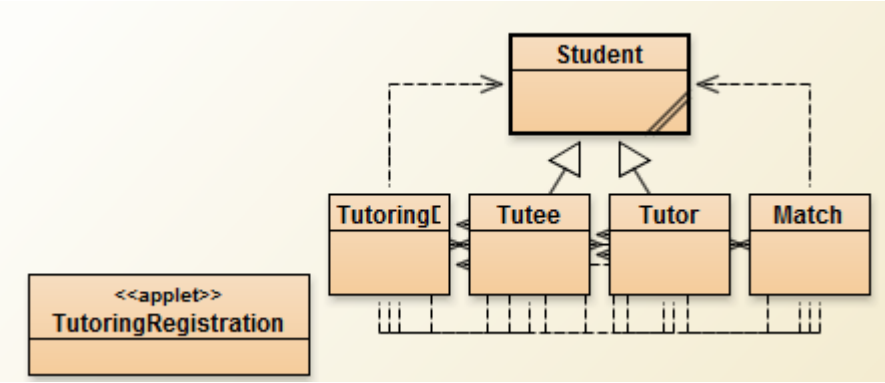
Criterion C: Development

a. Applet
b. During session program

The class structure:



Where Tutee and Tutor are subclasses of Student.

a. Applet

The applet interface is coded using Java's widget GUI toolkit swing.
The required classes and GUI components:

```
1  import java.awt.FlowLayout;
2  import java.awt.event.ActionListener;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.KeyEvent;
5  import java.awt.event.KeyListener;
6  import java.lang.NumberFormatException;
7  import java.text.NumberFormat;
8  import java.util.EventListener;
9  import javax.swing.ButtonGroup;
10 import javax.swing.event.DocumentEvent;
11 import javax.swing.event.DocumentListener;
12 import javax.swing.JApplet;
13 import javax.swing.JButton;
14 import javax.swing.JCheckBox;
15 import javax.swing.JComboBox;
16 import javax.swing.JFormattedTextField;
17 import javax.swing.JLabel;
18 import javax.swing.JRadioButton;
19 import javax.swing.JTextField;
20 import javax.swing.SpringLayout;
21 import javax.swing.SwingConstants;
22 import javax.swing.text.NumberFormatter;
```

I decided to use SpringLayout layout manager for laying out my components because it allows me to specify relationships between the edges of all the components, which I thought would be useful in making a GUI for a form, especially to align all components and to have them in proper order.

The declaration of some of the components:

```
97      // name
98      private static JTextField firstNameField;
99      private static JLabel firstNameLabel;
100     private static JTextField lastNameField;
101     private static JLabel lastNameLabel;
102     // gender
103     private static JRadioButton maleButton;
104     private static JRadioButton femaleButton;
105     private static JLabel genderLabel;
106     private static ButtonGroup genderGroup;
107     // other personal
108     private static JTextField studentNumberField;
109     private static JLabel studentNumberLabel;
110     private static JTextField teacherField;
111     private static JLabel teacherLabel;
112     private static JTextField emailField;
113     private static JLabel emailLabel;
114     // grade
115     private static JRadioButton[] gradeButton;
116     private static JLabel gradeLabel;
117     private static ButtonGroup gradeGroup;
118     // tutor or tutee
119     private static JLabel tutorTuteeLabel;
120     private static JRadioButton tutorButton;
121     private static JRadioButton tuteeButton;
122     private static ButtonGroup tutorTuteeGroup;
123     // days free
124     private static JLabel daysFreeLabel;
125     private static JCheckBox[] day;
```

This was very extensive, since each element of the form needed the element itself, as well as an appropriate label.

