Nahom Haile

October 30th, 2020

Programming III (420-G30-HR)

Assignment 2 – Hangman Using Linked Lists Test Cases

**Black Box Test Case Design for Hangman**

**HangmanFrame class**

Entering your name

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Must be a string | 1. “Nahom” | 1. Numbers |
| Must have a length above 0 | 1. “Nahom” | 1. “” |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| String(“Nahom”) | 1,3 |
| Int(124) | 2 |
| String(“”) | 4 |

**Hangman class**

Choosing a Letter

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Must be a Letter | 1. A-Z or a-z | 1. #’s 2. Characters (!@$) |
| Must be a single letter | 1. A-Z or a-z | 1. String 2. Double 3. Float number |
| Cannot guess the same letter twice | 1. If the letter is guessed again then an error is thrown | 1. If the letter is guessed again no error is shown |
| All occurrences with the guess should be shown in the word | 1. A letter guessed displays in the word in all occurrences | 1. Only one letter appears in the word when there are three occurrences |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| Char(‘Q’) | 1,4 |
| Int(4) | 2 |
| Char(‘!’) | 3 |
| String(“Hello”) | 5 |
| Double(12.00) | 6 |
| Float(12) | 7 |
| Word: doorbell  \_ \_ \_ \_ \_ \_ \_ \_  Letter guessed: o  \_ o o \_ \_ \_ \_ \_ | 10, 1, 4 |
| Word: doorbell  \_ \_ \_ \_ \_ \_ \_ \_  Letter guessed: o  \_ o \_ \_ \_ \_ \_ \_ | 11, 1, 4 |
| Word: doorbell  \_ \_ \_ \_ \_ \_ \_ \_  Letter guessed: o  \_ o o\_ \_ \_ \_ \_  Second letter guessed: o  Throw error(‘letter guessed twice’) | 8, 1, 4 |
| Word: doorbell  \_ \_ \_ \_ \_ \_ \_ \_  Letter guessed: o  \_ o o\_ \_ \_ \_ \_  Second letter guessed: o  \_ o o\_ \_ \_ \_ \_ | 9, 1, 4 |

Hint() method

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Cannot exceed 1 hint | 1. Want a hint for the first time | 1. Want a hint for the second time |
| Cannot receive a hint after you solved the word | 1. Word isn’t solved and want a hint | 1. Word is solved and you want a hint |
| Cannot receive a hint that is a letter that was already guessed | 1. hint given was not guessed | 1. hint given is in the word |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| The number of hints I took = 0  want a hint | 1 |
| The number of hints I took = 1  want a hint | 2 |
| Word= d o o r b e\_ \_  want a hint | 3 |
| Word= d o o r b e l l  want a hint | 4 |
| Word \_ a \_ \_  Hint given is ‘a’ | 6 |
| Word: \_ \_ \_ k  Hint given is d  Word: d \_ \_ k | 5 |

Equals()

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Two Word Objects | 1. The two words I am comparing are Word objects | 1. Two different types of objects |
| Two words are equals | 1. Words have the same value | 1. Words are not equal |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| new Word(“Nahom”)  new Word(“Nahom”) | 1, 5 |
| new Word(“Nahom”)  new Player(“Nahom”) | 2 |
| new Word(“Nahom”)  new Word(“Serge”) | 1, 6 |

checkGuess(char letter)

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Letter is valid | 1. Letter is is char 2. Letter is in the word | 1. Letter is not a char 2. Letter is not in the word |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| Word = “Nahom”  Letter is A | 1, 2 |
| Word = “Nahom”  Letter is C | 1, 4 |
| Word is “Nahom”  Letter is 1 | 3, 4 |

isGameWon()

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Words matches the word they are trying to guess | 1. All letter guessed are in the word | 1. Word isn’t solved yet |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| Word: “H e l l o”  Comparing to is “H e l l o” | 1 |
| Word: “H e l l \_”  Comparing to is “H e l l o” | 2 |

isGameLose()

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Word doesn’t match word they are trying to guess | 1. Word isn’t solved yet | 1. Word is solved |
| Number of guesses is 6 before they finish the word | 1. Number is guessed < 6 | 1. Number is guesses is > 6 |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| Word: D a r \_  Comparing to D a r k  Number of guesses = 5 | 1, 4 |
| Word: D \_ \_ k  Comparing to D a r k  Number of guesses = 6 | 3 |
| Word: D \_ \_ k  Comparing to D a r k  Number of guesses = 4 | 4 |
| Word is D a r k  Comparing to D a r k  Number of guesses = 4 | 2 |

