JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2009**

Vol. 8, No. 1, January-February 2009

EDUCATOR'S CORNER

A Model-View Implementation of Linked Custom Grids in C#

Richard Wiener, Editor-in-Chief, JOT, Chair, Department of Computer Science, University of Colorado at Colorado Springs

1 INTRODUCTION

In our department's "Advanced Object Technology Using C#/.NET" course, a 300-level course, I spend some time discussing model-view design using delegate and event types. As is typical in such courses, time constraints as well as the student's backgrounds limit the size and scope of the examples that are presented in class and discussed.

To bring the concepts of model-view implementation to life, I have designed a major project that requires our students to delve much deeper into this design paradigm. A non-trivial grade book application that features multiple custom grids that may be linked provides a compelling opportunity for the students to exercise the model-view design principles studied in class.

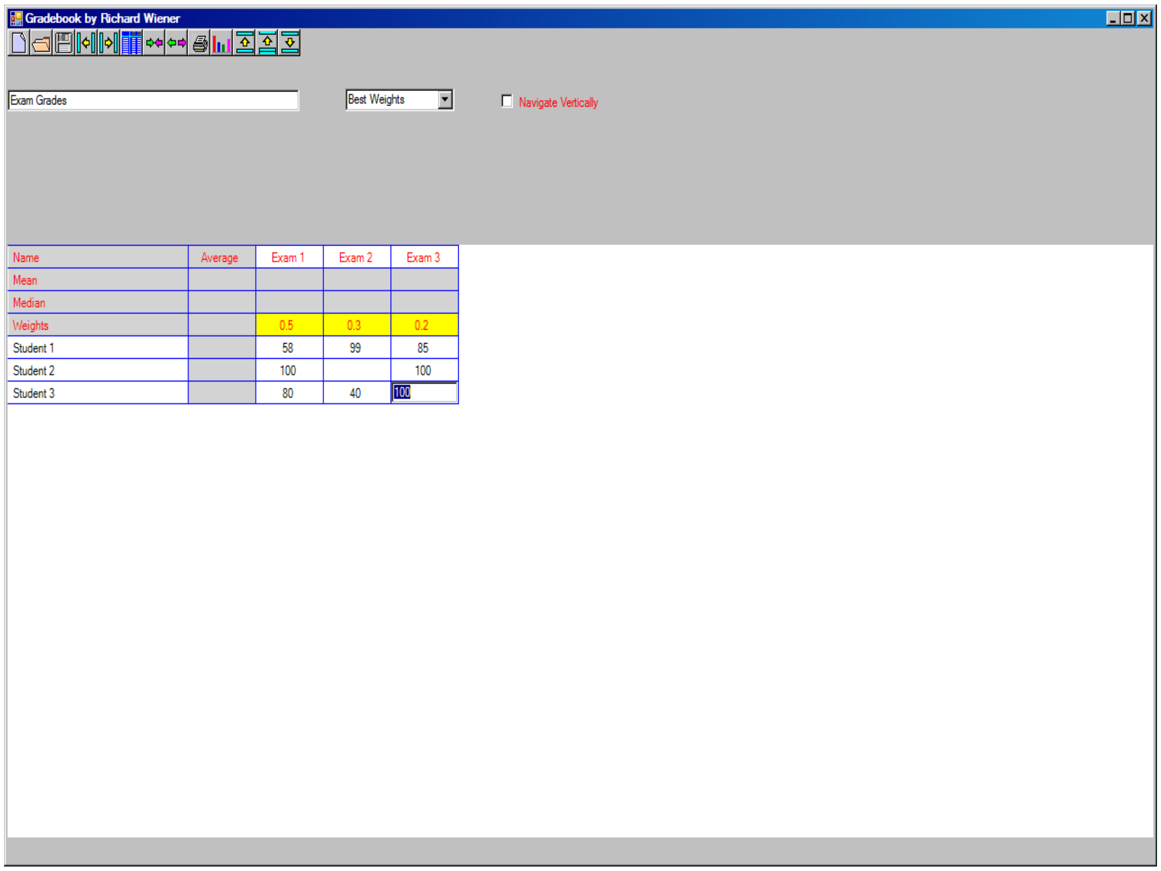
In order to jump-start their work I provide the students with a stripped down simple one-grid implementation that establishes a simple framework of model-view design that they must significantly expand in order to meet the requirements of the application.

In this paper the requirements of this project are presented, a few highlights of the framework that is provided to the students are outlined and the challenges for future maintenance upgrades are described.

2 HIGH-LEVEL REQUIREMENTS

A grade book is to be implemented in C# using a model-view approach to design. Class *GradeBookUI* is used to implement the view (GUI). Class *Gradebook* is used to implement the model (stores grades and perform computations on these grades when needed by the view and later provides the basis for linking grade book grids).

A screen shot of the GUI with three students and three grades is given below.


Gradebook by Richard Wiener

Exam Grades

Best Weights

☐ Navigate Vertically

Name	Average	Exam 1	Exam 2	Exam 3
Mean				
Median				
Weights		0.5	0.3	0.2
Student 1		58	99	85
Student 2		100		100
Student 3		80	40	100

The user can click any cell (other than ones colored gray) and then navigate around the grid using the up, down, left and right arrow keys. The Return key behaves like a right arrow key except if the "Navigate Vertically" box is checked. Then it acts like a down arrow key. It is illegal for any grayed cell to be selected or reached through navigation.

String data is entered by typing text into the cell selected by the mouse and either clicking another cell or moving away from the current cell with the Return key or one of the navigation keys.

In the left-most column (the column that holds names), any characters may be typed. In any of the columns that expect numerical input, the string is erased if it does not conform to the format of a numeric value.

An implementation of a stripped down and limited grade book with only one grid is provided containing classes *GradeBookUI*, *Gradebook*, and *Global*. Two additional classes complete this simple implementation (*GradeBookUI.Designer* and *Program*). Graphics (.gif) files are provided for the buttons that are used. Tooltips are implemented only for the buttons that are activated in this simple one-grid implementation.

3 DESIGN

The challenge in implementing the GUI is determining how to allow user input and navigation around the grid. One's first temptation might be to create a matrix of *TextBox* controls. This would allow user input from any cell in the grid. The problem with such an approach is its cost. Active controls such as a *TextBox* consume significant resources and perform inefficiently. In a large grid there could be hundreds of controls.

The approach used here is to have only one active *TextBox* control (field *textbox* in class *GradeBookUI*). The rest of the grid is provided as panel graphics. This includes the horizontal and vertical lines, and all the String data that is displayed. These graphics are written to a *Panel* component with white background color. When the user clicks the mouse in a cell or navigates from one cell to another with arrow keys, the *TextBox* control is moved to the location in the panel that corresponds to the cell that has been selected. User input can be obtained through this roving *TextBox* control. When the input is completed by navigating to another adjoining cell or clicking the mouse in another cell location, the *String* data, if legal, is provided to the *Gradebook* object stored as a field of class *GradeBookUI*. Through this mechanism, the model is updated. An indexer with two parameters is provided in the *Gradebook* class to accomplish this.

A file *Delegates.cs* is defined that initially houses four delegate types. These provide the basis for the event types that provide the conduit of communication between each of the *Gradebook* models and their respective *GradeBookUI* views.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Gradebook {

    public delegate void DisplayData(String [,] data);
    public delegate void DisplayCellData(String content,
                                         int row, int col);
    public delegate void DrawGrid(int numberStudents,
                                         int numberGrades);
    public delegate void PlaceTextBox(int row, int col, String
text);
}
```

The students will need to provide additional delegate classes later as they complete this project.

To illustrate the use of this communication consider the implementation of the button handlers for adding a new student (i.e. a new row to the grid) and adding a new column to the grid in the implementation of the simple grid and model provided to the students.

The class *Global* contains a few public static objects that are accessible throughout the application. This includes *NumberStudents* (the same on every grid).

