

NESDA Tournament Manager 2013

IB HL Computer Science Dossier: May 2013

**By: Zbyněk Stara, 000889-045
International School of Prague**

Teacher: John Scott Rayworth

IB Higher Level

Table of Contents

A1: Analysis.....	3
A2: Criteria for Success	9
A3: Prototype Solutions	13
B1: Data Structures	23
B2: Algorithms.....	53
B3: Modular Organization	71
C2: Error Handling	83
C3: Success of the Program	111
D1: Test Output.....	115
D2: Evaluating Solutions	135
Documentation of Mastery Aspects.....	145
C1: Code	151
MainGui.java	152
BinarySearchTree.java.....	265
Database.java	273
School.java.....	286
Teacher.java	308
Student.java.....	310
StudentPair.java	316
DAStudentPair.java	318
DebateStudentPair.java	319
Qualification.java.....	320
Event.java.....	328
QualificationEvent.java	332
Round.java	339
Room.java	343
Judge.java.....	347
Entity.java	349
Appendix.....	351
Appendix 1: First Interview with the Client	351
Appendix 2: Second Interview with the Client.....	365
Appendix 3: Initial Prototype Interview	369

Section A1: Analysis

Problem Statement:

The organizers have to distribute various participants of a Speech and Debate tournament according to complex criteria, and the way they do it now leads to errors and delays.

Description of the Situation:

My client is the Speech and Debate coach at International School of Prague, and she has taken part in the organization of the recent NESDA Speech and Debate tournament in Prague. NESDA stands for New European Speech and Debate Association¹ and it consists of 12 schools² sending their representatives to semiannual tournaments³. These are the students, who compete in five events (Original Oratory, Oral Interpretation, Impromptu Speaking, Duet Acting, and Debate; the latter two are attended by teams of two students);⁴ and the teachers, scoring the students' performance in those events.⁵

For most of the events, there are two qualification rounds, in which each participant gets a chance to perform;⁶ and one final round, for which only a number of the best students are selected, according to their scores.⁷ (The only exception is debate, which has three qualification rounds – in the first two, the pair argues for the affirmative side and the negative side of the question; the third round is an impromptu, unprepared debate. The scores from these rounds determine the best four teams, who then participate in the semi-finals; the winners then face each other in the final debate.)⁸

The organizers have to place all the participants of a given event to a room for each of the rounds they participate in and they have to make sure several conditions are met. The most prominent among these rules holds that a particular student cannot be scored twice by the same judge for the same

¹ Appendix 1: First Interview with the Client, line 13

² Appendix 1: First Interview with the Client, line 85

³ Appendix 1: First Interview with the Client, line 18

⁴ Appendix 1: First Interview with the Client, lines 18-28

⁵ Appendix 1: First Interview with the Client, lines 61-73

⁶ Appendix 1: First Interview with the Client, lines 147-162

⁷ Appendix 1: First Interview with the Client, lines 167

⁸ Appendix 1: First Interview with the Client, lines 185-206

event.⁹ Others – less rigid, and sometimes breakable – rules hold that a student should not be scored by a teacher from his own school, and that the student should not meet the same students in the second round as he did in the first round.¹⁰

Organizers need to record and add up the scores of each student for each event.¹¹ After all the qualification rounds are completed, the list of finalists has to be posted, so that the participants know who is to compete in the final.

And, in the final, each judge is asked to – after they have seen all performers in a given round – rank the students with the placement he thinks he got among the other finalists. The student with the least sum of these rankings (since it is best to be ranked with small numbers, for example 1, as opposed to, say, 5) is declared the winner, and the two students placing after him are declared second and third, respectively.

⁹ Appendix 1: First Interview with the Client, lines 349-361

¹⁰ Appendix 1: First Interview with the Client, lines 349-361, 68-73

¹¹ Appendix 1: First Interview with the Client, lines 175-183

Input and output:

Input data ¹²		
Input name	Input example	Description
School name	International School of Prague	Identification of the school
Teacher name	Caskie	Identification of the teacher
Student name	Zbyněk Stara	Name of the student
Original Oratory	true	Events the student participates in
Oral Interpretation	true	
Impromptu Speaking	false	
Duet Acting	false	
Debate	true	
Qualification Scores	27, 28, 26, 27	Scores the student got from one event; separate for each event and each student
Final Rankings	1, 1, 1, 1, 2	If the student gets to the final, the rankings he received; separate for each event and each student

Output data ¹³		
Output name	Output example	Description
Participant Code	E42	Code identifying the participant (student or pair) throughout the tournament
Event	Original Oratory	Identifies the current event
Round	OO Q2	Identifies the current round
Room	Room 1	Identifies the current room

¹² Appendix 1: First Interview with the Client¹³ Appendix 1: First Interview with the Client

Judges	Badeau, O'Murchu	Judges in a given room in a given round
Participants	E42, B23, F75, L144, A5, C25	Participants in a given room in a given round
Finalists	E42, B22, F77, L140, A7, C30	Finalists for a given event
Winner	E42	Winner of a given event
Second place	L140	Second place in a given event
Third place	B22	Third place in a given event
Student choice	E42	Student choice award winner in a given event

How has the problem been solved before?

For the distribution of people, the organizers currently have to rely on tools like Microsoft Excel, which requires them to manually input students and judges into different rooms for all qualification rounds;¹⁴ there is a lot of trial and error involved in this process as the selection needs to comply with a fairly strict set of rules. It has been reported that this part of the process is very tedious and time-consuming and gives way to human error. Moreover, the client reported that sometimes, adjustments to the distribution have to be made (in case of illness of the student, or if an error has been made with the student's events).¹⁵

Furthermore, students' scores are recorded on sheets of paper and then sent to a central office, in which usually two people match the sheets with students, to be able to record (to an Excel spreadsheet) the particular student's score from given event after each round and, eventually, be able to determine, which students got to the final rounds.¹⁶ (The client has reported that even this task currently has to be done manually, with the officials having to literally

¹⁴ Appendix 1: First Interview with the Client, lines 97-114

¹⁵ Appendix 1: First Interview with the Client, lines 235-244

¹⁶ Appendix 1: First Interview with the Client, lines 169-183

skim through the complete list of participants in an event and see which scores are the highest.)¹⁷

The client has expressed an opinion that “there must be”¹⁸ some computerized solutions already in place for organizing Speech and Debate tournaments. A search of the options has revealed that this assertion is indeed true. *Tournaman*, *TRPC*, *Joy of Tournaments*, *Middle School Public Debate Program*, *Tabbie*, *PlusTab*, *Speech and Debate Together*, or *EverythingTab*, are all programs that specialize in facilitating the process of organizing Speech and Debate tournaments.¹⁹ However, none of these would be ideal for the purposes of NESDA, since it has multiple events and most from the aforementioned programs can only handle one event – debate.

Furthermore, since most of the solutions have been produced in the United States, without any knowledge of NESDA and its specifics (for example, sorting people by scores in the qualification, but ranking them in the finals), no program can appropriately handle the challenges of organizing a NESDA tournament. Since even the current solution to the problem is not ideal, as my client has expressed, the challenges outlined by the user would be best solved by a specialized, customized program that would bear NESDA in mind from the very first stages of development.

¹⁷ Appendix 1: First Interview with the Client, lines 376-383

¹⁸ Appendix 1: First Interview with the Client, lines 388-390

¹⁹ Programs taken from list at <http://debate.uvm.edu/software.html>

**Section A2:
Criteria for Success**

What the program should perform:

The interview with client (which reflects the information in the section A1: Analysis and goes into more detail), has revealed that there are two essential functions the program must be able to perform:

1. It has to record all the students and teachers from all the schools and add them to the appropriate schools (along with the events and pairs information for students); it should assign student codes to the students;²⁰
2. The program has to be able to place all the participants – students and judges – into correct rooms, so that several criteria set by NESDA are not broken;²¹

The client has also shared several extension ideas which could, but do not have to, be included in the program:

1. The program could record all scores for all of the students, as well as the final rankings for the finalists;²²
2. The program could be able to indicate – after scores are collected for a given event – which students are moving to the final for that event;²³
3. After the finals, the program could indicate the first and other awarded places – as well as the student choice award.²⁴

Performance limits:

Speed limits:

1. The assignment of student codes is expected to be fast,²⁵ therefore, it should take no time longer than one second to assign the student codes;
2. The allocation of participants does not need to be instantaneous, but it should be fast enough to allow for last-minute re-allocation.²⁶ Limit for feasibly doing that is 60 seconds, or 12 second per event.

²⁰ Appendix 2: Second Interview with the Client, line 3

²¹ Appendix 2: Second Interview with the Client, line 4

²² Appendix 2: Second Interview with the Client, line 7

²³ Appendix 2: Second Interview with the Client, line 8

²⁴ Appendix 2: Second Interview with the Client, line 9

²⁵ Appendix 2: Second Interview with the Client, line 15

3. The presentation of a list of finalists and, similarly, of the winners is expected to be completed (seemingly) instantaneously,²⁷ so, again, no time longer than one second would be acceptable to the user;

Memory limits:

1. The minimum number of schools the program needs to be able to work with is 12; the number of students is $12 \times 12 = 144$; the number of judges is at least $2 \times 12 = 24 + 10 = 34$.²⁸ However, the numbers might grow even more. The program's data structures must therefore be able to hold at least this number of database entries, with a room for extension, as well.
2. The client has no knowledge of how much memory the program should use,²⁹ but making an educated guess by adopting a 4-year old laptop computer – low-end in performance at the International School of Prague – as a standard, the program should not require no more than 2 GiB of memory. (However, that is the farthest limit, the program will likely require much less than that.)

Response to extreme/erroneous data:

1. The user has stated that the program should show a warning dialog window to alert the user that an extreme or erroneous entry has been typed into the program;³⁰ therefore the response to erroneous data entry in the program should be a dialog window informing the user that input in a wrong format has been typed in;
2. The scores have a defined range of acceptable values (0-30),³¹ therefore any numerical value typed as a score that is outside the given range would have to be identified as extreme and the user would have to be alerted about that fact.

²⁶ Appendix 2: Second Interview with the Client, line 14

²⁷ Appendix 2: Second Interview with the Client, line 15

²⁸ Appendix 1: First Interview with the Client, lines 242-252

²⁹ Appendix 2: Second Interview with the Client, line 19

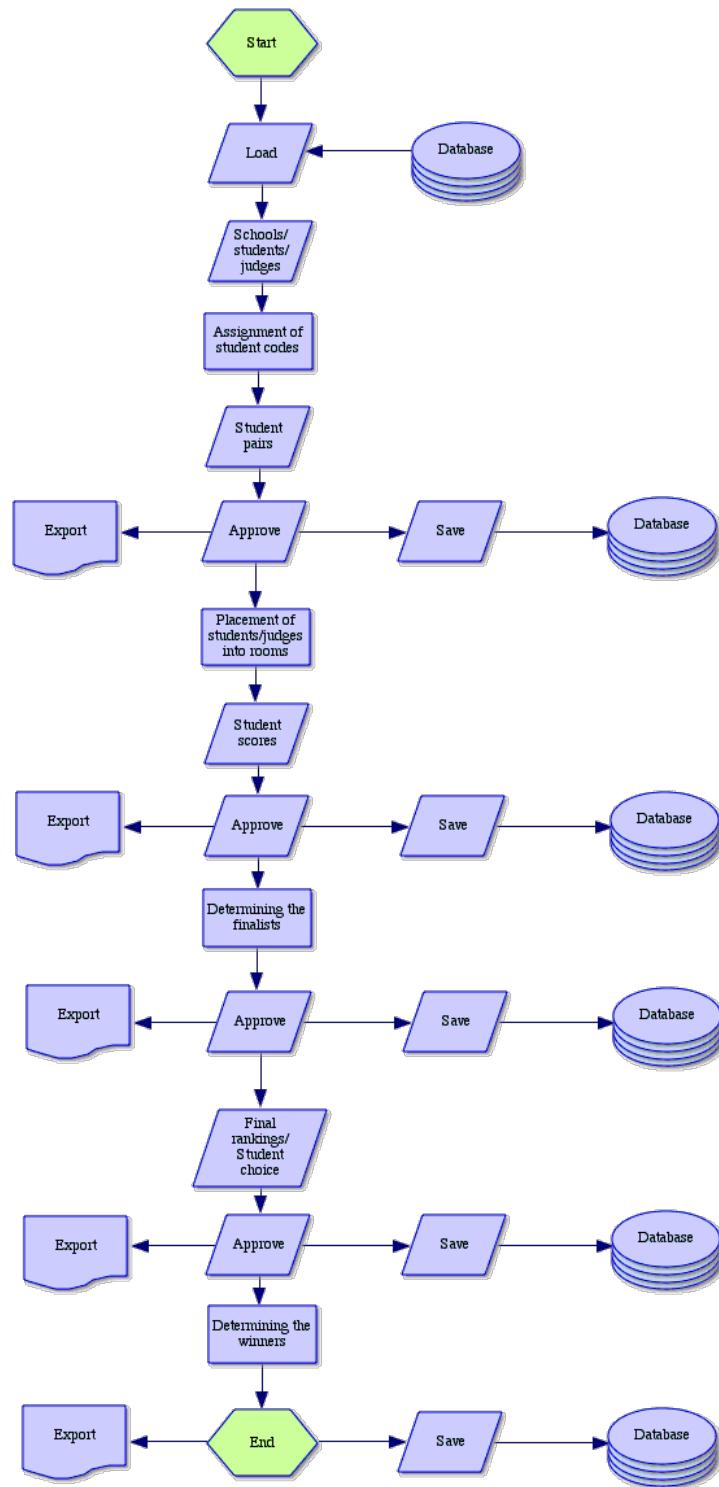
³⁰ Appendix 2: Second Interview with the Client, line 23

³¹ Appendix 1: First Interview with the Client, lines 43

**Section A3:
Prototype Solutions**

Initial Design:

This is a rough overview diagram of the initial design of the program (created in the program Inspiration).



Interaction with the Client:

An initial prototype of the GUI was produced and shown to the client to ensure we both have the same ideas in mind. This proved to be the case for the most part, thanks to the specificity of the initial interview, except for several points. These areas for improvement were identified by the user during the discussion of the screenshots of the program. These points were marked up on the scanned images of the initial prototype used for the interview.

NESDA Tournament Manager 2012 by Zbyněk Stara

Event settings:		Students / Room		Students / Event / School		Judges / Room		General settings:	
Original Oratory	6	4	2	Rooms available	8	Maximal tournament time	16	hours	
Oral Interpretation	6	4	2	Schools in tournament	10	Teachers / School	2		
Impromptu Speaking	6	4	2	Students / School	8	Events / Student	3		
Duet Acting	4	pairs	2	pairs	2				
Debate (Q & semifinals)			2	pairs	3				
Finals (OO, OI, IS, DA)	6	6	6	4	pairs	5			

Load Approve Reset
Save

Overall progress

NESDA Tournament Manager 2012 by Zbyněk Stara

Schools	Teachers	Students	Student codes	Assign events	Assign pairs
<No elements>	<No selection>	<No selection>	<No selection>	<input type="checkbox"/> Original Oratory <input type="checkbox"/> Oral Interpretation <input type="checkbox"/> Impromptu Speaking <input type="checkbox"/> Duet Acting <input type="checkbox"/> Debate	Duet Acting Debate <input type="button" value="Pair"/> <input type="button" value="Remove pair"/>
Add Remove Edit Search	Add Remove Edit Search	Add Remove Edit Search	Assign codes Assign events Search	Assign events Assign pairs Pair Remove pair	
Export Approve Save Progress				Overall progress	

client likes the export button for easy access to data

client likes checking dialogs for additional safety

*spouse
co-pilot*

NESDA Tournament Manager 2012 by Zbyněk Stara

Qualification

Event	Round	Room	Judges	Student codes	Scores
<input type="button" value="Assign event"/> <input type="button" value="Edit name"/> <input type="button" value="Substitute judge"/> <input type="button" value="Search"/> <input type="button" value="Add"/> <input type="button" value="Remove"/> <input type="button" value="Export"/> <input type="button" value="Approve"/> <input type="button" value="Reset"/> <input type="button" value="Save"/>					
Progress					
Overall progress					

NESDA Tournament Manager 2012 by Zbyněk Stara

Finalists

Original Oratory	Oral Interpretation	Impromptu Speaking	Duet Acting	Debate semifinalists	Debate Finalists
<input type="button" value="Set winner A"/> <input type="button" value="Set winner B"/> <input type="button" value="Export"/> <input type="button" value="Approve"/> <input type="button" value="Reset"/> <input type="button" value="Save"/>					
Progress					
Overall progress					

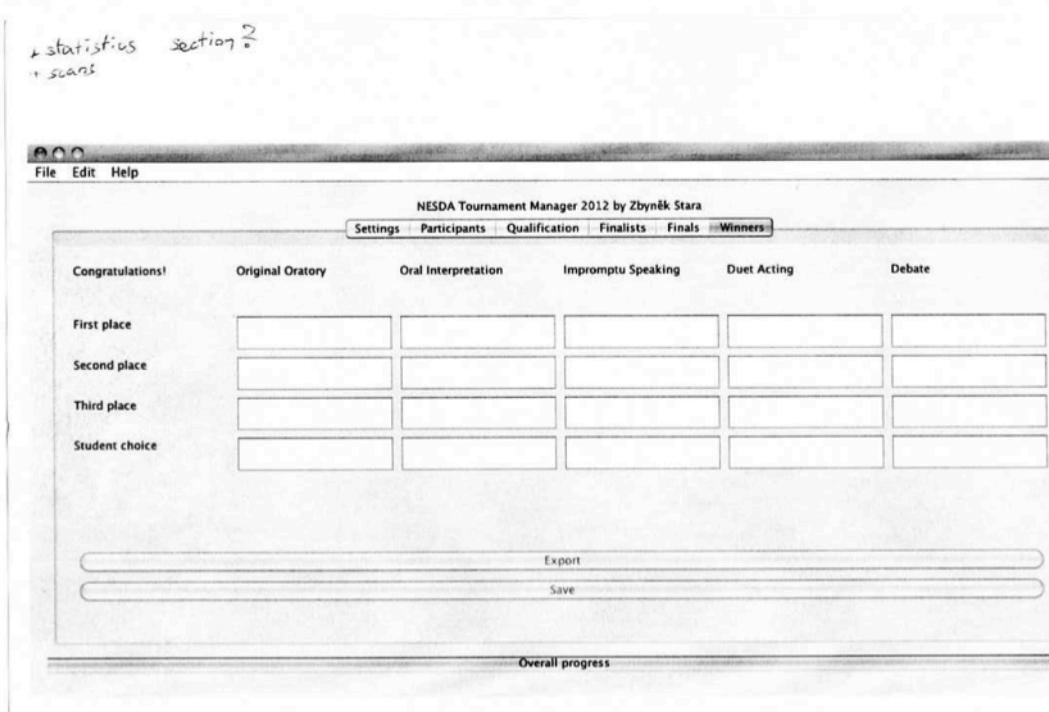
*assumed to be
just one*

NESDA Tournament Manager 2012 by Zbyněk Stara

Finals

Event	Round	Room	Judges	Student codes	Rankings
<input type="button" value="Assign event"/> <input type="button" value="Edit name"/> <input type="button" value="Substitute judge"/> <input type="button" value="Search"/> <input type="button" value="Add"/> <input type="button" value="Remove"/> <input type="button" value="Set students' vote"/> <input type="button" value="Export"/> <input type="button" value="Approve"/> <input type="button" value="Reset"/> <input type="button" value="Save"/>					
Progress					
Overall progress					

*rounds
specify
in theory
x what
about
what's
going on?*



Overall, the client tends to agree with the design – I tried to reflect as closely as possible the outcomes of the detailed first interview –, but in some important points, she identifies possible weak points and areas for improvement.

The Maximal tournament time and Rooms available settings need to be split for two days.³² Furthermore, overlapping events and times per event need to be added to the settings.³³ NESDA logos need to be added to the top of the program.³⁴ And finally, the incomplete approval of the Qualification section will be necessary to be accounted for.³⁵ (An optional feature was also identified, the statistics section.³⁶)

The changes that were the outcome of this interaction can be easily implemented, requiring only minor changes to the program (although the incomplete approval necessitates more profound changes). All of these changes could be done in NetBeans by manipulating the current GUI features.

³² Appendix 3: Initial Prototype Interview, line 3

³³ Appendix 3: Initial Prototype Interview, line 45

³⁴ Appendix 3: Initial Prototype Interview, line 11

³⁵ Appendix 3: Initial Prototype Interview, line 31

³⁶ Appendix 3: Initial Prototype Interview, line 54

Final Prototype:

This section presents screenshots of the final prototype of the program.

This version of the GUI is almost identical to the final version used for the testing and evaluation sections of this dossier.

Legend:

- **Red color** labels the elements that were changed thanks to the feedback from the user obtained in the initial prototype discussion.
- **Blue labels** point to elements that perform an action inside the program.
- **Green labels** describe what is shown in lists.
- **Pink text** labels actions that take user input into the program.

Settings tab:

NESDA Tournament Manager 2012 by Zbyněk Stara

File Edit Help

NESDA logos were added

The program settings with default values

Settings Participants Qualification Finalists Finals Winners

Event settings: Students / Room Students / Event / School Judges / Room General settings:

Original Oratory	6	4	2	Rooms available (Day 1, 2)	8	8
Oral Interpretation	6	4	2	Tournament time (Day 1, 2)	16	hours
Impromptu Speaking	6	4	2	Schools in tournament	10	
Duet Acting	4 pairs	2 pairs	2	Teachers / School	2	
Debate (Q & semifinals)		2 pairs	3	Students / School	8	
Finals (OO, OI, IS, DA)	6 6 6 4 pairs		5	Combined Events	00	00

Approves the input values – checks for errors
- if no errors found, enables the participants tab

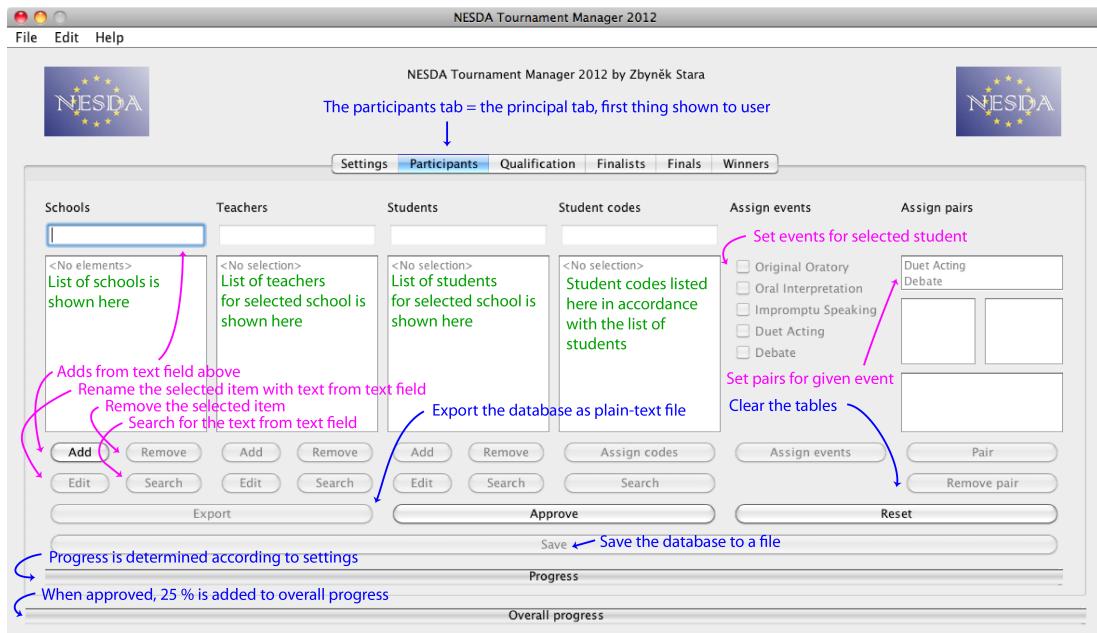
Allows for loading files from previous sessions

Load Approve Reset Save

Overall progress

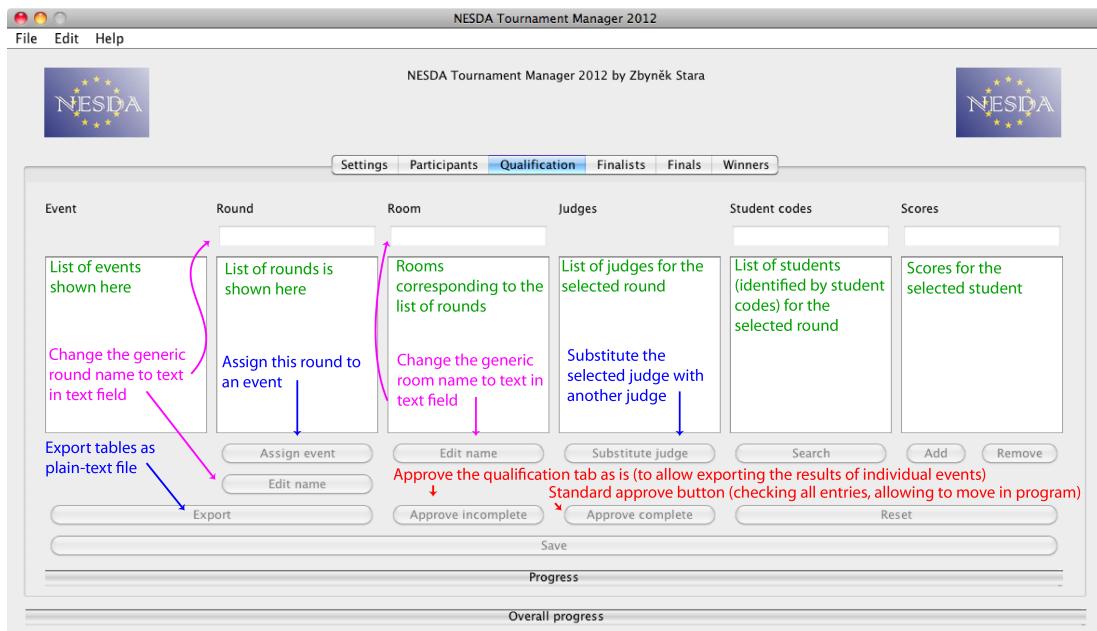
This tab allows for customization of the tournament through a multitude of various settings. Note the addition of NESDA logos and the splitting of the room and time setting into two days.

Participants tab:



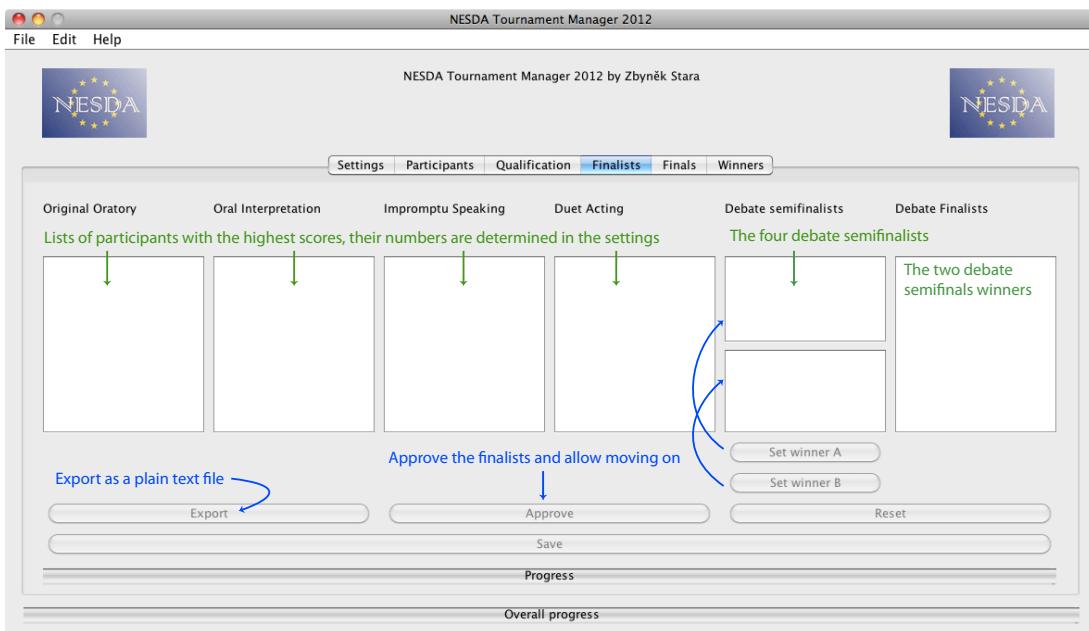
The first tab shown to the user (the settings tab is pre-approved by default). It allows for addition of participants to the tournament, along with the events they participate in and their pair partners.

Qualification tab:



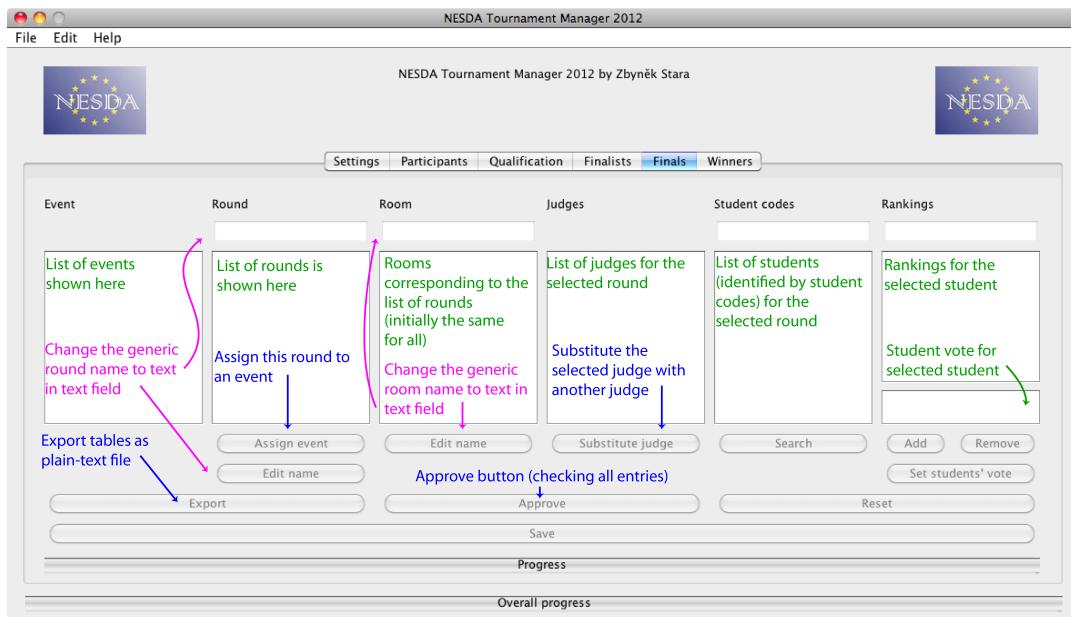
Note that the approve button was split into two different buttons – Approve incomplete, and Approve complete. Thanks to that, the user can move to the qualification tab and export results for events separately, without the need to submit the scores for all events.

Finalists tab:



The export button in this tab can now be used without this tab being approved completely – to allow for fast printing of a list of finalists for events when those results are available.

Finals tab:



This tab is essentially the same as the Qualification tab, however, the approve button is no longer split (as there is no need anymore for incomplete approval); and the student votes have been added to the Rankings section.

Winners tab:

Congratulations! The first three places in all events, and the student choice award; name and student code are shown

	Original Oratory	Oral Interpretation	Impromptu Speaking	Duet Acting	Debate
First place					
Second place					
Third place					
Student choice					

Names of both students are shown for pair events

Export as plain text

Export

Save

Save the file

Overall progress

The final tab, showing the first three places in all events, along with the student choice winners. For each of those, the student code is shown along with the participant name (or names, in case of pair events).

**Section B1:
Data Structures**

Data Structures Anticipated at the Design Stage:***Primitives:***

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
ooStudentRoomLimit (Database class)	int	6	Setting, maximal number of students per room for OO event	
oiStudentRoomLimit (Database class)	int	6	Setting, maximal number of students per room for OI event	
isStudentRoomLimit (Database class)	int	6	Setting, maximal number of students per room for IS event	
daStudentRoomLimit (Database class)	int	6	Setting, maximal number of students per room for DA event	
ooStudentFinalLimit (Database class)	int	5	Setting, maximal number of students per room for OO final	
oiStudentFinalLimit (Database class)	int	5	Setting, maximal number of students per room for OI final	
isStudentFinalLimit (Database class)	int	5	Setting, maximal number of students per room for IS final	
daStudentFinalLimit (Database class)	int	5	Setting, maximal number of students per room for DA final	
ooStudentSchoolLimit (Database class)	int	5	Setting, maximal number of students per school in OO	
oiStudentSchoolLimit (Database class)	int	5	Setting, maximal number of students per school in OI	
isStudentSchoolLimit (Database class)	int	5	Setting, maximal number of students per school in IS	
daStudentSchoolLimit (Database class)	int	5	Setting, maximal number of students per school in DA	
debateStudentSchoolLimit (Database class)	int	2	Setting, maximal number of students per school in debate	
ooJudgeRoomLimit (Database class)	int	2	Setting, number of judges per room for OO event	
oiJudgeRoomLimit (Database class)	int	2	Setting, number of judges per room for OI event	
isJudgeRoomLimit (Database class)	int	2	Setting, number of judges per room for IS event	
daJudgeRoomLimit (Database class)	int	2	Setting, number of judges per room for DA event	
debateJudgeRoomLimit (Database class)	int	3	Setting, number of judges per room for debate	
finalsJudgeRoomLimit (Database class)	int	5	Setting, number of judges per room for a final event	
roomLimit1 (Database class)	int	8	Setting, number of rooms available for 1 st day of the tournament	
roomLimit2 (Database class)	int	8	Setting, number of rooms available for 2 nd day of the	

			tournament	
timeLimit1 (Database class)	int	8	Setting, amount of time in hours available for 1 st day of the tournament	
timeLimit2 (Database class)	int	8	Setting, amount of time in hours available for 2 nd day of the tournament	
schoolNumber (Database class)	int	12	Setting, the expected number of schools in a tournament	
teacherSchoolNumber (Database class)	int	2	Setting, the expected number of teachers per school	
studentSchoolNumber (Database class)	int	12	Setting, the expected number of students per school	
defaultOOStudentRoomLimit (Database class)	int	6	<i>The default value of the ooStudentRoomLimit setting</i>	<i>An example of the variables storing the default values of setting variables. All of the variables presented above have their default variable.</i>
name (School class)	String	“International School of Prague”	Name of school in the tournament	
name (Teacher class)	String	“Caskie”	Teacher’s name (convention is to only include surname)	
name (Student class)	String	“Zbyněk Stara”	Student’s name (serves to identify winners, therefore the whole name is to be included)	
studentCode (Student class)	String	“B19”	Student’s code (letter corresponds to code for school; number corresponds to student’s alphabetical order number in school, counted continually from previous schools – e.g. “...A11, A12, B13, B14...”, where A12 is the last person from A, and B13 is the first from B)	
isPair (Entity class)	boolean	true	Stores information about whether a given entity is a pair entity (for DA and debate events)	
name (Round class)	String	“OO Round I”	Name of a given round; generic names are assigned by default but can be changed by user	
name (Room class)	String	“Room 3”	Name of a given room; generic names are assigned by default but can be changed by user	

Important GUI Elements:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
generalSettingsCECheckBox (MainGUI class)	JCheckBox	*selected	Setting, selection allows for the rounds of two qualification events to occur simultaneously to save time	Elements for specifying whether rounds for events are to occur at the same time, and if yes, which events
generalSettingsCEBox1 (MainGUI class)	JComboBox	*OO *OI *IS *DA	Setting, selection in the combo box determines the first event that will be shared (IS is the default option)	
generalSettingsCEBox2 (MainGUI class)	JComboBox	*OO *OI *IS *DA	Setting, selection in the combo box determines the second event that will be shared (DA is the default option)	
participantsAssignEventsOriginalOratoryCheckBox (MainGUI class)	JCheckBox	*selected	Selection means that the student will participate in Original Oratory	Check boxes for the specification of events which a given student is to participate in
participantsAssignEventsOralInterpretationCheckBox (MainGUI class)	JCheckBox	*selected	Selection means that the student will participate in Oral Interpretation	
participantsAssignEventsImpromptuSpeakingCheckBox (MainGUI class)	JCheckBox	*selected	Selection means that the student will participate in Impromptu Speaking	
participantsAssignEventsDuetActingCheckBox (MainGUI class)	JCheckBox	*selected	Selection means that the student will participate in Duet Acting	
participantsAssignEventsDebateCheckBox (MainGUI class)	JCheckBox	*selected	Selection means that the student will participate in Debate	
participantsAssignPairsEventsList (MainGUI class)	JList	*“DA” *“Debate”	First step to assigning pairs – user chooses event in which to pair students	Lists specific for the assignment of pairs
participantsAssignPairsLeftList (MainGUI class)	JList	*“B13” * “B15” *“B19”	List for choosing the first member of the pair. (Only students from given school participating in the selected event are shown.)	
participantsAssignPairRightList (MainGUI class)	JList	*“B15” *“B19”	List for choosing the second member of the pair. Shown after the first member was selected, it does not duplicate that student in the list.	
participantsAssignPairsPairsList (MainGUI class)	JList	*“B13-B19”	List of pairs already assigned for given school	
eventSettingsOOSR (MainGUI class)	JTextField	*“6”	Setting, the number typed determines the value of the ooStudentRoomLimit variable (Database class)	An example of the function of the event settings text fields at the Settings tab

<i>generalSettingsTS (MainGUI class)</i>	<i>JTextField</i>	<i>*“2”</i>	<i>Setting, the number typed determines the value of the teacherSchoolLimit variable (Database class)</i>	<i>An example of the function of the general settings text fields at the Settings tab</i>
<i>participantsSchoolsTextField (MainGUI class)</i>	<i>JTextField</i>	<i>*“Vienna International School”</i>	<i>Text field for the input of information for the Schools list</i>	<i>An example of the typical bundle of text field and a list, working together</i>
<i>participantsSchoolsList (MainGUI class)</i>	<i>JList</i>	<i>*“ISP” *“ISB” *“VIS” *“BBIS”</i>	<i>A list, displaying the contents of the schoolTree (Database class). Clicking a list item usually reveals other lists – here, studentList and teacherList – and allows their editing. * (Full names of schools would be used in the actual list, not acronyms.)</i>	<i>An example of the typical bundle of text field and a list, working together</i>
<i>qualificationEventList (MainGUI class)</i>	<i>JList</i>	<i>*“OO” *“OI” *“IS” *“DA” *“Debate”</i>	<i>A list, displaying all the available events on tournament. Still clickable to reveal other lists – here, the roundList – and allow their editing. It is not editable itself as the events are not changing. * (Full names of events would be written in the actual list.)</i>	<i>An example of a non-editable list (without a text field)</i>
<i>winnersDuetActingFirstPlaceList (MainGUI class)</i>	<i>JList</i>	<i>*“__B19” *“__B13”</i>	<i>A list displaying all members of a winning entity – their name (in place of the __'s) along with their codes. If it is a non-pair entity (OO, OI, IS), only one item is shown for only one member of a given entity.</i>	<i>An example of the text fields at the Winners tab</i>

Data Structures:

Identifier	Type	Example	Description	Discussion
schoolTree (MainGUI class)	BinarySearchTree	{ISB, ISP, VIS}	An ADT to hold schools added to the database (the School object then has a teacherTree and a studentTree)	Initialized as empty trees, these are added to by the GUI addButtons. The entries can be renamed by the editButtons, and deletion is also possible, thanks to the removeButtons. (All of these belong to the Participants section of the GUI.)
teacherTree (Database class)	BinarySearchTree	{Barlien, Caskie, O'Murchu}	Holds teachers added to particular school	
studentTree (Database class)	BinarySearchTree	{Arwan, Friin, Stara}	Holds students added to particular school	
eventArray (Student class)	boolean []	{true, true, false, false, false}	The events this student participates in	Initialized with only false entries, this is changed when events are set for the student with the assignEvents button
eventArray (Qualification class)	Event []	{OO, OI, IS, DA, Debate}	The events of the qualification (these then hold different rounds)	These arrays are initialized with the five events and are never changed.
eventArray (Finals class)	Event []	{OO, OI, IS, DA, Debate}	Analogous to the eventArray of Qualification class, this stores the events of the finals (which then hold rounds)	
roundArray (Event class)	Round []	{DA-Q1, DA-Q2}	This array stores the rounds for this event	These arrays are initialized when events are constructed – mostly two entries, debate has three. The initialization is never changed.
finalistArray (QualificationEvent class)	Entity []	{A1-A3, C14-C23}	The finalist entities of this event	Initialized as empty. The finalists are determined from entities by the sum of scores in the qualification (for debate, they are selected by hand from semifinalists, see below).
semifinalistArray (DebateEvent class)	Entity []	{B13-B12, C25-C28, A2-A9, D34-D36}	The semifinalist entities of this debate event	Initialized as empty. The semifinalists are determined by win/loss ratio and scores in debate rounds.
winnersArray (FinalEvent class)	Entity []	{A2-A9, D34-D36, A1-A3, A2-A5}	The array of winners of this final event	Initialized as empty. The winners are determined by the sum of rankings in the final round.
judgesTree (Round class)	BinarySearchTree	{Barlien, Caskie, O'Murchu}	The tree from which judges are drawn for judging one of the round's rooms	Initialized with teachers, as inputted to the Participants tab. This does not change afterwards.
entityTree (Round class)	BinarySearchTree	{A1-A3, A2-A5, B11-B17, B14-B22, C14-C23}	The tree of entities (individuals or pairs) for this event	Initialized with all individuals/pairs, to which this event was assigned. This does not change afterwards.
freeEntityArray (Round class)	Entity []	{A1-A3, A2-A5, B11-B17, B14-B22, C14-C23}	Array with all entities of this round which are not in any room yet – which will still have to be accounted for	Initialized with a tree of all entities with this event. Entities processed by the allocation algorithm are removed. The program can only proceed if there are no entities left.
freeJudgeArray (Round class)	Judge []	{Barlien}	Array of judges which were not called initially (it's their free time); might be called upon if a judge needs to be substituted	Initialized after the primary allocation is done and entities and judges distributed. These are the spare judges for this round. Changed when a judge is substituted.
roundArray (Judge class)	Round []	{OO-Q1, IS-Q1, Debate-Q1, OI-Q2, Debate-Imp, DA-Q2, Debate-Q2}	Array of rounds this judge attends	Initialized according to the outputs of the allocation algorithm. After that, this only changes if the judge substitutes for someone or is substituted.
studentArray (Entity class)	Student []	{Stara, Borovička}	Array of students in this entity (either one – if it is an individual –, or two – if it is a pair)	Constructed according to the data from Participants tab – the events and pairs information. After that, this array does not change.

roundArray (Entity class)	Round []	{OO-Q1, OO-Q2}	The rounds this entity is participating in	This is determined during the allocation phase. After that, this array does not change.
scoreArray (Entity class)	int []	{24, 21, 18, 26}	The scores this entity has gained in the rounds	Size of this array is determined to be (number_of_rounds * number_of_judges). It is initialized with flag values -1, rewritten when scores are submitted from the rounds this entity participates in.
rankingArray (Entity class)	int []	{1, 1, 2, 4, 4}	The rankings this entity has gained from the judges in finals	Size of this array is found as the number of finals judges. It is initialized with -1's, and rewritten during the finals.

Two data structures will be used primarily in this program – arrays and binary search trees. These are used in different situations because of their different strengths and weak spots. Arrays are used in cases, in which it is known beforehand how many entries to the structure there will be. Moreover, in those cases, it is important to know exactly which entry we are addressing, and for this, the array system (e.g. roundArray[2]) is the most efficient.

Binary search trees are, then, to be utilized when it is not known in advance how many entries will be added into the structure, while keeping a quick way to access, search for and add entries is desired. Thus, this structure is ideal for storing data about schools, students, teachers, judges, and entities – the number of which is not known in advance, as it is inputted by the user. Linked lists theoretically achieve the same effect – however, these are not desirable, as they would lack the advantages of binary search trees (such as automatic ordering of new nodes and effective searching). Hash tables and arrays are similarly unsuitable for these tasks, since they are wasteful most of the time and always finite in length.

Instances of Custom Objects:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
database (MainGUI class)	Database		The overarching object holding methods and attributes for manipulating the database part of the program	These three classes were created to bring more organization into the system of methods and attributes, which would otherwise have to be jumbled together in the MainGUI class.
qualification (MainGUI class)	Qualification		The overarching object holding methods and attributes for manipulating the qualification part of the program	
finals (MainGUI class)	Finals		The overarching object holding methods and attributes for manipulating the finals part of the program	
school (Teacher class)	School	International School of Prague	The school this teacher comes from	This information is important to know for the qualification creation phase.
school (Student class)	School	International School of Prague	The school this student comes from	
teacher (Judge class)	Teacher	Caskie	The teacher, of which this judge is a representation	Judges correspond to teachers, input in the Participants tab. This class holds this information.
school (Judge class)	School	International School of Prague	The school this judge comes from	Important to know for the room allocation phase of the program.
school (Entity class)	School	International School of Prague	The school the members of this entity come from	
room (Round class)	Room	Room 3	A room, in which one set of the round's entities compete	There are several rooms for each round. They hold individual entities and judges.

There are also other custom object instances used in the program in arrays; these are discussed in the Data Structures section above.

File Objects:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
br (MainGUI class)	BufferedReader	BufferedReader br = new BufferedReader(new FileReader())	Reads information in from a saved file – takes care of the loading of information	These methods together facilitate the sequential file functionality
(MainGUI class)	FileReader		<i>Facilitates the reading proper of the file</i>	
bw (MainGUI class)	BufferedWriter	BufferedWriter bw = new BufferedWriter(new FileWriter())	Writes information in to a file (either old, or newly created) – takes care of saving	These methods together facilitate the sequential file functionality
(MainGUI class)	FileWriter		<i>Facilitates the file access proper</i>	

Data Structures Used:

What follows is an overview of all the classes used in the program and the attributes they are using. A brief description always accompanies the class. Attributes and data structures that were not anticipated at the design stage are highlighted with yellow color.

Classes in the Dossier Program:

- gui package:
 - MainGUI
- data package:
 - Database
 - School
 - Teacher
 - Student
 - StudentPair
 - DAStudenPair *extends StudentPair*
 - DebateStudentPair *extends StudentPair*
 - Qualification
 - Event
 - QualificationEvent *extends Event*
 - Round
 - Room
 - Judge
 - Entity
 - BinarySearchTree
 - *BinarySearchTree.Node*

Class MainGUI

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
<code>currentNewNoNameSchoolNumber</code>	<code>int</code>	<code>1</code>	Number of schools added without a name	Example of an int attribute storing the number of unnamed elements in database lists.
<code>schoolsNumber</code>	<code>int</code>	<code>1</code>	Number of schools currently in the database	Example of an int attribute storing the number of elements in database lists.
<code>reassignmentText</code>	<code>boolean</code>	<code>true</code>	Whether the text on assignEventsButton should read “Re-assign events”	
<code>studentCodes</code>	<code>boolean</code>	<code>true</code>	Whether student codes were already assigned	
<code>studentChanges</code>	<code>boolean</code>	<code>true</code>	Whether there were changes made after codes were assigned	
<code>schoolsIndex</code>	<code>int</code>	<code>1</code>	Index of the currently selected item in schoolsList	Example of an int attribute storing the index of currently selected element in database lists.
<code>assignPairsEventsIndex</code>	<code>int</code>	<code>1</code>	Index of the currently selected item in assignPairsEventsList	Example of an int attribute storing the index of currently selected element in Student Pairs part of the Participants tab.
<code>eventsIndex</code>	<code>int</code>	<code>1</code>	Index of the currently selected item in qualification's eventsList	Example of an int attribute storing the index of currently selected element in qualification lists.

Arrays:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
<code>approvedSections</code>	<code>Boolean []</code>	{true, false, false, false, false, false}	Which sections were approved for saving	Initialized as {true, false, false, false, false, false}, changed when a section is approved/disapproved

Abstract Data Types:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no ADTs in this class

Object Instances:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
<code>database</code>	<code>Database</code>		Contains methods to work with participants, holds all the participants' data	
<code>qualification</code>	<code>Qualification</code>		Contains methods to work with judges and entities for the qualification	

Files and File Readers/Writers:

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
<code>loadFile</code>	<code>File</code>	<code>~/Documents/file.txt</code>	Stores the last file that was used to load information to the program.	
<code>br</code>	<code>BufferedReader</code>		Facilitates reading of	In readFile() method.

			files	
bw	BufferedWriter		Facilitates writing into files	In saveFile() method.

GUI Objects:

Identifier	Type	Example	Description	Discussion
participantsAssignEventsOriginalOratoryCheckBox	JCheckBox	*selected	Selection means that the student will participate in Original Oratory	Example of a check box for the specification of events which a given student is to participate in
participantsAssignPairsEventsList	Jlist	*“DA” *“Debate”	First step to assigning pairs – user chooses event in which to pair students	Lists specific for the assignment of pairs
participantsAssignPairsLeftList	Jlist	*“B13” * “B15” *“B19”	List for choosing the first member of the pair. (Only students from given school participating in the selected event are shown.)	
participantsAssignPairRightList	Jlist	*“B15” *“B19”	List for choosing the second member of the pair. Shown after the first member was selected, it does not duplicate that student in the list.	
participantsAssignPairsPairsList	Jlist	*“B13-B19”	List of pairs already assigned for given school	
eventSettingsOOSR	JtextField	*“6”	Setting, the number typed determines the value of the ooStudentRoomLimit variable (Database class)	An example of the function of the event settings text fields at the Settings tab
generalSettingsTS	JtextField	*“2”	Setting, the number typed determines the value of the teacherSchoolLimit variable (Database class)	An example of the function of the general settings text fields at the Settings tab
participantsSchoolsTextField	JtextField	*“Vienna International School”	Text field for the input of information for the Schools list	An example of the typical bundle of text field and a list, working together
participantsSchoolsList	Jlist	*“ISP” *“ISB” *“VIS” *“BBIS”	A list, displaying the contents of the schoolTree (Database class). Clicking a list item usually reveals other lists – here, studentList and teacherList – and allows their editing. * (Full names of schools would be used in the actual list, not acronyms.)	
qualificationEventList	Jlist	*“OO” *“OI” *“IS” *“DA” *“Debate”	A list, displaying all the available events on tournament. Still clickable to reveal other lists – here, the roundList – and allow their editing. It is not editable itself as the events are not changing. * (Full names of	An example of a non-editable list (without a text field)

			<i>events would be written in the actual list.)</i>	
settingsErrorDialog	Jdialog		A dialog displayed when one of the settings value cannot be parsed. It identifies the offending textField and does not let the user proceed unless the error is corrected.	
exceptionDialog	Jdialog		An all-purpose dialog displayed when an error is encountered.	
illegalActionDialog	Jdialog		A dialog displayed when the user tries to add an illegal element to one of the trees.	
resetWarningDialog	Jdialog		A warning to the user that he tries to revert to an earlier version of the file.	
approveParticipantsDialog	Jdialog		An information dialog about variables which do not have the expected value (8 schools were added to the database, but 12 are expected due to the settings).	
allocationInfoDialog	Jdialog		A pop-up dialog showing the progress of the qualification allocation.	

MainGUI is the main class of the program. It contains all the GUI elements and also creates both of the hierarchical data structures: database and qualification. Much of the program's error checking of erroneous input is also performed through MainGUI's methods. MainGUI is also responsible for handling the events triggered by users' clicking on buttons and in lists.

It can be seen that much of the GUI was expected beforehand, but all the necessary primitives were figured out while coding, as was the need for additional dialogs. Also the loadFile has been added, to store the last save location for resetting the database.

Class Database

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
<i>ooStudentRoomLimit</i>	<i>int</i>	<i>6</i>	Setting, maximal number of students per room for OO event	An example of a variable storing a value of a setting variable.
<i>defaultOOStudentRoomLimit</i>	<i>int</i>	<i>6</i>	The default value of the <i>ooStudentRoomLimit</i> setting	An example of a variable storing a default value of a setting variable.
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				No arrays in this class.
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
<i>schoolTree</i>	<i>BinarySearchTree</i>	{ISB, ISP, VIS}	An ADT to hold schools added to the database (the School object then has a teacherTree and a studentTree)	A tree holding data about schools in the database. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no object instances in this class
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

The Database class contains the schoolTree. Technically, the schoolTree could have been a part of the MainGUI, but logically, it makes bigger sense to have it as a part of a separate hierarchical data structure – hence, the Database.

It can be seen that it was not originally expected for the database to actually hold the schoolTree, however, during the coding this proved to be the most logical solution. Also, the necessary primitives were figured out while coding

Class School

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
name	String	“ISP”	Name of the school	
codeLetter	char	‘A’	The letter used to signify this school in student codes	
beginCodeNumber	int	1	First student code number belonging to the school	An example of an int attribute of the class. Value is accessed directly by get/set methods.
ooStudentNumber	int	6	The number of students in the school	An example of a int attribute of the class, whose value is changed only by incrementation or decrementation.

Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				No arrays in this class.

Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
teacherTree	BinarySearchTree	{Barlien, Caskie}	An ADT to hold teachers to the school.	Identical in use to the studentTree, this tree holds data about teachers in the school. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
unpairedDAStudentTree	BinarySearchTree	{Borovička, Trinh, Zhou}	An ADT to hold students who are not participating in the duet acting event	Along with a similar unpairedDebate StudentTree, this tree holds students who do not participate in duet acting. It is needed to have this tree to identify whether these students are available to make a new DA pair. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
daPairTree	BinarySearchTree	{A1-A3}	An ADT to hold pairs of students	Along with a similar debatePairTree, this tree holds duet acting pairs. We need this because this data is used by qualification to produce DA/debate entities. By using a

				<i>tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.</i>
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no object instances in this class</i>
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no files or file readers/writers in this class</i>
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no GUI objects in this class</i>

School contains teachers and students trees as well as the unpaired student and pairs trees. As can be seen by the yellow color, this is not what was expected at the design stage. (It was expected that the Database would hold these trees.) However, this solution is much more logical, because the teachers and students are always shown and worked upon according to schools. Therefore it makes sense to have the teacher- and studentTrees inside School. Also, the necessary primitives were not anticipated at the design stage.

Class Teacher

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
name	String	“Caskie”	Name of the teacher	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no arrays in this class</i>
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no ADTs in this class</i>
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
school	School	ISP	The school this teacher belongs to	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no files or file readers/writers in this class</i>
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no GUI objects in this class</i>

A teacher is just identified by the name and school. Nothing has changed in the Teacher class between the design stage and the final stage.

Class Student

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
name	String	"Stara"	Name of the student.	
code	String	A1	Code of the student.	
daUnpaired	boolean	true	Value showing whether the student is unpaired in duet acting or not.	Identical is the debateUnpaired variable, which applies to debate.
daStudentPairTempString	String	"A1-A3	A temporary variable to store the name of the student pair this student belongs to before the pairs tree is initialized (while loading)	Identical as the debateStudentPairTempString, which applies to debatePair
reassignmentText	boolean	false	Value specifying whether there should be "Re-assign events" on the assign events button. This is always true except for the first time this student is encountered.	

Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
events	boolean []	{false, false, false, false, false}	An array, which is true at indices corresponding to events the student was assigned to.	

Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no ADTs in this class

Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
school	School	ISP	The school this student belongs to.	
daStudentPair	DAStudentPair	A1-A3	The student pair of which this student is a part.	Identical in usage to the debateStudentPair.

Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class

GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

Student has required more primitive variables, as well as an indication of the studentPair it is a part of. Other changes to my design plans were not necessary.

Class StudentPair(extended by **DAStrudentPair** and **DebateStudentPair**)

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
originalCode	String	"A1-A3"	The code under which this pair has been entered into the pair tree.	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no arrays in this class
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no ADTs in this class
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
student1	Student	Stara	The first student belonging to this pair.	Identical to the student2, the second student in the pair.
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

Neither the StudentPair class nor its extensions were anticipated at all at the design stage. They were included during the coding because they make a logical addition to the levels of hierarchy in the data system and allow for an easy conversion to duet acting/debate entities.

Class Qualification

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
eventArrayLength	int	2	Number of events for which the entities were created	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
eventArray	QualificationEvent []	{OO, OI, IS, DA, Debate}	An array of all possible events	
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
judgeTree	BinarySearchTree	{Barlien, Caskie}	An ADT to hold all judges created.	This tree holds data about judges created for this school. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
ooEntityTree	BinarySearchTree	{Borovička, Trinh, Zhou}	An ADT to hold entities for the Original Oratory event	An example of a tree holding entities for different events, this tree holds entities for original oratory. It is needed to have this tree to identify whether these students are available to make a new DA pair. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no object instances in this class
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

Qualification was made more important with the addition of the trees.

Class Event

(extended by QualificationEvent)

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
shortEventName	String	"OO"	The shortened name for this event	
judgesPerRoom	int	2	Number of judges needed in this room	
entitiesPerRoom	int	6	Maximum number of entities in this room	
isSemifinal	boolean	false	Whether this event is a semifinal	
isFinal	boolean	false	Whether this event is a final	
debateSemifinalTime	String	"Day 2, 08-09"	The time at which the debate semifinal is happening	This is purely for orientation, derived from personal experience on NESDA tournaments. It is not calculated, nor binding.
ooFinalTime	String	"Day 2, 10-11"	The time at which the oo final is taking place	An example of similar FinalTime Strings for all events.
<i>Subclass QualificationEvent</i>				
maxCombinations	long	123234421	The maximum number of combinations for the given organization of judges and entities	Determines how much is a 100 % in the progress bar.
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
roundArray	Round []	{OO R1, OO R2}	An array of all rounds required for this event	
ooRoundTimes	String []	{"Day 1, 11-12", "Day 1, 16-17"}	A string array with the times of original oratory rounds	One of roundTimes arrays, which store strings with times for different rounds.
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
judgeTree	BinarySearchTree	{Barlien, Caskie}	An ADT to hold all judges created.	This tree holds data about judges created for this school. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
entityTree	BinarySearchTree	{Borovička, Trinh, Zhou}	An ADT to hold entities for the event	This tree holds data about entities in this event. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
qualification	Qualification		The qualification this event belongs to	
type	Type	ORIGINAL_ORATORY	The type of this event	Specified in the sub-enum of this class, Type: ORIGINAL_ORATORY, ORAL_INTERPRETATION.

				IMPROPTU_SPEAKING, DUET_ACTING, DEBATE, UNDEFINED;
Subclass QualificationEvent				
combinationFormat	java.text. NumberFormat	"000000000000"	The number format for currentCombination.	Adds leading appropriate number of leading zeros to currentCombination, so that it is as long as maxCombinations.
thisEvent	QualificationEvent	OO	Reference to this event, needed for communication with AllocationWorker.	
task	AllocationWorker		New task to automatically update allocationInfoDialog to show progress in allocation	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				<i>no files or file readers/writers in this class</i>
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>no GUI objects in this class</i>

The Event has seen a lot added to itself during the programming. All of the primitives had to be added, as was the array of roundTimes and all of the object instances. For instance, it was not known at all at the design stage what objects would be needed for the progress reporting.

Also important is the addition of the Type sub-enum to the Event, because it is used for event-type identification purposes throughout the qualification hierarchy. Note, as well, the addition of the AllocationWorker, the indispensable background-task handling class that allowed the reporting of the progress in the allocation.

Class Round

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
name	String	“OO R1: Day 1, 11-12”	The name of this round.	
judgesPerRoom	int	2	The number of judges in each room.	
entitiesPerRoom	int	6	The max number of entities in each room.	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
roomArray	Room []	{Room 1, Room 2, Room 3, Room 4}	An array of all rooms in this round.	
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
judgeTree	BinarySearchTree	{Barlien, Caskie}	An ADT to hold judges for this round.	This tree holds data about judges for this round. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
entityTree	BinarySearchTree	{Borovička, Trinh, Zhou}	An ADT to hold entities for this round.	This tree holds data about entities for this round. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
freeJudgeTree	BinarySearchTree	{Barlien}	An ADT to hold currently available judges for allocation.	In this tree, there are only judges which were not allocated yet. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
freeEntityTree	BinarySearchTree	{Borovička, Trinh, Zhou}	An ADT to hold currently available entities for allocation.	In this tree, there are only entities which were not allocated yet. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				

<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
event	Event	OO	The event this round belongs to	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

In the round, it was not expected that an indication of the number of judges and entities per room would be needed, as was not expected the need for a back-reference to the round's event. Furthermore, the anticipated freeJudge/freeEntityArrays were transformed into freeJudge/freeEntityTrees.

Class Room

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
name	String	“Room 1”	The name of this round.	
judgesLimit	int	2	The required number of judges.	
entitiesLimit	int	6	The maximum number of entities.	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no arrays in this class
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
judgeTree	BinarySearchTree	{Barlien, Caskie}	An ADT to hold judges for this round.	This tree holds data about judges allocated to this room. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
entityTree	BinarySearchTree	{Borovička, Trinh, Zhou}	An ADT to hold entities for this round.	This tree holds data about entities allocated to this room. By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
round	Round	OO RI	The round this room belongs to	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

At the design stage, it was not anticipated that the judges and entities would be stored in Rooms, which proved very logical solution. The limit attributes are new, too.

Class Judge

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
ID	int	1	The internally assigned ID of this judge.	The ID is assigned according to alphabetical order of judges at the time of creation of the qualification.
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no arrays in this class
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
encounteredJudgeTree	BinarySearchTree	{Caskie}	An ADT to hold judges this judge has already been judging with (and therefore cannot be with again).	By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
encounteredJudgeTempIDTree	BinarySearchTree	{1}	An ADT to hold ID's of encountered judges. This is needed to allow for adding encountered judges before the judge tree is completely constructed (as in loading).	By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
school	School	ISP	The school this judge belongs to.	
teacher	Teacher	Barlien	The teacher this judge is constructed from.	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

A new widely-used identification attribute, ID, was devised during the coding for the judges. It is much better than distinguishing them by school-name combination. Also, the encounteredJudgeTree was found necessary for the allocation algorithm, as was the encounteredJudgeTempIDTree for loading.

Class Entity

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
isPair	boolean	false	True if the entity is a pair (as entities for DA and debate are).	
code	String	"A2"	The code of the entity (in the form "A1-A3" if it is a pair).	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
studentArray	Student []	{A1, A3}	The students making up this entity (if it is not a pair, the student array has 1 element with the constituent entity)	
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
encounteredJudgeTree	BinarySearchTree	{Caskie}	An ADT to hold judges this entity has already been judged by (and therefore cannot be judged by them again).	By using a tree, I am able to have the data automatically sorted, which would not be the case if I used a linked list. Moreover, a tree is not limited in size as an array would be.
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
type	Event.Type	ORIGINAL_ORATORY	The event type, as defined in Event class's Type enum.	
school	School	ISP	The school this student belongs to.	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

The Entity now newly stores also the encounteredJudgeTree for allocation algorithm. Also, type was not anticipated. And the code was deemed necessary during the programming as well, as an easy identification attribute.

Class BinarySearchTree

(Abstract Data Type)

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
size	int	0	The size of the tree.	
dataArrayPositionCounter	int	0	The current position of dataArrayHelper within the tree.	
dataArrayChanged	boolean	true	Whether the dataArray was changed (and therefore new has to be made before returned).	
nodeArrayPositionCounter	int	0	The current position of nodeArrayHelper within the tree.	

Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
dataArray	Object []		A sorted array of the data of all nodes in the tree.	
nodeArray	Node []		A sorted array of all the nodes in the tree.	

Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no ADTs in this class

Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
root	BinarySearchTree.Node		The root of the binary search tree	see class BinarySearchTree.Node for details

Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class

GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

This essential abstract data type, the BinarySearchTree, was not included in the design stage, that means that it had to be figured out completely during the coding. That is the reason for all the yellow color.

Class *BinarySearchTree.Node*(subclass of *BinarySearchTree*)

Primitives:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
data	Object		The data saved in the node	
keyString	String	"Caskie"	String, according to which the node is sorted.	
Arrays:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no arrays in this class
Abstract Data Types:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no ADTs in this class
Object Instances:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
left	BinarySearchTree.Node		The node left of the current node.	
right	BinarySearchTree.Node		The node right of the current node.	
Files and File Readers/Writers:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no files or file readers/writers in this class
GUI Objects:				
<i>Identifier</i>	<i>Type</i>	<i>Example</i>	<i>Description</i>	<i>Discussion</i>
				no GUI objects in this class

The Node, as a subclass of the BinarySearchTree, was also not anticipated during the design stage. That is, again the reason for all the yellow color.

Section B2: Algorithms

Initial Version of Algorithms:

Save File:

Pre-conditions:

User clicks one of the Save buttons in the program.

Post-conditions:

The program contents for all sections of the program are saved to a file.

Description:

A JFileChooser dialog is invoked, in which the user selects a file to which to save, or specifies a new one.

The save file has several imaginary sub-sections, according to the sections of the program (Settings, Participants...); these would be separated by distinctive tokens.

If a section of the program was not entered yet, the program saves a notice about that at the start of that section's space.

If the section was entered already, the program writes the values of important variables (mainly for the Settings section) into the sub-section of the file. Furthermore, the program transcribes the contents of all binary search trees of that section and writes them into a corresponding sub-section of the file. The hierarchy of the binary search trees is conserved with the use of tokens.

Load File:

Pre-conditions:

User clicks the Load button in the Settings section of the program.

Post-conditions:

The program contents from a previous saved session of the program are loaded from a file.

Description:

A JFileChooser dialog is invoked, in which the user selects a file to be loaded into the program.

Notices about availability of sub-sections are checked for, so that no errors are recorded from expecting data not entered into the file.

The program assigns the values from the file to specific variables. It is able to do so thanks to the use of tokens. In a similar way, it is able to reconstruct the binary search trees from the values recorded in the file.

Export section:

Pre-conditions:

User clicks the Export button in one of the sections of the program.

Post-conditions:

A file is generated with the plain-text contents of a section of the program.

Description:

A JFileChooser dialog is invoked, in which the user selects a file to which to export the information, or specifies a new one.

The contents of the trees of a given section are written out into the file separated with tabs (allowing for better orientation in the data) and saved.

The resulting file stores all the information from a given section of the program in plain-text form. This allows for better accessibility of the data stored in the program, which can be easily copy-pasted into a word processor and formatted at will.

Construct Database:

Pre-conditions:

The program was started.

Post-conditions:

The basic structure of the program's database is created.

Description:

An instance of the Database object is created and the default values are applied to the setting variables.

An empty School binary search tree is constructed, to be filled in with user input. (A binary search tree is used here, because it is an extremely effective data structure and it is automatically sorting itself.)

Construct Qualification:

Pre-conditions:

The Participants section of the program is approved.

Post-conditions:

The Qualification section of the program is determined.

Participants in different events are distributed into rooms and rounds, as well as the judges for these events.

Description:

Create an instance of the Qualification object. Fill that object's eventArray with four QualificationEvent and one DebateEvent objects, as corresponding to the five NESDA events.

For each of those Events, construct an array of Rounds, according to the NESDA rules (2 for OO/OI/IS/DA events, 3 for Debate events).

For each round, make a copy of the specific *eventEntityHashTable* of the corresponding event. Also make a copy of the *judgeHashTable*. (Hash tables are used here because they are better suited to getting alphabetically sorted entries while still being effective for saving data.)

For each round, construct an array of Rooms, so that there are approximately as many students as the setting *eventStudentRoomLimit* mandates. (The number of rooms, however, may not be higher than the value of a general setting of the program, the *roomLimit*.)

For each room, choose six entities at random from the round's copy of the *eventEntityHashTable* (after, the entity is deleted from the hash table) and put them one by one to the *entityArray* of that room. Then add two judges from the copy of *judgeHashTable* to the *judgeArray* of the same room.

Several rules apply to this process of room allocation:

1. None of the judges may come from the same school as one of the participating entities.
2. There must not be more than one entity from each school in one room.
3. In more advanced rounds, no judge may see one participant for a second time.
4. No two entities that were together in a round can be together again.

(Notice, however, that these rules do not apply across events. One student may be judged by the same teacher in OO and OI event)

If it is found during the room allocation step of the algorithm that one of these rules cannot be honored anymore for a certain room, disperse the previous room's entities and judges and choose a different assortment of entities and judges for that room.

If a satisfactory room organization for a more advanced round was not found even after all possibilities of reordering were exhausted, it means that the previous round as a whole has to be re-entered and its rooms reordered.

If a satisfactory room organization was not found even for the first round, try to find an acceptable organization while breaking one of the rules (in order from the least important, nr. 4, to the most important, nr. 1). Furthermore, display an alert window to the user and inform him or her that some of the rules of room allocation will be broken in the qualification section.

If a room organization is found for all rounds of all events at last, the Qualification was successfully constructed and the user may proceed to entering scores for participants of the different rounds.

Construct Finals:

Pre-conditions:

The Finalists section of the program is approved.

Post-conditions:

The Finals section of the program is determined.

The finalists are in their final rounds, and judges are determined to score them.

Description:

Create an instance of the Finals object. Fill that object's eventArray with five FinalEvent objects, as corresponding to the five finals. For each of those Events, then, construct one Round. Construct one Room in each.

For each final, choose five judges from a copy of the judgeHashTable, so that the rules of judge allocation are fulfilled. (No judge may have judged a finalist in the qualification of the event, and no judge may come from the same school as the finalist. As in qualification, however, these rules do not apply across events – one student may be judged by the same teacher in OO and OI final provided that he meets the criteria.)

If a satisfactory room organization cannot be found for a final, try to find an acceptable organization while breaking one of the rules (first the one about judges coming from the student's school). Furthermore, a major warning has to be displayed to the user informing him or her that some of the rules of judge allocation will be broken. (Which should never happen.)

If judges were found for every final, the Finals were successfully constructed and the user may proceed to entering rankings and student choice votes to the finalists in the Finals section.

Assign Student Codes:

Pre-conditions:

The Database is constructed and there is at least one student recorded in it.

Post-conditions:

All current students in the database are assigned a unique student code.

Description:

The algorithm loops through the tree of schools in alphabetical order and assigns each a letter of the alphabet.

For each school, the algorithm loops through the tree of students. To each of them, it assigns a number from one. (Important: the numbering is not re-started with a new school, so that each number is unique.)

The two parts of the code are combined into a single string (“C34”, for example), which is then stored by the Student object as the studentCode.

Construct Entities:

Pre-conditions:

The database was approved.

Post-conditions:

There are five hash tables of entities – separate for entities from every event.

Description:

Loop through schools in the schoolTree of the database. For each school, loop through its studentTree.

For each Student, construct a corresponding Entity in every event he participates in and put him into a corresponding hash table. For pair events, look for the chosen student’s partner in the hash table. If the partner is found, it means that there already is an entity for both, and therefore no new entity is created for current student in the pair event in question. This applies to the two pair events separately, as one student can be paired with two different people for the two pair events.

Construct Judges:

Pre-conditions:

The database was approved.

Post-conditions:

There is a hash table of judges.

Description:

Loop through schools in the schoolTree of the database and for each school, loop through its teacherTree.

For each Teacher, construct a corresponding Judge and put him or her into a united hash table.

Detect Completed Qualification:

Pre-conditions:

One of the qualification section's Approve buttons is clicked.

Post-conditions:

A Boolean array is returned with the information of which events are completed.

Description:

The algorithm loops through all entityArrays of the Qualification and checks the number of scores each entity has. If there are more – or fewer – scores than the number of judges, the event is deemed incomplete (“false” in the returned array). Else, the event is deemed complete (“true” in the array).

Determine Finalists:

Pre-conditions:

One of the approve buttons is clicked in the Qualification section of the program.

The eventCompletionArray has at least one true entry (see above).

Post-conditions:

The finalists are determined for the use in the Finalists section.

Description:

The algorithm detects completed events and loops through a copy of the hashTable of entities for that event.

It creates an array of six entities (default value of the corresponding setting) from that event with the highest average score (combined from all rounds). (Arrays are used here because we know the number of elements from the beginning and the order of the elements.) Ties are broken by looking at the average from the first round (for debate, it is the number of wins).

When the algorithm reaches the end of the hashTable, the entities in the array are the finalists of the event.

If there is a tie or a very small difference between the 6th and 7th ranking participant, the program asks the user if it would be possible to extend the limit. Similarly, if there is a huge gap between the 5th and the 6th place, the program suggests to drop the 6th participant. The final decision, however, is entirely up to the user of the program.

Determine Winners:

Pre-conditions:

The Finals section of the program is approved.

Post-conditions:

The three best-ranking participants (or pairs) in the events are determined, as well as the participant with the highest number of student choice votes.

Description:

The algorithm looks through the rankings of the finalists for each event (each judge is asked to rank the finalists, if a participant is 1st, he gets 1 point, 2nd gets 2, 3rd gets 3, and all others get 4), adds them all up, and the finalist with the least result is the winner. The second-ranking finalist gets the 2nd place, and the third-ranking finalist gets the 3rd.

(Ties are broken by the number of points in the qualification, or the number of points in the 1st round. If the tie cannot be broken, two places of the same rank are awarded, for example two 1st and a 3rd, or one 1st and two 2nds, or one 1st, one 2nd and two 3rds.)

The student choice votes are recorded for each finalist and the one with the most votes wins the student choice award.

(These exact rules apply for pairs as well. The only difference to individual participants is that since pairs have two students, both of them are listed in the GUI.)

Final Version of Algorithms:

During the coding, all of the algorithms have changed from what they were devised to be, the allocation algorithm a lot. That is understandable, as by the time of the design process it was very difficult to predict what all will need to be done.

Save File

(in saveFile() in MainGUI class)

Pre-conditions:

User clicks one of the Save buttons in the program.

Post-conditions:

The program contents for all the sections up to the one including the save button clicked are saved. (If Settings, Participants and Qualification tabs all have data and the user clicks Save in the Participants, only Settings and Participants will be saved. If the Qualification's save button is clicked, all three sections are saved.)

Description:

A JFileChooser dialog is invoked. The user selects a file to which to save, or specifies a new one.

The save file has several imaginary sub-sections, according to the sections of the program (Settings, Participants...). Each one of these is introduced by a notice with the “\$” sign. That way, the program knows it has reached another section.

If a section is beyond the section up to which the file is saved, it is skipped.

The algorithm first writes the file notice, then the values of MainGUI attributes, then the values of settings one by one.

If Participants are being saved, it loops through the school tree, writing attributes of each school to the file, then looping through its teacherTree and studentTree and writing their information as well, so that all the contents of the school are saved. (The attribute used for identification of the class is written first, and then, after a colon “：“, the other attributes are written out, separated by comma “，” tokens.

If Qualification is being saved as well, the program loops through rounds, looping through each one's rooms, and each room's judge and entity trees, writing out the information so these can be exactly reproduced when loading.

Load File

(in loadFile() in MainGUI class)

(uses readFile() in MainGUI class)

Pre-conditions:

User clicks the Load button in the Settings section of the program.

Post-conditions:

The program contents from a previous saved session of the program are loaded from a user-specified file, if the file format is right. Otherwise, an exception is thrown, which invokes an exception dialog to the user.

Description:

A JFileChooser dialog is invoked. The user selects a file from which to load the information. The address of the file is saved. Then, the file is read by the readFile() method.

First, the MainGUI attributes are loaded, including the approvedSectionsIndex. If a section has not been approved (as specified by the approvedSectionIndex), then, that section, and all subsequent ones, is skipped.

The program loops, reading the section line by line, and depending on the starting characters of each line determines what information it is taking in at a given moment. According to that, it parses the information to properly reproduce the information in its class hierarchy.

If a blank line is encountered, it means an end of the section has been found, and that therefore, next section can be taken in. Therefore, the open objects are wrapped up and the reconstruction of the hierarchy is completely finalized

Export Section

(in exportParticipantsActionBox() and exportQualificationActionBox() in MainGUI class)

Pre-conditions:

User clicks the Export button in one of the sections of the program.

Post-conditions:

A file is generated with the plain-text contents of a section of the program.

Description:

A JFileChooser dialog is invoked. The user selects a file to which to export the information, or specifies a new one.

The program loops through the class hierarchy of the tree and writes out the contents of each tree into the specified file separated by tabs (allowing for better orientation in the export text) and saved.

The resulting file stores all the information from a given section of the program in plain-text form. This allows for better accessibility of the data stored in the program, which can be easily copy-pasted into a word processor and formatted at will.

Assigning Student Codes

(in assignStudentCodes() in Database class)

Pre-conditions:

The Database is constructed and there is at least one student recorded in it.

The studentCodesAssignCodesButton was clicked by the user.

