**SingleGame**

**Test Case 1**: Hint()

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| hint must guess a single letter | 1. guess is a single letter that is part of the solution word  2.guess is a single letter that isn’t part of the solution word | 3. guess is a special character  4.guess is a number |
| hint must guess a letter that’s part of the solution word | 5. guess is a letter part of the solution word | 6. guess is not part of the solution word |
| hint can’t guess the same letter twice | 7. guess is a unique letter that hasn’t been guessed yet | 8. hint guessed a letter that has already been guessed |
|  |  |  |
|  |  |  |

Assume that theWord = “I’ll-be-Testing”

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| ‘T’ | 1,5,7 |
| ‘-’ | 3 |
| ‘9’ | 4 |
| ‘y’ | 6, 7 |
| ‘t’ | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| SingleGame game = new SingleGame(); | Test instantiation of a new SingleGame object | badGuessNum=0;  wordLetters = new SinglyLinkedList<String>();  guessedLetters = new SinglyLinkedList<String>();  theWord = parentGame.receiveNextWord();  //for Testing purposes theWord //is hardcoded  theWord = “I’lll-be-testing”  toString() = \*’\*\*-\*\*-\*\*\*\*\*\*\* |  |
| game.hint() | Test hint functionality | guessedLetters has a random letter of the word added to it  For this example it’ll be “T”  toString()=\*’\*\*-\*\*-t\*\*t\*\*\* |  |
| game.hint() | Test hint functionality a second time | guessedLetter has a random letter from wordLetter added to it, but not the same one.  For this example it’ll be “e”  toString()=\*’\*\*-\*\*-te\*t\*\*\* |  |
| game.hint() | Test hint() with a special character | guessLetter has nothing added to it, and randomize another index of theWord(), then guessLetter again |  |
| game.hint() | Test hint with a number | guessLetter has nothing added to it, and randomize another index of theWord(), then guessLetter again |  |
| game.hint() | Test hint with a letter that isn’t in the word | guessLetter has nothing added to it, and randomize another index of theWord(), then guessLetter again |  |
| game.hint() | Test hint with a a letter that’s already been guessed | guessLetter has nothing added to it, and randomize another index of theWord(), then guessLetter again |  |

**Test Case 2:** getGuessesLeft () and guessLetter(char guess)

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| guess in guessLetter must be a char | 1. guess is a char | 2. guess is any other data type |
| guess must be a letter, upper or lower case | 3. guess is a letter, upper Case  4.guess is a char, lower Case | 5. guess is a number  6. guess is a special character other than “-“ |

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| ‘C’ | 1, 3 |
| ‘e’ | 1, 4 |
| “letter” | 2 |
| ‘9’ | 5 |
| ‘\*’ | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| SingleGame game = new SingleGame(); | Test instantiation of a new SingleGame object | badGuessNum=0;  wordLetters = new SinglyLinkedList<String>();  guessedLetters = new SinglyLinkedList<String>();  theWord = parentGame.receiveNextWord();  //for Testing purposes theWord //is hardcoded  theWord = “Test” |  |
| game.getGuessesLeft() | test getGuessesLeft() |  | 6 |
| game.guessLetter (‘C’) | Test guessLetter functionality with a wrong letter | badGuessNum=1  guessedLetter has ‘C’ added | false |
| game.getGuessesLeft() | test getGuessesLeft() |  | 5 |
| game.guessLetter (‘e’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘E’ added | true |
| game.getGuessesLeft() | test getGuessesLeft() |  | 5 |
| game.guessLetter(‘9’) | test guessLetter with a char that is a number | guessedLetters has nothing added to it | false  new IllegalArgumentException thrown |
| game.guessLetter(‘\*’) | test guessLetter with a char that is a special Character | guessedLetters has nothing added to it | false  new IllegalArgumentException thrown |

**Test Case 3:** toString()

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| when a letter is guessed, all instances of it in the solution word must be displayed | 1.All letters are displayed in the word | 2.Only one of two letters are displayed |
| toString() must return a string | 3.returns a string | 4.returns any other data type |
|  |  |  |
|  |  |  |
|  |  |  |

Assume that theWord = “Test”

