BRAILLE BOX AUTHORING APP

Group 9

Testing Document

JEREMY (214 915 854)
NISHA (213 251 830)
TYLER THOMSON (215 081 904)

Table of Contents

1.0 Introduction:

For the midterm submission, our group has created classes to enable the user to save, create, and edit files. As we created these classes, we created the corresponding test cases to make sure our program works as intended. However, our group has not created any test cases for the GUIs as we believe that our manual testing is sufficient. The classes given to us are assumed to be working and so we shall not provide test cases for these.

2.0 Data Classes:

2.1 Card:

The card class is for storing each question along with it's corresponding data. Most of the methods in card are very simple as most of them don't check any conditions and just set the data to the inputted value.

Card(int id, String name, String type) (constructor)

The card constructor was tested to make sure that each field of the card class was initialized correctly. This includes the id, name, type, lists, and text. This is also testing all of the get methods as we used these methods to receive the info.

setText(String text)

This method is used for replacing the text the card contains with the new text given. As it is a basic method which just sets the text to the given text, we had a few different inputs, along with empty and null strings, just to make sure it works correctly.

addText(String text)

The addText method is for if we want to add text onto the current text. We have inputs for checking if it adds text correctly if there is text already stored, if the text stored is null and if we try to add both null and empty strings.

setSound(String newSound)

This method is used for setting the sound of the card if the user wants to play a sound in the prompt. We tested inputting basic strings as well as null and empty strings and all of them worked as intended.

setBList(ArrayList<DataButton> newList)

Used to set the card's list of buttons to the given list. It does not make a copy of the list, it just switches it's reference to the new list. In this test we checked that changing the list after does affect the card's list and made sure it switches to a new list properly.

setCells(ArrayList<BrailleCell> newList)

setCells is very similar to the setBList method as it takes in a new list and sets the card's reference to the new list. The testing we did was also similar with testing a new list, updating that list, and switching to a new list

setName(String s)

This is the last method of card which just sets the new name to the passed in argument. This method was tested with basic, null, and empty strings to make sure it worked as intended.

Card has 100% test coverage. The testing done is sufficient as most of these methods are simple setter methods which just set the current value to the given value. So as long as we test these with a few inputs and the slightly more complicated methods more thoroughly, like we did, then these test cases are enough.

2.2 DataButton:

This class is similar to card as most of the methods are just basic getters and setters. This class does however have a copy constructor so we had to test that as well

DataButton(int id)

This constructor just sets the id to the passed in id and the strings to empty. The tester for this constructor also tests the getter methods as that is how it receives the data.

DataButton(DataButton other)

This is a copy constructor which creates a copy of the other DataButton. As the fields of this class are int and String, then it is fine that we are just setting each field to the other DataButton's value.

getID()

This method returns the id and was tested with a few different DataButtons as there is no setter for this field.

getText()

Returns the button's text and was tested with adding different text and checking the returned string.

getAudio()

Returns the button's audio file and was tested with setting the audio to different strings. A few random strings were passed in using setAudio just to make sure it works.

addText(String newText)

This method adds on text to what was already stored in the button. A few different strings were passed in to make sure it works consistently.

setAudio(String audioPath)

This method sets the audio to the passed in string. It doesn't check whether the string is a valid audio file, so we just passed in a few strings and compared them to what was returned.

setText(String newText)

Sets the text to the passed in value. Once again, a few random strings were passed in just to make sure it works.

DataButton has 100% test coverage. Since most methods are very basic methods which just sets the value stored to the given value, not much unexpected can happen so as long as we test with a few test inputs it is good enough. For the one conditional method both conditions were met so this is good as well.

3.0 Parsers

3.1 FileToCardParser (FTCP)

This parser is for reading a given text file and storing the data into a list of cards.

FileToCardParser()

The constructor was very simple as it was just initialized the card list, initial prompt, and ending prompt. Using the get cards, initial, and ending methods we checked if these fields were initialized.

checkNumLines(String scenarioFile)

Checks both the number of lines in the file as well as it checks where the first /~ line is (this line is called start and gets tested in testInitial). Both of these numbers are used later on. The number of lines is for when we reach the end of the file to add the last card onto the list and the first occurrence of /~ is used when separating the initial prompt. The functionality for both of these will be checked after. The tester for this method only checks whether the number of lines is correct. Two blocks of text were checked as well as an empty and null block of text which all returned the right value.

checkButtonsAndCells()

This method reads the first two lines and checks if they match “Cell #” and “Button #”. This method throws an exception if it doesn’t match. We tested with a basic input and made sure the exception was thrown for empty, null, and not matching inputs.

checkLast()

This method checks the last card in the list and if it only contains a prompt it is treated as the ending prompt and gets removed from the list. We tested this with reading the three given scenario files and each one worked as intended.

