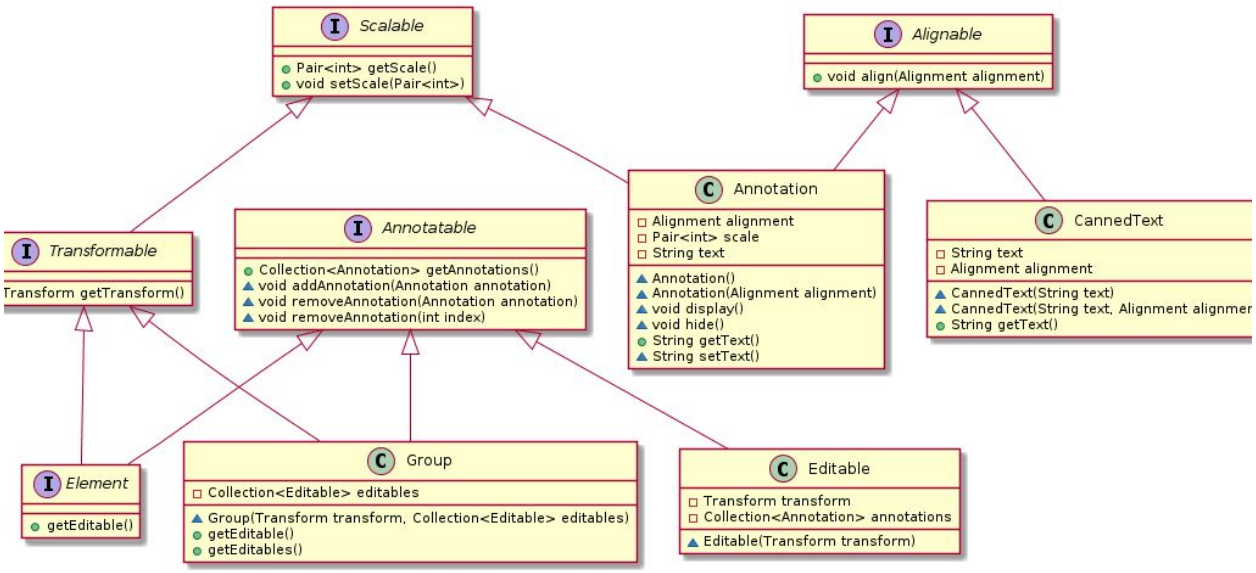
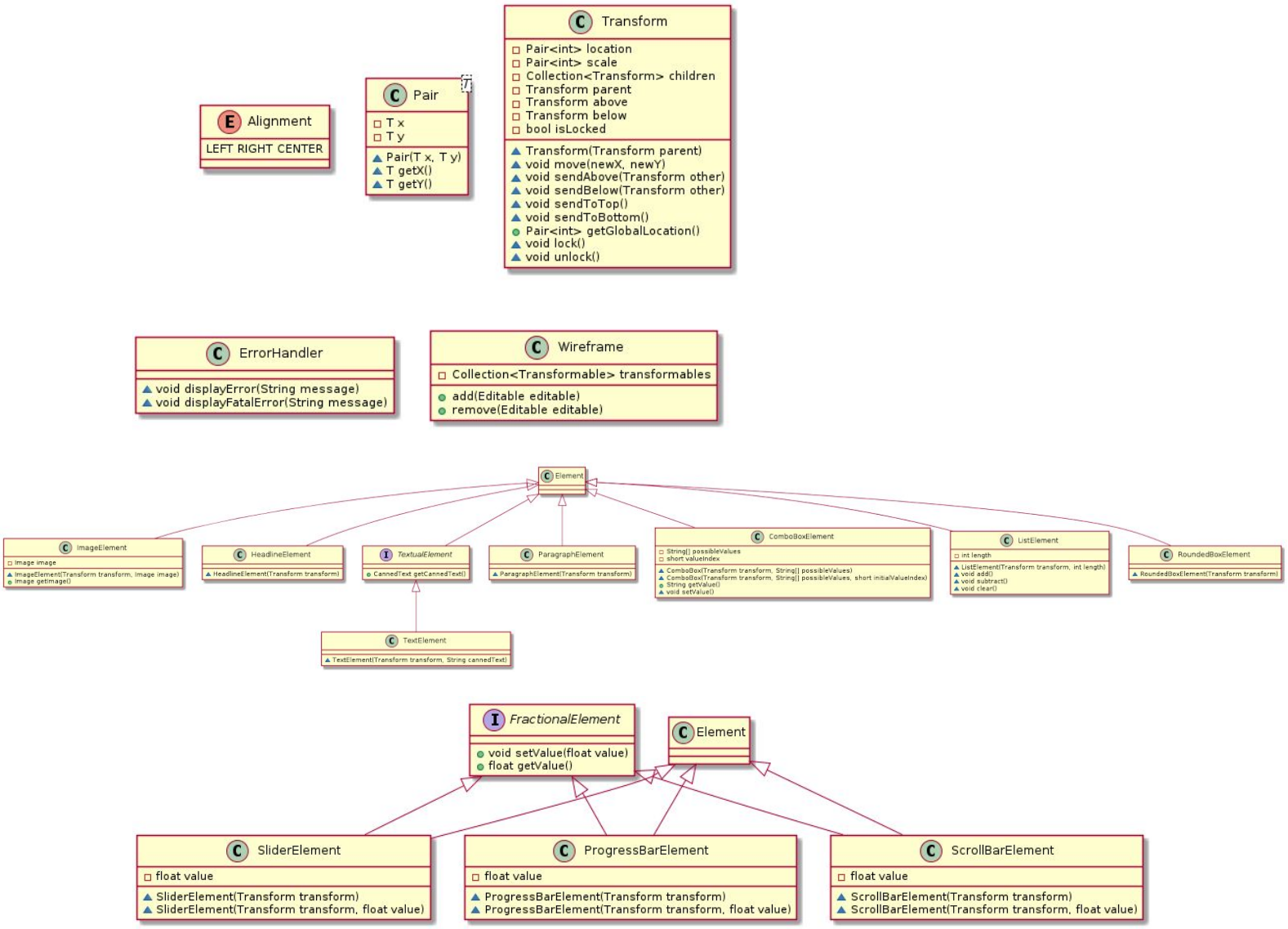
Brennan McFarland
November 15, 2017
Wireframe Design Document

