APPLICATION 1: IMAGE EVOLVER APP

**Initial Comments**

With this application, users can create images that are evolved by a multilevel process, where core components are created and composed into a tree structure of various types, and then decorators and filters can be applied to any part of that image.

Since the all-encompassing class ImageComponent implements the Mutatable Interface, it allows for multi-level overriding of key functions inside the interface, thus allowing for extreme customization of every aspect of the complex growing tree, from what kinds of filters will be used, the extremity of an applied filter, or which leaves (drawing blocks) these filters apply to. Reasonable defaults can be used inside abstract classes to provide functionality to all subclasses, whether an override of the interface is provided or not.

Since the application implements a more general interface for Image Creation, other apps can use the ImageEvolverApp to carry out an evolution from a starting set of objects, or create one from scratch. For instance, an app that utilizes the image building framework may be linked to a GUI that allows creation of these objects (ie, Photoshop). ImageEvolverApp can be an image modifying feature present within this application, a module that allows users to create a picture and then allow the TanzaniteEngine to carry out evolution on the image.

**Main Classes**

ImageEvolverApp

* The client of the TanzaniteEngine Framework
* A Facade to the underlying picture creation/representation framework

ImageComponent (abstract)

* Implements Mutatable
  + able to build itself randomly, mutate, swap subtrees (crossover), etc
* Utilizes Composite Pattern to construct itself

ImageLeaf (abstract)

* Represents different abstract ways to construct an image
  + Stencil drawing, shape building, position, etc,

ImageComposite

* Composite pattern realization, holds up to N ImageComponents

ImageDecorator (abstract)

* Capable of performing image enhancing modifications on any level of the composite tree
  + Contrast, Brightness, Hue/Saturation, Etc.

Settings (abstract)

* Holds information about how to decorate a subtree
* This exact information is beyond scope of application design (very complex), but will approximately represented using a byte array
* Utilizes Bridge Pattern to separate application of settings from the Decorator
* Settings are applied to ImageComponents

ImageComponentFactory (abstract)

* Used to create ImageComponents
* Allows for random creation, useful for evolution

ImageConverter (abstract)

* Chances are, clients don’t know how to see ImageComponents. So, converting and returning a common image file type is necessary via exporting.

ImageGatherer (abstract)

* Initial populations for evolution can be selected using various kinds of ImageGatherers

**Subclasses (self-explanatory)**

ImageLeaf (abstract)

* BrushStrokeLeaf (abstract)
* ShapeLeaf (abstract)
* GradientLeaf (abstract)
* etc.

ImageDecorator (abstract)

* BrightnessContrastDecorator
* HueSaturationDecorator
* LevelsDecorator
* CurvesDecorator
* FilterDecorator (abstract)
* etc.

Settings (abstract)

* BrightnessContrastSettings
* HueSaturationSettings
* LevelsSettings
* CurvesSettings
* FilterSettings (abstract)
* etc.

FilterDecorator (abstract)

* BlurFilter (abstract)
* RenderFilter (abstract)
* Artistic (abstract)

Usage