AcademiCS Application Framework

Specifications for the Second Set of Milestones and Deliverables

The second set of milestones/deliverables is concerned with both the application framework and the demonstration application (i.e., The Big Pixel). From the standpoint of the application framework, it is concerned with the basics of generic documents. From the standpoint of the demonstration application, it is concerned with the presentation of documents for The Big Pixel.

1. Glossary

Document	An object that is created/opened/edited by an application. In a business suite, examples include word processing documents, spreadsheets, drawings/illustrations, and presentations.
Document Manager	An object that controls the actions that can be performed on a document.
Document Factory	An object that can be used to create one or more documents.

2. Engineering Design

The relationships between the various classes that must be implemented for the first set of milestones/deliverables is illustrated in the UML class diagram (that is available as an SVG file). In addition to the specifications in that diagram, the classes/interfaces must comply with the following specifications.

2.1 The `PropertyConstants` Class

A utility class that contains a variety of constants related to `Property` values, `PropertyChangeListener` objects, and `PropertySupport` objects. These constants are kept here rather than in the classes that use them most frequently to avoid excess coupling (similar to the use of the `SwingConstants` class).

2.2 The `Configuration` Class

A class that can be used to store/read configuration information from a file. For simplicity, configuration files are stored in the same directory/folder as the application. This will cause problems if the application does not have write access to that directory/folder. Hence, in the future, we might need to include the path to the appropriate directory using a system property (e.g., set at run-time using the `-D` switch).

The `load()` method (which must be called in the constructor) must always read the configuration from a file named `current.cfg`. If there is no such file, or there is a problem reading that file, it must read from the file named `default.cfg` (which is guaranteed to exist because it will be part of the product deployment).

The accessors that include a default must return the default if the requested property does not exist. The accessors that do not include a default must throw a `NoSuchElementException` if the requested property does not exist.

2.3 The `Editable` Interface

The `Editable` interface describes the functionality of "document" objects. The parameter `D` is used to constrain the `DocumentManager`.

2.4 The `DocumentManager` Class

A `DocumentManager` object controls the actions that can be performed on a document (i.e., an `Editable`).

It has methods that fire property change events (using its `Support` attribute) when:

- A "new" document is activated;
- The "active" document is closed;
- A "new" document is created;
- The "active" document has been edited; and
- The `File` associated with the "active" document has changed (which must be confirmed before the event is fired).

where the document `d` passed to the method is "new" when it is not the same as the `document` attribute and it is "active" when it is the same as the `document` attribute.

It uses `PropertyConstants` to indicate which type of event has occurred when it invokes the `firePropertyChange()` method to inform its observers.

Note that when a "new" document is created the `DocumentManager` must subsequently activate it, and that when a "new" document is activated the `DocumentManager` must subsequently indicate that the `File` of the newly activated document has changed.

2.5 The `AbstractEditable` Class

The `AbstractEditable` class provides the functionality required by the Editable interface.

The abstract `getThis()` method is included in this class to deal with situations in which references to the owning object (i.e., `this`) must be type-safe (e.g., in calls to the `fireDocumentEdited()` method in the `DocumentManager` class).

The `notifyDocumentManager()` method must inform the `DocumentManager` that the document has been edited. It must be invoked by both the `setEdited()` and `setUnedited()` methods.

Note that, after an `AbstractEditable` is constructed, it must be in the *unmodified* state.

2.6 The `EditableFactory` Interface

The `EditableFactory` interface describes functionality of objects that can create `Editable` objects. The parameter `P` indicates the class of the object being created (i.e., the product of the factory) and the parameter `S` indicates the class of the source object (i.e., the input used to create the product).

Note that the methods in this interface use the word "Product" instead of "Instance" because "Instance" is used with the Singleton Pattern and some factories may be singletons.

Note also that, the product created by an `EditableFactory` must be in the *unmodified* state.

2.7 The `StringDocument` Class

A `StringDocument` is a simple `Editable` that can be used for demonstration and testing purposes. It is a simple wrapper around a String.

Note that, after a `StringDocument` is constructed, it must be in the *unmodified* state.

2.8 The `StringDocumentFactory` Class

A `StringDocumentFactory` can be used to create `StringDocument` objects (the product, `P`) from `String` objects (the source, `S`).

Note that, the product created by an `StringDocumentFactory` must be in the *unmodified* state.

2.9 The `ResourceLoader` Class

A `ResourceLoader` object can be used to retrieve resources (e.g., icons, images) from the file system or a `.jar` file. It uses a `ClassLoader` to find the resource.

It keeps a `Map` object for most types of resource so that it only needs to actually load the resource once.

The `loadedObject` passed to the various methods is used to "point to" the location of the resource (e.g., it might be an object in the same directory/folder as the resource).

2.10 The `BigPixelElement` Class

A `BigPixelElement` object is a colored rectangle that can be included in a `BigPixelDocument`. The `column` and `row` indicate the upper-left corner of the rectangle.

The default constructor must initialize all `int` attributes to `0` and all `Color` attributes to `Color.BLACK`.

The constructor that is passed a `String` must use the `String` representation to initialize the attributes. The String representation has six fields delimited by a `";"` character. The six fields correspond to the `column`, `row`, `width`, `height`, `strokeColor`, and `fillColor`. Each `Color` uses a comma-separated-value

representation. In the event of any problems with the `String` representation, the attributes must be initialized to the default values.

The `toString()` method must return a `String` representation (as discussed above).

2.10 The `BigPixelDocument` Class

A `BigPixelDocument` object is an `Editable` that consists of a collection of `BigPixelElement` objects.

The two `add()` methods must add a `BigPixelElement` to the collection. The only difference between the two is the way in which the element is "described". The method that is passed a `BigPixelElement` must add an alias (not a copy). The version that is passed multiple parameters must use the `brushSize` as the `width` and `height`, and the `currentColor` as both the `strokeColor` and `fillColor`.

The `toString()` method must return a `String` representation that consists of the `String` representations of all of its `BigPixelElement` objects, one per "line" (i.e., with a `"\n"` terminating each `String` representation of a `BigPixelElement`).

Note that, after a `BigPixelDocument` is constructed, it must be in the *unmodified* state. However, when a `BigPixelDocument` is modified in any way (e.g., after a call to `add()` or `clear()`) it must change its state to *modified*.

2.10 The `BigPixelDocumentFactory` Class

A `BigPixelDocumentFactory` object is an `EditableFactory` that can be used to create `BigPixelDocument` objects.

The String representation of a `BigPixelDocument` is described in the specification of its `toString()` method.

Note that, the product created by an `BixPixelDocumentFactory` must be in the *unmodified* state.

2.10 The `BigPixelEditor` Class

A `BigPixelEditor` is a `GridComponent` that responds to property change events. It will, in the future, be used to edit a `BigPixelDocument` object. This version can only display it.

The `setDocumentManager()` method must remove the owning object from the old `DocumentManager` (if there is one) and add the owning object to the given `DocumentManager`. This version need only listen for `PropertyConstants.DOCUMENT_ACTIVATED` events.

In response to such events (i.e., in the `propertyChange()` method) it must get the "new" `BigPixelDocument` in the `PropertyChangeEvent` it is passed.

It's `paint()` method must render all of the `BigPixelElement` objects in its `BigPixelDocument` and then render the grid (as appropriate). It must use a `GridConverter` to convert from cell coordinates to pixel coordinates.