Classes and Interfaces

The following UML class diagram describes the overall design architecture. The structure of the diagram should be sufficient to describe each class and interface's position in the inheritance hierarchy, as they are indicated by the inheritance arrows. To maintain readability, composition relationships are not illustrated with arrows, but are indicated by each class's fields. The graph also indicates public methods and private data structures, along with some other attributes with more restricted access modifiers subject to change in the implementation but are still included for clarity and as an aide in implementation.

I: interface C: class E: enum
Square: private access Triangle: package-private access Circle: public access





Scalable: An interface ensuring scalable behavior; an object that is scalable. The scale field is not part of the interface but likely to be a field in all implementing classes. The scale can be set or retrieved.

Alignable: An interface ensuring alignable behavior; an object that can be aligned.

Transformable: An interface ensuring transformable behavior; an object that can be transformed. Enforces the existence of a method to return a Transform associated with the implementing object.

Annotatable: An interface ensuring that annotatable behavior; an object that can be annotated. Enforces methods for adding to, removing from, and retrieving a collection of annotations.

Alignment: An enum for the alignment/justification of text or some other applicable object.

Pair: A pair, specifically of two related fields of the same type, for example x and y integer coordinates.

Annotation: An annotation, which contains mutable text and it alignable and scalable.

CannedText: A block of canned text. CannedText is alignable but its text is immutable, necessitating that the class be final.

Transform: A transform object, representing the position, scale, ordering, and any other geometric attribute of an object composed with a transform. This class handles the moving, ordering, locking and unlocking of an object. Any applicable transforms can check if the object is locked before performing any operation and do nothing if the object is locked. Location is stored relative to a parent transform, which is the same as the global transform if there is no parent. Ordering is done by changing the above and below references to other Transforms, following the chain of Transforms like a linked list if moving to the top or bottom. Ordering is nested by group, so any group members are all either above or below any non-group members.

ErrorHandler: The singleton object that handles errors. More details in the Error Handling paragraph below.

Wireframe: The wireframe, essentially the main/driver class for the module or application. Responsible for creating and performing operations on wireframe elements. The transformable collection holds the transforms of elements.

Group: A group of Groups and/or Elements. An annotation is annotatable and transformable, and it has a getter for its collection of Editables.

Editable: An editable is something that is transformable and annotatable. Groups and Elements contain an instance of an Editable, responsible for transform and annotation state.

Element: A wireframe element; the interface for all wireframe elements. Enforces that an element must include a method to get an associated Editable.

TextualElement: A textual wireframe element; the interface for all textual wireframe elements. Enforces that a textual element must include a method to get an associated CannedText object.

FractionalElement: A wireframe element based off a fractional, continuous value; the interface for all fractional wireframe elements. For example, a slider is a fractional element as the position of the slider is some fraction of all the way to the right, and similar logic applies to the scroll bar and progress bar. FractionalElement enforces public access to get or set the appropriate value.

For the sake of brevity, descriptions for the abstractions captured by the classes representing a specific wireframe element are omitted, as these abstractions were already described in the assignment. Since the aim of this architecture is to manage complexity by breaking it apart at the class and method level, no methods should be sufficiently complex to necessitate pseudocode.

Error Handling

All errors that is not a bug in the code will be handled by invoking the displayError() or displayFatalError() method in the ErrorHandler class, which is a singleton, and no exceptions will be thrown except by the Java libraries. These will be caught in the method that throws them and will result in an invocation of an ErrorHandler method. As most state is immutable, in most cases displayError() can be called to notify the user as failure atomicity is assured, and the program can continue. In exceptional cases, such as when an error occurs when propagating the child transforms of a group when, for example, moving it, displayFatalError() is called instead which will cause the program to halt. Arguments to constructors and non-private methods will be validated with assertions.

Testing

All private methods will be accessible from a nested test harness class, a description of which is omitted from the rest of this document to avoid redundancy. Methods will be unit tested with full branch coverage and an aim for covering as many good data, bad data and boundary cases as possible. Methods will be tested with the JUnit4 testing framework.