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| ‘t’ | 1, 2 |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| SingleGame game = new SingleGame(); | Test instantiation of a new SingleGame object | badGuessNum=0;  wordLetters = new SinglyLinkedList<String>();  guessedLetters = new SinglyLinkedList<String>();  theWord = parentGame.receiveNextWord();  //for Testing purposes theWord //is hardcoded  theWord = “Test” |  |
| game.toString() | test toString () when there are no correct letters guessed |  | \*\*\*\* |
| game.guessLetter (‘t’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘t’ added |  |
| game.toString() | test toString() when some letter has been guessed |  | t\*\*t |
|  |  |  |  |

**Test Case 4 :** containsStars()

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| containsStarts() must return true if the solution word still contains stars, or false if it does not | 1. returns true or false | 2. returns null |
| Must return a Boolean | 3.returns a Boolean | 4.returns any other data type (int) |
|  |  |  |
|  |  |  |
|  |  |  |

Assume that theWord = “Test”

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| ‘t’ | 1, 2 |
|  |  |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| SingleGame game = new SingleGame(); | Test instantiation of a new SingleGame object | badGuessNum=0;  wordLetters = new SinglyLinkedList<String>();  guessedLetters = new SinglyLinkedList<String>();  theWord = parentGame.receiveNextWord();  //for Testing purposes theWord //is hardcoded  theWord = “Test” |  |
| game.toString() | test toString () when there are no correct letters guessed |  | \*\*\*\* |
| game.containsStars() | Test containsStars() with the word only contatining stars |  | true |
| game.guessLetter (‘E’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘E’ added |  |
| game.containsStars() | Test containsStars() with the word containing some stars |  | true |
| game.guessLetter (‘T’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘E’ added |  |
| game.guessLetter (‘S’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘E’ added |  |
| game.containsStars() | Test containsStars() with the word containing no stars |  | false |

**Test Case 5:** isGameOver()

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| SingleGame game = new SingleGame(); | Test instantiation of a new SingleGame object | badGuessNum=0;  wordLetters = new SinglyLinkedList<String>();  guessedLetters = new SinglyLinkedList<String>();  theWord = parentGame.receiveNextWord();  //for Testing purposes theWord //is hardcoded  theWord = “Test” |  |
| SingleGamr game2 = newSingleGame | Test instantiation of a new SingleGame object | badGuessNum=0;  wordLetters = new SinglyLinkedList<String>();  guessedLetters = new SinglyLinkedList<String>();  theWord = parentGame.receiveNextWord();  //for Testing purposes theWord //is hardcoded  theWord = “Test” |  |
| game.isGameOver() | Test isGameOver() when game is not over |  | false |
| game.guessLetter (‘A’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘A’ added  badGuessNum=1 |  |
| game.guessLetter (‘B’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘B’ added  badGuessNum=2 |  |
| game.guessLetter (‘C’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘C’ added  badGuessNum=3 |  |
| game.guessLetter (‘D’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘D’ added  badGuessNum=4 |  |
| game.guessLetter (‘Z’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘Z’ added  badGuessNum=5 |  |
| game.guessLetter (‘F’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘F’ added  badGuessNum=6 |  |
| game.isGameOver() | Test isGameOver() when the game is over and loss | gameWon=false; | true |
| game2.isGameOver() | Test isGameOver() when game is not over |  | false |
| game2.guessLetter(‘T’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘T’ added |  |
| game2.guessLetter(‘E’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘E’ added |  |
| game2.guessLetter(‘S’) | Test guessLetter functionality with a correct letter | guessedLetter has ‘S’ added |  |
| game2.isGameOver() | Test isGameOver() when game is over and won | gameWon=true | true |

**Player**

**Test Case 6:** incWins() and incLosses(), and instantiation using both Player Constructors

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| Player p1 = new Player(); | Test instantiation of a new Player object | timesPlayed =0;  playerName=”Anonymous”;  wins = 0; | a new blank player is created with the default name “Anonymous” |
| Player p2 = new Player(“Noah”); | Test instantiation of a new Player object | playerName = “Noah”  wins = 0;  timesPlayed =0; | a new blank player is created with the name of “Noah” |
| p1.incWins() | Test incWins() | wins=1  timesPlayed=1 |  |
| p2.incLosses() | Test incLosses() | timesPlayed = 1 |  |
| p2.getLosses() | Test getLosses() |  | 1 |
| p1.getWins() | Test getWins() |  | 1 |