```
private void AddRowClick(Object sender, EventArgs e) {
  Global.NumberStudents++;
   gradebook.FireDrawGrid();
}

private void AddColClick(Object sender, EventArgs e) {
  gradebook.NumberGrades++;
   gradebook.FireDrawGrid();
}
```

The command to *FireDrawGrid*, sent to the *gradebook* model, causes the *drawGrid* event to be activated in the *gradebook* object. This event is listened to in the *GradeBookUI* class in method *DrawGrid*.

A key event handler in *GradeBookUI*, the handler for mouse events, is given as:

```
private void MouseClickHandler(Object sender, MouseEventArgs
e) {
  int origX = gradebook.ColClick; // The last column clicked
   int origY = gradebook.RowClick; // The last row clicked
   String content = textbox.Text;

   int x = e.X;
   int y = e.Y;
   int row, col;
  if    (x    >    NAME_WIDTH    +    CELL_WIDTH    *    (1    +
gradebook.NumberGrades) ||
                y > CELL_HEIGHT * (4 + Global.NumberStudents))
{
        row = -1;
      col = -1;
      return;
  } else {
        row = y / CELL_HEIGHT;
      if (x <= NAME_WIDTH) {
        col = 0;
      } else {
        col = 1 + (x - NAME_WIDTH) / CELL_WIDTH;
      }
    }
```

```
     if (row == 1 || row == 2 || col == 1 || (col == 0 && row
<= 3)) {
          return;
     }
     if (e.Button == MouseButtons.Right) {
          textbox.Visible = false;
          gradebook.Sort(col);
          gradebook.FireDisplayData();
     } else {
          gradebook.ColClick = col;
          gradebook.RowClick = row;
          gradebook.FirePlaceTextBox();
          if (origY == 3 && origX >= 1) {
               DrawRectangle(Color.Yellow, origY, origX);
          }
          gradebook[origY, origX] = content;
          gradebook.FireDisplayCellData(origX, origY);
      }
}
```

If the right mouse button is clicked, the columns are sorted. If the name column (leftmost column) is right-mouse clicked, the grid is sorted based on alphabetizing the names. If any other column is right-mouse clicked the grid is sorted from largest to smallest (descending order) according to the selected column. The implementation of the *Sort* method in class *Gradebook* is given below. It uses LINQ and is quite elegant (C# 3.0 under Visual Studio 2008 is required for LINQ to be supported). The challenge is that the model contains a two-dimensional array of base-type String with column zero containing names and all other columns containing string representations of numeric values. An additional complication is that some cells in this two-dimensional array are blank.

```
public void Sort(int sortOnCol) {
   // Load up rollbook
    String [][] rollbook = new String[Global.NumberStudents][];
   for (int row = 0; row < Global.NumberStudents; row++) {
          rollbook[row] = new String[NumberGrades + 2];
        for (int col = 0; col < NumberGrades + 2; col++) {
          if (sortOnCol == 0 || data[row + 4, col] != null &&
                                   data[row        +        4,
col].Trim().Length > 0) {
                 rollbook[row][col] = data[row + 4, col];
          } else {
                 rollbook[row][col] = "-1000.0";
            }
          }
   }
    if (sortOnCol == 0) { // Ascending order
```

