AcademiCS Application Framework

Specifications for the First Set of Milestones and Deliverables

The first set of milestones/deliverables is concerned only with the demonstration application called The Big Pixel.

1. Glossary

Big Picture Element	A rectangular set of cells. A big picture element has an upper-left corner, a width and height (measured in cells), and a stroke and fill color.
Brush	A brush is used to "paint" a big picture element. At any point in time, it has a size and a color. The brush is "controlled" by the mouse.
Cell	A rectangular set of pixels (in the GUI). Each cell is 1/100th of the usable area of the component that renders it (i.e., 1/10th of the width and 1/10th of the height).
Grid	The outline of the cells in a picture.
Margin	Unusable space around (on all four sides) of the cells.
Picture	An ordered collection of big picture elements.
Usable Area	The pixels in a component not including the margins.

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 `CellID` Class

The `CellID` class is an encapsulation of the identifier for an entry in a table. It consists of a column "index" and a row "index", each of which can be of any type.

In some applications, the row and column "indexes" will be integers, however, one can imagine many situations in which this will not be the case. For example, the column indexes could be months and the row indexes could be salespeople. As another example, the column indexes could be letter grades and the row indexes could be course names.

The four-parameter constructor must throw an `InvalidCoordinateException` if either the given row or column parameter are invalid. The two-parameter constructor must not throw an exception.

The `setColumn()` and `setRow()` method must not change the associated attribute if the parameter is `null` or invalid.

The `toString()` method must return the column "index", followed by a comma, followed by the row "index". In other words, it must return the column "index" and row "index" formatted using the format string `"%s,%s"`.

2.2 The `CellConverter` Interface

A `CellConverter` has the ability to convert between the column,row "indexes" of a `CellID` and the x,y-coordinates of a pixel.

`fromPixels()` converts from an x,y-coordinate measured in pixels to the column,row of a `CellID`. It must throw an `InvalidCoordinateException` if the x,y-coordinate is not in any cell.

`toPixels()` converts from the column,row of a `CellID` to an x,y-coordinate measured in pixels. Since a cell is larger than a pixel, the result is the `Rectangle2D.Double` that the cell occupies (with upper-left corner at x,y). It must throw an `InvalidCoordinateException` if the column or row of the `CellID` is invalid.

2.3 The `ColorFactory` Class

The `ColorFactory` class is a utility class that is used to construct `Color` objects from `String` representations (and **vice versa**).

CSV representations have a format of `"%d,%d,%d"`.

Hex representations have a format of `"#%02x%02x%02x"`.

The three components must be in the closed interval [0,255] base-10.

When converting from either representation, if the components are out of range or missing an `IllegalArgumentException` must be thrown. In addition, when converting from Hex, an `IllegalArgumentException` must be thrown if the `String` does not begin with a "#".

2.4 The `GridConverter` Class

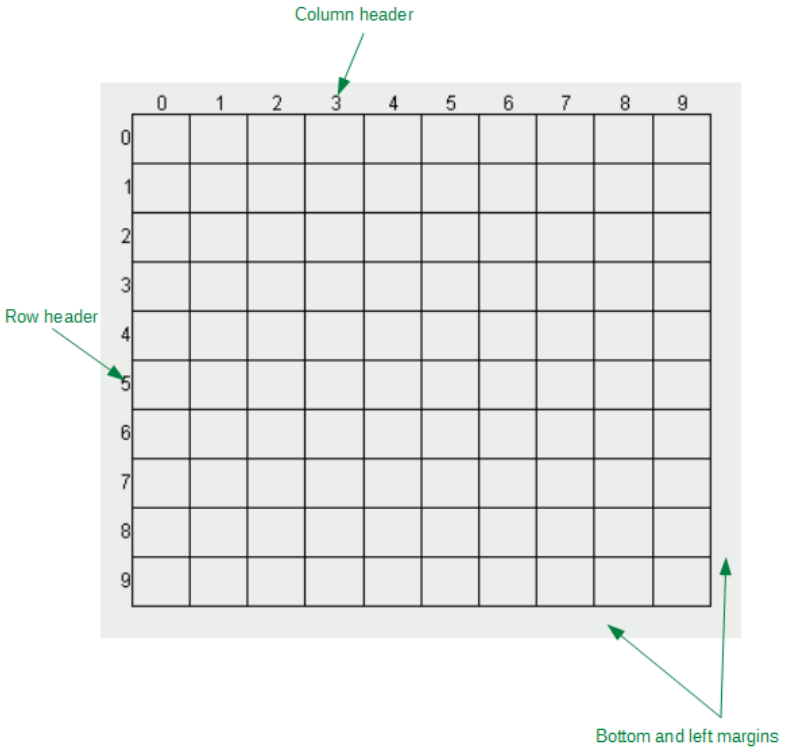
A `GridConverter` is a `CellConverter` for cells that have a `CellID` in which both the column "index" and the row "index" are `Integer` objects.

The explicit value constructor has four parameters. `columns` and `rows` contain the number of columns and rows in the grid, respectively; `pixels` contains the width and height of the entire grid (in pixels); and `margin` contains the number of pixels in the margin around the cells (on all four sides).

For example, consider a 5x5 `GridConverter` that is 500 pixels wide and 500 pixels high with no margin. Its `toPixels()` method must convert the `CellID` at 2,3 to a rectangle that has an upper left corner of 200,300 and a width and height of 100.

2.5 The `GridComponent` Class

A `GridComponent` is a `JComponent` that renders a grid with column and row "indexes" that are both `Integer` objects. It may or may not include column/row headers (containing the "indexes") and margins.



The default constructor must initialize the columns, rows, and margin to 10, 10, and 20, respectively.

The `getMinimumSize()` method must assume that cells are 10 pixels by 10 pixels. The `getPreferredSize()` method must assume that cells are 30 pixels by 30 pixels.

The `paint()` method must render the grid and the headers. It may use other private methods for these purposes. If they are to be rendered, it must render the column and row headers in the top and left margins. Each column label must be centered above the relevant column, just above the top grid line. Each row label must be centered just to the left of the relevant row and must be aligned flush right.

Note that this method will need to use the `toPixels()` method in a `GridConverter` object but it must never pass it invalid parameters. Hence, it will never need to handle an `InvalidCoordinateException` so **it must use an assertion to ensure that this is the case during testing.** When constructing the `GridConverter`, it must use its current size (obtained using `getSize()`) as the `Dimension`.

The `setGridColor()`, `setGridVisibility()` and `setHeaderVisibility()` methods must call `repaint()` after they change the object's state.