Design Document

Connor Ahearn, Micah Arndt and Kevin Chan

Table of Contents

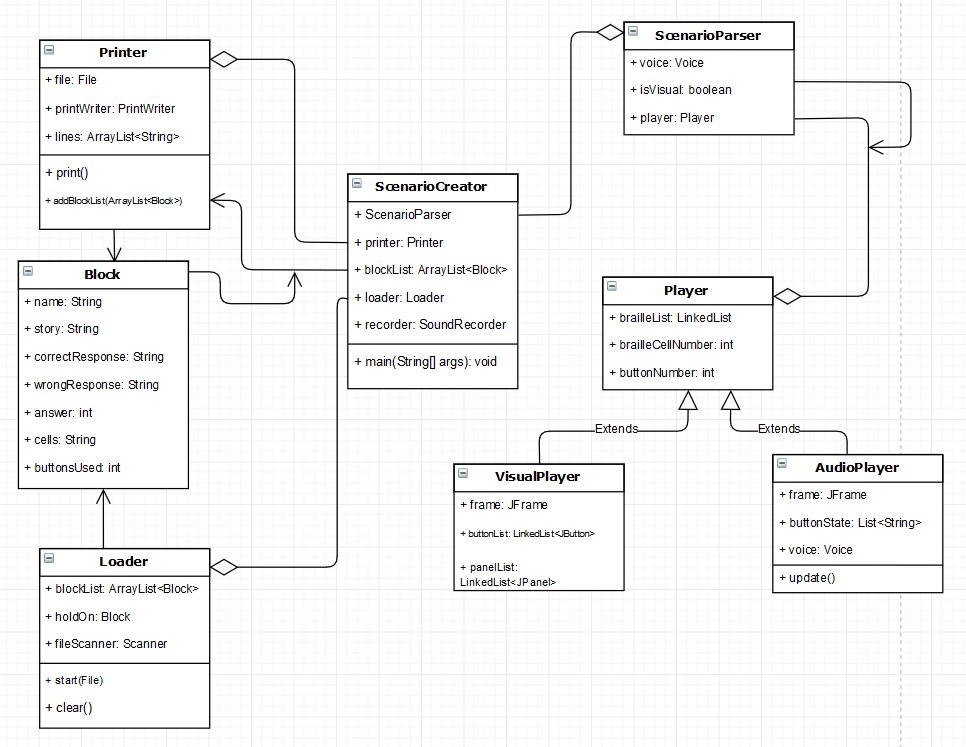
1. Introduction
2. Class Diagram
3. Sequence Diagram
4. Individual Classes
   1. AudioPlayer.java
   2. Block.java
   3. BrailleCell.java
   4. BrailleCellPanel.java
   5. BrailleInterpreter.java
   6. CorruptFileException.java
   7. InvalidBlockException.java
   8. InvalidCellException.java
   9. Loader2.java
   10. OddSpecialCharacterException.java
   11. Player.java
   12. Printer.java
   13. ScenarioCreator.java
   14. ScenarioParser.java
   15. SoundRecorder.java
   16. ToyAuthoring.java
   17. VisualPlayer.java
5. Maintenance Scenarios
6. Introduction

The purpose of this document is to describe the way the application operates at a high level. This is done using a class diagram and a sequence diagram.

The runtime of the application is much easier understood if you look at it from the perspective of **ScenarioCreator**. ScenarioCreator is the class with the main method, it has all the GUI’s for creating scenarios along with code to launch testing scenarios in our app. From start-up to close, if the Authoring App is running, ScenarioCreator is running. Through Aggregation, ScenarioCreator (As seen in the class diagram) uses every class in the application, although not always directly.

1. Class Diagram

The class diagram for the Authoring app can be seen below:



While looking at the diagram above, its important to note that some classes didn’t make it onto the diagram. These classes were omitted to make the diagram easier to understand and were deemed less important than the ones you’re seeing above. The omitted classes are listed below:

* BrailleCell.java
* BrailleCellPanel.java
* BrailleInterpreter.java
* SoundRecorder.java
* ToyAuthoring.java
* All Exceptions used throughout the program

Most of these classes are unessential to understand at a high level or are straight forward enough that they don’t need to be in the diagram.