```
                var sortedMatrix =
                from value1 in rollbook
                  orderby value1[sortOnCol]
                  select value1;
                  // Copy results back to data
                        int r = 0, c = 0;
                      foreach (String[] row in sortedMatrix) {
                        c = 0;
                          foreach (String value in row) {
                                  data[r + 4, c] = value;
                              c++;
                          }
                          r++;
                      }
        } else { // descending order
                var sortedMatrix =
                from value1 in rollbook
                  orderby           Convert.ToDouble(value1[sortOnCol])
     descending
                  select value1;
                  // Copy results back to data
                  int r = 0, c = 0;
                  foreach (String[] row in sortedMatrix) {
                        c = 0;
                      foreach (String value in row) {
                        if (!value.Equals("-1000.0")) {
                              data[r + 4, c] = value;
                          } else {
                              data[r + 4, c] = "";
                          }
                          c++;
                        }
                      r++;
                }
        }
    }
```

The static query, *Convert.ToDouble(value1[sortOnCol)*, is used in the LINQ script for descending in order to ensure that values are compared by numeric value rather than string value (e.g. the String “4” is larger than the String “20”). In order to ensure that the value is in the correct format (of a “double” string), the string “-1000.0” is inserted temporarily in the *rollbook* matrix wherever a null or blank cell is found. Then these values are filtered out when the results are copied back to the *data* matrix.

There are four event fields defined in the *Gradebook* class. Each of these fields has a “Register” method that allows a listener in the *GradeBookUI* class to connect itself to the appropriate event.

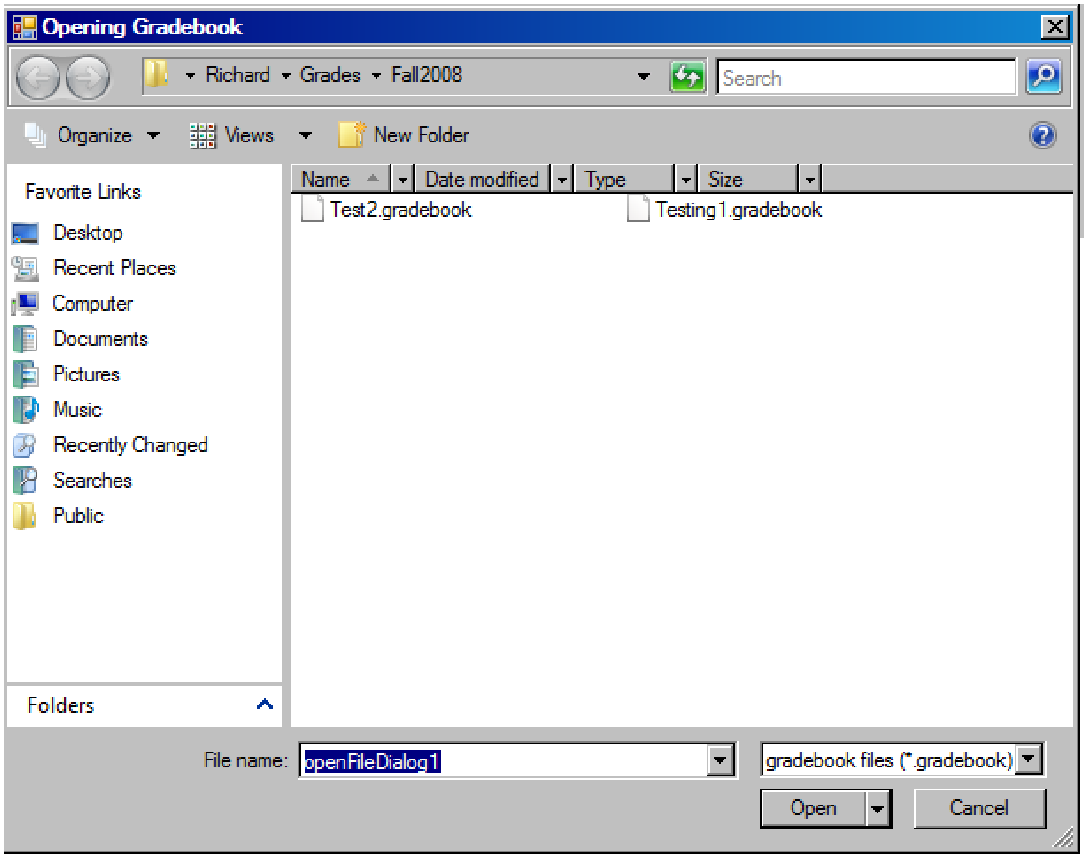
4 MORE COMPLETE GRADE BOOK SPECIFICATIONS

The specifications for the more complete grade book that the students are to implement are given in this section.

The functionality that must be implemented is triggered by the following buttons (each with a live tool tip):

Open file …

When the "Open file …" button is clicked, a modal dialog appears as follows:



Test2.gradebook
Testing1.gradebook

File name: openFileDialog1

gradebook files (*.gradebook)

This allows navigation to any sub-directory. Only files with the suffix, "gradebook" (e.g. Test2.gradebook) are displayed. A choice is made by double-clicking a file name.