Post-conditions:

All current students in the database are assigned a unique student code according to their school and alphabetical order in their school

Description:

The algorithm loops through the tree of schools in alphabetical order and assigns each a letter of the alphabet.

For each school, the algorithm loops through the tree of students. To each of them, it assigns a number from 1. (Important: the numbering is not restarted with a new school, so that each number is unique.)

The two parts of the code are combined into a single string ("C34", for example), which is then stored by the Student object as its code.

Generating Entities and Judges from the Database

(in generateEntities() in Qualification class)

Pre-conditions:

The database was approved.

The user clicks on the approveParticipantsIgnoreButton.

Post-conditions:

There is a tree of judges.

There are five trees of entities – separate for entities for every event.

Description:

Loop through schools in the schoolTree of the database. For each school, loop through its teacherTree.

Make new judge from the current teacher and add it to the qualification's judgeTree.

For original oratory, loop through the school tree. For each school, loop through its student tree.

If current student's event at index 0 (corresponds to OO) is true, make a new entity of the student and add it to the ooEntityTree.

Do the same process for oral interpretation (current student's events[1] is true, and addin is done to the oiEntityTree), and for impromptu speaking (current student's events[2] is true, and adding is done to the isEntityTree).

For duet acting, loop through the school tree. For each school, loop through its daPairTree. For each pair there, construct a corresponding Entity and add it to the daEntityTree.

Do the same for debate (debatePairTree, debateEntityTree).

Allocate Entities and Judges

*(in initializeEvents() in Qualification class)
(uses initializeRounds() in QualificationEvent class,
execute() in QualificationEvent.AllocationWorker class,
allocate() in QualificationEvent class)*

Pre-conditions:

The participantsApproveToggleButton was clicked and the subsequent approveParticipants warning dialog's ignoreButton was clicked.

A qualification was constructed – entities and judges trees are already set up.

Post-conditions:

Entities and judges participating in different events are distributed into rooms in all rounds they are participating in.

Description:

Events are processed one by one. For each one, if the number of entities is deemed satisfactory, the initializeRound() method of the current event is called. That method sets up an AllocationWorker to perform the allocation proper in the background, while updating the AllocationInfoDialog, which is fitted with a progress bar. Then, the AllocationWorker is set to execute.

The AllocationWorker calls the recursive allocate() method of the QualificationEvent. Because it is the very first call, a round is set to be created.

This iteration of the recursive method creates the round and prepares its roomArray. Then the maximum number of combinations possible is determined and saved as an attribute of the QualificationEvent. Then the allocate() method is called again, now to allocate a judge.

This iteration of the recursive method first updates the sets up a do-while loop to facilitate two tries to allocate the current judge. One try means updating the currentCombination value for the progress bar, selecting a random untried judge from the judgeTree, and trying to insert him to the current room. If this happens without a problem, the roomIndex is incremented (or changed back to 0) and a new iteration of the allocate() algorithm is called. This is either a judge (if still some judges need to be allocated – as determined by the difference between judgesPerRoom *

roomArrayLengths and numAllocatedJudges) or an entity (if all judges have been allocated by this judge).

If the allocation fails for a given judge, another judge is tried (the second try). If the allocation also fails for this one, a ImpossibleToAllocateException is thrown and this iteration of the algorithm halts. That means that the previous iteration was not a correct one. So now, the previous iteration tries to allocate with another judge. This back-and-forth dynamics of allocation and de-allocation is the heart of the recursive algorithm.

If even the second try for the very first judge fails, it means that no solution of the situation could be found. (This does not mean that a solution does not exist, however. Due to the exponentially increasing complexity of the calculations of the complete simulation, the number of tries has been capped to two. Therefore, it is possible – albeit unlikely – that a solution exists even if the algorithm was unable to find one. Then it is worthwhile to try to allocate for a second time.)

If an entity is allocated in a given iteration, a random untried entity is selected from the entityTree and tried tried to be applied to the room. Now, there are more criteria than there were for the judges: not only must not the entity be in one room with a judge or entity from its school, it also cannot be with a judge the entity has already encountered, in any round. If the allocation of the entity succeeds, another call to the allocate() algorithm is made to allocate entity (if there are still entities left in the entityTree) or to allocate a new round (if there are no more entities to allocate).

If the allocation of the entity fails twice, a new ImpossibleToAllocateException is thrown and this iteration halts. That is a signal to the previous iteration that the allocation was not successful. Another entity is tried, then, or that iteration throws an ImpossibleToAllocateException. This is possible also for the first entity. It might well happen that the organization of judges that has been the result of a random selection will not be ideal on the first try. In that case, the first entity's ImpossibleToAllocateException is caught by the last allocated judge and processed accordingly.

It is also possible that a round allocates flawlessly, but the following round cannot be allocated and its first judge throws an ImpossibleToAllocateException. In that case, the exception halts the second-round-setting iteration as well, before being processed by the last entity's iteration. This ensures another organization is tried for the round so that the following round can be successfully allocated.

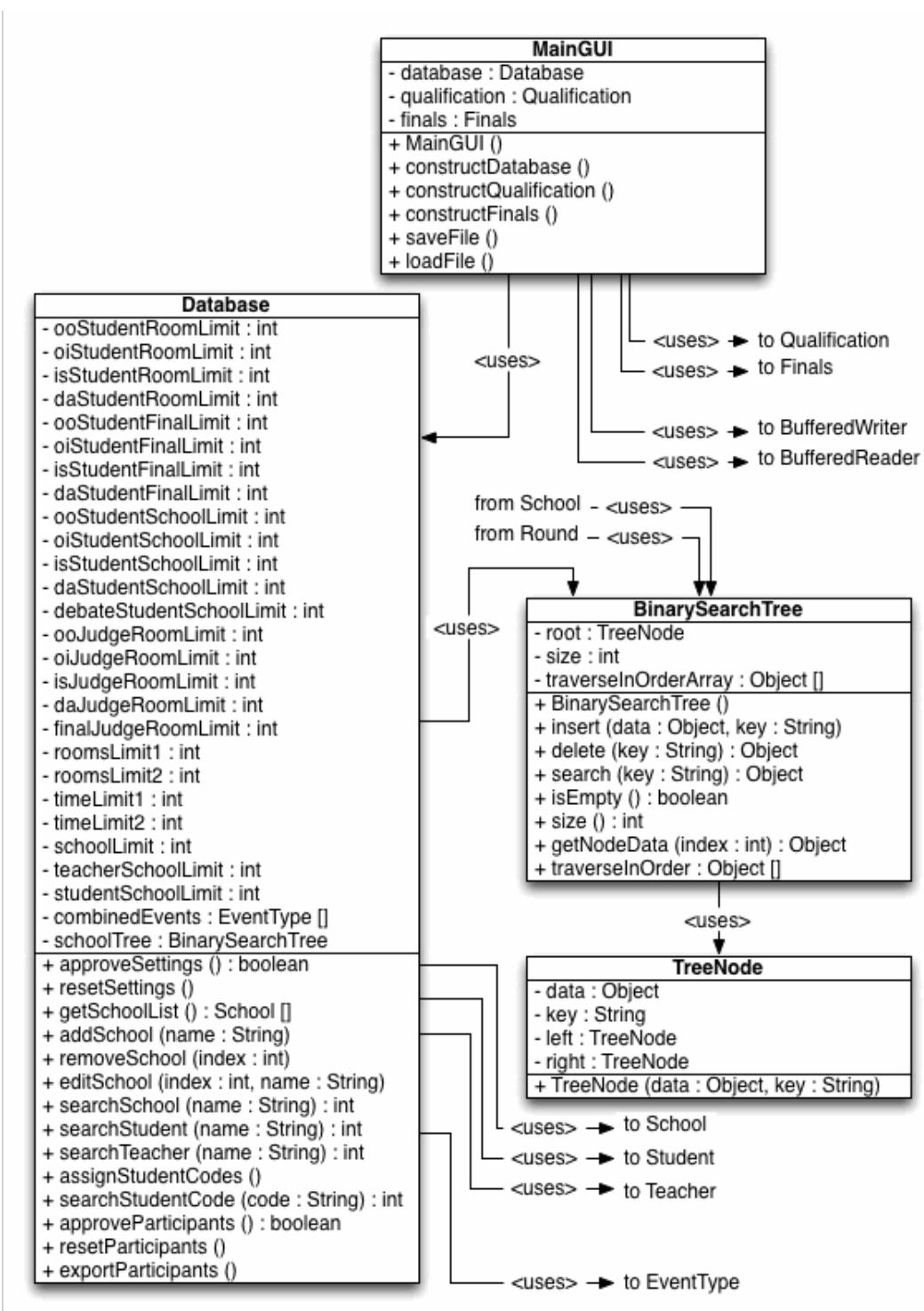
Finally, if, after all of this process, a round is set to be allocated, but the roundIndex is higher than the event's number of rounds, it means that the allocation was successful and the iteration halts. This halts all the preceding iterations as well, breaking the loops, and the event's participants are set for the rounds.

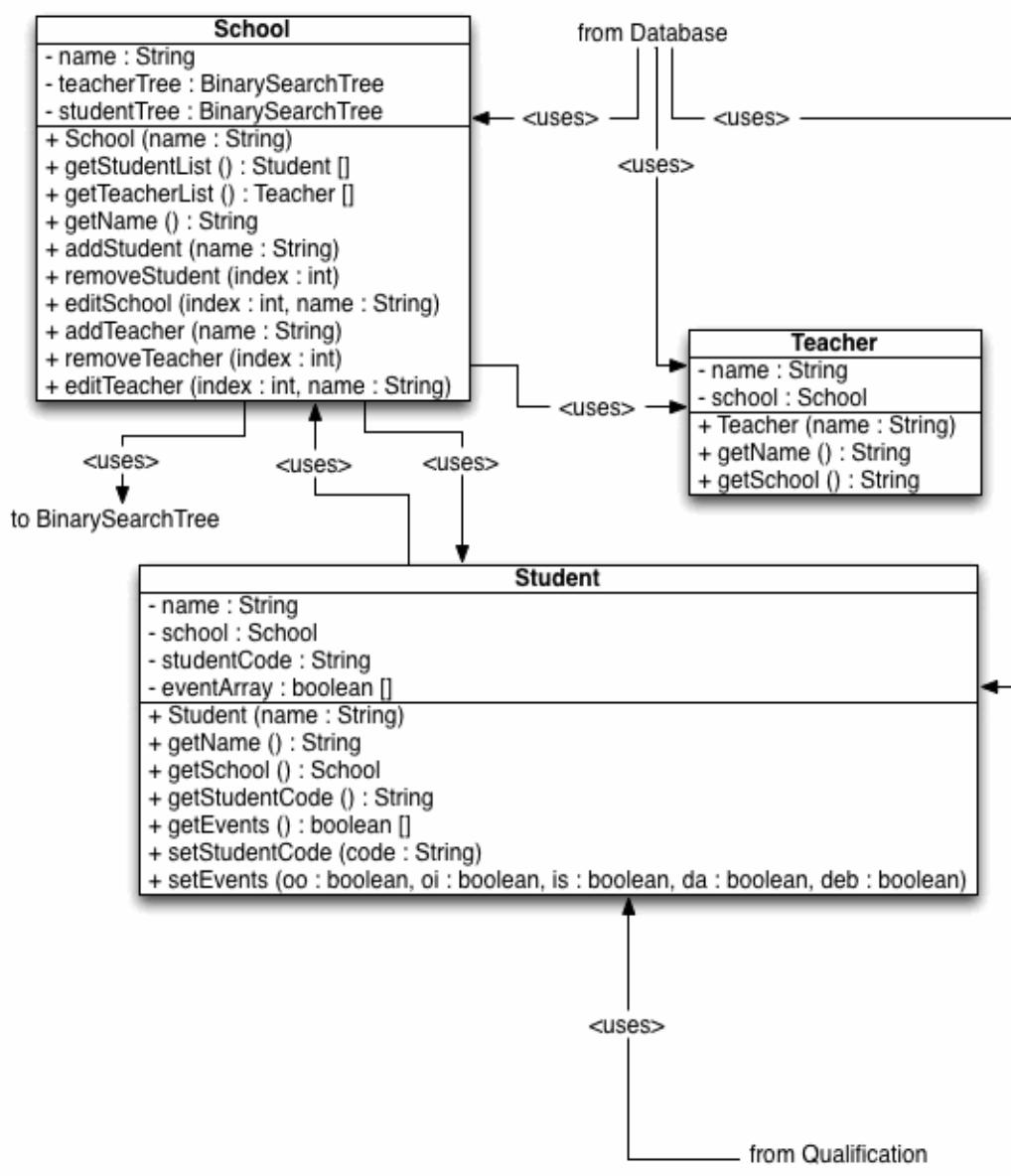
On the other hand, there is one other possibility of the allocate() method halting unexpectedly. That is when it is found during the allocation that there is not enough judges. That is an error unsolvable by the algorithm and therefore it has to stop. This does not come without previous warning, though – when the user clicks the approveParticipantsToggleButton, he is warned of the possibility that there will not be enough judges.

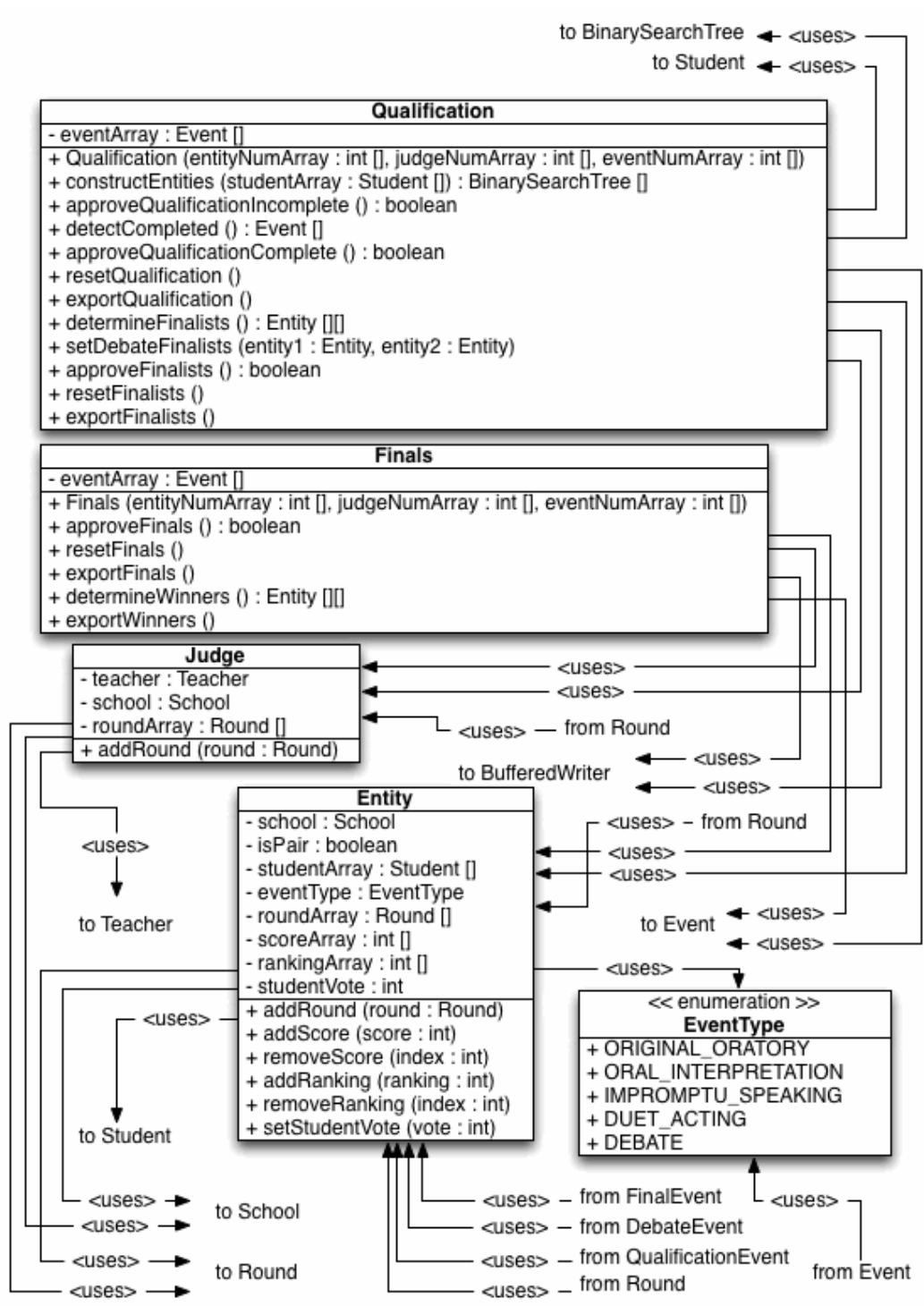
However, if the allocate() method finishes successfully, the AllocationWorker hides the progress dialog, and this event's initializeRounds() method finishes successfully, as well.

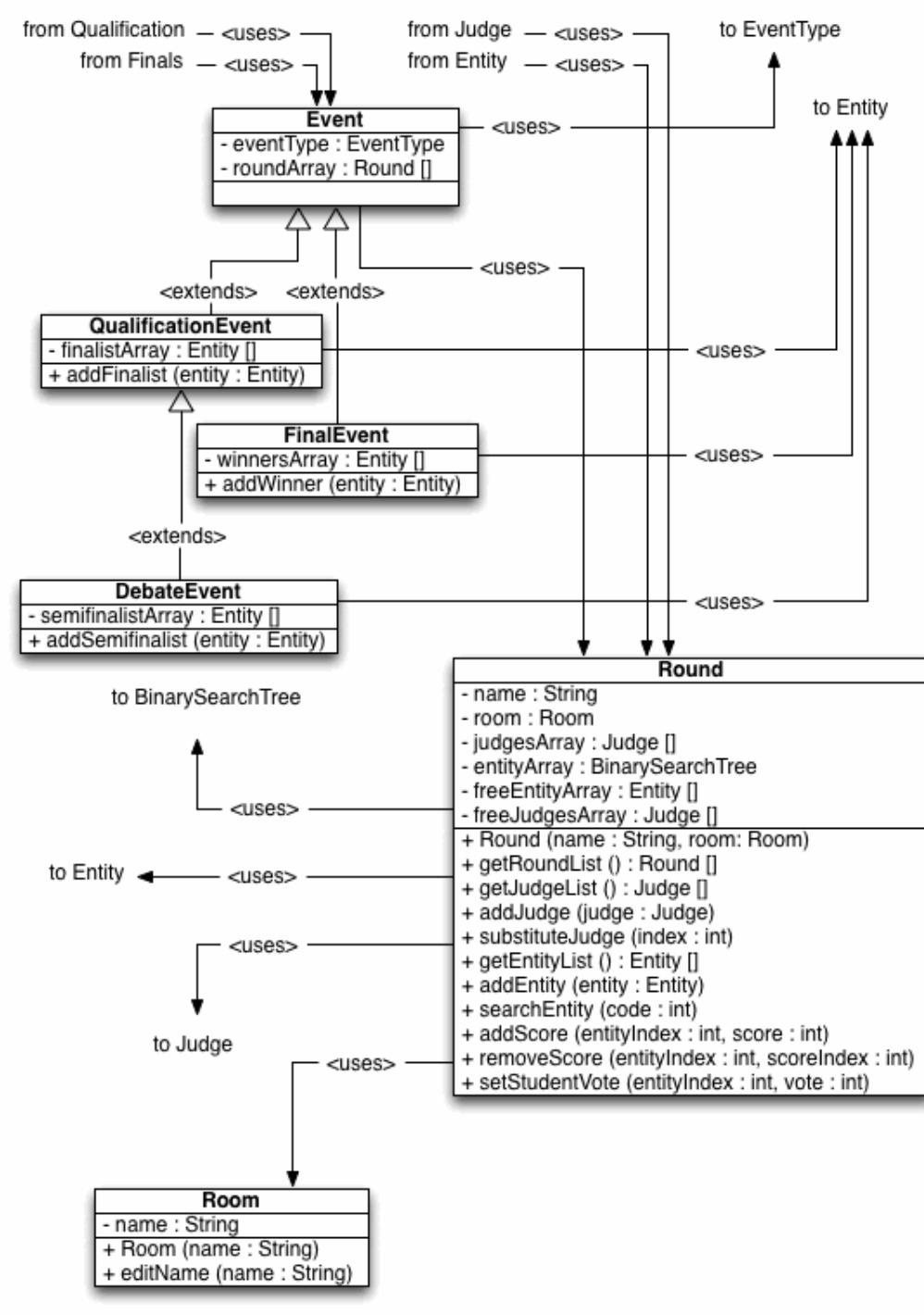
Then, the initializeEvents() method can progress to try to initialize the following event in the eventArray().

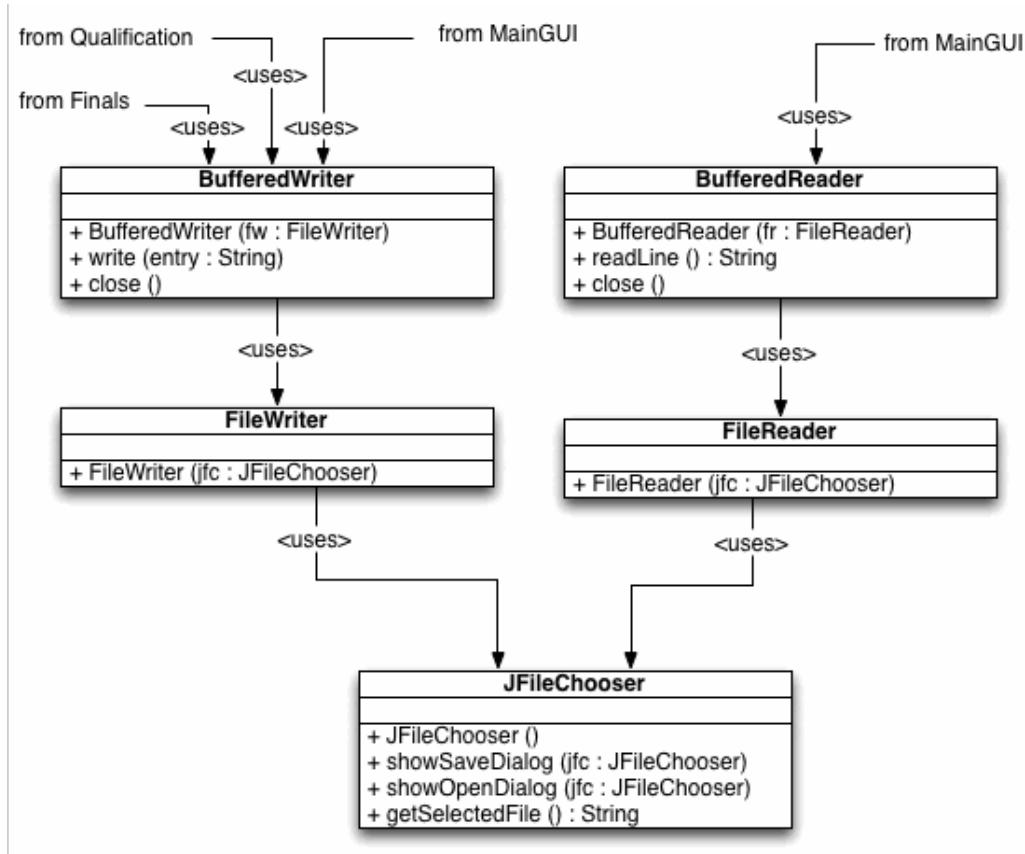
**Section B3:
Modular Organization**

Class Diagram:









The stencil application OmniGraffle has been used to create these diagrams.

Discussion of Classes:

MainGUI:

This class has its own right to existence, as it is initiating all principal processes of the program, as well as constructing its graphical user interface. It serves as an overarching “shell” to the program, the Database, Qualification, and Finals classes were separated from it for clarity separation of unrelated concepts. It is also responsible for loading files and all saving (through the **BufferedWriter** and **BufferedReader** classes).

Database:

This class takes care of the database section of the program (in the GUI demonstrated in the Settings and Participants tabs). It stores the values of the settings and a binary search tree of schools in the tournament. It is responsible for all actions concerning schools and “global” actions concerning teachers and students (searching; and also assigning student

codes). Database also exports the participants (through a BufferedWriter).

BinarySearchTree:

This class maintains binary search trees, important data structure used for storing database entries (schools, and students and teachers in those schools). It is used at multiple occasions throughout the program and therefore should be made separate.

TreeNode:

An auxiliary class to BinarySearchTree, a TreeNode stores a specific data entry in the binary search tree structure. It can be contained as a subclass to the BinarySearchTree.

School:

Since several students and teachers come from the same school, and there are several schools in the tournament, the School class stores the respective trees of students and teachers from a given school and is responsible for adding, removing, and editing of data entries in these trees. All of the classes for the Database structure were made so that a logical hierarchy of objects is created.

Teacher:

The teacher class serves to store relevant information about a given teacher – his or her school and name. It is analogous to the Student class, but there is not that much relevant information to consider.

Student:

The student class stores relevant information about a given student. This includes his or her name, school, student code, and the events he or she participates in (stored as a five-member array; one member for each event). All of this information will be important for the construction of Entities in the Qualification phase.

Qualification:

An important class responsible for the distribution of participants and judges into rounds, overarching the processes of the Qualification and Finalists tabs of the graphical user interface. It constructs the appropriate number of Rounds and populates them with Judges and Entities (constructed from the trees of Teachers and Students,

respectively). It also exports the Qualification and Finalists information through BufferedWriter, and checks the correctness of the data when the Approve buttons are clicked. All of these actions could have been part of the Main GUI itself, but were separated for a more organized code.

Finals:

A class, analogous to Qualification, responsible for the Finals and Winners tabs of the GUI. It distributes finalist Entities into their respective finals and makes sure that the allocation of Judges for these finals complies with the NESDA rules (no judge was judging the finalists before, no judge from a school that is in the finals...). It was created for a reason similar to the Qualification class – to separate attributes and methods relevant to finals from other aspects of the program.

Judge:

A class serving to store information about Judges. These are different logical constructs than Teachers (although in reality, of course, these refer to the same people) because a judge stores the information about the Rounds it was a part of, in an analogous way to Entities. As Entities, Judges are dynamic parts of the Qualification and Finals section, whereas Teachers are static, entered by the user to the Database.

Entity:

An Entity is more distinct to Student than Judge is to Teacher. This is due to the fact that one Student can be a part of multiple Entities (for different events he takes a part in), and one Entity does not always correspond to one Student (because there are pair events, in which two students act as one unit). The Entity construct, therefore, takes into account the complicating influence of pair events in the conversion between the Database and Qualification/Finals sections of the program. An entity stores an array of Student(s), which are a part of it; and an array of rounds of a specific EventType it is participating in. It also stores a corresponding array of scores gathered in these rounds, as well as the rankings and student votes the entity gathered in finals (if

applicable). It can be rightfully argued that the Entity class is the most important of all classes, as it is the integral part of the system of participant allocation, the most important function of the program.

EventType:

Technically, EventType is not a class, but an enumeration of the five different events at NESDA tournaments. The purpose of the enumeration is to avoid the use of code numbers for the different events and therefore add clarity to the resulting program.

Event:

The Rounds for each event (distinguished by the stored identifying EventType) are stored in the appropriate Event class (or, more specifically, the specific extensions to it), since there are more Rounds to an Event – making an Event a logical overarching construct to a Round.

QualificationEvent:

An extension to the Event class, serving to accommodate the finalist determination system with an array of finalists for that event.

DebateEvent:

An extension to the QualificationEvent class, adding another array for the storage of semifinalists in debate. This is to accommodate for the atypical structure of debate among other qualification events, which the extra selective semifinal round.

FinalEvent:

Another extension to the Event class, created for the specific purposes of final events (contains a special winners array).

Round:

One tournament session of a given Event, taking place in a multitude of Rooms. Contains an array of Entities and Judges from which the allocation system chooses which ones to allocate to which room so that the NESDA allocation rules are honored. It also takes care of the substitution of judges should it be necessary.

Room:

An instance of a specific round of a given event. It contains trees of entities and judges in a given room. Rooms are important class on their

own because of their role as the most elemental logical construction of the Qualification section of the program.

BufferedWriter (system class):

A system class necessary for saving data from the program as well as exporting given tabs as plain text files for easy access to information.

FileWriter (system class):

An auxiliary system class to BufferedWriter, it is used for saving and exporting data.

BufferedReader (system class):

A system class necessary for loading data for the program via the load option in the Settings tab.

FileReader (system class):

A system class helping used by the BufferedReader to load data for the program.

JFileChooser (system class):

A system class storing methods for presenting user with open and save dialogs necessary for the loading and saving procedures.

Class Interaction:

The classes were devised in such a way as to allow as much as possible for the separation of roles of the different classes. Extensions were used where appropriate (the Event family of classes), and a complex structure of class usages emerged amongst these classes in the process of creating the modular organization diagram. The program can be separated into two main areas, overarched by the MainGUI class: the database section, orchestrated by the Database class; and the qualification/finals section, characterized by the automated allocation systems.

Two interrelated hierarchies therefore come into existence to accommodate for the difference between these two areas. In the database, students (and teachers) are elemental units (with different events assigned), belonging to different schools. On the other hand, in the qualification and finals, entities (created from students and pairs of students of a specific events) and judges (created from teachers) are assigned to rooms, which are parts of

rounds, which are in turn parts of events. These were created to reflect the real structure of the tournament.

There are also classes, which are used in both of these areas and therefore can be seen as standing out from the dual structure of the program. These include the BinarySearchTree class (along with the TreeNode class), and the system classes for reading and writing files – including the BufferedReader/BufferedWriter classes, the FileReader/FileWriter classes, and the JFileChooser class.

The modular organization of the program was devised to reflect as closely and as logically as possible the hierarchies of logical objects used in the organization of the NESDA tournaments. For that purpose, all of the classes presented in the class diagram were devised and are expected to prove useful in the programming of the tournament manager.

Section C2: Error Handling

Java Checked Exceptions:

BufferedReader's IOException

(MainGUI class, page 162)

```
private void readFile(File readFile) throws FileIOException, IOException {
    BufferedReader br = new BufferedReader(new FileReader(readFile));

    String currentLine = "";
    currentLine = br.readLine();
    if (currentLine == null || !currentLine.equals("{NESDA Tournament Manager Save
File}")) {
        throw new FileIOException();
    }
    //...
}
```

(MainGUI class, page 162)

```
private void loadFile() throws UserIOException, FileIOException, IOException {
    JFileChooser jfc = new JFileChooser();

    int loadDialogReturn = jfc.showOpenDialog(this);

    if (loadDialogReturn == JFileChooser.APPROVE_OPTION) {
        loadFile = jfc.getSelectedFile();
        readFile(loadFile);
    } else if (loadDialogReturn == JFileChooser.CANCEL_OPTION) {
        throw new UserIOException();
    } else {
        throw new IOException();
    }
}
```

(MainGUI class, page 223)

```
private void settingsLoadButtonMouseReleased(java.awt.event.MouseEvent evt) {
    try {
        this.loadFile();
        this.synchronizeTFSSettings(database);
    } catch (UserIOException ex) {
        // this is okay - window closed
    } catch (FileNotFoundException ex) {
        exceptionDialogTextArea.setText("The file you tried to open is not
        saved in a format compatible with this program. Make sure you are
        trying to load in a valid file and try to open it once more.");

        exceptionDialog.setVisible(true);
    } catch (IOException ex) {
        exceptionDialogTextArea.setText("An input/output exception has occurred
        during the loading of the file. Try to load the file once more.");

        exceptionDialog.setVisible(true);
    }
}
```

BufferedReader's required IOException catching is implemented as follows:

In the case of an error during the data input, readFile() throws IOException, which passes through loadFile(), and is caught by MainGUI's settingsLoadButtonMouseReleased(). The exception is caught, and a dialog is shown, informing the user of the problem and asking him to try to load the file once more.

BufferedWriter's IOException*(MainGUI class, page 174)*

```

private void saveFile() throws UserIOException, IOException {
    JFileChooser jfc = new JFileChooser();

    int saveDialogReturn = jfc.showSaveDialog(this);

    if (saveDialogReturn == JFileChooser.APPROVE_OPTION) {
        BufferedWriter bw = new BufferedWriter(new
FileWriter(jfc.getSelectedFile()));
        //...
    }
    //...
}

```

(MainGUI class, page 224)

```

private void settingsSaveToggleButtonMouseReleased(java.awt.event.MouseEvent evt)
{
    if (settingsSaveToggleButton.isSelected()) {
        try {
            this.saveFile();

            settingsApproveToggleButton.setEnabled(false);
        } catch (UserIOException ex) {
            settingsSaveToggleButton.setSelected(false);
        } catch (IOException ex) {
            settingsSaveToggleButton.setSelected(false);

            exceptionDialogTextArea.setText("An input/output exception has occurred
during the saving of the file. Try to save the file once more.");

            exceptionDialog.setVisible(true);
        }
    } else {
        settingsApproveToggleButton.setEnabled(true);
    }
}

```

BufferedWriter's required IOException catching is implemented as follows: In case of an exception while writing the output stream, the saveFile() method throws anIOException, which is then caught by MainGUI's settingsSaveToggleButtonMouseReleased() method. An exception dialog is shown, informing the user of the exception and asking him to try once more.

This type of error handling is also used in:

- exportParticipantsActionBox() (MainGUI class, page 189)
- exportQualificationActionBox() (MainGUI class, page 190)

Common Runtime Error Handling:

ArrayIndexOutOfBoundsException

(MainGUI class, page 256)

```

private void
qualificationEventsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        eventsIndex = evt.getLastIndex();

        roundsIndex = -1;
        roomsIndex = -1;
        judgesIndex = -1;
        qStudentCodesIndex = -1;

        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);

            qualificationRoundsList.setEnabled(true);

            this.updateRoundsListModel(currentEvent);

        } catch (ArrayIndexOutOfBoundsException ex) {
            qualificationRoundsList.setEnabled(false);
            //qualificationRoundList.setSelectedIndex(-1);
            qualificationRoundsList.setModel(noSelectionModel);

        } catch (NullPointerException ex) {
            //...
        }
        //...
    }
}

```

(MainGUI class, page 257)

```

private void
qualificationRoundsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {

        roundsIndex = evt.getLastIndex();

        roomsIndex = -1;
        judgesIndex = -1;
        qStudentCodesIndex = -1;

        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);
            Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);

            qualificationRoomsList.setEnabled(true);

            qualificationRoundsTextField.setEnabled(true);
            qualificationRoundsEditNameButton.setEnabled(true);

            this.updateRoomsListModel(currentRound);

        } catch (ArrayIndexOutOfBoundsException ex) {
            qualificationRoomsTextField.setText("");
            qualificationRoomsTextField.setEnabled(false);
            qualificationRoundsEditNameButton.setEnabled(false);

            qualificationRoomsList.setEnabled(false);
            //qualificationRoomList.setSelectedIndex(-1);
            qualificationRoomsList.setModel(noSelectionModel);

        } catch (NullPointerException ex) {
            //...
        }
        //...
    }
}

```

(MainGUI class, page 258)

```

private void
qualificationRoomsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        roomsIndex = evt.getLastIndex();

        judgesIndex = -1;
        qStudentCodesIndex = -1;

        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);
            Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);
            Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);

            qualificationJudgesList.setEnabled(true);
            qualificationStudentCodesList.setEnabled(true);

            qualificationRoomsTextField.setEnabled(true);
            qualificationRoomsEditNameButton.setEnabled(true);

            qualificationStudentCodesTextField.setEnabled(true);
            qualificationStudentCodesSearchButton.setEnabled(true);

            this.updateJudgesListModel(currentRoom);
            this.updateQStudentCodesListModel(currentRoom);

        } catch (ArrayIndexOutOfBoundsException ex) {
            //...
        }

        } catch (NullPointerException ex) {
            qualificationRoomsTextField.setText("");
            qualificationRoomsTextField.setEnabled(false);
            qualificationRoomsEditNameButton.setEnabled(false);

            qualificationJudgesList.setEnabled(false);
            //qualificationJudgesList.setSelectedIndex(-1);
            qualificationJudgesList.setModel(emptyModel);

            qualificationStudentCodesList.setEnabled(false);
            //qualificationStudentCodesList.setSelectedIndex(-1);
            qualificationStudentCodesList.setModel(emptyModel);
        }
        //...
    }
}

```

In the above examples, `ArrayIndexOutOfBoundsException` is being thrown by the `getEventArrayElement()`, `getRoundArrayElement()`, and `getRoomArrayElement()` methods when the parameter of -1 is passed to them. The reason it is important to check for that is the fact that when a list is clicked, two `listSelectionEvents` are thrown: when the mouse is pressed (index returned is -1), and when the mouse is released (index returned is the index of the selection). The value of -1 therefore has to be caught without the program registering problems, so that the correct subsequent value can pass through.

NullPointerException*(MainGUI class, page 256)*

```

private void
qualificationEventsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        eventsIndex = evt.getLastIndex();

        roundsIndex = -1;
        roomsIndex = -1;
        judgesIndex = -1;
        qStudentCodesIndex = -1;

        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);

            qualificationRoundsList.setEnabled(true);

            this.updateRoundsListModel(currentEvent);

        } catch (ArrayIndexOutOfBoundsException ex) {
            //...
        }

        try {
            qualificationRoundsList.setEnabled(false);
            //qualificationRoundList.setSelectedIndex(-1);
            qualificationRoundsList.setModel(emptyModel);
        }
        //...
    }
}

```

(MainGUI class, page 257)

```

private void
qualificationRoundsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {

        roundsIndex = evt.getLastIndex();

        roomsIndex = -1;
        judgesIndex = -1;
        qStudentCodesIndex = -1;

        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);
            Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);

            qualificationRoomsList.setEnabled(true);

            qualificationRoundsTextField.setEnabled(true);
            qualificationRoundsEditNameButton.setEnabled(true);

            this.updateRoomsListModel(currentRound);

        } catch (ArrayIndexOutOfBoundsException ex) {
            //...
        }

        try {
            qualificationRoomsTextField.setText("");
            qualificationRoomsTextField.setEnabled(false);
            qualificationRoundsEditNameButton.setEnabled(false);

            qualificationRoomsList.setEnabled(false);
            //qualificationRoomList.setSelectedIndex(-1);
            qualificationRoomsList.setModel(emptyModel);
        }
        //...
    }
}

```

(MainGUI class, page 258)

```

private void
qualificationRoomsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        roomsIndex = evt.getLastIndex();

        judgesIndex = -1;
        qStudentCodesIndex = -1;

        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);
            Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);
            Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);

            qualificationJudgesList.setEnabled(true);
            qualificationStudentCodesList.setEnabled(true);

            qualificationRoomsTextField.setEnabled(true);
            qualificationRoomsEditNameButton.setEnabled(true);

            qualificationStudentCodesTextField.setEnabled(true);
            qualificationStudentCodesSearchButton.setEnabled(true);

            this.updateJudgesListModel(currentRoom);
            this.updateQStudentCodesListModel(currentRoom);

        } catch (ArrayIndexOutOfBoundsException ex) {
            //...
        }

        } catch (NullPointerException ex) {
            qualificationRoomsTextField.setText("");
            qualificationRoomsTextField.setEnabled(false);
            qualificationRoomsEditNameButton.setEnabled(false);

            qualificationJudgesList.setEnabled(false);
            //qualificationJudgesList.setSelectedIndex(-1);
            qualificationJudgesList.setModel(emptyModel);

            qualificationStudentCodesList.setEnabled(false);
            //qualificationStudentCodesList.setSelectedIndex(-1);
            qualificationStudentCodesList.setModel(emptyModel);
        }
        //...
    }
}

```

In the above examples, `NullPointerException` is being thrown by the `updateRoundListModel()`, `updateRoomsListModel()`, `updateJudgesListModel()`, and `updateQStudentCodesListModel()` methods when a null parameter is passed to them. The reason it is important to check for that is the fact that when a list is clicked, two `listSelectionEvents` are thrown: when the mouse is pressed (index returned is -1), and when the mouse is released (index returned is the index of the selection). The value of -1 is what causes problems in the update methods and these problems need to be caught. However, the exception does not need to be reported to the user because it is not fatal and right after, the correct value will come.

NumberFormatException*(MainGUI class, page 162)*

```
private void readFile(File readFile) throws FileIOException, IOException {
    BufferedReader br = new BufferedReader(new FileReader(readFile));
    String currentLine = "";
    //...
    currentLine = br.readLine();
    try {
        approvedSectionsIndex = Integer.parseInt(currentLine);
    } catch (NumberFormatException ex) {
        throw new FileIOException();
    }
    //...
}
```

As the excerpt shows, the *NumberFormatException* is thrown if the parsing of information fails because of the fact that the string to be parsed is not a number. The exception is then relayed as a *FileIOException*, which is then presented to the user in a GUI window (see below for more information). This exception is widely used by the *readFile()* method and at a small scale by other methods.

This type of error handling is also used in:

- *getTextFieldValue()* (MainGUI class, page 155)

Error Handling with If-Else Statements

(MainGUI class, page 228)

```

private void
participantsSchoolsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    schoolsIndex = participantsSchoolsList.getSelectedIndex();

    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

    if (schoolsIndex != -1) { // if schools index was set to be something (not -1)
        // SCHOOLS:
        // enable the selected-school-specific buttons (remove, edit)
        participantsSchoolsRemoveButton.setEnabled(true);
        participantsSchoolsEditButton.setEnabled(true);

        // TEACHERS:
        // enable the addition of teachers to the school
        participantsTeachersTextField.setEnabled(true);
        participantsTeachersAddButton.setEnabled(true);

        if (currentSchool.getTeacherTreeSize() > 0) { // if there are some
            teachers in the selected school
            // update and enable the teachers list
            participantsTeachersList.setEnabled(true);
            this.updateTeachersListModel(currentSchool);
        } else { // if teacherTree is empty
            // disable teachers list and set the teacher model to be the empty
            model
            participantsTeachersList.setEnabled(false);
            participantsTeachersRemoveButton.setEnabled(false);
            participantsTeachersEditButton.setEnabled(false);

            participantsTeachersList.setModel(emptyModel);
        }

        // STUDENTS (+ ASSIGN EVENTS):
        // enable adding of students
        participantsStudentsTextField.setEnabled(true);
        participantsStudentsAddButton.setEnabled(true);

        if (currentSchool.getStudentTreeSize() > 0) { // if there are some
            students in the database
            // enable studentsList
            participantsStudentsList.setEnabled(true);

            // show students and studentCodes for current school
            this.updateStudentsListModel(currentSchool);
            this.updateStudentCodesListModel(currentSchool);

            // if there is a potential for assigning pairs (minimum of 2 DA or
            debate students and codes assigned)
            if (((currentSchool.getUnpairedDAStudentTreeSize() / 2) > 0) ||
                ((currentSchool.getUnpairedDebateStudentTreeSize() / 2) > 0)) && studentCodes) {
                // enable assign pairs events list
                participantsAssignPairsEventsList.setSelectedIndex(-1);
                participantsAssignPairsEventsList.setModel(eventsListModel);
                participantsAssignPairsEventsList.setEnabled(true);
            } else if ((currentSchool.getDAPairTreeSize() > 0) ||
                (currentSchool.getDebatePairTreeSize() > 0)) { // if some pairs were assigned
                // enable assign pairs events list
                participantsAssignPairsEventsList.setSelectedIndex(-1);
                participantsAssignPairsEventsList.setModel(eventsListModel);
                participantsAssignPairsEventsList.setEnabled(true);
            } else { // if there is no potential for pairing or removing pairs
                // disable assign pairs events list
                participantsAssignPairsEventsList.setSelectedIndex(-1);
                participantsAssignPairsEventsList.setModel(eventsListModel);
                participantsAssignPairsEventsList.setEnabled(false);
            }
        }

        // disable the other assign pairs lists and set their models to be
        blank
        participantsAssignPairsLeftList.setSelectedIndex(-1);
        participantsAssignPairsLeftList.setModel(blankModel);
        participantsAssignPairsLeftList.setEnabled(false);
    }
}

```

```

participantsAssignPairsRightList.setSelectedIndex(-1);
participantsAssignPairsRightList.setModel(blankModel);
participantsAssignPairsRightList.setEnabled(false);

participantsAssignPairsPairsList.setSelectedIndex(-1);
participantsAssignPairsPairsList.setModel(blankModel);
participantsAssignPairsPairsList.setEnabled(false);

// disable the pair and remove pair buttons
participantsAssignPairsPairButton.setEnabled(false);
participantsAssignPairsRemovePairButton.setEnabled(false);
} else { // if studentTree is empty
    participantsStudentsList.setEnabled(false);
    participantsStudentsRemoveButton.setEnabled(false);
    participantsStudentsEditButton.setEnabled(false);

    this.disableAssignEventsActionBox();

    participantsStudentsList.setModel(emptyModel); // needed - #20
    participantsStudentCodesList.setModel(emptyModel);

    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(false);

    participantsAssignPairsLeftList.setSelectedIndex(-1);
    participantsAssignPairsLeftList.setModel(blankModel);
    participantsAssignPairsLeftList.setEnabled(false);

    participantsAssignPairsRightList.setSelectedIndex(-1);
    participantsAssignPairsRightList.setModel(blankModel);
    participantsAssignPairsRightList.setEnabled(false);

    participantsAssignPairsPairsList.setSelectedIndex(-1);
    participantsAssignPairsPairsList.setModel(blankModel);
    participantsAssignPairsPairsList.setEnabled(false);

    participantsAssignPairsPairButton.setEnabled(false);
    participantsAssignPairsRemovePairButton.setEnabled(false);
}
} else { // if schoolsIndex == -1
    // SCHOOLS:
    participantsSchoolsRemoveButton.setEnabled(false);
    participantsSchoolsEditButton.setEnabled(false);

    // TEACHERS:
    participantsTeachersList.setModel(noSelectionModel);
    participantsTeachersList.setEnabled(false);

    if (teachersNumber != 0) {
        participantsTeachersTextField.setEnabled(true);
        participantsTeachersSearchButton.setEnabled(true);
    } else {
        participantsTeachersTextField.setEnabled(false);
        participantsTeachersSearchButton.setEnabled(false);
    }

    participantsTeachersAddButton.setEnabled(false);
    participantsTeachersRemoveButton.setEnabled(false);
    participantsTeachersEditButton.setEnabled(false);

    // STUDENTS (+ ASSIGN EVENTS):
    this.disableAssignEventsActionBox();

    participantsStudentsList.setModel(noSelectionModel);
    participantsStudentsList.setEnabled(false);

    participantsStudentCodesList.setModel(noSelectionModel);
    participantsStudentCodesList.setEnabled(false);

    if (studentsNumber != 0) {
        participantsStudentsTextField.setEnabled(true);
        participantsStudentsSearchButton.setEnabled(true);
    } else {
        participantsStudentsTextField.setEnabled(false);
        participantsStudentsSearchButton.setEnabled(false);
    }

    participantsStudentsAddButton.setEnabled(false);
}

```

```

participantsStudentsRemoveButton.setEnabled(false);
participantsStudentsEditButton.setEnabled(false);

// ASSIGN PAIRS:
participantsAssignPairsEventsList.setSelectedIndex(-1);
participantsAssignPairsEventsList.setModel(eventsListModel);
participantsAssignPairsEventsList.setEnabled(false);

participantsAssignPairsLeftList.setSelectedIndex(-1);
participantsAssignPairsLeftList.setModel(blankModel);
participantsAssignPairsLeftList.setEnabled(false);

participantsAssignPairsRightList.setSelectedIndex(-1);
participantsAssignPairsRightList.setModel(blankModel);
participantsAssignPairsRightList.setEnabled(false);

participantsAssignPairsPairsList.setSelectedIndex(-1);
participantsAssignPairsPairsList.setModel(blankModel);
participantsAssignPairsPairsList.setEnabled(false);

participantsAssignPairsPairButton.setEnabled(false);
participantsAssignPairsRemovePairButton.setEnabled(false);
}
}

```

In the above example, a simple if-else block is used to handle list errors. When the program detests a selection index of -1 (meaning “no selection was made in the list”), it disables all the dependent fields. If the value is -1, the program also does not attempt to call the update methods and therefore other error handling is not necessary.

This type of error handling is also used in:

- participantsTeachersListValueChanged() (MainGUI class, page 232)
- participantsStudentsListValueChanged() (MainGUI class, page 237)
- participantsStudentCodesListValueChanged() (MainGUI class, page 241)
- participantsAssignPairsEventsListValueChanged() (MainGUI class, page 244)
- participantsAssignPairsLeftListValueChanged() (MainGUI class, page 245)
- participantsAssignPairsRightListValueChanged() (MainGUI class, page 245)
- participantsAssignPairsPairsListValueChanged() (MainGUI class, page 248)

Handling Erroneous Data Input from the User with Code:

UserIOException

(MainGUI class, page 189)

```

private void exportParticipantsActionBox() throws UserIOException, IOException {
    JFileChooser jfc = new JFileChooser();
    int saveDialogReturn = jfc.showSaveDialog(this);
    try {
        if (saveDialogReturn == JFileChooser.APPROVE_OPTION) {
            BufferedWriter bw = new BufferedWriter(new
FileWriter(jfc.getSelectedFile()));
            for (int i = 0; i < database.getSchoolTreeSize(); i++) {
                School currentSchool = (School) database.getSchoolTreeNodeData(i);
                bw.write(currentSchool.getName() + "\n");
                for (int j = 0; j < currentSchool.getTeacherTreeSize(); j++) {
                    Teacher currentTeacher = (Teacher)
currentSchool.getTeacherTreeNodeData(j);
                    bw.write("\t" + currentTeacher.getName() + "\n");
                }
                bw.write("\n");
                for (int j = 0; j < currentSchool.getStudentTreeSize(); j++) {
                    Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(j);
                    bw.write("\t" + currentStudent.getCode() + "\t" +
currentStudent.getName() + "\n");
                }
                bw.write("\n");
            }
            bw.close();
        } else if (saveDialogReturn == JFileChooser.CANCEL_OPTION) {
            throw new UserIOException();
        } else {
            throw new IOException();
        }
    } catch (UserIOException ex) {
        // do nothing
    } catch (IOException ex) {
        exceptionDialogTextArea.setText("An unexpected error occurred "
+ "during the accessing of the specified file. Data was "
+ "not exported. Please try again.");
        exceptionDialog.setVisible(true);
    }
}
}

```

The UserIOException is thrown when the user cancels the file chooser dialog. Without a file to which to export, it is impossible for the program to do the export and therefore, the action is stopped. No exception window is shown, though, because users are not expected to be shown a dialog telling them that they have cancelled the action (they clicked the button, they know that already).

This type of error handling is also used in:

- loadFile() (MainGUI class, page 162)
- saveFile() (MainGUI class, page 174)
- exportQualificationActionBox() (MainGUI class, page 190)

FileNotFoundException

(MainGUI class, page 162)

```

private void readFile(File readFile) throws FileNotFoundException, IOException {
    BufferedReader br = new BufferedReader(new FileReader(readFile));
    String currentLine = "";
    // s<editor-fold defaultstate="collapsed" desc="Introduction">
    currentLine = br.readLine();
    if (currentLine == null || !currentLine.equals("{NESDA Tournament Manager Save
File}")) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    if (br.readLine() == null) {
        throw new FileNotFoundException();
    }
    // </editor-fold>
    // <editor-fold defaultstate="collapsed" desc="$Attributes">
    currentLine = br.readLine();
    if (currentLine == null || !currentLine.equals("$Attributes")) {
        throw new FileNotFoundException();
    }
    currentLine = br.readLine();
    try {
        approvedSectionsIndex = Integer.parseInt(currentLine);
    } catch (NumberFormatException ex) {
        throw new FileNotFoundException();
    }
    currentLine = br.readLine();
    if (currentLine != null) {
        String noNameNumbersString = currentLine;
        StringTokenizer st = new StringTokenizer(noNameNumbersString, ",");
        String noNameSchoolNumberString = st.nextToken();
        currentNewNoNameSchoolNumber = Integer.parseInt(noNameSchoolNumberString);
        String noNameTeacherNumberString = st.nextToken();
        currentNewNoNameTeacherNumber =
        Integer.parseInt(noNameTeacherNumberString);
        String noNameStudentNumberString = st.nextToken();
        currentNewNoNameStudentNumber =
        Integer.parseInt(noNameStudentNumberString);
    } else {
        throw new FileNotFoundException();
    }
    currentLine = br.readLine();
    if (currentLine != null) {
        String numbersString = currentLine;
        StringTokenizer st = new StringTokenizer(numbersString, ",");
        String schoolsNumberString = st.nextToken();
        schoolsNumber = Integer.parseInt(schoolsNumberString);
        String teachersNumberString = st.nextToken();
        teachersNumber = Integer.parseInt(teachersNumberString);
        String studentsNumberString = st.nextToken();
        studentsNumber = Integer.parseInt(studentsNumberString);
    } else {
        throw new FileNotFoundException();
    }
    currentLine = br.readLine();
    if (currentLine != null) {
        String booleansString = currentLine;
    }
}

```

```

 StringTokenizer st = new StringTokenizer(booleanString, ",");
 String reassignmentTextString = st.nextToken();
 reassignmentText = Boolean.parseBoolean(reassignmentTextString);
 String studentCodesString = st.nextToken();
 studentCodes = Boolean.parseBoolean(studentCodesString);
 String studentChangesString = st.nextToken();
 studentChanges = Boolean.parseBoolean(studentChangesString);
} else {
    throw new FileIOException();
}
if (br.readLine() == null) {
    throw new FileIOException();
}
// </editor-fold>
//...
}

```

(MainGUI class, page 223)

```

private void settingsLoadButtonMouseReleased(java.awt.event.MouseEvent evt) {
    try { // try to load a file
        this.loadFile();

        // set the setting tab's settings values to be the same as those in
        database
        this.synchronizeTFSettings(database);

        // which tabs were saved in the file?
        if (approvedSectionsIndex == 0) { // only settings
            this.enableSettingsTabActionBar();
        } else if(approvedSectionsIndex == 1) { // settings and participants
            tabbedPane.setSelectedIndex(1);

            this.enableParticipantsTabActionBar();

            this.updateParticipantsProgressBar();
        } else if(approvedSectionsIndex == 2) { // settings, participants, and
        qualification
            tabbedPane.setSelectedIndex(2);

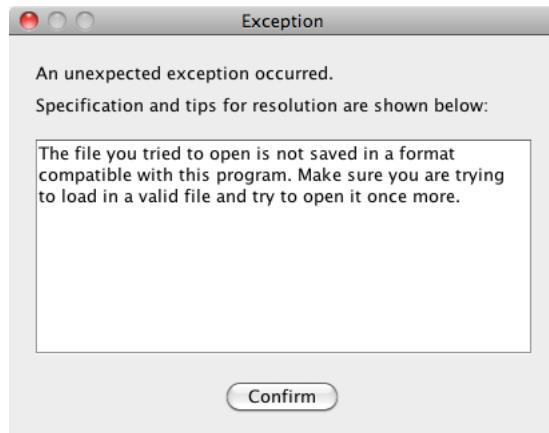
            this.enableQualificationTabActionBar();

            this.updateQualificationProgressBar(100);
        }
    } catch (UserIOException ex) { // if the user cancels the file chooser window
        //...
    } catch (FileNotFoundException ex) { // if the file is illegal
        // show the following exception dialog
        exceptionDialogTextArea.setText("The file you tried to open is not "
            + "saved in a format compatible with this program. Make "
            + "sure you are trying to load in a valid file and try "
            + "to open it once more.");
        exceptionDialog.setVisible(true);
    } catch (IOException ex) { // if there is a general IOException
        //...
    }
}

```

As can be seen in the excerpt of the `readFile()` method, a `FileIOException` is thrown when the information being read in is not what is expected. It is also thrown when the file abruptly ends at an unexpected point. This exception passes through the `loadFile()` method, and is caught by the `settingsLoadButtonMouseReleased()` method.

If a FileIOException is registered during the loading of the file, the following dialog is shown:



This exception is also thrown by the saveFile() method, when it is detected that an event to be saved is of undefined type. This is not allowed and should not happen, but if it does, the program is prepared.

This type of error handling is also used in:

- saveFile() (MainGUI class, page 174)
- settingsSaveToggleButtonMouseReleased() (MainGUI class, page 224)
- participantsSaveToggleButtonMouseReleased() (MainGUI class, page 253)
- qualificationSaveToggleButtonMouseReleased() (MainGUI class, page 260)

IllegalArgumentException*(BinarySearchTree class, page 266)*

```

    public void insert(Object data, String keyString) throws IllegalArgumentException {
        if (isEmpty()) root = new Node(data, keyString); // if the tree is empty, set
        the root to be a new node with the data provided
        else root = insertHelper(data, keyString, root); // else, call insert helper
        (throws IllegalArgumentException if keyString is used)

        // if there was no exception, increase the size of the tree and acknowledge
        the change in the tree
        size += 1;
        dataArrayChanged = true;
    }

```

(BinarySearchTree class, page 266)

```

private Node insertHelper(Object data, String keyString, Node currentNode) throws
IllegalArgumentException {
    if (keyString.compareTo(currentNode.keyString) < 0) { // if the key of the
    current node is higher than the one that will be added
        // go to the left
        if (currentNode.left == null) { // if the left node is currently null
            // construct a new node there
            currentNode.left = new Node(data, keyString);
            return currentNode;
        } else { // if there is something to the left
            // iterate for that node
            currentNode.left = insertHelper(data, keyString, currentNode.left);
            return currentNode;
        }
    } else if (keyString.compareTo(currentNode.keyString) > 0) { // if the key of
    the current node is lower than the one that will be added
        // go to the right
        if (currentNode.right == null) { // if the right node is currently null
            // construct a new node there
            currentNode.right = new Node(data, keyString);
            return currentNode;
        } else { // if there is something to the right
            // iterate for that node
            currentNode.right = insertHelper(data, keyString, currentNode.right);
            return currentNode;
        }
    } else { // if key of the current node is the same as the one that is being
    added
        throw new IllegalArgumentException("The same keyString is already used in
        the tree. They need unique values.");
    }
}

```

(School class, page 305)

```

public void insertDebatePair(Student student1, Student student2) throws
IllegalArgumentException {
    DebateStudentPair newStudentPair;
    try {
        newStudentPair = new DebateStudentPair(student1, student2);
    } catch (IllegalArgumentException ex) {
        throw new IllegalArgumentException();
    }

    if (!debatePairTree.contains(newStudentPair.getCode())) {
        debatePairTree.insert(newStudentPair, newStudentPair.getCode());

        Student[] pairStudentArray = newStudentPair.getStudentArray();

        Student leftStudent = pairStudentArray[0];
        Student currentStudent = (Student)
studentTree.search(leftStudent.getName());
        currentStudent.setDebateStudentPair(newStudentPair);

        Student rightStudent = pairStudentArray[1];
        currentStudent = (Student) studentTree.search(rightStudent.getName());
    }
}

```

```

        currentStudent.setDebateStudentPair(newStudentPair);
    }
    else throw new IllegalArgumentException();
}

(MainGUI class, page 225)

private void participantsSchoolsAddButtonMouseReleased(java.awt.event.MouseEvent evt)
{
    String newSchoolName = participantsSchoolsTextField.getText();

    try {
        database.searchSchool(newSchoolName); // should result in a NoSuchElementException
    exception

        // if an element with the same name is already in the database:
        // show the following exception dialog
        illegalActionDialogTextArea.setText("The name you typed in is "
            + "already taken. All schools must have unique names. If "
            + "you really wish to add a school with this name, first "
            + "locate the existing school and rename it.");

        illegalActionDialog.setVisible(true);
    } catch (IllegalArgumentException ex) { // if the name is an empty string
        // set the new school's name to a generic name with the current new no
        name school number and insert the new school
        newSchoolName = "<No name " + currentNewNoNameSchoolNumber + ">";
        database.insertSchool(newSchoolName);

        // increment the current new no name school number
        currentNewNoNameSchoolNumber += 1;

        // increment the number of schools
        schoolsNumber += 1;

        // set the progress bar to reflect the change in the number of schools
        this.updateParticipantsProgressBar();

        // new assignment of codes will be needed
        studentChanges = true;

        // empty the schools text field
        participantsSchoolsTextField.setText("");

        // find the new school in the database
        int newSchoolIndex = database.searchSchool(newSchoolName);
        this.autoselectSchoolActionBox(newSchoolIndex);

        // set the student codes button to the correct status
        this.recheckStudentCodesActionBox();
    } catch (NoSuchElementException ex) { // if the name was not encountered in
    the database
        // insert the school
        database.insertSchool(newSchoolName);

        // increment the number of schools in the database
        schoolsNumber += 1;

        // new assignment of codes will be needed
        studentChanges = true;

        // empty the schools text field
        participantsSchoolsTextField.setText("");

        // find the new school in the database
        int newSchoolIndex = database.searchSchool(newSchoolName);
        this.autoselectSchoolActionBox(newSchoolIndex);

        // set the student codes button to the correct status
        this.recheckStudentCodesActionBox();
    } finally { // do in any case
        participantsSchoolsTextField.grabFocus();
    }
}

```

The `IllegalArgumentException`, on the most general level, is an exception thrown when an argument passed to a method is illegal.

In the case of the `BinarySearchTree`, an identifier of an object to be inserted is illegal if it is exactly equal to an object already in the binary search tree. The action is prevented by the throwing of an `IllegalArgumentException`. The error handling is demonstrated in the third excerpt from the `School` class.

The exception is used in the same way in many other tree-insertion methods in other classes.

This usage of the `IllegalArgumentException` is found in:

- All methods with the word “Insert” in their identifier

However, there is also another usage of the `IllegalArgumentException`, relating to the adding, editing, and searching of entries in the participants trees. This is shown in the fourth excerpt. The exception is thrown when the new identifier is an empty string. That is acceptable in the case of adding – an entry with a generic name will be added to the tree –, but not acceptable in the case of editing or searching (clicking on the buttons is ignored in those cases).

This usage of the `IllegalArgumentException` is found in:

- All methods with the word “Add” in their identifier
- All methods with the word “Edit” in their identifier
- All methods with the word “Search” in their identifier

NoSuchElementException*(BinarySearchTree class, page 268)*

```

public Object delete(String keyString) throws NoSuchElementException {
    if (isEmpty()) { // if the tree is empty
        throw new NoSuchElementException("The data to be deleted is not in the
tree.");
    } else if (root.keyString.equals(keyString)) { // if the root will have to be
deleted
        // make a temporary root with the current root as a left node
        Node tempRoot = new Node();
        tempRoot.left = root;
        // then call the delete helper on the tempRoot
        Node deletedNode = deleteHelper(keyString, tempRoot, root);
        // after that, set the root to be the left node of the tempRoot
        root = tempRoot.left;
        // decrement size, acknowledge changes
        size -= 1;
        dataArrayChanged = true;
        // return the data object of the deleted node
        return deletedNode.data;
    } else { // if the node to be deleted is any other node than the root
        // call the delete helper (throws NoSuchElementException if the key to be
deleted is not in the tree
        Node deletedNode = deleteHelper(keyString, null, root);
        // decrement size, acknowledge changes
        size -= 1;
        dataArrayChanged = true;
        // return the data object of the deleted node
        return deletedNode.data;
    }
}
)

```

(BinarySearchTree class, page 268)

```

private Node deleteHelper(String keyString, Node parent, Node currentNode) throws
NoSuchElementException {
    if (keyString.compareTo(currentNode.keyString) < 0) { // if the key of the
current node is higher than the one searched for
        //
        if (currentNode.left != null) {
            return deleteHelper(keyString, currentNode, currentNode.left);
        } else {
            throw new NoSuchElementException("The data to be deleted is not in the
tree.");
        }
    } else if (keyString.compareTo(currentNode.keyString) > 0) { // if the key of
the current node is lower than the one searched for
        if (currentNode.right != null) {
            return deleteHelper(keyString, currentNode, currentNode.right);
        } else {
            throw new NoSuchElementException("The data to be deleted is not in the
tree.");
        }
    } else { // key is equal to what was being looked for - correct data found
        Node deletedNode = new Node();
        deletedNode.setNodeContents(currentNode);

        if (currentNode.left != null && currentNode.right != null) {
            currentNode.setNodeContents(minNode(currentNode.right)); // set
currentNode to have data of the minimum node to the right

            deleteHelper(currentNode.keyString, currentNode, currentNode.right);
// delete the smallest value on its original position
        }

        // one or both of the left/right nodes is missing
        else if (parent.left == currentNode) { // if we are to the left from the
parent
            parent.left = (currentNode.left != null) ? currentNode.left :
currentNode.right;
            // if left is not null, set this to be left
        }
    }
}

```

```

        // else set this to be right
    }

    else if (parent.right == currentNode) { // if we are to the right from the
parent
        parent.right = (currentNode.left != null) ? currentNode.left :
currentNode.right;
            // if left is not null, set this to be left
            // else set this to be right
    }
    return deletedNode;
}
}

```

(Database class, page 281)

```

public int [] globalSearchTeacher(String name, int preferredSchoolIndex) throws
IllegalArgumentException, NoSuchElementException {
    if (name.equals("")) throw new IllegalArgumentException();

    Object[] schoolTreeArray = schoolTree.getDataArray();

    if (preferredSchoolIndex == -1) {
        preferredSchoolIndex = 0;
    } else if (preferredSchoolIndex >= 0 && preferredSchoolIndex <
schoolTreeArray.length) {
        // do nothing
    } else {
        throw new IllegalArgumentException();
    }

    try {
        School currentSchool = (School) schoolTreeArray[preferredSchoolIndex];

        int teacherIndex = currentSchool.searchTeacher(name);

        int[] teacherIndices = {preferredSchoolIndex, teacherIndex};
        return teacherIndices;
    } catch (IllegalArgumentException ex) { // the teacher name was an empty
string
        throw new IllegalArgumentException();
    } catch (NoSuchElementException ex) { // teacher not found in this shool
        // do nothing, let it continue
    }

    for (int schoolIndex = 0; schoolIndex < schoolTreeArray.length; schoolIndex++)
{
        if (schoolIndex == preferredSchoolIndex) continue;

        try {
            School currentSchool = (School) schoolTreeArray[schoolIndex];

            int teacherIndex = currentSchool.searchTeacher(name);

            int [] teacherIndices = {schoolIndex, teacherIndex};
            return teacherIndices;
        } catch (IllegalArgumentException ex) { // the teacher name was an empty
string
            throw new IllegalArgumentException();
        } catch (NoSuchElementException ex) { // teacher not found in this shool
            continue;
        }
    }

    throw new NoSuchElementException(); // if NoSuchElementExceptions encountered
for all schools
}

```

(MainGUI class, page 236)

```

private void
participantsTeachersSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
    String searchedName = participantsTeachersTextField.getText();
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    int searchedTeacherSchoolIndex;
    int searchedTeacherIndex;

```

```

try {
    participantsTeachersTextField.setText("");
    int[] searchedTeacherIndices = database.globalSearchTeacher(searchedName,
schoolsIndex); // should continue
    searchedTeacherSchoolIndex = searchedTeacherIndices[0]; // the schoolsList
index
    School teacherSchool = (School)
database.getSchoolTreeNodeData(searchedTeacherSchoolIndex);
    this.updateSchoolsListModel();
    participantsSchoolsList.setSelectedIndex(searchedTeacherSchoolIndex);
    participantsSchoolsList.ensureIndexIsVisible(searchedTeacherSchoolIndex);
    searchedTeacherIndex = searchedTeacherIndices[1]; // the teachersList
index
    this.updateTeachersListModel(teacherSchool);
    participantsTeachersList.setSelectedIndex(searchedTeacherIndex);
    participantsTeachersList.ensureIndexIsVisible(searchedTeacherIndex);
} catch (IllegalArgumentException ex) { // if the name is an empty string (=BAD)
    // do nothing
} catch (NoSuchElementException ex) { // if the name was not encountered (=BAD)
    participantsTeachersTextField.setText("");
    searchedTeacherIndex = -1;
    this.updateTeachersListModel(currentSchool);
    participantsTeachersList.setSelectedIndex(searchedTeacherIndex);
} finally { // do in any case
    participantsTeachersTextField.grabFocus();
}
}
}

```

The NoSuchElementException is an exception thrown when an object is not found. In the case of the BinarySearchTree, a NoSuchElementException is thrown when the program tries to search or delete a node that is not in the tree (1st and 2nd excerpt). The exception can be seen in many other removal methods in other classes.

The third excerpt provides an example of a search that throws a NoSuchElementException if the desired identifier is not found.

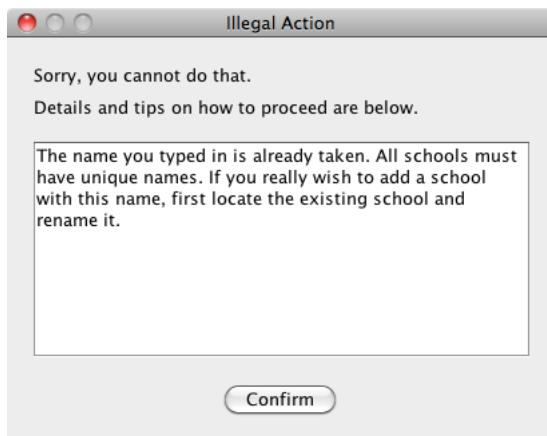
The fourth excerpt is an example of the handling of the NoSuchElementException in the case of searching. If the desired element is not found after clicking the Search button, the program recognizes it and takes action to show that the element was not found (deselecting previously selected teacher without selecting another). The user is not presented with an exception dialog, though.

This usage of the NoSuchElementException is found in:

- All methods with the word “Remove” in their identifier
- All methods with the word “Search” in their identifier

Search is also an integral part of the process of adding a data entry to the database, though. If and only if a search for the identifier provided to the insert method throws a NoSuchElementException (and provided that there is not an IllegalArgumentException as well), the entry can be taken into the database. (This can be seen in the fourth excerpt in the IllegalArgumentException section above.)

Otherwise, an exception dialog is shown, of which an example is reproduced below:



This usage of the NoSuchElementException is found in:

- All methods with the word “Add” in their identifier

Checking of Settings Values

(Database class, page 275)

```

public boolean [] checkSettings() {
    boolean [] checkArray = new boolean[29]; // true means "no problem", false
means "this value is illegal"

    checkArray[0] = (ooStudentRoomLimit > 0) ? true : false; // if the variable
has a legal value, return true, else return false
    checkArray[1] = (oiStudentRoomLimit > 0) ? true : false;
    checkArray[2] = (isStudentRoomLimit > 0) ? true : false;
    checkArray[3] = (daStudentRoomLimit > 0) ? true : false;
    checkArray[4] = (ooFinalStudentRoomLimit > 0) ? true : false;
    checkArray[5] = (oiFinalStudentRoomLimit > 0) ? true : false;
    checkArray[6] = (isFinalStudentRoomLimit > 0) ? true : false;
    checkArray[7] = (daFinalStudentRoomLimit > 0) ? true : false;

    checkArray[8] = (ooStudentSchoolLimit > 0) ? true : false;
    checkArray[9] = (oiStudentSchoolLimit > 0) ? true : false;
    checkArray[10] = (isStudentsSchoolLimit > 0) ? true : false;
    checkArray[11] = (daStudentSchoolLimit > 0) ? true : false;
    checkArray[12] = (debateStudentSchoolLimit > 0) ? true : false;

    checkArray[13] = (ooJudgeRoomLimit > 0) ? true : false;
    checkArray[14] = (oiJudgeRoomLimit > 0) ? true : false;
    checkArray[15] = (isJudgeRoomLimit > 0) ? true : false;
    checkArray[16] = (daJudgeRoomLimit > 0) ? true : false;
    checkArray[17] = (debateJudgeRoomLimit > 0) ? true : false;
    checkArray[18] = (finalsJudgeRoomLimit > 0) ? true : false;

    checkArray[19] = (roomLimit1 > 0) ? true : false;
    checkArray[20] = (roomLimit2 > 0) ? true : false;
    checkArray[21] = (timeLimit1 > 0 && timeLimit1 <= 24) ? true : false;
    checkArray[22] = (timeLimit2 > 0 && timeLimit2 <= 24) ? true : false;
    checkArray[23] = (schoolNumber > 0) ? true : false;
    checkArray[24] = (teacherSchoolNumber > 0) ? true : false;
    checkArray[25] = (studentSchoolNumber > 0) ? true : false;

    checkArray[26] = true; // this can only be true or false (which are both legal
possibilities, so there cannot be any problems

    checkArray[27] = (allowCombinedEvents) ? ((combinedEvent1 >= 0 &&
combinedEvent1 < 4) ? true : false) : true; // only reports a problem if combined
events are allowed
    checkArray[28] = (allowCombinedEvents) ? ((combinedEvent2 >= 0 &&
combinedEvent2 < 4) ? true : false) : true;

    return checkArray;
}

```

(MainGUI class, page 156)

```

private String approveDatabaseSettings(Database database) {
    boolean[] checkArray = database.checkSettings();

    String errorString = "";
    String returnString;

    errorString += checkArray[0] ? "" : "Illegal value: OO students/room limit\n";
// if the variable has a legal value, return true, else return false
    errorString += checkArray[1] ? "" : "Illegal value: OI students/room limit\n";
    errorString += checkArray[2] ? "" : "Illegal value: IS students/room limit\n";
    errorString += checkArray[3] ? "" : "Illegal value: DA students/room limit\n";
    errorString += checkArray[4] ? "" : "Illegal value: OO finals students/room
limit\n";
    errorString += checkArray[5] ? "" : "Illegal value: OI finals students/room
limit\n";
    errorString += checkArray[6] ? "" : "Illegal value: IS finals students/room
limit\n";
    errorString += checkArray[7] ? "" : "Illegal value: DA finals students/room
limit\n";

    errorString += checkArray[8] ? "" : "Illegal value: OO students/event/school
limit\n";

```

```

    errorString += checkArray[9] ? "" : "Illegal value: OI students/event/school
limit\n";
    errorString += checkArray[10] ? "" : "Illegal value: IS students/event/school
limit\n";
    errorString += checkArray[11] ? "" : "Illegal value: DA students/event/school
limit\n";
    errorString += checkArray[12] ? "" : "Illegal value: Debate
student/event/school limit\n";

    errorString += checkArray[13] ? "" : "Illegal value: OO judges/room limit\n";
    errorString += checkArray[14] ? "" : "Illegal value: OI judges/room limit\n";
    errorString += checkArray[15] ? "" : "Illegal value: IS judges/room limit\n";
    errorString += checkArray[16] ? "" : "Illegal value: DA judges/room limit\n";
    errorString += checkArray[17] ? "" : "Illegal value: Debate judges/room
limit\n";
    errorString += checkArray[18] ? "" : "Illegal value: Finals judges/room
limit\n";

    errorString += checkArray[19] ? "" : "Illegal value: Rooms available day 1\n";
    errorString += checkArray[20] ? "" : "Illegal value: Rooms available day 2\n";
    errorString += checkArray[21] ? "" : "Illegal value: Tournament time day 1\n";
    errorString += checkArray[22] ? "" : "Illegal value: Tournament time day 2\n";
    errorString += checkArray[23] ? "" : "Illegal value: Schools in tournament\n";
    errorString += checkArray[24] ? "" : "Illegal value: Teachers/school\n";
    errorString += checkArray[25] ? "" : "Illegal value: Students/school\n";

// checkArray[26] can only be true

    errorString += checkArray[27] ? "" : "Illegal value: Combined events 1\n";
    errorString += checkArray[28] ? "" : "Illegal value: Combined events 2\n";

    if (generalSettingsCEComboBox1.getSelectedIndex() ==
generalSettingsCEComboBox2.getSelectedIndex()) {
        errorString += "Illegal input: Combined events combo boxes both report the
same event type\n";
    }

    if (!errorString.equals("")) {
        returnString = errorString.substring(0, (errorString.length() - 1));
    } else {
        returnString = errorString;
    }

    return returnString;
}

```

(MainGUI class, page 224)

```

private void settingsApproveToggleButtonMouseReleased(java.awt.event.MouseEvent evt) {
    if (settingsApproveToggleButton.isSelected()) { // if the approve button was
selected by this action
        try { // try to approve settings
            // set the database settings to correspond to settings in the settings
tab
            this.changeDatabaseSettings(database);

            // see if the settings are correct
            if (this.approveDatabaseSettings(database).equals("")) { // if the
settings are correct
                this.enableParticipantsTabActionBox();

                tabbedPane.setSelectedIndex(1);
            } else { // if the settings are incorrect
                // show an exception dialog listing all the settings that are
incorrect
                settingsErrorDialogTextArea.setText(this.approveDatabaseSettings(database));
                settingsErrorDialog.setVisible(true);

                // deselect the approve button
                settingsApproveToggleButton.setSelected(false);
            }
        } catch (IllegalArgumentException ex) { // if, for whatever reason, the
error handling failed at previous levels
            // show the following exception dialog
            exceptionDialogTextArea.setText("Some of the settings values "
+ "were incorrect and this could not be resolved by "

```

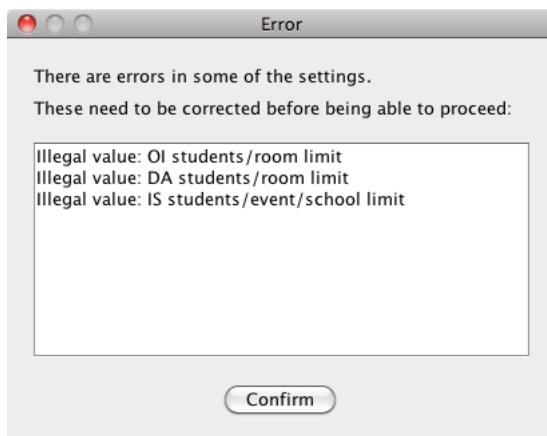
```
+ "the program. Look for erroneous values in the "
+ "settings text fields and correct them yourself.");
exceptionDialog.setVisible(true);
settingsApproveToggleButton.setSelected(false);
}
} else { // if approveToggleButton was deselected by this action
    this.enableSettingsTabActionBar();
}
}
```

Checking of settings for correctness is a distinct kind of error handling, one more suitable for the batch processing of data as can be seen with the approval of settings. When the user clicks the approve settings button, the original values of settings in the database are rewritten by those from the input text fields. Then, these are checked for correctness by the checkSettings() method. For most of the text fields, “correct” means that they have a positive numerical value, they are correct. Some numerical values (hours per day, for example) have an upper bound, and the value is deemed incorrect also if it is higher than that limit value.

