# MonthlyAgendaDisplay.java

```
 1  package caltool.view.view;
 2
 3  import caltool.model.schedule.*;
 4  import caltool.model.view.*;
 5  import caltool.view.*;
 6  import mvp.*;
 7  import java.util.*;
 8  import java.awt.*;
 9  import javax.swing.*;
10
11
12  /****
13   *
14   * Class MonthlyAgendaDisplay is the companion view of a MonthlyAgenda model.
15   * The fixed layout of the display is a three-part vertical box.  The box
16   * contains a date banner, a row of header labels for the days of the week, and
17   * the seven-column grid for the days of the month.  The size and layout of the
18   * days grid is computed dynamically by the update method, based on the data
19   * from the model.
20   *
21   * @author Gene Fisher (gfisher@calpoly.edu)
22   * @version 13apr15
23   *
24   */
25  public class MonthlyAgendaDisplay extends mvp.View {
26
27      /**
28       * Construct this by constructing subpanels for the three parts of the
29       * display.  Also construct an array of 31 day displays.  This array
30       * provides direct access to the individual day displays by date number,
31       * which is handy for referencing the companion day models directly.
32       *
33       * Compute the default size of the days grid to be 5 rows by 7 columns of a
34       * default-size day display.  If there are 4 or 6 rows, then the default
35       * rows are taller or shorter than they are wide.  This formating is per
36       * the requirements.
37       *
38       * Initialize the displayedOnce flag to false.  The display is only set to
39       * the default size the first time it is displayed.  After that, the
40       * display retains its size, including any resizing done by the user.
41       */
42      public MonthlyAgendaDisplay(Screen s, MonthlyAgenda monthlyAgenda) {
43          super(s, monthlyAgenda);
44          days = new SmallDayViewDisplay[31];
45          dateBanner = new JPanel(new GridLayout(1, 1));
46          daysOfWeek = new JPanel(new GridLayout(1, 7));
47          dayGrid = new JPanel(new GridLayout(0, 7));
48          dayGrid.setBackground(Color.white);
49          defaultSize = new Dimension(
50              7 * defaultCellWidth, 5 * defaultCellHeight);
51          displayedOnce = false;
52      }
53
54      /**
55       * Compose this as a vertical box, consisting of a date-banner row, a
56       * days-of-the-week labels row, and an empty days grid.  The grid will be
57       * populated by update.
58       */
59      public Component compose() {
60
61          /*
62           * Make a new window for this.
63           */
64          window = new mvp.Window();
65
66          /*
67           * Make an outer box.
68           */
69          vbox = new JPanel();
70          vbox.setLayout(new BoxLayout(vbox, BoxLayout.Y_AXIS));
71          vbox.setBorder(BorderFactory.createLineBorder(Color.black));
72
73          /*
74           * Compose the top two rows.
75           */
76          JPanel bannerBox = composeDateBanner();
77          JPanel labelBox = composeDaysOfWeek();
78
79          /*
80           * Add the date banner and days-of-week labels to the outer vbox.
81           */
82          vbox.add(bannerBox);
83          vbox.add(labelBox);
84
85          /*
86           * Add the empty day grid to the vbox.  It will be populated by update.
87           */
88          vbox.add(dayGrid);
89
90          /*
91           * Add the vbox to the window and we're outta here.
92           */
93          window.add(vbox);
94          window.setTitle("Monthly Agenda");
95          return window;
96      }
97
98      /**
99       * Compose the date banner.  For now it's a dummy label.  In the full
100      * implementation, it will contain prev,next,today buttons and the
101      * current month/year.
102      */
103     protected JPanel composeDateBanner() {
104         JLabel bannerLabel = new JLabel(((MonthlyAgenda)model).
105             getFullMonthName());
106         JPanel bannerBox = new JPanel();
107
108         bannerBox.setLayout(new BoxLayout(bannerBox, BoxLayout.Y_AXIS));
109         bannerLabel.setForeground(Color.black);
110         bannerLabel.setFont(bannerLabel.getFont().deriveFont(Font.BOLD));
111         bannerLabel.setHorizontalAlignment(SwingConstants.CENTER);
112         dateBanner.add(bannerLabel);
```

```
113         dateBanner.setMaximumSize(new Dimension(
114             2000, 2 * bannerLabel.getFont().getSize()));
115
116         bannerBox.add(Box.createVerticalStrut(4));
117         bannerBox.add(dateBanner);
118         bannerBox.add(Box.createVerticalStrut(4));
119         bannerBox.setBorder(BorderFactory.createLineBorder(Color.black));
120
121         return bannerBox;
122     }
123
124     /*
125      * Compose the days-of-the-week labeling row.  It's a 1x7 grid of labels,
126      * so they'll align properly with the columns of the day grid
127      */
128     JPanel composeDaysOfWeek() {
129         JLabel dayLabel = new JLabel("");
130         JPanel labelBox = new JPanel();
131
132         labelBox.setLayout(new BoxLayout(labelBox, BoxLayout.Y_AXIS));
133         for (int dayNumber = 0; dayNumber < 7; dayNumber++) {
134             dayLabel = new JLabel(DayName.values()[dayNumber].toString().substring(0,3));
135             dayLabel.setHorizontalAlignment(SwingConstants.CENTER);
136             dayLabel.setForeground(Color.black);
137             dayLabel.setFont(dayLabel.getFont().deriveFont(Font.BOLD));
138             daysOfWeek.add(dayLabel);
139         }
140         daysOfWeek.setMaximumSize(new Dimension(
141             2000, 2 * dayLabel.getFont().getSize()));
142
143         labelBox.add(Box.createVerticalStrut(4));
144         labelBox.add(daysOfWeek);
145         labelBox.add(Box.createVerticalStrut(4));
146         labelBox.setBorder(BorderFactory.createLineBorder(Color.black));
147
148         return labelBox;
149     }
150
151     /**
152      * Display the model data in the appropriate daily positions.  The data are
153      * produced by firstDay and nextDay iterator methods.  The display is a
154      * 7-column grid, with 4, 5, or 6 rows, depending on the configuration of
155      * the month.