One may create a new grade book by clicking anywhere in the gray zone outside of the white grid. Rows may be added using the button,

for adding a single row or the button

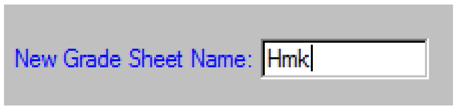
for adding multiple rows (number given the small text box below the button)

Rows may be later deleted using the button for deleting a single row.

Columns may be added (one at a time) using the button

and later deleted using the button

A new worksheet, such as one designed to keep track of homework scores, may be created by first entering the worksheet name in the textbox provided. As an example,

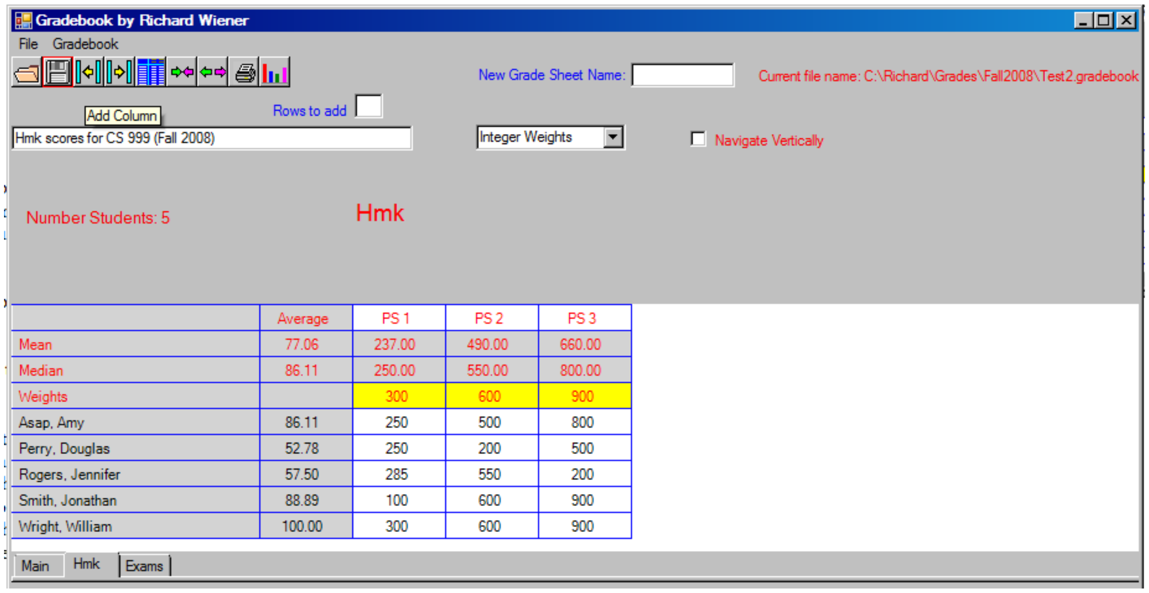

New Grade Sheet Name: Hmk

Next the button

is clicked to create a new worksheet with the new grade sheet name (“Hmk” in the example shown). When a new grid is created a tab with its name appears at the bottom of the grid.

All of the names which are editable in the main worksheet are grayed out in the new worksheet. Changes to this list of names may be made only in the main worksheet. They are propagated to all supporting worksheets such as Hmk.

Hmk grid



Gradebook by Richard Wiener

File Gradebook

New Grade Sheet Name:

Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Add Column

Rows to add

Hmk scores for CS 999 (Fall 2008)

Integer Weights

Navigate Vertically

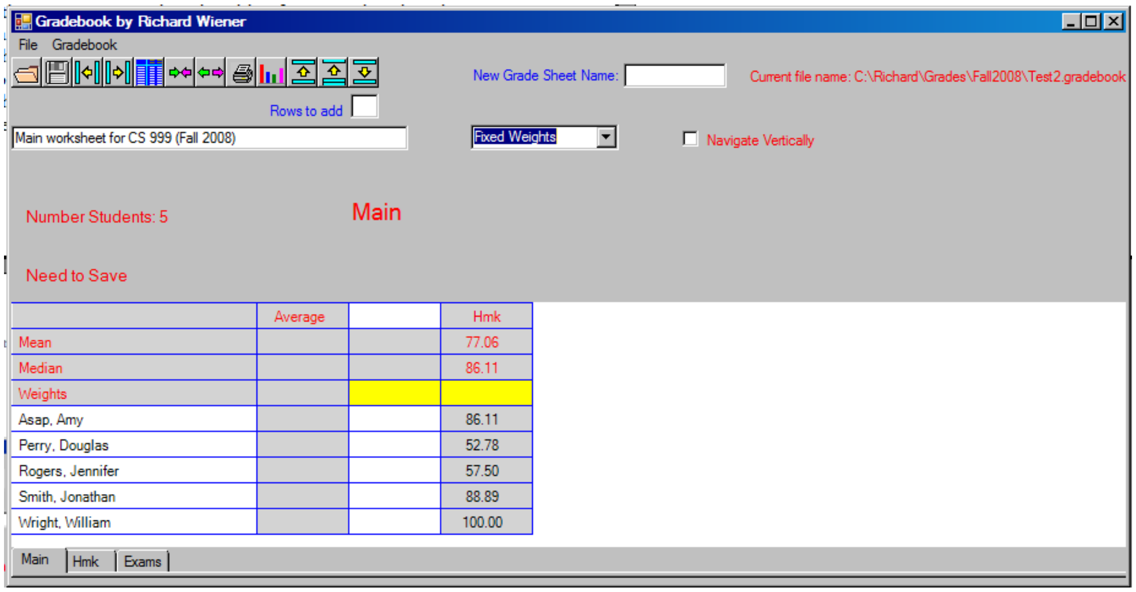
Number Students: 5

Hmk

	Average	PS 1	PS 2	PS 3
Mean	77.06	237.00	490.00	660.00
Median	86.11	250.00	550.00	800.00
Weights		300	600	900
Asap, Amy	86.11	250	500	800
Perry, Douglas	52.78	250	200	500
Rogers, Jennifer	57.50	285	550	200
Smith, Jonathan	88.89	100	600	900
Wright, William	100.00	300	600	900

Main Hmk Exams

Main grid



Gradebook by Richard Wiener

File Gradebook

New Grade Sheet Name:

Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

Main worksheet for CS 999 (Fall 2008)

Fixed Weights

Navigate Vertically

Number Students: 5

Main

Need to Save

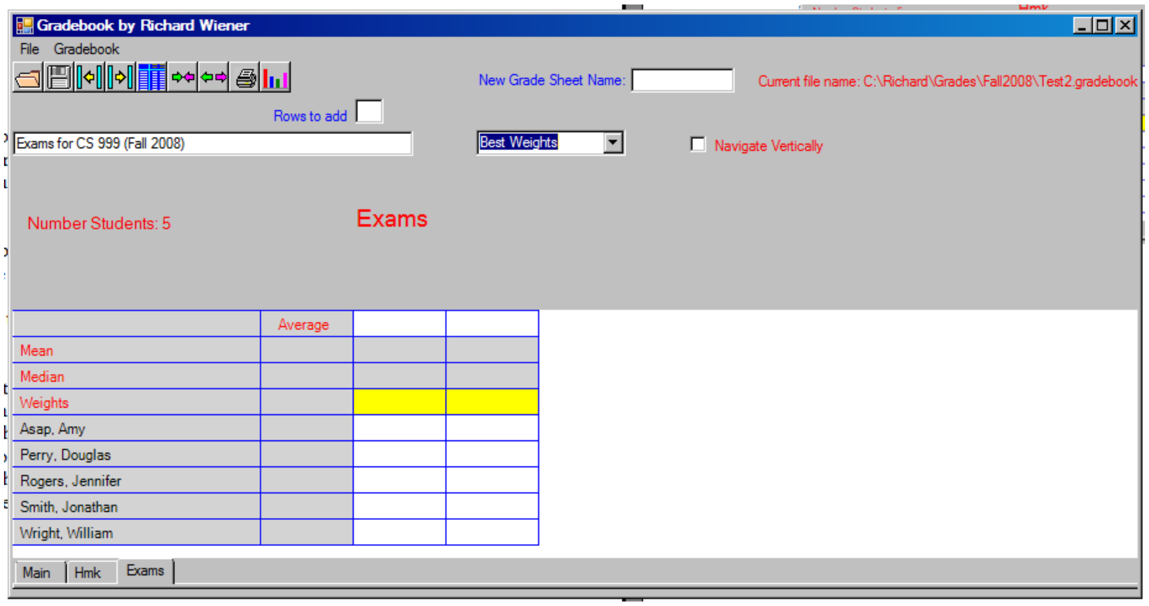
	Average		Hmk
Mean			77.06
Median			86.11
Weights			
Asap, Amy			86.11
Perry, Douglas			52.78
Rogers, Jennifer			57.50
Smith, Jonathan			88.89
Wright, William			100.00

Main Hmk Exams

The “Average” column in the Hmk grid is linked to the fourth column in the Main grid. Any changes that might be made to the data in the Hmk grid will be immediately propagated to the fourth column in the Main grid. This linking must be achieved using the *Gradebook* models for the two grids (Hmk and Main). The effect of linking is then visible in the two views (Main *GradeBookUI* and Hmk *GradeBookUI*).