Initialization and adding to the applet of some of the components:

```
371     private void createComponents()
372     {
373         // name
374         firstNameLabel = new JLabel("First name:");
375         firstNameField = new JTextField(20);
376
377         lastNameLabel = new JLabel("Last name:");
378         lastNameField = new JTextField(20);
379
380         // gender
381         genderLabel = new JLabel("Gender:");
382         maleButton = new JRadioButton("Male");
383         femaleButton = new JRadioButton("Female");
384         genderGroup = new ButtonGroup();
385         genderGroup.add(maleButton);
386         genderGroup.add(femaleButton);
387
388         // other personal
389         studentNumberLabel = new JLabel("Student Number:");
390
391         // student number field
392         studentNumberField = new JTextField();
393         studentNumberField.setColumns(9);
394
395         // teacher
396         teacherLabel = new JLabel("Homeroom Teacher:");
397         teacherField = new JTextField(20);
```

```
544     private void addComponents()
545     {
546         add(firstNameLabel);
547         add(firstNameField);
548         add(lastNameLabel);
549         add(lastNameField);
550         add(genderLabel);
551         add(maleButton);
552         add(femaleButton);
553         add(studentNumberLabel);
554         add(studentNumberField);
555         add(teacherLabel);
556         add(teacherField);
557         add(emailLabel);
558         add(emailField);
559         add(gradeLabel);
560         for (int i = 0; i < 4; i++)
561         {
562             add(gradeButton[i]);
563         } // end of for (int i = 0; i < 4; i++)
564         add(tutorTuteeLabel);
565         add(tutorButton);
566         add(tuteeButton);
567         add(daysFreeLabel);
568         for (int i = 0; i < DAY.length; i++)
569         {
570             add(day[i]);
571         } // end of or (int i = 0; i < DAY.length; i++)
```

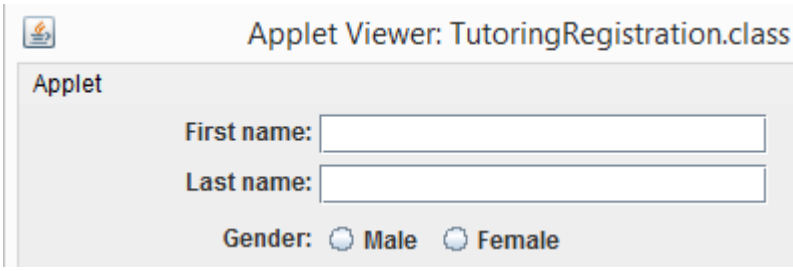
Once again, these processes very extensive due to the number of components that were on this form.

At this point, the components are added but without a specific layout. I used SpringLayout to lay out the components by defining the position of each component's left or right, and top or bottom edge in relation to another component or to the applet's window.

Laying out of the first seven components in the applet:

```
580     private void layoutComponents()
581     {
582         // name
583         layout.putConstraint(LEFT, firstNameField, LEFT_OF_TEXTFIELD, LEFT, this);
584         layout.putConstraint(TOP, firstNameField, 5, TOP, this);
585         layout.putConstraint(TOP, firstNameLabel, 5, TOP, this);
586         layout.putConstraint(RIGHT, firstNameLabel, -4, LEFT, firstNameField);
587
588         layout.putConstraint(LEFT, lastNameField, LEFT_OF_TEXTFIELD, LEFT, this);
589         layout.putConstraint(TOP, lastNameField, 5, BOT, firstNameField);
590         layout.putConstraint(TOP, lastNameLabel, 0, TOP, lastNameField);
591         layout.putConstraint(RIGHT, lastNameLabel, 0, RIGHT, firstNameLabel);
592
593         // gender
594         layout.putConstraint(RIGHT, genderLabel, 0, RIGHT, firstNameLabel);
595         layout.putConstraint(TOP, genderLabel, 12, BOT, lastNameLabel);
596         layout.putConstraint(LEFT, maleButton, 5, RIGHT, genderLabel);
597         layout.putConstraint(TOP, maleButton, 5, BOT, lastNameField);
598         layout.putConstraint(LEFT, femaleButton, 5, RIGHT, maleButton);
599         layout.putConstraint(TOP, femaleButton, 5, BOT, lastNameField);
```

This involved some tweaking of numbers to get the layout clean and uniform. The creating, adding, and laying out of the first 7 components comes out to look like:



The ability for the applet to add and remove combo boxes for the courses was the hardest part of creating the applet.
To add a course combo box:

```
/**
 * Adds a course selection combo box and appropriate minus button.
 */
private void addCourse()
{
    // preserve state of previous combo boxes
    String[] choice = preserveState();
    // remove previous minus buttons and combo boxes
    removePrevious();

    numberOfCourses++;
```

I had to preserve the state of all the combo boxes (the currently selected courses) and remove all combo boxes.

The preserveState() method:

```
private String[] preserveState()
{
    String[] choice = new String[numberOfCourses];
    for (int i = 0; i < numberOfCourses; i++)
    {
        choice[i] = (String)(course[i].getSelectedItem());
    } // end of for (int i = 0; i < numberOfCourses; i++)

    return choice;
} // end of method preserveState()
```

Allows me to record the state of every combo box, to assign to the new combo boxes once those are created.