How is abnormal input detected, however? It can be detected because all of abnormal input is converted by one of the helper methods getTextFieldValue() to the value of -1. And -1 is never a correct value, so abnormal input is always identified.

The result of the checkSettings() method is an array, in which all settings with illegal values are marked as false. This array is used as a basis for the MainGUI’s method approveDatabaseSettings(), which checks all of the elements in the array and if they are false, adds a string notice about that to an error string.

Finally, then, if the error string is not empty after the checking, it means that there was an erroneous input in at least one of the text fields. The settings are not approved, and an exception dialog is shown, listing all fields with erroneous input.



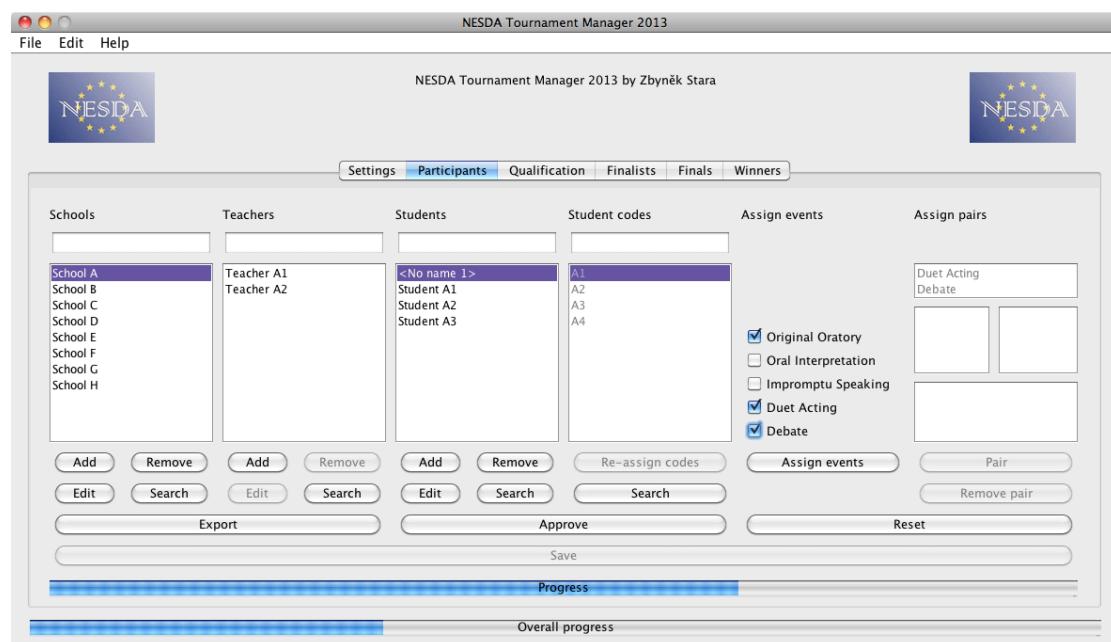
In the above case, therefore, the values of OI students per room, DA students per room and IS students per school need to be corrected.

Error Prevention:

At several places in the program, the user input is limited with GUI elements. This is a form of error prevention and error handling because this way, the user is choosing one of correct values. It is not possible for the user to input an erroneous value because there are no erroneous values available for him to select, and there is no possibility of the user adding something else.

Assign Events:

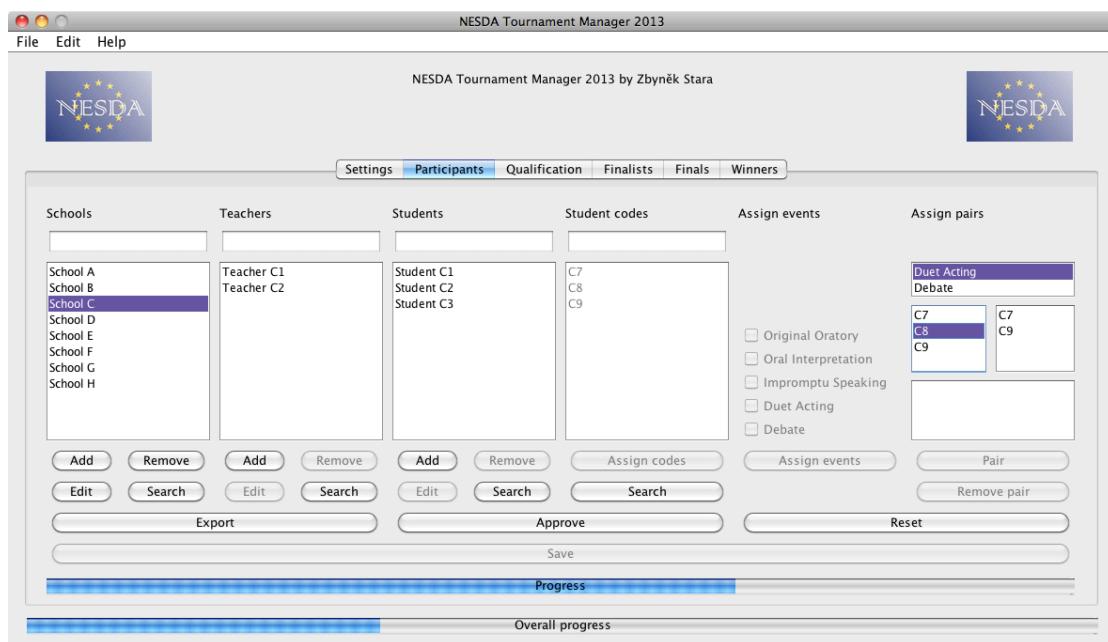
The user input for assigning events is limited by a group of check boxes. They are designed to allow for precise input of events without the need for writing them in or the need for using acronyms. They can be seen in the fifth column of the participants tab:



By having the user's input restricted to these choices, there does not have to be error checking in place for the assignment of events.

Assign Pairs:

The user input for assigning pairs is limited to two lists of unpaired students for a given event (which is either duet acting or debate). That way, the user input is limited to only include valid relevant students – those that participate in the specific event while not yet being paired. The assign pairs lists can be seen in the sixth column of the participants tab:



Again, the user can only decide to form the pair from two valid judges, one from the left list, and one from the right list. Moreover, the pair button is only enabled when two students are selected for the pair. No textual input is necessary, so there is no space for errors.

**Section C3:
Success of the Program**

All of the essential tasks work. During the programming stage, it became clear that the two criteria for success in fact encompassed much more work than was expected either by me or the user, and that there were several important tasks that needed to be done as well. Unfortunately, then, no time was left to implement the extension ideas suggested by the user, as outlined in the section A2, although the program is prepared for them. Look to the section D2 for more detailed evaluation.

The first of the original criteria had everything to do with the Participants section (“it has to record all the students and teachers from all the schools and add them to the appropriate schools (along with the events and pairs information for students); it should assign student codes to the students”). The results of the task and relating necessary sub-tasks can be seen below:

- Adding of entries (schools, teachers, students) to the database (SUCCESS).
- Assigning of student codes according to NESDA conventions (SUCCESS).
- Assigning of events for students (SUCCESS).
- Adding of student pairs for Duet Acting and Debate events (SUCCESS).
- Exporting of the database (SUCCESS).
- Saving the database (SUCCESS).
- Loading files (SUCCESS).

The second essential criterion related entirely to the Qualification section (“the program has to be able to place all the participants – students and judges – into correct rooms, so that several criteria set by NESDA are not broken”). The summary can be seen below:

- Constructing entities from students and student pairs (SUCCESS).
- Allocating entities into qualification rounds (SUCCESS).
- Allocation progress bar (SUCCESS).
- Exporting of the qualification (SUCCESS).
- Saving the qualification (SUCCESS).

As there was not enough time to implement the extension ideas presented in the section A2, they had to be dropped from this build of the program. This did not affect the core functionality of the program. Summary:

- Recording scores for students (DROPPED).
- Indicating finalists based on scores (DROPPED).
- Recording rankings for finalists (DROPPED).
- Indicating the winners based on rankings (DROPPED).

The program manages to meet the speed limits, as indicated in section A2, except for those that were pertaining to dropped extension functionality. Summary:

- Assignment of student codes should take no longer than one second (SUCCESS – time is in milliseconds).
- Allocation of judges and entities should take no longer than 60 seconds (SUCCESS – maximal allocation time is 10 seconds per event, which is 50 seconds in total)
- Presenting finalists should take no longer than one second (DROPPED – functionality not implemented)
- Presenting winners should take no longer than one second (DROPPED – functionality not implemented)

The program fulfills the memory limits presented in A2. Summary:

- Be able to work with a minimum of 12 schools, 144 students and 34 teachers (SUCCESS).
- Be able to accommodate for more data entries than the minimum (SUCCESS).
- The program must not use more than 2 GiB of memory (SUCCESS – the program requires around 200 MiB of memory).

The program responds to extreme/erroneous data input as was laid out in the section A2. Summary:

- Showing a warning dialog when extreme or erroneous data entry was typed into the program (SUCCESS – widely used error handling technique).
- Alerting the user when scores typed in are out of the normal range (DROPPED – functionality not implemented)

**Section D1:
Test Output**

As this program has the ambitions to be used in the real world by real-life NESDA tournament organizers, testing its key areas is very important to ensure that there are no unpleasant surprises to the users.

The testing showed that the program performs well under normal conditions and is also able to handle various limit, extreme, and abnormal situations. And that applies not only to situations that include user input, but also those that are linked to the internal functionality of the program.

User input:

There are several input areas, to which data is entered directly by the user. These are the ones that require the most checking. One of these areas is the entry of settings values; other is the adding of schools/teachers/students into database, the other is searching in database; and the last one is renaming of the rounds/rooms in qualification.

Settings:

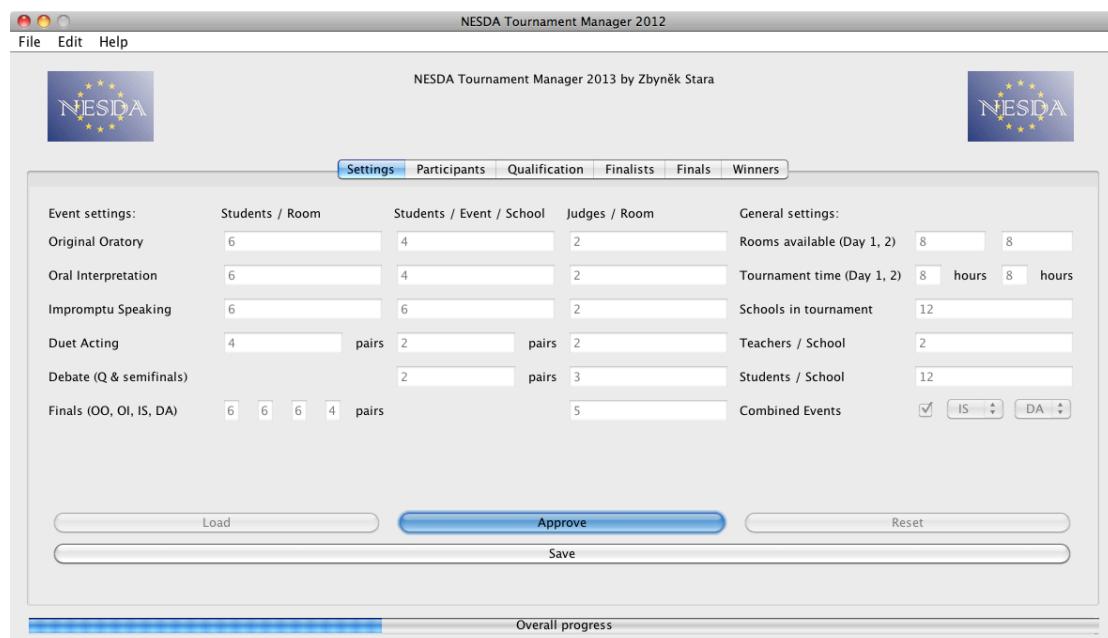
All settings that allow a direct text input need to be a number higher than zero (normal data). However, it is possible that the user would type in a negative number (extreme data) or even a letter (abnormal data) by mistake. The program needs to account for that.

The following is a screenshot of the settings tab. The pre-set values of the text fields, serve to communicate to the user that an integer is expected:

Event settings:	Students / Room	Students / Event / School	Judges / Room	General settings:
Original Oratory	6	4	2	Rooms available (Day 1, 2) 8 hours 8 hours
Oral Interpretation	6	4	2	Tournament time (Day 1, 2) 8 hours 8 hours
Impromptu Speaking	6	4	2	Schools in tournament 12
Duet Acting	4 pairs	2 pairs	2	Teachers / School 2
Debate (Q & semifinals)		2 pairs	3	Students / School 12
Finals (OO, OI, IS, DA)	6 6 6 4 pairs		5	Combined Events <input checked="" type="checkbox"/> IS DA

Normal:

The program checks the input values once the Approve button is clicked by the user. If a normal data was input, the program proceeds normally and the approve button becomes selected (also, the text fields in settings are deselected and participants tab becomes enabled). Notice that in order to proceed with the program, settings must be approved, so the data will always be checked. The situation after changing the data selected in the previous image is shown below:



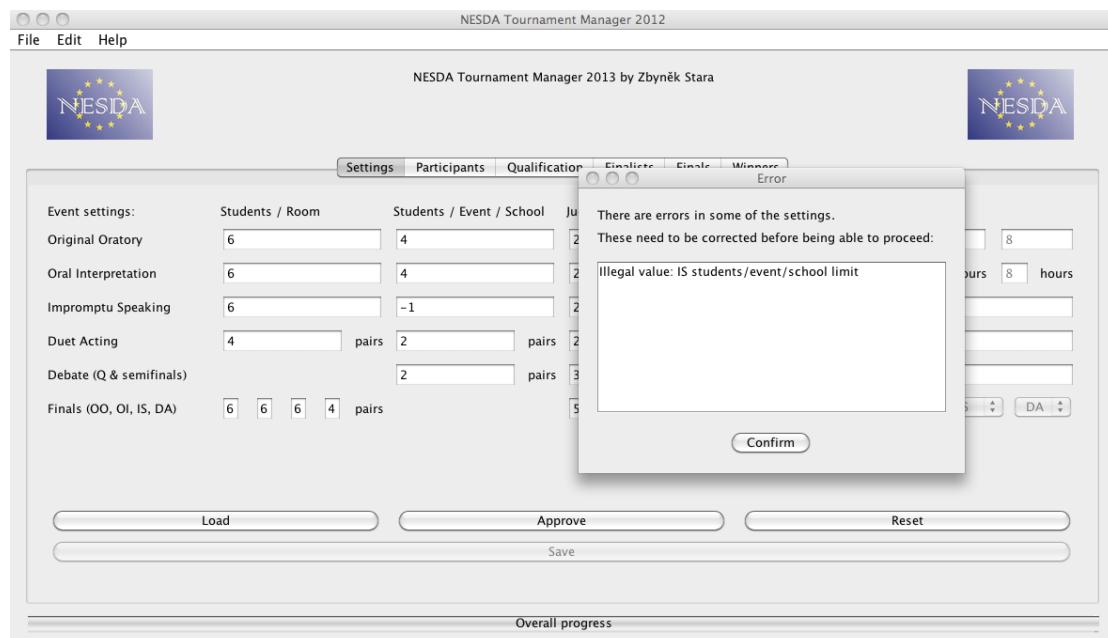
Notice that the value in the impromptu speaking students per event per school text field has changed from 4 to 6 and that the approve button has become selected.

Limit:

The program accepts the limit values of zero. The reason for that is that a given event might not be held at the tournament in question. For that reason, the limit values are treated as normal values.

Extreme:

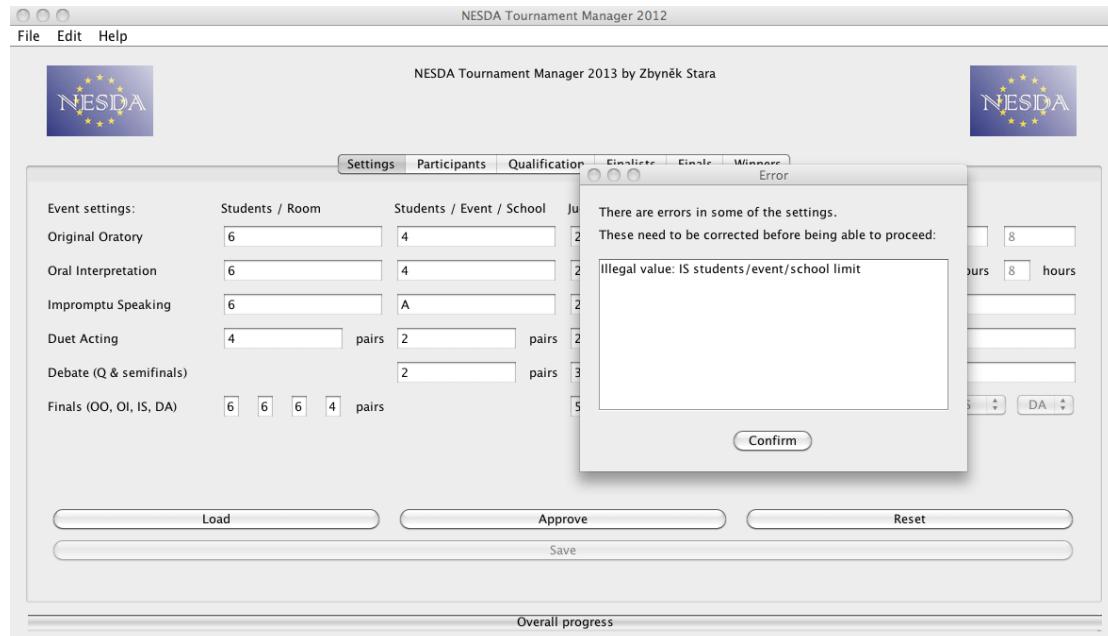
If the user inputs a number smaller than zero, the program picks upon that fact and notifies the user with a modal window once the approve button is clicked. The window lists all the text fields that have non-standard values in a text area. Refer to the following diagram:



The window is modal, so the user is not allowed to proceed until the confirm button is clicked. After the button is clicked, the program goes back to previous state, without approving, and the user has to change the erroneous value.

Abnormal:

If abnormal data is input to the program, it shows the same modal dialog and the user is informed that the value of the text field in question is illegal. Again, when the button is clicked, the settings tab waits for correction.

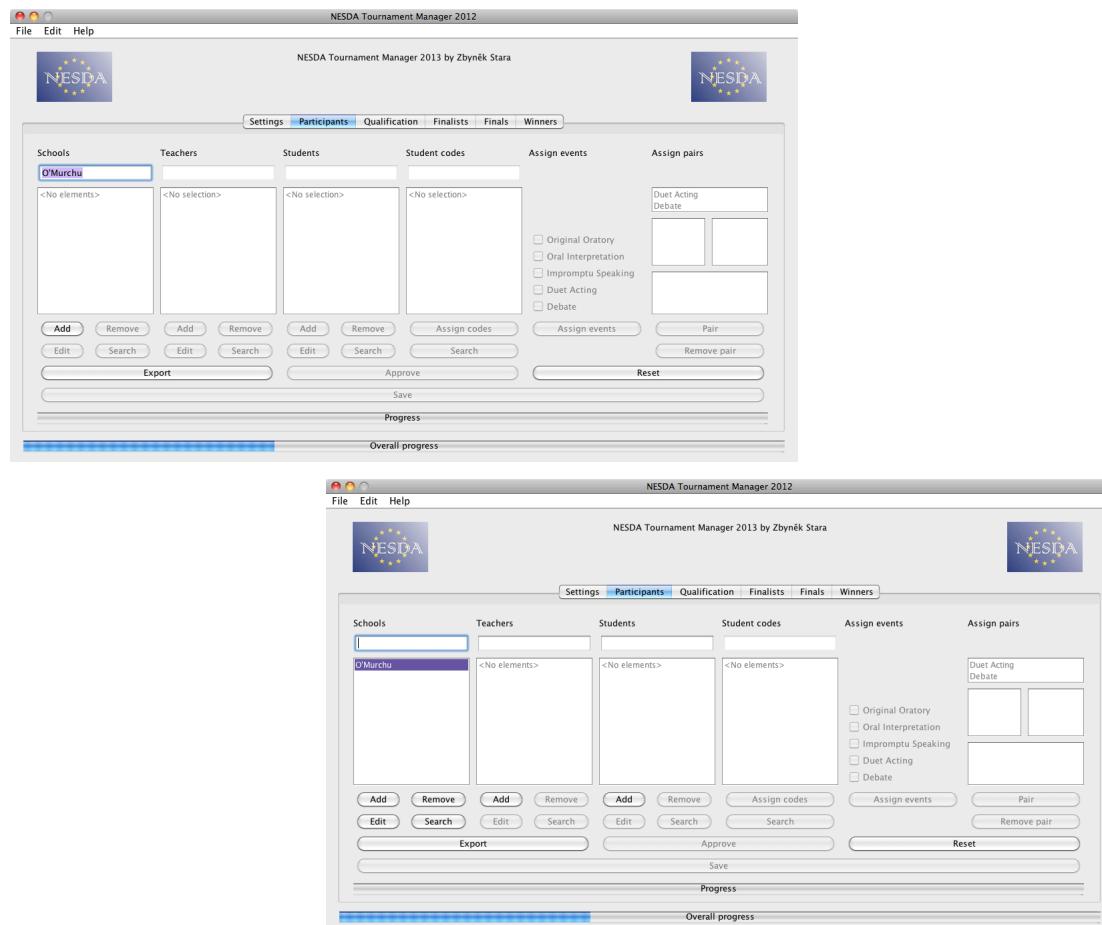


Adding a school/teacher/student to the database:

All database entries need a name to be identified with. This name is taken from a text field, where it is first typed in by the user. A non-empty string value is expected and the name must not already be used.

Normal:

A name can be any string; it is meant to be the name of the entry, but there is nothing to stop the user from typing in a number or a symbol as a placeholder. Because the names can be changed afterwards using the edit button, this is not a problem. As long as the string is not already used and the string is not empty, the value is considered normal. In that case, the entry is added to the database.



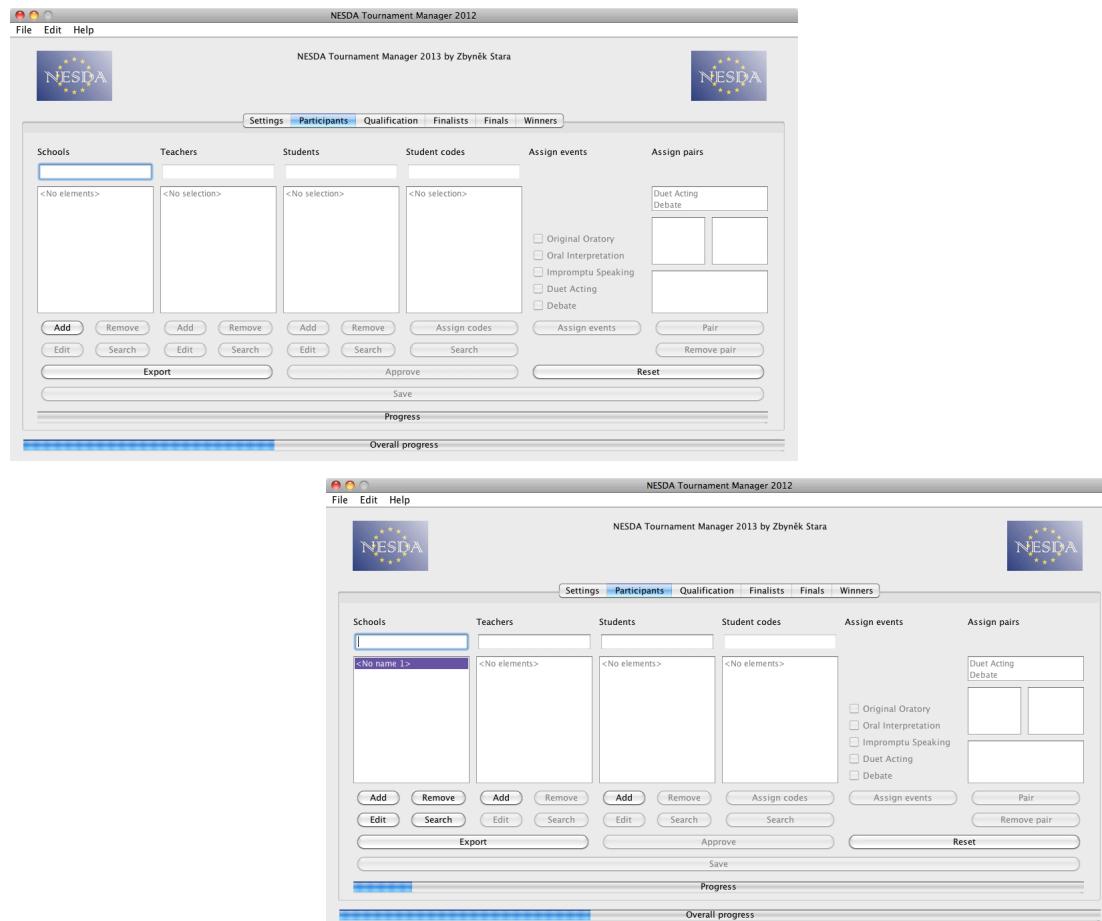
Notice the addition of the school to the school tree.

Extreme:

In the case of adding an entry to the database, extreme data is an empty string.

If the add button is clicked when there is nothing in the text field, this is resolved by making a new school with placeholder name in the form

<No Name _>, where a number is inserted instead of the _ symbol.

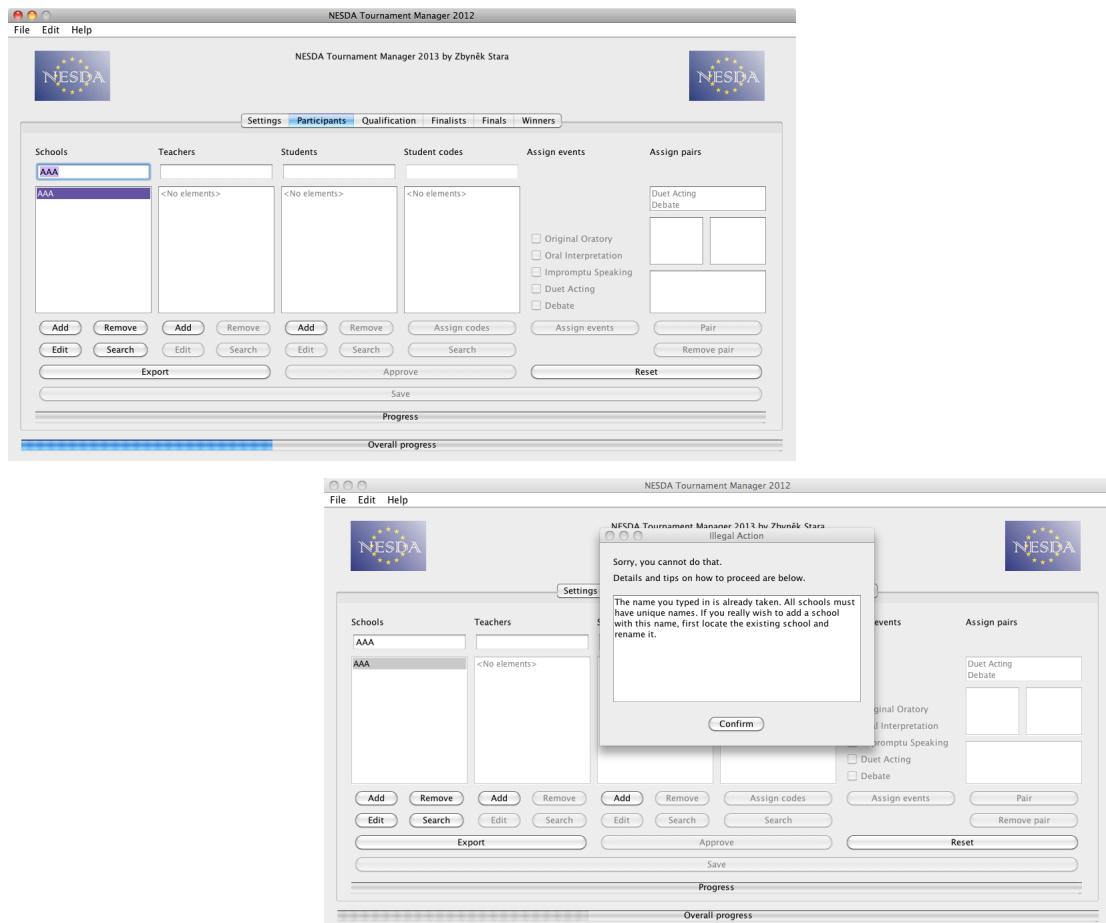


Notice the addition of the <No Name 1> to the school tree (number 1 because this is the first placeholder name school in the database).

Abnormal:

Abnormal data input in this case is a string that is already used in the database.

There cannot be two schools of the same name, nor two students or teachers (however, there can be two identically named students/teachers in two different schools).



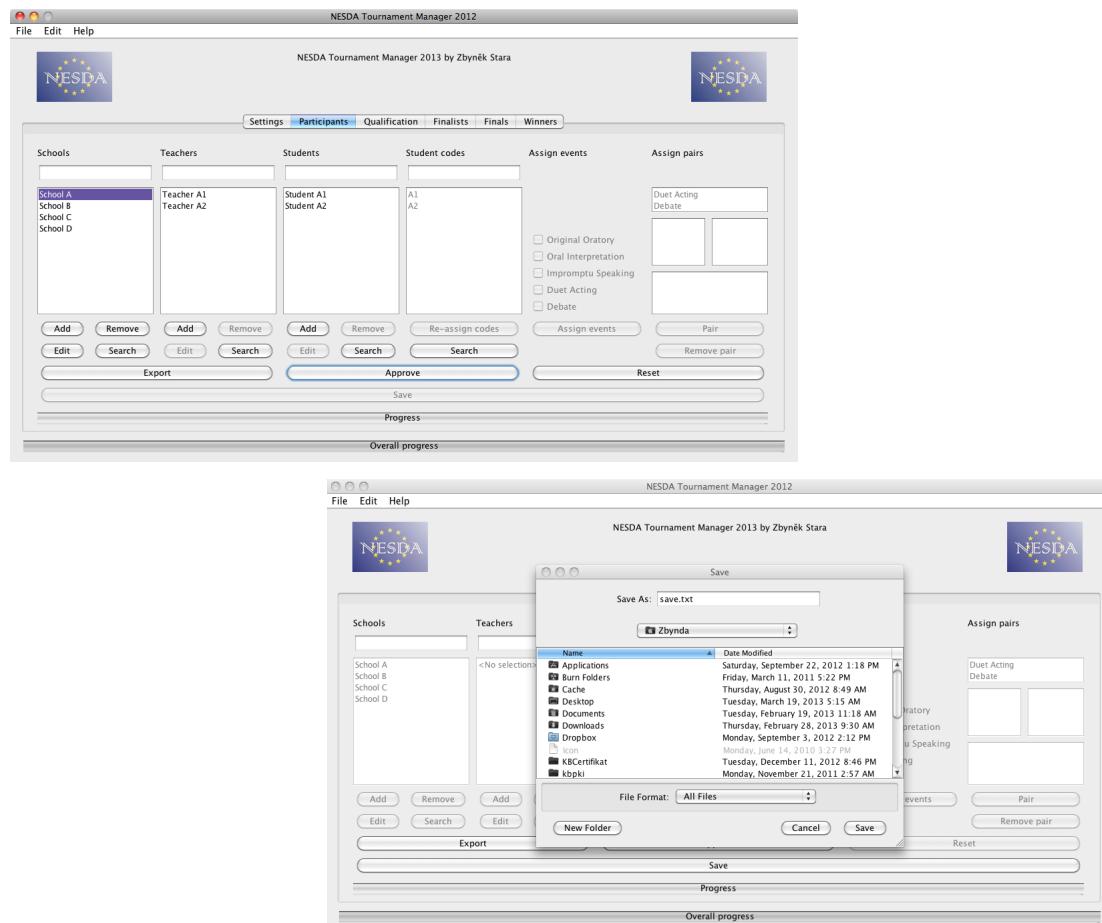
After the user clicks the add button trying to add a school of a name that is already found in the database, the program prevents him from doing that showing an error dialog, proposing tips on how to proceed to the user (choose a different name or rename the other school with the edit button). After the dialog is confirmed, the program goes back to the initial situation, with the text field waiting to be changed.

Internal functionality of the program:

Saving:

The program can be saved whenever it is checked error-free – that means whenever the approve button is selected. This is to ensure that the data input is correct before saving can be done. The program is saved up to the section, in which the save button has been clicked. This is to make it possible to save just a part of the program, say, the settings, without the need to have participants approved.

Let us take the participants save button as an example. A sample of data has been input into the database to show that the saving works.



When the save button is clicked, the user is prompted to choose a file to which to save. It is also possible to create a new file. If the choosing of file is aborted with the cancel button, the program proceeds as if there was no saving. The output in the file is shown below:

```

[NESDA Tournament Manager Save File]
Created by NESDA Tournament Manager 2012 on Mar-19-2013
05:21:53

IMPORTANT: Do not modify this file. Your modifications
might disrupt the program's parsing of saved information
and lead to errors or cause a possible loss of data.

NOTE: This file is not an export file of the data inputted
into the NESDA tournament manager. If you wish to obtain a
plain-text copy of a select section of the program, click
the appropriate "Export" button in the program's user
interface.

$Attributes
1
3,1,1
4,8,8
true,true,false

$Settings
#       6
#       6
#       6
#       4
#       6
#       6
#       6
#       6
#       4
#       4
#       4
#       4
#       2
#       2
#       2
#       2
#       2
#       2
#       5
#       8
#       8
#       8
#       12
#       2
#       12
>      true
?

#       8
#       12
#       2
#       12
>      true
#       2
#       3
&      true
&      false

$Participants
.           School A:A,1,2
?           Teacher A1
?           Teacher A2
!           Student
A1:A1,true,true,false,false,false,false,null,null,true
!
!           Student
A2:A2,false,false,true,true,false,true,false,null,null,true
/
!           Student A2
.           School B:B,3,4
?           Teacher B1
?           Teacher B2
!           Student
B1:B3,true,false,false,false,false,false,null,null,true
!
!           Student
B2:B4,false,false,true,false,false,false,false,null,null,true
.
.           School C:C,5,6
?           Teacher C1
?           Teacher C2
!           Student
C1:C5,false,true,true,false,false,false,null,null,true
!
!           Student
C2:C6,true,false,false,false,true,false,true,null,null,true
|
!           Student C2
.           School D:D,7,8
?           Teacher D1
?           Teacher D2
!           Student
D1:D7,true,false,true,true,false,true,false,null,null,true
!
!           Student
D2:D8,true,true,false,false,true,true,true,null,null,true
/
|
!           Student D1
!           Student D2

```

As can be seen, the inside of the saved file (here viewed with the application TextEdit) is ready-formatted for loading into the program. This file is not intended to be viewed by the user (there is the export functionality, which outputs the database entries as plain text, without viewing the settings, and without the flags visible on the left hand side).

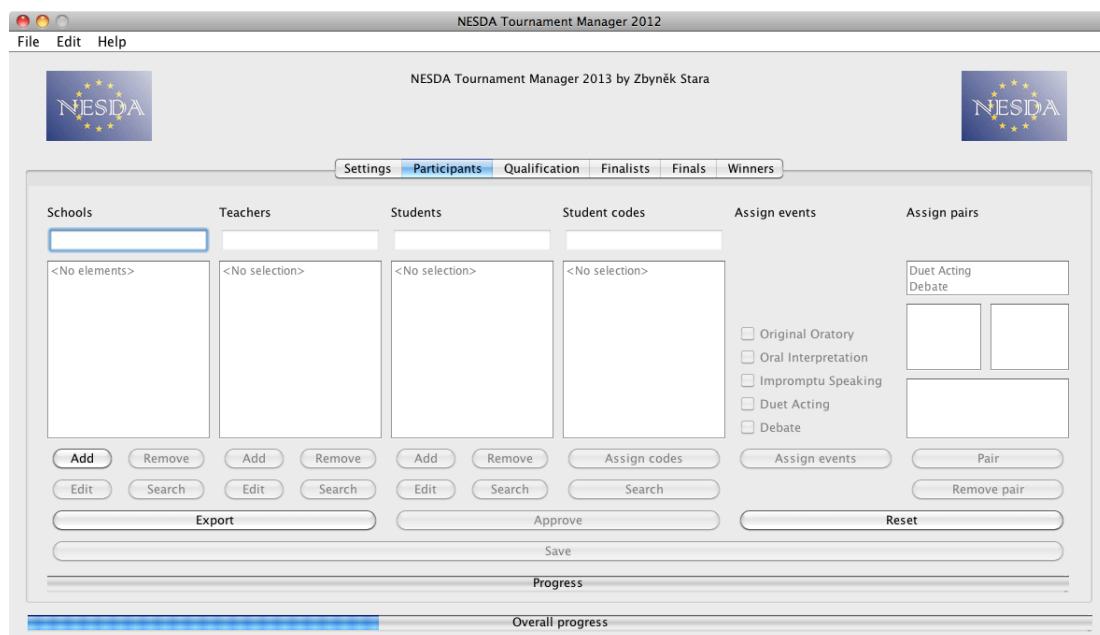
Loading:

The program has the functionality of loading in a saved file. Thus, the contents of the program when it was saved are exactly recreated in the database.

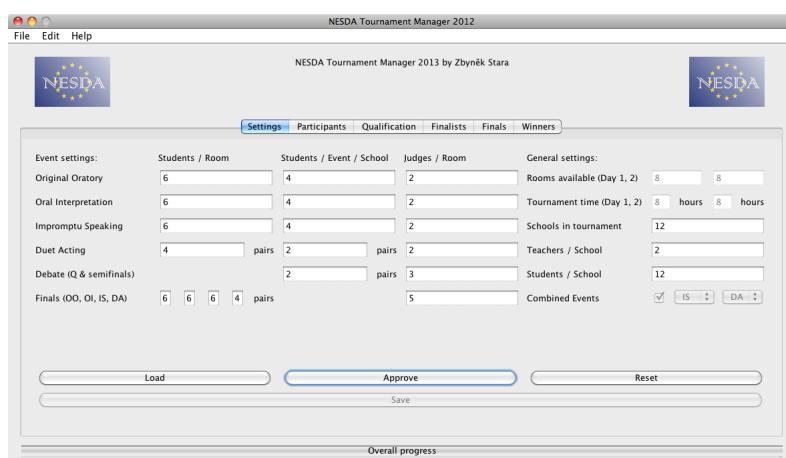
Loading is done by the load button in the settings section.

Normal:

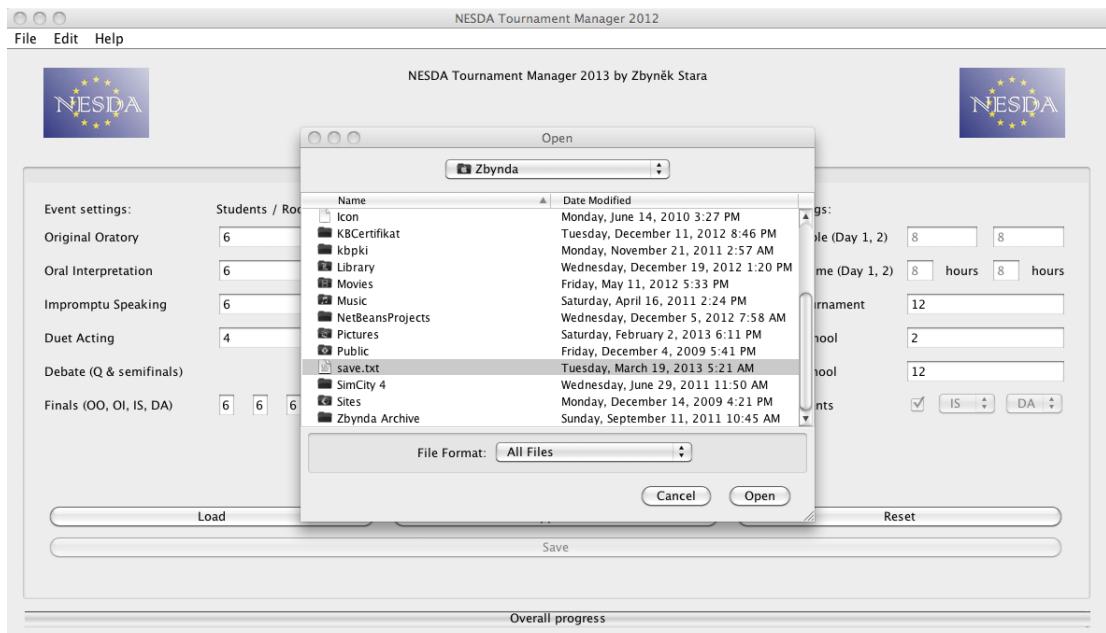
In the case of loading, a normal situation occurs when the user chooses a file and it is a file created by the program. This is illustrated below.



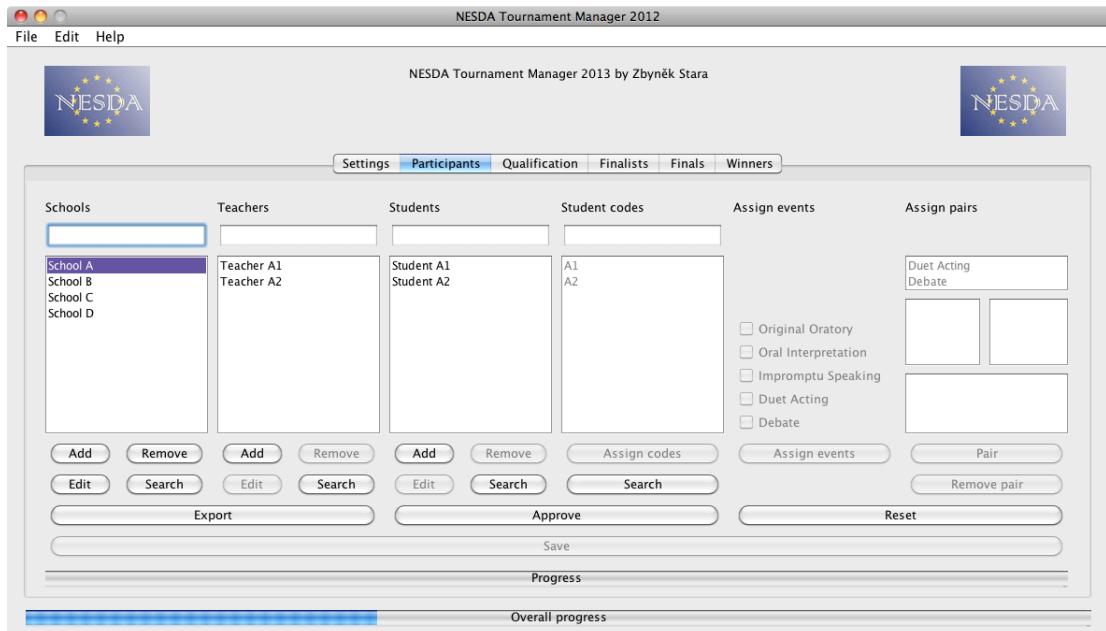
We start with a blank database.



The load button is located in the settings tab. To access it, it is necessary to de-select the settings approve button.



After the load button is clicked, a loading dialog is invoked and, allowing the user to select a file (we will select the one we saved in the saving section, save.txt).



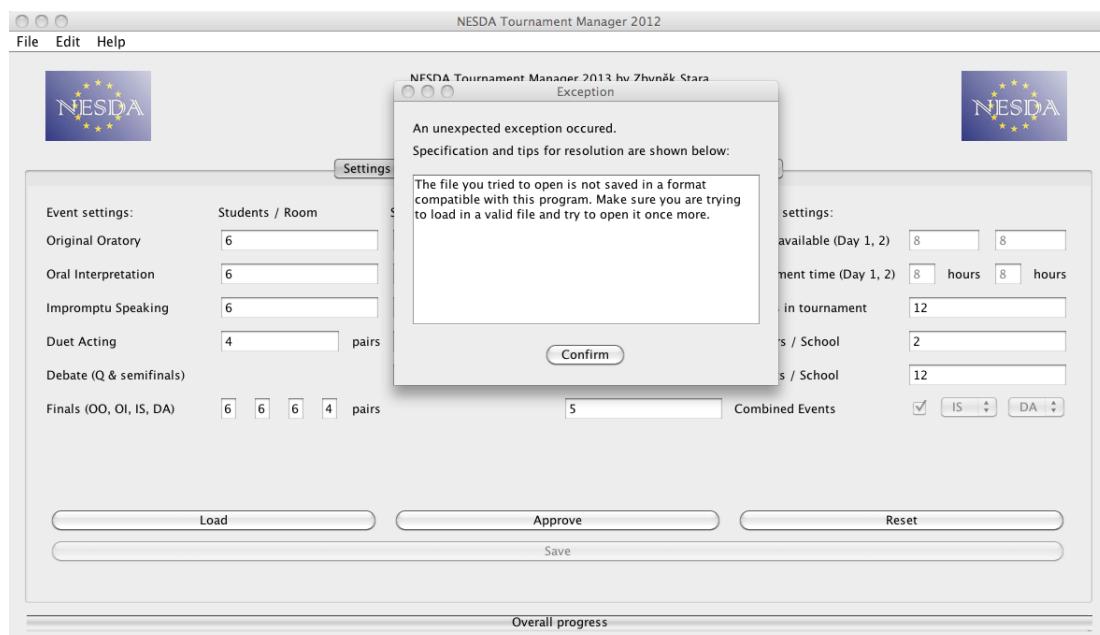
As we can see, the database is now updated to be the same as the one saved in our file.

Extreme:

In the case of loading, an extreme situation arises when the user cancels the selection of a load file. In that case, the program ends the loading process and proceeds as if the loading has never been initiated (it goes back to the settings tab to the situation illustrated in the second screenshot of the normal input).

Abnormal:

In this case, the file chosen to load is not a file created by this program, or it is, but in a wrong format (which could be a result of the user trying to edit the file). In that case, the program shows an exception dialog with tips to resolve the problem (make sure you chose the correct file), as illustrated below:



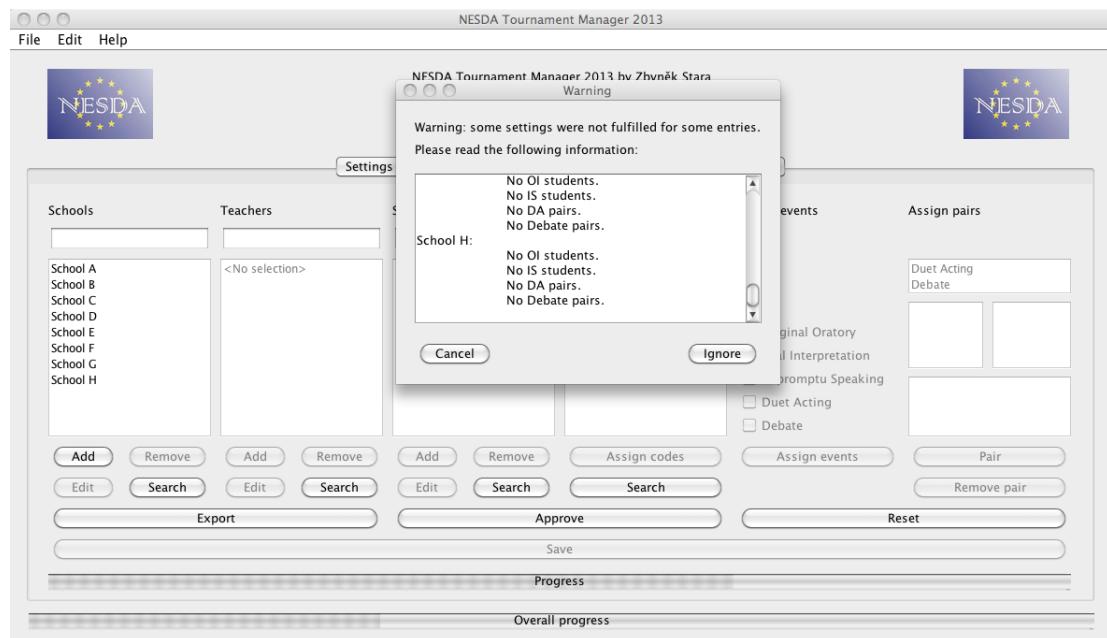
The exception dialog is modal, that means that it does not allow the user to proceed until the confirm button is clicked. After the button is clicked, the dialog is closed and the program proceeds as if there were not an attempt to load (meaning, without any changes to the database).

Allocation of entities and judges:

Allocation of entities and judges can be argued to be the main part of the program. Therefore, thorough testing of it with all sorts of data is necessary.

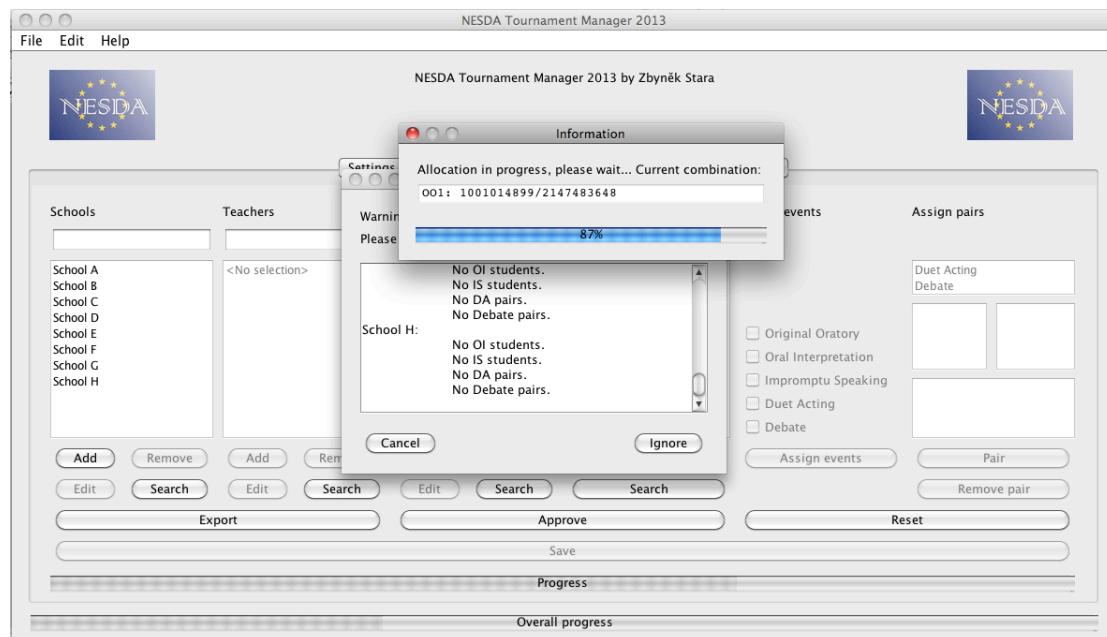
Normal:

Normal situation, in this context, is the situation when there is enough entities and judges for the allocation. After the user clicks on the ApproveParticipantsToggleButton, a standard warning dialog is shown:

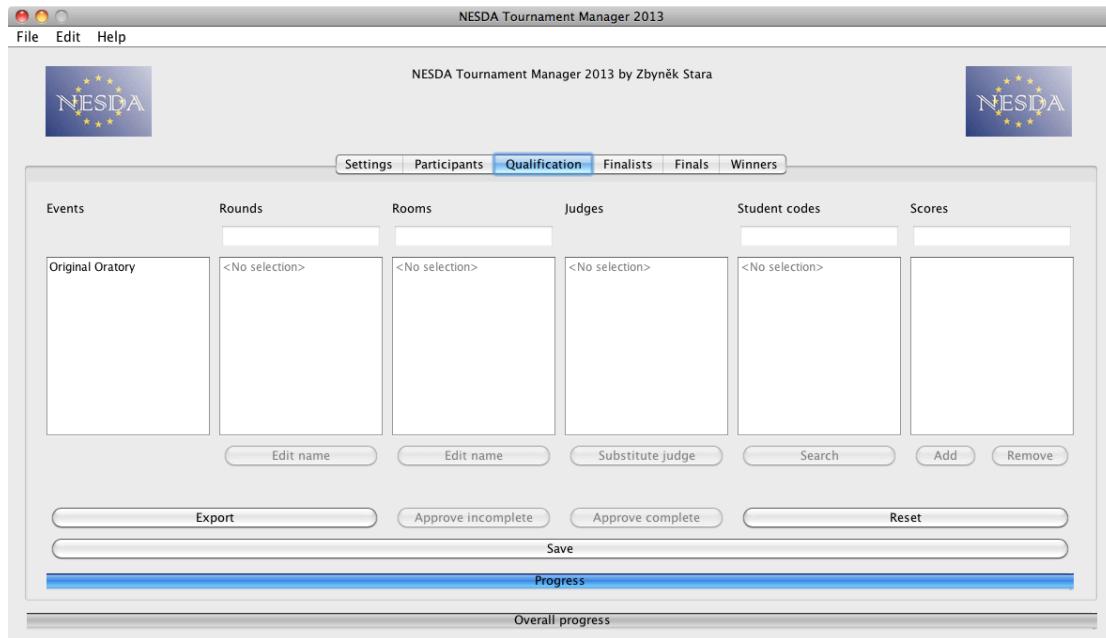


After the ignore button on the warning dialog is clicked, the allocation starts.

That can be seen on the following screenshot:



You can notice the progress dialog that has been made visible by the AllocationWorker that executes the allocation algorithm. The progress bar works, however there are some non-critical glitches (the current combination exceeding the maximum combination and the like). Still, the progress dialog fulfills its mission: it communicates to the user that something is being done.



This is the result. After the allocation finishes, the tab is switched to Qualification and the progress dialog and the approveParticipants warning dialog are both hidden. If we clicked on the Original Oratory event, we would see the different rounds in its roundArray, and in them the different rooms, and in them the different entities and judges. (The original oratory event is shown alone in the eventsList because we only had entities participating in original oratory, but the result would be the same with other events.) Alternatively, we could click the export button and have all of the information written out for us.

Limit:

The limit situation, in this case, is the situation when there is just more than the minimum number of participants in a given event. Using our example, this means 7 original oratory participants. The program should not have a problem with this data set.

The export of the Participants tab, along with one of the Qualification tab, are shown below:

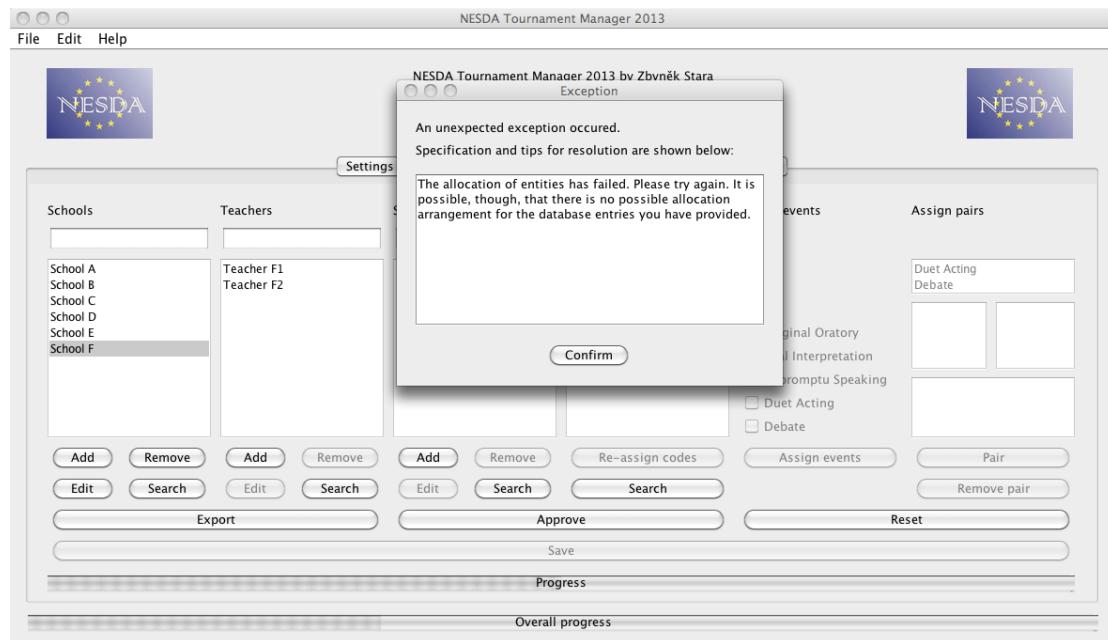
School A		Original Oratory	
		00 R1: Day 1, 11-12	
		Room 1	
A1	Student A3		Teacher E2
School B		Teacher F2	
			Teacher F2
Teacher B1			A1
Teacher B2			B2
B2	Student B3		C3
School C		G7	
Teacher C1		Room 2	Teacher A1
Teacher C2			Teacher C1
C3	Student C3		D4
School D			E5
			F6
Teacher D1		00 R2: Day 1, 16-17	
Teacher D2		Room 1	
D4	Student D3		
School E			Teacher F1
			Teacher B1
Teacher E1			A1
Teacher E2			C3
E5	Student E3		E5
School F			G7
Teacher F1		Room 2	Teacher C2
Teacher F2			Teacher E1
F6	Student F3		
School G			B2
			D4
Teacher G1			F6
Teacher G2			
G7	Student G3		

As can be seen from the test export on the right, all of the seven entities were successfully allocated.

Extreme:

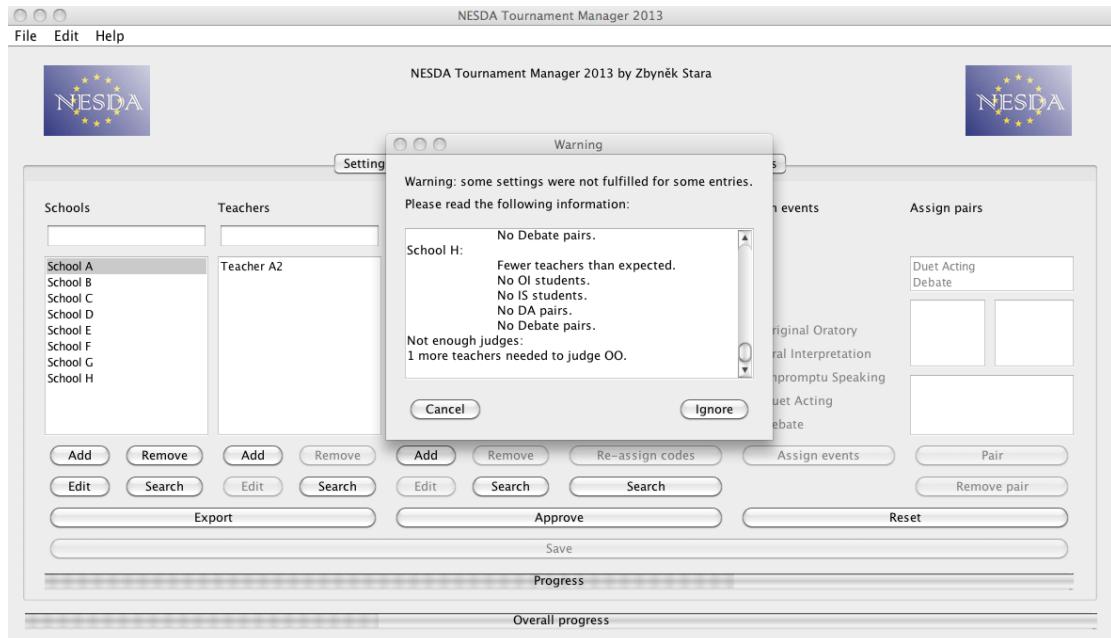
Extreme situation in this context is a situation when there are not enough entities or judges to allow for allocation.

On the next page is the result when the user tries to allocate with six entities:

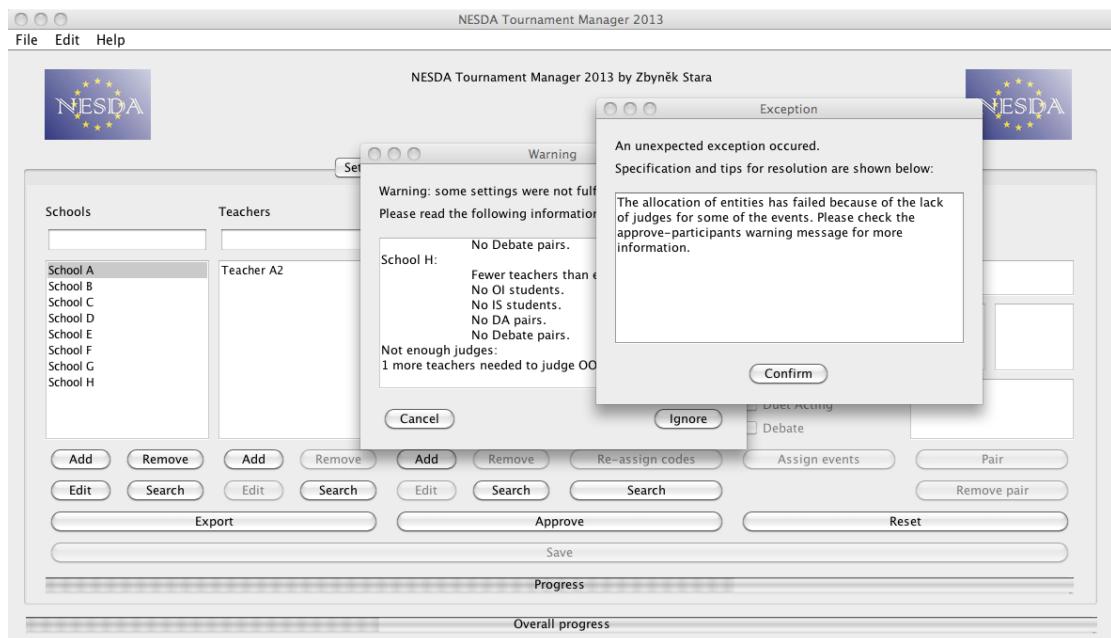


An exception dialog was invoked informing the user that the allocation of entities has failed.

What happens, then, when there is not enough judges? In a situation with 24 entities, 8 judges are needed. What if only 7 are provided?



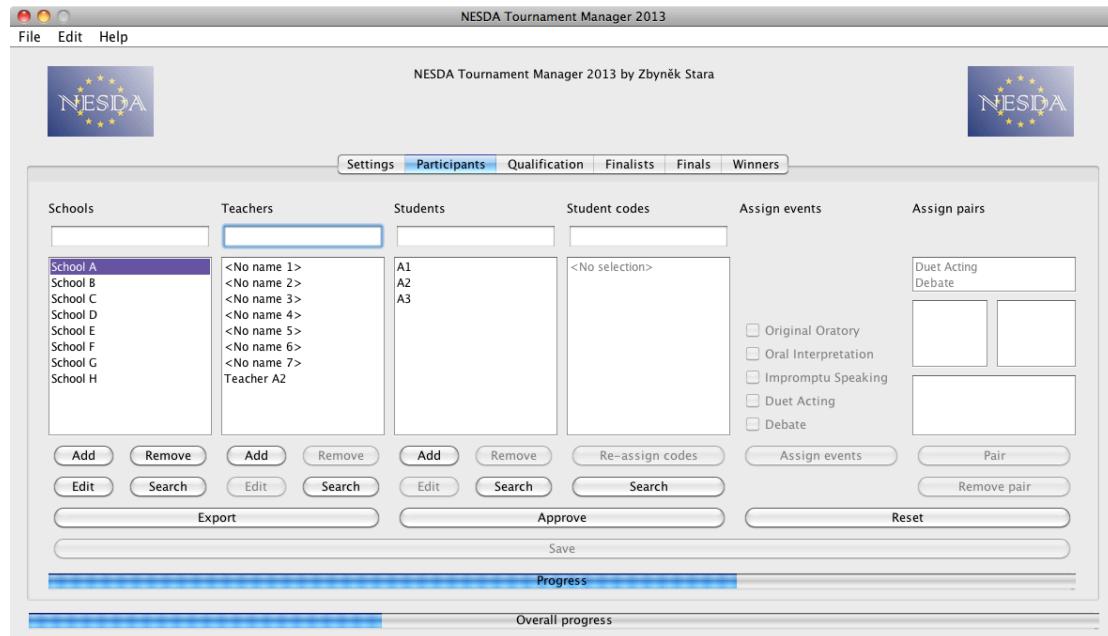
We can see that in this case, the warning window shows a notice that “1 more teacher is needed to judge OO” (original oratory). However, the user still can proceed by clicking the ignore button. How does the program handle that situation?



The program displays an exception dialog explaining that there is not enough judges for a successful allocation and prompting the user to re-check the approve-participants warning message.

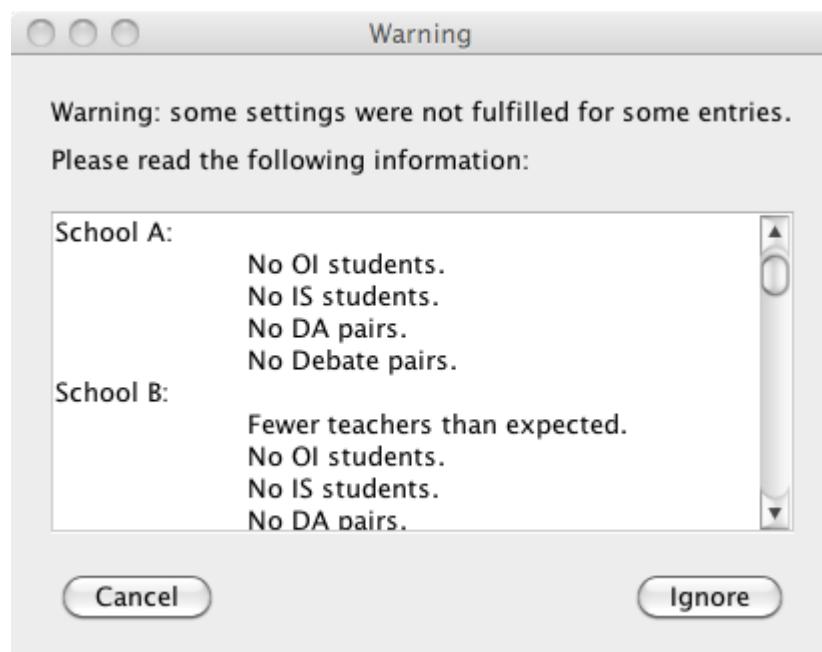
Abnormal:

Abnormal, in this context, is the situation when there seems to be enough judges for successful allocation, but due to the lack of variety (in the schools from which the judges are coming), it is impossible to actually do so.

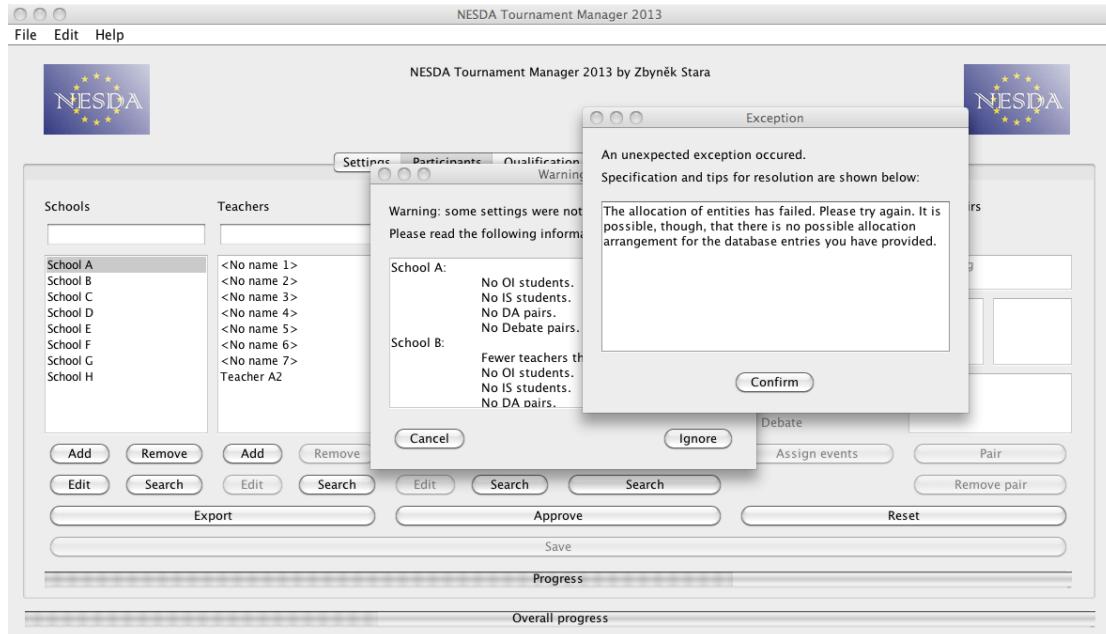


For this test, eight judges were provided, all in the School A, as shown.

If we try to allocate now, what will the result be? The warning dialog now shows that all schools except the School A have fewer teachers than expected:



If the ignore button is clicked, then, an exception dialog is shown informing the user that the allocation has failed.



The program cannot currently show any more specific error message, but the exception has been caught and processed without the program crashing, which is still good.

**Section D2:
Evaluating Solutions**

Effectiveness of the program:

Did the program work?

The core functionality of the program works. There are some glitches in the graphical user interface (for example, some room lists only update the first time they are selected), but they are minor in nature and do not affect the usability of the program, especially when there are other ways to access the data (through exporting). Apart from these, the program runs flawlessly.

Appropriateness of initial design:

During the coding of the program, it became clear that some of the original goals I have set as an extension to the core functionality of allocating judges and entities were unrealistic. I have dreamt too big.

For example, it became clear at one point during the coding that there will be not enough time to implement the finalist tab handling methods. Since they were not crucial for the allocation of judges and entities, I decided to focus on the allocation and set the finalists as a feature to implement in subsequent releases of the program.

Despite that, the program is still a success, even in its bare form without the additional functionality, because the most useful part of the program – the allocation of judges and entities – has stayed.

Moreover, thanks to the modularity of the program, there would be no problem with implementing this additional functionality in the future releases. (Visually unpleasant about this solution is the fact that some of the GUI elements are temporarily grayed out. But again, this has nothing to do with the actual function of the program.)

Addressing criteria for success:

1. Adding of entries (schools, teachers, students) to the database (SUCCESS)
 - a. The database is fully editable; new entries may be added, edited or removed as the user needs. Binary search trees are used to hold the entries to ensure efficient access to stored data.
 - b. Proper checks were put in place to ensure that the data is put in correctly.
 - c. The database is fully searchable; the user may find any entry in the database by its name (or, in the case of students, by their codes).
 - d. The graphical user interface is dynamically responding to user's actions (such as clicking buttons, selecting items in lists) with enabling/disabling appropriate buttons, thus leading the user to see what is possible to be done and what is not.
2. Assigning of student codes according to NESDA conventions (SUCCESS)
 - a. The students are given student codes automatically by the program.
 - b. These codes are ensured to be correct to NESDA standards, with the first part being the letter of the student's school and the second part the student's number.
3. Assigning of events for students (SUCCESS)
 - a. Events can be assigned to all students and the program responds appropriately to the changes in these.
 - b. Events are checked with the initial settings to be able to construct a list of missing or unexpected student event choices in the database. This adds another layer of security to the program, allowing the user to review the information before constructing qualification.

4. Adding of student pairs for Duet Acting and Debate events (SUCCESS)

- a. All students from a given school in one of the two events are listed side by side and are easily put together or separated.
- b. The program treats student pairs as unitary entities and is able to allocate them correctly to their rounds.

5. Exporting of the database (SUCCESS)

- a. The contents of the database can be saved as a plain-text file, allowing for the subsequent use on brochures, or in any other medium.

6. Saving the database (SUCCESS)

- a. The database can be saved as a file and transferred for use in other copy of the program or backed up, to ensure that information is not lost when the program closes.
- b. All participant tab trees, and all settings are written out into a text file, no information is left out, so that when the file is loaded back into the program, exactly the same information is available to the user.

7. Loading save files (SUCCESS)

- a. The program can take in save files and reconstruct the database. If the program was saved with the qualification, it is able to reconstruct the qualification as well.
- b. Error checking was put in place to inform the user that an error has occurred while loading the program, either that the file is not readable by the program, or that an I/O exception has occurred.

8. Constructing entities from students and student pairs (SUCCESS)

- a. Entities are the operands of the allocation system. They are one student's event participations – thus, one student is able to participate in multiple events.
- b. Judges are also constructed from the database teacher trees

9. Allocating entities into qualification rounds (SUCCESS)

- a. After the entities are approved, the program runs the allocation algorithm, which recursively tries the possible combinations of judges and student entities to find one which is correct to the NESDA rules (no judges from the same school, no entities from the same school, no judge seeing the same entity twice).
- b. The algorithm was made as efficient as possible, with the number of tries for every place in room capped to two, which allows for trying different options, but not too much as to make the process unbearably long

10. Allocation progress bar (SUCCESS)

- a. Since the allocation algorithm may require a lot of calculations, a progress bar is invoked while the calculation runs to inform the user of the current state of the calculation and give him/her a visual aid as to see how much more time will it take to allocate the entities in rooms.

11. Exporting of the qualification (SUCCESS)

- a. The qualification records of allocated events, rounds, and rooms with individual judges and entities can be exported as plaintext allowing for further processing by the organizers as needed (for programs, brochures, etc.)

12. Saving the qualification (SUCCESS)

- a. Like the database, qualification can also be saved by the program to a file. When such a save file is loaded, the qualification is recreated, with the exact same organization of judges and entities as the original file.

Did the program work for some data sets, but not for others?

During the testing phase, it was found that the allocation works correctly for the allocation of Original Oratory and Oral Interpretation entities. On the other hand, as the Oral Interpretation and Duet Acting require to be considered together as combined events, they require additional level of coordination the program is not able to offer.

Debate events are allocated correctly, but it is up to the organizers to set for each pair the affirmative or negative side in each debate. Supposedly, this could fall outside of the reach of this program, but since the program aspires to become the single comprehensive NESDA tournament organization utility, this issue would need to be addressed programmatically in the final release.

Does the program have any limitations?

The program requires at least 8 schools to be added to the database complete with teachers and students. Otherwise, there is not enough variety in the set of teachers and students provided, which means that the allocation criteria cannot be met (no judges/entities from one school encountering each other).

The program does not attempt to allocate an event in qualification if fewer students than the expected room number were set to be part of it. This makes sense, because that means that all the participants would continue to final regardless of their score.

With the allocation calculations, seemingly paradoxically, the program takes noticeably longer to allocate smaller data sets of judges and entities (as opposed to huge data sets) – for a database with 12 schools/3 teachers/4 students, the allocation lasts less than a second, while for 8 schools/2 teachers/3 students, the calculation time is consistently around 10 seconds. This is because small data sets have fewer successful combinations to solve the allocation problem, so more combinations have to be explored before a correct one is found.