The process of linking is demonstrated by walking through an example.

A third grid, “Exams” is created. It looks as follows:


Gradebook by Richard Wiener

File Gradebook

New Grade Sheet Name:

Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

Exams for CS 999 (Fall 2008)

Best Weights

Navigate Vertically

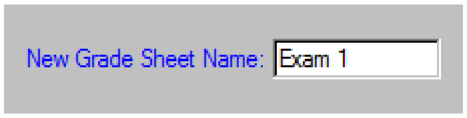
Number Students: 5

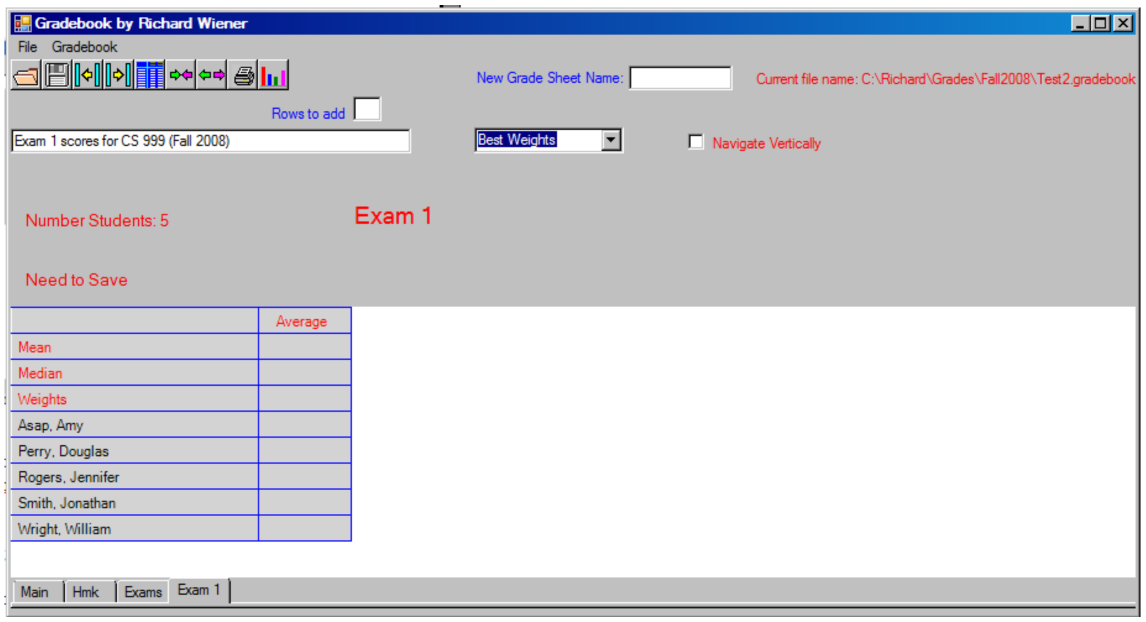
Exams

	Average		
Mean			
Median			
Weights		🟨	🟨
Asap, Amy			
Perry, Douglas			
Rogers, Jennifer			
Smith, Jonathan			
Wright, William			

Main Hmk Exams

Exam 1 scores are entered by creating a fourth grid.


New Grade Sheet Name: Exam 1


Gradebook by Richard Wiener

File Gradebook

New Grade Sheet Name:

Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

Exam 1 scores for CS 999 (Fall 2008)

Best Weights

Navigate Vertically

Number Students: 5

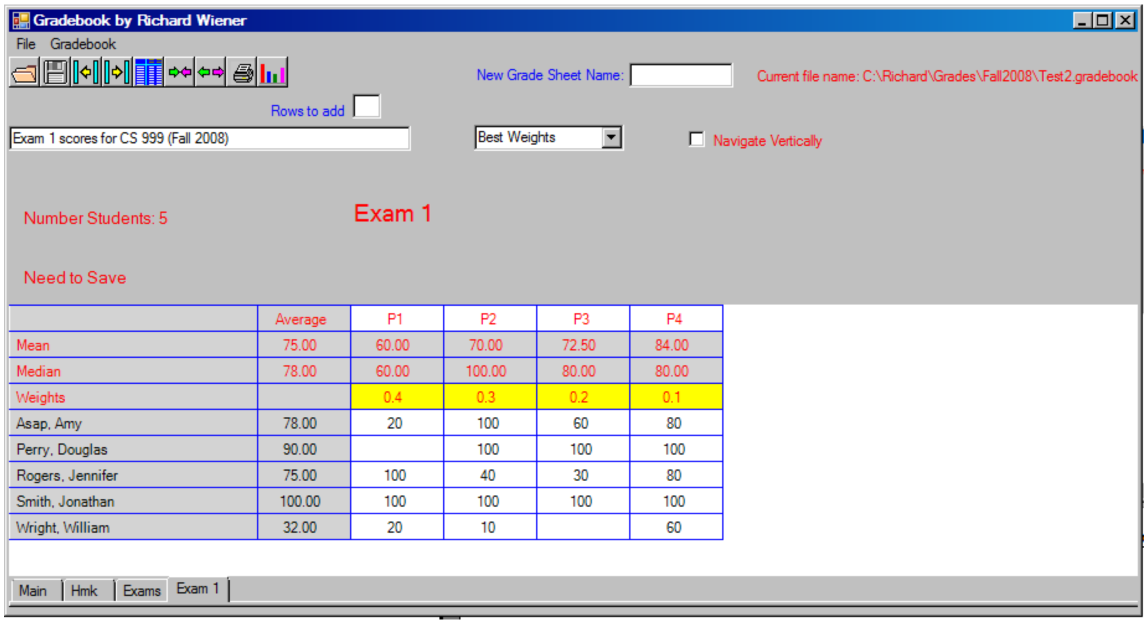
Exam 1

Need to Save

	Average
Mean	
Median	
Weights	
Asap, Amy	
Perry, Douglas	
Rogers, Jennifer	
Smith, Jonathan	
Wright, William	

Main Hmk Exams Exam 1

Suppose there were four problems on this exam with best 40, 30, 20, 10 weights (the best of the four scores is weighted 40 percent, the second best score is weighted 30 percent, the third best scores is weighted 20 percent and the lowest score is weighted 10 percent). This grid might look as follows after the grades have been entered:



Exam 1 scores for CS 999 (Fall 2008)

Number Students: 5

Exam 1

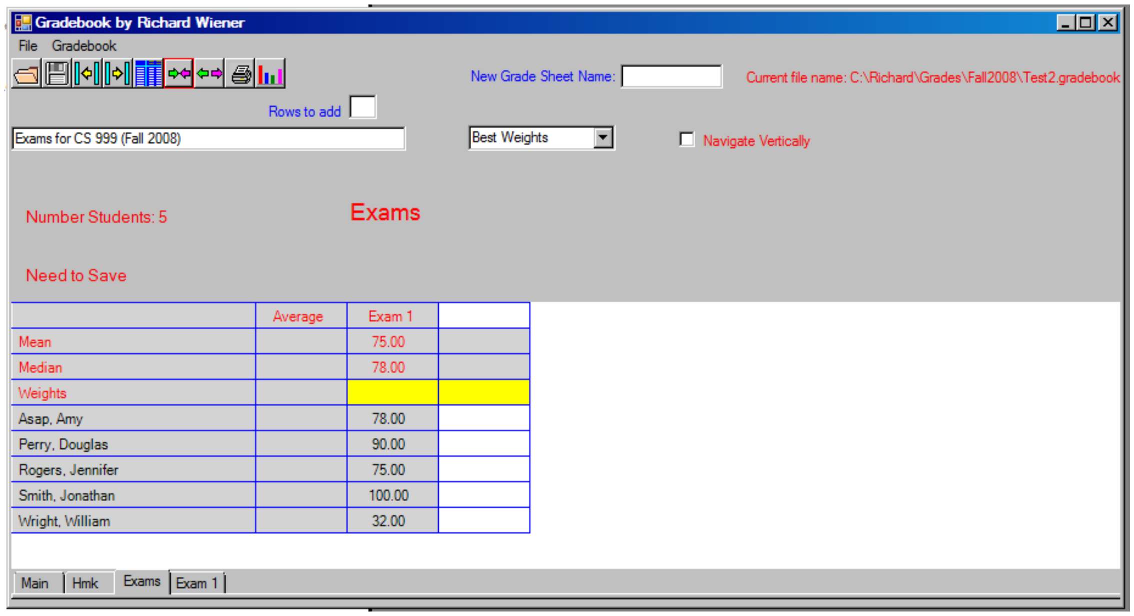
Need to Save

	Average	P1	P2	P3	P4
Mean	75.00	60.00	70.00	72.50	84.00
Median	78.00	60.00	100.00	80.00	80.00
Weights		0.4	0.3	0.2	0.1
Asap, Amy	78.00	20	100	60	80
Perry, Douglas	90.00		100	100	100
Rogers, Jennifer	75.00	100	40	30	80
Smith, Jonathan	100.00	100	100	100	100
Wright, William	32.00	20	10		60

The mean and median values for each problem and the weighted averages for each student are displayed. The only cells that can be edited have white background.

To link this Exam 1 grid to the third column in the Exams grid, select any cell in column 3 in the Exams grid. Then click the link button,

A dialog box that shows all the grids that one can link to will open. After selecting Exam 1, the Exams grid will become:



Exams for CS 999 (Fall 2008)

Number Students: 5

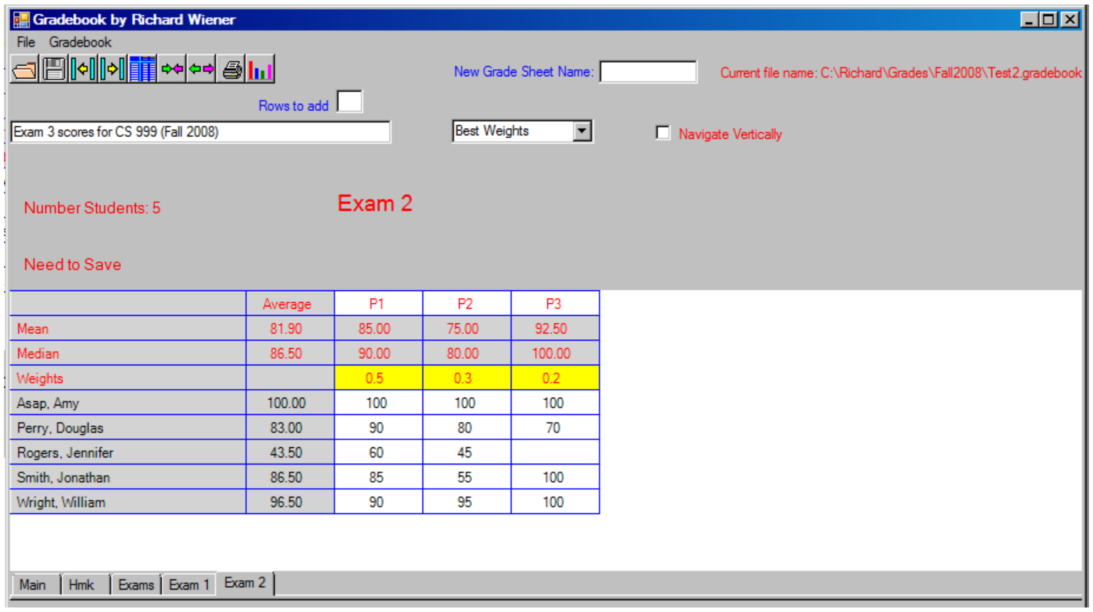
Exams

Need to Save

	Average	Exam 1	
Mean		75.00	
Median		78.00	
Weights			
Asap, Amy		78.00	
Perry, Douglas		90.00	
Rogers, Jennifer		75.00	
Smith, Jonathan		100.00	
Wright, William		32.00	

One could unlink this third column by selecting column 3 (now there is only one editable cell, colored yellow, that you may click to accomplish this) and then clicking the unlink button