The rest of this section will go based off of the tester’s methods instead of class’s as the rest of the parsing is done in one method and was tested separately to make it simpler.

testInitial()

This method tests whether the correct initial prompt gets returned. It uses the start value defined in checkNumLines. It was tested with the three scenario files and each one returned the correct prompt.

testEnding()

This method tests whether the ending prompt received is correct. It was tested for the three given scenarios and works for scenario 2 and 3, however, it does not yet work for scenario 1.

testGetCards()

This method is for ckecking whether the general parser works overall. This is currently checking different parts of both scenario 1 and 2 to make sure the card text, button text, and cell states are all working as intended.

print() (in FTCP)

This method is just a method we used for manual testing purposes to make sure that the data was being displayed correctly and is not part of the actual program, so we believe no testing of this is necessary.

FileToCardParser only has 86% coverage, however, most of this is specific conditions and exception handling. The testing for this class is enough as all of the common functionality is thoroughly tested. Also a good indication is that this class works for the given scenarios as explained in 3.3.

3.2 CardsToFileParser (CTFP)

CardsToFileParser(ArrayList<Card> cards, int numButtons, int numCells, String initialPrompt,
String endingPrompt)

The constructor was for setting the cards, number of buttons, number of cells, as well as initial and ending prompts. Since the only get method this class has is getText(), we were only able to test if the text was empty as it should be. The other parts of this constructor are essentially tested within the other tests.

The rest of this section will go based off of the tester's methods instead of class's as the rest of the parsing is done in one method and was tested separately to make it simpler.

testCellButtonInitialEnding()

This method is to test whether the file correctly has the right number of cells and buttons and has the correct initial and ending prompts. We tested the CTFP with an empty card list and basic inputs for the rest to make sure these sections were being converted to a string correctly.

testWriteButtons()

This method is for testing to make sure that the buttons and all of it's data is being displayed correctly. First, we tested with a basic input, then input with a sound file, and finally testing if the display pin indicator properly gets converted to the corresponding text. Overall this tests all functionality of the buttons.

testCells()

This method tests whether the cells are properly converted into the text format. This method tests with 2/3 cells, with a new cell, and with a null cell.

testPrompt()

This method is for testing whether each card's prompt is properly being displayed. It is tested with null, empty, and basic prompts to make sure it works as intended.

testWriteCard()

This method is an overall tester for the CTFP class as it tests a combination of card prompt, cells, and buttons. For this we only have 1 test as each functionality was already tested separately.

CardsToFileParser has 96.8% coverage, but what's missing is either conditional or one for loop which gets tested when parsing the loaded files as explained below. Once again the testing for this class is good as the basic functionality is properly tested and this class works for the given scenarios as explained in 3.3.

3.3 Overall Parsers

This section is just to emphasize that a good indicator for us that the parsers were working as intended was that if you load any of the given scenarios using FTCP and then save that scenario using CTFP, it creates a file which looks quite similar to the original and works the same.

4.0 File Creator

4.1 ScenarioWriter

The scenario writer class is what we used to write to a file after we created the string representation using CTFP.

ScenarioWriter(String filePath)

Constructor for the scenario writer which sets the path to filePath. This constructor is used if we want to create a file. Our test case has a basic input which uses write() to create the file then checks if the created file has the same input.

ScenarioWriter(String filePath, Boolean appendVal)

This constructor is used if we want to add text onto a previously created file. Once again we have a basic input using write() and then we check to make sure it added the text onto the end.

write(String text)

Used to write the text to either the end or overwrite the text in a file depending on which constructor was used to instantiate the object. The tests we did for the constructors both used this method to write to the file and so this was tested in the previous tests.

getName()

Parses the path and returns the name of the file. We tested this with a basic input with a path and without a path and both of them worked.

ScenarioWriter has 100% test coverage. This testing is good enough because the functionality of everything was tested and this was used within other testing so we know it works.

5.0 GUIs

Our GUIs include Initial View, Scenario Form, Authoring Viewer, and Recorder Frame. All of our GUIs are only manually tested. After a quick manual test we can achieve 97.7%, 97.7%, 92.4%, and 84.7% code coverage, however we are constantly testing our GUIs and often these tests cover the scenarios which were not covered in this quick test.