At the current time, because of the way the allocation task is performed in the background by an AllocationWorker, the program does not inform the user if the allocation failed; the exceptions are not

passed from the function. Addressing this limitation would be one of the next steps to do with this program. However, it is a next step in the process.

Did the program work within the performance limits?

Regarding speed of the assigning of student codes, the program was successful, it seems instantaneous to the user.

Regarding the speed of allocation of judges and entities, the program is successful, its allocation takes a maximum of 10 seconds per event (and usually is much shorter), which fits into the 60 second total time threshold.

In terms of the memory limitations, the program is safely below the threshold of 2 GiB of RAM. On average, it uses 200 MiB, which is a tenth of the limit amount.

Improvements to the program:

Alternative approaches:

Thinking outside the box, this program could use a different technique for allocating judges and entities than the one currently used. Instead of trying to allocate judges and entities one by one, a graph traversal algorithm could be used. It would not explore the possibilities blindly one by one, but would use a metric to evaluate the probability that a given combination is going to lead to a successful end and use the combination that is the most desirable. However, unlike in pathfinding, there is no obvious metric to evaluate the probable success of a combination. Therefore, this new approach would need thorough consideration before being implemented.

Additional functionality:

In the subsequent versions of the program, additional functionality could be added, which would increase even more the value of the program to NESDA when organizing Speech and Debate tournaments.

These could include:

1. Independent allocation of IS/DA and Debate events
2. Substituting judges in qualification
3. Adding scores to entities
4. Automated announcing of the entities with the highest scores as finalists
5. Organizing the final rounds with collecting of judge rankings
6. Automated announcing of winners

It is important to note that while this functionality would make the program more central in the planning of the Speech and Debate tournaments, it is not needed for the crucial mission of the program, which is the allocation of entities. Unlike the allocation (which the program is able to do extremely fast), these other tasks can be performed with similar efficiency by humans.

Improvements to the process:

This program has been a second big computer science project in my life (first was my extended essay), and the two have shown me my strengths and limitations when getting things done. I have found that it works better for me to have several smaller tasks with clear deadlines than having a single big task. I tend to procrastinate if I do not feel under pressure; with the big task, there is a danger that I will not have enough time to finish because I start working on it later than I should. This is less of a problem with many small tasks. Therefore, my process would have been much improved if I were setting more clear deadlines for myself throughout the time allocated for finishing the program.

In addition, I have discovered that I tend to be quite idealistic about the number of features I would be able to implement in the program in the given time. Moreover, I often get sucked into solving an interesting but marginal problem in the program, robbing myself of time to address the necessary features. Next time I do a big programming project like this one, I will know that I must prioritize what needs to be done and carefully consider every additional feature I would want to add. And only go on to solving it if there is not any other more important task to implement. Because the interesting but marginal features can be added later, unlike the absolute essentials, which need to be done because without them the program would not work.

Some of the parts of the dossier process were very useful for me, like the initial UML, which I used as a guidance when I did not know what the next step in my programming should be. I think my biggest strength in such projects is the coding itself, with which I do not have problems, and my ability to debug my programs.

This project has certainly been an excellent insight into the real-world program development process. It has given me some good ideas of my strengths and areas for improvement, which will be very valuable for me in the future, at university and after. Thanks to the dossier, I have seen that as long as I stay on top of my tendency to overshoot in my aspirations, as long as I can prioritize what to do, and set close goals for myself to avoid procrastination, I can be extremely successful in computer science projects.

Documentation of Mastery Aspects

This dossier has made use of ten higher level mastery aspects. A list of these is included below:

1. Recursion:

- a. Recursion forms an absolute core part of the program because it is used in the allocation algorithm – the allocate() method in the QualificationEvent class ([page 334](#)). The reason a recursive solution to the problem was chosen is that there is a variable number of separate allocation tries for judges and entities. That means that an alternative solution (such as having nested loops) does not fall into question – because it is impossible to program a variable number of nested loops. With the recursive solution, each judge and entity is allocated on a new level of recursion. That means that each try is individual, but the changes can still propagate to the following recursion levels – if there was a success–, or to the previous ones, if there was a failure (by throwing an exception which is caught in the previous recursion level). The algorithm works as follows: First, the allocate() method is called to initialize a round ([page 334](#)). Then the allocate() method is called to try to allocate a judge ([page 335](#)). If the allocation was successful, the allocate() method is called to try to allocate another judge. This repeats for the following recursive levels until the last required judge is allocated, when the allocate() method is called to allocate the first entity ([page 336](#)). This repeats inside the successive recursion levels until the last entity is allocated, which means that in the following recursion level, a new round will be allocated. If all rounds are already allocated, this leads to the end of the allocate() method ([page 335](#)) and all the previous recursion levels finish. At any recursion level, allocation of judge or entity has failed two times, an exception is thrown, which signalizes to the previous allocation round that a reallocation is needed. If this happens for the very first allocated judge, it means that no allocation organization was found. (This algorithm is also discussed more closely in the section B2: Algorithms.)

2. Encapsulation:

- a. Encapsulation is used consistently throughout the whole program. All classes use private data types to prevent potentially disruptive editing to be done on the information stored in the variables. The only way to change these, then, is to use one of the set methods provided, which often have additional checks in place to ensure the data is correct. See the School class ([page 286](#)) for a primary example. Auxiliary examples can be seen in the Student class (page 310), or the Round class (page 339) for specific examples.

3. Parsing a Text File or Other Data Stream:

- a. The `readFile()` method ([page 162](#)) is an integral part of the process of loading files and virtually all of its activity is parsing text files – the program's own save files – and converting the information therein into values of attributes and re-constructing trees of information. This is done by applying a StringTokenizer separately on every line and taking the information from there as appropriate by the situation. The reason one overarching StringTokenizer is not used for this is error prevention – the overarching StringTokenizer would not be able to detect erroneous or missing data. The program's current implementation enables checking the data before it is processed, thus adding another layer of security.

4. Implementing a Hierarchical Composite Data Structure:

- a. There are in fact two hierarchical data structures in the program – the database and qualification. In both cases, they were created to mimic the real-life organization of the Speech and Debate tournaments.
- b. The Database – primary example – ([pages 273 and 278](#)) contains a BinarySearchTree of Schools. Schools, in turn, contain Teacher and Student trees. As well as mimicking the real-life organization of the tournament, the structure of the program also allows for a precise identification of a student and teacher, which is important for the program's functionality.

- c. The second hierarchical structure (auxiliary example) is the Qualification (page 320) hierarchy. In the qualification, there is an array of Events. Every event has an array of rounds, and every round has an array of rooms. Finally, in rooms, there are judges and entities. This, again, reflects the real-life organization of the tournaments, and is therefore the structure the users would expect to see in the program.

5. The Use of Any Five Standard Level Mastery Factors:

- a. Arrays are widely used throughout the program. As a primary example, Events are stored in Qualification in an array ([page 320](#)). Auxiliary examples are the array of Rounds in Events (page 329) and Rooms in Rounds ([page 339](#)).
- b. The program would not be possible without user-defined objects. An example is the Teacher class, of which an instance is made at [page 290](#). (The class is declared at page 308.)
- c. Simple selection (if-else) is a commonly used technique in the program. An example, used for error handling, can be seen at [page 323](#).
- d. Similarly, complex selection (a switch block) is occasionally used where it is appropriate – an example of that is the allocate() method, where a switch is used to differentiate approach to different eventTypes (original oratory to debate) ([page 334](#)).
- e. Searching is used extensively in the Database – primary example is the searching for a school ([page 280](#)). Auxiliary examples are searching for teachers (page 280) or students (page 282).

6. (4 aspects) Implementation of Abstract Data Types (ADTs):

- a. For this mastery aspect, a binary search tree was implemented (class BinarySearchTree) ([page 265](#)). At many points in my program, it is not known in advance how many entries there will be in a structure (schoolTree, studentTree, teacherTree...), but at the same time, the structure has to be easily searchable. I used a binary search tree in these cases because it has all of these desired qualities. Other data structures are not suitable – arrays and hash tables are limited in

length, and linked lists are not (unlike binary search trees) automatically sorted and thus they are not easily searchable.

b. Aspect 1 (6th in total):

- i. The BinarySearchTree has an appropriate constructor (page 266).
- ii. The functions getNodeData() (page 270), getDataArray() (page 270) or the private getNodeArray() (page 271) make it possible to get data elements.
- iii. The method setNodeData() (page 265) allows setting data elements.

c. Aspect 2 (7th in total):

- i. The method insert() (page 266) adds to the correct point in the tree. The correct point is determined alphabetically, according to the key strings of the data in the tree.
- ii. The method delete() (page 268) deletes from the correct point in the tree, and ensures that the tree is re-linked after.

d. Aspect 3 (8th in total):

- i. Proper checks were put into place to ensure that errors are detected and reported. (Refer to section C2: Error Handling for more detailed information.)
- ii. An IllegalArgumentException is thrown when an attempt is made to insert a node with a key that is already in the tree (page 266).
- iii. A NoSuchElementException is thrown when an attempt is made to delete or search for a node that is not in the tree (page 268).

e. Aspect 4 (9th in total):

- i. All possible error conditions are checked for. (Refer to section C2: Error Handling.)
- ii. All appropriate methods are implemented. These include:
 - o size() – returning the number of elements in the tree (page 270)
 - o isEmpty() – checking whether the tree is empty (page 270)

- contains() – checking whether the tree contains a key
(page 267)
- minNode() – finding the lowest node from a given node
(page 269)
- balance() – re-linking of tree so that it is most
efficiently organized (page 271)

10. Use of Additional Libraries:

- a. The most prominent use of additional libraries can be seen in the QualificationEvent class. The class AllocationWorker ([page 332](#)), an extension of the javax.swing.SwingWorker<V, T>, is used to facilitate the execution of the computation-intensive allocation algorithm in the background. The reason for using this additional library is that it allows for an easy handling of threads. Otherwise, I would have to do that myself, and I would not have time for that.

**Section C1:
Code**

Note:

Yellow highlighting shows code that was generated by the IDE.

MainGUI.java

```

  /**
   * NESDA Tournament Manager 2013
   *
   * By Zbyněk Stara, 000889-045
   * International School of Prague
   * Czech Republic
   *
  10  * Computer used:
   * Macbook unibody aluminum late 2008
   * Mac OS X 10.6.8
   * 8 GiB of RAM
   *
  15  * IDE used:
   * NetBeans 6.9.1
   */
  20 /**
   * Purpose:
   * The purpose of this program is to help the organizers of the NESDA Speech and
   * Debate tournaments with allocation of participants and judges to different
   * events, rounds and room. The allocation algorithm uses a user-filled database
   * of participating schools, teachers and students.
  25 */
  30 // ACCOMPLISHED The GUI front end is done
  // ACCOMPLISHED Code for several GUI buttons
  // ACCOMPLISHED Basic functions for school
  // ACCOMPLISHED Database functions almost completely written
  // ACCOMPLISHED Database functions annotated with javadoc
  // ACCOMPLISHED Resetting and approving of settings
  // ACCOMPLISHED Information dialogs
  // ACCOMPLISHED Wrap up the settingsApproveButton code
  // ACCOMPLISHED JFileChooserDialog for saving
  // ACCOMPLISHED Exception dialog
  // ACCOMPLIAHED Saving of settings
  // ACCOMPLISHED Loading of settinds
  // ACCOMPLISHED Tree-balancing algorithm to be used before saving of trees
  40
  // CHECK-IN Nov 27, 2012:
  // ACCOMPLISHED getIndex method for trees for selecting searched/added/edited elements
  // in lists
  // ACCOMPLISHED participants tab: schools buttons work completely
  45
  // CHECK-IN Dec 5, 2012:
  // ACCOMPLISHED getIndex and insert methods for teachers and students
  // ACCOMPLISHED teachersAddButton code adapted from schoolsAddButton's code
  50 // ACCOMPLISHED Change school's teacher and student methods to correspond with
  // database's (Jan-4-2013)
  // ACCOMPLISHED Change database globalSearchTeacher methods to use school's
  // searchTeacher
  // ACCOMPLISHED Do the same for globalSearchStudent
  // ACCOMPLISHED Global searching works for teachers and students
  // ACCOMPLISHED StudentCode elements respond properly to other element changes
  // ACCOMPLISHED StudentCode is entirely functional
  // ACCOMPLISHED Clean up the MainGUI code (move Database-specific tasks and fix
  // errors)
  60 // ACCOMPLISHED Show which events are selected for which student
  // ACCOMPLISHED Assign events button is only available when events were changed (Feb-
  // 14-2013)
  65 // ACCOMPLISHED All buttons from participants tab now respond, except the technical
  // ones (FE-25-2013)
  // ACCOMPLISHED Make pairs section respond to removal of students
  // ACCOMPLISHED Make Pairs respond to the changing of events of paired students
  70 // ACCOMPLISHED Working with the participants database is now bug-free

```

```

// ACCOMPLISHED Allocation algorithm is done
// ACCOMPLISHED Saving, loading and exporting works

75 // TODO List updates in Qualification

    // OLD:
    // TODO Move database variables (studentsNumber etc.) from MainGUI
    // TODO Edit assigning and searching student codes in Database
80 // TODO Confirmation dialog when removing a school with teachers and/or students
    // TODO Move on to School, Student, and Teacher functions
    // TODO Make a robust system for assigning noName numbers, their reuse, and saving - using dynamic queues?
    // TODO Make a robust system for student code assignment - keep track of what is added, and if it subsequently removed, do not require re-assignment
85 // TODO Rework the trees' handling of indices (now, they have to be converted to arrays to find them; what's the point of having trees, then?)
        //         use link-list approach for nodes to link them by indices
        //         re-work the getIndex method to work with this: otherwise, its work could
90 be done with the search function
        //         re-write the search index not to rely on the arrays to make use of the full use of the trees' search potential
    // TODO Check the mainGUI methods for duplicacies in enabling/disabling codes and maybe create methods to group things that always go together

95 package gui;

import data.*;
import javax.swing.*;
100 import java.io.*;
import java.util.*;
import java.text.*;

/**
 * MainGUI is the main class of the program. The class controls the graphical user interface. It contains listeners to the events triggered by user's interaction with the program.
 *
 * @author Zbyněk Stara
 * @since Jan-28-2012
 */
public class MainGUI extends javax.swing.JFrame {
    // CUSTOM EXCEPTIONS:
    /**
     * The UserIOException is sent when an input/output operation fails because the process is canceled by the user. It extends IOException.
     */
    private class UserIOException extends IOException {
        private void UserIOException() {
            throw new RuntimeException();
        }
    }

    /**
     * The FileIOException is sent when an input/output operation fails because the file selected by the user is invalid. It extends IOException.
     */
    private class FileIOException extends IOException {
        private void FileIOException() {
            throw new RuntimeException();
        }
    }

    // ATTRIBUTES:
    private Database database = new Database(); // contains data of schools, teachers and students to be used by the program, and methods to manipulate the data
    private Qualification qualification; // contains data of events, rounds and rooms, which hold the allocated entities and judges, and methods to manipulate them

135    // LIST MODELS:
    private DefaultListModel blankModel = new DefaultListModel(); // a blank list model
    private DefaultListModel emptyModel = new DefaultListModel(); // a list model reading <No elements>, used to show that the displayed list is empty
        private DefaultListModel noSelectionModel = new DefaultListModel(); // a list model reading <No selection>, used to show that the list requires a selection in other list higher in data hierarchy

```

```

150     private DefaultListModel eventsListModel = new DefaultListModel(); // a list model
featuring the events for participantsEventsList (Duet Acting and Debate)

155     // SAVING/LOADING/RESETTING VARIABLES:
private File loadFile = null; // the last used load file (used for resetting)
private int approvedSectionsIndex = 0; // the last section to be saved/loaded

160     // NO NAME VARIABLES:
private int currentNewNoNameSchoolNumber = 1; // number of the new no name school
private int currentNewNoNameTeacherNumber = 1; // number of the new no name
teacher
private int currentNewNoNameStudentNumber = 1; // number of the new no name
student

165     // NUMBERS OF DATABASE ENTRIES:
private int schoolsNumber = 0; // current number of schools in the database
private int teachersNumber = 0; // current number of teachers in the database
private int studentsNumber = 0; // current number of students in the database

170     // GUI VARIABLES:
private boolean reassignmentText = false; // indicates whether the text on the
assign codes button should read "Re-assign codes"
private boolean studentCodes = false; // indicates whether there is a student code
assignment effective at the current moment
private boolean studentChanges = false; // indicates whether a new student codes
assignment is needed (schools/teachers/students added, removed, or edited)

175     // PARTICIPANTS LIST SELECTION VARIABLES:
private int schoolsIndex = -1; // index of the school currently selected in the
schoolsList
private int teachersIndex = -1; // index of the teacher currently selected in the
teachersList
private int studentsIndex = -1; // index of the student currently selected in the
studentsList
private int studentCodesIndex = -1; // index of the student code currently
selected in the studentCodesList

180     // ASSIGN PAIRS LIST SELECTION VARAIBLES:
private int assignPairsEventsIndex = -1; // index of the event currently selected
in the assignPairsEventsList
private int assignPairsLeftIndex = -1; // index of the student currently selected
in the assignPairsLeftList
private int assignPairsRightIndex = -1; // index of the student currently selected
in the assignPairsRightList
private int assignPairsPairsIndex = -1; // index of the student pair currently
selected in the assignPairsPairsList

185     // QUALIFICATION LIST SELECTION VARIABLES:
private int eventsIndex = -1; // index of the event currently selected in the
eventsList
private int roundsIndex = -1; // index of the round currently selected in the
roundsList
private int roomsIndex = -1; // index of the room currently selected in the
roomList
private int judgesIndex = -1; // index of the judge currently selected in the
judgesList
private int qStudentCodesIndex = -1; // index of the entity currently selected in
the (qualification) studentCodesList

190     // INITIALIZATION:
210 public MainGUI() {
    initComponents();
    myInitComponents();
}

215 private void myInitComponents() {
    emptyModel.addElement("<No elements>");
    noSelectionModel.addElement("<No selection>");

    eventsListModel.addElement("Duet Acting");
    eventsListModel.addElement("Debate");

220     // PARTICIPANTS TAB INITIALIZATION
    participantsSchoolsList.setModel(emptyModel);

    participantsTeachersList.setModel(noSelectionModel);
    participantsStudentsList.setModel(noSelectionModel);
    participantsStudentCodesList.setModel(noSelectionModel);
}

```

```

230 // SETTINGS METHODS:
231 /**
232 * The getTextFieldValue method returns the value of a selected settings
233 * text field.
234 * <p>
235 * The method attempts to parse the text contents of a text field as an
236 * integer. If it succeeds, it returns that integer. If the text contents
237 * cannot be parsed as an int, the value of -1 is returned instead.
238 *
239 * @param textField the textField whose value we are taking in
240 *
241 * @return an int with the numerical value of the specified settings text
242 * field
243 *
244 * @author Zbyněk Stara
245 * @version 1.0 (Nov-18-2012)
246 * @since Nov-18-2012
247 */
248 private int getTextFieldValue(javax.swing.JTextField textField) {
249     String tempText = textField.getText();
250     int value;
251     try {
252         value = Integer.parseInt(tempText);
253     } catch (NumberFormatException ex) {
254         value = -1;
255     }
256     return value;
257 }
258 /**
259 * The changeDatabaseSettings method replaces the values of database
260 * settings variables with the values of settings text fields.
261 * <p>
262 * The values of the settings variables are replaced by the values returned
263 * by the getTextFieldValue function for each of the settings text fields.
264 * This means that if there is an invalid value entered into a text field,
265 * it is propagated into the database variables as -1. The values of the
266 * settings variables are set by a call to Database's setSettings method.
267 *
268 * @param database a Database object
269 *
270 * @throws IllegalArgumentException if the array internally constructed to
271 * hold the values of text fields does not have appropriate length for as
272 * required by the Database.setSettings method.
273 *
274 * @author Zbyněk Stara
275 * @version 1.0 (Nov-18-2012)
276 * @since Nov-18-2012
277 *
278 * @see data.Database.setSettings()
279 */
280 private void changeDatabaseSettings(Database database) throws
281 IllegalArgumentException {
282     Object[] valueArray = new Object[29];
283
284     valueArray[0] = getTextFieldValue(eventSettingsOOSR);
285     valueArray[1] = getTextFieldValue(eventSettingsOISR);
286     valueArray[2] = getTextFieldValue(eventSettingsISSR);
287     valueArray[3] = getTextFieldValue(eventSettingsDASR);
288     valueArray[4] = getTextFieldValue(eventSettingsOOFinalSR);
289     valueArray[5] = getTextFieldValue(eventSettingsOIFinalSR);
290     valueArray[6] = getTextFieldValue(eventSettingsISFinalSR);
291     valueArray[7] = getTextFieldValue(eventSettingsDAFFinalSR);
292
293     valueArray[8] = getTextFieldValue(eventSettingsOOSES);
294     valueArray[9] = getTextFieldValue(eventSettingsOISES);
295     valueArray[10] = getTextFieldValue(eventSettingsISSES);
296     valueArray[11] = getTextFieldValue(eventSettingsDASES);
297     valueArray[12] = getTextFieldValue(eventSettingsDebateSES);
298
299     valueArray[13] = getTextFieldValue(eventSettingsOOJR);
300     valueArray[14] = getTextFieldValue(eventSettingsOIJR);
301     valueArray[15] = getTextFieldValue(eventSettingsISJR);
302     valueArray[16] = getTextFieldValue(eventSettingsDAJR);
303     valueArray[17] = getTextFieldValue(eventSettingsDebateJR);
304     valueArray[18] = getTextFieldValue(eventSettingsFinalsJR);
305
306     valueArray[19] = getTextFieldValue(generalSettingsRoomsAvailable1);

```

```

310     valueArray[20] = getTextFieldValue(generalSettingsRoomsAvailable2);
     valueArray[21] = getTextFieldValue(generalSettingsMaximalTime1);
     valueArray[22] = getTextFieldValue(generalSettingsMaximalTime2);
     valueArray[23] = getTextFieldValue(generalSettingsSchoolsNumber);
     valueArray[24] = getTextFieldValue(generalSettingsTS);
     valueArray[25] = getTextFieldValue(generalSettingsSS);

315     valueArray[26] = generalSettingsCECheckBox.isSelected();
     valueArray[27] = generalSettingsCEComboBox1.getSelectedIndex();
     valueArray[28] = generalSettingsCEComboBox2.getSelectedIndex();

320     try {
         database.setSettings(valueArray);
     } catch (IllegalArgumentException ex) {
         throw new IllegalArgumentException();
     }
 }

325 /**
 * The resetDatabaseSettings method resets the values of the database's
 * settings variables to their default values.
 */
330 * This method calls the Database.resetSettings method.
 *
 * @param database a Database object
 *
335 * @author Zbyněk Stara
 * @version 1.0 (Nov-18-2012)
 * @since Nov-18-2012
 *
 * @see data.Database.resetSettings()
 */
340 private void resetDatabaseSettings(Database database) {
    database.resetSettings();
}

345 /**
 * The approveDatabaseSettings method checks the correctness of database
 * settings values
 */
350 * This method utilizes a boolean array from Database.checkSettings method
 * to output a textual evaluation of the correctness of the value of a
 * specific setting.
 *
 * @param database a Database object
 *
355 * @return String with the settings which have illegal values; may be empty
 * if all values were correct
 *
 * @author Zbyněk Stara
 * @version 1.0 (Nov-18-2012)
 * @since Nov-18-2012
 */
360 * @see data.Database.checkSettings()
 */
365 private String approveDatabaseSettings(Database database) {
    boolean[] checkArray = database.checkSettings();

    String errorString = "";
    String returnString;

    errorString += checkArray[0] ? "" : "Illegal value: OO students/room limit\n";
370 // if the variable has a legal value, return true, else return false
    errorString += checkArray[1] ? "" : "Illegal value: OI students/room limit\n";
    errorString += checkArray[2] ? "" : "Illegal value: IS students/room limit\n";
    errorString += checkArray[3] ? "" : "Illegal value: DA students/room limit\n";
    errorString += checkArray[4] ? "" : "Illegal value: OO finals students/room
limit\n";
375     errorString += checkArray[5] ? "" : "Illegal value: OI finals students/room
limit\n";
     errorString += checkArray[6] ? "" : "Illegal value: IS finals students/room
limit\n";
380     errorString += checkArray[7] ? "" : "Illegal value: DA finals students/room
limit\n";

    errorString += checkArray[8] ? "" : "Illegal value: OO students/event/school
limit\n";

```

```

385     errorString += checkArray[9] ? "" : "Illegal value: OI students/event/school
limit\n";
        errorString += checkArray[10] ? "" : "Illegal value: IS students/event/school
limit\n";
        errorString += checkArray[11] ? "" : "Illegal value: DA students/event/school
limit\n";
        errorString += checkArray[12] ? "" : "Illegal value: Debate
student/event/school limit\n";

395     errorString += checkArray[13] ? "" : "Illegal value: OO judges/room limit\n";
        errorString += checkArray[14] ? "" : "Illegal value: OI judges/room limit\n";
        errorString += checkArray[15] ? "" : "Illegal value: IS judges/room limit\n";
        errorString += checkArray[16] ? "" : "Illegal value: DA judges/room limit\n";
        errorString += checkArray[17] ? "" : "Illegal value: Debate judges/room
limit\n";
400     errorString += checkArray[18] ? "" : "Illegal value: Finals judges/room
limit\n";

        errorString += checkArray[19] ? "" : "Illegal value: Rooms available day 1\n";
        errorString += checkArray[20] ? "" : "Illegal value: Rooms available day 2\n";
        errorString += checkArray[21] ? "" : "Illegal value: Tournament time day 1\n";
        errorString += checkArray[22] ? "" : "Illegal value: Tournament time day 2\n";
        errorString += checkArray[23] ? "" : "Illegal value: Schools in tournament\n";
        errorString += checkArray[24] ? "" : "Illegal value: Teachers/school\n";
        errorString += checkArray[25] ? "" : "Illegal value: Students/school\n";
410
        // checkArray[26] can only be true

        errorString += checkArray[27] ? "" : "Illegal value: Combined events 1\n";
        errorString += checkArray[28] ? "" : "Illegal value: Combined events 2\n";
415
        if (generalSettingsCEComboBox1.getSelectedIndex() ==
generalSettingsCEComboBox2.getSelectedIndex()) {
            errorString += "Illegal input: Combined events combo boxes both report the
same event type\n";
420
        }

        if (!errorString.equals("")) {
            returnString = errorString.substring(0, (errorString.length() - 1));
425
        } else {
            returnString = errorString;
        }

        return returnString;
    }
430
    /**
     * The synchronizeTFSettings method correlates the text field values in the
     * program with database variable values.
     * <p>
     * This method takes the values of the database settings variables and puts
     * them into the appropriate settings text fields. In a sense, then, this
     * method is the opposite of the changeDatabaseSettings method, which takes
     * the text-field values and translates them into database variable values.
     *
440     * @param database a Database object
     *
     * @author Zbyněk Stara
     * @version 1.0 (Nov-18-2012)
     * @since Nov-18-2012
     *
445     * @see gui.MainGUI.changeDatabaseSettings
     */
    private void synchronizeTFSettings(Database database) {
450
        Object[] settingsArray = database.getSettings();

        eventSettingsOOSR.setText((Integer) settingsArray[0] + "");
        eventSettingsOISR.setText((Integer) settingsArray[1] + "");
        eventSettingsISSR.setText((Integer) settingsArray[2] + "");
        eventSettingsDASR.setText((Integer) settingsArray[3] + "");
455
        eventSettingsOOFinalSR.setText((Integer) settingsArray[4] + "");
        eventSettingsOIFinalSR.setText((Integer) settingsArray[5] + "");
        eventSettingsISFinalSR.setText((Integer) settingsArray[6] + "");
        eventSettingsDAFinalSR.setText((Integer) settingsArray[7] + "");

460
        eventSettingsOOSSES.setText((Integer) settingsArray[8] + "");
        eventSettingsOISES.setText((Integer) settingsArray[9] + "");
        eventSettingsISSES.setText((Integer) settingsArray[10] + "");
        eventSettingsDASES.setText((Integer) settingsArray[11] + ");

```

```

465     eventSettingsDebateSES.setText((Integer) settingsArray[12] + "");
466     eventSettingsOOJR.setText((Integer) settingsArray[13] + "");
467     eventSettingsOIJR.setText((Integer) settingsArray[14] + "");
468     eventSettingsISJR.setText((Integer) settingsArray[15] + "");
469     eventSettingsDAJR.setText((Integer) settingsArray[16] + "");
470     eventSettingsDebateJR.setText((Integer) settingsArray[17] + "");
471     eventSettingsFinalsJR.setText((Integer) settingsArray[18] + "");

475     generalSettingsRoomsAvailable1.setText((Integer) settingsArray[19] + "");
476     generalSettingsRoomsAvailable2.setText((Integer) settingsArray[20] + "");
477     generalSettingsMaximalTime1.setText((Integer) settingsArray[21] + "");
478     generalSettingsMaximalTime2.setText((Integer) settingsArray[22] + "");
479     generalSettingsSchoolsNumber.setText((Integer) settingsArray[23] + "");
480     generalSettingsTS.setText((Integer) settingsArray[24] + "");
481     generalSettingsSS.setText((Integer) settingsArray[25] + "");

485 }
486
487 // PARTICIPANTS METHODS:
488 /**
489 * The approveParticipants method checks the database and reports to the
490 * user any unexpected values.
491 *
492 * @param database the current database
493 * @return a string report with all the warnings to the user
494 *
495 * @author Zbyněk Stara
496 */
497 private String approveParticipants(Database database) { // initializes
qualification
500     qualification = new Qualification(database);
501
502     Object[] settingsArray = database.getSettings();
503     /*settingsArray[0] = ooStudentRoomLimit;
504     settingsArray[1] = oiStudentRoomLimit;
505     settingsArray[2] = isStudentRoomLimit;
506     settingsArray[3] = daStudentRoomLimit;
507     settingsArray[4] = ooFinalStudentRoomLimit;
508     settingsArray[5] = oiFinalStudentRoomLimit;
509     settingsArray[6] = isFinalStudentRoomLimit;
510     settingsArray[7] = daFinalStudentRoomLimit;
511
512     settingsArray[8] = ooStudentSchoolLimit; // THIS
513     settingsArray[9] = oiStudentSchoolLimit; // THIS
514     settingsArray[10] = isStudentSchoolLimit; // THIS
515     settingsArray[11] = daStudentSchoolLimit; // THIS
516     settingsArray[12] = debateStudentSchoolLimit; // THIS
517
518     settingsArray[13] = ooJudgeRoomLimit;
519     settingsArray[14] = oiJudgeRoomLimit;
520     settingsArray[15] = isJudgeRoomLimit;
521     settingsArray[16] = daJudgeRoomLimit;
522     settingsArray[17] = debateJudgeRoomLimit;
523     settingsArray[18] = finalsJudgeRoomLimit;
524
525     settingsArray[19] = roomLimit1;
526     settingsArray[20] = roomLimit2;
527     settingsArray[21] = timeLimit1;
528     settingsArray[22] = timeLimit2;
529     settingsArray[23] = schoolNumber; // THIS
530     settingsArray[24] = teacherSchoolNumber; // THIS
531     settingsArray[25] = studentSchoolNumber; // THIS
532
533     settingsArray[26] = allowCombinedEvents;
534
535     settingsArray[27] = combinedEvent1;
536     settingsArray[28] = combinedEvent2;/*
537
538     String returnString = "";
539
540     if (database.getSchoolTreeSize() < (Integer) settingsArray[23]) {
541         returnString += "Fewer schools than expected.\n";
542     } else if (database.getSchoolTreeSize() > (Integer) settingsArray[23]) {
543         returnString += "More schools than expected.\n";
544     }

```

```

    }

545

    for (int i = 0; i < database.getSchoolTreeSize(); i++) {
        School currentSchool = (School) database.getSchoolTreeNodeData(i);
        returnString += this.approveSchool(database, currentSchool);
    }

    // additional judges calculations
    String judgesIntroString = "Not enough judges:\n";
    Boolean judgesIntroShown = false;

    int judgesTotal = qualification.getJudgeTreeSize();

550

    int ooEntitiesTotal = qualification.getOOEntityTreeSize();
    int ooRoomTotal = (int) Math.ceil(((double) ooEntitiesTotal) / ((double)
        ((Integer) settingsArray[0])));
    int ooJudgesRequired = ooRoomTotal * ((Integer) settingsArray[13]);

    int oiEntitiesTotal = qualification.getOIEntityTreeSize();
    int oiRoomTotal = (int) Math.ceil(((double) oiEntitiesTotal) / ((double)
        ((Integer) settingsArray[1])));
    int oiJudgesRequired = oiRoomTotal * ((Integer) settingsArray[14]);

555

    int isEntitiesTotal = qualification.getISEntityTreeSize();
    int isRoomTotal = (int) Math.ceil(((double) isEntitiesTotal) / ((double)
        ((Integer) settingsArray[2])));
    int isJudgesRequired = isRoomTotal * ((Integer) settingsArray[15]);

    int daEntitiesTotal = qualification.getDAEntityTreeSize();
    int daRoomTotal = (int) Math.ceil(((double) daEntitiesTotal) / ((double)
        ((Integer) settingsArray[3])));
    int daJudgesRequired = daRoomTotal * ((Integer) settingsArray[16]);

560

    int debateEntitiesTotal = qualification.getDebateEntityTreeSize();
    int debateRoomTotal = (int) Math.ceil(((double) debateEntitiesTotal) /
        ((double) ((Integer) settingsArray[4])));
    int debateJudgesRequired = debateRoomTotal * ((Integer) settingsArray[17]);

565

    if (ooJudgesRequired > judgesTotal) {
        if (!judgesIntroShown) {
            returnString += judgesIntroString;
            judgesIntroShown = true;
        }
        int ooDifference = ooJudgesRequired - judgesTotal;
        returnString += ooDifference + " more teachers needed to judge OO.\n";
    }

570

    if (oiJudgesRequired > judgesTotal) {
        if (!judgesIntroShown) {
            returnString += judgesIntroString;
            judgesIntroShown = true;
        }
        int oiDifference = oiJudgesRequired - judgesTotal;
        returnString += oiDifference + " more teachers needed to judge OI.\n";
    }

575

    if ((isJudgesRequired + daJudgesRequired) > judgesTotal) {
        if (!judgesIntroShown) {
            returnString += judgesIntroString;
            judgesIntroShown = true;
        }
        int isdaDifference = (isJudgesRequired + daJudgesRequired) - judgesTotal;
        returnString += isdaDifference + " more teachers needed to judge combined
IS/DA.\n";
    }

580

    if (debateJudgesRequired > judgesTotal) {
        if (!judgesIntroShown) {
            returnString += judgesIntroString;
            judgesIntroShown = true;
        }
        int debateDifference = debateJudgesRequired - judgesTotal;
        returnString += debateDifference + " more teachers needed to judge
Debate.\n";
    }
}

```

```

        return returnString;
    }

625 /**
 * The approveSchool method is used by approveParticipants to check and
 * report warnings about a given school.
 *
630 * @param database the current database
 * @param school the school to be checked
 * @return a string with the warnings about the given school
 *
635 * @author Zbyněk Stara
 */
635 private String approveSchool(Database database, School school) {
    Object[] settingsArray = database.getSettings();
    /*settingsArray[0] = ooStudentRoomLimit;
    settingsArray[1] = oiStudentRoomLimit;
    settingsArray[2] = isStudentRoomLimit;
    settingsArray[3] = daStudentRoomLimit;
    settingsArray[4] = ooFinalStudentRoomLimit;
    settingsArray[5] = oiFinalStudentRoomLimit;
    settingsArray[6] = isFinalStudentRoomLimit;
    settingsArray[7] = daFinalStudentRoomLimit;

645     settingsArray[8] = ooStudentSchoolLimit; // THIS
    settingsArray[9] = oiStudentSchoolLimit; // THIS
    settingsArray[10] = isStudentSchoolLimit; // THIS
    settingsArray[11] = daStudentSchoolLimit; // THIS
    settingsArray[12] = debateStudentSchoolLimit; // THIS

655     settingsArray[13] = ooJudgeRoomLimit;
    settingsArray[14] = oiJudgeRoomLimit;
    settingsArray[15] = isJudgeRoomLimit;
    settingsArray[16] = daJudgeRoomLimit;
    settingsArray[17] = debateJudgeRoomLimit;
    settingsArray[18] = finalsJudgeRoomLimit;

660     settingsArray[19] = roomLimit1;
    settingsArray[20] = roomLimit2;
    settingsArray[21] = timeLimit1;
    settingsArray[22] = timeLimit2;
    settingsArray[23] = schoolNumber; // THIS
    settingsArray[24] = teacherSchoolNumber; // THIS
    settingsArray[25] = studentSchoolNumber; // THIS

665     settingsArray[26] = allowCombinedEvents;

670     settingsArray[27] = combinedEvent1;
    settingsArray[28] = combinedEvent2;*/

    String returnString = "";
    String firstString = school.getName() + ":\n";
    boolean somethingReturned = false;

675     if (school.getTeacherTreeSize() < (Integer) settingsArray[24]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tFewer teachers than expected.\n";
    } else if (school.getTeacherTreeSize() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tNo teachers.\n";
    }

690     if (school.getStudentTreeSize() > (Integer) settingsArray[25]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tMore students than expected.\n";
    } else if (school.getStudentTreeSize() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
    }
}

```

```

        returnString += "\tNo students.\n";
    }

705   if (school.getOOStudentNumber() > (Integer) settingsArray[8]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tMore OO students than expected.\n";
    } else if (school.getOOStudentNumber() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tNo OO students.\n";
    }

715   }

720   if (school.getOIStudentNumber() > (Integer) settingsArray[9]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tMore OI students than expected.\n";
    } else if (school.getOIStudentNumber() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tNo OI students.\n";
    }

730   }

735   if (school.getISStudentNumber() > (Integer) settingsArray[10]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tMore IS students than expected.\n";
    } else if (school.getISStudentNumber() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tNo IS students.\n";
    }

740   }

745   if (school.getDAPairTreeSize() > (Integer) settingsArray[11]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tMore DA pairs than expected.\n";
    } else if (school.getDAPairTreeSize() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tNo DA pairs.\n";
    }

750   }

755   if (school.getDebatePairTreeSize() > (Integer) settingsArray[12]) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tMore Debate pairs than expected.\n";
    } else if (school.getDebatePairTreeSize() == 0) {
        if (!somethingReturned) {
            somethingReturned = true;
            returnString += firstString;
        }
        returnString += "\tNo Debate pairs.\n";
    }

760   }

765   }

770   }

775   }

        return returnString;
    }

// SAVING/LOADING METHODS:
/***
 * The loadFile method invokes a file chooser dialog, prompting the user to

```

```

780     * select a file to be loaded. Then, it calls the readFile method to parse
781     * the file contents.
782     *
783     * @throws UserIOException if the file chooser dialog is canceled by the
784     * user
785     * @throws FileIOException if the file format does not correspond to what
786     * the program is expecting
787     * @throws IOException if an error is encountered during the reading of
788     * data from the file, or the save dialog returned the ERROR_OPTION
789     *
790     * @author Zbyněk Stara
791     */
792     private void loadFile() throws UserIOException, FileIOException, IOException {
793         JFileChooser jfc = new JFileChooser();
794
795         int loadDialogReturn = jfc.showOpenDialog(this);
796
797         if (loadDialogReturn == JFileChooser.APPROVE_OPTION) {
798             loadFile = jfc.getSelectedFile();
799             readFile(loadFile);
800         } else if (loadDialogReturn == JFileChooser.CANCEL_OPTION) {
801             throw new UserIOException();
802         } else {
803             throw new IOException();
804         }
805     }
806
807     /**
808      * The readFile method tries to parse contents of a provided file. A certain
809      * file format is expected; if the file provided is not in that format, an
810      * exception is thrown.
811      *
812      * @param readFile the file to be read in
813      * @throws FileIOException if the file cannot be parsed
814      * @throws IOException if another unspecified input/output exception has
815      * occurred during the parsing of the file
816      *
817      * @author Zbyněk Stara
818      */
819     private void readFile(File readFile) throws FileIOException, IOException {
820         BufferedReader br = new BufferedReader(new FileReader(readFile));
821
822         String currentLine = "";
823
824         // s<editor-fold defaultstate="collapsed" desc="Introduction">
825         currentLine = br.readLine();
826         if (currentLine == null || !currentLine.equals("{NESDA Tournament Manager Save
827             File}")) {
828             throw new FileIOException();
829         }
830
831         if (br.readLine() == null) {
832             throw new FileIOException();
833         }
834         if (br.readLine() == null) {
835             throw new FileIOException();
836         }
837         if (br.readLine() == null) {
838             throw new FileIOException();
839         }
840         if (br.readLine() == null) {
841             throw new FileIOException();
842         }
843         if (br.readLine() == null) {
844             throw new FileIOException();
845         }
846         if (br.readLine() == null) {
847             throw new FileIOException();
848         }
849         if (br.readLine() == null) {
850             throw new FileIOException();
851         }
852         // </editor-fold>
853
854         // <editor-fold defaultstate="collapsed" desc="$Attributes">
855         currentLine = br.readLine();
856         if (currentLine == null || !currentLine.equals("$Attributes")) {
857             throw new FileIOException();
858         }

```

```

860     currentLine = br.readLine();
861     try {
862         approvedSectionsIndex = Integer.parseInt(currentLine);
863     } catch (NumberFormatException ex) {
864         throw new IOException();
865     }
866
867     currentLine = br.readLine();
868     if (currentLine != null) {
869         String noNameNumbersString = currentLine;
870         StringTokenizer st = new StringTokenizer(noNameNumbersString, ",");
871
872         String noNameSchoolNumberString = st.nextToken();
873         currentNewNoNameSchoolNumber = Integer.parseInt(noNameSchoolNumberString);
874
875         String noNameTeacherNumberString = st.nextToken();
876         currentNewNoNameTeacherNumber =
877             Integer.parseInt(noNameTeacherNumberString);
878
879         String noNameStudentNumberString = st.nextToken();
880         currentNewNoNameStudentNumber =
881             Integer.parseInt(noNameStudentNumberString);
882     } else {
883         throw new IOException();
884     }
885
886     currentLine = br.readLine();
887     if (currentLine != null) {
888         String numbersString = currentLine;
889         StringTokenizer st = new StringTokenizer(numbersString, ",");
890
891         String schoolsNumberString = st.nextToken();
892         schoolsNumber = Integer.parseInt(schoolsNumberString);
893
894         String teachersNumberString = st.nextToken();
895         teachersNumber = Integer.parseInt(teachersNumberString);
896
897         String studentsNumberString = st.nextToken();
898         studentsNumber = Integer.parseInt(studentsNumberString);
899     } else {
900         throw new IOException();
901     }
902
903     currentLine = br.readLine();
904     if (currentLine != null) {
905         String booleansString = currentLine;
906         StringTokenizer st = new StringTokenizer(booleansString, ",");
907
908         String reassignmentTextString = st.nextToken();
909         reassignmentText = Boolean.parseBoolean(reassignmentTextString);
910
911         String studentCodesString = st.nextToken();
912         studentCodes = Boolean.parseBoolean(studentCodesString);
913
914         String studentChangesString = st.nextToken();
915         studentChanges = Boolean.parseBoolean(studentChangesString);
916     } else {
917         throw new IOException();
918     }
919
920     if (br.readLine() == null) {
921         throw new IOException();
922     }
923     // </editor-fold>
924
925     // <editor-fold defaultstate="collapsed" desc="$Settings">
926     if (approvedSectionsIndex >= 0) {
927         currentLine = br.readLine();
928         if (currentLine == null || !currentLine.equals("$Settings")) {
929             throw new IOException();
930         }
931
932         Object[] settingsArray = new Object[29];
933         for (int i = 0; i < 29; i++) {
934             currentLine = br.readLine();
935             if (currentLine == null) {
936                 throw new IOException();
937             }
938
939         }
940
941     }
942
943 
```

```

        if (currentLine.startsWith("#\t")) {
            settingsArray[i] = Integer.parseInt(currentLine.substring(2));
        } else if (currentLine.startsWith(">\t")) {
            settingsArray[i] = Boolean.parseBoolean(currentLine.substring(2));
        } else {
            throw new FileIOException();
        }
    }
    database.setSettings(settingsArray);

    currentLine = br.readLine();
    if (currentLine == null) {
        throw new FileIOException();
    }
} else {
    return;
}
// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="$Participants">
if (approvedSectionsIndex >= 1) {
    currentLine = br.readLine();
    if (currentLine == null || !currentLine.equals("$Participants")) {
        throw new FileIOException();
    }

    database = new Database();

    School currentSchool = null;
    boolean updateSchool = false;

    while (true) {
        currentLine = br.readLine();

        if (currentLine == null) {
            throw new FileIOException();
        }
        else if (currentLine.equals("")) {
            // insert the previous school to database
            if (currentSchool != null && updateSchool) {
                currentSchool.updateStudentPairs();
                database.insertSchool(currentSchool);
                updateSchool = false;
            }
            break;
        }
        else if (currentLine.startsWith(".\t")) { // SCHOOL
            // insert the previous school to database
            if (currentSchool != null && updateSchool) {
                currentSchool.updateStudentPairs();
                database.insertSchool(currentSchool);
                updateSchool = false;
            }
            // read the current line
            StringTokenizer st1 = new StringTokenizer(currentLine, ":");

            // get principal attribute (name)
            String firstPart = st1.nextToken();
            String schoolName = firstPart.substring(2);

            // get the other attributes (behind the ":")
            String secondPart = st1.nextToken();
            StringTokenizer st2 = new StringTokenizer(secondPart, ",");

            // (codeLetter)
            char schoolCodeLetter;
            try {
                schoolCodeLetter = (st2.nextToken().toCharArray())[0];
            } catch (ArrayIndexOutOfBoundsException ex) {
                throw new FileIOException();
            }

            // (beginCodeNumber)
            int schoolBeginCodeNumber;
            try {
                schoolBeginCodeNumber = Integer.parseInt(st2.nextToken());
            }

```

```

    } catch (NumberFormatException ex) {
        throw new IOException();
    }

1020    // (endCodeNumber)
    int schoolEndCodeNumber;
    try {
        schoolEndCodeNumber = Integer.parseInt(st2.nextToken());
    } catch (NumberFormatException ex) {
        throw new IOException();
    }

1025    // construct the current school
    currentSchool = new School(schoolName, schoolCodeLetter,
        schoolBeginCodeNumber, schoolEndCodeNumber);

1030    updateSchool = true;

1035    } else if (currentLine.startsWith("?\\t\\t") && currentSchool != null) {
        // TEACHER
        // get principal attribute (name)
        String teacherName = currentLine.substring(3);

1040        // construct the current teacher
        Teacher currentTeacher = new Teacher(teacherName, currentSchool);

        // insert the current teacher to current school
        try {
            currentSchool.insertTeacher(currentTeacher);
        } catch (IllegalArgumentException ex) {
            throw new IOException();
        }
    }

1045    } else if (currentLine.startsWith("!\\t\\t") && currentSchool != null) {
        // STUDENT
        // read the current line
        StringTokenizer st1 = new StringTokenizer(currentLine, ":");

1050        // get principal attribute (name)
        String firstPart = st1.nextToken();
        String studentName = firstPart.substring(3);

        // get the other attributes (behind the ":")
        String secondPart = st1.nextToken();
        StringTokenizer st2 = new StringTokenizer(secondPart, ",");

        // (code)
        String studentCode = st2.nextToken();

1055        // (events)
        boolean[] studentEvents = new boolean[5];
        for (int i = 0; i < 5; i++) {
            String currentEventString = st2.nextToken();
            if (currentEventString.equalsIgnoreCase("true")) {
                studentEvents[i] = true;

                switch (i) {
                    case 0: // oo
                        currentSchool.incrementOOSTudentNumber();
                        break;
                    case 1: // oi
                        currentSchool.incrementOIStudentNumber();
                        break;
                    case 2: // is
                        currentSchool.incrementISSStudentNumber();
                        break;
                }
            } else if (currentEventString.equalsIgnoreCase("false")) {
                studentEvents[i] = false;
            } else {
                throw new IOException();
            }
        }
    }

1060    // (daUnpaired)
    boolean studentDAUnpaired;
    String studentDAUnpairedString = st2.nextToken();
    if (studentDAUnpairedString.equalsIgnoreCase("true")) {
        studentDAUnpaired = true;
    }
}

```

```

    } else if (studentDAUnpairedString.equalsIgnoreCase("false")) {
        studentDAUnpaired = false;
    } else {
        throw new IOException();
    }

    // (debateUnpaired)
    boolean studentDebateUnpaired;
    String studentDebateUnpairedString = st2.nextToken();
    if (studentDebateUnpairedString.equalsIgnoreCase("true")) {
        studentDebateUnpaired = true;
    } else if (studentDebateUnpairedString.equalsIgnoreCase("false"))
    {
        studentDebateUnpaired = false;
    } else {
        throw new IOException();
    }

    // (daStudentPairTempString)
    String daTokenString = st2.nextToken();
    String studentDAStudentPairTempString;
    if (!daTokenString.equals("null")) {
        studentDAStudentPairTempString = daTokenString;
    } else {
        studentDAStudentPairTempString = "";
    }

    // (debateStudentPairTempString)
    String debateTokenString = st2.nextToken();
    String studentDebateStudentPairTempString;
    if (!debateTokenString.equals("null")) {
        studentDebateStudentPairTempString = debateTokenString;
    } else {
        studentDebateStudentPairTempString = "";
    }

    // (reassignmentText)
    boolean studentReassignmentText;
    String studentReassignmentTextString = st2.nextToken();
    if (studentReassignmentTextString.equalsIgnoreCase("true")) {
        studentReassignmentText = true;
    } else if
    (studentReassignmentTextString.equalsIgnoreCase("false")) {
        studentReassignmentText = false;
    } else {
        throw new IOException();
    }

    // construct the current student
    Student currentStudent = new Student(studentName, currentSchool,
studentCode, studentEvents, studentDAUnpaired, studentDebateUnpaired,
                studentDAStudentPairTempString,
studentDebateStudentPairTempString, studentReassignmentText);

    // insert the current teacher to current school
    try {
        currentSchool.insertStudent(currentStudent);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    }

    } else if (currentLine.startsWith("/\t\t") && currentSchool != null) {
// UNPAIRED DA
    // get principal attribute (name)
    String unpairedDAStudentName = currentLine.substring(3);

    // find a student with the same name
    Student unpairedDAStudent;
    try {
        unpairedDAStudent =
currentSchool.getStudent(unpairedDAStudentName);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    } catch (NoSuchElementException ex) {
        throw new IOException();
    }

    // insert the current unpaired da student to current school
    try {

```

```

1175             currentSchool.insertUnpairedDASStudent(unpairedDASStudent);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    }
}
1180 } else if (currentLine.startsWith("\t\t") && currentSchool != null) {
// UNPAIRED DEBATE
    // get principal attribute (name)
    String unpairedDebateStudentName = currentLine.substring(3);
}
1185     // find a student with the same name
    Student unpairedDebateStudent;
    try {
        unpairedDebateStudent =
currentSchool.getStudent(unpairedDebateStudentName);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    } catch (NoSuchElementException ex) {
        throw new IOException();
    }
}
1190     // insert the current unpaired debate student to current school
try {
    currentSchool.insertUnpairedDebateStudent(unpairedDebateStudent);
} catch (IllegalArgumentException ex) {
    throw new IOException();
}
1200 } else if (currentLine.startsWith("-\t\t") && currentSchool != null) {
// DA PAIR
    // read the current line
    StringTokenizer st1 = new StringTokenizer(currentLine, ":");

    // get principal attribute (originalCode)
    String firstPart = st1.nextToken();
    String daPairOriginalCode = firstPart.substring(2);

    // get the other attributes (behind the ":")
    String secondPart = st1.nextToken();
    StringTokenizer st2 = new StringTokenizer(secondPart, ",");

    // (student1)
    String daPairStudent1Name = st2.nextToken();

    Student daPairStudent1;
    try {
        daPairStudent1 = currentSchool.getStudent(daPairStudent1Name);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    } catch (NoSuchElementException ex) {
        throw new IOException();
    }

    // (student2)
    String daPairStudent2Name = st2.nextToken();

    Student daPairStudent2;
    try {
        daPairStudent2 = currentSchool.getStudent(daPairStudent2Name);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    } catch (NoSuchElementException ex) {
        throw new IOException();
    }
}
1220     // construct the current da pair
    DASStudentPair daPair;
    try {
        daPair = new DASStudentPair(daPairStudent1, daPairStudent2);
    } catch (IllegalArgumentException ex) {
        throw new IOException();
    }

    // check that the constructed da pair has the same code as it
1245    should
    if (!daPair.getOriginalCode().equals(daPairOriginalCode)) {
        throw new IOException();
    }
}
1250

```

```

1255          // insert the current da pair to current school
1256          try {
1257              currentSchool.insertDAPair(daPair);
1258          } catch (IllegalArgumentException ex) {
1259              throw new IOException();
1260          }
1261
1262      } else if (currentLine.startsWith("<\t\t") && currentSchool != null) {
1263          // DEBATE PAIR
1264          // read the current line
1265          StringTokenizer st1 = new StringTokenizer(currentLine,":");
1266
1267          // get principal attribute (originalCode)
1268          String firstPart = st1.nextToken();
1269          String debatePairOriginalCode = firstPart.substring(2);
1270
1271          // get the other attributes (behind the ":")
1272          String secondPart = st1.nextToken();
1273          StringTokenizer st2 = new StringTokenizer(secondPart,",");
1274
1275          // (student1)
1276          String debatePairStudent1Name = st2.nextToken();
1277
1278          Student debatePairStudent1;
1279          try {
1280              debatePairStudent1 =
1281                  currentSchool.getStudent(debatePairStudent1Name);
1282          } catch (IllegalArgumentException ex) {
1283              throw new IOException();
1284          } catch (NoSuchElementException ex) {
1285              throw new IOException();
1286          }
1287
1288          // (student2)
1289          String debatePairStudent2Name = st2.nextToken();
1290
1291          Student debatePairStudent2;
1292          try {
1293              debatePairStudent2 =
1294                  currentSchool.getStudent(debatePairStudent2Name);
1295          } catch (IllegalArgumentException ex) {
1296              throw new IOException();
1297          } catch (NoSuchElementException ex) {
1298              throw new IOException();
1299          }
1300
1301          // construct the current debate pair
1302          DebateStudentPair debatePair;
1303          try {
1304              debatePair = new DebateStudentPair(debatePairStudent1,
1305                  debatePairStudent2);
1306          } catch (IllegalArgumentException ex) {
1307              throw new IOException();
1308          }
1309
1310          // check that the constructed debate pair has the same code as it
1311          should
1312          if (!debatePair.getOriginalCode().equals(debatePairOriginalCode))
1313          {
1314              throw new IOException();
1315          }
1316
1317          // insert the current debate pair to current school
1318          try {
1319              currentSchool.insertDebatePair(debatePair);
1320          } catch (IllegalArgumentException ex) {
1321              throw new IOException();
1322          }
1323
1324          } else {
1325              throw new IOException();
1326          }
1327      }
1328  } else {
1329      database.eraseSchoolTree();
1330      return;
1331  }
1332 // </editor-fold>

```

```

1335 // <editor-fold defaultstate="collapsed" desc="$Qualification">
1336 if (approvedSectionsIndex >= 2) {
1337     currentLine = br.readLine();
1338     if (currentLine == null || !currentLine.equals("$Qualification")) {
1339         throw new FileIOException();
1340     }
1341     qualification = new Qualification(database);
1342
1343     int currentEventIndex = -999;
1344     int eventJudgesPerRoom = -999;
1345     int eventEntitiesPerRoom = -999;
1346     BinarySearchTree currentJudgeTree = new BinarySearchTree();
1347     BinarySearchTree currentEntityTree = new BinarySearchTree();
1348     boolean updateEvent = false;
1349
1350     int currentRoundIndex = 0;
1351     boolean updateRound = false;
1352
1353     int currentRoomIndex = 0;
1354     boolean updateRoom = false;
1355
1356     while (true) {
1357         currentLine = br.readLine();
1358
1359         if (currentLine == null) {
1360             throw new FileIOException();
1361
1362         } else if (currentLine.equals("")) { // LAST LINE
1363             if (updateEvent) {
1364                 currentRoundIndex = 0;
1365
1366                 updateEvent = false;
1367             }
1368             if (updateRound) {
1369                 currentRoundIndex += 1;
1370
1371                 currentRoomIndex = 0;
1372
1373                 updateRound = false;
1374             }
1375             if (updateRoom) {
1376                 currentRoomIndex += 1;
1377
1378                 updateRoom = false;
1379             }
1380             break;
1381
1382         } if (currentLine.startsWith("@\t")) { // EVENT
1383             // update event
1384             if (updateEvent) {
1385                 currentEventIndex += 1;
1386
1387                 currentRoundIndex = 0;
1388
1389                 updateEvent = false;
1390             }
1391
1392             // read the current line
1393             StringTokenizer st1 = new StringTokenizer(currentLine, ":");

1394             // get principal attribute (typeEventName)
1395             String firstPart = st1.nextToken();
1396             String eventName = firstPart.substring(2);

1397             // get the other attributes (behind the ":")
1398             String secondPart = st1.nextToken();
1399             StringTokenizer st2 = new StringTokenizer(secondPart, ",");

1400             // (judgesPerRoom)
1401             try {
1402                 String eventJudgesPerRoomString = st2.nextToken();
1403                 eventJudgesPerRoom =
1404                     Integer.parseInt(eventJudgesPerRoomString);
1405             } catch (NumberFormatException ex) {
1406                 throw new FileIOException();
1407             }
1408
1409         }

```

```

    // (entitiesPerRoom)
1415   try {
      String eventEntitiesPerRoomString = st2.nextToken();
      eventEntitiesPerRoom =
      Integer.parseInt(eventEntitiesPerRoomString);
      } catch (NumberFormatException ex) {
          throw new IOException();
      }

      // get eventType and appropriate trees and event indices
      currentJudgeTree = qualification.getJudgeTree();

1425
      Event.Type eventType;
      if (eventName.equals(Event.Type.ORIGINAL_ORATORY.getEventName()))
      {
          eventType = Event.Type.ORIGINAL_ORATORY;
          currentEntityTree = qualification.getOOEntityTree();
          currentEventIndex = 0;

          } else if
      (eventName.equals(Event.Type.ORAL_INTERPRETATION.getEventName())) {
          eventType = Event.Type.ORAL_INTERPRETATION;
          currentEntityTree = qualification.getOIEntityTree();
          currentEventIndex = 1;

          } else if
      (eventName.equals(Event.Type.IMPROMPTU_SPEAKING.getEventName())) {
          eventType = Event.Type.IMPROMPTU_SPEAKING;
          currentEntityTree = qualification.getISEntityTree();
          currentEventIndex = 2;

          } else if
      (eventName.equals(Event.Type.DUET_ACTING.getEventName())) {
          eventType = Event.Type.DUET_ACTING;
          currentEntityTree = qualification.getDAEntityTree();
          currentEventIndex = 3;

          } else if (eventName.equals(Event.Type.DEBATE.getEventName())) {
          eventType = Event.Type.DEBATE;
          currentEntityTree = qualification.getDebateEntityTree();
          currentEventIndex = 4;

          } else {
              throw new IOException();
          }

          // construct the current event
          QualificationEvent currentEvent = new
          QualificationEvent(qualification, eventType, currentJudgeTree, currentEntityTree,
          eventJudgesPerRoom, eventEntitiesPerRoom);

          // set the currentEvent to be the event at the correct place in
1465 eventArray
          qualification.setEvent(currentEvent, currentEventIndex);

          updateEvent = true;

1470     } else if (currentLine.startsWith("\t\t")) { // ROUND
          // update round
          if (updateRound) {
              currentRoundIndex += 1;

              currentRoomIndex = 0;

              updateRound = false;

              updateRoom = false;
          }

          // read the current line and set name
          String roundName = currentLine.substring(3);

1485         Event currentEvent =
          qualification.getEventArrayElement(currentEventIndex);

          // construct the current round

```

```

1490     Round currentRound = new Round(currentEvent, currentRoundIndex,
1491     roundName, currentJudgeTree, currentEntityTree, eventJudgesPerRoom,
1492     eventEntitiesPerRoom);

1495         // add the new current round to be the round at the correct place
in roundArray
1496         currentEvent.setRound(currentRound, currentRoundIndex);

1500         updateRound = true;

1505     } else if (currentLine.startsWith("%\t\t\t")) { // ROOM
1506         // update room index
1507         if (updateRoom) {
1508             currentRoomIndex += 1;

1509             updateRoom = false;
1510         }

1515         // read the current line and set name
1516         String roomName = currentLine.substring(4);

1520         Event currentEvent =
qualification.getEventArrayElement(currentEventIndex);
1521         Round currentRound =
currentEvent.getRoundArrayElement(currentRoundIndex);

1525         // construct the current room
1526         Room currentRoom = new Room(currentRound, roomName,
1527         eventJudgesPerRoom, eventEntitiesPerRoom);

1530         // add the new current room to be the room at the correct place in
this round's roomArray
1531         currentRound.setRoom(currentRoom, currentRoomIndex);

1535         updateRoom = true;

1540     } else if (currentLine.startsWith("r\t\t\t\t")) { // JUDGE IN ROOM
1541         // get principal attribute (ID)
1542         int roomJudgeID;
1543         try {
1544             roomJudgeID = Integer.parseInt(currentLine.substring(5));
1545         } catch (NumberFormatException ex) {
1546             throw new IOException();
1547         }

1550         // get the judge under that ID
1551         Judge roomJudge;
1552         try {
1553             roomJudge = qualification.getJudge(roomJudgeID);
1554         } catch (NoSuchElementException ex) {
1555             throw new IOException();
1556         }

1560         // insert the judge into room judges
1561         Event currentEvent =
qualification.getEventArrayElement(currentEventIndex);
1562         Round currentRound =
currentEvent.getRoundArrayElement(currentRoundIndex);
1563         Room currentRoom =
currentRound.getRoomArrayElement(currentRoomIndex);

1565         try {
1566             currentRoom.allocateJudge(roomJudge);
1567         } catch (IllegalArgumentException ex) {
1568             throw new IOException();
1569         }

1570     } else if (currentLine.startsWith("o\t\t\t\t")) { // OO ENTITY IN ROOM
1571         // get principal attribute (code)
1572         String roomEntityCode;
1573         //try {
1574             roomEntityCode = currentLine.substring(5);
1575         //} catch (NumberFormatException ex) {
1576             //    throw new IOException();
1577         //}

1580         // get the entity under that code
1581         Entity roomEntity;
1582         try {

```

```

    roomEntity = qualification.getOOEntity(roomEntityCode);
1570   } catch (NoSuchElementException ex) {
    throw new FileIOException();
}
// insert the entity into room entities
Event currentEvent =
1575 qualification.getEventArrayElement(currentEventIndex);
Round currentRound =
currentEvent.getRoundArrayElement(currentRoundIndex);
Room currentRoom =
currentRound.getRoomArrayElement(currentRoomIndex);
1580
try {
    currentRoom.allocateEntity(roomEntity);
} catch (IllegalArgumentException ex) {
    throw new FileIOException();
}
1585
} else if (currentLine.startsWith("i\t\t\t\t")) { // OI ENTITY IN ROOM
// get principal attribute (code)
String roomEntityCode;
1590 //try {
    roomEntityCode = currentLine.substring(5);
//} catch (NumberFormatException ex) {
//    throw new FileIOException();
//}
1595
// get the entity under that code
Entity roomEntity;
try {
    roomEntity = qualification.getOIEntity(roomEntityCode);
} catch (NoSuchElementException ex) {
    throw new FileIOException();
}
1600
// insert the entity into room entities
Event currentEvent =
qualification.getEventArrayElement(currentEventIndex);
Round currentRound =
currentEvent.getRoundArrayElement(currentRoundIndex);
Room currentRoom =
currentRound.getRoomArrayElement(currentRoomIndex);
1605
try {
    currentRoom.allocateEntity(roomEntity);
} catch (IllegalArgumentException ex) {
    throw new FileIOException();
}
1610
} else if (currentLine.startsWith("m\t\t\t\t")) { // IS ENTITY IN ROOM
// get principal attribute (code)
String roomEntityCode;
1615 //try {
    roomEntityCode = currentLine.substring(5);
//} catch (NumberFormatException ex) {
//    throw new FileIOException();
//}
1620
// get the entity under that code
Entity roomEntity;
try {
    roomEntity = qualification.getISEntity(roomEntityCode);
} catch (NoSuchElementException ex) {
    throw new FileIOException();
}
1625
// insert the entity into room entities
Event currentEvent =
qualification.getEventArrayElement(currentEventIndex);
Round currentRound =
currentEvent.getRoundArrayElement(currentRoundIndex);
Room currentRoom =
currentRound.getRoomArrayElement(currentRoomIndex);
1630
try {
    currentRoom.allocateEntity(roomEntity);
} catch (IllegalArgumentException ex) {
    throw new FileIOException();
}
1635
1640
1645

```

```

        }

1650    } else if (currentLine.startsWith("d\t\t\t\t")) { // DA ENTITY IN ROOM
        // get principal attribute (code)
        String roomEntityCode;
        //try {
        //    roomEntityCode = currentLine.substring(5);
        //} catch (NumberFormatException ex) {
        //    throw new FileIOException();
        //}

        // get the entity under that code
        Entity roomEntity;
        try {
            roomEntity = qualification.getDAEntity(roomEntityCode);
        } catch (NoSuchElementException ex) {
            throw new FileIOException();
        }

1665    // insert the entity into room entities
        Event currentEvent =
qualification.getEventArrayElement(currentEventIndex);
        Round currentRound =
1670    currentEvent.getRoundArrayElement(currentRoundIndex);
        Room currentRoom =
currentRound.getRoomArrayElement(currentRoomIndex);

        try {
            currentRoom.allocateEntity(roomEntity);
        } catch (IllegalArgumentException ex) {
            throw new FileIOException();
        }

1680    } else if (currentLine.startsWith("e\t\t\t\t")) { // DEBATE ENTITY IN
ROOM
        // get principal attribute (code)
        String roomEntityCode;
        //try {
        //    roomEntityCode = currentLine.substring(5);
        //} catch (NumberFormatException ex) {
        //    throw new FileIOException();
        //}

1685    // get the entity under that code
        Entity roomEntity;
        try {
            roomEntity = qualification.getDebateEntity(roomEntityCode);
        } catch (NoSuchElementException ex) {
            throw new FileIOException();
        }

        // insert the entity into room entities
        Event currentEvent =
1700    qualification.getEventArrayElement(currentEventIndex);
        Round currentRound =
currentEvent.getRoundArrayElement(currentRoundIndex);
        Room currentRoom =
currentRound.getRoomArrayElement(currentRoomIndex);

1705    try {
            currentRoom.allocateEntity(roomEntity);
        } catch (IllegalArgumentException ex) {
            throw new FileIOException();
        }

1710    } else {
        throw new FileIOException();
    }
}

1715    }
} else { // if this section was not approved
    return; // end reading
}
// </editor-fold>
}

1720 /**
 * The saveFile saves the contents of the program into a user-specified
 * file.
 * <p>
1725

```

```

    * A file chooser dialog is invoked, prompting the user to select a file or
    * create a new one. The sections of the program up to and including the one
    * specified by the value of the approvedSectionsIndex are saved.
    *
1730    * @throws UserIOException if the file chooser dialog is canceled by the
    * user
    * @throws IOException if an error is encountered during the writing of the
    * data into the file, or the save dialog returned the ERROR_OPTION
    *
1735    * @author Zbyněk Stara
    */
    private void saveFile() throws FileIOException, UserIOException, IOException {
        JFileChooser jfc = new JFileChooser();
        int saveDialogReturn = jfc.showSaveDialog(this);

        if (saveDialogReturn == JFileChooser.APPROVE_OPTION) {
            BufferedWriter bw = new BufferedWriter(new
                FileWriter(jfc.getSelectedFile()));
            /*
             * <editor-fold defaultstate="collapsed" desc="Symbols used for parsing">
             *
             * (WATCH OUT! Only first 256 Unicode symbols can be properly rendered in
1745             *.txt files)
             *
             * $      - section
             *
             * #\t    - int setting
             * >\t    - boolean setting
             * &\t    - button selection control
             *
             * .\t    - school
             * ?\t\t  - teacher
             * !\t\t  - student
             * /\t\t  - unpairedDAStudent
             * |\t\t  - unpairedDebateStudent
             * -\t\t  - daPair
             * <\t\t  - debatePair
             *
1750             * J\t    - judge in tree
             * j\t\t  - judge ref
             * O\t    - oo entity in tree
             * jo\t\t - encountered judge for oo entity
             * I\t    - oi entity in tree
             * ji\t\t - encountered judge for oi entity
             * M\t    - is entity in tree
             * jm\t\t - encountered judge for is entity
             * D\t    - da entity in tree
             * jd\t\t - encountered judge for da entity
             * E\t    - debate entity int tree
             * je\t\t - encountered judge for debate entity
             * @\t    - event
             * ~\t\t  - round
             * f\t\t\t - free judge tree
             * %\t\t\t - room
             * r\t\t\t - judge
             * o\t\t\t - oo entity ref
             * i\t\t\t - oi entity ref
             * m\t\t\t - is entity ref
             * d\t\t\t - da entity ref
             * e\t\t\t - debate entity ref
             */
             // </editor-fold>
1790        System.out.println("before intro");

        // <editor-fold defaultstate="collapsed" desc="Introduction (8 lines)">
        Date currentDate = new Date();
        SimpleDateFormat dateFormat = new SimpleDateFormat("MMM-dd-yyyy
1795           HH:mm:ss");
        String currentTimestamp = dateFormat.format(currentDate);

        bw.write("{NESDA Tournament Manager Save File}\n");
        bw.write("\n");
        bw.write("Created by NESDA Tournament Manager 2012 on " + currentTimestamp
1800           + "\n");
        bw.write("\n");
        bw.write("IMPORTANT: Do not modify this file. Your modifications "

```

```

1805          + "might disrupt the program's parsing of saved information "
          + "and lead to errors or cause a possible loss of data.\n");
bw.write("\n");
bw.write("NOTE: This file is not an export file of the data "
1810          + "inputted into the NESDA tournament manager. If you wish "
          + "to obtain a plain-text copy of a select section of the "
          + "program, click the appropriate \"Export\" button in the "
          + "program's user interface.\n");
bw.write("\n");// </editor-fold>
1815 System.out.println("before attributes");
1820          // <editor-fold defaultstate="collapsed" desc="$Attributes">
bw.write("$Attributes\n");
bw.write(approvedSectionsIndex + "\n");
bw.write(currentNewNoNameSchoolNumber + "," +
currentNewNoNameTeacherNumber + "," + currentNewNoNameStudentNumber + "\n");
bw.write(schoolsNumber + "," + teachersNumber + "," + studentsNumber +
"\n");
1825          bw.write(reassignmentText + "," + studentCodes + "," + studentChanges +
"\n");
bw.write("\n");// </editor-fold>
System.out.println("before settings");
1830          // <editor-fold defaultstate="collapsed" desc="$Settings">
if (approvedSectionsIndex >= 0) {
    bw.write("$Settings\n");
1835          Object[] settingsArray = database.getSettings();
String settingsHelperString = "";
for (int i = 0; i < settingsArray.length; i++) {
    if (settingsArray[i].getClass().equals(Integer.class)) {
        settingsHelperString += ("#\t" + ((Integer)
1840 settingsArray[i]).toString() + "\n");
    } else if (settingsArray[i].getClass().equals(Boolean.class)) {
        settingsHelperString += (">\t" + ((Boolean)
settingsArray[i]).toString() + "\n");
    }
}
bw.write(settingsHelperString);
1845          bw.write("\n");
} else {
    bw.close();
    return;
} // </editor-fold>
1850 System.out.println("before participants");
1855          // <editor-fold defaultstate="collapsed" desc="$Participants">
if (approvedSectionsIndex >= 0) {
    bw.write("$Participants\n");
1860          String participantsHelperString = "";
for (int i = 0; i < database.getSchoolTreeSize(); i++) {
    School school = (School) database.getSchoolTreeNodeData(i);
participantsHelperString += (".\t" + school.getName() + ":" +
+ school.getCodeLetter() + "," +
school.getBeginCodeNumber() + "," + school.getEndCodeNumber() + "\n");
1865          for (int j = 0; j < school.getTeacherTreeSize(); j++) {
    Teacher teacher = (Teacher) school.getTeacherTreeNodeData(j);
participantsHelperString += ("?\t\t" + teacher.getName() +
"\n");
}
1870          for (int j = 0; j < school.getStudentTreeSize(); j++) {
    Student student = (Student) school.getStudentTreeNodeData(j);
participantsHelperString += ("!\t\t" + student.getName() + ":" +
+ student.getCode() + "," + student.getEvents()[0] +
","
1875          + student.getEvents()[1] + "," +
student.getEvents()[2] + ","
}
1880          }
}

```

```

    + student.getEvents()[3] + "," +
1885 student.getEvents()[4] + "," +
           + student.getDAUnpaired() + "," +
student.getDebateUnpaired() + ",");

           if (student.getDAStudentPair() != null) {
1890   participantsHelperString +=
student.getDAStudentPair().getOriginalCode() + ",";
           } else {
           participantsHelperString += "null" + ",";
           }

           if (student.getDebateStudentPair() != null) {
1895   participantsHelperString +=
student.getDebateStudentPair().getOriginalCode();
           } else {
           participantsHelperString += "null" + ",";
           }

           participantsHelperString += (student.getReassignmentText() +
"\n");
1905
}

for (int j = 0; j < school.getUnpairedDAStudentTreeSize(); j++) {
  Student student = (Student)
school.getUnpairedDAStudentTreeNodeData(j);

1910   participantsHelperString += ("/\t\t" + student.getName() +
"\n"); // found in students tree
}

1915 for (int j = 0; j < school.getUnpairedDebateStudentTreeSize();
j++) {
  Student student = (Student)
school.getUnpairedDebateStudentTreeNodeData(j);

1920   participantsHelperString += ("|\t\t" + student.getName() +
"\n"); // found in students tree
}

1925 for (int j = 0; j < school.getDAPairTreeSize(); j++) {
  DAStudentPair daStudentPair = (DAStudentPair)
school.getDAPairTreeNodeData(j);

1930   participantsHelperString += ("-/\t\t" +
daStudentPair.getOriginalCode() + ":" +
                     + daStudentPair.getStudentArray()[0].getName() + "," +
daStudentPair.getStudentArray()[1].getName() + "\n");
}

1935 for (int j = 0; j < school.getDebatePairTreeSize(); j++) {
  DebateStudentPair debateStudentPair = (DebateStudentPair)
school.getDebatePairTreeNodeData(j);

1940   participantsHelperString += (<|\t\t" +
debateStudentPair.getOriginalCode() + ":" +
                     + debateStudentPair.getStudentArray()[0] + "," +
debateStudentPair.getStudentArray()[1] + "\n");
}

1945   bw.write(participantsHelperString);

1950   bw.write("\n");
} else {
  bw.close();
  return;
} // </editor-fold>

1955 System.out.println("before qualification");

// <editor-fold defaultstate="collapsed" desc="$Qualification">
if (approvedSectionsIndex >= 2) {
  bw.write("$Qualification\n");

1960   String qualificationHelperString = "";

  for (int i = 0; i < qualification.getEventArrayLength(); i++) {
    Event currentEvent = qualification.getEventArrayElement(i);
}

```

```

qualificationHelperString += ("@\t" +
currentEvent.getTypeEventName() + ":" +
/*+ currentEvent.isSemifinal() + "," +
currentEvent.isFinal() + ",*/" +
/*+ currentEvent.getShortEventName() + "," */
+ currentEvent.getJudgesPerRoom() + "," +
+ currentEvent.getEntitiesPerRoom() + "\n");

for (int j = 0; j < currentEvent.getRoundArrayLength(); j++) {
    Round currentRound = currentEvent.getRoundArrayElement(j);

    qualificationHelperString += ("~\t\t" + currentRound.getName() +
+ "\n");

    /*for (int k = 0; k < currentRound.getFreeJudgeTreeSize();
    k++) {
        Judge currentFreeJudge =
        currentRound.getFreeJudgeTreeNodeData(k);
    */

    qualificationHelperString += ("f\t\t\t" +
currentFreeJudge.getID() + "\n");
    } */

    for (int k = 0; k < currentRound.getRoomArrayLength(); k++) {
        Room currentRoom = currentRound.getRoomArrayElement(k);

        qualificationHelperString += ("%\t\t\t" +
currentRoom.getName() + "\n");

        for (int l = 0; l < currentRoom.getJudgeTreeSize(); l++) {
            Judge currentJudge = (Judge)
            currentRoom.getJudgeTreeNodeData(l);

            qualificationHelperString += ("r\t\t\t\t" +
currentJudge.getID() + "\n");
        }

        for (int l = 0; l < currentRoom.getEntityTreeSize(); l++) {
            Entity currentEntity = (Entity)
            currentRoom.getEntityTreeNodeData(l);

            if (currentEntity.getType() ==
Event.Type.ORIGINAL_ORATORY) {
                qualificationHelperString += ("o\t\t\t\t\t" +
currentEntity.getCode() + "\n");
            } else if (currentEntity.getType() ==
Event.Type.ORAL_INTERPRETATION) {
                qualificationHelperString += ("i\t\t\t\t\t" +
currentEntity.getCode() + "\n");
            } else if (currentEntity.getType() ==
Event.Type.IMPROMPTU_SPEAKING) {
                qualificationHelperString += ("m\t\t\t\t\t" +
currentEntity.getCode() + "\n");
            } else if (currentEntity.getType() ==
Event.Type.DUET_ACTING) {
                qualificationHelperString += ("d\t\t\t\t\t" +
currentEntity.getCode() + "\n");
            } else if (currentEntity.getType() ==
Event.Type.DEBATE) {
                qualificationHelperString += ("e\t\t\t\t\t" +
currentEntity.getCode() + "\n");
            } else { // if type is UNDEFINED
                throw new IOException();
            }
        }
    }
}

bw.write(qualificationHelperString);

bw.write("\n");
} else {
    bw.close();
    return;
} // </editor-fold>

System.out.println("after qualification");

```

```

    // NOT USED:

    // <editor-fold defaultstate="collapsed" desc="$Finalists">
2045    if (approvedSectionsIndex >= 3) {
        bw.write("$Finalists\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="$Finals">
2050    if (approvedSectionsIndex >= 4) {
        bw.write("$Finals\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="$Winners">
2055    if (approvedSectionsIndex >= 5) {
        bw.write("$Winners\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="AllocationWorker">
2060    if (approvedSectionsIndex >= 6) {
        bw.write("AllocationWorker\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="AllocationInfoDialog">
2065    if (approvedSectionsIndex >= 7) {
        bw.write("AllocationInfoDialog\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="AllocationInfoTextField">
2070    if (approvedSectionsIndex >= 8) {
        bw.write("AllocationInfoTextField\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // <editor-fold defaultstate="collapsed" desc="AllocationInfoProgressBar">
2075    if (approvedSectionsIndex >= 9) {
        bw.write("AllocationInfoProgressBar\n");

        bw.write("\n");
    } else {
        bw.close();
        return;
    } // </editor-fold>

    // ACCESSING STATIC MAIN_GUI ELEMENTS:
2080 /**
 * This method allows access to the static allocationInfoDialog to other
 * parts of the program. It is used by the AllocationWorker as part of the
 * progress-reporting functionality.
 */
2085 * @return allocationInfoDialog
 *
 * @author Zbyněk Stara
 */
2090 public static javax.swing.JDialog getAllocationInfoDialog() {
    return allocationInfoDialog;
}

/**
 * This method allows access to the static allocationInfoTextField to other
2095 * parts of the program. It is used by the AllocationWorker as part of the
 * progress-reporting functionality.
 *
 * @return allocationInfoTextField
 *
 * @author Zbyněk Stara
 */
2100 public static javax.swing.JTextField getAllocationInfoTextField() {
    return allocationInfoTextField;
}

/**
 * This method allows access to the static allocationInfoProgressBar to
2105 * other parts of the program. It is used by the AllocationWorker as part of
 * the progress-reporting functionality.
 *
 * @return allocationInfoProgressBar
 *
 * @author Zbyněk Stara
 */
2110 public static javax.swing.JProgressBar getAllocationInfoProgressBar() {
    return allocationInfoProgressBar;
}

// MAIN_GUI UPDATES:

```

```

2120 /**
 * This method changes the value of the overallProgressBar
 *
 * @param value an int from 0 to 100, indicating the new value
 * @throws IllegalArgumentException if the value is outside the acceptable
 * range (smaller than 0 or larger than 100)
 *
 * @author Zbyněk Stara
 */
2125 private void updateOverallProgressBar(int value) throws IllegalArgumentException {
    if (value < 0 || value > 100) {
        throw new IllegalArgumentException("Value passed to overall progress bar
is not between 0 and 100.");
    } else {
        overallProgressBar.setValue(value);
    }
}

// PARTICIPANTS TAB UPDATES:
2130 /**
 * This method updates the value of the participantsProgressBar.
 *
 * @param value an int from 0 to 100, indicating the new value
 * @throws IllegalArgumentException if the value is outside the acceptable
 * range
 *
 * @author Zbyněk Stara
 */
2135 private void updateParticipantsProgressBar(int value) throws
IllegalArgumentException {
    if (value < 0 || value > 100) {
        throw new IllegalArgumentException("Value passed to participants progress
bar is not between 0 and 100.");
    } else {
        participantsProgressBar.setValue(value);
    }
}

/**
 * This method updates the participants progress bar according to the
 * current number of schools.
 *
 * @author Zbyněk Stara
 */
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
*/
 * This method updates the schoolsListModel to reflect changes in the school
 * tree. After that, the listModel is applied to the schoolsList.
 *
 * @author Zbyněk Stara
 */
private void updateSchoolsListModel() {
    DefaultListModel model = new DefaultListModel();

    School[] schoolArray = database.getSchoolArray();

    if (schoolArray.length != 0) {
        for (int i = 0; i < schoolArray.length; i++) {
            model.addElement(schoolArray[i].getName());
        }

        participantsSchoolsList.setModel(model);
    } else {
        participantsSchoolsList.setModel(emptyModel);
    }
}

/**
 * This method updates the teachersListModel to reflect changes in the
 * teacher tree of a specified school. After that, the listModel is applied

```

```

2200      * to the teachersList.
2201      *
2202      * @param currentSchool the school whose teacher tree is being used as a
2203      * basis of the teachers list
2204      *
2205      * @author Zbyněk Stara
2206      */
2207  private void updateTeachersListModel(School currentSchool) {
2208      DefaultListModel model = new DefaultListModel();
2209
2210      Teacher[] teacherArray = currentSchool.getTeacherArray();
2211
2212      if (teacherArray.length != 0) {
2213          for (int i = 0; i < teacherArray.length; i++) {
2214              model.addElement(teacherArray[i].getName());
2215          }
2216
2217          participantsTeachersList.setModel(model);
2218      } else {
2219          participantsTeachersList.setModel(emptyModel);
2220      }
2221
2222      /**
2223      * This method updates the studentsListModel to reflect changes in the
2224      * student tree of a specified school. After that, the listModel is applied
2225      * to the studentsList.
2226      *
2227      * @param currentSchool the school whose student tree is being used as a
2228      * basis of the students list
2229      *
2230      * @author Zbyněk Stara
2231      */
2232  private void updateStudentsListModel(School currentSchool) {
2233      DefaultListModel model = new DefaultListModel();
2234
2235      Student[] studentArray = currentSchool.getStudentArray();
2236
2237      if (studentArray.length != 0) {
2238          for (int i = 0; i < studentArray.length; i++) {
2239              model.addElement(studentArray[i].getName());
2240          }
2241
2242          participantsStudentsList.setModel(model);
2243      } else {
2244          participantsStudentsList.setModel(emptyModel);
2245      }
2246
2247      /**
2248      * This method updates the studentCodesListModel to reflect changes in the
2249      * student tree of a specified school. After that, the listModel is applied
2250      * to the studentCodesList.
2251      *
2252      * @param currentSchool the school whose student tree is being used as a
2253      * basis of the studentCodesList
2254      *
2255      * @author Zbyněk Stara
2256      */
2257  private void updateStudentCodesListModel(School currentSchool) {
2258      DefaultListModel model = new DefaultListModel();
2259
2260      Student[] studentArray = currentSchool.getStudentArray();
2261
2262      if (studentArray.length != 0) {
2263          for (int i = 0; i < studentArray.length; i++) {
2264              model.addElement(studentArray[i].getCode());
2265          }
2266
2267          participantsStudentCodesList.setModel(model);
2268      } else {
2269          participantsStudentCodesList.setModel(emptyModel);
2270      }
2271
2272      /**
2273      * This method updates the assignPairsLeftListModel to reflect changes in
2274      * the pairs trees of a specified school. After that, the listModel is
2275      * applied to the assignPairsLeftList.