After Exam 2 grades are obtained (at the end of the semester), another grid, Exam 2, is built. It is linked to column 4 in Exams. Now the Exams grid has two linked columns.



New Grade Sheet Name:

Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

Exam 3 scores for CS 999 (Fall 2008)

Best Weights

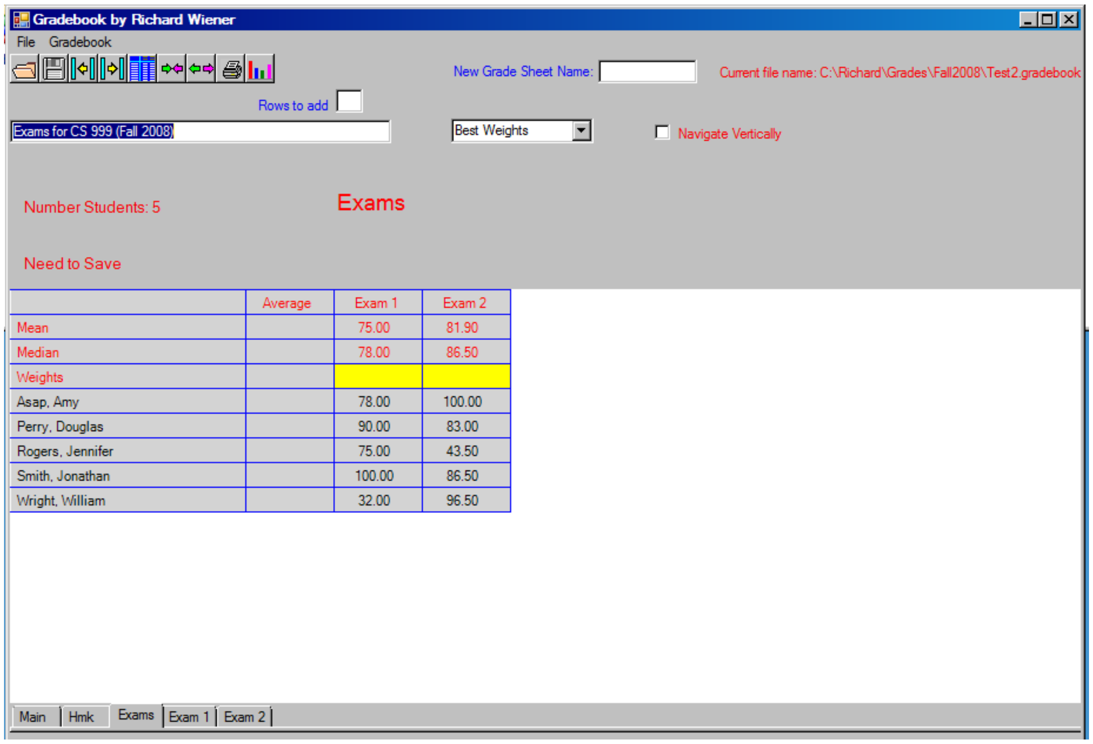
Navigate Vertically

Number Students: 5

Exam 2

Need to Save

	Average	P1	P2	P3
Mean	81.90	85.00	75.00	92.50
Median	86.50	90.00	80.00	100.00
Weights		0.5	0.3	0.2
Asap, Amy	100.00	100	100	100
Perry, Douglas	83.00	90	80	70
Rogers, Jennifer	43.50	60	45	
Smith, Jonathan	86.50	85	55	100
Wright, William	96.50	90	95	100



New Grade Sheet Name:

Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

Exams for CS 999 (Fall 2008)

Best Weights

Navigate Vertically

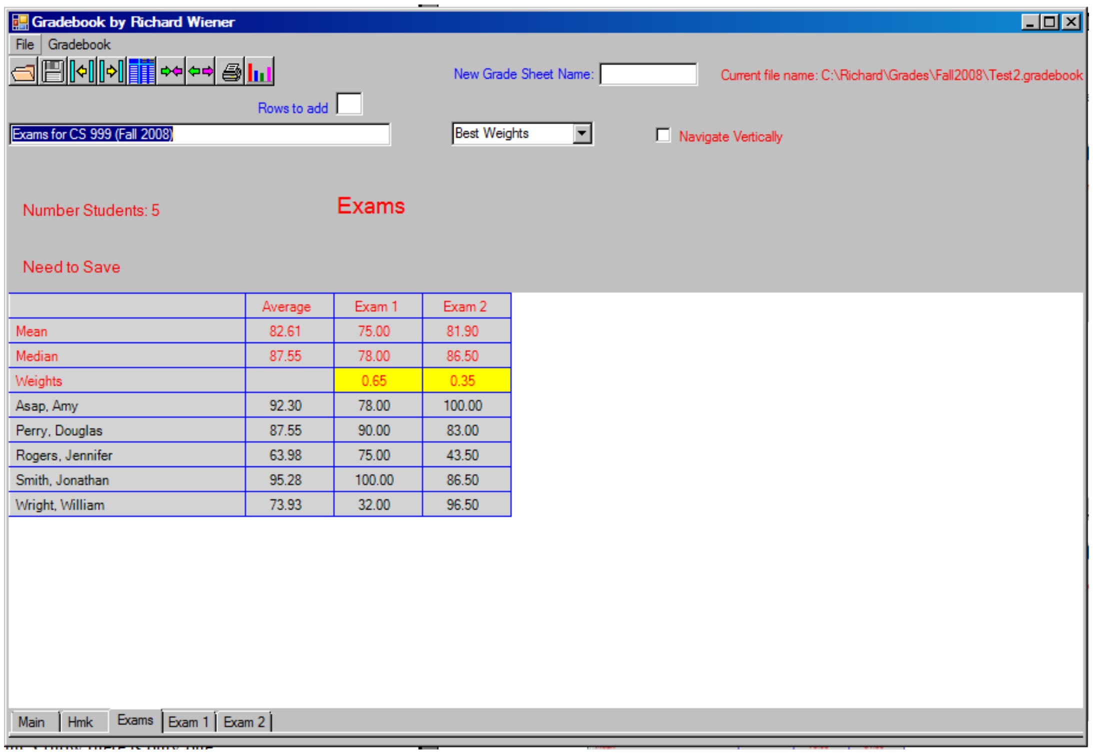
Number Students: 5

Exams

Need to Save

	Average	Exam 1	Exam 2
Mean		75.00	81.90
Median		78.00	86.50
Weights			
Asap, Amy		78.00	100.00
Perry, Douglas		90.00	83.00
Rogers, Jennifer		75.00	43.50
Smith, Jonathan		100.00	86.50
Wright, William		32.00	96.50

Best 65, 35 weights are applied to the Exams grid and the Exams grid is linked to column 3 of the Main grid to produce the final results:



Gradebook by Richard Wiener

File Gradebook

New Grade Sheet Name: Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

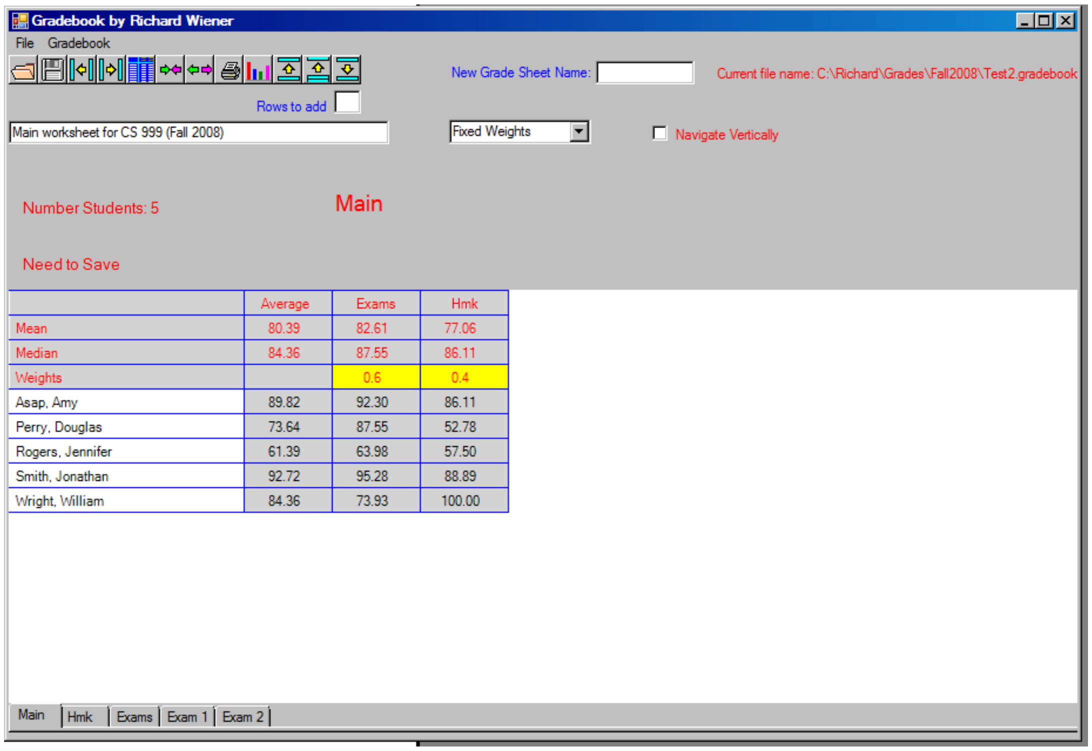
Exams for CS 999 (Fall 2008) Best Weights Navigate Vertically

Number Students: 5 Exams

Need to Save

	Average	Exam 1	Exam 2
Mean	82.61	75.00	81.90
Median	87.55	78.00	86.50
Weights		0.65	0.35
Asap, Amy	92.30	78.00	100.00
Perry, Douglas	87.55	90.00	83.00
Rogers, Jennifer	63.98	75.00	43.50
Smith, Jonathan	95.28	100.00	86.50
Wright, William	73.93	32.00	96.50

Main Hmk Exams Exam 1 Exam 2



Gradebook by Richard Wiener

File Gradebook

New Grade Sheet Name: Current file name: C:\Richard\Grades\Fall2008\Test2.gradebook

Rows to add

Main worksheet for CS 999 (Fall 2008) Fixed Weights Navigate Vertically

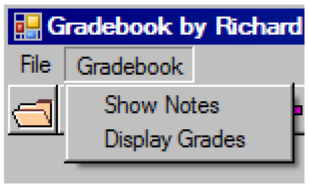
Number Students: 5 Main

Need to Save

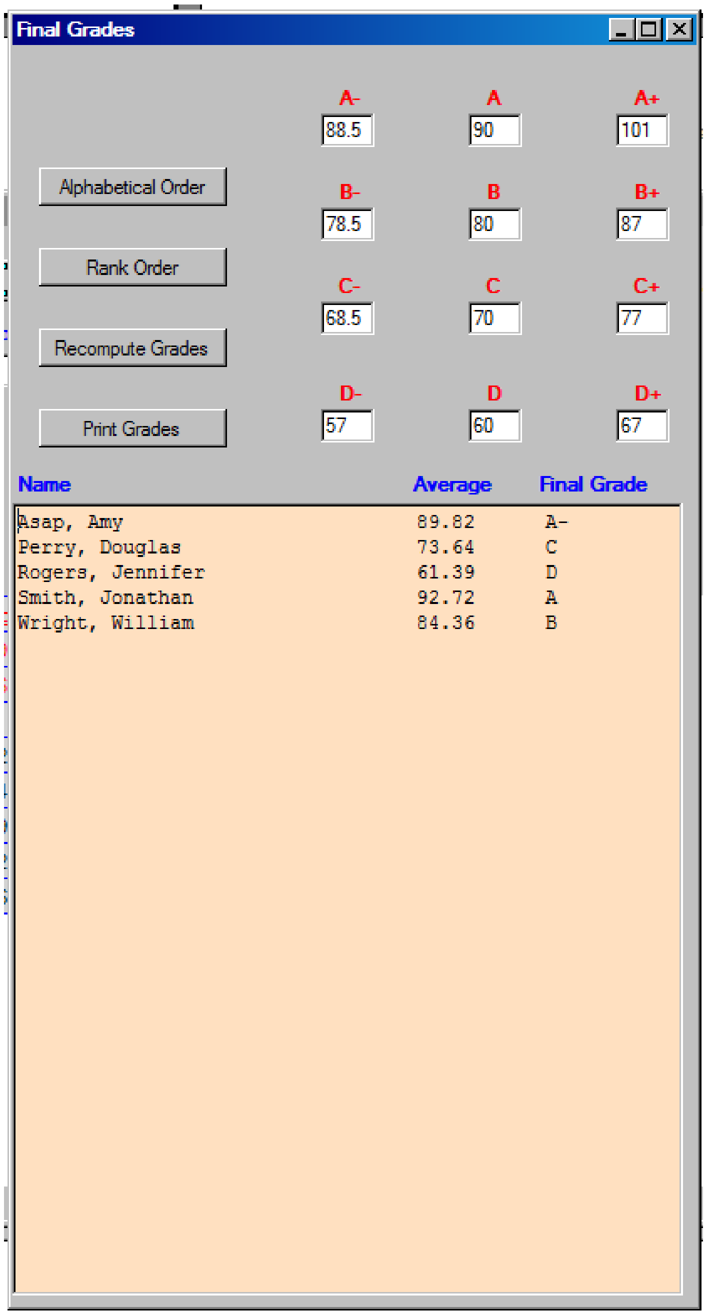
	Average	Exams	Hmk
Mean	80.39	82.61	77.06
Median	84.36	87.55	86.11
Weights		0.6	0.4
Asap, Amy	89.82	92.30	86.11
Perry, Douglas	73.64	87.55	52.78
Rogers, Jennifer	61.39	63.98	57.50
Smith, Jonathan	92.72	95.28	88.89
Wright, William	84.36	73.93	100.00

Main Hmk Exams Exam 1 Exam 2

All that remains is to obtain the final grades. This is accomplished using the Gradebook pull-down menu,



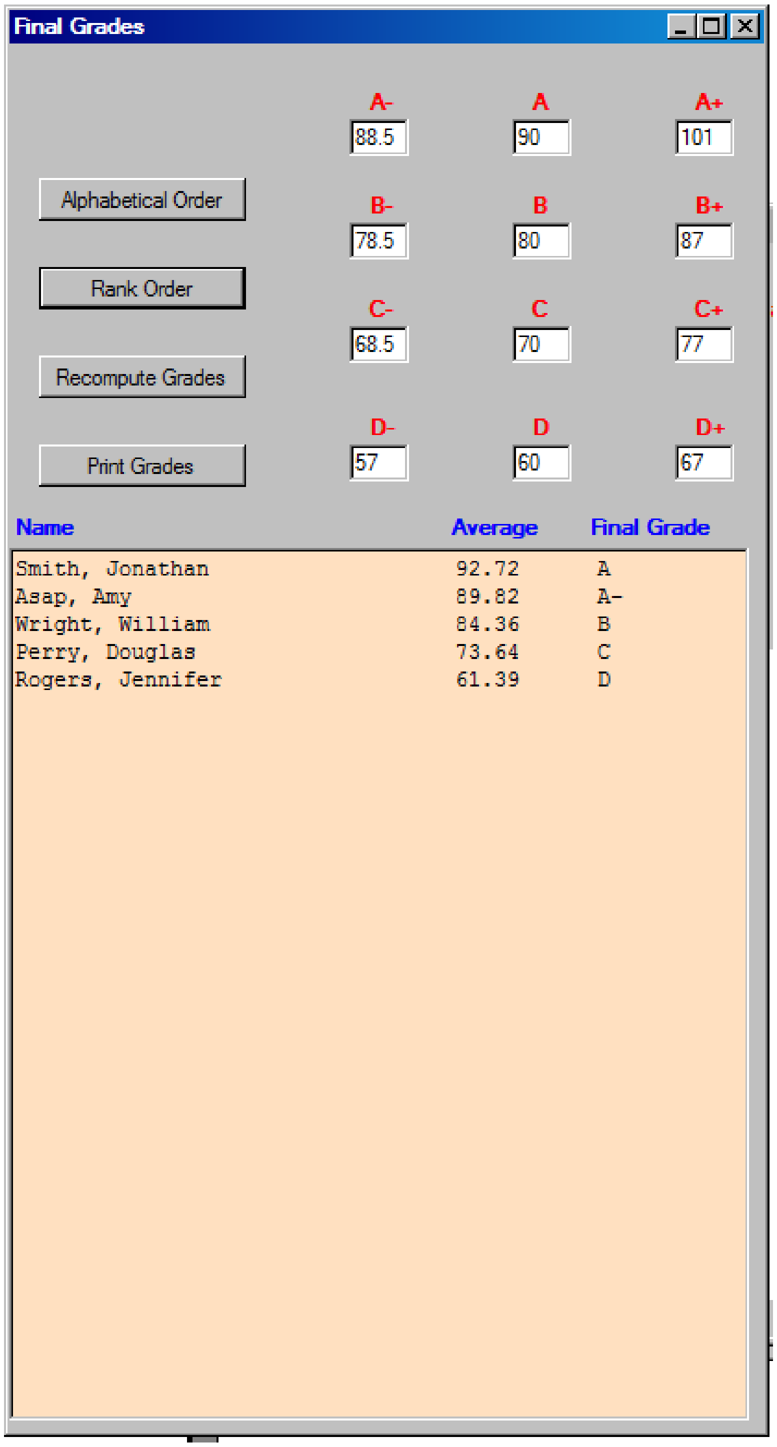
When "Display Grades" is clicked, the following dialog appears:



A-	A	A+
88.5	90	101
B-	B	B+
78.5	80	87
C-	C	C+
68.5	70	77
D-	D	D+
57	60	67