The removePrevious() method:

```
185     private void removePrevious()
186     {
187         // remove all minus buttons
188         if (numberOfCourses != 1)
189         {
190             for (int i = 0; i < numberOfCourses; i++)
191             {
192                 remove(minusButton[i]);
193             } // end of for (int i = 0; i < numberOfCourses; i++)
194         } // end of if (numberOfCourses != 1)
195         // remove previous combo boxes
196         for (int i = 0; i < numberOfCourses; i++)
197         {
198             remove(course[i]);
199         } // end of for (int i = 0; i < numberOfCourses; i++)
200     } // end of method removePrevious()
```

Removes all previous combo boxes, as well as the minus buttons that correspond to the combo boxes, unless there is only one combo box.

Creating new minus buttons, combo boxes, and assigning state to the combo boxes. Also adding the new components to the applet and setting their layout:

```
340         // reorganize the courses section
341         // assigning new references
342         course = new JComboBox[numberOfCourses];
343         minusButton = new JButton[numberOfCourses];
344         for (int i = 0; i < numberOfCourses; i++)
345         {
346             course[i] = new JComboBox(COURSE);
347             minusButton[i] = new JButton("-");
348         } // end of for (int i = 0; i < numberOfCourses; i++)
349
350         // assign state to previous combo boxes
351         for (int i = 0; i < choice.length; i++)
352         {
353             course[i].setSelectedItem(choice[i]);
354         } // end of for (int i = 0; i < choice.length; i++)
355
356         // adding to applet
357         for (int i = 0; i < numberOfCourses; i++)
358         {
359             add(course[i]);
360             add(minusButton[i]);
361         } // end of for (int i = 0; i < numberOfCourses; i++)
362
363         // laying out changed components
364         for (int i = 0; i < numberOfCourses; i++)
365         {
366             if (i == 0)
367             {
368                 layout.putConstraint(LEFT, course[i], LEFT_OF_TEXTFIELD, LEFT, this);
369                 layout.putConstraint(TOP, course[i], 5, BOT, day[0]);
370             }
```

Add button is moved to next to the last combo box:

```
381         layout.putConstraint(TOP, addButton, 0, TOP, course[course.length - 1]);
382         layout.putConstraint(LEFT, addButton, 50, RIGHT, course[course.length - 1]);
```

Adding listeners for the new minus buttons:

```
384         // listeners for minus buttons
385         for (int i = 0; i < numberOfCourses; i++)
386         {
387             //minusButton[i].setActionCommand("-");
388             minusButton[i].addActionListener(this);
389         } // end of for (int i = 0; i < numberOfCourses; i++)
390         repaint();
391         validate();
392     } // end of method addCourse()
```

That is how I added a combo box. To remove a combo box is a similar process: remove all previous components, record state of them, create new combo boxes, assign previous states, move minus/add buttons, add listeners.

When any of the buttons are pressed:

```
205     public void actionPerformed(ActionEvent actionEvent)
206     {
207         // when submit is pressed
208         if (actionEvent.getSource().equals(submitButton))
209         {
210             // if the information in the form is valid
211             if (validateSubmit())
212             {
213                 edit.setVisible(false);
214                 submitAndReview();
215             }
216             // if not, make visible the lable that prompts for an edit in the form
217             else
218             {
219                 edit.setVisible(true);
220             } // end of if (validateSubmit())
221 
222         } // end of if (actionEvent.getSource().equals(submitButton))
223 
224         // when the add buttons for the courses are pressed
225         if (actionEvent.getSource().equals(addButton))
226         {
227             addCourse();
228         } // end of if (actionEvent.getSource().equals(addButton))
229 
230         // when any of the remove buttons for the courses are pressed
231         for (int i = 0; i < numberOfCourses; i++)
232         {
233             if (actionEvent.getSource().equals(minusButton[i]))
234             {
235                 removeCourse(i);
236             } // end of if (actionEvent.getSource().equals(minusButton[i]))
237         } // end of for (int i = 0; i < numberOfCourses; i++)
238     } // end of method actionPerformed(ActionEvent actionEvent)
```

The appropriate methods are called.

