

Treasure Box Braille

Testing Document

Contents

1. All test cases.....	3
2. Test cases and Their Derivation	8
3. Why are these test cases sufficient?	10
4. Test Coverage Discussion.....	10

1. All test cases

The following test cases have been selected to insure our application runs as intended. However, we do not fully depend on these tests cases because the application has components which depend and interact with each other. This means that in addition to these test cases, quality assurance by running the application and testing the features manually before launch is also key to a stable release. Not all the following test cases have been implemented at this time and further cases may be added as the need arises.

Controller

insert test cases here

FocusPolicy

- I. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.
- II. **test01_ctorLoadParser** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects.
- III. **test02_addOrder** - Tests that the Vector<Component> order field contains the correct Vector list of components. Ensures manually set focus order is in effect.

ListManager

- I. **test00_init** - Tests initialization of the constructor. Checks that the number of cells and number of buttons is correctly set.
- II. **test01_add** - Tests that adding the next node will correctly increment the index as well as insert the correct key phrase and data into the node.
- III. **test02_size** - Tests that the size of the current list grows are more nodes are added.
- IV. **test03_index** - Tests that the index correctly increases are new nodes are added.
- V. **test04_getData** - Tests that retrieving data from nodes out of bounds will throw the proper IndexOutOfBoundsException.
- VI. **test05_getKeyPhrase** - Tests that retrieving key phrases from nodes out of bounds will throw the proper IndexOutOfBoundsException.
- VII. **test06_createJunction** - Tests the creation of a junction. Assert that the names of each new branch is the same as the given name in the parameter.
- VIII. **test07_junctionGoto** - Tests that going to a node in a junction by index returns the correct node. Checks that an uninitialized branch correctly returns null.

VII. **test08_junctionSearch** - Tests that searching a node by name will return the correct index. If the name is not found, then assert that -1 is returned instead.

VII. **test09_remove** - Tests that removing a node will only remove the current node and correctly changes the index. Removal of a junction node should remove the junction (and all its branches) and append the rest of the list. Removal of a button node should remove the contents of the button but should not modify the rest of the data structure.

LoadParser

- I. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.
- II. **test01_ctorLoadParser** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects.
- III. **test02*_fromText**- Test the fromText(**String scenarioFile**) method in the class LoadParser. Test cases checks that the method reads the **String scenarioFile** and that the method checks the first and splits the first two lines of the file and gets the cell and button number from the file and then creates a ListManager class object. Checks that method Throws IllegalArgumentException if the cell and button values are not positive integers. The test cases also ensure that scenario file is iterated through, elements in the file are identified and nodes with corresponding data are created for every valid KeyPhrase identified in the file. Checks that IOException is thrown by class if error occurs. Test case also checks that the created ListManager object is a valid list and is returned All tests for fromText(**String scenarioFile**) are labeled alphabetically in place of the * character.
- IV. **test03*_junction** - Tests the junction(**ListManager result**) method in the class LoadParser. This test case is responsible for checking that junction nodes are created in **ListManager result**. It checks that the method creates separate list objects for different junctions and catches IllegalArgumentException thrown by the method. Test also ensure that a final path is created where all the junctions lead up to. All tests for junction(**ListManager result**) are labeled alphabetically in place of the * character.
- V. **test04_nextLine** - Tests the nextLine() method in the class LoadParser. This test case checks that the next line in a file reader is returned when the method is called. Test case also catches IOException.
- VI. **test05_scenario_01** - Tests LoadParser() by loading the example scenario file: scenario_01.txt and checking that navigating through the parsed data structure using ListManager yields the expected node data and list lengths.
- VII. **test05_scenario_02** - Tests LoadParser() by loading the example scenario file: scenario_02.txt and checking that navigating through the parsed data structure using ListManager yields the expected nodes and list lengths.
- VIII. **test05_scenario_03** - Tests LoadParser() by loading the example scenario file: scenario_03.txt and checking that navigating through the parsed data structure using ListManager yields the expected nodes and list lengths.