**Test Case 7**: getPlayerName() and setPlayerName(name)

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| name must be a string | 1. name is a String | 2. name is any other data type |
| name must not be empty or set to null | 3. name length is >=0 | 4. name length is <0  5. name is null |

For mapping and testing for the name, see Test Case # , addPlayer(String player)

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| SEE TEST CASE #10 , addPlayer(playerName) | 1, 2, 3, 4, 5 |
| “true” | 6 |
| “W” | 7 |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Purpose** | **Changed Object State** | **Expected result** |
| Player p1 = new Player(); | Test instantiation of a new Player object | timesPlayed =0;  playerName=”Anonymous”;  wins = 0; | a new blank player is created |
| Player p2 = new Player(“Noah”); | Test instantiation of a new Player object | playerName = “Noah”  wins = 0;  timesPlayed =0; | a new blank player is created with the name of “Noah” |
| p2.getPlayerName() | Test getPlayerName |  | Noah |
| p1.setPlayerName(“Tristan”) | Test setPlayerName(name) | wins=1  timesPlayed=1  playerName =”Tristan” |  |
| p1.getPlayerName() | Testing getPlayerName |  | Tristan |

**ScoreBoard**

**Test Case 8**: gamePlayed(name, winLoss)

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| name must be a string | 1. name is a String | 2. name is any other data type |
| name must not be empty or set to null | 3. name length is >=0 | 4. name length is <0  5. name is null |
| winLoss must be a boolean | -winLoss is a boolean | -winLoss is any other data type |

For mapping and testing for the name, see Test Case # , addPlayer(String player)

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| SEE TEST CASE #10 , addPlayer(playerName) | 1, 2, 3, 4, 5 |
| “true” | 6 |
| “W” | 7 |
|  |  |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| ScoreBoard board = new ScoreBoard(); | Test instantiation of a new ScoreBoard | players = new DoublyLinkedList<>() | A new Scoreboard window is created |
| players.add(new Player(“Noah”) | Test adding a new Player to players | a new Player is created with the name “Noah” and is added to players |  |
| board.gamePlayed(“Noah”, true) | Test gamePlayed(name, winLoss) with a Player that already exists, and a win | The Player Noah has a win added |  |
| board.gamePlayed(“Tristan”, true) | Test gamePlayed(name, winLoss) with a Player that does not exist, and a win | a new Player is created with the name “Tristan”, and added to players. The Player “Tristan” has a win added |  |
| board.gamePlayed(“Tristan”, false) | Test gamePlayed with a player that exists, and a loss | the Player “Tristan” has a loss added |  |
| board.gamePlayed(“Abbi”, false | Test gamePlayed(name, winLoss) with a Player that does not exist, and a loss | a new Player is created with the name “Abbi”, and added to players. The Player “Abbi” has loss added |  |
| board.gamePlayed(“Abbi”, “W”); | Test gamePlayed with an Illegal Data type for winLoss |  | IllegalArgumentException |

**Test Case 9:** getNextPlayer(index)

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| index must be an int | 1. index is an int | 2. index is any other data type |
| index must be greater than or equal to 0, and cannot be null | 3.index >0  4.index = 0 | 5. index is <0  6.index is null |
| index must be less than to the length of players | 7.index < players.getLength() | 8.index >players.getLength()  9.index = players.getLength() |

Assume players = {“Noah”, “Tristan”, “Abbi”}

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| 1 | 1, 3, 7, 8 |
| 0 | 1, 4 |
| null | 6 |
| -1 | 5 |
| 3 | 9 |