The method validateSubmit() looks at every component in the form and as soon as one of the components has an invalid piece of information, the method returns false:

```
502     private boolean validateSubmit()
503     {
504         // disable submit if text fields are empty
505         // name
506         if (firstNameField.getText().equals("")) return false;
507         if (lastNameField.getText().equals("")) return false;
508
509         // student number must be 9 digits
510         try
511         {
512             int number = Integer.parseInt(studentNumberField.getText());
513             if (number < 100000000 || number > 999999999) return false;
514         }
515         catch (NumberFormatException error)
516         {
517             return false;
518         }
519         if (studentNumberField.getText().equals("")) return false;
520
521         // teacher
522         if (teacherField.getText().equals("")) return false;
523
524         // email
525         if (emailField.getText().length() < 5) return false;
526         // other email specific checks
527         String email = emailField.getText();
528         if (!email.contains("@") && !email.contains(".")) return false;
529         String[] part = email.split("@");
530         if (part.length != 2) return false;
531         if (part[0].length() == 0 || part[1].length() == 0) return false;
532         part = part[1].split("\\.");
533         if (part[part.length - 1].length() == 0 || part[part.length - 2].length() == 0)
534             return false;
535
536         // checking radio buttons
```

Each component has its own specific checks for validity.

The paint() method puts all of the above methods together into the appropriate order so that the applet is created and activated:

```
166     private void paint()
167     {
168         /*
169          * create components
170          * (assigning reference points)
171          */
172         createComponents();
173
174         /*
175          * establish layout
176          */
177         setLayout(layout);
178
179         /*
180          * add components to applet
181          */
182         addComponents();
183
184         /*
185          * edit layout
186          */
187         layoutComponents();
188
189         // focus
190         firstNameField.setFocusable(true);
```

The paint() method is called by the init() method of the applet.

b. Offline portion

The databases:

The match database:

```
10000|323016576|321654987|MDM4
00100|321321321|321456878|SNC2
00001|321456987|321456789|ENG4
```

With the fields being match days (10100 means Monday and Wednesday match day), the tutor's student number, the tutee's student number, and the course(s) being tutored.

The tutee databases:

```
THUMMIM|PARK|F|321654987|MUNRO|THUMMIMPARK@GMAIL.COM|12|0|00100|MPM1|9
STEPHEN|YANG|M|321456789|WANG|STEPHEN.Y@GMAIL.COM|12|0|10110|ENG4|3
VENETIA|CHAN|F|321456878|COMSTOCK|VCHAN@GMAIL.COM|10|0|00001|SNC2|0
```

With the fields being name (first then last), gender, student number, teacher, email, grade, tutor/tutee designation, days free, courses, and attendance number (how many times the person has been there). The tutor database is exactly the same, except for the 0 being a 1 in the fourth from the right field.

Classes:

In the Student class:

```
31      /*
32       * instance fields
33       */
34      private String firstName;
35      private String lastName;
36      private int studentNumber;
37      private char gender;
38      private String email;
39      private String homeroom;
40      private int grade;
41      private String courses = "";
42      private boolean tutor; // false means tutee
43      private char[] daysFree;
44      private int attendance = 0;
```

These are the instance fields that every student will know. The Student class has the appropriate methods for the access and mutation of all of these instance fields.

Creating a student object reads all of the information from either the tutor or tutee database file based on a given student number:

```
67      public Student(int number, boolean tutor)
68      {
69          String[] data;
70          daysFree = new char[DAYS_IN_WEEK];
71          try
72          {
73              // tutor or tutee file
74              BufferedReader studentReader;
75              if (tutor)
76                  studentReader = new BufferedReader(new FileReader(fileTutor));
77              else
78                  studentReader = new BufferedReader(new FileReader(fileTutee));
79
80              String input = "";
81              boolean exit = false;
82              while (((input = studentReader.readLine()) != null) && !exit)
83              {
84                  data = input.split("\\|");
85                  // found the right one, read in the data and assign to this student
86                  if (number == Integer.parseInt(data[INDEX_NUMBER]))
87                  {
88                      firstName = data[0];
89                      lastName = data[1];
90                      gender = data[2].charAt(0);
91                      studentNumber = Integer.parseInt(data[3]);
92                      homeroom = data[4];
93                      email = data[5];
```

With the appropriate data being assigned to the instance fields declared above.

The method saveTutor(Student[] tutor), used to save changes made to an array of pre-existing tutors in the database:

```
155         // write lines
156         PrintWriter writer = new PrintWriter(fileTutor);
157         for (int i = 0; i < count; i++)
158         {
159             String[] data;
160             data = line[i].split("\\|");
161             // change the records in the parameter array
162             if (Student.inArray(tutor, Integer.parseInt(data[INDEX_NUMBER])))
163             {
164                 writer.println(Student.findInArray(tutor, Integer.parseInt(data[INDEX_NUMBER])));
165             }
166             // rewrite others
167             else
168             {
169                 writer.println(line[i]);
170             } // end of if (Student.inArray(tutor, Integer.parseInt(data[INDEX_NUMBER])))
171         } // end of for (int i = 0; i < count; i++)
172         writer.close();
```

Rewrites lines that are not in the array of given tutors, and writes the appropriate changes to a line when that tutor is in the array of given tutors. A similar method exists to save changes made to tutees.