Main***insert test cases here***Node

- IX. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.
- X. **test01*_ctorNode** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.
- XI. **test02*_junction** - Tests the junction(`ArrayList<ArrayList<Node>> buttons`, `HashMap<Integer, String> buttonsNames`, `ArrayList<Node> next`) method of Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits `buttons` and `buttonsNames` from values provided when the method is called. All tests for junction(`ArrayList<ArrayList<Node>> buttons`, `HashMap<Integer, String> buttonsNames`, `ArrayList<Node> next`) are labeled alphabetically in place of the * character.
- XII. **test03*_button** - Tests the button(`String name`, `ArrayList<Node> prev`) method in Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits `name` and `prev` from values provided when the method is called. It also checks that the method returns the created node object. All tests for button(`String name`, `ArrayList<Node> prev`) are labeled alphabetically in place of the * character.
- XIII. **test04*_buttonNext** - Tests the buttonNext(`ArrayList<Node> prev`) method in Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits `prev` from values provided when the method is called. It also checks that the method returns the created node object. All tests for buttonNext(`ArrayList<Node> prev`) are labeled alphabetically in place of the * character.
- XIV. **test05*_tail** - Tests the tail(`ArrayList<Node> next`) method in Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits `next` value provided when the method is called. It also checks that the method returns the created node object. All tests for tail(`ArrayList<Node> next`) are labeled alphabetically in place of the * character.
- XV. **test06_setKeyPhrase** - Tests the setKeyphrase(`String s`) method of Node class. This test case checks that the key phrase of current node is set to `String s` value provided when the method is called.

- XVI. **test07_setData** - Tests the setData(**String s**) method of Node class. This test case checks that the data of current node is set to **String s** value provided when the method is called.
- XVII. **test08_getKeyPhrase** - Tests the getKeyPhrase() method of Node class. This test case checks that correct key phrase of type string is returned when the method is called.
- XVIII. **test09_getData** - Tests the getData() method of Node class. This test case checks that correct Data of type string is returned when the method is called.
- XIX. **test10_getButtons** - Tests the getButtons() method of Node class. This test case checks that correct Buttons of type ArrayList<ArrayList<Node>> is returned when the method is called.
- XX. **test11*_braileToLetter** - Tests the braileToLetter(**String s**) method. This test case checks that for a set of inputted char, the correct brail sequence is returned. It takes in the **String s** and checks it against the braille index then calls the braileToLetters method with the **String s** and checks that it returns the correct value. All tests for braileToLetter(**String s**) are labeled alphabetically in place of the * character.

RequestFocusListener

- IV. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.
- V. **test01_ctorLoadParser** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects.
- VI. **test02_ancestorAdded** - Tests that once RequestFocusListener is initialized ancestorAdded method has correctly applied requestFocusInWindow() on the component.

ScenarioComposer

- I. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.
- II. **test01*_ctorScenarioComposer** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.
- III. **test02*_returnStringFile** - Tests the returnStringFile(**ListManager aList**) method in ScenarioComposer class. This test case checks the file returned by the method. It checks that the for loop iterates through the given **ListManager** object **aList** and checks the keyphrase of each node. It checks that based on the keyphrase of a node, the returnStringFile method performs its related set

of actions. It also checks that the file writer object writes correctly to the text file. The test cases are also responsible for ensuring junction nodes in each `ListManager` `aList` are created to scenario file requirements. All tests for `returnStringFile` are labeled alphabetically in place of the * character.

SoundRecorder

- I. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.
- II. **test01*_ctorSoundRecorder** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.
- III. **test02*_CaptureAudio**- Tests the capture audio method. The test cases ensure check that the audio format is set and is supported, checks that the data line is opened for recording and closed afterwards to be reused. It also checks that the information received from the data line is stored as `byteArrayOutputStream` for play back and saving audio files. The test also ensures that the correct exceptions are thrown in case of errors. All tests for `CaptureAudio` method are labeled alphabetically in place of the * character.
- IV. **test03*_PlayAudio**- Tests the `PlayAudio` method. Implemented test cases ensure that the data saved as `byteArrayOutputStream` is converted to `InputStream` data. It checks that the audio being played is of the correct format and that an `audioline` is opened for the `InputStream`. The test also checks that the line is closed after use to prevent `LineUnavailableException` from being thrown. It also checks that the `JDialogBoxes` for saving and exporting files are displayed immediately after playback.
 The method `playAudio` is also responsible for playing audio imported to the Audio Studio. Tests ensuring the imported file is a wav file and that the wav file is playable are also included in this test case. The test also ensures that the correct exceptions are thrown in case of errors. All tests for `PlayAudio` method are labeled alphabetically in place of the * character.
- V. **test04_getFormat**- Tests the `getFormat` method. This test case ensures that the method returns the correct audio format when called.
- VI. **test05*_save**- Tests the `save(File)` method. Implemented test cases ensure that the data saved as `byteArrayOutputStream` is converted to `InputStream` data. It checks that the audio being played is of the correct format and that the `InputStream` data is written to the file provided to the method. The case also checks that this file is stored for exporting and then closes the `byteArrayOutputStream`. All tests for `save(File)` method are labeled alphabetically in place of the * character.

