# Introduction

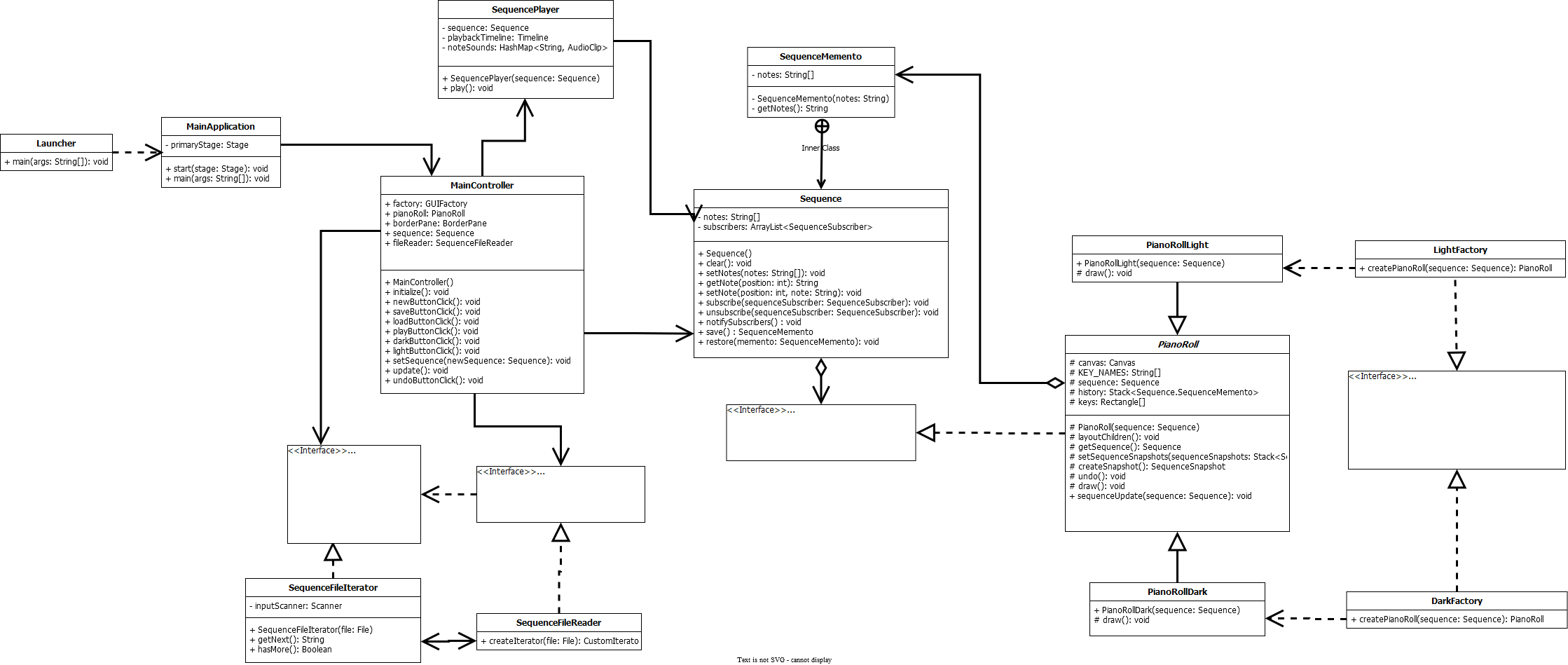
Sequence-o-matic is an attempt at creating a monophonic (i.e., one note at a time) software sequencer using JavaFX. It is composed of three key components:

* MainController
* Sequence
* PianoRoll

along with many other supporting components to implement all the necessary functionality.

Two major objectives of the project are: to create a reusable and easily extendable framework around which other similar application could be developed, and to practice using design patterns in a real-world application.

# UML diagram



UML Diagram

# Application startup

The application begins in Launcher, which is a dummy class whose only job is to provide a suitable entry point for running the application from a JAR. The real main() method resides in MainApplication, which calls the JavaFX launch() method which proceeds to call the start() method defined in MainApplication which sets the stage (in both a metaphorical and literal sense) for the rest of the program. After loading in the FXML for the program and creating a scene, MainApplication plays a short startup audio clip and then shows the stage, at which point the rest of the application starts.

# Main components

## MainController

Responsible for handling all user interaction with the application, except for the drawing of notes on the piano roll. Initializes all aspects of the application and sets up the publish-subscribe relationship between the Sequence and the PianoRoll.

## Sequence

Responsible for holding the sequence of notes and handling the notification of any subscribers on note changes. Plays a role in three design patterns, to be described in the section on the design of the application.

## PianoRoll

Responsible for containing the piano roll and responding to click events by modifying the Sequence. Also plays a role in the Memento design pattern. Its descendants, PianoRollLight and PianoRollDark, are responsible for actually drawing the piano roll.

# Design

There are four design patterns used in the implementation of the application: - Memento - Iterator - Abstract Factory - Observer

## Memento

The Memento pattern is used to store a history of Sequences to implement the “undo” functionality of the application. The role of Caretaker is fulfilled by the MainController class; the role of Originator is fulfilled by the Sequence class, which has an inner class named SequenceMemento which fulfils the role of Memento.

To use the implementation of the Memento pattern, call Sequence.save() on any sequence you wish to save and use Sequence.restore(memento) to restore the sequence from the memento.

## Iterator

The Iterator pattern is used to provide an easy way to iterate through the note entries in a file, which is done in MainController when the user clicks the “load” button and selects the file to be loaded. While reading text from a file is easy in Java, the usage of the Iterator pattern allows the possibility of future data sources (such as a binary file or a network service) to be added without having to modify much of the existing code.

The implementation of the pattern relies on two interfaces and two classes: - CustomIterator, the Iterator interface - IterableCollection, the IterableCollection interface - SequenceFileIterator, the ConcreteIterator - SequenceFileReader, the ConcreteCollection

To use the implementation, first instantiate a SequenceFileReader, passing in a Java File to open. Then, use the createIterator() method of SequenceFileReader to create a SequenceFileIterator. Finally, use the resulting iterator as you would any other iterator, bearing in mind that the hasMore() method returns a String.

## Abstract Factory

The Abstract Factory pattern is used to provide different themes for the application. Currently, there are two themes implemented: a light theme implemented by LightFactory and PianoRollLight, and a dark theme implemented by DarkFactory and PianoRollDark.

The implementation requires one abstract class, one interface, and three classes. The abstract factory is GUIFactory, and the concrete factories are LightFactory and DarkFactory. The one abstract product is defined in the abstract class PianoRoll, and the concrete products are PianoRollLight and PianoRollDark.

To use a specific theme, use either LightFactory.createPianoRoll() or DarkFactory.createPianoRoll() to create the corresponding type of PianoRoll which can then be used like any other JavaFX Pane.

The usage of this pattern is intended to make future themes easy to implement, and to allow theming other parts of the application in the future. For example, the sequencer could be extended to include a simple effects chain which would also support theming.

## Observer

The Observer pattern is used to allow interested parties to be notified about changes to a Sequence. Its implementation lies in three places: Sequence, SequenceSubscriber, and PianoRoll.

The SequenceSubscriber interface defines the interface that is used by Sequence to communicate with PianoRoll. Sequence is the Publisher and PianoRoll acts as a Concrete Subscriber. The unsubscribe() method that is usually found in a Publisher is intentionally omitted as it is unnecessary for the application, but could easily be added in the future if necessary.

To create another Concrete Subscriber, implement the SequenceSubscriber interface and call Sequence.subscribe(concreteSubscriber) somewhere in your program.