In the TutoringDBMS class:

```
64      private static Student[] tutor;
65      private static Student[] tutee;
66      private static Match[] match;
67      private static boolean[] entered;
68      private static boolean daySelected = false;
69      private static int day = 0;
```

These static variables are used throughout the operation of the database management program, with match being the array of matches on the day, tutor/tutee being the array of tutors/tutees on the day, entered being the array that records whether a tutor/tutee has already signed in on the day, and day being the day that has been selected (Monday, Tuesday, etc.).

Operation begins at the main method:

```
71      public static void main(String[] argument)
72      {
73          console = System.console();
74          passwordPrompt();
75
76          initialMenu();
77      } // end of method main(String[] argument)
```

In passwordPrompt(), prompting for the coordinator password until it is entered. I set the password to 123456789 to being with (it is a static variable).

```
90      do
91      {
92          //inputPassword = console.readPassword("Password: ");
93          //input = new String(inputPassword);
94          System.out.print("Password: ");// this is for testing
95          input = scanner.nextLine();// this is for testing only
96          if (input.equals(password))
97          {
98              exit = true;
99          } // end of if (input.equals(password))
100     }
101     while (!exit);
```

Choices in initialMenu():

```
118         // menu
119         int selection = 0;
120         System.out.println(BAR);
121         System.out.println("1. Session");
122         System.out.println("2. Coordinator Menu");
123         System.out.println("3. Exit");
```

Are filtered into these methods:

```
147         // different selections
148         if (selection == SELECTION_MINIMUM)
149         {
150             if (!daySelected) daySelect();
151             else blankPrompt();
152         } // end of if (selection == SELECTION_MINIMUM)
153         if (selection == TWO)
154         {
155             coordinatorMenu();
156         } // end of if (selection == TWO)
157         if (selection == SELECTION_INITIAL_MAXIMUM)
158         {
159             exit = true;
160         } // end of if (selection == SELECTION_INITIAL_MAXIMUM)
```

Choices in daySelect():

```
171         System.out.println("1. Monday");
172         System.out.println("2. Tuesday");
173         System.out.println("3. Wednesday");
174         System.out.println("4. Thursday");
175         System.out.println("5. Friday");
176         System.out.println("6. Back");
177         // input and validation
178         do
179         {
180             System.out.print("Enter selection: ");
181             input = scanner.nextLine();
```

End of daySelect():

```
197         // if none selected, go back
198         if (selection == SELECTION_DAY_MAXIMUM) return;
199         daySelected = true;
200         day = selection;
201         // else move on
202         readAttendance();
203     } // end of method daySelect()
```

Records whether a day was actually selected or not. If not, daySelect() will be called again later on.

Reading matches for the day that is selected:

```
233         // read those student numbers
234         match = new Match[count];
235         tutor = new Tutor[count];
236         tutee = new Tutee[count];
237         entered = new boolean[count * 2];
238         try
239         {
240             BufferedReader matchReader = new BufferedReader(new FileReader(fileMatch));
241             int index = 0;
242             int matchIndex = 0;
243             String input = "";
244             while ((input = matchReader.readLine()) != null)
245             {
246                 // make new matches and tutors/tutees from information read
247                 // only read for selected day
248                 if (input.charAt(day - 1) == '1')
249                 {
250                     match[index] = new Match(input);
251                     tutor[index] = new Tutor(Integer.parseInt(input.substring(6, 15)));
252                     //System.out.println(student[index]);//this is a check
253                     tutee[index] = new Tutee(Integer.parseInt(input.substring(16, 25)));
254                     //System.out.println(student[index]);//this is a check
255                     index++;
256                 } // end of if (input.charAt(day - 1) == '1')
257             } // end of while ((input = matchReader.readLine()) != null)
258
259             matchReader.close();
```

This is this initialization of the tutor, tutee, and match arrays from the reading of the match database. The tutor and tutee arrays also need to read from the tutor and tutee databases.