View

insert test cases here

2. Test cases and Their Derivation

Controller

insert test cases here

FocusPolicy

This class was found on docs.oracle.com regarding the Focus Subsystem. These tests are derived from reading the documentation from oracle and determining what aspects of the code are being used in our application. In our case, we use the constructor and addFocus() method.

ListManager

Test cases are meant to test that ListManager correctly handles adding and removing new data. ListManager acts as our data structure so it is important that the current index is properly incremented or decremented. The tests also cover the junction node creation and navigation. These tests were derived from comparative tests for lists, mainly the successful navigation and modification of the data structure.

LoadParser

These test cases were derived from a well-formed understanding of the required format of the scenario file and the structure of our ListManager data structure. The class is responsible for converting already existing scenario files to ListManager objects that can be iterated through with our program. The test cases test the core functionality of the program based on the requirements for the LoadParser class.

Main

insert test cases here

Node

Test cases in this class are derived from an understanding of the basic functionality of a node. The test cases ensure that the class successfully performs actions such as node creation, getting and setting fields of current node, keeping track of current node location, and keeping track of special nodes such as a junction node or a tail node.

RequestFocusListener

Test cases were derived for this convenience class based on what parts of the class is being used. The class was sourced from 'Rob Camick' (<https://tips4java.wordpress.com/2010/03/14/dialog-focus/>). We are only using the override of the ancestorAdded() method to force NVDA to focus on certain components. We test that requestFocusInWindow() has successfully forced focus on the component. This is the only method used by us in the class.

ScenarioComposer

Test cases for the class ScenarioComposer are written to ensure that the scenario file the class is responsible for creating follows the valid documented scenario format style. The ScenarioComposer class takes the information from our data structure ListManager and writes items in the data structure according to the keyphrase and data of every Node object in the provided ListManager Object.

SoundRecorder

The test cases implemented for this class include tests for methods in java sound api. Tests handle cases such as checking that the program sets the audio format for recorded audio, checking that audio lines are opened and closed for reuse and checking that saved files are .wav files and not of a different format. These tests were derived from an understanding of the java sound api and sound utilities .

View

insert test cases here

3. Why are these test cases sufficient?

Controller

insert test cases here

FocusPolicy

FocusPolicy overrides certain methods in FocusTraversalPolicy to allow a Vector<Component> field to dictate the order of focusing when pressing tab. The tests listed are sufficient because we are only using the constructor to override the normal traversal policy.

ListManager

These test cases are sufficient because they cover all the major methods of the class. There are many methods which are simply not used or were created for testing purposes. The test cases cover major aspects of a data structure such as adding and removing. It also tests the junction creation which allows for branching of the scenario. ListManager essentially manages many ArrayLists, so the testing covers the additional features of ListManager.

LoadParser

The test cases written for the class LoadParser are sufficient because they are written to test most possible scenarios for individual methods of the class. The tests check and ensure that the LoadParser class returns a valid ListManager object which can be iterated through with our program.

Main

insert test cases here

Node

The implemented test cases for this class are sufficient because they cover all the methods in the class. The tests ensure that these methods run as they are expected to and return whatever necessary values when called. It tests specifically, the conversion of characters to braille cell and ensures the correct braille cell value is returned for a given character.

RequestFocusListener

insert test cases here

ScenarioComposer

The test cases for the ScenarioComposer class are sufficient as they can cover and ensure all the functionality of traversing the provided ListManager object, check that the class is identifying each node and performing several actions based on the current node and checking the successful writing of the final Scenario File.