|  |  |  |  |
| --- | --- | --- | --- |
| ScoreBoard board = new ScoreBoard(); | Test instantiation of a new ScoreBoard | players = new DoublyLinkedList<>() | A new Scoreboard window is created |
| board.getNextPlayer(0) | Test getNextPlayer(index) with an empty list |  | IndexOutOfBoundsException |
| players.add(new Player(“Noah”) | Test adding a new Player to players | a new Player is created with the name “Noah” and is added to players |  |
| board.getNextPlayer(0) | Test getNextPlayer(index) with a list of length 1 |  | the Player “Noah” get’s returned |
| board.getNextPlayer(1) | Test getNextPlayer(index) with a list of length 1 |  | IndexOutOfBoundsException |
| players.add(new Player(“Tristan”) | Test adding a new Player to players | a new Player is created with the name “Noah” and is added to players |  |
| players.add(new Player(“Abbi”) | Test adding a new Player to players | a new Player is created with the name “Noah” and is added to players |  |
| board.getNextPlayer(0) | Test getNextPlayer(index) with a list of length 3 |  | the Player “Noah” gets returned |
| board.getNextPlayer(1) | Test getNextPlayer(index) with a list of length 3 |  | the Player “Tristan” gets returned |
| board.getNextPlayer(2) | Test getNextPlayer(index) with a list of length 3 |  | the Player “Abbi” gets returned |
| board.getNextPlayer(3) | Test getNextPlayer(index) with a list of length 3 |  | IndexOutOfBoundsException |
| board.getNextPlayer(null) | Test getNextPlayer(index) with null |  | IllegalArgumentException |
| board.getNextPlayer(-1) | Test getNextPlayer with a negative number |  | IllegalArgumentException |
|  |  |  |  |

**Test Case 10:** addPlayer(String playerName)

|  |  |  |
| --- | --- | --- |
| **Rules/Constraints** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| playerName must be a string | 1. playerName is a String | 2. playerName is any other data type |
| playerName cannot be null | 3. playerName length is >0  playerName length is 0 | 4. playerName length is <0  5.playerName is null |

|  |  |
| --- | --- |
| **Test value** | **Test equivalence # mapping** |
| “Noah” | 1, 3 |
| ‘N’ | 2 |
| “” | 4 |
| null | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| ScoreBoard board = new ScoreBoard(); | Test instantiation of a new ScoreBoard | players = new DoublyLinkedList<>() | A new Scoreboard window is created |
| board.addPlayer(“Noah”) | Test addPLayer(playerName) | a new Player is created with the name “Noah” and is added to players |  |
| board.addPlayer(“”) | Test addPlayer with a empty string | A new player with the default playerName of “Anonymous” and is added to players |  |
| broad.addPlayer(“Noah”) | Test addPlayer(playerName) with a string that already has a Player created for it | No new Player is created |  |
| broad.addPlayer(‘c’) | Test addPlayer with an invalid Data type |  | IllegalArumentException |
| broad.addPlayer(null) | Test addPlayer with null | A new player with the default playerName of “Anonymous” and is added to players. If it already exists, no new Player is created |  |

**Test Case 11**: sortScoreboard()

|  |  |  |  |
| --- | --- | --- | --- |
| ScoreBoard board = new ScoreBoard(); | Test instantiation of a new ScoreBoard | players = new DoublyLinkedList<>() | A new Scoreboard window is created |
| board.gamePlayed (“Noah”, true); | Test adding a new Player to players | a new Player is created with the name “Noah” and is added to players, also has a win added  players = {“Noah”} |  |
| board.gamePlayed (“Tristan”, true); | Test adding a new Player to players | a new Player is created with the name “ Tristan ” and is added to players  players = {“Noah”, “Tristan”} |  |
| board.gamePlayed (“Abbi”, true); | Test adding a new Player to players | a new Player is created with the name “ Abbi ” and is added to players  players = {“Noah”, “Tristan”, “Abbi”} |  |
| board.sort() | Testing sort method | players = {“Abbi”, “Noah”, “Tristan”} |  |

Dictionary

**Test Case 12:** Default constructor and nextWord()

|  |  |  |  |
| --- | --- | --- | --- |
| Dictionary dic = new Dictionary() | Test instantiation of a new Dictionary | allWords = {//all words in dictionary.txt file} | A new Dictionary is created |
| dic.nextWord() | Test nextWord() | allWords = {//all words except for the currentWord} | currentWord |
| //Imagine reaching the end of the file by doing dic.nextWord() n amount of times |  |  |  |
| dic.nextWord() | Test nextWord() when allWords is empty | allWords ={} | Out of words, reloading file |
| dic.nextWord() | Test nextWord() | allWords = {//all words except for the currentWord} | currentWord |