```
Name                          Average    Final Grade
Asap, Amy                     89.82      A-
Perry, Douglas                73.64      C
Rogers, Jennifer              61.39      D
Smith, Jonathan               92.72      A
Wright, William               84.36      B
```

The thresholds may be changed (and they will be saved if the grade book is saved). If the "Rank Order" button is clicked, the redisplay of grades is as follows:

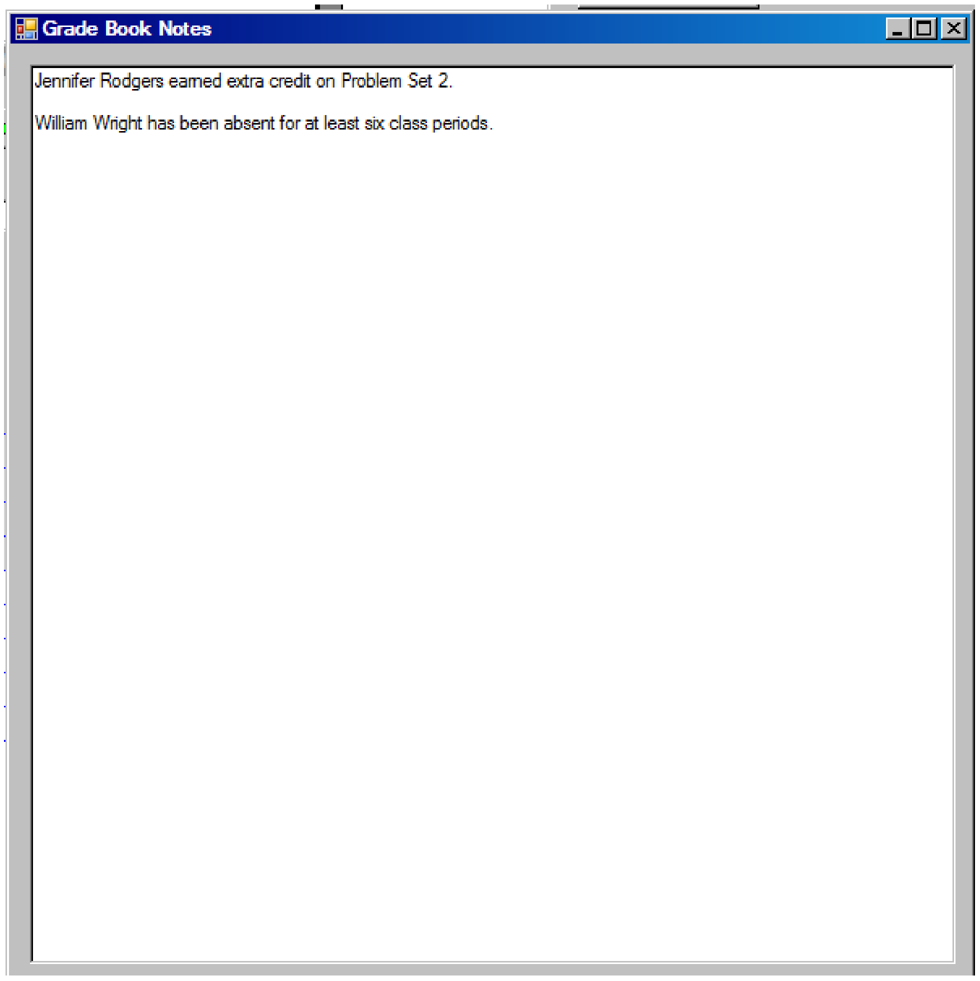


	A-	A	A+
	88.5	90	101
	B-	B	B+
	78.5	80	87
	C-	C	C+
	68.5	70	77
	D-	D	D+
	57	60	67