```

```

2280
    *
    * @param currentSchool the school whose pairs trees are being used as a
    * basis of the assignPairsLeftList
    *
    * @author Zbyněk Stara
    */
2285  private void updateAssignPairsLeftListModel(School currentSchool) {
        if (assignPairsEventsIndex == 0) { // DA
            DefaultListModel model = new DefaultListModel();

            Student[] studentArray = currentSchool.getUnpairedDASStudentArray();

2290        for (int i = 0; i < studentArray.length; i++) {
            model.addElement(studentArray[i].getCode());
        }

        participantsAssignPairsLeftList.setModel(model);
} else if (assignPairsEventsIndex == 1) { // Debate
        DefaultListModel model = new DefaultListModel();

        Student[] studentArray = currentSchool.getUnpairedDebateStudentArray();

2300    for (int i = 0; i < studentArray.length; i++) {
            model.addElement(studentArray[i].getCode());
        }

        participantsAssignPairsLeftList.setModel(model);
} else { // No selection
        participantsAssignPairsLeftList.setModel(blankModel);
    }
}

2310 /**
     * This method updates the assignPairsRightListModel to reflect changes in
     * the pairs trees of a specified school. After that, the listModel is
     * applied to the assignPairsRightList.
     *
     * @param currentSchool the school whose pairs trees are being used as a
     * basis of the assignPairsRightList
     *
     * @author Zbyněk Stara
     */
2320  private void updateAssignPairsRightListModel(School currentSchool) {
        if (assignPairsEventsIndex == 0) { // DA
            DefaultListModel model = new DefaultListModel();

            Student[] studentArray = currentSchool.getUnpairedDASStudentArray();

2325        for (int i = 0; i < studentArray.length; i++) {
            if (i == assignPairsLeftIndex) continue;
            else model.addElement(studentArray[i].getCode());
        }

        participantsAssignPairsRightList.setModel(model);
} else if (assignPairsEventsIndex == 1) { // Debate
        DefaultListModel model = new DefaultListModel();

2330        Student[] studentArray = currentSchool.getUnpairedDebateStudentArray();

        for (int i = 0; i < studentArray.length; i++) {
            if (i == assignPairsLeftIndex) continue;
            else model.addElement(studentArray[i].getCode());
        }

        participantsAssignPairsRightList.setModel(model);
} else { // No selection
        participantsAssignPairsRightList.setModel(blankModel);
    }
}

2340 /**
     * This method updates the assignPairsPairsListModel to reflect changes in
     * the pairs trees of a specified school. After that, the listModel is
     * applied to the assignPairsPairsList.
     *
     * @param currentSchool the school whose pairs trees are being used as a
     * basis of the assignPairsPairsList
     *
     * @author Zbyněk Stara

```

```

    */
2360 private void updateAssignPairsPairsListModel(School currentSchool) {
    if (assignPairsEventsIndex == 0) { // DA
        DefaultListModel model = new DefaultListModel();

        DAStudentPair[] studentPairArray = currentSchool.getDAPairArray();

2365     for (int i = 0; i < studentPairArray.length; i++) {
            model.addElement(studentPairArray[i].getCode());
        }

        participantsAssignPairsPairsList.setModel(model);
    } else if (assignPairsEventsIndex == 1) { // Debate
        DefaultListModel model = new DefaultListModel();

        DebateStudentPair[] studentPairArray = currentSchool.getDebatePairArray();

2375     for (int i = 0; i < studentPairArray.length; i++) {
            model.addElement(studentPairArray[i].getCode());
        }

        participantsAssignPairsPairsList.setModel(model);
    } else { // No selection
        participantsAssignPairsPairsList.setModel(blankModel);
    }
}

// QUALIFICATION TAB UPDATES:
2385 /**
 * This method updates the value of the qualificationProgressBar.
 *
 * @param value an int from 0 to 100, indicating the new value
 * @throws IllegalArgumentException if the value is outside the acceptable
 * range
 *
 * @author Zbyněk Stara
 */
2395 private void updateQualificationProgressBar(int value) {
    if (value < 0 || value > 100) {
        throw new IllegalArgumentException("Value passed to qualification progress
bar is not between 0 and 100.");
    } else {
        qualificationProgressBar.setValue(value);
    }
}

2405 /**
 * This method updates the qualificationEventsListModel
 *
 * @author Zbyněk Stara
 */
2410 private void updateEventsListModel() {
    DefaultListModel model = new DefaultListModel();
    Event[] eventArray = qualification.getEventArray();

    for (int i = 0; i < qualification.getEventArrayLength(); i++) {
        model.addElement(eventArray[i].type.getEventName());
    }

    qualificationEventsList.setModel(model);
}

2420 /**
 * This method updates the qualificationRoundsListModel to display the
 * rounds of a specified event.
 *
 * @param currentEvent the event whose roundsArray will serve as a basis for
 * this listModel.
 * @throws NullPointerException if the currentEvent supplied is null
 *
 * @author Zbyněk Stara
 */
2425 private void updateRoundsListModel(Event currentEvent) throws NullPointerException
{
    DefaultListModel model = new DefaultListModel();

    Round[] roundArray = currentEvent.getRoundArray();
}

```

```

        for (int i = 0; i < roundArray.length; i++) {
            model.addElement(roundArray[i].getName());
        }

2440    qualificationRoomsList.setModel(model);
}

2445 /**
 * This method updates the qualificationRoomsListModel to display the
 * rooms of a specified round.
 *
 * @param currentRound the round whose roomsArray will serve as a basis for
 * this listModel.
 * @throws NullPointerException if the currentRound supplied is null
 *
 * @author Zbyněk Stara
 */
private void updateRoomsListModel(Round currentRound) throws NullPointerException
{
    DefaultListModel model = new DefaultListModel();

    Room[] roomArray = currentRound.getRoomArray();

2460    for (int i = 0; i < roomArray.length; i++) {
        model.addElement(roomArray[i].getName());
    }

    qualificationRoomsList.setModel(model);
}

2465 /**
 * This method updates the qualificationJudgesListModel to display the
 * judges of a specified room.
 *
 * @param currentRoom the room whose judgesTree will serve as a basis for
 * this listModel.
 * @throws NullPointerException if the currentRoom supplied is null
 *
 * @author Zbyněk Stara
 */
private void updateJudgesListModel(Room currentRoom) throws NullPointerException {
    DefaultListModel model = new DefaultListModel();

    Judge[] judgeArray = currentRoom.getJudgeArray();

2480    for (int i = 0; i < judgeArray.length; i++) {
        model.addElement(judgeArray[i].getName());
    }

    qualificationJudgesList.setModel(model);
}

2485 /**
 * This method updates the qualificationStudentCodesListModel to display the
 * entities of a specified room.
 *
 * @param currentRoom the room whose entitiesTree will serve as a basis for
 * this listModel.
 * @throws NullPointerException if the currentRoom supplied is null
 *
 * @author Zbyněk Stara
 */
private void updateQStudentCodesListModel(Room currentRoom) throws
2500 NullPointerException {
    DefaultListModel model = new DefaultListModel();

    Entity[] entityArray = currentRoom.getEntityArray();

    for (int i = 0; i < entityArray.length; i++) {
        model.addElement(entityArray[i].getCode());
    }

    qualificationStudentCodesList.setModel(model);
}

2510 // UPDATE TABS ACTION BOXES:
/**
 * This method groups together the often-used group of statements that
 * disable elements of the settings tab.

```

```

2515      *
2516      * @author Zbyněk Stara
2517      */
2518      private void disableSettingsTabActionBar() {
2519          // <editor-fold defaultstate="collapsed" desc="Disabling text fields for
2520          settings">
2521              eventSettingsOOSR.setEnabled(false);
2522              eventSettingsOISR.setEnabled(false);
2523              eventSettingsISSR.setEnabled(false);
2524              eventSettingsDASR.setEnabled(false);
2525              eventSettingsOOFinalSR.setEnabled(false);
2526              eventSettingsOIFinalSR.setEnabled(false);
2527              eventSettingsISFinalSR.setEnabled(false);
2528              eventSettingsDAFFinalSR.setEnabled(false);
2529
2530              eventSettingsOOSES.setEnabled(false);
2531              eventSettingsOISES.setEnabled(false);
2532              eventSettingsISSES.setEnabled(false);
2533              eventSettingsDASES.setEnabled(false);
2534              eventSettingsDebateSES.setEnabled(false);
2535
2536              eventSettingsOOJR.setEnabled(false);
2537              eventSettingsOIJR.setEnabled(false);
2538              eventSettingsISJR.setEnabled(false);
2539              eventSettingsDAJR.setEnabled(false);
2540              eventSettingsDebateJR.setEnabled(false);
2541              eventSettingsFinalsJR.setEnabled(false);
2542
2543              generalSettingsRoomsAvailable1.setEnabled(false);
2544              generalSettingsRoomsAvailable2.setEnabled(false);
2545              generalSettingsMaximalTime1.setEnabled(false);
2546              generalSettingsMaximalTime2.setEnabled(false);
2547              generalSettingsSchoolsNumber.setEnabled(false);
2548              generalSettingsTS.setEnabled(false);
2549              generalSettingsSS.setEnabled(false);
2550              generalSettingsCECheckBox.setEnabled(false);
2551              generalSettingsCEComboBox1.setEnabled(false);
2552              generalSettingsCEComboBox2.setEnabled(false); // </editor-fold>
2553
2554      settingsLoadButton.setEnabled(false);
2555      settingsResetButton.setEnabled(false);
2556
2557      participantsSaveToggleButton.setEnabled(true);
2558      participantsApproveToggleButton.setEnabled(true);
2559  }
2560
2561 /**
2562 * This method groups together the often-used group of statements that
2563 * enable elements of the settings tab. This method disables participants
2564 * and qualification, as well.
2565 *
2566 * @author Zbyněk Stara
2567 */
2568 private void enableSettingsTabActionBar() {
2569     this.updateOverallProgressBar(0);
2570
2571     // <editor-fold defaultstate="collapsed" desc="Enabling text fields for
2572     settings">
2573         eventSettingsOOSR.setEnabled(true);
2574         eventSettingsOISR.setEnabled(true);
2575         eventSettingsISSR.setEnabled(true);
2576         eventSettingsDASR.setEnabled(true);
2577         eventSettingsOOFinalSR.setEnabled(true);
2578         eventSettingsOIFinalSR.setEnabled(true);
2579         eventSettingsISFinalSR.setEnabled(true);
2580         eventSettingsDAFFinalSR.setEnabled(true);
2581
2582         eventSettingsOOSES.setEnabled(true);
2583         eventSettingsOISES.setEnabled(true);
2584         eventSettingsISSES.setEnabled(true);
2585         eventSettingsDASES.setEnabled(true);
2586         eventSettingsDebateSES.setEnabled(true);
2587
2588         eventSettingsOOJR.setEnabled(true);
2589         eventSettingsOIJR.setEnabled(true);
2590         eventSettingsISJR.setEnabled(true);
2591         eventSettingsDAJR.setEnabled(true);
2592         eventSettingsDebateJR.setEnabled(true);
2593         eventSettingsFinalsJR.setEnabled(true);

```

```

2595    //generalSettingsRoomsAvailable1.setEnabled(true);
2596    //generalSettingsRoomsAvailable2.setEnabled(true);
2597    //generalSettingsMaximalTime1.setEnabled(true);
2598    //generalSettingsMaximalTime2.setEnabled(true);
2599    generalSettingsSchoolsNumber.setEnabled(true);
2600    generalSettingsTS.setEnabled(true);
2601    generalSettingsSS.setEnabled(true);
2602    //generalSettingsCECheckBox.setSelected(true);
2603    //generalSettingsCEComboBox1.setSelected(true);
2604    //generalSettingsCEComboBox2.setSelected(true); // </editor-fold>
2605
2606    settingsLoadButton.setEnabled(true);
2607    settingsResetButton.setEnabled(true);
2608
2609    this.disableParticipantsTabActionBar();
2610
2611    this.disableQualificationTabActionBar();
2612
2613    settingsSaveToggleButton.setSelected(false);
2614    settingsSaveToggleButton.setEnabled(false);
2615
2616    settingsApproveToggleButton.setSelected(false);
2617}
2618
2619 /**
2620 * This method groups together the often-used group of statements that
2621 * disable elements of the participants tab.
2622 *
2623 * @author Zbyněk Stara
2624 */
2625 private void disableParticipantsTabActionBar() {
2626     participantsSchoolsTextField.setEnabled(false);
2627
2628     participantsSchoolsList.setSelectedIndex(-1);
2629     participantsSchoolsList.setEnabled(false); // needed - #8
2630     updateSchoolsListModel();
2631
2632     participantsSchoolsAddButton.setEnabled(false);
2633     participantsSchoolsRemoveButton.setEnabled(false);
2634     participantsSchoolsEditButton.setEnabled(false);
2635     participantsSchoolsSearchButton.setEnabled(false);
2636
2637     participantsTeachersTextField.setEnabled(false); // needed - #9
2638
2639     participantsTeachersList.setSelectedIndex(-1);
2640     participantsTeachersList.setEnabled(false);
2641     participantsTeachersList.setModel(noSelectionModel);
2642
2643     participantsTeachersAddButton.setEnabled(false);
2644     participantsTeachersRemoveButton.setEnabled(false);
2645     participantsTeachersSearchButton.setEnabled(false);
2646     participantsTeachersEditButton.setEnabled(false);
2647
2648     participantsStudentsTextField.setEnabled(false); // needed - #10
2649
2650     participantsStudentsList.setSelectedIndex(-1);
2651     participantsStudentsList.setEnabled(false);
2652     participantsStudentsList.setModel(noSelectionModel);
2653
2654     participantsStudentsAddButton.setEnabled(false);
2655     participantsStudentsRemoveButton.setEnabled(false);
2656     participantsStudentsSearchButton.setEnabled(false);
2657     participantsStudentsEditButton.setEnabled(false);
2658
2659     this.disableStudentCodesActionBar();
2660     participantsStudentCodesList.setModel(noSelectionModel);
2661
2662     this.disableAssignEventsActionBar();
2663
2664     participantsAssignPairsEventsList.setSelectedIndex(-1);
2665     participantsAssignPairsEventsList.setEnabled(false);
2666     participantsAssignPairsEventsList.setModel(eventsListModel);
2667
2668     participantsAssignPairsLeftList.setSelectedIndex(-1);
2669     participantsAssignPairsLeftList.setEnabled(false);
2670     participantsAssignPairsLeftList.setModel(blankModel);
2671
2672     participantsAssignPairsRightList.setSelectedIndex(-1);

```

```

    participantsAssignPairsRightList.setEnabled(false);
    participantsAssignPairsRightList.setModel(blankModel);

2675    participantsAssignPairsPairsList.setSelectedIndex(-1);
    participantsAssignPairsPairsList.setEnabled(false);
    participantsAssignPairsPairsList.setModel(blankModel);

2680    participantsAssignPairsPairButton.setEnabled(false);
    participantsAssignPairsRemovePairButton.setEnabled(false);

    participantsExportButton.setEnabled(false);
    participantsResetButton.setEnabled(false);

2685    participantsSaveToggleButton.setEnabled(false);
    participantsSaveToggleButton.setSelected(false);
}

2690 /**
 * This method groups together the often-used group of statements that
 * enable elements of the participants tab. This method disables settings
 * and qualification, as well.
 *
 * @author Zbyněk Stara
 */
private void enableParticipantsTabActionBar() {
    this.disableSettingsTabActionBar();

2700    this.disableQualificationTabActionBar();

    this.updateOverallProgressBar(33);

2705    this.updateSchoolsListModel();
    settingsApproveToggleButton.setEnabled(true);
    settingsApproveToggleButton.setSelected(true);

2710    settingsSaveToggleButton.setEnabled(true);
    settingsSaveToggleButton.setSelected(false);

    participantsSchoolsTextField.setEnabled(true);
    participantsSchoolsAddButton.setEnabled(true);

2715    if (schoolsNumber > 0) {
        participantsSchoolsList.setEnabled(true);
        participantsSchoolsSearchButton.setEnabled(true);
    }

2720    if (teachersNumber > 0) {
        participantsTeachersTextField.setEnabled(true);
        participantsTeachersSearchButton.setEnabled(true);
    }

2725    if (studentsNumber > 0) {
        participantsStudentsTextField.setEnabled(true);
        participantsStudentsSearchButton.setEnabled(true);
    }

2730    recheckStudentCodesActionBar();

    if (studentCodes) {
        participantsStudentCodesTextField.setEnabled(true);
        participantsStudentCodesSearchButton.setEnabled(true);

2735        participantsApproveToggleButton.setEnabled(true);
    }
    if (studentChanges) {
        participantsStudentCodesAssignCodesButton.setEnabled(true);
    }

2740    participantsExportButton.setEnabled(true);
    participantsResetButton.setEnabled(true);

2745    participantsSaveToggleButton.setSelected(false);
    participantsSaveToggleButton.setEnabled(false);
}

2750 /**
 * This method groups together the often-used group of statements that
 * disable elements of the qualification tab.

```

```

    *
    * @author Zbyněk Stara
    */
2755 private void disableQualificationTabActionBar() {
    qualificationEventsList.setModel(blankModel);
    qualificationEventsList.setEnabled(false);

    qualificationRoundsTextField.setText("");
    qualificationRoundsTextField.setEnabled(false);
    qualificationRoundsEditNameButton.setEnabled(false);

    qualificationRoomsList.setEnabled(false);
    //qualificationRoomList.setSelectedIndex(-1);
2765 qualificationRoomsList.setModel(blankModel);

    qualificationRoomsTextField.setText("");
    qualificationRoomsTextField.setEnabled(false);
    qualificationRoomsEditNameButton.setEnabled(false);

2770 qualificationJudgesList.setEnabled(false);
//qualificationJudgesList.setSelectedIndex(-1);
    qualificationJudgesList.setModel(blankModel);

    qualificationJudgesSubstituteJudgeButton.setEnabled(false);

    qualificationStudentCodesList.setEnabled(false);
//qualificationStudentCodesList.setSelectedIndex(-1);
2780 qualificationStudentCodesList.setModel(blankModel);

    qualificationStudentCodesTextField.setText("");
    qualificationStudentCodesTextField.setEnabled(false);
    qualificationStudentCodesSearchButton.setEnabled(false);

2785 qualificationExportButton.setEnabled(false);
    qualificationResetButton.setEnabled(false);

    qualificationSaveToggleButton.setEnabled(false);
    qualificationSaveToggleButton.setSelected(false);
2790 }

/**
 * This method groups together the often-used group of statements that
 * enable elements of the qualification tab. This method disables settings
 * and participants, as well.
 *
 * @author Zbyněk Stara
 */
2795 private void enableQualificationTabActionBar() {
    this.disableSettingsTabActionBar();
    this.disableParticipantsTabActionBar();

    settingsApproveToggleButton.setEnabled(true);
    settingsApproveToggleButton.setSelected(true);

2805    settingsSaveToggleButton.setEnabled(true);
    settingsSaveToggleButton.setSelected(false);

    participantsApproveToggleButton.setEnabled(true);
    participantsApproveToggleButton.setSelected(true);

    participantsSaveToggleButton.setEnabled(true);
    participantsSaveToggleButton.setSelected(false);

2810    this.updateEventsListModel();

    qualificationRoundsList.setModel(noSelectionModel);
    qualificationRoomsList.setModel(noSelectionModel);
    qualificationJudgesList.setModel(noSelectionModel);
2820    qualificationStudentCodesList.setModel(noSelectionModel);

    qualificationEventsList.setEnabled(true);

    qualificationExportButton.setEnabled(true);
    qualificationResetButton.setEnabled(true);
    qualificationSaveToggleButton.setEnabled(true);

2825    this.updateOverallProgressBar(100);
}

```

```

// OTHER ACTION BOXES:
/**
 * This method groups together the often-used group of statements that
 * automatically selects a school at a given index.
 *
 * @param schoolIndex the index of the school to be selected
 *
 * @author Zbyněk Stara
 */
2835 private void autoselectSchoolActionBar(int schoolIndex) {
    participantsSchoolsList.setEnabled(true);
    participantsSchoolsSearchButton.setEnabled(true);

    refreshSchoolActionBar(schoolIndex);
}

/**
 * This method groups together the often-used group of statements that
 * refreshes the school list (updates it and sets the selected school to be
 * the school at the specified index).
 *
 * @param schoolIndex the index of the school to be selected
 *
 * @author Zbyněk Stara
 */
2840 private void refreshSchoolActionBar(int schoolIndex) {
    this.updateSchoolsListModel();

    participantsSchoolsList.setSelectedIndex(schoolIndex);
    participantsSchoolsList.ensureIndexIsVisible(schoolIndex);
}

/**
 * This method groups together the often-used group of statements that
 * rechecks the current student codes status and enables/disables GUI
 * elements as necessary.
 *
 * @author Zbyněk Stara
 */
2845 private void recheckStudentCodesActionBar() {
    if (studentCodes && !studentChanges) {
        participantsStudentCodesTextField.setEnabled(true);
        participantsStudentCodesSearchButton.setEnabled(true);

        participantsApproveToggleButton.setEnabled(true);
    } else {
        participantsStudentCodesTextField.setEnabled(false);
        participantsStudentCodesSearchButton.setEnabled(false);

        participantsApproveToggleButton.setEnabled(false);
    }

    if (studentChanges && studentsNumber > 0) {
        participantsStudentCodesAssignCodesButton.setEnabled(true);
    } else {
        participantsStudentCodesAssignCodesButton.setEnabled(false);
    }
}

/**
 * This method groups together the often-used group of statements that
 * toggle the search-related GUI elements of students.
 *
 * @param state the boolean value specifying whether the elements should be
 * enabled or disabled
 *
 * @author Zbyněk Stara
 */
2850 private void toggleStudentSearchActionBar(boolean state) {
    participantsStudentsSearchButton.setEnabled(state);
    participantsStudentsTextField.setEnabled(state);
}

/**
 * This method groups together the often-used group of statements that
 * disables the student-codes-related GUI elements.
 *
 * @author Zbyněk Stara
 */
2855

```

```

2910     private void disableStudentCodesActionBar() {
2911         participantsStudentCodesTextField.setEnabled(false);
2912         participantsStudentCodesList.setEnabled(false);
2913         participantsStudentCodesSearchButton.setEnabled(false);
2914         participantsStudentCodesAssignCodesButton.setEnabled(false);
2915     }
2916
2917     /**
2918      * This method groups together the often-used group of statements that
2919      * disables the assign-events-related GUI elements.
2920      *
2921      * @author Zbyněk Stara
2922      */
2923     private void disableAssignEventsActionBar() {
2924         participantsAssignEventsOriginalOratoryCheckBox.setEnabled(false);
2925         participantsAssignEventsOriginalOratoryCheckBox.setSelected(false);
2926
2927         participantsAssignEventsOralInterpretationCheckBox.setEnabled(false);
2928         participantsAssignEventsOralInterpretationCheckBox.setSelected(false);
2929
2930         participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(false);
2931         participantsAssignEventsImpromptuSpeakingCheckBox.setSelected(false);
2932
2933         participantsAssignEventsDuetActingCheckBox.setEnabled(false);
2934         participantsAssignEventsDuetActingCheckBox.setSelected(false);
2935
2936         participantsAssignEventsDebateCheckBox.setEnabled(false);
2937         participantsAssignEventsDebateCheckBox.setSelected(false);
2938
2939         participantsAssignEventsAssignEventsButton.setEnabled(false);
2940     }
2941
2942     /**
2943      * This method is called upon to export the participants tab into a
2944      * plaintext file. The user can then copy-paste the raw text and format it
2945      * as necessary for use in brochures and information materials for the
2946      * tournament
2947      *
2948      * @throws UserIOException if the user cancels the file chooser dialog
2949      * @throws IOException if a generic input/output exception has occurred
2950      *
2951      * @author Zbyněk Stara
2952      */
2953     private void exportParticipantsActionBar() throws UserIOException, IOException {
2954         JFileChooser jfc = new JFileChooser();
2955
2956         int saveDialogReturn = jfc.showSaveDialog(this);
2957
2958         try {
2959             if (saveDialogReturn == JFileChooser.APPROVE_OPTION) {
2960                 BufferedWriter bw = new BufferedWriter(new
2961                     FileWriter(jfc.getSelectedFile()));
2962
2963                 for (int i = 0; i < database.getSchoolTreeSize(); i++) {
2964                     School currentSchool = (School) database.getSchoolTreeNodeData(i);
2965
2966                     bw.write(currentSchool.getName() + "\n");
2967
2968                     for (int j = 0; j < currentSchool.getTeacherTreeSize(); j++) {
2969                         Teacher currentTeacher = (Teacher)
2970                             currentSchool.getTeacherTreeNodeData(j);
2971
2972                         bw.write("\t" + currentTeacher.getName() + "\n");
2973                     }
2974
2975                     bw.write("\n");
2976
2977                     for (int j = 0; j < currentSchool.getStudentTreeSize(); j++) {
2978                         Student currentStudent = (Student)
2979                             currentSchool.getStudentTreeNodeData(j);
2980
2981                         bw.write("\t" + currentStudent.getCode() + "\t" +
2982                             currentStudent.getName() + "\n");
2983                     }
2984
2985                     bw.write("\n");
2986                 }
2987
2988                 bw.close();
2989             }
2990
2991         }
2992
2993     }

```

```

2990             } else if (saveDialogReturn == JFileChooser.CANCEL_OPTION) {
2991                 throw new UserIOException();
2992             } else {
2993                 throw new IOException();
2994             }
2995         } catch (UserIOException ex) {
2996             // do nothing
2997         } catch (IOException ex) {
2998             exceptionDialogTextArea.setText("An unexpected error occurred "
2999                     + "during the accessing of the specified file. Data was "
3000                     + "not exported. Please try again.");
3001             exceptionDialog.setVisible(true);
3002         }
3003     }
3004
3005 /**
3006 * This method is called upon to export the qualification tab into a
3007 * plaintext file. The user can then copy-paste the raw text and format it
3008 * as necessary for use in brochures and information materials for the
3009 * tournament
3010 *
3011 * @throws UserIOException if the user cancels the file chooser dialog
3012 * @throws IOException if a generic input/output exception has occurred
3013 *
3014 * @author Zbyněk Stara
3015 */
3016 private void exportQualificationActionBar() throws UserIOException, IOException {
3017     JFileChooser jfc = new JFileChooser();
3018
3019     int saveDialogReturn = jfc.showSaveDialog(this);
3020
3021     try {
3022         if (saveDialogReturn == JFileChooser.APPROVE_OPTION) {
3023             BufferedWriter bw = new BufferedWriter(new
3024             FileWriter(jfc.getSelectedFile()));
3025
3026             for (int i = 0; i < qualification.getEventArrayLength(); i++) {
3027                 Event currentEvent = qualification.getEventArrayElement(i);
3028
3029                 bw.write(currentEvent.type.getEventName() + "\n");
3030
3031                 for (int j = 0; j < currentEvent.getRoundArrayLength(); j++) {
3032                     Round currentRound = (Round)
3033                     currentEvent.getRoundArrayElement(j);
3034
3035                     if (currentRound != null) {
3036                         bw.write("\t" + currentRound.getName() + "\n");
3037
3038                         for (int k = 0; k < currentRound.getRoomArrayLength();
3039                             k++) {
3040                             Room currentRoom = (Room)
3041                             currentRound.getRoomArrayElement(k);
3042
3043                             bw.write("\t\t" + currentRoom.getName() + "\n");
3044
3045                             for (int l = 0; l < currentRoom.getJudgeTreeSize();
3046                                 l++) {
3047                                 Judge currentJudge = (Judge)
3048                                 currentRoom.getJudgeTreeNodeData(l);
3049
3050                                 bw.write("\t\t\t" + currentJudge.getName() +
3051                                         "\n");
3052                             }
3053
3054                             bw.write("\n");
3055
3056                         for (int l = 0; l < currentRoom.getEntityTreeSize();
3057                             l++) {
3058                             Entity currentEntity = (Entity)
3059                             currentRoom.getEntityTreeNodeData(l);
3060
3061                             bw.write("\t\t\t" + currentEntity.getCode() +
3062                                     "\n");
3063                         }
3064
3065                         bw.write("\n");
3066                     }
3067                 }
3068             }
3069         }
3070     }

```

```

                                bw.write("\n");
3070                         } else {
                                continue;
                            }
                        }
                    }

3075                 bw.write("\n");
                }

            bw.close();
        } else if (saveDialogReturn == JFileChooser.CANCEL_OPTION) {
            throw new UserIOException();
        } else {
            throw new IOException();
        }
    } catch (UserIOException ex) {
        // do nothing
    } catch (IOException ex) {
        exceptionDialogTextArea.setText("An unexpected error occurred "
            + "during the accessing of the specified file. Data was "
            + "not exported. Please try again.");
3090
        exceptionDialog.setVisible(true);
    }
}

3095     @SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {
    settingsAcceptDialog = new javax.swing.JDialog();
    jLabel61 = new javax.swing.JLabel();
    jLabel62 = new javax.swing.JLabel();
    settingsAcceptDialogConfirmButton = new javax.swing.JButton();
    settingsErrorDialog = new javax.swing.JDialog();
    jLabel63 = new javax.swing.JLabel();
    jLabel64 = new javax.swing.JLabel();
    settingsErrorDialogConfirmButton = new javax.swing.JButton();
3100    settingsErrorDialogScrollPane = new javax.swing.JScrollPane();
    settingsErrorDialogTextArea = new javax.swing.JTextArea();
    exceptionDialog = new javax.swing.JDialog();
    jLabel65 = new javax.swing.JLabel();
    jLabel66 = new javax.swing.JLabel();
    exceptionDialogConfirmButton = new javax.swing.JButton();
    exceptionDialogScrollPane = new javax.swing.JScrollPane();
    exceptionDialogTextArea = new javax.swing.JTextArea();
3105    illegalActionDialog = new javax.swing.JDialog();
    jLabel67 = new javax.swing.JLabel();
    jLabel68 = new javax.swing.JLabel();
    illegalActionDialogConfirmButton = new javax.swing.JButton();
    illegalActionDialogScrollPane = new javax.swing.JScrollPane();
    illegalActionDialogTextArea = new javax.swing.JTextArea();
    resetWarningDialog = new javax.swing.JDialog();
3110    jLabel69 = new javax.swing.JLabel();
    jLabel70 = new javax.swing.JLabel();
    resetWarningDialogConfirmButton = new javax.swing.JButton();
    resetWarningDialogScrollPane = new javax.swing.JScrollPane();
    resetWarningDialogTextArea = new javax.swing.JTextArea();
    resetWarningDialogCancelButton = new javax.swing.JButton();
3115    approveParticipantsDialog = new javax.swing.JDialog();
    jLabel71 = new javax.swing.JLabel();
    jLabel72 = new javax.swing.JLabel();
    approveParticipantsDialogIgnoreButton = new javax.swing.JButton();
    approveParticipantsDialogScrollPane = new javax.swing.JScrollPane();
    approveParticipantsDialogTextArea = new javax.swing.JTextArea();
    approveParticipantsDialogCancelButton = new javax.swing.JButton();
3120    allocationInfoDialog = new javax.swing.JDialog();
    jLabel73 = new javax.swing.JLabel();
    allocationInfoProgressBar = new javax.swing.JProgressBar();
    allocationInfoTextField = new javax.swing.JTextField();
    tabbedPane = new javax.swing.JTabbedPane();
    participantsPanel = new javax.swing.JPanel();
3125    jLabel137 = new javax.swing.JLabel();
    eventSettingsOOSR = new javax.swing.JTextField();
    jLabel138 = new javax.swing.JLabel();
    eventSettingsOOSES = new javax.swing.JTextField();
    jLabel139 = new javax.swing.JLabel();
    eventSettingsOOJR = new javax.swing.JTextField();
3130
3135
3140
3145

```

```
3150 settingsResetButton = new javax.swing.JButton();
settingsLoadButton = new javax.swing.JButton();
settingsApproveToggleButton = new javax.swing.JToggleButton();
settingsSaveToggleButton = new javax.swing.JToggleButton();
eventSettingsOOFFinalSR = new javax.swing.JTextField();
generalSettingsRoomsAvailable1 = new javax.swing.JTextField();
generalSettingsMaximalTime1 = new javax.swing.JTextField();
generalSettingsSchoolsNumber = new javax.swing.JTextField();
jLabel142 = new javax.swing.JLabel();
jLabel143 = new javax.swing.JLabel();
jLabel144 = new javax.swing.JLabel();
jLabel145 = new javax.swing.JLabel();
jLabel146 = new javax.swing.JLabel();
jLabel147 = new javax.swing.JLabel();
jLabel148 = new javax.swing.JLabel();
jLabel149 = new javax.swing.JLabel();
jLabel150 = new javax.swing.JLabel();
jLabel151 = new javax.swing.JLabel();
jLabel152 = new javax.swing.JLabel();
jLabel153 = new javax.swing.JLabel();
jLabel154 = new javax.swing.JLabel();
jLabel155 = new javax.swing.JLabel();
eventSettingsDAFFinalSR = new javax.swing.JTextField();
generalSettingsSS = new javax.swing.JTextField();
generalSettingsTS = new javax.swing.JTextField();
jLabel157 = new javax.swing.JLabel();
eventSettingsOISR = new javax.swing.JTextField();
eventSettingsISSR = new javax.swing.JTextField();
eventSettingsDASR = new javax.swing.JTextField();
eventSettingsOISES = new javax.swing.JTextField();
eventSettingsISSES = new javax.swing.JTextField();
eventSettingsDASES = new javax.swing.JTextField();
eventSettingsDebateSES = new javax.swing.JTextField();
eventSettingsOIJR = new javax.swing.JTextField();
eventSettingsISJR = new javax.swing.JTextField();
eventSettingsDAJR = new javax.swing.JTextField();
eventSettingsFinalsJR = new javax.swing.JTextField();
eventSettingsDebateJR = new javax.swing.JTextField();
jLabel136 = new javax.swing.JLabel();
jLabel156 = new javax.swing.JLabel();
jLabel158 = new javax.swing.JLabel();
jLabel159 = new javax.swing.JLabel();
eventSettingsOIFinalSR = new javax.swing.JTextField();
eventSettingsISFinalSR = new javax.swing.JTextField();
generalSettingsRoomsAvailable2 = new javax.swing.JTextField();
generalSettingsMaximalTime2 = new javax.swing.JTextField();
jLabel160 = new javax.swing.JLabel();
generalSettingsCEComboBox2 = new javax.swing.JComboBox();
generalSettingsCEComboBox1 = new javax.swing.JComboBox();
generalSettingsCECheckBox = new javax.swing.JCheckBox();
participantsPane = new javax.swing.JPanel();
jLabel2 = new javax.swing.JLabel();
participantsSchoolsTextField = new javax.swing.JTextField();
jScrollPane = new javax.swing.JScrollPane();
participantsSchoolsList = new javax.swing.JList();
participantsSchoolsAddButton = new javax.swing.JButton();
participantsSchoolsRemoveButton = new javax.swing.JButton();
participantsSchoolsEditButton = new javax.swing.JButton();
participantsSchoolsSearchButton = new javax.swing.JButton();
jLabel13 = new javax.swing.JLabel();
participantsTeachersTextField = new javax.swing.JTextField();
jScrollPane2 = new javax.swing.JScrollPane();
participantsTeachersList = new javax.swing.JList();
participantsTeachersAddButton = new javax.swing.JButton();
participantsTeachersEditButton = new javax.swing.JButton();
participantsTeachersSearchButton = new javax.swing.JButton();
participantsTeachersRemoveButton = new javax.swing.JButton();
jLabel4 = new javax.swing.JLabel();
participantsStudentsTextField = new javax.swing.JTextField();
jScrollPane3 = new javax.swing.JScrollPane();
participantsStudentsList = new javax.swing.JList();
participantsStudentsAddButton = new javax.swing.JButton();
participantsStudentsEditButton = new javax.swing.JButton();
participantsStudentsSearchButton = new javax.swing.JButton();
participantsStudentsRemoveButton = new javax.swing.JButton();
jLabel5 = new javax.swing.JLabel();
participantsStudentCodesTextField = new javax.swing.JTextField();
jScrollPane4 = new javax.swing.JScrollPane();
participantsStudentCodesList = new javax.swing.JList();
```

```

    participantsStudentCodesSearchButton = new javax.swing.JButton();
    jLabel6 = new javax.swing.JLabel();
    participantsAssignEventsAssignEventsButton = new javax.swing.JButton();
    jLabel7 = new javax.swing.JLabel();
    jScrollPane6 = new javax.swing.JScrollPane();
    participantsAssignPairsEventsList = new javax.swing.JList();
    participantsAssignPairsRemovePairButton = new javax.swing.JButton();
    participantsAssignPairsPairButton = new javax.swing.JButton();
    participantsResetButton = new javax.swing.JButton();
    participantsExportButton = new javax.swing.JButton();
    participantsApproveToggleButton = new javax.swing.JToggleButton();
    participantsSaveToggleButton = new javax.swing.JToggleButton();
    participantsAssignEventsOriginalOratoryCheckBox = new javax.swing.JCheckBox();
    participantsAssignEventsOralInterpretationCheckBox = new
3230    javax.swing.JCheckBox();
    participantsAssignEventsImpromptuSpeakingCheckBox = new
    javax.swing.JCheckBox();
    participantsAssignEventsDuetActingCheckBox = new javax.swing.JCheckBox();
    participantsAssignEventsDebateCheckBox = new javax.swing.JCheckBox();
    jScrollPane5 = new javax.swing.JScrollPane();
    participantsAssignPairsLeftList = new javax.swing.JList();
    jScrollPane31 = new javax.swing.JScrollPane();
    participantsAssignPairsRightList = new javax.swing.JList();
    jScrollPane32 = new javax.swing.JScrollPane();
    participantsAssignPairsPairsList = new javax.swing.JList();
    participantsStudentCodesAssignCodesButton = new javax.swing.JButton();
    participantsProgressBar = new javax.swing.JProgressBar();
    qualificationPane = new javax.swing.JPanel();
    jLabel8 = new javax.swing.JLabel();
    jScrollPane7 = new javax.swing.JScrollPane();
    qualificationEventsList = new javax.swing.JList();
    qualificationProgressBar = new javax.swing.JProgressBar();
    jLabel9 = new javax.swing.JLabel();
    qualificationRoundsTextField = new javax.swing.JTextField();
    jScrollPane8 = new javax.swing.JScrollPane();
    qualificationRoundsList = new javax.swing.JList();
    qualificationRoundsEditNameButton = new javax.swing.JButton();
    jLabel10 = new javax.swing.JLabel();
    qualificationRoomsTextField = new javax.swing.JTextField();
    jScrollPane9 = new javax.swing.JScrollPane();
    qualificationRoomsList = new javax.swing.JList();
    qualificationRoomsEditNameButton = new javax.swing.JButton();
    jLabel11 = new javax.swing.JLabel();
    jScrollPane10 = new javax.swing.JScrollPane();
    qualificationJudgesList = new javax.swing.JList();
    qualificationJudgesSubstituteJudgeButton = new javax.swing.JButton();
    jLabel12 = new javax.swing.JLabel();
    qualificationStudentCodesTextField = new javax.swing.JTextField();
    jScrollPane11 = new javax.swing.JScrollPane();
    qualificationStudentCodesList = new javax.swing.JList();
    qualificationStudentCodesSearchButton = new javax.swing.JButton();
    jLabel13 = new javax.swing.JLabel();
    qualificationScoresTextField = new javax.swing.JTextField();
    jScrollPane12 = new javax.swing.JScrollPane();
    qualificationScoresList = new javax.swing.JList();
    qualificationScoresAddButton = new javax.swing.JButton();
    qualificationScoresRemoveButton = new javax.swing.JButton();
    qualificationResetButton = new javax.swing.JButton();
    qualificationExportButton = new javax.swing.JButton();
    qualificationSaveToggleButton = new javax.swing.JToggleButton();
    qualificationApproveIncompleteToggleButton = new javax.swing.JToggleButton();
    qualificationApproveCompleteToggleButton = new javax.swing.JToggleButton();
    finalistsPane = new javax.swing.JPanel();
    jLabel14 = new javax.swing.JLabel();
    jScrollPane13 = new javax.swing.JScrollPane();
    finalistsOriginalOratoryList = new javax.swing.JList();
    finalistsProgressBar = new javax.swing.JProgressBar();
    jLabel15 = new javax.swing.JLabel();
    jScrollPane14 = new javax.swing.JScrollPane();
    finalistsOralInterpretationList = new javax.swing.JList();
    jLabel16 = new javax.swing.JLabel();
    jScrollPane15 = new javax.swing.JScrollPane();
    finalistsImpromptuSpeakingList = new javax.swing.JList();
    jLabel17 = new javax.swing.JLabel();
    jScrollPane16 = new javax.swing.JScrollPane();
    finalistsDuetActingList = new javax.swing.JList();
    jLabel18 = new javax.swing.JLabel();
    jScrollPane17 = new javax.swing.JScrollPane();
    finalistsDebateSemifinalistsAList = new javax.swing.JList();
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300

```

```

3305     finalistsDebateSemifinalistsSetWinnerAButton = new javax.swing.JButton();
3306     jLabel19 = new javax.swing.JLabel();
3307     jScrollPane18 = new javax.swing.JScrollPane();
3308     finalistsDebateFinalistsList = new javax.swing.JList();
3309     finalistsResetButton = new javax.swing.JButton();
3310     finalistsExportButton = new javax.swing.JButton();
3311     finalistsApproveToggleButton = new javax.swing.JToggleButton();
3312     finalistsSaveToggleButton = new javax.swing.JToggleButton();
3313     jScrollPane33 = new javax.swing.JScrollPane();
3314     finalistsDebateSemifinalistsBList = new javax.swing.JList();
3315     finalistsDebateSemifinalistsSetWinnerBButton = new javax.swing.JButton();
3316     finalsPane = new javax.swing.JPanel();
3317     jLabel20 = new javax.swing.JLabel();
3318     jScrollPane19 = new javax.swing.JScrollPane();
3319     finalsEventList = new javax.swing.JList();
3320     finalsProgressBar = new javax.swing.JProgressBar();
3321     jLabel21 = new javax.swing.JLabel();
3322     finalsRoundTextField = new javax.swing.JTextField();
3323     jScrollPane20 = new javax.swing.JScrollPane();
3324     finalsRoundList = new javax.swing.JList();
3325     finalsRoundEditNameButton = new javax.swing.JButton();
3326     jLabel22 = new javax.swing.JLabel();
3327     finalsRoomTextField = new javax.swing.JTextField();
3328     jScrollPane21 = new javax.swing.JScrollPane();
3329     finalsRoomList = new javax.swing.JList();
3330     finalsRoomEditNameButton = new javax.swing.JButton();
3331     jLabel23 = new javax.swing.JLabel();
3332     jScrollPane22 = new javax.swing.JScrollPane();
3333     finalsJudgesList = new javax.swing.JList();
3334     finalsJudgesSubstituteJudgeButton = new javax.swing.JButton();
3335     jLabel24 = new javax.swing.JLabel();
3336     finalsStudentCodesTextField = new javax.swing.JTextField();
3337     jScrollPane23 = new javax.swing.JScrollPane();
3338     finalsStudentCodesList = new javax.swing.JList();
3339     finalsStudentCodesSearchButton = new javax.swing.JButton();
3340     jLabel25 = new javax.swing.JLabel();
3341     finalsRankingsTextField = new javax.swing.JTextField();
3342     jScrollPane24 = new javax.swing.JScrollPane();
3343     finalsRankingsList = new javax.swing.JList();
3344     finalsRankingsAddButton = new javax.swing.JButton();
3345     finalsRankingsRemoveButton = new javax.swing.JButton();
3346     finalsResetButton = new javax.swing.JButton();
3347     finalsExportButton = new javax.swing.JButton();
3348     finalsApproveToggleButton = new javax.swing.JToggleButton();
3349     finalsSaveToggleButton = new javax.swing.JToggleButton();
3350     jScrollPane34 = new javax.swing.JScrollPane();
3351     finalsStudentsVoteList = new javax.swing.JList();
3352     finalsStudentsVoteSetStudentsVoteButton = new javax.swing.JButton();
3353     winnersPane = new javax.swing.JPanel();
3354     jLabel27 = new javax.swing.JLabel();
3355     jScrollPane26 = new javax.swing.JScrollPane();
3356     winnersOriginalOratoryFirstPlaceList = new javax.swing.JList();
3357     jLabel28 = new javax.swing.JLabel();
3358     jLabel29 = new javax.swing.JLabel();
3359     jLabel30 = new javax.swing.JLabel();
3360     jLabel31 = new javax.swing.JLabel();
3361     winnersSaveToggleButton = new javax.swing.JToggleButton();
3362     jScrollPane35 = new javax.swing.JScrollPane();
3363     winnersOriginalOratorySecondPlaceList = new javax.swing.JList();
3364     jScrollPane36 = new javax.swing.JScrollPane();
3365     winnersOriginalOratoryThirdPlaceList = new javax.swing.JList();
3366     jScrollPane37 = new javax.swing.JScrollPane();
3367     winnersOriginalOratoryStudentChoiceList = new javax.swing.JList();
3368     jScrollPane27 = new javax.swing.JScrollPane();
3369     winnersOralInterpretationFirstPlaceList = new javax.swing.JList();
3370     jScrollPane38 = new javax.swing.JScrollPane();
3371     winnersOralInterpretationSecondPlaceList = new javax.swing.JList();
3372     jScrollPane39 = new javax.swing.JScrollPane();
3373     winnersOralInterpretationThirdPlaceList = new javax.swing.JList();
3374     jScrollPane40 = new javax.swing.JScrollPane();
3375     winnersOralInterpretationStudentChoiceList = new javax.swing.JList();
3376     jScrollPane28 = new javax.swing.JScrollPane();
3377     winnersImpromptuSpeakingFirstPlaceList = new javax.swing.JList();
3378     jScrollPane41 = new javax.swing.JScrollPane();
3379     winnersImpromptuSpeakingSecondPlaceList = new javax.swing.JList();
3380     jScrollPane42 = new javax.swing.JScrollPane();
3381     winnersImpromptuSpeakingThirdPlaceList = new javax.swing.JList();
3382     jScrollPane43 = new javax.swing.JScrollPane();
3383     winnersImpromptuSpeakingStudentChoiceList = new javax.swing.JList();

```

```

3385     jScrollPane29 = new javax.swing.JScrollPane();
winnersDuetActingFirstPlaceList = new javax.swing.JList();
jScrollPane44 = new javax.swing.JScrollPane();
winnersDuetActingSecondPlaceList = new javax.swing.JList();
jScrollPane45 = new javax.swing.JScrollPane();
winnersDuetActingThirdPlaceList = new javax.swing.JList();
jScrollPane46 = new javax.swing.JScrollPane();
winnersDuetActingStudentChoiceList = new javax.swing.JList();
jScrollPane30 = new javax.swing.JScrollPane();
winnersDebateFirstPlaceList = new javax.swing.JList();
jScrollPane47 = new javax.swing.JScrollPane();
winnersDebateSecondPlaceList = new javax.swing.JList();
jScrollPane48 = new javax.swing.JScrollPane();
winnersDebateThirdPlaceList = new javax.swing.JList();
jScrollPane49 = new javax.swing.JScrollPane();
winnersDebateStudentChoiceList = new javax.swing.JList();
jLabel32 = new javax.swing.JLabel();
jLabel33 = new javax.swing.JLabel();
jLabel34 = new javax.swing.JLabel();
jLabel35 = new javax.swing.JLabel();
jLabel26 = new javax.swing.JLabel();
winnersExportButton = new javax.swing.JButton();
jLabel1 = new javax.swing.JLabel();
jLabel40 = new javax.swing.JLabel();
jLabel41 = new javax.swing.JLabel();
overallProgressBar = new javax.swing.JProgressBar();
menuBar = new javax.swing.JMenuBar();
fileMenuItem = new javax.swing.JMenuItem();
openMenuItem = new javax.swing.JMenuItem();
saveMenuItem = new javax.swing.JMenuItem();
saveAsMenuItem = new javax.swing.JMenuItem();
exitMenuItem = new javax.swing.JMenuItem();
editMenu = new javax.swing.JMenu();
cutMenuItem = new javax.swing.JMenuItem();
copyMenuItem = new javax.swing.JMenuItem();
pasteMenuItem = new javax.swing.JMenuItem();
deleteMenuItem = new javax.swing.JMenuItem();
helpMenu = new javax.swing.JMenu();
contentsMenuItem = new javax.swing.JMenuItem();
aboutMenuItem = new javax.swing.JMenuItem();

3425     settingsAcceptDialog.setTitle("Information");
settingsAcceptDialog.setAlwaysOnTop(true);
settingsAcceptDialog.setIconImage(null);
settingsAcceptDialog.setLocationByPlatform(true);

3430     settingsAcceptDialog.setModalityType(java.awt.Dialog.ModalityType.APPLICATION_MODAL);
settingsAcceptDialog.setResizable(false);
settingsAcceptDialog.setSize(new java.awt.Dimension(411, 146));

3435         jLabel61.setText("No errors were found while checking the values.");
jLabel62.setText("You are free to proceed to the next section of the
program.");

3440         settingsAcceptDialogConfirmButton.setText("Confirm");
settingsAcceptDialogConfirmButton.addMouseListener(new
java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        settingsAcceptDialogConfirmButtonMouseReleased(evt);
    }
});

3445     });

3450         org.jdesktop.layout.GroupLayout settingsAcceptDialogLayout = new
org.jdesktop.layout.GroupLayout(settingsAcceptDialog.getContentPane());
settingsAcceptDialog.getContentPane().setLayout(settingsAcceptDialogLayout);
settingsAcceptDialogLayout.setHorizontalGroup(
settingsAcceptDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
    .add(settingsAcceptDialogLayout.createSequentialGroup()
        .add(settingsAcceptDialogLayout.createSequentialGroup()
            .add(settingsAcceptDialogLayout.createParallelGroup()
                .addParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                    .add(settingsAcceptDialogLayout.createSequentialGroup()
                        .add(settingsAcceptDialogLayout.createSequentialGroup()
                            .add(settingsAcceptDialogLayout.createSequentialGroup()
                                .addParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                                    .add(settingsAcceptDialogLayout.createSequentialGroup()
                                        .add(20, 20, 20)
                                            .add(jLabel61, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                                .add(settingsAcceptDialogLayout.createSequentialGroup()
                                                    .addSequentialGroup()

```

```

3465                               .add(20, 20, 20)
3466                               .add(jLabel162))
3467                         .add(settingsAcceptDialogLayout.createSequentialGroup()
3468                             .add(155, 155, 155)
3469                             .add(settingsAcceptDialogConfirmButton)))
3470               .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
3471 Short.MAX_VALUE))
3472     );
3473     settingsAcceptDialogLayout.setVerticalGroup(
3474
3475       settingsAcceptDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
3476           .add(settingsAcceptDialogLayout.createSequentialGroup()
3477               .add(20, 20, 20)
3478               .add(jLabel61)
3479               .add(8, 8, 8)
3480               .add(jLabel62)
3481               .add(18, 18, 18)
3482               .add(settingsAcceptDialogConfirmButton)))
3483   );
3484
3485     settingsErrorDialog.setTitle("Error");
3486     settingsErrorDialog.setAlwaysOnTop(true);
3487     settingsErrorDialog.setLocationByPlatform(true);
3488     settingsErrorDialog.setMinimumSize(new java.awt.Dimension(411, 325));
3489
3490     settingsErrorDialog.setModalityType(java.awt.Dialog.ModalityType.APPLICATION_MODAL);
3491     settingsErrorDialog.setResizable(false);
3492
3493     jLabel63.setText("There are errors in some of the settings.");
3494
3495     jLabel64.setText("These need to be corrected before being able to proceed:");
3496
3497     settingsErrorDialogConfirmButton.setText("Confirm");
3498     settingsErrorDialogConfirmButton.addMouseListener(new
3499 java.awt.event.MouseAdapter() {
3500         public void mouseReleased(java.awt.event.MouseEvent evt) {
3501             settingsErrorDialogConfirmButtonMouseReleased(evt);
3502         }
3503     });
3504
3505     settingsErrorDialogTextArea.setColumns(20);
3506     settingsErrorDialogTextArea.setEditable(false);
3507     settingsErrorDialogTextArea.setLineWrap(true);
3508     settingsErrorDialogTextArea.setRows(5);
3509     settingsErrorDialogTextArea.setWrapStyleWord(true);
3510     settingsErrorDialogScrollPane.setViewportView(settingsErrorDialogTextArea);
3511
3512     org.jdesktop.layout.GroupLayout settingsErrorDialogLayout = new
3513 org.jdesktop.layout.GroupLayout(settingsErrorDialog.getContentPane());
3514     settingsErrorDialog.getContentPane().setLayout(settingsErrorDialogLayout);
3515     settingsErrorDialogLayout.setHorizontalGroup(
3516         settingsErrorDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
3517             .add(settingsErrorDialogLayout.createSequentialGroup()
3518                 .add(settingsErrorDialogLayout.createSequentialGroup()
3519                     .add(settingsErrorDialogLayout.createSequentialGroup()
3520                         .add(20, 20, 20)
3521                         .add(jLabel63, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
3522 371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
3523                         .add(settingsErrorDialogLayout.createSequentialGroup()
3524                             .add(20, 20, 20)
3525                             .add(jLabel64, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
3526 371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
3527                         .add(settingsErrorDialogLayout.createSequentialGroup()
3528                             .add(20, 20, 20)
3529                             .add(settingsErrorDialogScrollPane,
3530 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 371,
3531 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
3532                         .add(settingsErrorDialogLayout.createSequentialGroup()
3533                             .add(20, 20, 20)
3534                             .add(settingsErrorDialogConfirmButton,
3535 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
3536 157, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
3537                         .addContainerGap(20, Short.MAX_VALUE))
3538     );
3539     settingsErrorDialogLayout.setVerticalGroup(
3540       settingsErrorDialogLayout.createSequentialGroup()
3541         .add(settingsErrorDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING))

```

```

    .add(settingsErrorDialogLayout.createSequentialGroup()
        .add(20, 20, 20)
        .add(jLabel63)
        .add(8, 8, 8)
        .add(jLabel64)
        .add(18, 18, 18)
        .add(settingsErrorDialogScrollPane,
            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 161,
            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
            .add(18, 18, 18)
            .add(settingsErrorDialogConfirmButton)
            .addContainerGap(39, Short.MAX_VALUE))
    );
};

3555 exceptionDialog.setTitle("Exception");
exceptionDialog.setAlwaysOnTop(true);
exceptionDialog.setLocationByPlatform(true);
exceptionDialog.setMinimumSize(new java.awt.Dimension(411, 325));
3560 exceptionDialog.setModalityType(java.awt.Dialog.ModalityType.APPLICATION_MODAL);
exceptionDialog.setResizable(false);

3565 jLabel65.setText("An unexpected exception occurred.");
jLabel66.setText("Specification and tips for resolution are shown below:");

3570 exceptionDialogConfirmButton.setText("Confirm");
exceptionDialogConfirmButton.addMouseListener(new
java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        exceptionDialogConfirmButtonMouseReleased(evt);
    }
});
};

3575 exceptionDialogTextArea.setColumns(20);
exceptionDialogTextArea.setEditable(false);
exceptionDialogTextArea.setLineWrap(true);
exceptionDialogTextArea.setRows(5);
exceptionDialogTextArea.setWrapStyleWord(true);
exceptionDialogScrollPane.setViewportView(exceptionDialogTextArea);

3580 org.jdesktop.layout.GroupLayout exceptionDialogLayout = new
org.jdesktop.layout.GroupLayout(exceptionDialog.getContentPane());
exceptionDialog.getContentPane().setLayout(exceptionDialogLayout);
exceptionDialogLayout.setHorizontalGroup(
    exceptionDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(exceptionDialogLayout.createSequentialGroup()
            .add(exceptionDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(exceptionDialogLayout.createSequentialGroup()
                    .add(20, 20, 20)
                    .add(jLabel65, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                        371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(exceptionDialogLayout.createSequentialGroup()
                        .add(20, 20, 20)
                        .add(jLabel66, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                            371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(exceptionDialogScrollPane,
                            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 371,
                            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(exceptionDialogLayout.createSequentialGroup()
                            .add(20, 20, 20)
                            .add(exceptionDialogLayout.createSequentialGroup()
                                .add(157, 157, 157)
                                .add(exceptionDialogConfirmButton)))
                            .addContainerGap(20, Short.MAX_VALUE))
            );
        exceptionDialogLayout.setVerticalGroup(
            exceptionDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(20, 20, 20)
                .add(jLabel65)
                .add(8, 8, 8)
                .add(jLabel66)
                .add(18, 18, 18)
        );
    );
}
3610
3615

```

```

3620           .add(exceptionDialogScrollPane,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 161,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
          .add(18, 18, 18)
          .add(exceptionDialogConfirmButton)
          .addContainerGap(39, Short.MAX_VALUE))
      );
3625
3630     illegalActionDialog.setTitle("Illegal Action");
illegalActionDialog.setAlwaysOnTop(true);
illegalActionDialog.setLocationByPlatform(true);
illegalActionDialog.setMinimumSize(new java.awt.Dimension(411, 325));
3635
3640     illegalActionDialog.setModalityType(java.awt.Dialog.ModalityType.APPLICATION_MODAL);
illegalActionDialog.setResizable(false);
3645     jLabel67.setText("Sorry, you cannot do that.");
jLabel68.setText("Details and tips on how to proceed are below.");
3650
3655     illegalActionDialogConfirmButton.setText("Confirm");
illegalActionDialogConfirmButton.addMouseListener(new
java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        illegalActionDialogConfirmButtonMouseReleased(evt);
    }
});
3660
3665     illegalActionDialogTextArea.setColumns(20);
illegalActionDialogTextArea.setEditable(false);
illegalActionDialogTextArea.setLineWrap(true);
illegalActionDialogTextArea.setRows(5);
illegalActionDialogTextArea.setWrapStyleWord(true);
illegalActionDialogScrollPane.setViewportView(illegalActionDialogTextArea);
3670
3675     org.jdesktop.layout.GroupLayout illegalActionDialogLayout = new
org.jdesktop.layout.GroupLayout(illegalActionDialog.getContentPane());
illegalActionDialog.getContentPane().setLayout(illegalActionDialogLayout);
illegalActionDialogLayout.setHorizontalGroup(
3680
3685         illegalActionDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .addGroup(illegalActionDialogLayout.createSequentialGroup()
                .addGap(20, 20, 20)
                .addComponent(jLabel67, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addGap(20, 20, 20)
                .addComponent(jLabel68, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addGap(20, 20, 20)
                .addComponent(illegalActionDialogScrollPane,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 371,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addGap(20, 20, 20)
                .addComponent(illegalActionDialogLayout.createSequentialGroup()
                    .addGap(157, 157, 157)
                    .addComponent(illegalActionDialogConfirmButton)))
            )
            .addGap(20, 20, 20)
        );
illegalActionDialogLayout.setVerticalGroup(
3690
3695         illegalActionDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .addGroup(illegalActionDialogLayout.createSequentialGroup()
                .addGap(20, 20, 20)
                .addComponent(jLabel67)
                .addGap(8, 8, 8)
                .addComponent(jLabel68)
                .addGap(18, 18, 18)
                .addComponent(illegalActionDialogScrollPane,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 161,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(illegalActionDialogConfirmButton)
                .addGap(39, Short.MAX_VALUE)
            )
        );

```

```

3700     resetWarningDialog.setTitle("Warning");
3701     resetWarningDialog.setAlwaysOnTop(true);
3702     resetWarningDialog.setLocationByPlatform(true);
3703     resetWarningDialog.setMinimumSize(new java.awt.Dimension(411, 325));
3704
3705     resetWarningDialog.setModalityType(java.awt.Dialog.ModalityType.APPLICATION_MODAL);
3706     resetWarningDialog.setResizable(false);
3707
3708     jLabel69.setText("Warning: your action needs confirmation.");
3709
3710     jLabel70.setText("Please read the following information:");
3711
3712     resetWarningDialogConfirmButton.setText("Confirm");
3713     resetWarningDialogConfirmButton.addMouseListener(new
3714         java.awt.event.MouseAdapter() {
3715             public void mouseReleased(java.awt.event.MouseEvent evt) {
3716                 resetWarningDialogConfirmButtonMouseReleased(evt);
3717             }
3718         });
3719
3720     resetWarningDialogTextArea.setColumns(20);
3721     resetWarningDialogTextArea.setEditable(false);
3722     resetWarningDialogTextArea.setLineWrap(true);
3723     resetWarningDialogTextArea.setRows(5);
3724     resetWarningDialogTextArea.setWrapStyleWord(true);
3725     resetWarningDialogScrollPane.setViewportView(resetWarningDialogTextArea);
3726
3727     resetWarningDialogCancelButton.setText("Cancel");
3728     resetWarningDialogCancelButton.addMouseListener(new
3729         java.awt.event.MouseAdapter() {
3730             public void mouseReleased(java.awt.event.MouseEvent evt) {
3731                 resetWarningDialogCancelButtonMouseReleased(evt);
3732             }
3733         });
3734
3735     org.jdesktop.layout.GroupLayout resetWarningDialogLayout = new
3736     org.jdesktop.layout.GroupLayout(resetWarningDialog.getContentPane());
3737     resetWarningDialog.getContentPane().setLayout(resetWarningDialogLayout);
3738     resetWarningDialogLayout.setHorizontalGroup(
3739         resetWarningDialogLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
3740             .add(resetWarningDialogLayout.createSequentialGroup()
3741                 .add(resetWarningDialogLayout.createParallelGroup()
3742                     .add(20, 20, 20)
3743
3744                     .add(resetWarningDialogLayout.createSequentialGroup()
3745                         .add(jLabel69, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
3746                             371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
3747                         .add(jLabel70, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
3748                             371, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
3749                         .add(resetWarningDialogScrollPane,
3750                             org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 371,
3751                             org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
3752                         .add(resetWarningDialogLayout.createSequentialGroup()
3753                             .add(resetWarningDialogLayout.createParallelGroup()
3754                                 .add(resetWarningDialogCancelButton)
3755                                 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,
3756                                     org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
3757                                 .add(resetWarningDialogConfirmButton)))
3758                             .add(20, 20, 20))
3759             );
3760             resetWarningDialogLayout.setVerticalGroup(
3761                 resetWarningDialogLayout.createSequentialGroup()
3762                     .add(resetWarningDialogLayout.createParallelGroup()
3763                         .add(20, 20, 20)
3764                         .add(jLabel69)
3765                         .add(8, 8, 8)
3766                         .add(jLabel70)
3767                         .add(18, 18, 18)
3768                         .add(resetWarningDialogScrollPane,
3769                             org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 161,
3770                             org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
3771                         .add(18, 18, 18)
3772
3773                     .add(resetWarningDialogLayout.createParallelGroup()
3774                         .add(resetWarningDialogCancelButton)
3775                         .add(resetWarningDialogConfirmButton)))

```

```

    );
3780     approveParticipantsDialog.setTitle("Warning");
3781     approveParticipantsDialog.setAlwaysOnTop(true);
3782     approveParticipantsDialog.setLocationByPlatform(true);
3783     approveParticipantsDialog.setMinimumSize(new java.awt.Dimension(411, 325));
3784
3785     approveParticipantsDialog.setModalityType(java.awt.Dialog.ModalityType.APPLICATION_MODAL);
3786         approveParticipantsDialog.setResizable(false);
3787         approveParticipantsDialog.getContentPane().setLayout(new
3788 org.netbeans.lib.awtextra.AbsoluteLayout());
3789
3790         jLabel71.setText("Warning: some settings were not fulfilled for some
3791 entries.");
3792         approveParticipantsDialog.getContentPane().add(jLabel71, new
3793 org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 371, -1));
3794
3795         jLabel72.setText("Please read the following information:");
3796         approveParticipantsDialog.getContentPane().add(jLabel72, new
3797 org.netbeans.lib.awtextra.AbsoluteConstraints(20, 44, 371, -1));
3798
3799         approveParticipantsDialogIgnoreButton.setText("Ignore");
3800         approveParticipantsDialogIgnoreButton.addMouseListener(new
3801 java.awt.event.MouseAdapter() {
3802             public void mouseReleased(java.awt.event.MouseEvent evt) {
3803                 approveParticipantsDialogIgnoreButtonMouseReleased(evt);
3804             }
3805         });
3806
3807     approveParticipantsDialog.getContentPane().add(approveParticipantsDialogIgnoreButton,
3808 new org.netbeans.lib.awtextra.AbsoluteConstraints(307, 257, -1, -1));
3809
3810         approveParticipantsDialogTextArea.setColumns(20);
3811         approveParticipantsDialogTextArea.setEditable(false);
3812         approveParticipantsDialogTextArea.setLineWrap(true);
3813         approveParticipantsDialogTextArea.setRows(5);
3814         approveParticipantsDialogTextArea.setWrapStyleWord(true);
3815
3816     approveParticipantsDialogScrollPane.setViewportView(approveParticipantsDialogTextArea);
3817 ;
3818
3819     approveParticipantsDialog.getContentPane().add(approveParticipantsDialogScrollPane,
3820 new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 78, 371, 161));
3821
3822         approveParticipantsDialogCancelButton.setText("Cancel");
3823         approveParticipantsDialogCancelButton.addMouseListener(new
3824 java.awt.event.MouseAdapter() {
3825             public void mouseReleased(java.awt.event.MouseEvent evt) {
3826                 approveParticipantsDialogCancelButtonMouseReleased(evt);
3827             }
3828         });
3829
3830     approveParticipantsDialog.getContentPane().add(approveParticipantsDialogCancelButton,
3831 new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 257, -1, -1));
3832
3833         allocationInfoDialog.setTitle("Information");
3834         allocationInfoDialog.setAlwaysOnTop(true);
3835         allocationInfoDialog.setIconImage(null);
3836         allocationInfoDialog.setLocationByPlatform(true);
3837         allocationInfoDialog.setMinimumSize(new java.awt.Dimension(411, 146));
3838         allocationInfoDialog.setModal(true);
3839         allocationInfoDialog.setResizable(false);
3840         allocationInfoDialog.getContentPane().setLayout(new
3841 org.netbeans.lib.awtextra.AbsoluteLayout());
3842
3843         jLabel73.setText("Allocation in progress, please wait... Current
3844 combination:");
3845         allocationInfoDialog.getContentPane().add(jLabel73, new
3846 org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 371, 15));
3847
3848         allocationInfoProgressBar.setStringPainted(true);
3849         allocationInfoDialog.getContentPane().add(allocationInfoProgressBar, new
3850 org.netbeans.lib.awtextra.AbsoluteConstraints(18, 87, 375, -1));
3851
3852         allocationInfoTextField.setEditable(false);
3853         allocationInfoTextField.setFont(new java.awt.Font("Courier", 0, 13));
3854         allocationInfoTextField.setAutoscrolls(false);
3855

```

```

    allocationInfoTextField.setDragEnabled(false);
    allocationInfoTextField.setFocusTraversalKeysEnabled(false);
    allocationInfoTextField.setFocusable(false);
    allocationInfoTextField.setRequestFocusEnabled(false);
    allocationInfoDialog.getContentPane().add(allocationInfoTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(18, 40, 375, -1));

3860

3865     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("NESDA Tournament Manager 2013");
    setLocationByPlatform(true);
    setMinimumSize(new java.awt.Dimension(1180, 678));
    setResizable(false);
    getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

3870     participantsPanel.setEnabled(false);
    participantsPanel.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

3875     jLabel37.setText("Students / Room");
    participantsPanel.add(jLabel37, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 20, -1, -1));

3880     eventSettingsOOSR.setText("6");
    eventSettingsOOSR.setEnabled(false);
    participantsPanel.add(eventSettingsOOSR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 44, 174, -1));

3885     jLabel38.setText("Students / Event / School");
    participantsPanel.add(jLabel38, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 20, -1, -1));

3890     eventSettingsOOSES.setText("4");
    eventSettingsOOSES.setEnabled(false);
    participantsPanel.add(eventSettingsOOSES, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 44, 174, -1));

3895     jLabel39.setText("Judges / Room");
    participantsPanel.add(jLabel39, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 20, -1, -1));

3900     eventSettingsOOJR.setText("2");
    eventSettingsOOJR.setEnabled(false);
    participantsPanel.add(eventSettingsOOJR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 44, 174, -1));

3905     settingsResetButton.setText("Reset");
    settingsResetButton.setEnabled(false);
    settingsResetButton.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            settingsResetButtonMouseReleased(evt);
        }
    });
    participantsPanel.add(settingsResetButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 344, 358, -1));

3910     settingsLoadButton.setText("Load");
    settingsLoadButton.setEnabled(false);
    settingsLoadButton.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            settingsLoadButtonMouseReleased(evt);
        }
    });
    participantsPanel.add(settingsLoadButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 344, 358, -1));

3915     settingsApproveToggleButton.setSelected(true);
    settingsApproveToggleButton.setText("Approve");
    settingsApproveToggleButton.addMouseListener(new java.awt.event.MouseAdapter()
{
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        settingsApproveToggleButtonMouseReleased(evt);
    }
});
    participantsPanel.add(settingsApproveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 344, 358, -1));

3920

3925     settingsSaveToggleButton.setText("Save");
    settingsSaveToggleButton.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            settingsSaveToggleButtonMouseReleased(evt);
        }
});
    participantsPanel.add(settingsSaveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 400, 358, -1));