Then the blankPrompt() method is called:

```
272     private static void blankPrompt()
273     {
274         String input = "";
275         boolean exit = false;
276         System.out.println(BAR);
277         do
278         {
279             // prompt for student number
280             System.out.print("Student Number: ");
281             input = scanner.next();
282             try
283             {
284                 // if the password is entered, print lines (to hide password)
285                 // and go back to previous menu
286                 if (input.equals(password))
287                 {
288                     exit = true;
289                     for (int i = 0; i < HUNDRED_LINES; i++)
290                     {
291                         System.out.println();
292                     } // end of for (int i = 0; i < HUNDRED_LINES; i++)
293                     return;
294                 } // end of if (input.equals(password))
```

This is the beginning of the “session” operation, where the program takes in a student number or the password. If the password is entered, the previous menu is shown.

If not, the student number that is entered is compared to the array of tutors/tutees that are to be there on that day (read in method readAttendance()). Since “entered” is an array double the length of the tutor and tutee arrays since it has both tutors and tutees, I adjust the index found of the student number in the tutor or tutee array.

```
300     if (!input.equals(password))
301     {
302         boolean found = false;
303         boolean repeat = false;
304         boolean isTutee = false;
305
306         // check if already signed in that day
307         // find in tutors
308         int index = Student.findIndex(tutor, number);
309         // if not tutor, then tutee
310         if (index == -1)
311         {
312             index = Student.findIndex(tutee, number);
313             // if found something, then it is a tutee
314             if (index != -1)
315                 isTutee = true;
316         } // end of if (index == -1)
317         // check if entered already
318         // adjust for tutee index in "entered"
319         if (isTutee)
320             index = match.length + index;
321         // if it is a recognized student number
322         if (index != -1)
323         {
324             // if the person has already signed in
325             if (entered[index])
326             {
327                 repeat = true;
328                 //System.out.println(new Student(entered[i
329             } // end of if (entered[index])
330         } // end of if (index != -1)
```

If not a repeat, then search the tutor/tutee array. If found, print a welcome message, increment attendance, and change the boolean value of found to true.

```
// if already signed in...
if (repeat)
{
    System.out.println("You have already signed in!\n");
}
// if not signed in
else
{
    for (int i = 0; i < tutor.length; i++)
    {
        // if the number entered should be there on the day
        if (number == tutor[i].getStudentNumber())
        {
            tutor[i].increaseAttendance();

            System.out.println("Welcome, " + tutor[i].getFirstName() + "!\n");
            found = true;
            // to record the people that already signed in
            entered[i] = true;
        } // end of if (number == tutor[i].getStudentNumber())
    } // end of for (int i = 0; i < tutor.length; i++)
    // if not tutor, check tutees
    if (!found)
    {
        for (int i = 0; i < tutee.length; i++)
        {
            // if the number entered should be there on the day
            if (number == tutee[i].getStudentNumber())
            {
                tutee[i].increaseAttendance();

                System.out.println("Welcome, " + tutee[i].getFirstName() + "!\n");
                found = true;
                // to record the people that already signed in
                entered[i + match.length] = true;
```

If the student number isn't found in the day's tutor/tutee array and it isn't a repeat, that student number either doesn't exist or shouldn't be there on the day:

```
// if the person shouldn't be there on the day
if (!found && !repeat)
{
    System.out.println("Not recognized.\n");
} // end of if (!found && !repeat)
```

Save changes made to the array (attendance data) using methods in class Student:

```
// save the changes made to attendance
Student.saveTutor(tutor);
Student.saveTutee(tutee);
} // end of method blankPrompt()
```

In the coordinator menu selection review, the method review():

```
        // print information for each match
        for (int i = 0; i < match.length; i++)
        {
            String output = "";
            // print tutor
            // if not present
            if (!entered[i])
                output = "!";
            output += match[i].getTutor().getFirstName() + " "
                + match[i].getTutor().getLastName();
            int initialLength = output.length();
            for (int j = 0; j < length - initialLength; j++)
            {
                output += " ";
            } // end of for (int j = 0; j < length - initialLength; j++)
            System.out.print(output);

            // print tutee
            output = "";
            // if not present
            if (!entered[i + match.length])
                output = "!";
            output += match[i].getTutee().getFirstName() + " "
                + match[i].getTutee().getLastName();
            initialLength = output.length();
            for (int j = 0; j < length - initialLength; j++)
            {
                output += " ";
            } // end of for (int j = 0; j < length - initialLength; j++)
            System.out.print(output);

            // print courses
            System.out.println(match[i].getCourses());
        } // end of for (int i = 0; i < match.length; i++)
    } // end of method review()
```

Output is lined up to a specified number of characters defined by “length.” A “!” is added in front of the name to indicate that the person has not yet signed in.