The focus of the program can be seen in 3 main areas:

* ScenarioCreator – The runtime of the entire program runs through this
* Block – The data type universally used throughout the program’s working parts
* ScenarioParser – The Testing of scenarios is all done through this program

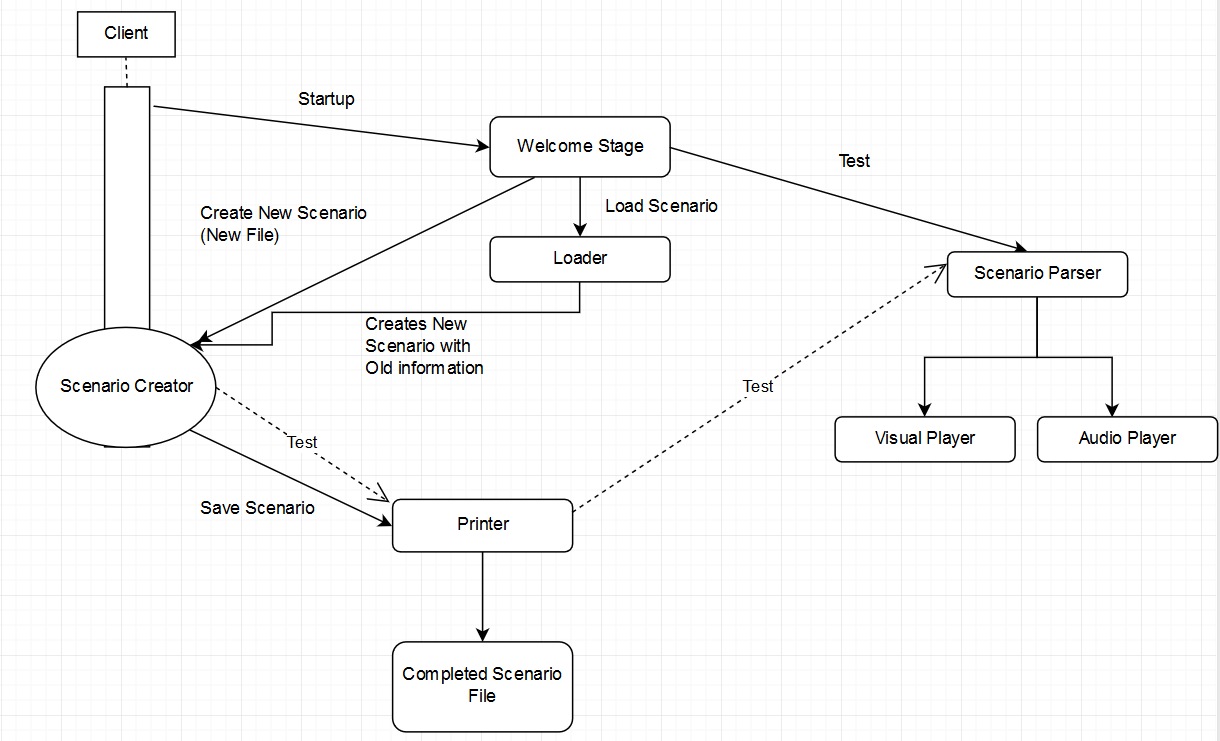
Blocks are the building block of scenarios in the application. The time frame of the box speaking up to the user pressing a button for the scenario is all stored in a block. This makes each scenario just a collection of blocks played in sequence by the braillebox. Having a standard datatype makes transferring data throughout the 3 main classes that use it (ScenarioCreater, Loader and Printer) much easier in practice.

ScenarioCreator was originally going to be the GUI used to create scenarios, but it evolved into being the central class of the application. All scenarios are written within its GUI, and advanced features and functions are completed via aggregation of other classes. Printer.java is used for creating new files, and Loader.java is used for loading existing files into ScenarioCreator.

ScenarioParser is central to testing scenarios. This code was provided to us by the professor.

1. Sequence Diagram

The sequence diagram of the Authoring App can be seen below.