```

```

        });
        participantsPanel.add(settingsSaveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 377, 1094, -1));
3940
        eventSettingsOOFinalSR.setText("6");
        eventSettingsOOFinalSR.setEnabled(false);
        participantsPanel.add(eventSettingsOOFinalSR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 224, -1, -1));
3945
        generalSettingsRoomsAvailable1.setText("8");
        generalSettingsRoomsAvailable1.setEnabled(false);
        participantsPanel.add(generalSettingsRoomsAvailable1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 44, 81, -1));
3950
        generalSettingsMaximalTime1.setText("8");
        generalSettingsMaximalTime1.setEnabled(false);
        participantsPanel.add(generalSettingsMaximalTime1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 80, 34, -1));
3955
        generalSettingsSchoolsNumber.setText("12");
        generalSettingsSchoolsNumber.setEnabled(false);
        participantsPanel.add(generalSettingsSchoolsNumber, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 116, 174, -1));
3960
        jLabel42.setText("Original Oratory");
        participantsPanel.add(jLabel42, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 50, 174, -1));
3965
        jLabel43.setText("Oral Interpretation");
        participantsPanel.add(jLabel43, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 86, 174, -1));
3970
        jLabel44.setText("Impromptu Speaking");
        participantsPanel.add(jLabel44, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 122, 174, -1));
3975
        jLabel45.setText("Duet Acting");
        participantsPanel.add(jLabel45, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 158, 174, -1));
        jLabel46.setText("Debate (Q & semifinals)");
        participantsPanel.add(jLabel46, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 194, 174, -1));
3980
        jLabel47.setText("Finals (OO, OI, IS, DA)");
        participantsPanel.add(jLabel47, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 230, 174, -1));
3985
        jLabel48.setText("Rooms available (Day 1, 2)");
        participantsPanel.add(jLabel48, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 50, 174, -1));
3990
        jLabel49.setText("Tournament time (Day 1, 2)");
        participantsPanel.add(jLabel49, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 86, 174, -1));
3995
        jLabel50.setText("Schools in tournament");
        participantsPanel.add(jLabel50, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 122, -1, -1));
4000
        jLabel51.setText("Students / School");
        participantsPanel.add(jLabel51, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 194, 121, -1));
4005
        jLabel52.setText("Teachers / School");
        participantsPanel.add(jLabel52, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 158, 121, -1));
        jLabel53.setText("Combined Events");
        participantsPanel.add(jLabel53, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 230, 174, -1));
4010
        jLabel54.setText("Event settings:");
        participantsPanel.add(jLabel54, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 103, -1));
        jLabel55.setText("General settings:");

```

```

4015     participantsPanel.add(jLabel55, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 20, 124, -1));

        eventSettingsDAFinalSR.setText("4");
        eventSettingsDAFinalSR.setEnabled(false);
        participantsPanel.add(eventSettingsDAFinalSR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(312, 224, -1, -1));

        generalSettingsSS.setText("12");
        generalSettingsSS.setEnabled(false);
        participantsPanel.add(generalSettingsSS, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 188, 174, -1));

        generalSettingsTS.setText("2");
        generalSettingsTS.setEnabled(false);
        participantsPanel.add(generalSettingsTS, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 152, 174, -1));

        jLabel57.setText("hours");
        participantsPanel.add(jLabel57, new
org.netbeans.lib.awtextra.AbsoluteConstraints(984, 86, 38, -1));

4035     eventSettingsOISR.setText("6");
        eventSettingsOISR.setEnabled(false);
        participantsPanel.add(eventSettingsOISR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 80, 174, -1));

        eventSettingsISSR.setText("6");
        eventSettingsISSR.setEnabled(false);
        participantsPanel.add(eventSettingsISSR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 116, 174, -1));

4045     eventSettingsDASR.setText("4");
        eventSettingsDASR.setEnabled(false);
        participantsPanel.add(eventSettingsDASR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 152, 132, -1));

        eventSettingsOISES.setText("4");
        eventSettingsOISES.setEnabled(false);
        participantsPanel.add(eventSettingsOISES, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 80, 174, -1));

4055     eventSettingsISSES.setText("4");
        eventSettingsISSES.setEnabled(false);
        participantsPanel.add(eventSettingsISSES, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 116, 174, -1));

        eventSettingsDASES.setText("2");
        eventSettingsDASES.setEnabled(false);
        participantsPanel.add(eventSettingsDASES, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 152, 132, -1));

4065     eventSettingsDebateSES.setText("2");
        eventSettingsDebateSES.setEnabled(false);
        participantsPanel.add(eventSettingsDebateSES, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 188, 132, -1));

        eventSettingsOIJR.setText("2");
        eventSettingsOIJR.setEnabled(false);
        participantsPanel.add(eventSettingsOIJR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 80, 174, -1));

4075     eventSettingsISJR.setText("2");
        eventSettingsISJR.setEnabled(false);
        participantsPanel.add(eventSettingsISJR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 116, 174, -1));

        eventSettingsDAJR.setText("2");
        eventSettingsDAJR.setEnabled(false);
        participantsPanel.add(eventSettingsDAJR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 152, 174, -1));

4085     eventSettingsFinalsJR.setText("5");
        eventSettingsFinalsJR.setEnabled(false);
        participantsPanel.add(eventSettingsFinalsJR, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 224, 174, -1));

        eventSettingsDebateJR.setText("3");
        eventSettingsDebateJR.setEnabled(false);

```

```

    participantsPanel.add(eventSettingsDebateJR, new
4095      org.netbeans.lib.awtextra.AbsoluteConstraints(572, 188, 174, -1));
        jLabel36.setText("pairs");
        participantsPanel.add(jLabel36, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(347, 230, -1, -1));
4100        jLabel56.setText("pairs");
        participantsPanel.add(jLabel56, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(347, 158, -1, -1));
        jLabel58.setText("pairs");
        participantsPanel.add(jLabel58, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(531, 158, -1, -1));
        jLabel59.setText("pairs");
        participantsPanel.add(jLabel59, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(531, 194, -1, -1));
        eventSettingsOIFinalSR.setText("6");
        eventSettingsOIFinalSR.setEnabled(false);
        participantsPanel.add(eventSettingsOIFinalSR, new
4115      org.netbeans.lib.awtextra.AbsoluteConstraints(240, 224, -1, -1));
        eventSettingsISFinalSR.setText("6");
        eventSettingsISFinalSR.setEnabled(false);
        participantsPanel.add(eventSettingsISFinalSR, new
4120      org.netbeans.lib.awtextra.AbsoluteConstraints(276, 224, -1, -1));
        generalSettingsRoomsAvailable2.setText("8");
        generalSettingsRoomsAvailable2.setEnabled(false);
        participantsPanel.add(generalSettingsRoomsAvailable2, new
4125      org.netbeans.lib.awtextra.AbsoluteConstraints(1032, 44, 82, -1));
        generalSettingsMaximalTime2.setText("8");
        generalSettingsMaximalTime2.setEnabled(false);
        participantsPanel.add(generalSettingsMaximalTime2, new
4130      org.netbeans.lib.awtextra.AbsoluteConstraints(1032, 80, 34, -1));
        jLabel60.setText("hours");
        participantsPanel.add(jLabel60, new
4135      org.netbeans.lib.awtextra.AbsoluteConstraints(1076, 86, 38, -1));
        generalSettingsCEComboBox2.setModel(new javax.swing.DefaultComboBoxModel(new
String[] { "OO", "OI", "IS", "DA" }));
        generalSettingsCEComboBox2.setSelectedIndex(3);
        generalSettingsCEComboBox2.setEnabled(false);
4140        generalSettingsCEComboBox2.setMinimumSize(new java.awt.Dimension(58, 27));
        generalSettingsCEComboBox2.setPreferredSize(new java.awt.Dimension(68, 27));
        participantsPanel.add(generalSettingsCEComboBox2, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(1044, 224, 70, -1));
4145        generalSettingsCEComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new
String[] { "OO", "OI", "IS", "DA" }));
        generalSettingsCEComboBox1.setSelectedIndex(2);
        generalSettingsCEComboBox1.setEnabled(false);
        generalSettingsCEComboBox1.setMinimumSize(new java.awt.Dimension(58, 27));
        generalSettingsCEComboBox1.setPreferredSize(new java.awt.Dimension(58, 27));
        participantsPanel.add(generalSettingsCEComboBox1, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(972, 224, 70, -1));
4150        generalSettingsCECheckBox.setSelected(true);
        generalSettingsCECheckBox.setEnabled(false);
        participantsPanel.add(generalSettingsCECheckBox, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(940, 226, -1, -1));
4155        tabbedPane.addTab("Settings", participantsPanel);
        participantsPane.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
        jLabel2.setText("Schools");
        participantsPane.add(jLabel2, new
4160      org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 174, -1));
        participantsPane.add(participantsSchoolsTextField, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(20, 44, 174, -1));
4165        participantsSchoolsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);

```

```

        participantsSchoolsList.setEnabled(false);
        participantsSchoolsList.setFocusable(false);
        participantsSchoolsList.addListSelectionListener(new
4175 javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                participantsSchoolsListValueChanged(evt);
            }
        });
        jScrollPane1.setViewportView(participantsSchoolsList);

        participantsPane.add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, 174, 191));
4185
        participantsSchoolsAddButton.setText("Add");
        participantsSchoolsAddButton.setEnabled(false);
        participantsSchoolsAddButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsSchoolsAddButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsSchoolsAddButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 278, -1, -1));
4190
        participantsSchoolsRemoveButton.setText("Remove");
        participantsSchoolsRemoveButton.setEnabled(false);
        participantsSchoolsRemoveButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsSchoolsRemoveButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsSchoolsRemoveButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(101, 278, -1, -1));
4195
        participantsSchoolsEditButton.setText("Edit");
        participantsSchoolsEditButton.setEnabled(false);
        participantsSchoolsEditButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsSchoolsEditButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsSchoolsEditButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 311, -1, -1));
4200
        participantsSchoolsSearchButton.setText("Search");
        participantsSchoolsSearchButton.setEnabled(false);
        participantsSchoolsSearchButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsSchoolsSearchButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsSchoolsSearchButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(101, 311, 93, -1));
4205
        jLabel3.setText("Teachers");
        participantsPane.add(jLabel3, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 20, 174, -1));
4210
        participantsTeachersTextField.setEnabled(false);
        participantsPane.add(participantsTeachersTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 44, 174, -1));
4215
        participantsTeachersList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        participantsTeachersList.setEnabled(false);
        participantsTeachersList.setFocusable(false);
        participantsTeachersList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                participantsTeachersListValueChanged(evt);
            }
        });
        jScrollPane2.setViewportView(participantsTeachersList);

        participantsPane.add(jScrollPane2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 80, 174, 191));
4220

```

```

    participantsTeachersAddButton.setText("Add");
    participantsTeachersAddButton.setEnabled(false);
    participantsTeachersAddButton.addMouseListener(new
4255 java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            participantsTeachersAddButtonMouseReleased(evt);
        }
    });
    participantsPane.add(participantsTeachersAddButton, new
4260 org.netbeans.lib.awtextra.AbsoluteConstraints(204, 278, -1, -1));

    participantsTeachersEditButton.setText("Edit");
    participantsTeachersEditButton.setEnabled(false);
    participantsTeachersEditButton.addMouseListener(new
4265 java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            participantsTeachersEditButtonMouseReleased(evt);
        }
    });
    participantsPane.add(participantsTeachersEditButton, new
4270 org.netbeans.lib.awtextra.AbsoluteConstraints(204, 311, -1, -1));

    participantsTeachersSearchButton.setText("Search");
    participantsTeachersSearchButton.setEnabled(false);
    participantsTeachersSearchButton.addMouseListener(new
4275 java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            participantsTeachersSearchButtonMouseReleased(evt);
        }
    });
    participantsPane.add(participantsTeachersSearchButton, new
4280 org.netbeans.lib.awtextra.AbsoluteConstraints(285, 311, 93, -1));

    participantsTeachersRemoveButton.setText("Remove");
    participantsTeachersRemoveButton.setEnabled(false);
    participantsTeachersRemoveButton.addMouseListener(new
4285 java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            participantsTeachersRemoveButtonMouseReleased(evt);
        }
    });
    participantsPane.add(participantsTeachersRemoveButton, new
4290 org.netbeans.lib.awtextra.AbsoluteConstraints(285, 278, -1, -1));

    jLabel4.setText("Students");
    participantsPane.add(jLabel4, new
4295 org.netbeans.lib.awtextra.AbsoluteConstraints(388, 20, 174, -1));

    participantsStudentsTextField.setEnabled(false);
    participantsPane.add(participantsStudentsTextField, new
4300 org.netbeans.lib.awtextra.AbsoluteConstraints(388, 44, 174, -1));

    participantsStudentsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
    participantsStudentsList.setEnabled(false);
    participantsStudentsList.setFocusable(false);
    participantsStudentsList.addListSelectionListener(new
4305 javax.swing.event.ListSelectionListener() {
        public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
            participantsStudentsListValueChanged(evt);
        }
    });
    jScrollPane3.setViewportView(participantsStudentsList);

    participantsPane.add(jScrollPane3, new
4315 org.netbeans.lib.awtextra.AbsoluteConstraints(388, 80, 174, 191));

    participantsStudentsAddButton.setText("Add");
    participantsStudentsAddButton.setEnabled(false);
    participantsStudentsAddButton.addMouseListener(new
4320 java.awt.event.MouseAdapter() {
        public void mouseReleased(java.awt.event.MouseEvent evt) {
            participantsStudentsAddButtonMouseReleased(evt);
        }
    });
    participantsPane.add(participantsStudentsAddButton, new
4325 org.netbeans.lib.awtextra.AbsoluteConstraints(388, 278, -1, -1));

```

```

4330
        participantsStudentsEditButton.setText("Edit");
        participantsStudentsEditButton.setEnabled(false);
        participantsStudentsEditButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsStudentsEditButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsStudentsEditButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 311, -1, -1));

        participantsStudentsSearchButton.setText("Search");
        participantsStudentsSearchButton.setEnabled(false);
        participantsStudentsSearchButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsStudentsSearchButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsStudentsSearchButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(469, 311, 93, -1));

        participantsStudentsRemoveButton.setText("Remove");
        participantsStudentsRemoveButton.setEnabled(false);
        participantsStudentsRemoveButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsStudentsRemoveButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsStudentsRemoveButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(469, 278, -1, -1));

        jLabel5.setText("Student codes");
        participantsPane.add(jLabel5, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 20, 174, -1));

        participantsStudentCodesTextField.setEnabled(false);
        participantsPane.add(participantsStudentCodesTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 44, 174, -1));

participantsStudentCodesList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        participantsStudentCodesList.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        participantsStudentCodesList.setEnabled(false);
        participantsStudentCodesList.setFocusable(false);
        participantsStudentCodesList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                participantsStudentCodesListValueChanged(evt);
            }
        });
        jScrollPane4.setViewportView(participantsStudentCodesList);

        participantsPane.add(jScrollPane4, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 80, 174, 191));

        participantsStudentCodesSearchButton.setText("Search");
        participantsStudentCodesSearchButton.setEnabled(false);
        participantsStudentCodesSearchButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsStudentCodesSearchButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsStudentCodesSearchButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 311, 174, -1));

        jLabel6.setText("Assign events");
        participantsPane.add(jLabel6, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 20, 174, -1));

        participantsAssignEventsAssignEventsButton.setText("Assign events");
        participantsAssignEventsAssignEventsButton.setEnabled(false);
        participantsAssignEventsAssignEventsButton.addMouseListener(new
java.awt.event.MouseAdapter() {

```

```

4410             public void mouseReleased(java.awt.event.MouseEvent evt) {
4411                 participantsAssignEventsAssignEventsButtonMouseReleased(evt);
4412             }
4413         });
4414         participantsPane.add(participantsAssignEventsAssignEventsButton, new
4415 org.netbeans.lib.awtextra.AbsoluteConstraints(756, 278, 174, -1));
4416
4417         jLabel7.setText("Assign pairs");
4418         participantsPane.add(jLabel7, new
4419 org.netbeans.lib.awtextra.AbsoluteConstraints(940, 20, 174, -1));
4420
4421         participantsAssignPairsEventsList.setModel(new javax.swing.AbstractListModel()
4422 {
4423             String[] strings = { "Duet Acting", "Debate" };
4424             public int getSize() { return strings.length; }
4425             public Object getElementAt(int i) { return strings[i]; }
4426         });
4427
4428         participantsAssignPairsEventsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
4429         participantsAssignPairsEventsList.setEnabled(false);
4430         participantsAssignPairsEventsList.addListSelectionListener(new
4431 javax.swing.event.ListSelectionListener() {
4432             public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
4433                 participantsAssignPairsEventsListValueChanged(evt);
4434             }
4435         });
4436         jScrollPane6.setViewportView(participantsAssignPairsEventsList);
4437
4438         participantsPane.add(jScrollPane6, new
4439 org.netbeans.lib.awtextra.AbsoluteConstraints(940, 80, 174, 38));
4440
4441         participantsAssignPairsRemovePairButton.setText("Remove pair");
4442         participantsAssignPairsRemovePairButton.setEnabled(false);
4443         participantsAssignPairsRemovePairButton.addMouseListener(new
4444 java.awt.event.MouseAdapter() {
4445             public void mouseReleased(java.awt.event.MouseEvent evt) {
4446                 participantsAssignPairsRemovePairButtonMouseReleased(evt);
4447             }
4448         });
4449         participantsPane.add(participantsAssignPairsRemovePairButton, new
4450 org.netbeans.lib.awtextra.AbsoluteConstraints(940, 311, 174, -1));
4451
4452         participantsAssignPairsPairButton.setText("Pair");
4453         participantsAssignPairsPairButton.setEnabled(false);
4454         participantsAssignPairsPairButton.addMouseListener(new
4455 java.awt.event.MouseAdapter() {
4456             public void mouseReleased(java.awt.event.MouseEvent evt) {
4457                 participantsAssignPairsPairButtonMouseReleased(evt);
4458             }
4459         });
4460         participantsPane.add(participantsAssignPairsPairButton, new
4461 org.netbeans.lib.awtextra.AbsoluteConstraints(940, 278, 174, -1));
4462
4463         participantsResetButton.setText("Reset");
4464         participantsResetButton.addMouseListener(new java.awt.event.MouseAdapter() {
4465             public void mouseReleased(java.awt.event.MouseEvent evt) {
4466                 participantsResetButtonMouseReleased(evt);
4467             }
4468         });
4469         participantsPane.add(participantsResetButton, new
4470 org.netbeans.lib.awtextra.AbsoluteConstraints(756, 344, 358, -1));
4471
4472         participantsExportButton.setText("Export");
4473         participantsExportButton.addMouseListener(new java.awt.event.MouseAdapter() {
4474             public void mouseReleased(java.awt.event.MouseEvent evt) {
4475                 participantsExportButtonMouseReleased(evt);
4476             }
4477         });
4478         participantsPane.add(participantsExportButton, new
4479 org.netbeans.lib.awtextra.AbsoluteConstraints(20, 344, 358, -1));
4480
4481         participantsApproveToggleButton.setText("Approve");
4482         participantsApproveToggleButton.setEnabled(false);
4483         participantsApproveToggleButton.addMouseListener(new
4484 java.awt.event.MouseAdapter() {
4485             public void mouseReleased(java.awt.event.MouseEvent evt) {
4486                 participantsApproveToggleButtonMouseReleased(evt);
4487             }
4488         });

```

```

        });
        participantsPane.add(participantsApproveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 344, 358, -1));

        participantsSaveToggleButton.setText("Save");
        participantsSaveToggleButton.setEnabled(false);
        participantsSaveToggleButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsSaveToggleButtonMouseReleased(evt);
            }
        });
        participantsPane.add(participantsSaveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 377, 1094, -1));

        participantsAssignEventsOriginalOratoryCheckBox.setText("Original Oratory");
        participantsAssignEventsOriginalOratoryCheckBox.setEnabled(false);
        participantsAssignEventsOriginalOratoryCheckBox.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsAssignEventsOriginalOratoryCheckBoxMouseReleased(evt);
            }
        });
        participantsPane.add(participantsAssignEventsOriginalOratoryCheckBox, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 146, 174, -1));

        participantsAssignEventsOralInterpretationCheckBox.setText("Oral
Interpretation");
        participantsAssignEventsOralInterpretationCheckBox.setEnabled(false);
        participantsAssignEventsOralInterpretationCheckBox.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsAssignEventsOralInterpretationCheckBoxMouseReleased(evt);
            }
        });
        participantsPane.add(participantsAssignEventsOralInterpretationCheckBox, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 171, 174, -1));

        participantsAssignEventsImpromptuSpeakingCheckBox.setText("Impromptu
Speaking");
        participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(false);
        participantsAssignEventsImpromptuSpeakingCheckBox.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsAssignEventsImpromptuSpeakingCheckBoxMouseReleased(evt);
            }
        });
        participantsPane.add(participantsAssignEventsImpromptuSpeakingCheckBox, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 196, -1, -1));

        participantsAssignEventsDuetActingCheckBox.setText("Duet Acting");
        participantsAssignEventsDuetActingCheckBox.setEnabled(false);
        participantsAssignEventsDuetActingCheckBox.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsAssignEventsDuetActingCheckBoxMouseReleased(evt);
            }
        });
        participantsPane.add(participantsAssignEventsDuetActingCheckBox, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 221, -1, -1));

        participantsAssignEventsDebateCheckBox.setText("Debate");
        participantsAssignEventsDebateCheckBox.setEnabled(false);
        participantsAssignEventsDebateCheckBox.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                participantsAssignEventsDebateCheckBoxMouseReleased(evt);
            }
        });
        participantsPane.add(participantsAssignEventsDebateCheckBox, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 246, -1, -1));

participantsAssignPairsLeftList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE
_SELECTION);
participantsAssignPairsLeftList.setEnabled(false);
participantsAssignPairsLeftList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
    public void valueChanged(javax.swing.event.ListSelectionEvent evt) {

```

```

        participantsAssignPairsLeftListValueChanged(evt);
    }
});
jScrollPane5.setViewportView(participantsAssignPairsLeftList);

participantsPane.add(jScrollPane5, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 126, 80, 72));

participantsAssignPairsRightList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
participantsAssignPairsRightList.setEnabled(false);
participantsAssignPairsRightList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
    public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
        participantsAssignPairsRightListValueChanged(evt);
    }
});
jScrollPane31.setViewportView(participantsAssignPairsRightList);

participantsPane.add(jScrollPane31, new
org.netbeans.lib.awtextra.AbsoluteConstraints(1030, 126, 84, 72));

participantsAssignPairsPairsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
participantsAssignPairsPairsList.setEnabled(false);
participantsAssignPairsPairsList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
    public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
        participantsAssignPairsPairsListValueChanged(evt);
    }
});
jScrollPane32.setViewportView(participantsAssignPairsPairsList);

participantsPane.add(jScrollPane32, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 206, 174, 65));

participantsStudentCodesAssignCodesButton.setText("Assign codes");
participantsStudentCodesAssignCodesButton.setEnabled(false);
participantsStudentCodesAssignCodesButton.addMouseListener(new
java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        participantsStudentCodesAssignCodesButtonMouseReleased(evt);
    }
});
participantsPane.add(participantsStudentCodesAssignCodesButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 278, 174, -1));

participantsProgressBar.setString("Progress");
participantsProgressBar.setStringPainted(true);
participantsPane.add(participantsProgressBar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 415, 1094, -1));

tabbedPane.addTab("Participants", participantsPane);

qualificationPane.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

jLabel8.setText("Events");
qualificationPane.add(jLabel8, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 174, -1));

qualificationEventsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
qualificationEventsList.setEnabled(false);
qualificationEventsList.setFocusable(false);
qualificationEventsList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
    public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
        qualificationEventsListValueChanged(evt);
    }
});
jScrollPane7.setViewportView(qualificationEventsList);

qualificationPane.add(jScrollPane7, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, 174, 191));

qualificationProgressBar.setString("Progress");

```

```

        qualificationProgressBar.setStringPainted(true);
        qualificationPane.add(qualificationProgressBar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 415, 1094, -1));

4650    jLabel9.setText("Rounds");
        qualificationPane.add(jLabel9, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 20, 174, -1));

4655    qualificationRoundsTextField.setEnabled(false);
        qualificationPane.add(qualificationRoundsTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 44, 174, -1));

4660    qualificationRoundsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        qualificationRoundsList.setEnabled(false);
        qualificationRoundsList.setFocusable(false);
        qualificationRoundsList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                qualificationRoundsListValueChanged(evt);
            }
        });
        jScrollPane8.setViewportView(qualificationRoundsList);

4670    qualificationPane.add(jScrollPane8, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 80, 174, 191));

4675    qualificationRoundsEditNameButton.setText("Edit name");
        qualificationRoundsEditNameButton.setEnabled(false);
        qualificationRoundsEditNameButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationRoundsEditNameButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationRoundsEditNameButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(204, 278, 174, -1));

4685    jLabel10.setText("Rooms");
        qualificationPane.add(jLabel10, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 20, 174, -1));

4690    qualificationRoomsTextField.setEnabled(false);
        qualificationPane.add(qualificationRoomsTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 44, 174, -1));

4695    qualificationRoomsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        qualificationRoomsList.setEnabled(false);
        qualificationRoomsList.setFocusable(false);
        qualificationRoomsList.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                qualificationRoomsListValueChanged(evt);
            }
        });
        jScrollPane9.setViewportView(qualificationRoomsList);

4705    qualificationPane.add(jScrollPane9, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 80, 174, 191));

4710    qualificationRoomsEditNameButton.setText("Edit name");
        qualificationRoomsEditNameButton.setEnabled(false);
        qualificationRoomsEditNameButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationRoomsEditNameButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationRoomsEditNameButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 278, 174, -1));

4720    jLabel11.setText("Judges");
        qualificationPane.add(jLabel11, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 20, 174, -1));

```

```

4725 qualificationJudgesList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        qualificationJudgesList.setEnabled(false);
        qualificationJudgesList.setFocusable(false);
        qualificationJudgesList.addListSelectionListener(new
4730 javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                qualificationJudgesListValueChanged(evt);
            }
        });
4735 jScrollPane10.setViewportView(qualificationJudgesList);

        qualificationPane.add(jScrollPane10, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 80, 174, 191));

4740 qualificationJudgesSubstituteJudgeButton.setText("Substitute judge");
        qualificationJudgesSubstituteJudgeButton.setEnabled(false);
        qualificationJudgesSubstituteJudgeButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationJudgesSubstituteJudgeButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationJudgesSubstituteJudgeButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 278, 174, -1));

4745 jLabel12.setText("Student codes");
        qualificationPane.add(jLabel12, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 20, 174, -1));

4750 qualificationStudentCodesTextField.setEnabled(false);
        qualificationPane.add(qualificationStudentCodesTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 44, 174, -1));

4755 qualificationStudentCodesList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        qualificationStudentCodesList.setEnabled(false);
        qualificationStudentCodesList.setFocusable(false);
        qualificationStudentCodesList.addListSelectionListener(new
4760 javax.swing.event.ListSelectionListener() {
            public void valueChanged(javax.swing.event.ListSelectionEvent evt) {
                qualificationStudentCodesListValueChanged(evt);
            }
        });
4765 jScrollPane11.setViewportView(qualificationStudentCodesList);

        qualificationPane.add(jScrollPane11, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 80, 174, 191));

4770 qualificationStudentCodesSearchButton.setText("Search");
        qualificationStudentCodesSearchButton.setEnabled(false);
        qualificationStudentCodesSearchButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationStudentCodesSearchButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationStudentCodesSearchButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 278, 174, -1));

4775 jLabel13.setText("Scores");
        qualificationPane.add(jLabel13, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 20, 174, -1));

4780 qualificationScoresTextField.setEnabled(false);
        qualificationPane.add(qualificationScoresTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 44, 174, -1));

4785 qualificationScoresList.setEnabled(false);
        jScrollPane12.setViewportView(qualificationScoresList);

        qualificationPane.add(jScrollPane12, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 80, 174, 191));

4790 qualificationScoresAddButton.setText("Add");
        qualificationScoresAddButton.setEnabled(false);

```

```

        qualificationPane.add(qualificationScoresAddButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 278, -1, -1));

4805    qualificationScoresRemoveButton.setText("Remove");
        qualificationScoresRemoveButton.setEnabled(false);
        qualificationPane.add(qualificationScoresRemoveButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(1021, 278, -1, -1));

4810    qualificationResetButton.setText("Reset");
        qualificationResetButton.setEnabled(false);
        qualificationResetButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationResetButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationResetButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 344, 358, -1));

4815

4820    qualificationExportButton.setText("Export");
        qualificationExportButton.setEnabled(false);
        qualificationExportButton.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationExportButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationExportButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 344, 358, -1));

4825

4830    qualificationSaveToggleButton.setText("Save");
        qualificationSaveToggleButton.setEnabled(false);
        qualificationSaveToggleButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                qualificationSaveToggleButtonMouseReleased(evt);
            }
        });
        qualificationPane.add(qualificationSaveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 377, 1094, -1));

4835

4840    qualificationApproveIncompleteToggleButton.setText("Approve incomplete");
        qualificationApproveIncompleteToggleButton.setEnabled(false);
        qualificationPane.add(qualificationApproveIncompleteToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 344, 174, -1));

4845    qualificationApproveCompleteToggleButton.setText("Approve complete");
        qualificationApproveCompleteToggleButton.setEnabled(false);
        qualificationPane.add(qualificationApproveCompleteToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 344, 174, -1));

4850    tabbedPane.addTab("Qualification", qualificationPane);

        jLabel14.setText("Original Oratory");

4855    finalistsOriginalOratoryList.setEnabled(false);
        jScrollPane13.setViewportView(finalistsOriginalOratoryList);

        finalistsProgressBar.setString("Progress");
        finalistsProgressBar.setStringPainted(true);

4860    jLabel15.setText("Oral Interpretation");

        finalistsOralInterpretationList.setEnabled(false);
        jScrollPane14.setViewportView(finalistsOralInterpretationList);

4865    jLabel16.setText("Impromptu Speaking");

        finalistsImpromptuSpeakingList.setEnabled(false);
        jScrollPane15.setViewportView(finalistsImpromptuSpeakingList);

4870    jLabel17.setText("Duet Acting");

        finalistsDuetActingList.setEnabled(false);
        jScrollPane16.setViewportView(finalistsDuetActingList);

4875    jLabel18.setText("Debate semifinalists");

        finalistsDebateSemifinalistsAList.setEnabled(false);
        jScrollPane17.setViewportView(finalistsDebateSemifinalistsAList);

4880

```

```

        finalistsDebateSemifinalistsSetWinnerAButton.setText("Set winner A");
        finalistsDebateSemifinalistsSetWinnerAButton.setEnabled(false);

4885      jLabel19.setText("Debate Finalists");
        finalistsDebateFinalistsList.setEnabled(false);
        jScrollPane18.setViewportView(finalistsDebateFinalistsList);

4890      finalistsResetButton.setText("Reset");
        finalistsResetButton.setEnabled(false);

        finalistsExportButton.setText("Export");
        finalistsExportButton.setEnabled(false);

4895      finalistsApproveToggleButton.setText("Approve");
        finalistsApproveToggleButton.setEnabled(false);

        finalistsSaveToggleButton.setText("Save");
        finalistsSaveToggleButton.setEnabled(false);

4900      finalistsDebateSemifinalistsBList.setEnabled(false);
        jScrollPane33.setViewportView(finalistsDebateSemifinalistsBList);

4905      finalistsDebateSemifinalistsSetWinnerBButton.setText("Set winner B");
        finalistsDebateSemifinalistsSetWinnerBButton.setEnabled(false);

        org.jdesktop.layout.GroupLayout finalistsPaneLayout = new
        org.jdesktop.layout.GroupLayout(finalistsPane);
        finalistsPane.setLayout(finalistsPaneLayout);
        finalistsPaneLayout.setHorizontalGroup(
            finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(finalistsPaneLayout.createSequentialGroup()
                    .addGap(20, 20, 20)
4915            .add(finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(finalistsPaneLayout.createSequentialGroup()
                    .addGap(10, 10, 10)
                    .add(jLabel14, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(10, 10, 10)
                    .add(jLabel15, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(10, 10, 10)
                    .add(jLabel16, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(10, 10, 10)
                    .add(jLabel17, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(10, 10, 10)
                    .add(jLabel18, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .add(10, 10, 10)
                    .add(jLabel19, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                    .add(finalistsPaneLayout.createSequentialGroup()
                        .addGap(10, 10, 10)
                        .add(finalistsExportButton,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 358,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(10, 10, 10)
                        .add(finalistsApproveToggleButton,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 358,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(10, 10, 10)
                        .add(finalistsResetButton,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 358,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(finalistsSaveToggleButton,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 1094,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(finalistsProgressBar,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 1094,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                        .add(finalistsPaneLayout.createSequentialGroup()
                            .addGap(10, 10, 10)
                            .add(jScrollPane13,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                            .add(10, 10, 10)
                        )
                    )
                )
            )
        )
    )

```

```

4960           .add(jScrollPane14,
4960           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4960           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
4960               .add(10, 10, 10)
4960               .add(jScrollPane15,
4960               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4960               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
4960                   .add(10, 10, 10)
4960                   .add(jScrollPane16,
4960                   org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4960                   org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
4960                       .add(10, 10, 10)
4970
4970           .add(finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
4970               .add(finalistsDebateSemifinalistsSetWinnerAButton,
4970               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4970               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
4970                   .add(finalistsPaneLayout.createSequentialGroup())
4975
4975           .add(finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.CENTER)
4975               .add(jScrollPane17,
4980           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4980           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
4980               .add(jScrollPane33,
4980               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4980               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
4985           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
4985               .add(jScrollPane18,
4985               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4985               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
4985                   .add(finalistsDebateSemifinalistsSetWinnerBButton,
4990           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
4990           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
4990               .add(49, 49, 49))
4995
4995           );
5000           finalistsPaneLayout.setVerticalGroup(
5000
5000           finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5000               .add(finalistsPaneLayout.createSequentialGroup()
5000                   .add(20, 20, 20)
5000
5000               .add(finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5000                   .add(jLabel14)
5000                   .add(jLabel15)
5000                   .add(jLabel16)
5000                   .add(jLabel17)
5000                   .add(jLabel18)
5000                   .add(jLabel19)))
5000                       .add(44, 44, 44)
5010
5010           .add(finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING,
5010           false)
5010               .add(jScrollPane13,
5010               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 191,
5010               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5010                   .add(jScrollPane14,
5010                   org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 191,
5010                   org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5010                       .add(jScrollPane15,
5010                       org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 191,
5010                       org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5010                           .add(jScrollPane16,
5010                           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 191,
5010                           org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5010                               .add(jScrollPane18,
5010                               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 191,
5010                               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5020
5020           .add(finalistsPaneLayout.createSequentialGroup()
5020               .add(jScrollPane17,
5020               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 93,
5020               org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5020                   .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
5020                   .add(jScrollPane33, 0, 0, Short.MAX_VALUE)))
5020                   .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
5020                   .add(finalistsDebateSemifinalistsSetWinnerAButton)
5025
5025           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
5025                   .add(finalistsDebateSemifinalistsSetWinnerBButton)
5025                     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
5030
5030           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

```

```

5040     .add(finalistsPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5041             .add(finalistsExportButton)
5042             .add(finalistsApproveToggleButton)
5043             .add(finalistsResetButton))
5044             .add(4, 4, 4)
5045             .add(finalistsSaveToggleButton)
5046             .add(9, 9, 9)
5047             .add(finalistsProgressBar,
5048                 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
5049                 org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
5050                 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
5051         );
5052
5053         tabbedPane.addTab("Finalists", finalistsPane);
5054
5055         finalsPane.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
5056
5057         jLabel20.setText("Event");
5058         finalsPane.add(jLabel20, new org.netbeans.lib.awtextra.AbsoluteConstraints(20,
5059             20, 174, -1));
5060
5061         finalsEventList.setEnabled(false);
5062         jScrollPane19.setViewportView(finalsEventList);
5063
5064         finalsPane.add(jScrollPane19, new
5065             org.netbeans.lib.awtextra.AbsoluteConstraints(20, 80, 174, 191));
5066
5067         finalsProgressBar.setString("Progress");
5068         finalsProgressBar.setStringPainted(true);
5069         finalsPane.add(finalsProgressBar, new
5070             org.netbeans.lib.awtextra.AbsoluteConstraints(20, 415, 1094, -1));
5071
5072         jLabel21.setText("Round");
5073         finalsPane.add(jLabel21, new
5074             org.netbeans.lib.awtextra.AbsoluteConstraints(204, 20, 174, -1));
5075
5076         finalsRoundTextField.setEnabled(false);
5077         finalsPane.add(finalsRoundTextField, new
5078             org.netbeans.lib.awtextra.AbsoluteConstraints(204, 44, 174, -1));
5079
5080         finalsRoundList.setEnabled(false);
5081         jScrollPane20.setViewportView(finalsRoundList);
5082
5083         finalsPane.add(jScrollPane20, new
5084             org.netbeans.lib.awtextra.AbsoluteConstraints(204, 80, 174, 191));
5085
5086         finalsRoundEditNameButton.setText("Edit name");
5087         finalsRoundEditNameButton.setEnabled(false);
5088         finalsPane.add(finalsRoundEditNameButton, new
5089             org.netbeans.lib.awtextra.AbsoluteConstraints(204, 278, 174, -1));
5090
5091         jLabel22.setText("Room");
5092         finalsPane.add(jLabel22, new
5093             org.netbeans.lib.awtextra.AbsoluteConstraints(388, 20, 174, -1));
5094
5095         finalsRoomTextField.setEnabled(false);
5096         finalsPane.add(finalsRoomTextField, new
5097             org.netbeans.lib.awtextra.AbsoluteConstraints(388, 44, 174, -1));
5098
5099         finalsRoomList.setEnabled(false);
5100         jScrollPane21.setViewportView(finalsRoomList);
5101
5102         finalsPane.add(jScrollPane21, new
5103             org.netbeans.lib.awtextra.AbsoluteConstraints(388, 80, 174, 191));
5104
5105         finalsRoomEditNameButton.setText("Edit name");
5106         finalsRoomEditNameButton.setEnabled(false);
5107         finalsPane.add(finalsRoomEditNameButton, new
5108             org.netbeans.lib.awtextra.AbsoluteConstraints(388, 278, 174, -1));
5109
5110         jLabel23.setText("Judges");
5111         finalsPane.add(jLabel23, new
5112             org.netbeans.lib.awtextra.AbsoluteConstraints(572, 20, 174, -1));
5113
5114         finalsJudgesList.setEnabled(false);
5115         jScrollPane22.setViewportView(finalsJudgesList);

```

```

5115     finalsPane.add(jScrollPane22, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 80, 174, 191));

5120         finalsJudgesSubstituteJudgeButton.setText("Substitute judge");
finalsJudgesSubstituteJudgeButton.setEnabled(false);
finalsPane.add(finalsJudgesSubstituteJudgeButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(572, 278, 174, -1));

5125         jLabel24.setText("Student codes");
finalsPane.add(jLabel24, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 20, 174, -1));

5130         finalsStudentCodesTextField.setEnabled(false);
finalsPane.add(finalsStudentCodesTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 44, 174, -1));

5135         finalsStudentCodesList.setEnabled(false);
jScrollPane23.setViewportView(finalsStudentCodesList);

5140         finalsPane.add(jScrollPane23, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 80, 174, 191));

5145         finalsStudentCodesSearchButton.setText("Search");
finalsStudentCodesSearchButton.setEnabled(false);
finalsPane.add(finalsStudentCodesSearchButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 278, 174, -1));

5150         jLabel25.setText("Rankings");
finalsPane.add(jLabel25, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 20, 174, -1));

5155         finalsRankingsTextField.setEnabled(false);
finalsPane.add(finalsRankingsTextField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 44, 174, -1));

5160         finalsRankingsList.setEnabled(false);
jScrollPane24.setViewportView(finalsRankingsList);

5165         finalsRankingsAddButton.setText("Add");
finalsRankingsAddButton.setEnabled(false);
finalsPane.add(finalsRankingsAddButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 278, -1, -1));

5170         finalsRankingsRemoveButton.setText("Remove");
finalsRankingsRemoveButton.setEnabled(false);
finalsPane.add(finalsRankingsRemoveButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(1021, 278, -1, -1));

5175         finalsResetButton.setText("Reset");
finalsResetButton.setEnabled(false);
finalsPane.add(finalsResetButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(756, 344, 358, -1));

5180         finalsExportButton.setText("Export");
finalsExportButton.setEnabled(false);
finalsPane.add(finalsExportButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 344, 358, -1));

5185         finalsApproveToggleButton.setText("Approve");
finalsApproveToggleButton.setEnabled(false);
finalsPane.add(finalsApproveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(388, 344, 358, -1));

5190         finalsSaveToggleButton.setText("Save");
finalsSaveToggleButton.setEnabled(false);
finalsPane.add(finalsSaveToggleButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(20, 377, 1094, -1));

5185         finalsStudentsVoteList.setEnabled(false);
jScrollPane34.setViewportView(finalsStudentsVoteList);

5190         finalsPane.add(jScrollPane34, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 233, 174, 38));

         finalsStudentsVoteSetStudentsVoteButton.setText("Set students' vote");
finalsStudentsVoteSetStudentsVoteButton.setEnabled(false);

```

```

5195     finalsPane.add(finalsStudentsVoteSetStudentsVoteButton, new
org.netbeans.lib.awtextra.AbsoluteConstraints(940, 311, 174, -1));
      tabbedPane.addTab("Finals", finalsPane);
5200     jLabel27.setText("Original Oratory");
      winnersOriginalOratoryFirstPlaceList.setEnabled(false);
      jScrollPane26.setViewportView(winnersOriginalOratoryFirstPlaceList);
5205     jLabel28.setText("Oral Interpretation");
      jLabel29.setText("Impromptu Speaking");
      jLabel30.setText("Duet Acting");
5210     jLabel31.setText("Debate");
      winnersSaveToggleButton.setText("Save");
      winnersSaveToggleButton.setEnabled(false);
5215     winnersOriginalOratorySecondPlaceList.setEnabled(false);
      jScrollPane35.setViewportView(winnersOriginalOratorySecondPlaceList);
      winnersOriginalOratoryThirdPlaceList.setEnabled(false);
      jScrollPane36.setViewportView(winnersOriginalOratoryThirdPlaceList);
5220     winnersOriginalOratoryStudentChoiceList.setEnabled(false);
      jScrollPane37.setViewportView(winnersOriginalOratoryStudentChoiceList);
      winnersOralInterpretationFirstPlaceList.setEnabled(false);
      jScrollPane27.setViewportView(winnersOralInterpretationFirstPlaceList);
5225     winnersOralInterpretationSecondPlaceList.setEnabled(false);
      jScrollPane38.setViewportView(winnersOralInterpretationSecondPlaceList);
      winnersOralInterpretationThirdPlaceList.setEnabled(false);
      jScrollPane39.setViewportView(winnersOralInterpretationThirdPlaceList);
5230     winnersOralInterpretationStudentChoiceList.setEnabled(false);
      jScrollPane40.setViewportView(winnersOralInterpretationStudentChoiceList);
      winnersImpromptuSpeakingFirstPlaceList.setEnabled(false);
      jScrollPane28.setViewportView(winnersImpromptuSpeakingFirstPlaceList);
5235     winnersImpromptuSpeakingSecondPlaceList.setEnabled(false);
      jScrollPane41.setViewportView(winnersImpromptuSpeakingSecondPlaceList);
      winnersImpromptuSpeakingThirdPlaceList.setEnabled(false);
      jScrollPane42.setViewportView(winnersImpromptuSpeakingThirdPlaceList);
5240     winnersImpromptuSpeakingStudentChoiceList.setEnabled(false);
      jScrollPane43.setViewportView(winnersImpromptuSpeakingStudentChoiceList);
      winnersDuetActingFirstPlaceList.setEnabled(false);
      jScrollPane29.setViewportView(winnersDuetActingFirstPlaceList);
5245     winnersDuetActingSecondPlaceList.setEnabled(false);
      jScrollPane44.setViewportView(winnersDuetActingSecondPlaceList);
      winnersDuetActingThirdPlaceList.setEnabled(false);
      jScrollPane45.setViewportView(winnersDuetActingThirdPlaceList);
5250     winnersDuetActingStudentChoiceList.setEnabled(false);
      jScrollPane46.setViewportView(winnersDuetActingStudentChoiceList);
      winnersDebateFirstPlaceList.setEnabled(false);
      jScrollPane30.setViewportView(winnersDebateFirstPlaceList);
5255     winnersDebateSecondPlaceList.setEnabled(false);
      jScrollPane47.setViewportView(winnersDebateSecondPlaceList);
      winnersDebateThirdPlaceList.setEnabled(false);
      jScrollPane48.setViewportView(winnersDebateThirdPlaceList);
5260     winnersDebateStudentChoiceList.setEnabled(false);
      jScrollPane49.setViewportView(winnersDebateStudentChoiceList);
      jLabel32.setText("First place");

```

```

5275     jLabel33.setText("Second place");
      jLabel34.setText("Third place");
      jLabel35.setText("Student choice");
5280     jLabel26.setText("Congratulations!");
      winnersExportButton.setText("Export");
      winnersExportButton.setEnabled(false);
5285     org.jdesktop.layout.GroupLayout winnersPaneLayout = new
      org.jdesktop.layout.GroupLayout(winnersPane);
      winnersPane.setLayout(winnersPaneLayout);
      winnersPaneLayout.setHorizontalGroup(
5290         winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
             .add(winnersPaneLayout.createSequentialGroup()
                 .addGap(20, 20, 20)
5295             .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING,
false)
                     .add(winnersPaneLayout.createSequentialGroup()
                         .addGap(112, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5300                         .add(72, 72, 72)
                         .add(jLabel27, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                         .addGap(10, 10, 10)
                         .add(jLabel28, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                         .addGap(10, 10, 10)
5305                         .add(jLabel29, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                         .addGap(10, 10, 10)
                         .add(jLabel30, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                         .addGap(10, 10, 10)
5310                         .add(jLabel31, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
174, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                     .add(winnersSaveToggleButton,
5315             org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 1094, Short.MAX_VALUE)
                     .add(winnersPaneLayout.createSequentialGroup()
5320                         .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING,
false)
                             .add(winnersPaneLayout.createSequentialGroup()
                                 .addGap(14, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5325                             .add(jScrollPane26,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                             .add(org.jdesktop.layout.GroupLayout.LEADING,
winnersPaneLayout.createSequentialGroup()
5330                         .add(jLabel34,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 105,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                         .addGap(79, 79, 79)
                         .add(jScrollPane36,
5335 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                             .add(org.jdesktop.layout.GroupLayout.LEADING,
winnersPaneLayout.createSequentialGroup()
5340                         .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                             .add(jLabel33,
5345 org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 126,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                             .add(jLabel35,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 75, Short.MAX_VALUE)))
                     .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5350                         .add(org.jdesktop.layout.GroupLayout.TRAILING,
jScrollPane35, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)

```

```

5355                               .add(org.jdesktop.layout.GroupLayout.TRAILING,
jScrollPane37, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
                           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5356                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                           .add(jScrollPane27,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane38,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane39,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane40,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5357                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                           .add(jScrollPane28,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane41,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane42,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane43,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5358                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                           .add(jScrollPane29,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane44,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane45,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane46,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5359                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                           .add(jScrollPane49,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane30,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane47,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                           .add(jScrollPane48,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 174,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                           .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5360                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                           .add(winnersExportButton,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                           .add(49, 49, 49))

5361   );
winnersPaneLayout.setVerticalGroup(
5362
winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                           .add(winnersPaneLayout.createSequentialGroup()
                           .add(20, 20, 20))
5363
                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5364
                           .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                           .add(jLabel127)
                           .add(jLabel126)))
5365

```

```

        .add(jLabel28)
        .add(jLabel29)
        .add(jLabel30)
        .add(jLabel31)))
    .add(44, 44, 44)

    .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)
        .add(winnersPaneLayout.createSequentialGroup()
            .add(jScrollPane30,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane47,
                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                    .add(jScrollPane48,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                            .add(jScrollPane49,
                                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
                                    .add(winnersPaneLayout.createSequentialGroup()
                                        .add(jScrollPane29,
                                            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                                            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                                                .add(jScrollPane44,
                                                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                                                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                                                        .add(jScrollPane45,
                                                            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                                                            org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                                                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                                                                .add(jScrollPane46,
                                                                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                                                                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
    5470
        .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING,
false)
            .add(winnersPaneLayout.createSequentialGroup()
                .add(jScrollPane28,
                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                    .add(jScrollPane41,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                            .add(jScrollPane42,
                                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                                    .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                                    .add(jScrollPane43, 0, 0, Short.MAX_VALUE))
                                    .add(org.jdesktop.layout.GroupLayout.TRAILING,
winnersPaneLayout.createSequentialGroup()
            .add(jScrollPane27,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(jScrollPane38,
                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                    org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                    .add(jScrollPane39,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                            .add(jScrollPane40, 0, 0, Short.MAX_VALUE))
                            .add(winnersPaneLayout.createSequentialGroup()
    5500
        .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(jScrollPane26,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .add(jLabel32))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

```

```

5510 .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5511     .add(jScrollPane35,
5512         org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
5513         org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5514     .add(jLabel133))
5515     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5516 .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5517     .add(jScrollPane36,
5518         org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
5519         org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5520     .add(jLabel134))
5521     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

5522 .add(winnersPaneLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
5523     .add(jScrollPane37,
5524         org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
5525         org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
5526     .add(jLabel135)))
5527     .add(88, 88, 88)
5528     .add(winnersExportButton)
5529     .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
5530     .add(winnersSaveToggleButton)
5531     .add(29, 29, 29))
5532 );
5533
5534 tabbedPane.addTab("Winners", winnersPane);
5535
5536 tabbedPane.setSelectedIndex(1);
5537
5538 getContentPane().add(tabbedPane, new
5539 org.netbeans.lib.awtextra.AbsoluteConstraints(10, 110, 1158, 490));
5540
5541 jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
5542 jLabel1.setText("NESDA Tournament Manager 2013 by zbyněk Stara");
5543 getContentPane().add(jLabel1, new
5544 org.netbeans.lib.awtextra.AbsoluteConstraints(20, 20, 1140, -1));
5545
5546 jLabel40.setIcon(new
5547 javax.swing.ImageIcon(getClass().getResource("/graphics/NesdaHB2006-
5548 72_001small.jpg"))); // NOI18N
5549 getContentPane().add(jLabel40, new
5550 org.netbeans.lib.awtextra.AbsoluteConstraints(1020, 20, -1, -1));
5551
5552 jLabel41.setIcon(new
5553 javax.swing.ImageIcon(getClass().getResource("/graphics/NesdaHB2006-
5554 72_001small.jpg"))); // NOI18N
5555 getContentPane().add(jLabel41, new
5556 org.netbeans.lib.awtextra.AbsoluteConstraints(40, 20, -1, -1));
5557
5558 overallProgressBar.setValue(33);
5559 overallProgressBar.setString("Overall progress");
5560 overallProgressBar.setStringPainted(true);
5561 getContentPane().add(overallProgressBar, new
5562 org.netbeans.lib.awtextra.AbsoluteConstraints(20, 600, 1140, -1));
5563
5564 fileMenuItem.setText("File");
5565
5566 openMenuItem.setText("Open");
5567 fileMenuItem.add(openMenuItem);
5568
5569 saveMenuItem.setText("Save");
5570 fileMenuItem.add(saveMenuItem);
5571
5572 saveAsMenuItem.setText("Save As ...");
5573 fileMenuItem.add(saveAsMenuItem);
5574
5575 exitMenuItem.setText("Exit");
5576 exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
5577     public void actionPerformed(java.awt.event.ActionEvent evt) {
5578         exitMenuItemActionPerformed(evt);
5579     }
5580 });
5581 fileMenuItem.add(exitMenuItem);
5582
5583 menuBar.add(fileMenuItem);
5584
5585 editMenuItem.setText("Edit");

```

```

5590         cutMenuItem.setText("Cut");
5591         editMenu.add(cutMenuItem);

5595         copyMenuItem.setText("Copy");
5596         editMenu.add(copyMenuItem);

5598         pasteMenuItem.setText("Paste");
5599         editMenu.add(pasteMenuItem);

5600         deleteMenuItem.setText("Delete");
5601         editMenu.add(deleteMenuItem);

5604         menuBar.add(editMenu);

5605         helpMenu.setText("Help");

5606         contentsMenuItem.setText("Contents");
5607         helpMenu.add(contentsMenuItem);

5608         aboutMenuItem.setText("About");
5609         helpMenu.add(aboutMenuItem);

5612         menuBar.add(helpMenu);

5613         setJMenuBar(menuBar);

5614         pack();
5615         }// </editor-fold>

5616 // GENERAL:
5617     // hadling the event when the user clicks the exitMenuItem
5618     private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
5619         System.exit(0);
5620     }

5621 // SETTINGS TAB:
5622     // hadling the event when the user clicks the settingsLoadButton
5623     private void settingsLoadButtonMouseReleased(java.awt.event.MouseEvent evt) {
5624         try { // try to load a file
5625             this.loadFile();
5626
5627             // set the setting tab's settings values to be the same as those in
5628             database
5629             this.synchronizeTFSSettings(database);
5630
5631             // which tabs were saved in the file?
5632             if (approvedSectionsIndex == 0) { // only settings
5633                 this.enableSettingsTabActionBar();
5634             } else if(approvedSectionsIndex == 1) { // settings and participants
5635                 tabbedPane.setSelectedIndex(1);
5636
5637                 this.enableParticipantsTabActionBar();
5638
5639                 this.updateParticipantsProgressBar();
5640             } else if(approvedSectionsIndex == 2) { // settings, participants, and
5641                 qualification
5642                 tabbedPane.setSelectedIndex(2);
5643
5644                 this.enableQualificationTabActionBar();
5645
5646                 this.updateQualificationProgressBar(100);
5647             }
5648         } catch (UserIOException ex) { // if the user cancels the file chooser window
5649             // this is acceptable - no exception dialog is invoked
5650         } catch (FileNotFoundException ex) { // if the file is illegal
5651             // show the following exception dialog
5652             exceptionDialogTextArea.setText("The file you tried to open is not "
5653                 + "saved in a format compatible with this program. Make "
5654                 + "sure you are trying to load in a valid file and try "
5655                 + "to open it once more.");
5656
5657             exceptionDialog.setVisible(true);
5658         } catch (IOException ex) { // if there is a general IOException
5659             // show the following exception dialog
5660             exceptionDialogTextArea.setText("An input/output exception has "
5661                 + "occurred during the loading of the file. Try to load "
5662                 + "the file once more.");
5663     }

```

```

            exceptionDialog.setVisible(true);
5670    }

        // hadling the event when the user clicks the settingsApproveToggleButton
        private void settingsApproveToggleButtonMouseReleased(java.awt.event.MouseEvent
evt) {
            if (settingsApproveToggleButton.isSelected()) { // if the approve button was
selected by this action
                try { // try to approve settings
                    // set the database settings to correspond to settings in the settings
tab
                    this.changeDatabaseSettings(database);

                    // see if the settings are correct
                    if (this.approveDatabaseSettings(database).equals("")) { // if the
settings are correct
                        this.enableParticipantsTabActionBox();

                        tabbedPane.setSelectedIndex(1);
                    } else { // if the settings are incorrect
                        // show an exception dialog listing all the settings that are
incorrect
5685
                        settingsErrorDialogTextArea.setText(this.approveDatabaseSettings(database));
                        settingsErrorDialog.setVisible(true);

                        // deselect the approve button
                        settingsApproveToggleButton.setSelected(false);
                    }
                } catch (IllegalArgumentException ex) { // if, for whatever reason, the
error handling failed at previous levels
                    // show the following exception dialog
                    exceptionDialogTextArea.setText("Some of the settings values "
+ "were incorrect and this could not be resolved by "
+ "the program. Look for erroneous values in the "
+ "settings text fields and correct them yourself.");
5705
                    exceptionDialog.setVisible(true);

                    settingsApproveToggleButton.setSelected(false);
                }
            } else { // if approveToggleButton was deselected by this action
                this.enableSettingsTabActionBox();
            }
        }

5715    // hadling the event when the user clicks the settingsResetButton
    private void settingsResetButtonMouseReleased(java.awt.event.MouseEvent evt) {
        // reset database settings to be the same as the default values in database
        this.resetDatabaseSettings(database);

5720    // set the settings tab's values to correspond to those in the database
        this.synchronizeTFSSettings(database);
    }

5725    // hadling the event when the user clicks the settingsSaveToggleButton
    private void settingsSaveToggleButtonMouseReleased(java.awt.event.MouseEvent evt)
{
        if (settingsSaveToggleButton.isSelected()) { // if the save button was
selected by this action
            try { // try to save the settings
                approvedSectionsIndex = 0; // current section is settings = 0

                this.saveFile();

                settingsApproveToggleButton.setEnabled(false); // disable approve
button until save button is deselected
            } catch (UserIOException ex) { // if the file chooser dialog was cancelled
by the user
                settingsSaveToggleButton.setSelected(false);
            } catch (IOException ex) { // if there was a general IO exception
                settingsSaveToggleButton.setSelected(false);

                // show the following exception dialog
                exceptionDialogTextArea.setText("An input/output exception has "
+ "occurred during the saving of the file. Try to save "
+ "the file once more.");
5740
            }
        }
    }

```

```

                exceptionDialog.setVisible(true);
            }
        } else { // if the save button was deselected by this action
            settingsApproveToggleButton.setEnabled(true); // enable the approve button
        }
    }

// PARTICIPANTS TAB:
// SCHOOLS:
// hadling the event when the user clicks the participantsSchoolsAddButton
private void participantsSchoolsAddButtonMouseReleased(java.awt.event.MouseEvent evt) {
    String newSchoolName = participantsSchoolsTextField.getText();

    try {
        database.searchSchool(newSchoolName); // should result in a NoSuchElementException
    }

    // if an element with the same name is already in the database:
    // show the following exception dialog
    illegalActionDialogTextArea.setText("The name you typed in is "
        + "already taken. All schools must have unique names. If "
        + "you really wish to add a school with this name, first "
        + "locate the existing school and rename it.");
    illegalActionDialog.setVisible(true);

} catch (IllegalArgumentException ex) { // if the name is an empty string
    // set the new school's name to a generic name with the current new no
    name school number and insert the new school
    newSchoolName = "<No name " + currentNewNoNameSchoolNumber + ">";
    database.insertSchool(newSchoolName);

    // increment the current new no name school number
    currentNewNoNameSchoolNumber += 1;

    // increment the number of schools
    schoolsNumber += 1;

    // set the progress bar to reflect the change in the number of schools
    this.updateParticipantsProgressBar();

    // new assignment of codes will be needed
    studentChanges = true;

    // empty the schools text field
    participantsSchoolsTextField.setText("");

    // find the new school in the database
    int newSchoolIndex = database.searchSchool(newSchoolName);
    this.autoselectSchoolActionBox(newSchoolIndex);

    // set the student codes button to the correct status
    this.recheckStudentCodesActionBox();
} catch (NoSuchElementException ex) { // if the name was not encountered in
the database
    // insert the school
    database.insertSchool(newSchoolName);

    // increment the number of schools in the database
    schoolsNumber += 1;

    // new assignment of codes will be needed
    studentChanges = true;

    // empty the schools text field
    participantsSchoolsTextField.setText("");

    // find the new school in the database
    int newSchoolIndex = database.searchSchool(newSchoolName);
    this.autoselectSchoolActionBox(newSchoolIndex);

    // set the student codes button to the correct status
    this.recheckStudentCodesActionBox();
} finally { // do in any case
    participantsSchoolsTextField.grabFocus();
}
}

// hadling the event when the user clicks the participantsSchoolsRemoveButton

```

```

    private void
participantsSchoolsRemoveButtonMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

5830    if (schoolsIndex >= 0) { // if a school is selected in the schoolsList
        // delete the currently selected school
        School deletedSchool = (School)
database.deleteSchool(currentSchool.getName());

5835        // decrement the number of schools
schoolsNumber -= 1;

        // set the progress bar to reflect the situation
this.updateParticipantsProgressBar();

5840        // decrease the number of teachers and students by their numbers in the
deleted school
        teachersNumber -= deletedSchool.getTeacherTreeSize();
        studentsNumber -= deletedSchool.getStudentTreeSize();

5845        // reorganization of student codes will be necessary (a letter
disappeared)
        studentChanges = true;
    }

5850    if (database.getSchoolTreeSize() != 0) { // if there is still a school in the
schoolTree after the removal
        // select the first school from the list
        this.autoselectSchoolActionBox(0);

5855    if (studentsNumber != 0) { // if there are still some students in the
database
        // enable searching for students
        this.toggleStudentSearchActionBox(true);

5860        // enable the assign codes button
        this.recheckStudentCodesActionBox();

        } else { // if there are no more students in the database
            // disable searching for students
            this.toggleStudentSearchActionBox(false);

            // disable the assign codes button
            this.disableStudentCodesActionBox();
        }

5870    } else { // if there is no school left in the schoolTree after the removal
        // disable everything (but text field and add button) about schools
        participantsSchoolsList.setEnabled(false);
        participantsSchoolsRemoveButton.setEnabled(false);
        participantsSchoolsEditButton.setEnabled(false);
        participantsSchoolsSearchButton.setEnabled(false);

        // set the schoolsList to be emptyModel
        participantsSchoolsList.setModel(emptyModel);

5880        // disable everything about teachers
        participantsTeachersTextField.setEnabled(false);
        participantsTeachersList.setEnabled(false);
        participantsTeachersAddButton.setEnabled(false);
        participantsTeachersRemoveButton.setEnabled(false);
        participantsTeachersSearchButton.setEnabled(false);
        participantsTeachersEditButton.setEnabled(false);

        // set the teachersList to be noSelectionModel
        participantsTeachersList.setModel(noSelectionModel);

5890        // disable everything about students
        participantsStudentsTextField.setEnabled(false);
        participantsStudentsList.setEnabled(false);
        participantsStudentsAddButton.setEnabled(false);
        participantsStudentsRemoveButton.setEnabled(false);
        participantsStudentsSearchButton.setEnabled(false);
        participantsStudentsEditButton.setEnabled(false);

5900        // set the studentsList to be noSelectionModel
        participantsStudentsList.setModel(noSelectionModel);

        // disable student codes
        this.disableStudentCodesActionBox();
    }
}

```

```

5905
    // set the studentCodesList to be noSelectionModel
    participantsStudentsList.setModel(noSelectionModel);

5910
    // disable assign events
    this.disableAssignEventsActionBox();

    // disable assign pairs
    participantsAssignPairsEventsList.setEnabled(false);
    participantsAssignPairsLeftList.setEnabled(false);
    participantsAssignPairsRightList.setEnabled(false);
    participantsAssignPairsPairsList.setEnabled(false);
    participantsAssignPairsPairButton.setEnabled(false);
    participantsAssignPairsRemovePairButton.setEnabled(false);

5915
    // set the lists to be blankModels
    participantsAssignPairsLeftList.setModel(blankModel);
    participantsAssignPairsRightList.setModel(blankModel);
    participantsAssignPairsPairsList.setModel(blankModel);
}

5920
    participantsSchoolsTextField.grabFocus();
}

5925
// hadling the event when the user clicks the participantsSchoolsEditButton
private void participantsSchoolsEditButtonMouseReleased(java.awt.event.MouseEvent
evt) {
    String newName = participantsSchoolsTextField.getText();

    try {
        database.searchSchool(newName); // should result in a NoSuchElementException
    exception

        // if an element with the same name is already in the database:
        // show the following exception dialog
        illegalActionDialogTextArea.setText("The name you typed in is "
            + "already taken. All schools must have unique names. If "
            + "you really wish to add a school with this name, first "
            + "locate the existing school and rename it.");
    }

5930
    illegalActionDialog.setVisible(true);
} catch (IllegalArgumentException ex) { // if the name is an empty string
    // do nothing
} catch (NoSuchElementException ex) { // if the name was not encountered in
the database
    // rename the school at the schools index
    database.editSchool(schoolsIndex, newName);

    // changes were made to the database - new assignment of student codes
will be needed
    studentChanges = true;

    // empty the school text field
    participantsSchoolsTextField.setText("");
}

5935
    // find the edited school in the database
    int editedSchoolIndex = database.searchSchool(newName);
    this.refreshSchoolActionBox(editedSchoolIndex);

    // set the student codes button to the correct status
    this.recheckStudentCodesActionBox();
} finally { // do in any case
    participantsSchoolsTextField.grabFocus();
}
}

5940
// hadling the event when the user clicks the participantsSchoolsSearchButton
private void
participantsSchoolsSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
    String searchedName = participantsSchoolsTextField.getText();
    int searchedSchoolIndex;

    try {
        // empty the school text field
        participantsSchoolsTextField.setText("");
    }

5945
        // search for the school
        searchedSchoolIndex = database.searchSchool(searchedName); // should
continue
}
}

```

```

5985      // select the found school
      this.refreshSchoolActionBox(searchedSchoolIndex);
    } catch (IllegalArgumentException ex) { // if the name is an empty string
      // do nothing
    } catch (NoSuchElementException ex) { // if the name was not encountered in
5990  the database
      // empty the schools text field
      participantsSchoolsTextField.setText("");
    }
5995      // set schools index to be -1
      this.refreshSchoolActionBox(-1);
    } finally { // do in any case
      participantsSchoolsTextField.grabFocus();
    }
6000  }
6005      // hadling the event when the user makes a selection in the
participantsSchoolsList
      private void
participantsSchoolsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
6010      schoolsIndex = participantsSchoolsList.getSelectedIndex();

      School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

      if (schoolsIndex != -1) { // if schools index was set to be something (not -1)
        // SCHOOLS:
        // enable the selected-school-specific buttons (remove, edit)
        participantsSchoolsRemoveButton.setEnabled(true);
        participantsSchoolsEditButton.setEnabled(true);

6015      // TEACHERS:
      // enable the addition of teachers to the school
        participantsTeachersTextField.setEnabled(true);
        participantsTeachersAddButton.setEnabled(true);

6020      if (currentSchool.getTeacherTreeSize() > 0) { // if there are some
teachers in the selected school
          // update and enable the teachers list
          participantsTeachersList.setEnabled(true);
          this.updateTeachersListModel(currentSchool);
        } else { // if teacherTree is empty
          // disable teachers list and set the teacher model to be the empty
model
          participantsTeachersList.setEnabled(false);
          participantsTeachersRemoveButton.setEnabled(false);
6030          participantsTeachersEditButton.setEnabled(false);

          participantsTeachersList.setModel(emptyModel);
        }
6035      // STUDENTS (+ ASSIGN EVENTS):
      // enable adding of students
      participantsStudentsTextField.setEnabled(true);
      participantsStudentsAddButton.setEnabled(true);

6040      if (currentSchool.getStudentTreeSize() > 0) { // if there are some
students in the database
        // enable studentsList
        participantsStudentsList.setEnabled(true);

6045      // show students and studentCodes for current school
        this.updateStudentsListModel(currentSchool);
        this.updateStudentCodesListModel(currentSchool);

6050      // if there is a potential for assigning pairs (minimum of 2 DA or
debate students and codes assigned)
        if (((currentSchool.getUnpairedDAStudentTreeSize() / 2) > 0) ||
((currentSchool.getUnpairedDebateStudentTreeSize() / 2) > 0)) && studentCodes) {
          // enable assign pairs events list
          participantsAssignPairsEventsList.setSelectedIndex(-1);
          participantsAssignPairsEventsList.setModel(eventsListModel);
          participantsAssignPairsEventsList.setEnabled(true);
        } else if ((currentSchool.getDAPairTreeSize() > 0) ||
6055  (currentSchool.getDebatePairTreeSize() > 0)) { // if some pairs were assigned
          // enable assign pairs events list
          participantsAssignPairsEventsList.setSelectedIndex(-1);
          participantsAssignPairsEventsList.setModel(eventsListModel);
          participantsAssignPairsEventsList.setEnabled(true);
        }
6060

```

```

    } else { // if there is no potential for pairing or removing pairs
        // disable assign pairs events list
        participantsAssignPairsEventsList.setSelectedIndex(-1);
        participantsAssignPairsEventsList.setModel(eventsListModel);
        participantsAssignPairsEventsList.setEnabled(false);
    }

6070      // disable the other assign pairs lists and set their models to be
blank
    participantsAssignPairsLeftList.setSelectedIndex(-1);
    participantsAssignPairsLeftList.setModel(blankModel);
    participantsAssignPairsLeftList.setEnabled(false);

6075      participantsAssignPairsRightList.setSelectedIndex(-1);
    participantsAssignPairsRightList.setModel(blankModel);
    participantsAssignPairsRightList.setEnabled(false);

6080      participantsAssignPairsPairsList.setSelectedIndex(-1);
    participantsAssignPairsPairsList.setModel(blankModel);
    participantsAssignPairsPairsList.setEnabled(false);

    // disable the pair and remove pair buttons
    participantsAssignPairsPairButton.setEnabled(false);
    participantsAssignPairsRemovePairButton.setEnabled(false);
} else { // if studentTree is empty
    participantsStudentsList.setEnabled(false);
    participantsStudentsRemoveButton.setEnabled(false);
    participantsStudentsEditButton.setEnabled(false);

    this.disableAssignEventsActionBar();

6095      participantsStudentsList.setModel(emptyModel); // needed - #20
    participantsStudentCodesList.setModel(emptyModel);

    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(false);

6100      participantsAssignPairsLeftList.setSelectedIndex(-1);
    participantsAssignPairsLeftList.setModel(blankModel);
    participantsAssignPairsLeftList.setEnabled(false);

6105      participantsAssignPairsRightList.setSelectedIndex(-1);
    participantsAssignPairsRightList.setModel(blankModel);
    participantsAssignPairsRightList.setEnabled(false);

    participantsAssignPairsPairsList.setSelectedIndex(-1);
    participantsAssignPairsPairsList.setModel(blankModel);
    participantsAssignPairsPairsList.setEnabled(false);

    participantsAssignPairsPairButton.setEnabled(false);
    participantsAssignPairsRemovePairButton.setEnabled(false);
} else { // if schoolsIndex == -1
    // SCHOOLS:
    participantsSchoolsRemoveButton.setEnabled(false);
    participantsSchoolsEditButton.setEnabled(false);

6120      // TEACHERS:
    participantsTeachersList.setModel(noSelectionModel);
    participantsTeachersList.setEnabled(false);

    if (teachersNumber != 0) {
        participantsTeachersTextField.setEnabled(true);
        participantsTeachersSearchButton.setEnabled(true);
    } else {
        participantsTeachersTextField.setEnabled(false);
        participantsTeachersSearchButton.setEnabled(false);
    }

    participantsTeachersAddButton.setEnabled(false);
    participantsTeachersRemoveButton.setEnabled(false);
    participantsTeachersEditButton.setEnabled(false);

6135      // STUDENTS (+ ASSIGN EVENTS):
    this.disableAssignEventsActionBar();

6140      participantsStudentsList.setModel(noSelectionModel);
    participantsStudentsList.setEnabled(false);

```

```

participantsStudentCodesList.setModel(noSelectionModel);
participantsStudentCodesList.setEnabled(false);

6145    if (studentsNumber != 0) {
        participantsStudentsTextField.setEnabled(true);
        participantsStudentsSearchButton.setEnabled(true);
    } else {
        participantsStudentsTextField.setEnabled(false);
        participantsStudentsSearchButton.setEnabled(false);
    }

6150    participantsStudentsAddButton.setEnabled(false);
participantsStudentsRemoveButton.setEnabled(false);
participantsStudentsEditButton.setEnabled(false);

6155    // ASSIGN PAIRS:
participantsAssignPairsEventsList.setSelectedIndex(-1);
participantsAssignPairsEventsList.setModel(eventsListModel);
participantsAssignPairsEventsList.setEnabled(false);

6160    participantsAssignPairsLeftList.setSelectedIndex(-1);
participantsAssignPairsLeftList.setModel(blankModel);
participantsAssignPairsLeftList.setEnabled(false);

6165    participantsAssignPairsRightList.setSelectedIndex(-1);
participantsAssignPairsRightList.setModel(blankModel);
participantsAssignPairsRightList.setEnabled(false);

6170    participantsAssignPairsPairsList.setSelectedIndex(-1);
participantsAssignPairsPairsList.setModel(blankModel);
participantsAssignPairsPairsList.setEnabled(false);

6175    participantsAssignPairsPairButton.setEnabled(false);
participantsAssignPairsRemovePairButton.setEnabled(false);
}

6180    // Teachers:
// hadling the event when the user clicks the participantsTeachersAddButton
private void participantsTeachersAddButtonMouseReleased(java.awt.event.MouseEvent
evt) {
    String newTeacherName = participantsTeachersTextField.getText();

6185    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

    try {
        currentSchool.searchTeacher(newTeacherName);      // should result in a
6190    NoSuchElementException exception
        // if not, then:
        illegalActionDialogTextArea.setText("The name you typed in is "
            + "already taken. All teachers must have unique names. If "
            + "you really wish to add a teacher with this name, first "
            + "locate the existing teacher and rename them.");
    }

6195    illegalActionDialog.setVisible(true);
} catch (IllegalArgumentException ex) { // if the name is an empty string (=

6200    OK)
    newTeacherName = "<No name " + currentNewNoNameTeacherNumber + ">";
    currentSchool.insertTeacher(newTeacherName);

    currentNewNoNameTeacherNumber += 1;

6205    teachersNumber += 1;
    participantsTeachersTextField.setText("");
}

6210    participantsTeachersList.setEnabled(true);
participantsTeachersSearchButton.setEnabled(true);

    this.updateTeachersListModel(currentSchool);

6215    int newTeacherIndex = currentSchool.searchTeacher(newTeacherName);
    participantsTeachersList.setSelectedIndex(newTeacherIndex);
    participantsTeachersList.ensureIndexIsVisible(newTeacherIndex);
} catch (NoSuchElementException ex) { // if the name was not encountered (=

6220    GOOD)
    currentSchool.insertTeacher(newTeacherName);
}