When adding a new tutor or tutee manually, check if they are already registered by comparing the student number to that of the registered tutors/tutees:

```
// check if already registered
try
{
    BufferedReader reader = new BufferedReader(new FileReader(Student.fileTutor));
    while ((input = reader.readLine()) != null)
    {
        if (input.contains(String.valueOf(number)))
        {
            System.out.println("Already registered.");
            return;
        } // end of if (input.contains(String.valueOf(number)))
    } // end of while ((input = reader.readLine()) != null)
}
catch (IOException error)
{} // end of try-catch block
```

To add a match:

```
Tutee.showTutees();
Student[] tuteeSelection = Tutee.getAllTutees();
// tutee selection
int number = -1;
String input = "";
boolean exit = false;
do
{
    System.out.print("Which tutee? (0 to exit) ");
    input = scanner.nextLine();
```

Show all tutees (from the tutee database) and ask for which tutee to make a match.
Take all tutors and filter them by day:

```
// filter compatible tutors by day
int count = 0;
for (int i = 0; i < tutorSelection.length; i++)
{
    count = 0;
    for (int j = 0; j < tuteeSelected.getDaysFree().length; j++)
    {
        // for free days of the tutee
        if (tutorSelection[i].getDaysFree()[j] == '1' &&
            tuteeSelected.getDaysFree()[j] == '1')
        {
            count++;
        } // end of if (tutorSelection[i].getDaysFree()[j] == '1'...)
    } // end of for (int j = 0; j < tuteeSelected.getDaysFree().length; j++)
    // delete the tutor from compatible ones if no free days overlap
    if (count == 0)
    {
        tutorSelection[i] = null;
    } // end of if (count == 0)
} // end of for (int i = 0; i < tutorSelection.length; i++)
```

And remove incompatible tutors from the selection:

```
790         // remove the incompatible tutors
791         count = 0;
792         for (int i = 0; i < tutorSelection.length; i++)
793         {
794             if (tutorSelection[i] == null)
795             {
796                 count++;
797             } // end of if (tutorSelection[i] == null)
798         } // end of for (int i = 0; i < tutorSelection.length; i++)
799         tutorDayFiltered = new Student[tutorSelection.length - count];
800         // put day filtered tutors into new array
801         count = 0;
802         for (int i = 0; i < tutorSelection.length; i++)
803         {
804             if (tutorSelection[i] != null)
805             {
806                 tutorDayFiltered[count] = tutorSelection[i];
807                 count++;
808             } // end of if (tutorSelection[i] != null)
809         } // end of for (int i = 0; i < tutorSelection.length; i++)
```

The filter tutors by course, and remove them in the same way as above:

```
311         // filter compatible tutors by course
312         count = 0;
313         String[] course = tuteeSelected.getCourses().split(",");
314         for (int i = 0; i < tutorDayFiltered.length; i++)
315         {
316             count = 0;
317
318             for (int j = 0; j < course.length; j++)
319             {
320                 // see if any of the same courses
321                 if (tutorDayFiltered[i].getCourses().contains(course[j]))
322                 {
323                     count++;
324                 } // end of if (tutorDayFiltered[i].getCourses().contains(course[j]))
325             } // end of for (int j = 0; j < course.length; j++)
326             // delete the tutor from compatible ones if no courses overlap
327             if (count == 0)
328             {
329                 tutorDayFiltered[i] = null;
330             } // end of if (count == 0)
331         } // end of for (int i = 0; i < tutorDayFiltered.length; i++)
```

Show all compatible tutors (after filtering) and select one:

```
353         // show compatible tutors
354         System.out.println("Compatible tutors: ");
355         String output = "";
356         count = 0;
357         for (int i = 0; i < tutorFiltered.length; i++)
358         {
359
360             output = "";
361             output += (i + 1) + ". ";
362             output += tutorFiltered[i].getFirstName() + " "
363                 + tutorFiltered[i].getLastName() + " free on "
364                 + new String(tutorFiltered[i].getDaysFree()) + " for "
365                 + tutorFiltered[i].getCourses();
366             System.out.println(output);
367         } // end of for (int i = 0; i < tutorFiltered.length; i++)
368         int selection = -1;
369         // choose which tutor
370         do
371         {
372             System.out.print("Pair " + tuteeSelected.getFirstName() + " with which tutor? (0 to exit) ");
373             try
374             {
375                 input = scanner.nextLine();
376                 selection = Integer.parseInt(input);
377             }
378             catch (NumberFormatException error)
379             {} // end of try-catch block
380         }
381         while (selection < 0 || selection > tutorFiltered.length);
382
383         if (selection == 0) return;
384
385         Match.newMatch(tutorFiltered[selection - 1], tuteeSelected);
```