The rectangle above / through Scenario Creator is the portion of the application that runs through that class that isn’t necessarily to do with scenario creation. It takes you to the welcome stage GUI where you’re given 3 options; New file, Edit file and Test file. Each of these options are arrows branching off the Welcome stage to the various areas of the program.

1. Individual Classes
   1. AudioPlayer.java

Audio Player is the class designed for visually impaired users to test scenarios. It does this by reading off extra information that the visual component does not, to better inform the user what the scenario is doing.

* 1. Block.java

Blocks are the standard data type throughout the application. It holds all the required information for a section of a scenario. This data type is then communicated between the Printer, Loader2 and ScenarioCreator.

* 1. BrailleCell.java

This was a provided class with the project, it helps display braille on the Visual Player.

* 1. BrailleCellPanel.java

This was a provided class with the project, it helps display braille on the Visual Player.

* 1. BrailleInterpreter.java

The Braille Interpreter takes braille settings / characters and translates to the opposite notation. Braille settings are saved as binary strings.

* 1. CorruptFileException.java

Corrupt File Exceptions were solely created for loading scenarios. If a scenario file is loaded and any part of the file isn’t recognized, this exception is thrown, and the file isn’t loaded. The Loader is specifically designed to load **our** authoring app’s scenarios, files created with other applications may not load properly. The message with the exception will describe the cause.

* 1. InvalidBlockException.java

Invalid Block Exceptions are thrown for when invalid data has been passed to a block constructor. The message with the exception describes the cause.

* 1. InvalidCellException.java

Invalid Cell Exceptions are thrown when the Braille Interpreter is used to translate a character that isn’t in the alphabet to Braille.

* 1. Loader2.java

Loader2 is the second iteration of the loader, and the one used in the final build for loading pre-made scenarios. It works for most scenarios, but some of the advanced functions related to custom sound and pin settings can be lost at times. The only function public for this utility class is called load() and it takes a File parameter.

* 1. OddSpecialCharacterException.java

Odd Special Character Exceptions are thrown by the printer when scenarios have an odd amount of special characters in a story section. Since special characters are used for advanced functions ( <, > and \* ) if there’s an odd amount, the printer can’t decipher what the user wants to happen in terms of custom sounds or pin settings.

* 1. Player.java

This is the parent class of both Visual Player and Audio Player. A lot of the function calls and processes related to testing scenarios come from this class.

* 1. Printer.java

The printer class is the class that converts a block list to a full scenario file. It’s called by the Scenario Creator, and the information put into the block list that’s passed to it are from the user input. The only methods called by ScenarioCreator are the constructor, addBlockList() and print(). print() is called once the scenario is finished, addBlockList is to send blocks to the printer. Typically both are called at the ”Save Scenario” button of the Scenario Creator.

* 1. ScenarioCreator.java

Scenario Creator is the center of the program. All classes are invoked in some way by Scenario Creator’s main method. Scenario Creator also houses the Authoring application itself, where users enter data to be printed into scenarios, or load information from old ones they made. This class has a massive amount of code in it due to being GUI code for the most part.

* 1. ScenarioParser.java

Scenario Parser is the point where the Player classes are called from. It has the capabilities to read scenario files. This class was provided for us.

* 1. SoundRecorder.java

This class is where new sound recordings are made from in the application. We recommend using another (better) application for this feature like Audacity, but the feature for sound recording is here.

The sound recorder requires a recording length to be set prior to beginning recording. After the constructor is called, the record() function provides sound recording capabilities.

* 1. ToyAuthoring.java

This class is designed to be a testing method for Scenario Parser. This class was provided for us.

* 1. VisualPlayer.java

Visual Player is the class used for visualizing what a Scenario would perform like once used with the Braille Box that we’re designing scenarios for. This class was provided for us.

1. Maintenance Scenario

Maintaining the authoring app is designed to be simple. This is accomplished through JavaDoc comments throughout the code, documentation such as the document you’re reading now and the organization / naming throughout the application.

New features to be added by a user are added through the Printer class. New features for loading are added through the Loader2 class. New features for sound recording are added to the SoundRecorder class. The majority of GUI changes would be made in ScenarioCreator, however things to do with testing scenarios would be found in Player, Visual Player, Audio Player and Scenario Parser.