156      *
157      * In the current implementation, the display is fully redrawn at each call
158      * to update, with no display efficiencies implemented.  Possible display
159      * efficiencies that might be implemented include the following. (1) If the
160      * model data have not changed at all, no updating is performed.  This is
161      * the presumably rare case where the user has executed a 'Goto Date'
162      * command for the current month.  (2) If the number of weeks in the new
163      * model month is the same as the current model month, the row boxes are
164      * not reallocated.
165      */
166     public void update(Observable o, Object arg) {
167         int row = 0;            // Week row number
168         int dayPosition;        // Ordinal position 0-41 of the current day
169         int i;                  // Loop index
170         SmallDayViewDisplay     // Loop var for each day's display
171             dayViewDisplay;
172         int numberOfWeeks =     // Number of weeks === number or rows
173             ((MonthlyAgenda)model).getNumberOfWeeks();
174         Dimension curSize =     // Current x/y size of grid
175             vbox.getSize();
176         Dimension cellDimension // Size of one day cell
177             = new Dimension(
178                 (int) (curSize.getWidth() / 7),
179                     (int) (curSize.getHeight() / numberOfWeeks));
180
181         /*
182          * Clear everything out and set the number of rows to the number of
183          * weeks in the model month.
184          */
185         Arrays.fill(days, null);
186         dayGrid.removeAll();
187         GridLayout layout = (GridLayout) dayGrid.getLayout();
188         layout.setRows(numberOfWeeks);
189
190         /*
191          * Put empty grey boxes up to the first day position.
192          */
193         SmallDayView dayView = ((MonthlyAgenda)model).getFirstDay();
194         dayPosition = dayView.getDay().ordinal();
195         for (i = 0; i < dayPosition; i++)
196             dayGrid.add(greyDay());
197
198         /*
199          * Populate the individual day displays with model data.
200          */
201         for (; dayView != null;
202                 dayView = ((MonthlyAgenda)model).getNextDay(), dayPosition++) {
203             dayViewDisplay = new SmallDayViewDisplay(
204                 screen, dayView, (MonthlyAgenda)model, cellDimension);
205             days[dayView.getDate()] = dayViewDisplay;
206             dayGrid.add(dayViewDisplay.getWidget());
207         }
208
209         /*
210          * Put empty grey boxes up to the last day position in the last row.
211          */
212         for (i = dayPosition; i % 7 != 0; i++) {
213             dayGrid.add(greyDay());
214         }
215
216         /*
217          * Set grid to the default size if this is the first time it's being
218          * displayed.  Otherwise, leave its size as it was.  In either case,
219          * pack the grid in order to "burn in" the layout.
220          */
221         window.getContentPane().setBackground(Color.blue);
222         if (! displayedOnce) {
223             dayGrid.setPreferredSize(defaultSize);
224             displayedOnce = true;
```

```
225             window.pack();
226         }
227     }
228
229     /**
230      * Build an empty grey-background, black-border day display.  A fresh one
231      * of these needs to be allocated for each use since JFC doesn't play
232      * reuses of a components in containers.
233      */
234     protected JPanel greyDay() {
235         JPanel panel = new JPanel();
236         panel.setBackground(Color.lightGray);
237         panel.setBorder(BorderFactory.createLineBorder(Color.black));
238
239         return panel;
240     }
241
242     /** Array of day displays for convenient access by date number.  This array
243         contains references to the same day-display objects that are laid out
244         in the day grid. */
245     protected SmallDayViewDisplay days[];
246
247     /** Outermost box of the laid-out display. */
248     protected JPanel vbox;
249
250     /** The date banner at the top of the display. */
251     protected JPanel dateBanner;
252
253     /** The days-of-the week labeling row. */
254     protected JPanel daysOfWeek;
255
256     /** The day grid. */
257     protected JPanel dayGrid;
258
259     /** Number or weeks (hence display rows) in the current display. */
260     protected int numberOfWeeks;
261
262     /** Flag that's true after the display has bee shown the first time. */
263     boolean displayedOnce;
264
265     /** Initial default size of the day grid. */
266     protected Dimension defaultSize;
267
268     /** Default constant for the height of one day display cell. */
269     protected final int defaultCellHeight = 75;
270
271     /** Default constant for the width of one day display cell. */
272     protected final int defaultCellWidth = 75;
273
274 }
```