Then select the meet days, ensuring that the days selected match up with the tutor and tutee's free days:

```
// select which days
Scanner scanner = new Scanner(System.in);
char[] attempt = new char[tutor.getDaysFree().length];
boolean good = true;
do
{
    good = true;
    System.out.print("Which days? (eg. 10100 for Mon and Wed meet days) ");
    attempt = scanner.nextLine().toCharArray();
    if (attempt.length == tutor.getDaysFree().length)
    {
        // check to see if the entered days work
        for (int i = 0; i < attempt.length; i++)
        {
            if (attempt[i] == '1' && days[i] != '1')
            {
                good = false;
            }
        }
    }
    else
    {
        good = false;
    }
}
while (!good);
```

Then make the tutor and tutee not free on those days anymore, and save changes:

```
// make tutor/tutee not free anymore on the match days
char[] newDaysTutor = tutor.getDaysFree();
char[] newDaysTutee = tutee.getDaysFree();
for (int i = 0; i < days.length; i++)
{
    if (attempt[i] == '1')
    {
        newDaysTutor[i] = '0';
        newDaysTutee[i] = '0';
    } // end of if (days[i] == '1')
} // end of for (int i = 0; i < days.length; i++)
tutor.setDaysFree(newDaysTutor);
tutee.setDaysFree(newDaysTutee);
// save changes
Student[] saveTutor = {tutor};
Student[] saveTutee = {tutee};
Student.saveTutor(saveTutor);
Student.saveTutee(saveTutee);
```

Finding the same courses:

```
121         // match courses
122         String[] course1 = tutor.getCourses().split(",");
123         String[] course2 = tutee.getCourses().split(",");
124
125         String courses = null;
126
127         for (int i = 0; i < course1.length; i++)
128         {
129             for (int j = 0; j < course2.length; j++)
130             {
131                 if (course1[i].equals(course2[j]))
132                 {
133                     if (courses == null)
134                     {
135                         courses = course1[i];
136                     }
137                     else
138                     {
139                         courses += "," + course1[i];
140                     } // end of if (courses == null)
141                 } // end of if (course1[i].equals(course2[j]))
142             } // end of for (int j = 0; j < course2.length; j++)
143         } // end of for (int i = 0; i < course1.length; i++)
```

To delete a match, show all matches and select one to delete:

```
389     private static void deleteMatch()
390     {
391         System.out.println(BAR);
392         Match.showCurrentMatches();
393         // get input for which to delete
394         int number = -1;
395         String input = "";
396         boolean exit = false;
397         do
398         {
399             System.out.print("Delete which match? (0 to exit) ");
400             input = scanner.nextLine();
```

Read all matches in the database into an array and then change the days free of the individuals in the match:

```
// edit the daysfree of the deleted pair
Match deleted = new Match(read);
char[] daysMatch = deleted.getMeetDays();
char[] daysTutor = deleted.getTutor().getDaysFree();
char[] daysTutee = deleted.getTutee().getDaysFree();
for (int i = 0; i < daysTutor.length; i ++)
{
    // make the tutor/tutee free again on the days where the pair met
    if (daysMatch[i] == '1')
    {
        daysTutor[i] = '1';
        daysTutee[i] = '1';
    } // end of if (daysMatch[i] == '1')
} // end of for (int i = 0; i < daysTutor.length; i ++)
deleted.getTutor().setDaysFree(daysTutor);
deleted.getTutee().setDaysFree(daysTutee);
// save to file
Student[] changedTutor = {deleted.getTutor()};
Student[] changedTutee = {deleted.getTutee()};
Student.saveTutor(changedTutor);
Student.saveTutee(changedTutee);
```

Write all matches back to file except the deleted one:

```
// rewrite lines
PrintWriter writer = new PrintWriter(fileMatch);
for (int i = 0; i < numberOfMatches; i++)
{
    // don't write the removed one
    if (i != number - 1)
    {
        writer.println(line[i]);
    } // end of if (i != number - 1)
} // end of for (int i = 0; i < numberOfMatches; i++)
writer.close();
```