**Scoreboard Class**

gamePlayed()

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Valid Player | 1. Player must be already in the scoreboard | 1. Player not in the scoreboard |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| Scoreboard names [Nahom, Serge, Roy]  Adding the games played for Ibrahim | 2 |
| Scoreboard names [Nahom, Serge, Roy]  Adding the games played for Serge | 1 |

sortList()

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| List in is ascending order | 1. Ascending order | 1. Not in ascending order |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| [“Nahom”, “Richard”] | 1 |
| [“Nahom”, “Richard”, “Ashley”] | 2 |

**Player Class:**

checkName(String name)

|  |  |  |
| --- | --- | --- |
| Rules/Constraints | Valid Equivalence Classes | Invalid Equivalent Classes |
| Name is valid | 1. Name is a string with letters | 1. Name is not a string 2. Name contains numbers 3. Name contains charatcers |

|  |  |
| --- | --- |
| Test Values: | Test Equivalence # Mapping |
| Name is “Nahom” | 1 |
| Name is 214.1 | 2,3 |
| Name is ? | 2,4 |

**Black Box Test Case Scenarios for Hangman**

**Scoreboard class**

Test Case 1 – create new object and getting numOfPlayers with the correct value

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0 | A new Scoreboard object is created |
| Scoreboard.getNumOfPlayers | Testing the number of players is the correct value | Unchanged | 0 |

addPlayer() Test Plan

Test Case 2 – new player

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0 | A new Scoreboard object is created |
| Scoreboard.addPlayer(“Nahom”) | To add a new player | numOfPlayers=1 | Nahom |
| Scoreboard.addPlayer(“Serge”) | To add a new player | numOfPlayers=2 | Serge |

gamePlayed() Test Plan

Test Case 3 – add a win and lose to a player

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0 | A new Scoreboard object is created |
| Scoreboard.gamePlayed(“Nahom”, true) | Add a game played to Nahom with a win | numOfPlayers=1 | Win |
| Scoreboard.gamePlayed(“Nahom”, false) | Add a game played to Nahom with a lose | numOfPlayers=1 | lose |

Test Case 4 – with an invalid Player

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0 | A new Scoreboard object is created |
| Scoreboard.gamePlayed(“Ibrahim”, true) | Add a game played with an invalid players |  | InvalidNameException – name: “Ibrahim” is not a player we found |

getNextPlayer() Test Plan

Test Case 5 – Players list is empty

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0  players=[] | A new Scoreboard object is created |
| Scoreboard.getNextPlayer(1) | Getting the next player from an empty list |  | EmptyLisrException – the players list is empty |

Test Case 6 – Players list has one player

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0  players=[“Nahom”] | A new Scoreboard object is created |
| Scoreboard.getNextPlayer(0) | Getting the next player with a single name in players | numOfPlayers=0  players=[“Nahom”] | Nahom |

Test Case 7 – Players list has three playerss

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Scoreboard scoreboard = new Scoreboard (); | Create a new Scoreboard object | numOfPlayers=0  players=[“Nahom”, “Serge”, “Ashley”] | A new Scoreboard object is created |
| Scoreboard.getNextPlayer(1) | Getting the next player with a single name in players | numOfPlayers=0  players=[“Nahom”, “Serge”, “Ashley”] | Serge |

**Players class**

Test Case 1 – a new player object and checking the variables

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Player player1 = new Player() | Instantiating a new player object | numGamesPlayed=0  numGamesWon=0 | A new player object |
| Player1.getNumGamesPlayed() | Testing to get the correct val for numGamesPlayed | numGamesPlayed=0  numGamesWon=0 | 0 |
| Player1.getNumGamesWon() | Testing to get the correct val for numGamesWon | numGamesPlayed=0  numGamesWon=0 | 0 |