ScenarioParser

insert test cases here

SoundRecorder

The test cases implemented for this class are sufficient because they cover every part of the basic functionality of the sound Recorder Program.

View



























insert test cases here

4. Test Coverage Discussion

Test Coverage is currently in progress. Test cases have been implemented for the data structure / backend classes of the application and will continue to progress.

Test Cases for ListManager class

Many of the test cases have been implemented. The ListManager class stores the data which is parsed from the scenario file. All the major functions are currently covered. Prev, next, remove methods test cases are outlined above and will be implemented soon.

ListManagerTest (Feb 22, 2018 10:56:11 PM)				
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▲  ListManager.java	 50.2 %	377	374	751
▲  ListManager	 50.2 %	377	374	751
● remove()	 0.0 %	0	127	127
● next()	 0.0 %	0	56	56
● prev()	 36.7 %	22	38	60
● getLabel(int)	 0.0 %	0	25	25
● ListManager(ListManager)	 0.0 %	0	22	22
● printString()	 0.0 %	0	22	22
● createJunction(HashMap<Integer, String>)	 90.3 %	121	13	134
● setIndex(int)	 0.0 %	0	11	11
● getData(int)	 63.0 %	17	10	27
● getKeyPhrase(int)	 60.0 %	15	10	25
● addNext(String, String)	 81.0 %	34	8	42
● goHome()	 0.0 %	0	8	8
● getNode()	 72.2 %	13	5	18
● junctionSearch(String)	 90.9 %	50	5	55
● getLabel()	 0.0 %	0	4	4
● getNextList()	 0.0 %	0	4	4
● getIndex()	 0.0 %	0	3	3
● junctionGoto(int)	 90.3 %	28	3	31
● ListManager(int, int)	 100.0 %	44	0	44
● getData()	 100.0 %	4	0	4
● getKeyPhrase()	 100.0 %	4	0	4
● printString(String)	 100.0 %	25	0	25

Test Cases for Node class

Major test cases for the Node class is implemented.

ListManagerTest (Feb 22, 2018 10:56:11 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Node	27.4 %	147	389	536
brailleToLetter(String)	0.0 %	0	226	226
Node(String, String)	19.6 %	37	152	189
setData(String)	0.0 %	0	4	4
setKeyPhrase(String)	0.0 %	0	4	4
getLabel()	0.0 %	0	3	3
button(String, ArrayList<Node>)	100.0 %	35	0	35
buttonNextt(ArrayList<Node>)	100.0 %	21	0	21
junction(ArrayList<ArrayList<Node>)	100.0 %	24	0	24
tail(ArrayList<Node>)	100.0 %	18	0	18
Node()	100.0 %	3	0	3
getButtons()	100.0 %	3	0	3
getData()	100.0 %	3	0	3
getKeyPhrase()	100.0 %	3	0	3

Overall Coverage (Work in Progress)

Overall coverage is not ideal at this time as the development process has meant that many classes have constantly changed. Test cases will be implemented in the near future as outlined in this document. We do not expect to implement full coverage as it is not feasible and is counter-productive because test cases are specific and limited while the application is complex and systems are coupled. Further testing must be done by running the application and testing features manually to ensure quality.

ListManagerTest (Feb 22, 2018 10:53:56 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Enamel	9.1 %	921	9,206	10,127
src	9.1 %	921	9,206	10,127
enamel	9.1 %	921	9,206	10,127
Controller.java	0.0 %	0	3,007	3,007
ScenarioParser.java	0.0 %	0	1,252	1,252
View.java	0.0 %	0	1,240	1,240
SoundRecorder.java	0.0 %	0	956	956
LoadParser.java	0.0 %	0	477	477
VisualPlayer.java	0.0 %	0	474	474
Node.java	27.4 %	147	389	536
ListManager.java	50.2 %	377	374	751
ScenarioComposer.java	0.0 %	0	369	369
BrailleCell.java	0.0 %	0	307	307
Player.java	0.0 %	0	171	171
FocusPolicy.java	0.0 %	0	70	70
ListManagerTest.java	87.8 %	397	55	452
RequestFocusListener.java	0.0 %	0	25	25
Main.java	0.0 %	0	24	24
ToyAuthoring.java	0.0 %	0	9	9
AudioPlayer.java	0.0 %	0	7	7