```

```

        teachersNumber += 1;

6225    participantsTeachersTextField.setText("");
        participantsTeachersList.setEnabled(true);
        participantsTeachersSearchButton.setEnabled(true);

6230    this.updateTeachersListModel(currentSchool);

        int newTeacherIndex = currentSchool.searchTeacher(newTeacherName);

        participantsTeachersList.setSelectedIndex(newTeacherIndex);
        participantsTeachersList.ensureIndexIsVisible(newTeacherIndex);
6235    } finally { // do in any case
        participantsTeachersTextField.grabFocus();
    }
}

6240 // hadling the event when the user clicks the participantsTeachersRemoveButton
private void
participantsTeachersRemoveButtonMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    Teacher currentTeacher = (Teacher)
6245 currentSchool.getTeacherTreeNodeData(teachersIndex);

    if (teachersIndex >= 0) {
        Teacher deletedTeacher = (Teacher)
currentSchool.deleteTeacher(currentTeacher.getName());
6250
        teachersNumber -= 1;
    }

6255    if (currentSchool.getTeacherTreeSize() != 0) {
        participantsTeachersList.setEnabled(true);
        participantsTeachersSearchButton.setEnabled(true);

        this.updateTeachersListModel(currentSchool);

6260    participantsTeachersList.setSelectedIndex(0);
        participantsTeachersList.ensureIndexIsVisible(0);
    } else {
        participantsTeachersList.setEnabled(false);
        participantsTeachersRemoveButton.setEnabled(false);
        participantsTeachersEditButton.setEnabled(false);

        participantsTeachersList.setModel(emptyModel);
6265
        if (teachersNumber == 0) {
            participantsTeachersSearchButton.setEnabled(false);
        }
    }

6270    participantsTeachersTextField.grabFocus();
6275
    // hadling the event when the user clicks the participantsTeachersEditButton
    private void participantsTeachersEditButtonMouseReleased(java.awt.event.MouseEvent
evt) {
        String newName = participantsTeachersTextField.getText();

        School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

6280        try {
            currentSchool.searchTeacher(newName); // should result in a NoSuchElementException
        exception
            // if not, then:
            illegalActionDialogTextArea.setText("The name you typed in is "
                + "already taken. All teachers must have unique names. If "
                + "you really wish to add a teacher with this name, first "
                + "locate the existing teacher and rename them.");
6285
            illegalActionDialog.setVisible(true);
        } catch (IllegalArgumentException ex) { // if the name is an empty string (=

6290        BAD) // do nothing
        } catch (NoSuchElementException ex) { // if the name was not encountered (=

6295        GOOD)
            currentSchool.editTeacher(teachersIndex, newName);
    }
}

```

```

6300
    studentChanges = true;
    participantsTeachersTextField.setText("");
6305
    this.updateTeachersListModel(currentSchool);
    int editedTeacherIndex = currentSchool.searchTeacher(newName);
6310
    participantsTeachersList.setSelectedIndex(editedTeacherIndex);
    participantsTeachersList.ensureIndexIsVisible(editedTeacherIndex);
    } finally { // do in any case
        participantsTeachersTextField.grabFocus();
    }
6315
}
// hadling the event when the user clicks the participantsTeachersSearchButton
private void
participantsTeachersSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
6320
    String searchedName = participantsTeachersTextField.getText();
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    int searchedTeacherSchoolIndex;
    int searchedTeacherIndex;
6325
    try {
        participantsTeachersTextField.setText("");
6330
        schoolsIndex); // should continue
        searchedTeacherSchoolIndex = searchedTeacherIndices[0]; // the schoolsList
index
6335
        School teacherSchool = (School)
database.getSchoolTreeNodeData(searchedTeacherSchoolIndex);
        this.updateSchoolsListModel();
6340
        participantsSchoolsList.setSelectedIndex(searchedTeacherSchoolIndex);
        participantsSchoolsList.ensureIndexIsVisible(searchedTeacherSchoolIndex);
        searchedTeacherIndex = searchedTeacherIndices[1]; // the teachersList
index
6345
        this.updateTeachersListModel(teacherSchool);
        participantsTeachersList.setSelectedIndex(searchedTeacherIndex);
        participantsTeachersList.ensureIndexIsVisible(searchedTeacherIndex);
6350
    } catch (IllegalArgumentException ex) { // if the name is an empty string (=BAD)
        // do nothing
    } catch (NoSuchElementException ex) { // if the name was not encountered (=BAD)
        participantsTeachersTextField.setText("");
6355
        searchedTeacherIndex = -1;
        this.updateTeachersListModel(currentSchool);
6360
        participantsTeachersList.setSelectedIndex(searchedTeacherIndex);
    } finally { // do in any case
        participantsTeachersTextField.grabFocus();
    }
6365
}
// hadling the event when the user makes a selection in the
participantsTeachersList
private void
6370
participantsTeachersListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    teachersIndex = participantsTeachersList.getSelectedIndex();
6375
    if (teachersIndex != -1) {
        participantsTeachersRemoveButton.setEnabled(true);
        participantsTeachersEditButton.setEnabled(true);
    } else { // if teachersIndex == -1
        participantsTeachersRemoveButton.setEnabled(false);
        participantsTeachersEditButton.setEnabled(false);
    }
}

```

```

6380         }
6381     }
6382     // Students:
6383     // hadling the event when the user clicks the participantsStudentsAddButton
6384     private void participantsStudentsAddButtonMouseReleased(java.awt.event.MouseEvent
6385     evt) {
6386         String newStudentName = participantsStudentsTextField.getText();
6387         School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
6388         try {
6389             currentSchool.searchStudent(newStudentName);      // should result in a
6390             NoSuchElementException exception
6391             // if not, then:
6392             illegalActionDialogTextArea.setText("The name you typed in is "
6393                 + "already taken. All students must have unique names. If "
6394                 + "you really wish to add a student with this name, first "
6395                 + "locate the existing student and rename them.");
6396             illegalActionDialog.setVisible(true);
6397         } catch (IllegalArgumentException ex) { // if the name is an empty string (=OK)
6398             newStudentName = "<No name " + currentNewNoNameStudentNumber + ">";
6399             currentSchool.insertStudent(newStudentName);
6400             currentNewNoNameStudentNumber += 1;
6401             studentsNumber += 1;
6402             studentChanges = true;
6403             participantsStudentsTextField.setText("");
6404             participantsStudentCodesTextField.setText("");
6405             participantsStudentsList.setEnabled(true);
6406             participantsStudentsSearchButton.setEnabled(true);
6407             participantsStudentCodesAssignCodesButton.setEnabled(true);
6408             this.updateStudentsListModel(currentSchool);
6409             this.updateStudentCodesListModel(currentSchool);
6410             int newStudentIndex = currentSchool.searchStudent(newStudentName);
6411             participantsStudentsList.setSelectedIndex(newStudentIndex);
6412             participantsStudentsList.ensureIndexIsVisible(newStudentIndex);
6413             participantsStudentCodesList.setSelectedIndex(newStudentIndex);
6414             participantsStudentCodesList.ensureIndexIsVisible(newStudentIndex);
6415             recheckStudentCodesActionBar();
6416         } catch (NoSuchElementException ex) { // if the name was not encountered (=GOOD)
6417             currentSchool.insertStudent(newStudentName);
6418             studentsNumber += 1;
6419             studentChanges = true;
6420             participantsStudentsTextField.setText("");
6421             participantsStudentCodesTextField.setText("");
6422             participantsStudentsList.setEnabled(true);
6423             participantsStudentsSearchButton.setEnabled(true);
6424             this.updateStudentsListModel(currentSchool);
6425             this.updateStudentCodesListModel(currentSchool);
6426             int newStudentIndex = currentSchool.searchStudent(newStudentName);
6427             participantsStudentsList.setSelectedIndex(newStudentIndex);
6428             participantsStudentsList.ensureIndexIsVisible(newStudentIndex);
6429             participantsStudentCodesList.setSelectedIndex(newStudentIndex);
6430             participantsStudentCodesList.ensureIndexIsVisible(newStudentIndex);
6431             recheckStudentCodesActionBar();
6432         } finally { // do in any case

```

```

        participantsStudentsTextField.grabFocus();
6460    }

    // hadling the event when the user clicks the participantsStudentsRemoveButton
    private void
6465 participantsStudentsRemoveButtonMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

        Student deletedStudent = (Student) currentSchool.deleteStudent(studentsIndex);

6470    studentChanges = true;
    studentsNumber -= 1;

        if (deletedStudent.getDAUnpaired() == true) {
            currentSchool.deleteUnpairedDAStudent(deletedStudent.getName());
        }
        if (deletedStudent.getDebateUnpaired() == true) {
            currentSchool.deleteUnpairedDebateStudent(deletedStudent.getName());
        }

6480    if (deletedStudent.getDAStudentPair() != null) {
        Student deletedPairOtherStudent =
    deletedStudent.getDAStudentPair().getOtherStudent(deletedStudent);

        // GET CORRECT CODE:

6485        /*Student student1;
        Student student2;

        if (deletedStudent.getCode().compareTo(deletedPairOtherStudent.getCode()) <= 0) {
            student1 = deletedStudent; // keep the alphabetical order
            student2 = deletedPairOtherStudent;
        } else {
            student1 = deletedPairOtherStudent; // flip them to be alphabetically
6495    ordered
            student2 = deletedStudent;
        }*/
    }

6500    currentSchool.deleteDAPair(deletedStudent.getDAStudentPair().getOriginalCode());

        currentSchool.insertUnpairedDAStudent(deletedPairOtherStudent);

        deletedPairOtherStudent.setDAUnpaired(true);
    }
    if (deletedStudent.getDebateStudentPair() != null) {
        Student deletedPairOtherStudent =
    deletedStudent.getDebateStudentPair().getOtherStudent(deletedStudent);

6510    // GET CORRECT CODE:

        /*Student student1;
        Student student2;

        if (deletedStudent.getCode().compareTo(deletedPairStudent.getCode()) <= 0)
    {
            student1 = deletedStudent; // keep the alphabetical order
            student2 = deletedPairStudent;
        } else {
            student1 = deletedPairStudent; // flip them to be alphabetically
6520    ordered
            student2 = deletedStudent;
        }*/
    }

6525    currentSchool.deleteDebatePair(deletedStudent.getDAStudentPair().getOriginalCode());

        currentSchool.insertUnpairedDebateStudent(deletedPairOtherStudent);

        deletedPairOtherStudent.setDebateUnpaired(true);
    }

    if (currentSchool.getStudentTreeSize() != 0) { // if the list for this school
6530    is not empty
        participantsStudentsList.setEnabled(true);
        participantsStudentsSearchButton.setEnabled(true);
    }

```

```

this.updateStudentsListModel(currentSchool);
this.updateStudentCodesListModel(currentSchool);

6540 participantsStudentsList.setSelectedIndex(0);
participantsStudentsList.ensureIndexIsVisible(0);

6545 participantsStudentCodesList.setSelectedIndex(0);
participantsStudentCodesList.ensureIndexIsVisible(0);

if (((((currentSchool.getUnpairedDASchoolTreeSize() / 2) > 0) ||
6550 ((currentSchool.getUnpairedDebateStudentTreeSize() / 2) > 0)) && studentCodes) {
    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(true);
} else if ((currentSchool.getDAPairTreeSize() > 0) ||
6555 (currentSchool.getDebatePairTreeSize() > 0)) {
    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(true);
} else {
    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(false);
}

participantsAssignPairsLeftList.setSelectedIndex(-1);
6560 participantsAssignPairsLeftList.setModel(blankModel);
participantsAssignPairsLeftList.setEnabled(false);

participantsAssignPairsRightList.setSelectedIndex(-1);
participantsAssignPairsRightList.setModel(blankModel);
participantsAssignPairsRightList.setEnabled(false);

6565 participantsAssignPairsPairsList.setSelectedIndex(-1);
participantsAssignPairsPairsList.setModel(blankModel);
participantsAssignPairsPairsList.setEnabled(false);

participantsAssignPairsPairButton.setEnabled(false);
participantsAssignPairsRemovePairButton.setEnabled(false);
6570 } else { // if the list for this school is empty
    participantsStudentsList.setEnabled(false);
    participantsStudentsRemoveButton.setEnabled(false);
    participantsStudentsEditButton.setEnabled(false);

    participantsStudentCodesList.setEnabled(false);

6575 participantsStudentsList.setModel(emptyModel);
participantsStudentCodesList.setModel(emptyModel);

participantsStudentsList.setSelectedIndex(-1);
participantsStudentCodesList.setSelectedIndex(-1);

6580 if (studentsNumber != 0) { // if there are still some students
    participantsStudentsSearchButton.setEnabled(true);
    participantsStudentsTextField.setEnabled(true);

    recheckStudentCodesActionBox();
} else { // if there are no students whatsoever
    studentCodes = false;
    studentChanges = false;

    participantsStudentsSearchButton.setEnabled(false);
6585 participantsStudentCodesTextField.setEnabled(false);
    participantsStudentCodesSearchButton.setEnabled(false);
    participantsStudentCodesAssignCodesButton.setEnabled(false);
}

6590 participantsAssignPairsEventsList.setEnabled(false);

participantsAssignPairsEventsList.setSelectedIndex(-1);
participantsAssignPairsEventsList.setModel(eventsListModel);
participantsAssignPairsEventsList.setEnabled(false);

6595 participantsAssignPairsEventsList.setSelectedIndex(-1);
participantsAssignPairsEventsList.setModel(blankModel);
participantsAssignPairsEventsList.setEnabled(false);

participantsAssignPairsLeftList.setSelectedIndex(-1);
participantsAssignPairsLeftList.setModel(blankModel);
participantsAssignPairsLeftList.setEnabled(false);
6600
6605
6610
6615

```

```

        participantsAssignPairsRightList.setSelectedIndex(-1);
        participantsAssignPairsRightList.setModel(blankModel);
        participantsAssignPairsRightList.setEnabled(false);

6620      participantsAssignPairsPairsList.setSelectedIndex(-1);
        participantsAssignPairsPairsList.setModel(blankModel);
        participantsAssignPairsPairsList.setEnabled(false);

6625      participantsAssignPairsPairButton.setEnabled(false);
        participantsAssignPairsRemovePairButton.setEnabled(false);
    }

    recheckStudentCodesActionBar();

6630      participantsStudentsTextField.grabFocus();

        // TODO MAKE PAIRS RESPOND TO THIS
    }

6635      // hadling the event when the user clicks the participantsStudentsEditButton
    private void participantsStudentsEditButtonMouseReleased(java.awt.event.MouseEvent
evt) {
        String newName = participantsStudentsTextField.getText();

6640      School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

        try {
            currentSchool.searchStudent(newName); // should result in a NoSuchElementException
        exception
6645      // if not, then:
        illegalActionDialogTextArea.setText("The name you typed in is "
            + "already taken. All students must have unique names. If "
            + "you really wish to add a student with this name, first "
            + "locate the existing student and rename them.");
6650      illegalActionDialog.setVisible(true);
    } catch (IllegalArgumentException ex) { // if the name is an empty string (=

BAD)
        // do nothing
    } catch (NoSuchElementException ex) { // if the name was not encountered (=

GOOD)
        currentSchool.editStudent(studentsIndex, newName);

        studentChanges = true;
6660      participantsStudentsTextField.setText("");

        this.updateStudentsListModel(currentSchool);
        this.updateStudentCodesListModel(currentSchool);

6665      int editedStudentIndex = currentSchool.searchStudent(newName);

        participantsStudentsList.setSelectedIndex(editedStudentIndex);
        participantsStudentsList.ensureIndexIsVisible(editedStudentIndex);

6670      participantsStudentCodesList.setSelectedIndex(editedStudentIndex);
        participantsStudentCodesList.ensureIndexIsVisible(editedStudentIndex);

        recheckStudentCodesActionBar();
    } finally { // do in any case
        participantsStudentsTextField.grabFocus();
    }
}

6680      // hadling the event when the user clicks the participantsStudentsSearchButton
    private void
participantsStudentsSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
        String searchedName = participantsStudentsTextField.getText();

6685      School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

        int searchedStudentSchoolIndex;
        int searchedStudentIndex;

6690      try {
            participantsStudentsTextField.setText("");

            int[] searchedStudentIndices = database.globalSearchStudent(searchedName,
schoolsIndex); // should continue

```

```

6695             searchedStudentSchoolIndex = searchedStudentIndices[0]; // the schoolsList
index

6700         School studentSchool = (School)
database.getSchoolTreeNodeData(searchedStudentSchoolIndex);

        this.updateSchoolsListModel();

6705         participantsSchoolsList.setSelectedIndex(searchedStudentSchoolIndex);
participantsSchoolsList.ensureIndexIsVisible(searchedStudentSchoolIndex);

        searchedStudentIndex = searchedStudentIndices[1]; // the studentsList
index

6710         this.updateStudentsListModel(studentSchool);
this.updateStudentCodesListModel(studentSchool);

        participantsStudentsList.setSelectedIndex(searchedStudentIndex);
participantsStudentsList.ensureIndexIsVisible(searchedStudentIndex);

6715         participantsStudentCodesList.setSelectedIndex(searchedStudentIndex);
participantsStudentCodesList.ensureIndexIsVisible(searchedStudentIndex);
} catch (IllegalArgumentException ex) { // if the name is an empty string (=

6720     BAD)           // do nothing
} catch (NoSuchElementException ex) { // if the name was not encountered (=

BAD)
        participantsStudentsTextField.setText("");

6725         searchedStudentIndex = -1;

        this.updateStudentsListModel(currentSchool);
this.updateStudentCodesListModel(currentSchool);

6730         participantsStudentsList.setSelectedIndex(searchedStudentIndex);
participantsStudentCodesList.setSelectedIndex(searchedStudentIndex);
} finally { // do in any case
        participantsStudentsTextField.grabFocus();
}
6735     }

        // hadling the event when the user makes a selection in the
participantsStudentsList
        private void
6740 participantsStudentsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
        School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

        studentsIndex = participantsStudentsList.getSelectedIndex();

6745         participantsStudentCodesList.setSelectedIndex(studentsIndex);
participantsStudentCodesList.ensureIndexIsVisible(studentsIndex);

        if (studentsIndex != -1) {
            Student currentStudent = (Student)
6750 currentSchool.getStudentTreeNodeData(studentsIndex);

            participantsStudentsRemoveButton.setEnabled(true);
participantsStudentsEditButton.setEnabled(true);

6755         if (!studentChanges && studentCodes) {
            participantsAssignEventsOriginalOratoryCheckBox.setEnabled(true);

participantsAssignEventsOriginalOratoryCheckBox.setSelected(currentStudent.getEvents()
6760 [0]);
            participantsAssignEventsOralInterpretationCheckBox.setEnabled(true);

participantsAssignEventsOralInterpretationCheckBox.setSelected(currentStudent.getEvent
s()[1]);
6765         participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(true);

participantsAssignEventsImpromptuSpeakingCheckBox.setSelected(currentStudent.getEvents()
() [2]);
6770         if (currentStudent.getDAStudentPair() != null) {
            participantsAssignEventsDuetActingCheckBox.setEnabled(false);

```

```

                                participantsAssignEventsDuetActingCheckBox.setText("Duet Acting
6775    (paired)");
                            }
                            else {
                                participantsAssignEventsDuetActingCheckBox.setEnabled(true);
                                participantsAssignEventsDuetActingCheckBox.setText("Duet Acting");
                            }
6780    participantsAssignEventsDuetActingCheckBox.setSelected(currentStudent.getEvents()[3]);
                            if (currentStudent.getDebateStudentPair() != null) {
                                participantsAssignEventsDebateCheckBox.setEnabled(false);
                                participantsAssignEventsDebateCheckBox.setText("Debate (paired)");
                            }
                            else {
                                participantsAssignEventsDebateCheckBox.setEnabled(true);
                                participantsAssignEventsDebateCheckBox.setText("Debate");
                            }
6790    participantsAssignEventsDebateCheckBox.setSelected(currentStudent.getEvents()[4]);
                            // check-box-item-change events only allow the button if changes were
6795    made
                            if (currentStudent.getReassignmentText() == true)
                                participantsAssignEventsAssignEventsButton.setText("Re-assign events");
6800    else participantsAssignEventsAssignEventsButton.setText("Assign
events");
                            } else {
                                participantsAssignEventsOriginalOratoryCheckBox.setEnabled(false);
                                participantsAssignEventsOriginalOratoryCheckBox.setSelected(false);
6805    participantsAssignEventsOralInterpretationCheckBox.setEnabled(false);
                                participantsAssignEventsOralInterpretationCheckBox.setSelected(false);
                                participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(false);
6810    participantsAssignEventsImpromptuSpeakingCheckBox.setSelected(false);
                                participantsAssignEventsDuetActingCheckBox.setEnabled(false);
                                participantsAssignEventsDuetActingCheckBox.setSelected(false);
6815    participantsAssignEventsDebateCheckBox.setEnabled(false);
                                participantsAssignEventsDebateCheckBox.setSelected(false);
                                participantsAssignEventsAssignEventsButton.setEnabled(false);
6820    participantsAssignEventsAssignEventsButton.setText("Assign events");
                            }
6825    } else { // if studentsIndex == -1
                    participantsStudentsRemoveButton.setEnabled(false);
                    participantsStudentsEditButton.setEnabled(false);
                    participantsAssignEventsOriginalOratoryCheckBox.setEnabled(false);
6830    participantsAssignEventsOriginalOratoryCheckBox.setSelected(false);
                    participantsAssignEventsOralInterpretationCheckBox.setEnabled(false);
                    participantsAssignEventsOralInterpretationCheckBox.setSelected(false);
                    participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(false);
6835    participantsAssignEventsImpromptuSpeakingCheckBox.setSelected(false);
                    participantsAssignEventsDuetActingCheckBox.setEnabled(false);
                    participantsAssignEventsDuetActingCheckBox.setSelected(false);
                    participantsAssignEventsDebateCheckBox.setEnabled(false);
6840    participantsAssignEventsDebateCheckBox.setSelected(false);
                    participantsAssignEventsAssignEventsButton.setEnabled(false);
                    participantsAssignEventsAssignEventsButton.setText("Assign events");
                }
6845    }
    // Dialogs:
    // hadling the event when the user clicks the settingsAcceptDialogConfirmButton
    private void
6850    settingsAcceptDialogConfirmButtonMouseReleased(java.awt.event.MouseEvent evt) {
        settingsAcceptDialog.setVisible(false);
    }
}

```

```

    // hadling the event when the user clicks the settingsErrorDialogConfirmButton
6855 private void settingsErrorDialogConfirmButtonMouseReleased(java.awt.event.MouseEvent evt) {
    settingsErrorDialog.setVisible(false);

    settingsErrorDialogTextArea.setText("");
}

6860 // hadling the event when the user clicks the ExceptionDialogConfirmButton
private void exceptionDialogConfirmButtonMouseReleased(java.awt.event.MouseEvent evt) {
    exceptionDialog.setVisible(false);
}

6865 exceptionDialogTextArea.setText("");
}

// hadling the event when the user clicks the illegalActionDialogConfirmButton
6870 private void illegalActionDialogConfirmButtonMouseReleased(java.awt.event.MouseEvent evt) {
    illegalActionDialog.setVisible(false);

    illegalActionDialogTextArea.setText("");
}

6875 // Student codes:
// hadling the event when the user clicks the
participantsStudentCodesAssignCodesButton
6880 private void participantsStudentCodesAssignCodesButtonMouseReleased(java.awt.event.MouseEvent evt)
{
    try {
        School currentSchool = (School)
6885 database.getSchoolTreeNodeData(schoolsIndex);

        database.assignStudentCodes(studentsNumber);
        database.correctPairCodes(currentSchool);

        if (schoolsIndex != -1) {
            if (currentSchool.getStudentTreeSize() != 0) {
                this.updateStudentCodesListModel(currentSchool);

                participantsStudentCodesList.setSelectedIndex(studentsIndex);
                participantsStudentCodesList.ensureIndexIsVisible(studentsIndex);
            } else {
                participantsStudentCodesList.setModel(emptyModel);
            }
        }
    }

6900 reassignmentText = true;
studentCodes = true;
studentChanges = false;

6905 participantsStudentCodesAssignCodesButton.setText("Re-assign codes");
recheckStudentCodesActionBar();

if (schoolsIndex != -1) {
    if (((currentSchool.getUnpairedDASTudentTreeSize() / 2) > 0) ||
((currentSchool.getUnpairedDebateStudentTreeSize() / 2) > 0)) && studentCodes) {
        participantsAssignPairsEventsList.setSelectedIndex(-1);
        participantsAssignPairsEventsList.setModel(eventsListModel);
        participantsAssignPairsEventsList.setEnabled(true);
    } else if ((currentSchool.getDAPairTreeSize() > 0) ||
(currentSchool.getDebatePairTreeSize() > 0)) {
        participantsAssignPairsEventsList.setSelectedIndex(-1);
        participantsAssignPairsEventsList.setModel(eventsListModel);
        participantsAssignPairsEventsList.setEnabled(true);
    } else {
        participantsAssignPairsEventsList.setSelectedIndex(-1);
        participantsAssignPairsEventsList.setModel(eventsListModel);
        participantsAssignPairsEventsList.setEnabled(false);
    }
}

6925 participantsAssignPairsLeftList.setSelectedIndex(-1);
participantsAssignPairsLeftList.setModel(blankModel);
participantsAssignPairsLeftList.setEnabled(false);
}

6930

```

```

        participantsAssignPairsRightList.setSelectedIndex(-1);
        participantsAssignPairsRightList.setModel(blankModel);
        participantsAssignPairsRightList.setEnabled(false);

6935      participantsAssignPairsPairsList.setSelectedIndex(-1);
        participantsAssignPairsPairsList.setModel(blankModel);
        participantsAssignPairsPairsList.setEnabled(false);

6940      participantsAssignPairsPairButton.setEnabled(false);
        participantsAssignPairsRemovePairButton.setEnabled(false);
    } catch (IllegalStateException ex) {
        // do nothing
    }
}

6945      // handling the event when the user clicks the participantsStudentCodesSearchButton
private void
participantsStudentCodesSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
    String searchedCode = participantsStudentCodesTextField.getText();

6950      School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

    int searchedStudentCodeSchoolIndex;
    int searchedStudentCodeIndex;

6955      try {
        participantsStudentCodesTextField.setText("");

        int[] searchedStudentCodeIndices =
6960      database.searchStudentCode(searchedCode); // should continue

        searchedStudentCodeSchoolIndex = searchedStudentCodeIndices[0]; // the
schoolsList index

6965      School studentCodeSchool = (School)
database.getSchoolTreeNodeData(searchedStudentCodeSchoolIndex);

        this.updateSchoolsListModel();
}

6970      participantsSchoolsList.setSelectedIndex(searchedStudentCodeSchoolIndex);

participantsSchoolsList.ensureIndexIsVisible(searchedStudentCodeSchoolIndex);

6975      searchedStudentCodeIndex = searchedStudentCodeIndices[1]; // the
studentsList index

        if (studentCodeSchool.getStudentTreeSize() != 0) {
            this.updateStudentsListModel(studentCodeSchool);
            this.updateStudentCodesListModel(studentCodeSchool);
}
6980      } else { // if studentTree is empty
            participantsStudentsList.setModel(emptyModel);
            participantsStudentCodesList.setModel(emptyModel);
        }

6985      participantsStudentsList.setSelectedIndex(searchedStudentCodeIndex);
participantsStudentsList.ensureIndexIsVisible(searchedStudentCodeIndex);

        participantsStudentCodesList.setSelectedIndex(searchedStudentCodeIndex);

6990      participantsStudentCodesList.ensureIndexIsVisible(searchedStudentCodeIndex);
        } catch (IllegalArgumentException ex) { // if the name is an empty string (=
BAD)
            // do nothing
}
6995      } catch (NoSuchElementException ex) { // if the name was not encountered (=
BAD)
            participantsStudentsTextField.setText("");
}

participantsStudentsList.setSelectedIndex(-1);
participantsStudentCodesList.setSelectedIndex(-1);

7000      if (currentSchool.getStudentTreeSize() != 0) {
            this.updateStudentsListModel(currentSchool);
            this.updateStudentCodesListModel(currentSchool);
}
7005      } else {
            participantsStudentsList.setModel(emptyModel);
            participantsStudentCodesList.setModel(emptyModel);
        }

} finally { // do in any case
    participantsStudentCodesTextField.grabFocus();
}

```

```

7010         }
    }

    // handling the event when the user makes a selection in the
7015    participantsStudentCodesList
        private void
participantsStudentCodesListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    // as of the moment, the list is always disabled, but this might be changed
upon consideration
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
7020
    studentCodesIndex = participantsStudentCodesList.getSelectedIndex();

    participantsStudentsList.setSelectedIndex(studentCodesIndex);
    participantsStudentsList.ensureIndexIsVisible(studentCodesIndex);
7025
    if (studentCodesIndex != -1) {
        Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(studentsIndex);

7030        participantsStudentsRemoveButton.setEnabled(true);
        participantsStudentsEditButton.setEnabled(true);

        participantsAssignEventsOriginalOratoryCheckBox.setEnabled(true);

7035    participantsAssignEventsOriginalOratoryCheckBox.setSelected(currentStudent.getEvents()
[0]);

        participantsAssignEventsOralInterpretationCheckBox.setEnabled(true);

7040    participantsAssignEventsOralInterpretationCheckBox.setSelected(currentStudent.getEvent
s()[1]);

        /*participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(true);

7045    participantsAssignEventsImpromptuSpeakingCheckBox.setSelected(currentStudent.getEvents()
[2]);

        if (currentStudent.getDASStudentPair() != null) {
            participantsAssignEventsDuetActingCheckBox.setEnabled(false);
            participantsAssignEventsDuetActingCheckBox.setText("Duet Acting
(paired)");
        }
        else {
            participantsAssignEventsDuetActingCheckBox.setEnabled(true);
            participantsAssignEventsDuetActingCheckBox.setText("Duet Acting");
        }

7050    participantsAssignEventsDuetActingCheckBox.setSelected(currentStudent.getEvents()[3]);
7055
        if (currentStudent.getDebateStudentPair() != null) {
            participantsAssignEventsDebateCheckBox.setEnabled(false);
            participantsAssignEventsDebateCheckBox.setText("Debate (paired)");
        }
        else {
            participantsAssignEventsDebateCheckBox.setEnabled(true);
            participantsAssignEventsDebateCheckBox.setText("Debate");
        }

7060    participantsAssignEventsDebateCheckBox.setSelected(currentStudent.getEvents()[4]);*/
7065
        if (currentStudent.getReassignmentText() == true)
participantsAssignEventsAssignEventsButton.setText("Re-assign events");
        else participantsAssignEventsAssignEventsButton.setText("Assign events");
    } else { // if studentCodesIndex == -1
        participantsStudentsRemoveButton.setEnabled(false);
        participantsStudentsEditButton.setEnabled(false);

        participantsAssignEventsOriginalOratoryCheckBox.setEnabled(false);
        participantsAssignEventsOriginalOratoryCheckBox.setSelected(false);

7070        participantsAssignEventsOralInterpretationCheckBox.setEnabled(false);
        participantsAssignEventsOralInterpretationCheckBox.setSelected(false);

        participantsAssignEventsImpromptuSpeakingCheckBox.setEnabled(false);
        participantsAssignEventsImpromptuSpeakingCheckBox.setSelected(false);

        participantsAssignEventsDuetActingCheckBox.setEnabled(false);
        participantsAssignEventsDuetActingCheckBox.setSelected(false);
    }
}

```

```

7090     participantsAssignEventsDebateCheckBox.setEnabled(false);
    participantsAssignEventsDebateCheckBox.setSelected(false);

    participantsAssignEventsAssignEventsButton.setEnabled(false);
7095     participantsAssignEventsAssignEventsButton.setText("Assign events");
}
}

// Assign events:
// hadling the event when the user clicks the
participantsAssignEventsAssignEventsButton
private void
participantsAssignEventsAssignEventsButtonMouseReleased(java.awt.event.MouseEvent evt)
{
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(studentsIndex);

    boolean[] originalEvents = currentStudent.getEvents();

7110    boolean addedDASstudent = false;
    boolean removedDASstudent = false;

    boolean addedDebateStudent = false;
    boolean removedDebateStudent = false;

    boolean oo = participantsAssignEventsOriginalOratoryCheckBox.isSelected();
    boolean oi = participantsAssignEventsOralInterpretationCheckBox.isSelected();
    boolean is = participantsAssignEventsImpromptuSpeakingCheckBox.isSelected();
7120    boolean da = participantsAssignEventsDuetActingCheckBox.isSelected();
    boolean debate = participantsAssignEventsDebateCheckBox.isSelected();

    boolean[] newEvents = {oo, oi, is, da, debate};

7125    currentStudent.setEvents(newEvents);

    participantsAssignEventsAssignEventsButton.setText("Re-assign events");

    currentStudent.setReassignmentText(true);

7130    if (originalEvents[0] != newEvents[0] && oo == true) {
        currentSchool.incrementOOStudentNumber();
    } else if (originalEvents[0] != newEvents[0] && oo == false) {
        currentSchool.decrementOOStudentNumber();
7135    }

    if (originalEvents[1] != newEvents[1] && oi == true) {
        currentSchool.incrementOIStudentNumber();
    } else if (originalEvents[1] != newEvents[1] && oi == false) {
        currentSchool.decrementOIStudentNumber();
7140    }

    if (originalEvents[2] != newEvents[2] && is == true) {
        currentSchool.incrementISStudentNumber();
    } else if (originalEvents[2] != newEvents[2] && is == false) {
        currentSchool.decrementISStudentNumber();
7145    }

    if (originalEvents[3] != newEvents[3] && da == true) {
        if (currentStudent.getDAUnpaired() == false &&
currentStudent.getDASStudentPair() == null) {
            currentSchool.insertUnpairedDASStudent(currentStudent);
            currentStudent.setDAUnpaired(true);
            addedDASstudent = true;
7150    }
    } else if (originalEvents[3] != newEvents[3] && da == false) {
        if (currentStudent.getDAUnpaired() == true) {
            currentSchool.deleteUnpairedDASStudent(currentStudent.getName());
            currentStudent.setDAUnpaired(false);
7155    } else if (currentStudent.getDASStudentPair() != null) {
        Student deletedPairStudent =
currentStudent.getDASStudentPair().getOtherStudent(currentStudent);
        currentSchool.deleteDAPair(currentStudent.getDASStudentPair().getOriginalCode());
        deletedPairStudent.setDASStudentPair(null);
        currentSchool.insertUnpairedDASStudent(deletedPairStudent);
        deletedPairStudent.setDAUnpaired(true);
7160    }
7165}

```

```

    }
    removedDASStudent = true;
}

if (originalEvents[4] != newEvents[4] && debate == true) {
    if (currentStudent.getDebateUnpaired() == false &&
currentStudent.getDebateStudentPair() == null) {
        currentSchool.insertUnpairedDebateStudent(currentStudent);
        currentStudent.setDebateUnpaired(true);
        addedDebateStudent = true;
    }
} else if (originalEvents[4] != newEvents[4] && debate == false) {
    if (currentStudent.getDebateUnpaired() == true) {
        currentSchool.deleteUnpairedDebateStudent(currentStudent.getName());
        currentStudent.setDebateUnpaired(false);
    } else if (currentStudent.getDebateStudentPair() != null) {
        Student deletedPairStudent =
currentStudent.getDebateStudentPair().getOtherStudent(currentStudent);

        currentSchool.deleteDebatePair(currentStudent.getDebateStudentPair().getOriginalCode());
    };
        deletedPairStudent.setDebateStudentPair(null);
        currentSchool.insertUnpairedDebateStudent(deletedPairStudent);
        deletedPairStudent.setDebateUnpaired(true);
    }
    removedDebateStudent = true;
}

if (studentsIndex + 1 < currentSchool.getStudentTreeSize()) {
    int newStudentIndex = studentsIndex + 1;

    this.updateStudentsListModel(currentSchool);
    this.updateStudentCodesListModel(currentSchool);
    participantsStudentsList.setSelectedIndex(newStudentIndex);
    participantsStudentsList.ensureIndexIsVisible(newStudentIndex);
    participantsStudentCodesList.setSelectedIndex(newStudentIndex);
    participantsStudentCodesList.ensureIndexIsVisible(newStudentIndex);
} else {
    int newSchoolIndex = (schoolsIndex + 1 < database.getSchoolTreeSize() ?
(schoolsIndex + 1) : 0);

    this.updateSchoolsListModel();
    participantsSchoolsList.setSelectedIndex(newSchoolIndex);
    participantsSchoolsList.ensureIndexIsVisible(newSchoolIndex);

    currentSchool = (School) database.getSchoolTreeNodeData(newSchoolIndex);

    this.updateTeachersListModel(currentSchool);
    participantsTeachersList.setSelectedIndex(-1);

    int newStudentIndex = 0;

    this.updateStudentsListModel(currentSchool);
    this.updateStudentCodesListModel(currentSchool);
    participantsStudentsList.setSelectedIndex(newStudentIndex);
    participantsStudentsList.ensureIndexIsVisible(newStudentIndex);
    participantsStudentCodesList.setSelectedIndex(newStudentIndex);
    participantsStudentCodesList.ensureIndexIsVisible(newStudentIndex);
}

if (((currentSchool.getUnpairedDASStudentTreeSize() / 2) > 0) ||
((currentSchool.getUnpairedDebateStudentTreeSize() / 2) > 0)) && studentCodes) {
    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(true);
} else if ((currentSchool.getDAPairTreeSize() > 0) ||
(currentSchool.getDebatePairTreeSize() > 0)) {
    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(true);
} else {
    participantsAssignPairsEventsList.setSelectedIndex(-1);
    participantsAssignPairsEventsList.setModel(eventsListModel);
    participantsAssignPairsEventsList.setEnabled(false);

    participantsAssignPairsLeftList.setSelectedIndex(-1);
    participantsAssignPairsLeftList.setModel(blankModel);
    participantsAssignPairsLeftList.setEnabled(false);
}

```

```

participantsAssignPairsRightList.setSelectedIndex(-1);
participantsAssignPairsRightList.setModel(blankModel);
participantsAssignPairsRightList.setEnabled(false);

7250
participantsAssignPairsPairsList.setSelectedIndex(-1);
participantsAssignPairsPairsList.setModel(blankModel);
participantsAssignPairsPairsList.setEnabled(false);

7255
participantsAssignPairsPairButton.setEnabled(false);
participantsAssignPairsRemovePairButton.setEnabled(false);
}

7260
participantsAssignEventsAssignEventsButton.setEnabled(false);

}

// Assign pairs:
// handling the event when the user makes a selection in the
7265
participantsAssignPairsEventsList
    private void
participantsAssignPairsEventsListValueChanged(javax.swing.event.ListSelectionEvent
evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

7270
assignPairsEventsIndex = participantsAssignPairsEventsList.getSelectedIndex();

    if (assignPairsEventsIndex == 0) { // Duet Acting
        if (currentSchool.getUnpairedDASTudentTreeSize() >= 2) { // if there are
at least two unpaired DA students in the school
            // set the contents of left list accordingly
            participantsAssignPairsLeftList.setEnabled(true);
            this.updateAssignPairsLeftListModel(currentSchool);
        } else if (currentSchool.getUnpairedDASTudentTreeSize() == 1) { // if
there is just one unpaired DA student in the school
            // write him down, but not allow him to be selected
            participantsAssignPairsLeftList.setEnabled(false);
            this.updateAssignPairsLeftListModel(currentSchool);
        } else { // if there are no unpaired DA students in the school
            // disable left list and ensure it is erased
            participantsAssignPairsLeftList.setEnabled(false);
            participantsAssignPairsLeftList.setModel(blankModel);
        }
    }

7290
if (currentSchool.getDAPairTreeSize() >= 1) { // if there is at least one
DA pair in the school
    // set the contents of the pairs list to reflect the fact
    participantsAssignPairsPairsList.setEnabled(true);
    this.updateAssignPairsPairsListModel(currentSchool);
} else { // if there are no DA pairs to write down
    // disable the pairs list and erase it
    participantsAssignPairsPairsList.setEnabled(false);
    participantsAssignPairsPairsList.setModel(blankModel);
}

7300
} else if (assignPairsEventsIndex == 1) { // Debate
    if (currentSchool.getUnpairedDebateStudentTreeSize() >= 2) { // if there
are at least two unpaired debate students in the school
        // set the contents of left list accordingly
        participantsAssignPairsLeftList.setEnabled(true);
        this.updateAssignPairsLeftListModel(currentSchool);
    } else if (currentSchool.getUnpairedDebateStudentTreeSize() == 1) { // if
there is just one unpaired debate student in the school
        // write him down, but not allow him to be selected
        participantsAssignPairsLeftList.setEnabled(false);
        this.updateAssignPairsLeftListModel(currentSchool);
    } else { // if there are no unpaired Debate students in the school
        // disable left list and ensure it is erased
        participantsAssignPairsLeftList.setEnabled(false);
        participantsAssignPairsLeftList.setModel(blankModel);
    }
}

7315
if (currentSchool.getDebatePairTreeSize() >= 1) { // if there is at least
one debate pair in the school
    // set the contents of the pairs list to reflect the fact
    participantsAssignPairsPairsList.setEnabled(true);
    this.updateAssignPairsPairsListModel(currentSchool);
} else { // if there are no debate pairs to write down
    // disable the pairs list and erase it
    participantsAssignPairsPairsList.setEnabled(false);
    participantsAssignPairsPairsList.setModel(blankModel);
}

```

```

    } else { // if events list has the value of -1
        // THE PROPAGATION OF THINGS DOES NOT WORK !!! - FIXED?
        participantsAssignPairsLeftList.setSelectedIndex(-1);
        participantsAssignPairsRightList.setSelectedIndex(-1);
        participantsAssignPairsPairButton.setEnabled(false);

        participantsAssignPairsPairsList.setSelectedIndex(-1);
        participantsAssignPairsRemovePairButton.setEnabled(false);
    }
}

// hadling the event when the user makes a selection in the
participantsAssignPairsLeftList
private void
7340 participantsAssignPairsLeftListValueChanged(javax.swing.event.ListSelectionEvent evt)
{
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

    assignPairsLeftIndex = participantsAssignPairsLeftList.getSelectedIndex();

    7345 if (assignPairsLeftIndex != -1) {
        Student[] studentArray;

        if (assignPairsEventsIndex == 0) { // DA
            // set the contents of right list
            participantsAssignPairsRightList.setEnabled(true);
            this.updateAssignPairsRightListModel(currentSchool);
        } else { // DEBATE
            // set the contents of right list
            participantsAssignPairsRightList.setEnabled(true);
            this.updateAssignPairsRightListModel(currentSchool);
        }
    } else { // if leftIndex is -1
        // erase contents of right list
        participantsAssignPairsRightList.setEnabled(false);
        participantsAssignPairsRightList.setModel(blankModel);

        // THE PROPAGATION OF THINGS DOES NOT WORK !!! - FIXED?
        participantsAssignPairsRightList.setSelectedIndex(-1);
        participantsAssignPairsPairButton.setEnabled(false);
    }
}

// hadling the event when the user makes a selection in the
7370 participantsAssignPairsRightList
private void
participantsAssignPairsRightListValueChanged(javax.swing.event.ListSelectionEvent evt)
{
    assignPairsRightIndex = participantsAssignPairsRightList.getSelectedIndex();

    7375 // THE PROPAGATION OF THINGS DOES NOT WORK !!! - FIXED?
    if (assignPairsRightIndex != -1) {
        participantsAssignPairsPairButton.setEnabled(true);
        //participantsAssignPairsPairsList.setSelectedIndex(-1); // => disable
    } else {
        participantsAssignPairsPairButton.setEnabled(false);
    }
}

// hadling the event when the user clicks the participantsAssignPairsPairButton
7385 private void
participantsAssignPairsPairButtonMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

    7390 Student[] unpairedStudentArray;
    StudentPair[] studentPairArray;

    if (assignPairsEventsIndex == 0) { // DA - FIXED
        Student leftStudent = (Student)
        currentSchool.getUnpairedDASStudentTreeNodeData(assignPairsLeftIndex);

        Student rightStudent;
        if (assignPairsLeftIndex > assignPairsRightIndex) { // if left comes after
right (missing piece in right is beyond selection)
            rightStudent = (Student)
        currentSchool.getUnpairedDASStudentTreeNodeData(assignPairsRightIndex);
        } else { // if right comes at or after the missing piece, we must
compensate for the erroneous indexes
    }
}

```

```

7405             rightStudent = (Student)
currentSchool.getUnpairedDAStudentTreeNodeData(assignPairsRightIndex + 1);
}

7410     currentSchool.deleteUnpairedDAStudent(leftStudent.getName()); // remove
the two unpaired students
currentSchool.deleteUnpairedDAStudent(rightStudent.getName());
unpairedStudentArray = currentSchool.getUnpairedDAStudentArray();

7415     DAStudentPair newDAPair = new DAStudentPair(rightStudent, leftStudent);
currentSchool.insertDAPair(newDAPair); // add the newly-made pair
studentPairArray = currentSchool.getDAPairArray();
int newStudentPairIndex = currentSchool.searchDAPair(newDAPair.getCode());
// the index of the new DA pair in the array

7420     // FIXED
if (currentSchool.getUnpairedDAStudentTreeSize() >= 2) { // still more
than two DA unpaired students
    participantsAssignPairsLeftList.setSelectedIndex(-1);
    participantsAssignPairsRightList.setSelectedIndex(-1);

7425     DefaultListModel model;

    // left list update:
model = new DefaultListModel();
for (int i = 0; i < unpairedStudentArray.length; i++) {
    model.addElement(unpairedStudentArray[i].getCode());
}
participantsAssignPairsLeftList.setModel(model);

7430     // right list update:
model = new DefaultListModel();
for (int i = 0; i < unpairedStudentArray.length; i++) {
    if (i == assignPairsLeftIndex) continue;
    else model.addElement(unpairedStudentArray[i].getCode());
}
participantsAssignPairsRightList.setModel(model);

7435     participantsAssignPairsLeftList.setSelectedIndex(0);
participantsAssignPairsRightList.setSelectedIndex(0);

7440     // pairs list enablement and update
participantsAssignPairsPairsList.setEnabled(true);

7445     model = new DefaultListModel();
for (int i = 0; i < studentPairArray.length; i++) {
    model.addElement(studentPairArray[i].getCode());
}
participantsAssignPairsPairsList.setModel(model);

7450     participantsAssignPairsPairsList.setSelectedIndex(newStudentPairIndex);
} else { // less than two unpaired DA students
    participantsAssignPairsLeftList.setSelectedIndex(-1);
    participantsAssignPairsRightList.setSelectedIndex(-1);

7455     DefaultListModel model = new DefaultListModel();

    // left list update:
model = new DefaultListModel();
for (int i = 0; i < unpairedStudentArray.length; i++) {
    model.addElement(unpairedStudentArray[i].getCode());
}
participantsAssignPairsLeftList.setModel(model);

7460     // right list update:
model = new DefaultListModel();
participantsAssignPairsRightList.setModel(model);

7465     // disablement of the pairs list
participantsAssignPairsLeftList.setEnabled(false);
participantsAssignPairsRightList.setEnabled(false);

7470     // pairs list enablement and update
participantsAssignPairsPairsList.setEnabled(true);

7475     model = new DefaultListModel();
for (int i = 0; i < studentPairArray.length; i++) {
    model.addElement(studentPairArray[i].getCode());
}

```

```

7485         }
    participantsAssignPairsPairsList.setModel(model);

    participantsAssignPairsPairsList.setSelectedIndex(newStudentPairIndex);
7490         }
        participantsAssignPairsRemovePairButton.setEnabled(true);
    } else if (assignPairsEventsIndex == 1) { // DEBATE = FIXED
        Student leftStudent = (Student)
currentSchool.getUnpairedDebateStudentTreeNodeData(assignPairsLeftIndex);
7495         Student rightStudent;
        if (assignPairsLeftIndex > assignPairsRightIndex) { // if left comes after
right (missing piece in right is beyond selection)
            rightStudent = (Student)
currentSchool.getUnpairedDebateStudentTreeNodeData(assignPairsRightIndex);
        } else { // if right comes at or after the missing piece, we must
compensate for the erroneous indexes
            rightStudent = (Student)
currentSchool.getUnpairedDebateStudentTreeNodeData(assignPairsRightIndex + 1);
        }

        currentSchool.deleteUnpairedDebateStudent(leftStudent.getName()); // remove the two unpaired students
        currentSchool.deleteUnpairedDebateStudent(rightStudent.getName());
        unpairedStudentArray = currentSchool.getUnpairedDebateStudentArray();

        DebateStudentPair newDebatePair = new DebateStudentPair(rightStudent,
leftStudent);
        currentSchool.insertDebatePair(newDebatePair); // add the newly-made pair
        studentPairArray = currentSchool.getDebatePairArray();
        int newStudentPairIndex =
currentSchool.searchDebatePair(newDebatePair.getCode()); // the index of the new
Debate pair in the array

7520         // FIXED
        if (currentSchool.getUnpairedDebateStudentTreeSize() >= 2) { // still more
than two Debate unpaired students
            participantsAssignPairsLeftList.setSelectedIndex(-1);
            participantsAssignPairsRightList.setSelectedIndex(-1);

7525             DefaultListModel model;

            // left list update:
            model = new DefaultListModel();
            for (int i = 0; i < unpairedStudentArray.length; i++) {
                model.addElement(unpairedStudentArray[i].getCode());
            }
            participantsAssignPairsLeftList.setModel(model);

7535             // right list update:
            model = new DefaultListModel();
            for (int i = 0; i < unpairedStudentArray.length; i++) {
                if (i == assignPairsLeftIndex) continue;
                else model.addElement(unpairedStudentArray[i].getCode());
            }
            participantsAssignPairsRightList.setModel(model);

7540             participantsAssignPairsLeftList.setSelectedIndex(0);
            participantsAssignPairsRightList.setSelectedIndex(0);

7545             // pairs list enablement and update
            participantsAssignPairsPairsList.setEnabled(true);

7550             model = new DefaultListModel();
            for (int i = 0; i < studentPairArray.length; i++) {
                model.addElement(studentPairArray[i].getCode());
            }
            participantsAssignPairsPairsList.setModel(model);

7555             participantsAssignPairsPairsList.setSelectedIndex(newStudentPairIndex);
        } else { // less than two unpaired Debate students
            participantsAssignPairsLeftList.setSelectedIndex(-1);
            participantsAssignPairsRightList.setSelectedIndex(-1);

7560             DefaultListModel model = new DefaultListModel();

```

```

    // left list update:
7565    model = new DefaultListModel();
    for (int i = 0; i < unpairedStudentArray.length; i++) {
        model.addElement(unpairedStudentArray[i].getCode());
    }
    participantsAssignPairsLeftList.setModel(model);

7570    // right list update:
    model = new DefaultListModel();
    participantsAssignPairsRightList.setModel(model);

7575    // disablement of the left and right list
    participantsAssignPairsLeftList.setEnabled(false);
    participantsAssignPairsRightList.setEnabled(false);

    // pairs list enablement and update
7580    participantsAssignPairsPairsList.setEnabled(true);

7585    model = new DefaultListModel();
    for (int i = 0; i < studentPairArray.length; i++) {
        model.addElement(studentPairArray[i].getCode());
    }
    participantsAssignPairsPairsList.setModel(model);

    participantsAssignPairsPairsList.setSelectedIndex(newStudentPairIndex);
7590
}
    participantsAssignPairsRemovePairButton.setEnabled(true);
}

7595    participantsStudentsList.setSelectedIndex(-1);
    participantsStudentsList.setSelectedIndex(-1);

    this.updateStudentsListModel(currentSchool);
    this.updateStudentCodesListModel(currentSchool);
}

7600    // hadling the event when the user makes a selection in the
participantsAssignPairsPairsList
    private void
7605 participantsAssignPairsPairsListValueChanged(javax.swing.event.ListSelectionEvent evt)
{
    assignPairsPairsIndex = participantsAssignPairsPairsList.getSelectedIndex();

    // THE PROPAGATION OF THINGS DOES NOT WORK !!!
    if (assignPairsPairsIndex != -1) {
        participantsAssignPairsRemovePairButton.setEnabled(true);

        // participantsAssignPairsLeftList.setSelectedIndex(-1); // => right(-1)
=> disable pair button
        } else {
            participantsAssignPairsRemovePairButton.setEnabled(false);
        }
    }

    // hadling the event when the user clicks the
7620 participantsAssignPairsRemovePairButton
    private void
participantsAssignPairsRemovePairButtonMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);

7625    if (assignPairsEventsIndex == 0) { // DA - FIXED
        DAStudentPair deletedPair = (DAStudentPair)
currentSchool.deleteDAPair(assignPairsPairsIndex);

        Student[] pairStudentArray = deletedPair.getStudentArray();
7630        Student leftStudent = pairStudentArray[0];
        currentSchool.insertUnpairedDAStudent(leftStudent);

        Student rightStudent = pairStudentArray[1];
        currentSchool.insertUnpairedDAStudent(rightStudent);

        // LIST UPDATES:
        Student[] unpairedStudentArray =
7635        currentSchool.getUnpairedDAStudentArray();
        DAStudentPair[] studentPairArray = currentSchool.getDAPairArray();
}

```

```

DefaultListModel model = new DefaultListModel();

7645 // left list enablement and update:
participantsAssignPairsLeftList.setEnabled(true);

model = new DefaultListModel();
for (int i = 0; i < unpairedStudentArray.length; i++) {
    model.addElement(unpairedStudentArray[i].getCode());
}
participantsAssignPairsLeftList.setModel(model);

7650 int leftStudentIndex =
currentSchool.searchUnpairedDASStudent(leftStudent.getName());
participantsAssignPairsLeftList.setSelectedIndex(leftStudentIndex);

// right list enablement and update:
participantsAssignPairsRightList.setEnabled(true);

7655 model = new DefaultListModel();
for (int i = 0; i < unpairedStudentArray.length; i++) {
    if (i == assignPairsLeftIndex) continue;
    else model.addElement(unpairedStudentArray[i].getCode());
}
participantsAssignPairsRightList.setModel(model);

7660 int rightStudentIndex =
currentSchool.searchUnpairedDASStudent(rightStudent.getName()) - 1; // it is beyond
left, correction needed
participantsAssignPairsRightList.setSelectedIndex(rightStudentIndex);

// pairs list update
model = new DefaultListModel();
for (int i = 0; i < studentPairArray.length; i++) {
    model.addElement(studentPairArray[i].getCode());
}
participantsAssignPairsPairsList.setModel(model);

7665 if (studentPairArray.length >= 1) {
    participantsAssignPairsPairsList.setEnabled(true);
    participantsAssignPairsPairsList.setSelectedIndex(0);
} else {
    participantsAssignPairsPairsList.setEnabled(false);
    participantsAssignPairsPairsList.setSelectedIndex(-1);
}
7670 } else if (assignPairsEventsIndex == 1) { // DEBATE
    DebateStudentPair deletedPair = (DebateStudentPair)
currentSchool.deleteDebatePair(assignPairsPairsIndex);

7675 Student[] pairStudentArray = deletedPair.getStudentArray();

7680 Student leftStudent = pairStudentArray[0];
currentSchool.insertUnpairedDebateStudent(leftStudent);
leftStudent.setDebateUnpaired(true);
leftStudent.setDebateStudentPair(null);

7685 Student rightStudent = pairStudentArray[1];
currentSchool.insertUnpairedDebateStudent(rightStudent);
rightStudent.setDebateUnpaired(true);
rightStudent.setDebateStudentPair(null);

7690 // LIST UPDATES:
7695 Student [] unpairedStudentArray =
currentSchool.getUnpairedDebateStudentArray();
DebateStudentPair [] studentPairArray =
currentSchool.getDebatePairArray();

7700 DefaultListModel model = new DefaultListModel();

7705 // left list enablement and update:
participantsAssignPairsLeftList.setEnabled(true);

7710 model = new DefaultListModel();
for (int i = 0; i < unpairedStudentArray.length; i++) {
    model.addElement(unpairedStudentArray[i].getCode());
}
participantsAssignPairsLeftList.setModel(model);

7715 int leftStudentIndex =
currentSchool.searchUnpairedDebateStudent(leftStudent.getName());

```

```

    participantsAssignPairsLeftList.setSelectedIndex(leftStudentIndex);

    // right list enablement and update:
    participantsAssignPairsRightList.setEnabled(true);

7725    model = new DefaultListModel();
    for (int i = 0; i < unpairedStudentArray.length; i++) {
        if (i == assignPairsLeftIndex) continue;
        else model.addElement(unpairedStudentArray[i].getCode());
    }
    participantsAssignPairsRightList.setModel(model);

7730    int rightStudentIndex =
currentSchool.searchUnpairedDebateStudent(rightStudent.getName()) - 1; // it is beyond
left, correction needed
    participantsAssignPairsRightList.setSelectedIndex(rightStudentIndex);

    // pairs list update
    model = new DefaultListModel();
    for (int i = 0; i < studentPairArray.length; i++) {
        model.addElement(studentPairArray[i].getCode());
    }
    participantsAssignPairsPairsList.setModel(model);

7740    if (studentPairArray.length >= 1) {
        participantsAssignPairsPairsList.setEnabled(true);
        participantsAssignPairsPairsList.setSelectedIndex(0);
    } else {
        participantsAssignPairsPairsList.setEnabled(false);
        participantsAssignPairsPairsList.setSelectedIndex(-1);
    }
}

7745    participantsStudentsList.setSelectedIndex(-1);
participantsStudentsList.setSelectedIndex(-1);

    this.updateStudentsListModel(currentSchool);
    this.updateStudentCodesListModel(currentSchool);
}

7750    // Assign events continued - checkboxes:
    // handling the event when the user clicks the
participantsAssignEventsOriginalOratoryCheckBox
    private void
7755    participantsAssignEventsOriginalOratoryCheckBoxMouseReleased(java.awt.event.MouseEvent
evt) {
        School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
        Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(studentsIndex);

7760    boolean [] originalEvents = new boolean[5];
        for (int i = 0; i < 5; i++) { // change-protected clone of the events array
before changes
            originalEvents[i] = currentStudent.getEvents()[i] ? true : false;
        }

7765    boolean [] newEvents = new boolean[5]; // the events array after the changes
newEvents[0] = participantsAssignEventsOriginalOratoryCheckBox.isSelected();
newEvents[1] =
7770    participantsAssignEventsOralInterpretationCheckBox.isSelected();
        newEvents[2] = participantsAssignEventsImpromptuSpeakingCheckBox.isSelected();
        newEvents[3] = participantsAssignEventsDuetActingCheckBox.isSelected();
        newEvents[4] = participantsAssignEventsDebateCheckBox.isSelected();

7775    boolean changesMade = false;

        for (int i = 0; i < 5; i++) {
            if (newEvents[i] != originalEvents[i]) {
                changesMade = true;
                break;
            }
        }

7780    participantsAssignEventsAssignEventsButton.setEnabled(changesMade); // set
enabled if changes were made
    }

7785    // handling the event when the user clicks the
participantsAssignEventsOralInterpretationCheckBox

```

```

7800     private void
participantsAssignEventsOralInterpretationCheckBoxMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    Student currentStudent = (Student)
7805    currentSchool.getStudentTreeNodeData(studentsIndex);

    boolean [] originalEvents = new boolean[5];
    for (int i = 0; i < 5; i++) { // change-protected clone of the events array
before changes
7810        originalEvents[i] = currentStudent.getEvents()[i] ? true : false;
    }

    boolean [] newEvents = new boolean[5]; // the events array after the changes
7815    newEvents[0] = participantsAssignEventsOriginalOratoryCheckBox.isSelected();
    newEvents[1] =
participantsAssignEventsOralInterpretationCheckBox.isSelected();
    newEvents[2] = participantsAssignEventsImpromptuSpeakingCheckBox.isSelected();
    newEvents[3] = participantsAssignEventsDuetActingCheckBox.isSelected();
    newEvents[4] = participantsAssignEventsDebateCheckBox.isSelected();

7820    boolean changesMade = false;

    for (int i = 0; i < 5; i++) {
        if (newEvents[i] != originalEvents[i]) {
            changesMade = true;
            break;
        }
    }

7830    participantsAssignEventsAssignEventsButton.setEnabled(changesMade); // set
enabled if changes were made
}

// hadling the event when the user clicks the
7835 participantsAssignEventsImpromptuSpeakingCheckBox
    private void
participantsAssignEventsImpromptuSpeakingCheckBoxMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(studentsIndex);

    boolean [] originalEvents = new boolean[5];
    for (int i = 0; i < 5; i++) { // change-protected clone of the events array
7840 before changes
        originalEvents[i] = currentStudent.getEvents()[i] ? true : false;
    }

    boolean [] newEvents = new boolean[5]; // the events array after the changes
7845    newEvents[0] = participantsAssignEventsOriginalOratoryCheckBox.isSelected();
    newEvents[1] =
participantsAssignEventsOralInterpretationCheckBox.isSelected();
    newEvents[2] = participantsAssignEventsImpromptuSpeakingCheckBox.isSelected();
    newEvents[3] = participantsAssignEventsDuetActingCheckBox.isSelected();
    newEvents[4] = participantsAssignEventsDebateCheckBox.isSelected();

    boolean changesMade = false;

    for (int i = 0; i < 5; i++) {
        if (newEvents[i] != originalEvents[i]) {
            changesMade = true;
            break;
        }
    }

7850    participantsAssignEventsAssignEventsButton.setEnabled(changesMade); // set
enabled if changes were made
}

7855 // hadling the event when the user clicks the
participantsAssignEventsDuetActingCheckBox
    private void
participantsAssignEventsDuetActingCheckBoxMouseReleased(java.awt.event.MouseEvent evt)
{
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(studentsIndex);

```

```

7880     boolean [] originalEvents = new boolean[5];
    for (int i = 0; i < 5; i++) { // change-protected clone of the events array
before changes
        originalEvents[i] = currentStudent.getEvents()[i] ? true : false;
    }

7885     boolean [] newEvents = new boolean[5]; // the events array after the changes
newEvents[0] = participantsAssignEventsOriginalOratoryCheckBox.isSelected();
newEvents[1] =
participantsAssignEventsOralInterpretationCheckBox.isSelected();
newEvents[2] = participantsAssignEventsImpromptuSpeakingCheckBox.isSelected();
7890 newEvents[3] = participantsAssignEventsDuetActingCheckBox.isSelected();
newEvents[4] = participantsAssignEventsDebateCheckBox.isSelected();

        boolean changesMade = false;

7895     for (int i = 0; i < 5; i++) {
            if (newEvents[i] != originalEvents[i]) {
                changesMade = true;
                break;
            }
        }

7900     participantsAssignEventsAssignEventsButton.setEnabled(changesMade); // set
enabled if changes were made
    }

7905     // hadling the event when the user clicks the
participantsAssignEventsDebateCheckBox
    private void
participantsAssignEventsDebateCheckBoxMouseReleased(java.awt.event.MouseEvent evt) {
    School currentSchool = (School) database.getSchoolTreeNodeData(schoolsIndex);
    Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(studentsIndex); // THIS IS PRONE TO PROBLEMS

7910     boolean [] originalEvents = new boolean[5];
    for (int i = 0; i < 5; i++) { // change-protected clone of the events array
before changes
        originalEvents[i] = currentStudent.getEvents()[i] ? true : false;
    }

7915     boolean [] newEvents = new boolean[5]; // the events array after the changes
newEvents[0] = participantsAssignEventsOriginalOratoryCheckBox.isSelected();
newEvents[1] =
participantsAssignEventsOralInterpretationCheckBox.isSelected();
newEvents[2] = participantsAssignEventsImpromptuSpeakingCheckBox.isSelected();
7920 newEvents[3] = participantsAssignEventsDuetActingCheckBox.isSelected();
newEvents[4] = participantsAssignEventsDebateCheckBox.isSelected();

        boolean changesMade = false;

7925     for (int i = 0; i < 5; i++) {
            if (newEvents[i] != originalEvents[i]) {
                changesMade = true;
                break;
            }
        }

7930     participantsAssignEventsAssignEventsButton.setEnabled(changesMade); // set
enabled if changes were made
    }

7935     // Technical buttons:
// hadling the event when the user clicks the participantsExportButton
private void participantsExportButtonMouseReleased(java.awt.event.MouseEvent evt)
{
    this.exportParticipantsActionBox();
}

7940     // hadling the event when the user clicks the participantsResetButton
private void participantsResetButtonMouseReleased(java.awt.event.MouseEvent evt) {
resetWarningTextArea.setText("You clicked the Reset button, which "
+ "means that you are about to erase all new content added to "
+ "the database since the latest file-load. (Note that this "
+ "applies globally - contents of all tabs will be affected.) "
+ "If you click confirm, all of this data will be irrevocably "
+ "erased. Click Confirm now only if you are sure this is "
+ "exactly what you want to do. Otherwise, click Cancel.");
}

```

```

        resetWarningDialog.setVisible(true);
7960    }
    // hadling the event when the user clicks the participantsApproveToggleButton
    private void participantsApproveToggleButtonMouseReleased(java.awt.event.MouseEvent evt) {
        if (participantsApproveToggleButton.isSelected()) {
            // SHOW (= in a window) INFORMATION ABOUT THE THINGS THAT DO NOT GO
7965        ACCORDNING TO THE SETTINGS

            // IF THAT IS IGNORED, APPROVE IT AND DESELECT EVERYTHING APART FROM THE
            SAVE BUTTON
7970        String approveText = approveParticipants(database);

            approveParticipantsDialogTextArea.setText(approveText);
            approveParticipantsDialog.setVisible(true);
        } else { // if the approve button is deselected
            this.enableParticipantsTabActionBox(); // QUALIFICATION IS DISABLED AS
7975        WELL
            }
        }

7980    // hadling the event when the user clicks the participantsSaveToggleButton
    private void participantsSaveToggleButtonMouseReleased(java.awt.event.MouseEvent evt) {
        if (participantsSaveToggleButton.isSelected()) {
            try {
7985        approvedSectionsIndex = 1;

                this.saveFile();

                participantsApproveToggleButton.setEnabled(false);
            } catch (UserIOException ex) {
                participantsSaveToggleButton.setSelected(false);
            } catch (IOException ex) {
                participantsSaveToggleButton.setSelected(false);

7990        exceptionDialogTextArea.setText("An input/output exception has "
                + "occurred during the saving of the file. Try to save "
                + "the file once more.");
            }

7995        exceptionDialog.setVisible(true);
        }
    } else {
        participantsApproveToggleButton.setEnabled(true);
    }
8000}

8005    // Technical dialogs:
    // hadling the event when the user clicks the resetWarningDialogConfirmButton
    private void
resetWarningDialogConfirmButtonMouseReleased(java.awt.event.MouseEvent evt) {
8010        resetWarningDialog.setVisible(false);

        resetWarningDialogTextArea.setText("");

        try {
8015        if (loadFile != null) {
            this.readFile(loadFile);
            this.synchronizeTFSettings(database);

            participantsSchoolsList.setSelectedIndex(-1);
            this.updateSchoolsListModel();
        }

8020        participantsSchoolsTextField.setEnabled(true);
        participantsSchoolsAddButton.setEnabled(true);

        if (schoolsNumber > 0) {
            participantsSchoolsList.setEnabled(true);
            participantsSchoolsSearchButton.setEnabled(true);
        }

8025        if (teachersNumber > 0) {
            participantsTeachersTextField.setEnabled(true);
            participantsTeachersSearchButton.setEnabled(true);
        }

8030        if (studentsNumber > 0) {
            participantsStudentsTextField.setEnabled(true);
        }
8035
    }
}

```

```

        participantsStudentsSearchButton.setEnabled(true);
    }

8040    recheckStudentCodesActionBox();

    if (studentCodes) {
        participantsStudentCodesTextField.setEnabled(true);
        participantsStudentCodesSearchButton.setEnabled(true);
    }
    if (studentChanges) {
        participantsStudentCodesAssignCodesButton.setEnabled(true);
    }
} else {
    this.resetDatabaseSettings(database);
    this.synchronizeTFSettings(database);

    database.eraseSchoolTree();

8055    currentNewNoNameSchoolNumber = 1;

    currentNewNoNameTeacherNumber = 1;

8060    currentNewNoNameStudentNumber = 1;

    schoolsNumber = 0;
    teachersNumber = 0;
    studentsNumber = 0;

8065    reassignmentText = false;
    studentCodes = false;
    studentChanges = false;

8070    schoolsIndex = -1;
    teachersIndex = -1;
    studentsIndex = -1;
    studentCodesIndex = -1;

8075    assignPairsEventsIndex = -1;
    assignPairsLeftIndex = -1;
    assignPairsRightIndex = -1;
    assignPairsPairsIndex = -1;

8080    participantsSchoolsList.setSelectedIndex(-1);
    participantsSchoolsList.setEnabled(false);
    participantsSchoolsList.setModel(emptyModel);

    participantsSchoolsRemoveButton.setEnabled(false);
    participantsSchoolsEditButton.setEnabled(false);
    participantsSchoolsSearchButton.setEnabled(false);
}

8085    participantsTeachersTextField.setEnabled(false); // needed - #9

8090    participantsTeachersList.setSelectedIndex(-1);
    participantsTeachersList.setEnabled(false);
    participantsTeachersList.setModel(noSelectionModel);

    participantsTeachersAddButton.setEnabled(false);
    participantsTeachersRemoveButton.setEnabled(false);
    participantsTeachersSearchButton.setEnabled(false);
    participantsTeachersEditButton.setEnabled(false);

8100    participantsStudentsTextField.setEnabled(false); // needed - #10

    participantsStudentsList.setSelectedIndex(-1);
    participantsStudentsList.setEnabled(false);
    participantsStudentsList.setModel(noSelectionModel);

8105    participantsStudentsAddButton.setEnabled(false);
    participantsStudentsRemoveButton.setEnabled(false);
    participantsStudentsSearchButton.setEnabled(false);
    participantsStudentsEditButton.setEnabled(false);

8110    this.disableStudentCodesActionBox();
    participantsStudentCodesList.setModel(noSelectionModel);

    this.disableAssignEventsActionBox();

8115    participantsAssignPairsEventsList.setSelectedIndex(-1);

```

```

participantsAssignPairsEventsList.setEnabled(false);
participantsAssignPairsEventsList.setModel(eventsListModel);

8120 participantsAssignPairsLeftList.setSelectedIndex(-1);
participantsAssignPairsLeftList.setEnabled(false);
participantsAssignPairsLeftList.setModel(blankModel);

participantsAssignPairsRightList.setSelectedIndex(-1);
participantsAssignPairsRightList.setEnabled(false);
participantsAssignPairsRightList.setModel(blankModel);

8125 participantsAssignPairsPairsList.setSelectedIndex(-1);
participantsAssignPairsPairsList.setEnabled(false);
participantsAssignPairsPairsList.setModel(blankModel);

8130 participantsAssignPairsPairButton.setEnabled(false);
participantsAssignPairsRemovePairButton.setEnabled(false);
} catch (FileNotFoundException ex) {
    exceptionDialogTextArea.setText("The previous saved version of the "
        + "database could not be located, possibly because of a "
        + "filename or destination change. The operation could not "
        + "be completed. Try to load the previous file manually "
        + "using the Load button in the Settings tab.");
}

8135 exceptionDialog.setVisible(true);
} catch (IOException ex) {
    exceptionDialogTextArea.setText("An input/output exception has "
        + "occurred during the resetting of the file. Try to reset "
        + "the file once more.");
}

8140 exceptionDialog.setVisible(true);
} catch (IOException ex) {
    exceptionDialogTextArea.setText("An input/output exception has "
        + "occurred during the resetting of the file. Try to reset "
        + "the file once more.");
}

8145 exceptionDialog.setVisible(true);
}

8150 // hadling the event when the user clicks the resetWarningDialogCancelButton
private void resetWarningDialogCancelButtonMouseReleased(java.awt.event.MouseEvent
evt) {
    resetWarningDialog.setVisible(false);
}

8155 resetWarningDialogTextArea.setText("");
}

8160 // hadling the event when the user clicks the
approveParticipantsDialogIgnoreButton
private void
approveParticipantsDialogIgnoreButtonMouseReleased(java.awt.event.MouseEvent evt) {
    long time1 = System.currentTimeMillis();

8165 try {
    qualification.initializeEvents();
    disableParticipantsTabActionBar();
}

8170 approveParticipantsDialog.setVisible(false);
tabbedPane.setSelectedIndex(2);
this.enableQualificationTabActionBar();

8175 this.updateQualificationProgressBar(100);
} catch (Qualification.ImpossibleToAllocateException ex) {
    exceptionDialogTextArea.setText("The allocation of entities has "
        + "failed. Please try again. It is possible, though, that "
        + "there is no possible allocation arrangement for the "
        + "database entries you have provided.");
}

8180 exceptionDialog.setVisible(true);
} catch (Qualification.NotEnoughJudgesException ex) {
    exceptionDialogTextArea.setText("The allocation of entities has "
        + "failed because of the lack of judges for some of the "
        + "events. Please check the approve-participants warning "
        + "message for more information.");
}

8185 exceptionDialog.setVisible(true);
}

8190
long time2 = System.currentTimeMillis();
System.out.println("Total calculation time: " + ((time2 - time1) / 1000) + " seconds");
}

```

```

8195     }
8196     // hadling the event when the user clicks the
8197     approveParticipantsDialogCancelButton
8198     private void
8199     approveParticipantsDialogCancelButtonMouseReleased(java.awt.event.MouseEvent evt) {
8200         approveParticipantsDialog.setVisible(false);
8201         approveParticipantsDialogTextArea.setText("");
8202         participantsApproveToggleButton.setSelected(false);
8203     }
8204
8205     // QUALIFICATION TAB:
8206     // Events:
8207     // hadling the event when the user makes a selection in the
8208     qualificationEventsList
8209     private void
8210     qualificationEventsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
8211         if (!evt.getValueIsAdjusting()) {
8212             eventsIndex = evt.getLastIndex();
8213
8214             roundsIndex = -1;
8215             roomsIndex = -1;
8216             judgesIndex = -1;
8217             qStudentCodesIndex = -1;
8218
8219             try {
8220                 Event currentEvent = qualification.getEventArrayElement(eventsIndex);
8221                 qualificationRoundsList.setEnabled(true);
8222                 this.updateRoundsListModel(currentEvent);
8223             } catch (ArrayIndexOutOfBoundsException ex) {
8224                 qualificationRoundsList.setEnabled(false);
8225                 //qualificationRoundList.setSelectedIndex(-1);
8226                 qualificationRoundsList.setModel(noSelectionModel);
8227
8228             } catch (NullPointerException ex) {
8229                 qualificationRoundsList.setEnabled(false);
8230                 //qualificationRoundList.setSelectedIndex(-1);
8231                 qualificationRoundsList.setModel(emptyModel);
8232             }
8233
8234             qualificationRoundsTextField.setText("");
8235             qualificationRoundsTextField.setEnabled(false);
8236             qualificationRoundsEditNameButton.setEnabled(false);
8237
8238             qualificationRoomsList.setEnabled(false);
8239             //qualificationRoomList.setSelectedIndex(-1);
8240             qualificationRoomsList.setModel(noSelectionModel);
8241
8242             qualificationRoomsTextField.setText("");
8243             qualificationRoomsTextField.setEnabled(false);
8244             qualificationRoomsEditNameButton.setEnabled(false);
8245
8246             qualificationJudgesList.setEnabled(false);
8247             //qualificationJudgesList.setSelectedIndex(-1);
8248             qualificationJudgesList.setModel(noSelectionModel);
8249
8250             qualificationJudgesSubstituteJudgeButton.setEnabled(false);
8251
8252             qualificationStudentCodesList.setEnabled(false);
8253             //qualificationStudentCodesList.setSelectedIndex(-1);
8254             qualificationStudentCodesList.setModel(noSelectionModel);
8255
8256             qualificationStudentCodesTextField.setText("");
8257             qualificationStudentCodesTextField.setEnabled(false);
8258             qualificationStudentCodesSearchButton.setEnabled(false);
8259         }
8260     }
8261
8262     // Rounds:
8263     // hadling the event when the user clicks the qualificationRoundsEditNameButton
8264     private void
8265     qualificationRoundsEditNameButtonMouseReleased(java.awt.event.MouseEvent evt) {
8266         Event currentEvent = qualification.getEventArrayElement(eventsIndex);
8267         Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);

