reveals a random letter (multiple occurrences) | Reveal letters | guessedLetters: '\_' -> 'r' -> 'm' -> 'e' -> 'n' -> 'i' -> '\_' | guessedLetters: 'a' -> 'r' -> 'm' -> 'e' -> 'n' -> 'i' -> 'a'  numberOfMistakes++  return true |

**Test Cases for Scoreboard class**

**Test Case 1: addGamePlayed(playerName: String, winOrLoss: boolean) : boolean**

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid**  **Equivalence**  **Classes** |
| * The Game added must be either a win or a loss | 1. win 2. loss | 1. Everything else |

|  |  |
| --- | --- |
| **Test equivalence # mapping** | **Test value** |
| 1 | win |
| 2 | loss |
| 3 | winloss |

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario/Operation** | **Purpose** | **Object State** | **Expected Result** |
| 1. win | check winOrLoss | winOrLoss="win" | proceeds with the method |
| 1. loss | check winOrLoss | winOrLoss="loss" | proceeds with the method |
| 1. invalid input | check winOrLoss | winOrLoss="winloss" | throws an InvalidInputException |