```
Name                              Average   Final Grade
Smith, Jonathan                   92.72     A
Asap, Amy                         89.82     A-
Wright, William                   84.36     B
Perry, Douglas                    73.64     C
Rogers, Jennifer                  61.39     D
```

Whenever a grid is partially or completely obstructed by another window, the graphics must be restored using a *Paint* handler in the Panel containing the graphics.

If the “Show Notes” menu is clicked (Gradebook/Show Notes), a dialog is produced as in the following:



Jennifer Rodgers earned extra credit on Problem Set 2.

William Wright has been absent for at least six class periods.

These notes are saved when the grade book is saved. The notes are associated with the .grade book file and not an individual grid.

5 DETAILED REQUIREMENTS

Navigation:

When the arrow keys are used, only white-background (editable) cells should be reachable. A non-editable cell (column 1, rows 1 and 2 and any linked column except for rows 0 and 3) should be skipped over.

Updating statistics:

For Equal Weights, whenever a numeric score is entered, the row average and mean and median values should be displayed. For Fixed, Best or Integer weights, only if all weights in row 3 (cells colored yellow) are specified should an average score be computed. The means and medians for each column will always be updated as new values are entered. A blank score will not affect the mean and median but will always be counted as a zero in computing the row average for a given student. If all scores are removed from a row, the average (in column 2) should disappear. If all scores are removed from a column, the mean and median should disappear. The actual computation of scores should be performed by the model class *Gradebook*. Using an appropriately defined event which is fired whenever statistics are computed, the view (class *GradeBookUI*) must be notified.

Linking:

This is one of the more challenging features of this project. When the average column of a source grid is linked to a column in a target grid, it is the models that are linked. When data in the source *Gradebook* is changed, the appropriate cell in the target *Gradebook* must be immediately changed. The result of this will not be seen until the user selects the target grid (by clicking the target tab).

The application through its *Gradebook* model classes must ensure that no circular links exist otherwise this would cause a stack overflow problem. For example, if grid 1 were linked to grid 2, then grid 2 should not be allowed to link to grid 1 otherwise an infinite sequence of updates would occur. Your program logic must therefore control the choices that are available for linking in the link dialog window that is displayed to the user so as to avoid a circular linking problem.

A grid is allowed to be linked only once (to some target grid). So when a grid has been linked, it must be removed from the list of choices for future linking. If a target has a column unlinked, then the original source grid may again be added to the list of possible future links.

Tab control:

All of the grids that have been created must be selectable by the tab control at the bottom of the GUI. When a tab is clicked the existing window must be removed and replaced by the grid that you have selected. For purposes of this project an entire grid form may be displayed. In future maintenance only the graphics portion of a grid form is to be displayed (to avoid flicker and jumping).

Adding rows and columns:

When a new row is added, the cells in a non-editable column must be grayed and not reachable through navigation or mouse clicks. When a column is added, rows 2 and 3 must be non-editable and row 4 colored yellow.

Resizing:

When any grid is resized, all other grids must be displayed with the new size. This value must be saved when the grid is saved. Next time the grids are loaded (through the Open button), their new size must be remembered and used. The tab control must remain properly positioned as the vertical size of a grid is changed.

Saving:

When certain actions occur, the "Need To Save" label is displayed. This occurs when the data in a cell is changed or when the Notes are displayed or when the Final Grades dialog is displayed. Hitting "Ctrl-S" saves all data to the .gradebook file and removes this label from the GUI. When exiting the program, the user should be prompted to save only if the "Need To Save" label is visible.

Display final grades:

This specialized form is to be implemented. When the form is closed, all threshold values must be saved.

Show notes:

This specialized form is to be implemented. When the form is closed the content of the *RichTextBox* control embedded in the form must be saved.

Open and Save-As Dialogs:

The standard *OpenFileDialog* and *SaveFileDialog* controls are to be used in these dialogs. In the OpenDialog control only files ending with the suffix **.gradebook** are to be displayed. The application must remember the last sub-directory location in which a file was opened so that it will return to that same location in the future. This spares the user the need to navigate to the sub-directory containing the *.gradebook* files each time the Open option is selected.

Only the *Gradebook* models are to be saved using object serialization. When a *.gradebook* file is opened, the *Gradebook* models are recovered along with any other data that was saved using object deserialization. From these recovered Gradebook models, the tab control and all the grid views may be reconstructed. This is where a major payoff for the model-view design occurs.

6 DELIVERABLES

The only written document that should accompany the project CD is a list of known bugs and omissions and a self-evaluation of the work.

It is expected that in a project of this complexity there will be bugs or omissions. It is better for the students to implement a sub-set of the requirements with quality than to implement all features with mediocre quality (many bugs). The linking features of this grade book application are very important and central. Omitting this feature will result in a greater loss of credit than omitting a more minor feature such as displaying notes or computing and displaying final grades. The highest priority features, in order of importance, are:

1) Correct navigation controls. Cells that are not editable (column 1 and columns that have been linked) should not be reachable.

2) Being able to create new grids and navigate to them using the tab control. When a new row is added to the main grid (this is the only grid that should allow new rows to be added), this new row must immediately be added to all the existing grade book models so that next time a grid is displayed (through the tab control), this new row is displayed. The name column (left-most column) on all grids except the main grid should be non-editable. This allows only the main grid to be in charge of editing, adding or deleting name entries.
3) Updating and producing the correct statistics on each grid (row averages and column means and medians) when new data is entered. When the grading method is changed by the user, all row average statistics need to be recomputed and redisplayed for the particular grid.
4) Being able to link the average column of one grid to a column in another grid (by linking the models of each grid). Although listed as priority four, this is a very important priority.
5) Being able to save a grade book file and later recover and display all its information
6) All other features are of more minor importance

Items 1 through 4 on the above priority list should be considered “must-do” features of your project.

After completing your testing, a bugs and omissions document should be carefully prepared. It is much better for the student to identify and report a bug (or omission) than for it to be discovered when the project is evaluated.

7 FUTURE MAINTENANCE

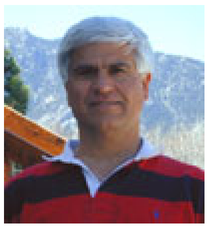
Some important features that one might wish to add to this application to make it more useful in the future include:

1) When the user selects a grid using the tab control, only the graphics panel on the form should be displayed. The rest of the form does not need to be redisplayed.
2) A vertical scroll bar should be added to the form if the number of rows created by the user exceeds the vertical space in the panel. In the current application no scrolling is provided so on a small screen some rows may not be visible.
3) The user should be able to change the name of an existing form or delete it. Deletion of a form may first require some unlinking from source or target grids.
4) A partial grading mechanism should be added so that averages based on incomplete data (e.g. only two out of three exam scores available) for each of the grading types may be computed.
5) An entry of “Excused” should be allowed in any cell that expects a numeric value. This could be used in cases where a legitimate student absence should result in a

partial grading algorithm being used instead of the regular grading algorithm for that particular student.

6) An adjust grade column should be provided on the right side of each panel to allow extra credit or any other constant grade adjustment to be made for a particular student.
7) The ability to export a column of grades to the clipboard or import a column of grades from the clipboard should be provided. This would allow the data on a grid to be easily transported to or from a standard spreadsheet.

About the author



Richard Wiener is Chair of Computer Science at the University of Colorado at Colorado Springs. He is also the Editor-in-Chief of JOT and former Editor-in-Chief of the Journal of Object Oriented Programming. In addition to University work, Dr. Wiener has authored or co-authored 22 books and works actively as a consultant and software contractor whenever the possibility arises. His latest book, published by Thomson, Course Technology in April 2006, is entitled *Modern Software Development Using C#/.NET*.