Test Case 2 – incrementing games played and games won

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Player player1 = new Player() | Instantiating a new player object | numGamesPlayed=0  numGamesWon=0 | A new player object |
| Player1.setNumGamesPlayed(1)  Player1.getNumGamesPlayed() | Incrementing games played to test the correct val | numGamesPlayed=1  numGamesWon=0 | 1 |
| Player1.setNumGamesWon(1)  Player1.getNumGamesPlayed() | Incrementing games won to test the correct val | numGamesPlayed=0  numGamesWon=1 | 1 |

**Hangman class**

Test Cases 1 – create a new hangman object and check the word

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Hangman hangman = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| Hangman.getWord() | Testing to get the word from a hangman object | numOfGuesses=0  wordChossen=”Doorbell” | Doorbell |

**Test Cases for hangman.equals()**

Test Case 1 – Two identical hangman’s

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Hangman hangman1 = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| Hangman hangman2 = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| Hangman1.equals(hangman2) | Verify that the equals() method works for two hangman’s with the same words | unchanged | true |
| Hangman2.equals(hangman1) | Verify that the equals() method works for two hangman’s with the same words (confirm symmetry of equals) | unchanged | true |

Test case 2 – two hangman’s with different words

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Hangman hangman1 = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| Hangman hangman2 = new Hangman(“Dish”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=” Dish” | A new hangman object |
| Hangman1.equals(hangman2) | Verify that the equals() method works for two hangman’s with the same words | unchanged | False |
| Hangman2.equals(hangman1) | Confirm symmetry | unchanged | False |

Test Case 3 – One hangman with an invalid object

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Hangman hangman1 = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| Player hangman2 = new Player(“Dish”) | Creating an object that isn’t an hangman | numGamesPlayed=0  numGamesWon=0  name=”Dish” | A new Player object |
| Hangman1.equals(hangman2) | Verify that the equals() method does not work for a hangman object and a plyer object |  | False |

**Test Cases for hangman.checkGuess()**

Test Case 1 – testing if the letter is in the word

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Hangman hangman = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| hangman.checkGuess(‘o’) | To check if the letter is in the word | guessLetter=’o’ | true |

Test Case 2 – testing if the letter is not in the word

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| Hangman hangman = new Hangman(“Doorbell”) | Instanitating a new Hangman object | numOfGuesses=0  wordChossen=”Doorbell” | A new hangman object |
| hangman.checkGuess(‘t’) | To check if the letter is in the word | guessLetter=’t’ | false |

**HangmanWordList Class**

Test Case for HangmanWordList.readFile()

Test Case 1 – reading in a file with not filename

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| HangmanWordList list = new HangmanWordList() | Instanitating a new HangmanWordList object | Filename=”” | A new HangmanWordList object |
| List.readFile() | Checking if it reads a file from an invalid filename |  | NoFilenameException – filename is an empty string |

Test Case 2 – reading in a file that doesn’t exist

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| HangmanWordList list = new HangmanWordList() | Instanitating a new HangmanWordList object | Filename=”ddd” | A new HangmanWordList object |
| List.readFile() | Checking if it reads a file from an invalid filename |  | InvalidFilenameException – filename doesn’t exist |

Test Case 3 – reading in a file does exist

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| HangmanWordList list = new HangmanWordList() | Instanitating a new HangmanWordList object | Filename=”dictionary.txt” | A new HangmanWordList object |
| List.readFile() | Checking if it reads a file with a valid filename |  | Read the file into the singlylinkedlist |

Test Cases for HangmanWordList.saveGame()

Test Case 1 – writing to a file with not filename

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| HangmanWordList list = new HangmanWordList() | Instanitating a new HangmanWordList object | Filename=”” | A new HangmanWordList object |
| List.saveGame() | Checking if it reads a file from an invalid filename |  | NoFilenameException – filename is an empty string |

Test Case 2 – writing to a file that doesn’t exist

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| HangmanWordList list = new HangmanWordList() | Instanitating a new HangmanWordList object | Filename=”ddd” | A new HangmanWordList object |
| List.saveGame() | Checking if it reads a file from an invalid filename |  | InvalidFilenameException – filename doesn’t exist |

Test Case 3 – writing to a file that does exist

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Purpose | Object State | Excepted Result |
| HangmanWordList list = new HangmanWordList() | Instanitating a new HangmanWordList object | Filename=”savedGame.txt” | A new HangmanWordList object |
| List.saveGame() | Checking if it reads a file from a valid filename |  | Game saved in saveGame.txt |