```

```

8275     String newName = qualificationRoundsTextField.getText();
8276     if (!newName.equals("")) {
8277         currentRound.setName(newName);
8278         qualificationRoundsTextField.setText("");
8279         this.updateRoundsListModel(currentEvent);
8280     }
8285     // hadling the event when the user makes a selection in the
8286     qualificationRoundsList
8287     private void
8288     qualificationRoundsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
8289         if (!evt.getValueIsAdjusting()) {
8290             roundsIndex = evt.getLastIndex();
8291
8292             roomsIndex = -1;
8293             judgesIndex = -1;
8294             qStudentCodesIndex = -1;
8295
8296             try {
8297                 Event currentEvent = qualification.getEventArrayElement(eventsIndex);
8298                 Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);
8299
8300                 qualificationRoomsList.setEnabled(true);
8301
8302                 qualificationRoundsTextField.setEnabled(true);
8303                 qualificationRoundsEditNameButton.setEnabled(true);
8304
8305                 this.updateRoomsListModel(currentRound);
8306
8307             } catch (ArrayIndexOutOfBoundsException ex) {
8308                 qualificationRoundsTextField.setText("");
8309                 qualificationRoundsTextField.setEnabled(false);
8310                 qualificationRoundsEditNameButton.setEnabled(false);
8311
8312                 qualificationRoomsList.setEnabled(false);
8313                 //qualificationRoomList.setSelectedIndex(-1);
8314                 qualificationRoomsList.setModel(noSelectionModel);
8315
8316             } catch (NullPointerException ex) {
8317                 qualificationRoundsTextField.setText("");
8318                 qualificationRoundsTextField.setEnabled(false);
8319                 qualificationRoundsEditNameButton.setEnabled(false);
8320
8321                 qualificationRoomsList.setEnabled(false);
8322                 //qualificationRoomList.setSelectedIndex(-1);
8323                 qualificationRoomsList.setModel(emptyModel);
8324             }
8325
8326             qualificationRoomsTextField.setText("");
8327             qualificationRoomsTextField.setEnabled(false);
8328             qualificationRoomsEditNameButton.setEnabled(false);
8329
8330             qualificationJudgesList.setEnabled(false);
8331             //qualificationJudgesList.setSelectedIndex(-1);
8332             qualificationJudgesList.setModel(noSelectionModel);
8333
8334             qualificationJudgesSubstituteJudgeButton.setEnabled(false);
8335
8336             qualificationStudentCodesList.setEnabled(false);
8337             //qualificationStudentCodesList.setSelectedIndex(-1);
8338             qualificationStudentCodesList.setModel(noSelectionModel);
8339
8340             qualificationStudentCodesTextField.setText("");
8341             qualificationStudentCodesTextField.setEnabled(false);
8342             qualificationStudentCodesSearchButton.setEnabled(false);
8343         }
8344     }
8345
8346     // Rooms:
8347     // hadling the event when the user clicks the qualificationRoomsEditNameButton
8348     private void
8349     qualificationRoomsEditNameButtonMouseReleased(java.awt.event.MouseEvent evt) {
8350         Event currentEvent = qualification.getEventArrayElement(eventsIndex);
8351         Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);

```

```

    Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);

8355    String newName = qualificationRoomsTextField.getText();

        if (!newName.equals("")) {
            currentRoom.setName(newName);

8360        qualificationRoomsTextField.setText("");
            this.updateRoomsListModel(currentRound);
        }
    }

8365    // hadling the event when the user makes a selection in the qualificationRoomsList
    private void
qualificationRoomsListValueChanged(javax.swing.event.ListSelectionEvent evt) {
        if (!evt.getValueIsAdjusting()) {
            roomsIndex = evt.getLastIndex();

            judgesIndex = -1;
            qStudentCodesIndex = -1;

8375        try {
            Event currentEvent = qualification.getEventArrayElement(eventsIndex);
            Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);
            Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);

8380        qualificationJudgesList.setEnabled(true);
            qualificationStudentCodesList.setEnabled(true);

            qualificationRoomsTextField.setEnabled(true);
            qualificationRoomsEditNameButton.setEnabled(true);

8385        qualificationStudentCodesTextField.setEnabled(true);
            qualificationStudentCodesSearchButton.setEnabled(true);

            this.updateJudgesListModel(currentRoom);
            this.updateQStudentCodesListModel(currentRoom);

8390        } catch (ArrayIndexOutOfBoundsException ex) {
            qualificationRoomsTextField.setText("");
            qualificationRoomsTextField.setEnabled(false);
            qualificationRoomsEditNameButton.setEnabled(false);

8395        qualificationJudgesList.setEnabled(false);
            //qualificationJudgesList.setSelectedIndex(-1);
            qualificationJudgesList.setModel(noSelectionModel);

8400        qualificationStudentCodesTextField.setText("");
            qualificationStudentCodesList.setEnabled(false);
            //qualificationStudentCodesList.setSelectedIndex(-1);
            qualificationStudentCodesList.setModel(noSelectionModel);

8405        } catch (NullPointerException ex) {
            qualificationRoomsTextField.setText("");
            qualificationRoomsTextField.setEnabled(false);
            qualificationRoomsEditNameButton.setEnabled(false);

8410        qualificationJudgesList.setEnabled(false);
            //qualificationJudgesList.setSelectedIndex(-1);
            qualificationJudgesList.setModel(emptyModel);

8415        qualificationStudentCodesList.setEnabled(false);
            //qualificationStudentCodesList.setSelectedIndex(-1);
            qualificationStudentCodesList.setModel(emptyModel);
        }
    }

8420    qualificationJudgesSubstituteJudgeButton.setEnabled(false);
}
}

// Judges:
8425    // hadling the event when the user clicks the
qualificationJudgesSubstituteJudgeButton
    private void
qualificationJudgesSubstituteJudgeButtonMouseReleased(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
}

```

```

    // hadling the event when the user makes a selection in the
qualificationJudgesList
    private void
8435 qualificationJudgesListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        judgesIndex = evt.getLastIndex();

        Event currentEvent = qualification.getEventArrayElement(eventsIndex);
        Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);
        Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);
        Judge currentJudge = (Judge)
        currentRoom.getJudgeTreeNodeData(judgesIndex);

        if (judgesIndex != -1) {
            //qualificationJudgesSubstituteJudgeButton.setEnabled(true);
        } else {
            qualificationJudgesSubstituteJudgeButton.setEnabled(false);
        }
    }
}

// Student Codes:
// hadling the event when the user clicks the
qualificationStudentCodesSearchButton
    private void
8455 qualificationStudentCodesSearchButtonMouseReleased(java.awt.event.MouseEvent evt) {
    String code = qualificationStudentCodesTextField.getText();

8460    Event currentEvent = qualification.getEventArrayElement(eventsIndex);
    Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);

    try {
        qualificationRoundsTextField.setText("");
8465        int[] searchIndices = currentRound.searchStudentCode(code);
        int searchedRoomIndex = searchIndices[0];
        int searchedEntityIndex = searchIndices[1];

        this.updateRoomsListModel(currentRound);
        qualificationRoomsList.setSelectedIndex(searchedRoomIndex);
        qualificationRoomsList.ensureIndexIsVisible(searchedRoomIndex);

8470        Room searchedRoom = currentRound.getRoomArrayElement(searchedRoomIndex);
        this.updateJudgesListModel(searchedRoom);
        this.updateQStudentCodesListModel(searchedRoom);
        qualificationStudentCodesList.setSelectedIndex(searchedEntityIndex);
        qualificationStudentCodesList.ensureIndexIsVisible(searchedEntityIndex);
    } catch (IllegalArgumentException ex) {
        // empty string - do nothing
    } catch (NoSuchElementException ex) {
        Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);
        qualificationRoundsTextField.setText("");
        qualificationStudentCodesList.setSelectedIndex(-1);
        this.updateQStudentCodesListModel(currentRoom);
    } finally { // do in any case
        qualificationStudentCodesTextField.grabFocus();
    }
8490}

    // hadling the event when the user makes a selection in the
qualificationStudentCodesList
    private void
8495 qualificationStudentCodesListValueChanged(javax.swing.event.ListSelectionEvent evt) {
    if (!evt.getValueIsAdjusting()) {
        qStudentCodesIndex = evt.getLastIndex();
        Event currentEvent = qualification.getEventArrayElement(eventsIndex);
        Round currentRound = currentEvent.getRoundArrayElement(roundsIndex);
        Room currentRoom = currentRound.getRoomArrayElement(roomsIndex);
        Entity currentEntity = (Entity)
        currentRoom.getEntityTreeNodeData(qStudentCodesIndex);
    }
}

// Technical Buttons:
// hadling the event when the user clicks the qualificationExportButton
    private void qualificationExportButtonMouseReleased(java.awt.event.MouseEvent evt)
8505 {
    this.exportQualificationActionBox();
}

```

```

    }

    // hadling the event when the user clicks the qualificationResetButton
    private void qualificationResetButtonMouseReleased(java.awt.event.MouseEvent evt)
8515    {
        // TODO add your handling code here:
        }

        // hadling the event when the user clicks the qualificationSaveToggleButton
        private void qualificationSaveToggleButtonMouseReleased(java.awt.event.MouseEvent evt)
8520    {
        if (qualificationSaveToggleButton.isSelected()) {
            try {
                approvedSectionsIndex = 2;
                this.saveFile();
                // deactivate the whole qualification tab
            } catch (UserIOException ex) {
                qualificationSaveToggleButton.setSelected(false);
            } catch (IOException ex) {
                qualificationSaveToggleButton.setSelected(false);
                exceptionDialogTextArea.setText("An input/output exception has "
                    + "occurred during the saving of the file. Try to save "
                    + "the file once more.");
                exceptionDialog.setVisible(true);
            }
        } else {
            // reactivate the whole qualification tab
        }
    }

8540 /**
 * @param args the command line arguments
 */
8545 public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            new MainGUI().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
8555 private javax.swing.JMenuItem aboutMenuItem;
private static javax.swing.JDialog allocationInfoDialog;
private static javax.swing.JProgressBar allocationInfoProgressBar;
private static javax.swing.JTextField allocationInfoTextField;
private javax.swing.JDialog approveParticipantsDialog;
8560 private javax.swing.JButton approveParticipantsDialogCancelButton;
private javax.swing.JButton approveParticipantsDialogIgnoreButton;
private javax.swing.JScrollPane approveParticipantsDialogScrollPane;
private javax.swing.JTextArea approveParticipantsDialogTextArea;
private javax.swing.JMenuItem contentsMenuItem;
8565 private javax.swing.JMenuItem copyMenuItem;
private javax.swing.JMenuItem cutMenuItem;
private javax.swing.JMenuItem deleteMenuItem;
private javax.swing.JMenu editMenu;
private javax.swing.JTextField eventSettingsDAFinalsSR;
8570 private javax.swing.JTextField eventSettingsDAJR;
private javax.swing.JTextField eventSettingsDASES;
private javax.swing.JTextField eventSettingsDASR;
private javax.swing.JTextField eventSettingsDebateJR;
private javax.swing.JTextField eventSettingsDebateSES;
8575 private javax.swing.JTextField eventSettingsFinalsJR;
private javax.swing.JTextField eventSettingsISFinalsSR;
private javax.swing.JTextField eventSettingsISJR;
private javax.swing.JTextField eventSettingsISSES;
private javax.swing.JTextField eventSettingsISSR;
8580 private javax.swing.JTextField eventSettingsOIFinalsSR;
private javax.swing.JTextField eventSettingsOIJR;
private javax.swing.JTextField eventSettingsOISES;
private javax.swing.JTextField eventSettingsOISR;
private javax.swing.JTextField eventSettingsOOFinalsSR;
8585 private javax.swing.JTextField eventSettingsOOJR;
private javax.swing.JTextField eventSettingsOOSES;
private javax.swing.JTextField eventSettingsOOSR;
private javax.swing.JDialog exceptionDialog;
private javax.swing.JButton exceptionDialogConfirmButton;
private javax.swing.JScrollPane exceptionDialogScrollPane;

```

```

8590    private javax.swing.JTextArea exceptionDialogTextArea;
8591    private javax.swing.JMenuItem exitMenuItem;
8592    private javax.swing.JMenu fileMenu;
8593    private javax.swing.JToggleButton finalistsApproveToggleButton;
8594    private javax.swing.JList finalistsDebateFinalistsList;
8595    private javax.swing.JList finalistsDebateSemifinalistsAList;
8596    private javax.swing.JList finalistsDebateSemifinalistsBList;
8597    private javax.swing.JButton finalistsDebateSemifinalistsSetWinnerAButton;
8598    private javax.swing.JButton finalistsDebateSemifinalistsSetWinnerBButton;
8599    private javax.swing.JList finalistsDuetActingList;
8600    private javax.swing.JButton finalistsExportButton;
8601    private javax.swing.JList finalistsImpromptuSpeakingList;
8602    private javax.swing.JList finalistsOralInterpretationList;
8603    private javax.swing.JList finalistsOriginalOratoryList;
8604    private javax.swing.JPanel finalistsPane;
8605    private javax.swing.JProgressBar finalistsProgressBar;
8606    private javax.swing.JButton finalistsResetButton;
8607    private javax.swing.JToggleButton finalistsSaveToggleButton;
8608    private javax.swing.JToggleButton finalistsApproveToggleButton;
8609    private javax.swing.JList finalsEventList;
8610    private javax.swing.JButton finalsExportButton;
8611    private javax.swing.JList finalsJudgesList;
8612    private javax.swing.JButton finalsJudgesSubstituteJudgeButton;
8613    private javax.swing.JPanel finalsPane;
8614    private javax.swing.JProgressBar finalsProgressBar;
8615    private javax.swing.JButton finalsRankingsAddButton;
8616    private javax.swing.JList finalsRankingsList;
8617    private javax.swing.JButton finalsRankingsRemoveButton;
8618    private javax.swing.JTextField finalsRankingsTextField;
8619    private javax.swing.JButton finalsResetButton;
8620    private javax.swing.JButton finalsRoomEditNameButton;
8621    private javax.swing.JList finalsRoomList;
8622    private javax.swing.JTextField finalsRoomTextField;
8623    private javax.swing.JButton finalsRoundEditNameButton;
8624    private javax.swing.JList finalsRoundList;
8625    private javax.swing.JTextField finalsRoundTextField;
8626    private javax.swing.JToggleButton finalsSaveToggleButton;
8627    private javax.swing.JList finalsStudentCodesList;
8628    private javax.swing.JButton finalsStudentCodesSearchButton;
8629    private javax.swing.JTextField finalsStudentCodesTextField;
8630    private javax.swing.JList finalsStudentsVoteList;
8631    private javax.swing.JButton finalsStudentsVoteSetStudentsVoteButton;
8632    private javax.swing.JCheckBox generalSettingsCECheckBox;
8633    private javax.swing.JComboBox generalSettingsCEComboBox1;
8634    private javax.swing.JComboBox generalSettingsCEComboBox2;
8635    private javax.swing.JTextField generalSettingsMaximalTime1;
8636    private javax.swing.JTextField generalSettingsMaximalTime2;
8637    private javax.swing.JTextField generalSettingsRoomsAvailable1;
8638    private javax.swing.JTextField generalSettingsRoomsAvailable2;
8639    private javax.swing.JTextField generalSettingsSS;
8640    private javax.swing.JTextField generalSettingsSchoolsNumber;
8641    private javax.swing.JTextField generalSettingsTS;
8642    private javax.swing.JMenu helpMenu;
8643    private javax.swing.JDialog illegalActionDialog;
8644    private javax.swing.JButton illegalActionDialogConfirmButton;
8645    private javax.swing.JScrollPane illegalActionDialogScrollPane;
8646    private javax.swing.JTextArea illegalActionDialogTextArea;
8647    private javax.swing.JLabel jLabel1;
8648    private javax.swing.JLabel jLabel10;
8649    private javax.swing.JLabel jLabel11;
8650    private javax.swing.JLabel jLabel12;
8651    private javax.swing.JLabel jLabel13;
8652    private javax.swing.JLabel jLabel14;
8653    private javax.swing.JLabel jLabel15;
8654    private javax.swing.JLabel jLabel16;
8655    private javax.swing.JLabel jLabel17;
8656    private javax.swing.JLabel jLabel18;
8657    private javax.swing.JLabel jLabel19;
8658    private javax.swing.JLabel jLabel2;
8659    private javax.swing.JLabel jLabel20;
8660    private javax.swing.JLabel jLabel21;
8661    private javax.swing.JLabel jLabel22;
8662    private javax.swing.JLabel jLabel23;
8663    private javax.swing.JLabel jLabel24;
8664    private javax.swing.JLabel jLabel25;
8665    private javax.swing.JLabel jLabel26;
8666    private javax.swing.JLabel jLabel27;
8667    private javax.swing.JLabel jLabel28;
8668    private javax.swing.JLabel jLabel29;

```

```
8670     private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel30;
private javax.swing.JLabel jLabel31;
private javax.swing.JLabel jLabel32;
private javax.swing.JLabel jLabel33;
private javax.swing.JLabel jLabel34;
8675     private javax.swing.JLabel jLabel35;
private javax.swing.JLabel jLabel36;
private javax.swing.JLabel jLabel37;
private javax.swing.JLabel jLabel38;
8680     private javax.swing.JLabel jLabel39;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel40;
private javax.swing.JLabel jLabel41;
private javax.swing.JLabel jLabel42;
8685     private javax.swing.JLabel jLabel43;
private javax.swing.JLabel jLabel44;
private javax.swing.JLabel jLabel45;
private javax.swing.JLabel jLabel46;
private javax.swing.JLabel jLabel47;
8690     private javax.swing.JLabel jLabel48;
private javax.swing.JLabel jLabel49;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel50;
private javax.swing.JLabel jLabel51;
8695     private javax.swing.JLabel jLabel52;
private javax.swing.JLabel jLabel53;
private javax.swing.JLabel jLabel54;
private javax.swing.JLabel jLabel55;
private javax.swing.JLabel jLabel56;
8700     private javax.swing.JLabel jLabel57;
private javax.swing.JLabel jLabel58;
private javax.swing.JLabel jLabel59;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel60;
8705     private javax.swing.JLabel jLabel61;
private javax.swing.JLabel jLabel62;
private javax.swing.JLabel jLabel63;
private javax.swing.JLabel jLabel64;
private javax.swing.JLabel jLabel65;
8710     private javax.swing.JLabel jLabel66;
private javax.swing.JLabel jLabel67;
private javax.swing.JLabel jLabel68;
private javax.swing.JLabel jLabel69;
private javax.swing.JLabel jLabel7;
8715     private javax.swing.JLabel jLabel70;
private javax.swing.JLabel jLabel71;
private javax.swing.JLabel jLabel72;
private javax.swing.JLabel jLabel73;
private javax.swing.JLabel jLabel74;
8720     private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane10;
private javax.swing.JScrollPane jScrollPane11;
private javax.swing.JScrollPane jScrollPane12;
private javax.swing.JScrollPane jScrollPane13;
8725     private javax.swing.JScrollPane jScrollPane14;
private javax.swing.JScrollPane jScrollPane15;
private javax.swing.JScrollPane jScrollPane16;
private javax.swing.JScrollPane jScrollPane17;
private javax.swing.JScrollPane jScrollPane18;
8730     private javax.swing.JScrollPane jScrollPane19;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JScrollPane jScrollPane20;
private javax.swing.JScrollPane jScrollPane21;
8735     private javax.swing.JScrollPane jScrollPane22;
private javax.swing.JScrollPane jScrollPane23;
private javax.swing.JScrollPane jScrollPane24;
private javax.swing.JScrollPane jScrollPane26;
private javax.swing.JScrollPane jScrollPane27;
8740     private javax.swing.JScrollPane jScrollPane28;
private javax.swing.JScrollPane jScrollPane29;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JScrollPane jScrollPane30;
private javax.swing.JScrollPane jScrollPane31;
8745     private javax.swing.JScrollPane jScrollPane32;
private javax.swing.JScrollPane jScrollPane33;
private javax.swing.JScrollPane jScrollPane34;
private javax.swing.JScrollPane jScrollPane35;
```

```

8750     private javax.swing.JScrollPane jScrollPane36;
private javax.swing.JScrollPane jScrollPane37;
private javax.swing.JScrollPane jScrollPane38;
private javax.swing.JScrollPane jScrollPane39;
private javax.swing.JScrollPane jScrollPane4;
private javax.swing.JScrollPane jScrollPane40;
private javax.swing.JScrollPane jScrollPane41;
private javax.swing.JScrollPane jScrollPane42;
private javax.swing.JScrollPane jScrollPane43;
private javax.swing.JScrollPane jScrollPane44;
private javax.swing.JScrollPane jScrollPane45;
8760     private javax.swing.JScrollPane jScrollPane46;
private javax.swing.JScrollPane jScrollPane47;
private javax.swing.JScrollPane jScrollPane48;
private javax.swing.JScrollPane jScrollPane49;
private javax.swing.JScrollPane jScrollPane5;
private javax.swing.JScrollPane jScrollPane6;
private javax.swing.JScrollPane jScrollPane7;
private javax.swing.JScrollPane jScrollPane8;
private javax.swing.JScrollPane jScrollPane9;
private javax.swing.JMenuBar menuBar;
8770     private javax.swing.JMenuItem openMenuItem;
private javax.swing.JProgressBar overallProgressBar;
private javax.swing.JToggleButton participantsApproveToggleButton;
private javax.swing.JButton participantsAssignEventsAssignEventsButton;
private javax.swing.JCheckBox participantsAssignEventsDebateCheckBox;
private javax.swing.JCheckBox participantsAssignEventsDuetActingCheckBox;
8775     private javax.swing.JCheckBox participantsAssignEventsImpromptuSpeakingCheckBox;
private javax.swing.JCheckBox participantsAssignEventsOralInterpretationCheckBox;
private javax.swing.JCheckBox participantsAssignEventsOriginalOratoryCheckBox;
private javax.swing.JList participantsAssignPairsEventsList;
private javax.swing.JList participantsAssignPairsLeftList;
private javax.swing.JButton participantsAssignPairsPairButton;
private javax.swing.JList participantsAssignPairsPairsList;
private javax.swing.JButton participantsAssignPairsRemovePairButton;
private javax.swing.JList participantsAssignPairsRightList;
8785     private javax.swing.JButton participantsExportButton;
private javax.swing.JPanel participantsPane;
private javax.swing.JPanel participantsPanel;
private javax.swing.JProgressBar participantsProgressBar;
private javax.swing.JButton participantsResetButton;
private javax.swing.JToggleButton participantsSaveToggleButton;
8790     private javax.swing.JButton participantsSchoolsAddButton;
private javax.swing.JButton participantsSchoolsEditButton;
private javax.swing.JList participantsSchoolsList;
private javax.swing.JButton participantsSchoolsRemoveButton;
private javax.swing.JButton participantsSchoolsSearchButton;
8795     private javax.swing.JTextField participantsSchoolsTextField;
private javax.swing.JButton participantsStudentCodesAssignCodesButton;
private javax.swing.JList participantsStudentCodesList;
private javax.swing.JButton participantsStudentCodesSearchButton;
private javax.swing.JTextField participantsStudentCodesTextField;
8800     private javax.swing.JButton participantsStudentsAddButton;
private javax.swing.JButton participantsStudentsEditButton;
private javax.swing.JList participantsStudentsList;
private javax.swing.JButton participantsStudentsRemoveButton;
private javax.swing.JButton participantsStudentsSearchButton;
private javax.swing.JTextField participantsStudentsTextField;
8805     private javax.swing.JButton participantsTeachersAddButton;
private javax.swing.JButton participantsTeachersEditButton;
private javax.swing.JList participantsTeachersList;
private javax.swing.JButton participantsTeachersRemoveButton;
private javax.swing.JButton participantsTeachersSearchButton;
private javax.swing.JTextField participantsTeachersTextField;
8810     private javax.swing.JMenuItem pasteMenuItem;
private javax.swing.JToggleButton qualificationApproveCompleteToggleButton;
private javax.swing.JToggleButton qualificationApproveIncompleteToggleButton;
8815     private javax.swing.JList qualificationEventsList;
private javax.swing.JButton qualificationExportButton;
private javax.swing.JList qualificationJudgesList;
private javax.swing.JButton qualificationJudgesSubstituteJudgeButton;
8820     private javax.swing.JPanel qualificationPane;
private javax.swing.JProgressBar qualificationProgressBar;
private javax.swing.JButton qualificationResetButton;
private javax.swing.JButton qualificationRoomsEditNameButton;
private javax.swing.JList qualificationRoomsList;
private javax.swing.JTextField qualificationRoomsTextField;
8825     private javax.swing.JButton qualificationRoomsEditNameButton;
private javax.swing.JList qualificationRoomsList;

```

```
    private javax.swing.JTextField qualificationRoundsTextField;
    private javax.swing.JToggleButton qualificationSaveToggleButton;
    private javax.swing.JButton qualificationScoresAddButton;
    private javax.swing.JList qualificationScoresList;
    private javax.swing.JButton qualificationScoresRemoveButton;
    private javax.swing.JTextField qualificationScoresTextField;
    private javax.swing.JList qualificationStudentCodesList;
    private javax.swing.JButton qualificationStudentCodesSearchButton;
    private javax.swing.JTextField qualificationStudentCodesTextField;
    private javax.swing.JDialog resetWarningDialog;
    private javax.swing.JButton resetWarningDialogCancelButton;
    private javax.swing.JButton resetWarningDialogConfirmButton;
    private javax.swing.JScrollPane resetWarningDialogScrollPane;
    private javax.swing.JTextArea resetWarningDialogTextArea;
    private javax.swing.JMenuItem saveAsMenuItem;
    private javax.swing.JMenuItem saveMenuItem;
    private javax.swing.JDialog settingsAcceptDialog;
    private javax.swing.JButton settingsAcceptDialogConfirmButton;
    private javax.swing.JToggleButton settingsApproveToggleButton;
    private javax.swing.JDialog settingsErrorDialog;
    private javax.swing.JButton settingsErrorDialogConfirmButton;
    private javax.swing.JScrollPane settingsErrorDialogScrollPane;
    private javax.swing.JTextArea settingsErrorDialogTextArea;
    private javax.swing.JButton settingsLoadButton;
    private javax.swing.JButton settingsResetButton;
    private javax.swing.JToggleButton settingsSaveToggleButton;
    private javax.swing.JTabbedPane tabbedPane;
    private javax.swing.JList winnersDebateFirstPlaceList;
    private javax.swing.JList winnersDebateSecondPlaceList;
    private javax.swing.JList winnersDebateStudentChoiceList;
    private javax.swing.JList winnersDebateThirdPlaceList;
    private javax.swing.JList winnersDuetActingFirstPlaceList;
    private javax.swing.JList winnersDuetActingSecondPlaceList;
    private javax.swing.JList winnersDuetActingStudentChoiceList;
    private javax.swing.JList winnersDuetActingThirdPlaceList;
    private javax.swing.JButton winnersExportButton;
    private javax.swing.JList winnersImpromptuSpeakingFirstPlaceList;
    private javax.swing.JList winnersImpromptuSpeakingSecondPlaceList;
    private javax.swing.JList winnersImpromptuSpeakingStudentChoiceList;
    private javax.swing.JList winnersImpromptuSpeakingThirdPlaceList;
    private javax.swing.JList winnersOralInterpretationFirstPlaceList;
    private javax.swing.JList winnersOralInterpretationSecondPlaceList;
    private javax.swing.JList winnersOralInterpretationStudentChoiceList;
    private javax.swing.JList winnersOralInterpretationThirdPlaceList;
    private javax.swing.JList winnersOriginalOratoryFirstPlaceList;
    private javax.swing.JList winnersOriginalOratorySecondPlaceList;
    private javax.swing.JList winnersOriginalOratoryStudentChoiceList;
    private javax.swing.JList winnersOriginalOratoryThirdPlaceList;
    private javax.swing.JPanel winnersPane;
    private javax.swing.JToggleButton winnersSaveToggleButton;
    // End of variables declaration
}
```

BinarySearchTree.java

8880

```

    /**
     * NESDA Tournament Manager 2013
     *
8885     * By Zbyněk Stara, 000889-045
     * International School of Prague
     * Czech Republic
     *
     * Computer used:
     * Macbook unibody aluminum late 2008
8890     * Mac OS X 10.6.8
     * 8 GiB of RAM
     *
     * IDE used:
     * NetBeans 6.9.1
8895     */

```

```

package data;

import java.util.*;

/**
 * The BinarySearchTree is a heavily used abstract data type in this program. It
 * features all necessary functions, including insert(), delete(), search(),
 * contains(), size(), isEmpty(), getNodeData(), getDataArray() or
 * balance().
 *
 * @author Zbyněk Stara
 */
public class BinarySearchTree {
    /**
     * The Node class is a private class of the BinarySearchTree that forms the
     * building blocks of the tree. It stores data designated by a keyString
     * descriptor. Also, it stores references to the successive nodes of the
     * tree hierarchy.
     *
     * @author Zbyněk Stara
     */
    private class Node {
        private Object data = null;
        private String keyString = null;

        private Node left = null; // the smaller one
        private Node right = null; // the larger one
    }

    private Node() {
    }

    private Node(Object data, String keyString) {
        this.data = data;
        this.keyString = keyString;
    }

    /**
     * This method directly sets the data and keyString of this node to be
     * the same as those of the node passed as a parameter.
     *
     * @param node the node whose contents will be copied
     *
     * @author Zbyněk Stara
     */
    private void setNodeContents(Node node) {
        this.data = node.data;
        this.keyString = node.keyString;
    }

    /**
     * This method sets the node's data to be the object specified in
     * parameter.
     *
     * @param data a data object to be stored in this node
     *
     * @author Zbyněk Stara
     */
8950    private void setNodeData(Object data) {

```

```

        this.data = data;
    }
}

8960 private Node root = null; // the root node of the tree

private int size = 0; // size of the tree

8965 private Object [] dataArray; // an array of data objects of nodes in the tree
private int dataArrayPositionCounter = 0; // the current element in the data array
to which to write (used by getDataArrayHelper)
private boolean dataArrayChanged = true; // indication whether a new dataArray
will have to be made (changes were done to the tree)

8970 private Node [] nodeArray; // an array of nodes in the tree
private int nodeArrayPositionCounter = 0; // the current element in the node array
to which to write (used by getNodeArrayHelper)

8975 public BinarySearchTree() {
}

8980 /**
 * The insert method inserts given data to the tree.
 *
 * @param data object to be stored in the tree
 * @param keyString string identifier of the data, used for initial
 * placement and reference
 * @throws IllegalArgumentException if the keyString is already used in the
 * tree (duplicates are not allowed)
 *
 * @author Zbyněk Stara
 */
8990 public void insert(Object data, String keyString) throws IllegalArgumentException
{
    if (isEmpty()) root = new Node(data, keyString); // if the tree is empty, set
the root to be a new node with the data provided
    else root = insertHelper(data, keyString, root); // else, call insert helper
(throws IllegalArgumentException if keyString is used)

8995     // if there was no exception, increase the size of the tree and acknowledge
the change in the tree
    size += 1;
    dataArrayChanged = true;
}
9000 }

9005 /**
 * The insertHelper is a recursive method that facilitates the insertion of
 * data to a non-empty tree (it recursively finds an empty spot to add the
 * data to).
 *
 * @param data object to be stored in the tree
 * @param keyString string identifier of the data
 * @param currentNode current node the insertHelper is on
 * @return the current node with the modifications due to the addition
 * @throws IllegalArgumentException if the keyString is already used in the
 * tree (duplicates are not allowed)
 *
 * @author Zbyněk Stara
 */
9010 private Node insertHelper(Object data, String keyString, Node currentNode) throws
IllegalArgumentException {
    if (keyString.compareTo(currentNode.keyString) < 0) { // if the key of the
current node is higher than the one that will be added
        // go to the left
        if (currentNode.left == null) { // if the left node is currently null
            // construct a new node there
            currentNode.left = new Node(data, keyString);
            return currentNode;
        } else { // if there is something to the left
            // iterate for that node
            currentNode.left = insertHelper(data, keyString, currentNode.left);
            return currentNode;
        }
    } else if (keyString.compareTo(currentNode.keyString) > 0) { // if the key of
the current node is lower than the one that will be added
        // go to the right
        if (currentNode.right == null) { // if the right node is currently null
            // construct a new node there
        }
    }
}

```

```

9035         currentNode.right = new Node(data, keyString);
9036         return currentNode;
9037     } else { // if there is something to the right
9038         // iterate for that node
9039         currentNode.right = insertHelper(data, keyString, currentNode.right);
9040         return currentNode;
9041     }
9042     } else { // if key of the current node is the same as the one that is being
9043     added
9044     throw new IllegalArgumentException("The same keyString is already used in
9045     the tree. They need unique values.");
9046     }
9047
9048 /**
9049 * This method checks whether the tree contains a specified key.
9050 *
9051 * @param keyString the string whose presence we want to check for
9052 * @return a boolean value indicating whether the key was found
9053 *
9054 * @author Zbyněk Stara
9055 */
9056 public boolean contains(String keyString) {
9057     // try to find the key, if that fails, return false, else return true
9058     try {
9059         searchHelper(keyString, root);
9060         return true;
9061     } catch (NoSuchElementException ex) {
9062         return false;
9063     }
9064 }
9065
9066 /**
9067 * This method looks for a specified key in the tree and returns the data
9068 * object it is associated with.
9069 *
9070 * @param keyString the key to look for
9071 * @return the data object in the node identified by the key
9072 * @throws NoSuchElementException if the key was not found
9073 *
9074 * @author Zbyněk Stara
9075 */
9076 public Object search(String keyString) throws NoSuchElementException {
9077     // search helper is called
9078     Node searchNode = searchHelper(keyString, root);
9079     return searchNode.data; // throws NoSuchElementException if the data does not
9080     exist
9081 }
9082
9083 /**
9084 * The searchHelper is a recursive method that facilitates the searching of
9085 * keys in the tree (it recursively checks whether the key would be to the
9086 * left or to the right of the current node). If it finds the node with
9087 * the key, it returns it.
9088 *
9089 * @param keyString the string identifier to look for in the tree
9090 * @param currentNode the node the searchHelper is currently on
9091 * @return the node that was searched for
9092 * @throws NoSuchElementException if the key could not be found
9093 *
9094 * @author Zbyněk Stara
9095 */
9096 private Node searchHelper(String keyString, Node currentNode) throws
9097 NoSuchElementException {
9098     if (currentNode == null) { // if the search has led to an empty node
9099         throw new NoSuchElementException("The searched data is not in the tree.");
9100     } else { // if the current node is not an empty node
9101         if (keyString.compareTo(currentNode.keyString) == 0) { // if the current
9102             node is the node that is looked for
9103             return currentNode;
9104         } else if (keyString.compareTo(currentNode.keyString) < 0) { // if the key
9105             of the current node is higher than the one searched for
9106             // iterate for the node to the left
9107             return searchHelper(keyString, currentNode.left);
9108         } else { // if the key of the current node is lower than the one searched
9109             for
9110                 // iterate for the node to the right
9111                 return searchHelper(keyString, currentNode.right);
9112             }
9113     }

```

```

9115     }
9116
9117     /**
9118      * This method deletes a node with a specified key from the tree.
9119      *
9120      * @param keyString the string identifier of the node to be deleted
9121      * @return the data object of the deleted node
9122      * @throws NoSuchElementException if the specified key was not found in the
9123      * tree
9124      *
9125      * @author Zbyněk Stara
9126      */
9127     public Object delete(String keyString) throws NoSuchElementException {
9128         if (isEmpty()) { // if the tree is empty
9129             throw new NoSuchElementException("The data to be deleted is not in the
9130             tree.");
9131         } else if (root.keyString.equals(keyString)) { // if the root will have to be
9132             deleted
9133                 // make a temporary root with the current root as a left node
9134                 Node tempRoot = new Node();
9135                 tempRoot.left = root;
9136
9137                 // then call the delete helper on the tempRoot
9138                 Node deletedNode = deleteHelper(keyString, tempRoot, root);
9139
9140                 // after that, set the root to be the left node of the tempRoot
9141                 root = tempRoot.left;
9142
9143                 // decrement size, acknowledge changes
9144                 size -= 1;
9145                 dataArrayChanged = true;
9146
9147                 // return the data object of the deleted node
9148                 return deletedNode.data;
9149             } else { // if the node to be deleted is any other node than the root
9150                 // call the delete helper (throws NoSuchElementException if the key to be
9151                 deleted is not in the tree
9152                 Node deletedNode = deleteHelper(keyString, null, root);
9153
9154                 // decrement size, acknowledge changes
9155                 size -= 1;
9156                 dataArrayChanged = true;
9157
9158                 // return the data object of the deleted node
9159                 return deletedNode.data;
9160             }
9161         }
9162
9163         /**
9164          * The deleteHelper is a recursive method that facilitates the deletion of
9165          * nodes in the tree (it recursively checks whether the key to be deleted
9166          * would be to the left or the right of the current node). If it finds the
9167          * node, it deletes it, and returns its object data. After the deletion, the
9168          * tree still has a valid organization.
9169          *
9170          * @param keyString the key of the node to delete from the tree
9171          * @param parent the parent node of the current node
9172          * @param currentNode the currently investigated node
9173          * @return the deleted node
9174          * @throws NoSuchElementException if the node to be deleted cannot be found
9175          * in the tree
9176          *
9177          * @author Zbyněk Stara
9178          */
9179     private Node deleteHelper(String keyString, Node parent, Node currentNode) throws
9180     NoSuchElementException {
9181         if (keyString.compareTo(currentNode.keyString) < 0) { // if the key of the
9182         current node is higher than the one searched for
9183             //
9184             if (currentNode.left != null) {
9185                 return deleteHelper(keyString, currentNode, currentNode.left);
9186             } else {
9187                 throw new NoSuchElementException("The data to be deleted is not in the
9188                 tree.");
9189             }
9190         } else if (keyString.compareTo(currentNode.keyString) > 0) { // if the key of
the current node is lower than the one searched for

```

```

    if (currentNode.right != null) {
        return deleteHelper(keyString, currentNode, currentNode.right);
    } else {
        throw new NoSuchElementException("The data to be deleted is not in the
tree.");
    }
}

9200    } else { // key is equal to what was being looked for - correct data found
    Node deletedNode = new Node();
    deletedNode.setNodeContents(currentNode);

    if (currentNode.left != null && currentNode.right != null) {
        currentNode.setNodeContents(minNode(currentNode.right)); // set
currentNode to have data of the minimum node to the right
        deleteHelper(keyString, currentNode, currentNode.right);
    } // delete the smallest value on its original position
}

9210    // one or both of the left/right nodes is missing
    else if (parent.left == currentNode) { // if we are to the left from the
parent
        parent.left = (currentNode.left != null) ? currentNode.left :
currentNode.right;
        // if left is not null, set this to be left
        // else set this to be right
    }
9220    else if (parent.right == currentNode) { // if we are to the right from the
parent
        parent.right = (currentNode.left != null) ? currentNode.left :
currentNode.right;
        // if left is not null, set this to be left
        // else set this to be right
    }
    return deletedNode;
}
9230 }

9235 /**
 * This method returns the smallest node from the current node.
 */
9240 * @param currentNode the node from which to start looking
 * @return the smallest node (leftmost) from the current node; if the left
 * node is null, the current node is returned
 */
9245 * @author Zbyněk Stara
*/
private Node minNode(Node currentNode) {
    if (currentNode.left != null) {
        return minNode(currentNode.left);
    } else { // if the node to the left is null
        return currentNode;
    }
}

9250 /**
 * This method sets the data object of a node at a given index
 */
9255 * @param index int value specifying the index of the node that should be
 * changed
 * @param data object with the data to be stored at that node
 * @throws IllegalArgumentException if illegal value of index is provided
 */
9260 * @author Zbyněk Stara
*/
public void setNodeData(int index, Object data) throws IllegalArgumentException {
    if (index >= size || index < 0) {
        throw new IllegalArgumentException();
    } else {
        Node [] localNodeArray = getNodeArray();
        localNodeArray[index].setNodeData(data);
        dataArrayChanged = true;
    }
}

9270 /**
 * This method checks whether the tree is empty.

```

```

    *
    * @return true if the tree is empty, false if it is not
    */
9275 public boolean isEmpty() {
    return root == null;
}

9280 /**
    * This method returns the number of nodes in the tree.
    *
    * @return an int value with the size of the tree.
    */
9285 public int size() {
    return size;
}

9290 /**
    * This method returns the data object of a node at a specified index.
    *
    * @param index an int value specifying the index of the node whose data is
    * requested
    * @return the data object of the node at the index
    * @throws IllegalArgumentException if the index has an illegal value
    *
    * @author Zbyněk Stara
    */
9295 public Object getNodeData(int index) throws IllegalArgumentException {
9300     if (index >= size || index < 0) {
        throw new IllegalArgumentException();
    } else {
        if (!dataArrayChanged) {
            if (index >= 0 && index < size) {
                return dataArray[index];
            }
            else return null;
        } else {
            getDataArray();
            if (index >= 0 && index < size) {
                return dataArray[index];
            }
            else return null;
        }
    }
}

9315 /**
    * This method returns an array of the data stored in all nodes in the tree,
    * in order according to the nodes' key strings.
    *
    * @return an object array with the data of all nodes in the tree
    *
    * @author Zbyněk Stara
    */
9320 public Object [] getDataArray() {
    dataArray = new Object [size];

    if (!this.isEmpty()) {
        dataArrayPositionCounter = 0;
        getDataArrayHelper(root);
        dataArrayChanged = false;
    }

    return dataArray;
}

9335 /**
    * This is a helper method that facilitates the getting of the data array of
    * the tree. It recursively traverses the tree and adds every node's data
    * object to the data array.
    *
    * @param node the currently explored node
    *
    * @author Zbyněk Stara
    */
9340 private void getDataArrayHelper(Node node) {
    if (node.left != null) getDataArrayHelper(node.left);
    dataArray[dataArrayPositionCounter] = node.data;
    dataArrayPositionCounter += 1;
    if (node.right != null) getDataArrayHelper(node.right);
}

```

```

    }

    /**
     * This method facilitates the getting of a node array of the tree.
     *
     * @return a node array with all nodes of the tree
     *
     * @author Zbyněk Stara
     */
9355 private Node [] getNodeArray() {
    nodeArray = new Node [size];
    nodeArrayPositionCounter = 0;
    getNodeArrayHelper(root);
    return nodeArray;
}

9360 /**
     * This is a helper method that facilitates the getting of the node array of
     * the tree. It recursively traverses the tree and adds every node to the
     * node array
     *
     * @param node the current node
     */
9365 private void getNodeArrayHelper(Node node) {
    if (node.left != null) getNodeArrayHelper(node.left);
    nodeArray[nodeArrayPositionCounter] = new Node(node.data, node.keyString);
    nodeArrayPositionCounter += 1;
    if (node.right != null) getNodeArrayHelper(node.right);
}

9370 /**
     * This method "balances" the tree. That means that the tree is made as
     * compact as possible by adding nodes from the node array in an organized
     * fashion. The root is the node in the middle of the tree, its two children
     * are nodes at the boundary of the first and third quartile, and so on,
     * until there are no more nodes left in the node array.
     * <p>
     * Since this algorithm is not very effective (converting a tree to an array
     * and then back), it should be used scarcely.
     *
     * @return a BinarySearchTree with the new organization of nodes
     *
     * @author Zbyněk Stara
     */
9375 public BinarySearchTree balance() {
    BinarySearchTree newTree = new BinarySearchTree();
    if (!this.isEmpty()) {
        Node[] elementArray = this.getNodeArray();

9380 /**
     * This method "balances" the tree. That means that the tree is made as
     * compact as possible by adding nodes from the node array in an organized
     * fashion. The root is the node in the middle of the tree, its two children
     * are nodes at the boundary of the first and third quartile, and so on,
     * until there are no more nodes left in the node array.
     * <p>
     * Since this algorithm is not very effective (converting a tree to an array
     * and then back), it should be used scarcely.
     *
     * @return a BinarySearchTree with the new organization of nodes
     *
     * @author Zbyněk Stara
     */
9385
9390
9395
9400
9405
9410
9415
9420
9425

```

```
9430                     preElementIndex += indexInterval;
                     continue;
                 }
             } else break;
9435         preElementIndex += indexInterval;
     }
}

int i = 0;
while (remainder > 0) {
    if (elementArray[i] != null) {
        newTree.insert(elementArray[i].data, elementArray[i].keyString);
        elementArray[i] = null;
        remainder -= 1;
9445    }
    i += 2;
}
9450     return newTree;
} else {
    return newTree;
}
9455 }
/**
 * This method is used for debugging. It returns a string that is the size
 * of the tree.
 *
 * @return string with the size of the tree
 */
@Override
public String toString() {
    return size + "";
9465 }
```

Database.java

```

9470 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
 * IDE used:
 * NetBeans 6.9.1
 */
9480
9485 package data;
import java.util.*;
9490 /**
 * The Database class is the basis of the hierarchical data structure of this
 * program. It takes care of the Participants tab – that means that it holds a
 * BinarySearchTree of schools (which then hold trees of teachers, students,
 * unpaired students and pairs), and methods to work upon them. It also holds
 * methods that relate to students or teachers, but whose scope includes all of
 * the database – like the global search of students.
 *
 * @author Zbynda
 */
9495 public class Database {
    private BinarySearchTree schoolTree = new BinarySearchTree();
9500
    // <editor-fold defaultstate="collapsed" desc="Default settings variables
declaration">
        private final int defaultOOStudentRoomLimit = 6;
        private final int defaultOISStudentRoomLimit = 6;
        private final int defaultISStudentRoomLimit = 6;
        private final int defaultDASStudentRoomLimit = 4; // pairs
        private final int defaultOOFinalStudentRoomLimit = 6;
        private final int defaultOIFinalStudentRoomLimit = 6;
        private final int defaultISFinalStudentRoomLimit = 6;
        private final int defaultDAFFinalStudentRoomLimit = 4; // pairs
9510
        private final int defaultOOStudentSchoolLimit = 4;
        private final int defaultOISStudentSchoolLimit = 4;
        private final int defaultISStudentSchoolLimit = 4;
        private final int defaultDASStudentSchoolLimit = 2; // pairs
        private final int defaultDebateStudentSchoolLimit = 2; // pairs
9515
        private final int defaultOOJudgeRoomLimit = 2;
        private final int defaultOISJudgeRoomLimit = 2;
        private final int defaultISJudgeRoomLimit = 2;
        private final int defaultDAJudgeRoomLimit = 2;
        private final int defaultDebateJudgeRoomLimit = 3;
        private final int defaultFinalsJudgeRoomLimit = 5;
9520
        private final int defaultRoomLimit1 = 8;
        private final int defaultRoomLimit2 = 8;
        private final int defaultTimeLimit1 = 8; // hours
        private final int defaultTimeLimit2 = 8; // hours
9525
        private final int defaultSchoolNumber = 12;
        private final int defaultTeacherSchoolNumber = 2;
        private final int defaultStudentSchoolNumber = 12;
9530
        private final boolean defaultAllowCombinedEvents = true;
9535
        private final int defaultCombinedEvent1 = 2;
        private final int defaultCombinedEvent2 = 3; // </editor-fold>
9540
        // <editor-fold defaultstate="collapsed" desc="Settings variables declaration">
        private int ooStudentRoomLimit = defaultOOStudentRoomLimit;
        private int oiStudentRoomLimit = defaultOISStudentRoomLimit;
        private int isStudentRoomLimit = defaultISStudentRoomLimit;
        private int daStudentRoomLimit = defaultDASStudentRoomLimit;

```

```

9545    private int ooFinalStudentRoomLimit = defaultOOFinalStudentRoomLimit;
9546    private int oiFinalStudentRoomLimit = defaultOIFinalStudentRoomLimit;
9547    private int isFinalStudentRoomLimit = defaultISFinalStudentRoomLimit;
9548    private int daFinalStudentRoomLimit = defaultDAFinalStudentRoomLimit;

9550    private int ooStudentSchoolLimit = defaultOOStudentSchoolLimit;
9551    private int oiStudentSchoolLimit = defaultOIStudentSchoolLimit;
9552    private int isStudentSchoolLimit = defaultISStudentSchoolLimit;
9553    private int daStudentSchoolLimit = defaultDASchoolLimit;
9554    private int debateStudentSchoolLimit = defaultDebateStudentSchoolLimit;

9555    private int ooJudgeRoomLimit = defaultOOJudgeRoomLimit;
9556    private int oiJudgeRoomLimit = defaultOIJudgeRoomLimit;
9557    private int isJudgeRoomLimit = defaultISJudgeRoomLimit;
9558    private int daJudgeRoomLimit = defaultDAJudgeRoomLimit;
9559    private int debateJudgeRoomLimit = defaultDebateJudgeRoomLimit;
9560    private int finalsJudgeRoomLimit = defaultFinalsJudgeRoomLimit;

9565    private int roomLimit1 = defaultRoomLimit1;
9566    private int roomLimit2 = defaultRoomLimit2;
9567    private int timeLimit1 = defaultTimeLimit1; // hours
9568    private int timeLimit2 = defaultTimeLimit2; // hours
9569    private int schoolNumber = defaultSchoolNumber;
9570    private int teacherSchoolNumber = defaultTeacherSchoolNumber;
9571    private int studentSchoolNumber = defaultStudentSchoolNumber;

9570    private boolean allowCombinedEvents = defaultAllowCombinedEvents;

9571    private int combinedEvent1 = defaultCombinedEvent1;
9572    private int combinedEvent2 = defaultCombinedEvent2;// </editor-fold>

9575    public Database() {
9576        }
9577
9578    /**
9579     * The setSettings method sets the values of the settings variables.
9580     * <p>
9581     * The values of text field settings all have to be higher than zero; if
9582     * some of them is not, it is recorded. In addition, the two timeLimit
9583     * variables both have to be less than or equal to 24. The two combinedEvent
9584     * combo boxes can only have values of 0 to 3 inclusive, therefore any other
9585     * value is illegal.
9586     *
9587     * @param valueArray an array of objects, storing the values of the settings
9588     * variables
9589     * @throws IllegalArgumentException if the Objects in the array are of wrong
9590     * class (they are mostly Integers, except for #26, which is a Boolean);
9591     * also thrown if the array has incorrect length (other than 29)
9592     *
9593     * @author Zbyněk Stara
9594     * @version 1.0 (Nov-11-1012)
9595     * @since Nov-11-2012
9596     */
9597    public void setSettings(Object [] valueArray) throws IllegalArgumentException {
9598        if (valueArray.length == 29) {
9599            try{
9600                ooStudentRoomLimit = (Integer) valueArray[0];
9601                oiStudentRoomLimit = (Integer) valueArray[1];
9602                isStudentRoomLimit = (Integer) valueArray[2];
9603                daStudentRoomLimit = (Integer) valueArray[3];
9604                ooFinalStudentRoomLimit = (Integer) valueArray[4];
9605                oiFinalStudentRoomLimit = (Integer) valueArray[5];
9606                isFinalStudentRoomLimit = (Integer) valueArray[6];
9607                daFinalStudentRoomLimit = (Integer) valueArray[7];

9610                ooStudentSchoolLimit = (Integer) valueArray[8];
9611                oiStudentSchoolLimit = (Integer) valueArray[9];
9612                isStudentSchoolLimit = (Integer) valueArray[10];
9613                daStudentSchoolLimit = (Integer) valueArray[11];
9614                debateStudentSchoolLimit = (Integer) valueArray[12];

9615                ooJudgeRoomLimit = (Integer) valueArray[13];
9616                oiJudgeRoomLimit = (Integer) valueArray[14];
9617                isJudgeRoomLimit = (Integer) valueArray[15];
9618                daJudgeRoomLimit = (Integer) valueArray[16];
9619                debateJudgeRoomLimit = (Integer) valueArray[17];
9620                finalsJudgeRoomLimit = (Integer) valueArray[18];

```

```

9625                     roomLimit1 = (Integer) valueArray[19];
9626                     roomLimit2 = (Integer) valueArray[20];
9627                     timeLimit1 = (Integer) valueArray[21]; // hours
9628                     timeLimit2 = (Integer) valueArray[22]; // hours
9629                     schoolNumber = (Integer) valueArray[23];
9630                     teacherSchoolNumber = (Integer) valueArray[24];
9631                     studentSchoolNumber = (Integer) valueArray[25];
9632
9633                     allowCombinedEvents = (Boolean) valueArray[26];
9634
9635                     combinedEvent1 = (Integer) valueArray[27];
9636                     combinedEvent2 = (Integer) valueArray[28];
9637                 } catch (ClassCastException ex) {
9638                     throw new IllegalArgumentException();
9639                 }
9640             } else throw new IllegalArgumentException();
9641
9642         /**
9643          * The checkSettings method checks the correctness of the setting values.
9644          * <p>
9645          * The values of text field settings all have to be higher than zero; if
9646          * some of them is not, it is recorded. In addition, the two timeLimit
9647          * variables both have to be less than or equal to 24. The two combinedEvent
9648          * combo boxes can only have values of 0 to 3 inclusive, therefore any other
9649          * value is illegal.
9650          *
9651          * @return array of booleans, showing if the values are legal (true means
9652          * "no problem", false means "this value is illegal")
9653          *
9654          * @author Zbyněk Stara
9655          * @version 1.0 (Nov-11-1012)
9656          * @since Nov-11-2012
9657          */
9658         public boolean [] checkSettings() {
9659             boolean [] checkArray = new boolean[29]; // true means "no problem", false
9660             means "this value is illegal"
9661
9662             checkArray[0] = (ooStudentRoomLimit > 0) ? true : false; // if the variable
9663             has a legal value, return true, else return false
9664             checkArray[1] = (oiStudentRoomLimit > 0) ? true : false;
9665             checkArray[2] = (isStudentRoomLimit > 0) ? true : false;
9666             checkArray[3] = (daStudentRoomLimit > 0) ? true : false;
9667             checkArray[4] = (ooFinalStudentRoomLimit > 0) ? true : false;
9668             checkArray[5] = (oiFinalStudentRoomLimit > 0) ? true : false;
9669             checkArray[6] = (isFinalStudentRoomLimit > 0) ? true : false;
9670             checkArray[7] = (daFinalStudentRoomLimit > 0) ? true : false;
9671
9672             checkArray[8] = (ooStudentSchoolLimit > 0) ? true : false;
9673             checkArray[9] = (oiStudentSchoolLimit > 0) ? true : false;
9674             checkArray[10] = (isStudentschoolLimit > 0) ? true : false;
9675             checkArray[11] = (daStudentSchoolLimit > 0) ? true : false;
9676             checkArray[12] = (debateStudentSchoolLimit > 0) ? true : false;
9677
9678             checkArray[13] = (ooJudgeRoomLimit > 0) ? true : false;
9679             checkArray[14] = (oiJudgeRoomLimit > 0) ? true : false;
9680             checkArray[15] = (isJudgeRoomLimit > 0) ? true : false;
9681             checkArray[16] = (daJudgeRoomLimit > 0) ? true : false;
9682             checkArray[17] = (debateJudgeRoomLimit > 0) ? true : false;
9683             checkArray[18] = (finalsJudgeRoomLimit > 0) ? true : false;
9684
9685             checkArray[19] = (roomLimit1 > 0) ? true : false;
9686             checkArray[20] = (roomLimit2 > 0) ? true : false;
9687             checkArray[21] = (timeLimit1 > 0 && timeLimit1 <= 24) ? true : false;
9688             checkArray[22] = (timeLimit2 > 0 && timeLimit2 <= 24) ? true : false;
9689             checkArray[23] = (schoolNumber > 0) ? true : false;
9690             checkArray[24] = (teacherSchoolNumber > 0) ? true : false;
9691             checkArray[25] = (studentSchoolNumber > 0) ? true : false;
9692
9693             checkArray[26] = true; // this can only be true or false (which are both legal
9694             possibilities, so there cannot be any problems
9695
9696             checkArray[27] = (allowCombinedEvents) ? ((combinedEvent1 >= 0 &&
9697             combinedEvent1 < 4) ? true : false) : true; // only reports a problem if combined
9698             events are allowed
9699             checkArray[28] = (allowCombinedEvents) ? ((combinedEvent2 >= 0 &&
9700             combinedEvent2 < 4) ? true : false) : true;
9701
9702         return checkArray;

```

```

    }

9705 /**
 * The resetSettings() resets all database attributes back to their default
 * values.
 * <p>
 * Along with resetting the database attributes back to default values,
 * this method returns an array of the default values themselves.
 *
9710 * @return array of objects with the default values of the settings
 *
 * @author Zbyněk Stara
 * @version 1.0 (Nov-8-2012)
 * @since Nov-8-2012
 */
public Object [] resetSettings() {
    Object [] settingsArray = new Object [29];

9720     settingsArray[0] = ooStudentRoomLimit = defaultOOStudentRoomLimit;
    settingsArray[1] = oiStudentRoomLimit = defaultOISStudentRoomLimit;
    settingsArray[2] = isStudentRoomLimit = defaultISStudentRoomLimit;
    settingsArray[3] = daStudentRoomLimit = defaultDASStudentRoomLimit;
9725     settingsArray[4] = ooFinalStudentRoomLimit = defaultOOFinalStudentRoomLimit;
    settingsArray[5] = oiFinalStudentRoomLimit = defaultOIFinalStudentRoomLimit;
    settingsArray[6] = isFinalStudentRoomLimit = defaultISFinalStudentRoomLimit;
    settingsArray[7] = daFinalStudentRoomLimit = defaultDAFinalStudentRoomLimit;

9730     settingsArray[8] = ooStudentSchoolLimit = defaultOOStudentSchoolLimit;
    settingsArray[9] = oiStudentSchoolLimit = defaultOISStudentSchoolLimit;
    settingsArray[10] = isStudentSchoolLimit = defaultISStudentSchoolLimit;
    settingsArray[11] = daStudentSchoolLimit = defaultDASStudentSchoolLimit;
    settingsArray[12] = debateStudentSchoolLimit =
9735     defaultDebateStudentSchoolLimit;
    settingsArray[13] = ooJudgeRoomLimit = defaultOOJudgeRoomLimit;
    settingsArray[14] = oiJudgeRoomLimit = defaultOIJudgeRoomLimit;
    settingsArray[15] = isJudgeRoomLimit = defaultISJudgeRoomLimit;
    settingsArray[16] = daJudgeRoomLimit = defaultDAJudgeRoomLimit;
    settingsArray[17] = debateJudgeRoomLimit = defaultDebateJudgeRoomLimit;
    settingsArray[18] = finalsJudgeRoomLimit = defaultFinalsJudgeRoomLimit;

9740     settingsArray[19] = roomLimit1 = defaultRoomLimit1;
    settingsArray[20] = roomLimit2 = defaultRoomLimit2;
    settingsArray[21] = timeLimit1 = defaultTimeLimit1; // hours
    settingsArray[22] = timeLimit2 = defaultTimeLimit2; // hours
    settingsArray[23] = schoolNumber = defaultSchoolNumber;
    settingsArray[24] = teacherSchoolNumber = defaultTeacherSchoolNumber;
9745     settingsArray[25] = studentSchoolNumber = defaultStudentSchoolNumber;

9750     settingsArray[26] = allowCombinedEvents = defaultAllowCombinedEvents;

    settingsArray[27] = combinedEvent1 = defaultCombinedEvent1;
    settingsArray[28] = combinedEvent2 = defaultCombinedEvent2;
9755
        return settingsArray;
    }

9760 /**
 * The getSettings() returns the values of the settings attributes.
 *
 * @return array of objects with the values of the settings
 *
 * @author Zbyněk Stara
 * @version 1.0 (Nov-14-2012)
 * @since Nov-14-2012
 */
public Object [] getSettings() {
    Object [] settingsArray = new Object [29];
9770
    settingsArray[0] = ooStudentRoomLimit;
    settingsArray[1] = oiStudentRoomLimit;
    settingsArray[2] = isStudentRoomLimit;
    settingsArray[3] = daStudentRoomLimit;
9775     settingsArray[4] = ooFinalStudentRoomLimit;
    settingsArray[5] = oiFinalStudentRoomLimit;
    settingsArray[6] = isFinalStudentRoomLimit;
    settingsArray[7] = daFinalStudentRoomLimit;

9780     settingsArray[8] = ooStudentSchoolLimit;

```

```

        settingsArray[9] = oiStudentSchoolLimit;
        settingsArray[10] = isStudentSchoolLimit;
        settingsArray[11] = daStudentSchoolLimit;
        settingsArray[12] = debateStudentSchoolLimit;
9785      settingsArray[13] = ooJudgeRoomLimit;
        settingsArray[14] = oiJudgeRoomLimit;
        settingsArray[15] = isJudgeRoomLimit;
9790      settingsArray[16] = daJudgeRoomLimit;
        settingsArray[17] = debateJudgeRoomLimit;
        settingsArray[18] = finalsJudgeRoomLimit;

        settingsArray[19] = roomLimit1;
        settingsArray[20] = roomLimit2;
        settingsArray[21] = timeLimit1; // hours
        settingsArray[22] = timeLimit2; // hours
        settingsArray[23] = schoolNumber;
        settingsArray[24] = teacherSchoolNumber;
9800      settingsArray[25] = studentSchoolNumber;

        settingsArray[26] = allowCombinedEvents;

        settingsArray[27] = combinedEvent1;
        settingsArray[28] = combinedEvent2;
9805      return settingsArray;
    }

    /**
     * This method sets the school tree of the database to be a new
     * BinarySearchTree. That means that the original contents of the tree are
     * erased.
     *
     * @author Zbyněk Stara
     */
9815      public void eraseSchoolTree() {
        schoolTree = new BinarySearchTree();
    }

9820      /**
     * The getSchoolArray returns the schoolArray in a sorted fashion.
     * <p>
     * The schoolTree is traversed in order and returned as an array of schools.
     *
     * @return array of schools reflecting the schoolTree
     *
     * @author Zbyněk Stara
     * @version 1.1 (Nov-6-1012)
     * @since Nov-6-2012
     */
9830      public School [] getSchoolArray() {
        Object [] objectArray = schoolTree.getDataArray();
        School [] schoolArray = new School [objectArray.length];

9835      for (int i = 0; i < objectArray.length; i++) {
            schoolArray[i] = (School) objectArray[i];
        }

        return schoolArray;
    }

    /**
     * This method returns the school from the database that has a given name.
     *
     * @param name the name of the school to search for
     * @return the school with the corresponding name
     * @throws IllegalArgumentException if the name given to this method is an
     * empty string
     * @throws NoSuchElementException if the name could not be found in the
     * database
     *
     * @author Zbyněk Stara
     */
9845      public School getSchool(String name) throws IllegalArgumentException,
9850      NoSuchElementException {
        if (!name.equals("")) {
            int schoolIndex = this.searchSchool(name);
            School school = (School) this.getSchoolTreeNodeData(schoolIndex);
            return school;
        }
    }

```

```

9860         } else throw new IllegalArgumentException();
9861     }
9862
9863     /**
9864      * The getSchoolTreeNodeData method returns the data for a given node of
9865      * the schoolTree.
9866      * <p>
9867      * A node of the schoolTree, identified by the index parameter, is examined
9868      * and the content of its data attribute is returned as object.
9869      *
9870      * @param index the node which will be examined
9871      * @return the data this node carries
9872      *
9873      * @author Zbyněk Stara
9874      * @version 1.0 (Jan-3-2013)
9875      * @since Jan-3-2013
9876      */
9877     public Object getSchoolTreeNodeData(int index) {
9878         return schoolTree.getNodeData(index);
9879     }
9880
9881     /**
9882      * The getSchoolTreeSize returns the number of schools in the schoolTree
9883      *
9884      * @return the size of the schoolTree, as an integer value
9885      *
9886      * @author Zbyněk Stara
9887      * @version 1.0 (Jan-3-2013)
9888      * @since Jan-3-2013
9889      */
9890     public int getSchoolTreeSize() {
9891         return schoolTree.size();
9892     }
9893
9894     /**
9895      * This insertSchool method adds a new school to the schoolTree.
9896      * <p>
9897      * The school is added with the name specified in the parameter.
9898      *
9899      * @param name the name of the school to be added
9900      * @throws IllegalArgumentException if there is already a school of the
9901      * specified name or the name is an empty string
9902      *
9903      * @author Zbyněk Stara
9904      * @version 1.6 (Jan-3-2012)
9905      * @since Nov-6-2012
9906      */
9907     public void insertSchool(String name) throws IllegalArgumentException {
9908         if (!name.equals("")) {
9909             School newSchool = new School(name);
9910             schoolTree.insert(newSchool, name);
9911         }
9912         else throw new IllegalArgumentException();
9913     }
9914
9915     /**
9916      * This insertSchool method adds a specified school to the schoolTree.
9917      *
9918      * @param school the school to be added
9919      * @throws IllegalArgumentException if there is already a school of the
9920      * specified name or the name is an empty string
9921      *
9922      * @author Zbyněk Stara
9923      * @version 1.0 (Jan-3-2012)
9924      * @since Jan-3-2012
9925      */
9926     public void insertSchool(School school) throws IllegalArgumentException {
9927         if (!school.getName().equals("")) {
9928             schoolTree.insert(school, school.getName());
9929         }
9930         else throw new IllegalArgumentException();
9931     }
9932
9933     /**
9934      * The deleteSchool method removes a specific school from the schoolTree.
9935      * <p>
9936      * The school of the specified name is deleted, provided that it exists. It
9937      * is then returned by the function.
9938      *

```

```

9940      * @param name the name of the school to be deleted
9941      * @return the deleted school
9942      * @throws NoSuchElementException if there is no school of the specified
9943      * name
9944      *
9945      * @author Zbyněk Stara
9946      * @version 1.4 (Jan-3-2013)
9947      * @since Nov-6-2012
9948      */
9949      public School deleteSchool(String name) throws NoSuchElementException {
9950          return (School) schoolTree.delete(name);
9951      }
9952
9953      /**
9954      * The deleteSchool method removes a specific school from the schoolTree.
9955      * <p>
9956      * The school at the specified index is removed from the schoolTree and is
9957      * returned by this function
9958      *
9959      * @param index the index of the school in the shoolTree
9960      * @return the deleted school
9961      * @throws IllegalArgumentException if the index is not within the bounds of
9962      * the schoolTree
9963      *
9964      * @author Zbyněk Stara
9965      * @version 1.2 (Jan-3-2013)
9966      * @since Nov-6-2012
9967      */
9968      public School deleteSchool(int index) throws IllegalArgumentException {
9969          if (index >= 0 && index < schoolTree.size()) {
9970              School deletedSchool = (School) schoolTree.getNodeData(index);
9971              String deleteKey = deletedSchool.getName();
9972              schoolTree.delete(deleteKey);
9973              return deletedSchool;
9974          }
9975          else throw new IllegalArgumentException();
9976      }
9977
9978      /**
9979      * The editSchool renames a specific school.
9980      * <p>
9981      * The school is first removed from the schoolTree, renamed, and then
9982      * re-inserted under a new name.
9983      *
9984      * @param index the school's index in the shoolTree
9985      * @param name new name for the school
9986      * @throws IllegalArgumentException if the name is already used in the
9987      * schoolTree or the new name is an empty string or the index is not within
9988      * the bounds of the schoolTree
9989      *
9990      * @author Zbyněk Stara
9991      * @version 1.1 (Nov-6-2012)
9992      * @since Nov-6-2012
9993      */
9994      public void editSchool(int index, String name) throws IllegalArgumentException {
9995          if (!schoolTree.contains(name) && !name.equals("") && index >= 0 && index <
schoolTree.size()) {
9996              School editedSchool = (School) schoolTree.getNodeData(index);
9997              String editKey = editedSchool.getName();
9998              schoolTree.delete(editKey);
9999              editedSchool.setName(name);
10000             schoolTree.insert(editedSchool, name);
10001         }
10002         else throw new IllegalArgumentException();
10003     }
10004
10005     /**
10006     * The searchSchool finds a specific school.
10007     * <p>
10008     * If the school exists, this method finds it and returns its index in the
10009     * schoolTree.
10010     *
10011     * @param name name of school to search for
10012     * @return school's index in the schoolTree
10013     * @throws IllegalArgumentException if the name is an empty string
10014     * @throws NoSuchElementException if the name has not been found
10015     *
10016     * @author Zbyněk Stara
10017     * @version 1.1 (Nov-6-2012)
10018 
```

```

    * @since Nov-6-2012
    */
10020 public int searchSchool(String name) throws IllegalArgumentException,
NoSuchElementException {
    if (!name.equals("")) {
        Object [] schoolTreeArray = schoolTree.getDataArray();
        boolean schoolFound = false;
        int schoolIndex = -999;
        for (int i = 0; i < schoolTreeArray.length; i++) {
            if (((School) schoolTreeArray[i]).getName().equals(name)) {
                schoolFound = true;
                schoolIndex = i;
                break;
            }
        }
        if (schoolFound) {
            return schoolIndex;
        }
        else throw new NoSuchElementException();
    }
    else throw new IllegalArgumentException();
}

10040 /**
     * The globalSearchTeacher finds a specific teacher.
     * <p>
     * If the teacher exists, this method finds him and returns an array of
     * indices to show where he is. The first element refers to the schoolTree
     * and the second element is the position in the teacherTree of that school
     * where the teacher is.
     * <p>
     * This method starts from the first entry in the schoolTree and goes
     * sequentially forward.
     *
     * @param name name of teacher to search for
     * @return array of indices of the found teacher (schoolTree, teacherTree)
     * @throws IllegalArgumentException if the name is an empty string
     * @throws NoSuchElementException if the name has not been found
     *
     * @author Zbyněk Stara
     * @version 1.4 (Jan-4-2013)
     * @since Nov-6-2012
     */
10055 public int [] globalSearchTeacher(String name) throws IllegalArgumentException,
NoSuchElementException {
    if (name.equals("")) throw new IllegalArgumentException();

10060     Object[] schoolTreeArray = schoolTree.getDataArray();

10065     for (int schoolIndex = 0; schoolIndex < schoolTreeArray.length; schoolIndex++)
    {
10070         try {
10071             School currentSchool = (School) schoolTreeArray[schoolIndex];

10075             int teacherIndex = currentSchool.searchTeacher(name);

10080             int [] teacherIndices = {schoolIndex, teacherIndex};
10085             return teacherIndices;
10090         } catch (IllegalArgumentException ex) { // the teacher name was an empty
string
10095             throw new IllegalArgumentException();
         } catch (NoSuchElementException ex) { // teacher not found in this shool
             continue;
         }
     }

10085         throw new NoSuchElementException(); // if NoSuchElementExceptions encountered
for all schools
    }

    /**
     * The globalSearchTeacher finds a specific teacher.
     * <p>
     * If the teacher exists, this method finds him and returns an array of
     * indices to show where he is. The first element refers to the schoolTree
     * and the second element is the position in the teacherTree of that school
     * where the teacher is.
     * <p>
     * This method starts with the specified entry in the schoolTree; if teacher

```

```

    * was not found in there, it goes from the first entry and sequentially
    * forward, skipping the specified entry in the process.
    *
10100   * @param name name of teacher to search for
    * @param preferredSchoolIndex index of the school with which to start the
    * search (value of -1 is accepted and treated as 0); in these two cases,
    * this method behaves exactly as the method without preferredSchoolIndex
    * parameter.
10105   * @return array of indices of the found teacher (schoolTree, teacherTree)
    * @throws IllegalArgumentException if the name is an empty string or the
    * index is not within the bounds of the schoolTree
    * @throws NoSuchElementException if the name has not been found
    *
10110   * @author Zbyněk Stara
    * @version 1.1 (Jan-4-2013)
    * @since Jan-4-2013
    */
10115   public int [] globalSearchTeacher(String name, int preferredSchoolIndex) throws
IllegalArgumentException, NoSuchElementException {
    if (name.equals("")) throw new IllegalArgumentException();

    Object[] schoolTreeArray = schoolTree.getDataArray();

10120   if (preferredSchoolIndex == -1) {
        preferredSchoolIndex = 0;
    } else if (preferredSchoolIndex >= 0 && preferredSchoolIndex <
schoolTreeArray.length) {
        // do nothing
    } else {
        throw new IllegalArgumentException();
    }

10125   try {
        School currentSchool = (School) schoolTreeArray[preferredSchoolIndex];

        int teacherIndex = currentSchool.searchTeacher(name);

        int[] teacherIndices = {preferredSchoolIndex, teacherIndex};
        return teacherIndices;
    } catch (IllegalArgumentException ex) { // the teacher name was an empty
string
        throw new IllegalArgumentException();
    } catch (NoSuchElementException ex) { // teacher not found in this shool
        // do nothing, let it continue
    }

10130   for (int schoolIndex = 0; schoolIndex < schoolTreeArray.length; schoolIndex++)
{
    if (schoolIndex == preferredSchoolIndex) continue;

    try {
        School currentSchool = (School) schoolTreeArray[schoolIndex];

10140   int teacherIndex = currentSchool.searchTeacher(name);

        int [] teacherIndices = {schoolIndex, teacherIndex};
        return teacherIndices;
    } catch (IllegalArgumentException ex) { // the teacher name was an empty
string
        throw new IllegalArgumentException();
    } catch (NoSuchElementException ex) { // teacher not found in this shool
        continue;
    }
}

10145   }

10150   throw new NoSuchElementException(); // if NoSuchElementExceptions encountered
for all schools
}
10155 /**
    * The globalSearchStudent finds a specific student.
    * <p>
    * If the student exists, this method finds him and returns an array of
    * indices to show where he is. The first element refers to the schoolTree
    * and the second element is the position in the studentTree of that school
    * where the student is.
    * <p>
    * This method starts from the first entry in the schoolTree and goes
    * sequentially forward.
10160
10165
10170
10175

```

```

*
* @param name name of student to search for
* @return array of indices of the found student (schoolTree, studentTree)
* @throws IllegalArgumentException if the name is an empty string
* @throws NoSuchElementException if the name has not been found
*
* @author Zbyněk Stara
* @version 1.3 (Jan-4-2013)
* @since Nov-6-2012
*/
public int [] globalSearchStudent(String name) throws IllegalArgumentException,
NoSuchElementException {
    if (name.equals("")) throw new IllegalArgumentException();

10190    Object[] schoolTreeArray = schoolTree.getDataArray();

    for (int schoolIndex = 0; schoolIndex < schoolTreeArray.length; schoolIndex++)
{
    try {
        School currentSchool = (School) schoolTreeArray[schoolIndex];

        int studentIndex = currentSchool.searchStudent(name); // only if the
student was found, the following comes to happen:

10200        int [] studentIndices = {schoolIndex, studentIndex};
        return studentIndices;
    } catch (IllegalArgumentException ex) { // the student name was an empty
string
        throw new IllegalArgumentException();
    } catch (NoSuchElementException ex) { // student not found in this shool
        continue;
    }
}

10210    throw new NoSuchElementException(); // if NoSuchElementExceptions encountered
for all schools
}

10215 /**
* The globalSearchStudent finds a specific student.
* <p>
* If the student exists, this method finds him and returns an array of
* indices to show where he is. The first element refers to the schoolTree
* and the second element is the position in the studentTree of that school
* where the student is.
* <p>
* This method starts with the specified entry in the schoolTree; if student
* was not found in there, it goes from the first entry and sequentially
* forward, skipping the specified entry in the process.

10220
10225
10230
10235
10240
10245
10250
    * @param name name of student to search for
    * @param preferredSchoolIndex index of the school with which to start the
    * search (value of -1 is accepted and treated as 0); in these two cases,
    * this method behaves exactly as the method without preferredSchoolIndex
    * parameter.
    * @return array of indices of the found student (schoolTree, studentTree)
    * @throws IllegalArgumentException if the name is an empty string or the
    * index is not within the bounds of the schoolTree
    * @throws NoSuchElementException if the name has not been found
    *
    * @author Zbyněk Stara
    * @version 1.1 (Jan-4-2013)
    * @since Jan-4-2013
    */
    public int [] globalSearchStudent(String name, int preferredSchoolIndex) throws
IllegalArgumentException, NoSuchElementException {
        if (name.equals("")) throw new IllegalArgumentException();

        Object[] schoolTreeArray = schoolTree.getDataArray();

        try {
            School currentSchool = (School) schoolTreeArray[preferredSchoolIndex];

            int studentIndex = currentSchool.searchStudent(name);

            int[] studentIndices = {preferredSchoolIndex, studentIndex};
            return studentIndices;
        } catch (IllegalArgumentException ex) { // the student name was an empty
string
    }
}

```

```

10255         throw new IllegalArgumentException();
10256     } catch (NoSuchElementException ex) { // student not found in this school
10257         // do nothing, let it continue
10258     }
10259
10260     for (int schoolIndex = 0; schoolIndex < schoolTreeArray.length; schoolIndex++)
10261     {
10262         if (schoolIndex == preferredSchoolIndex) continue;
10263
10264         try {
10265             School currentSchool = (School) schoolTreeArray[schoolIndex];
10266
10267             int studentIndex = currentSchool.searchStudent(name);
10268
10269             int [] studentIndices = {schoolIndex, studentIndex};
10270             return studentIndices;
10271         } catch (IllegalArgumentException ex) { // the student name was an empty
10272             string
10273                 throw new NoSuchElementException();
10274         } catch (NoSuchElementException ex) { // student not found in this school
10275             continue;
10276         }
10277     }
10278
10279     throw new NoSuchElementException(); // if NoSuchElementExceptions encountered
10280     for all schools
10281     }
10282
10283 /**
10284 * The assignStudentCodes assigns student codes to students.
10285 * <p>
10286 * All students are assigned a code consisting of two parts: a letter, and
10287 * a number. The letter identifies the school the student comes from, while
10288 * the number uniquely identifies the student in the tournament.
10289 *
10290 * @param studentsNumber the number of students in the database
10291 *
10292 * @throws IllegalStateException if the schoolTree is empty.
10293 *
10294 * @author Zbyněk Stara
10295 * @version 1.3 (Jan-5-2013)
10296 * @since Nov-6-2012
10297 */
10298 public void assignStudentCodes(int studentsNumber) throws IllegalStateException {
10299     if (!schoolTree.isEmpty() && studentsNumber > 0) {
10300         char currentLetter = 'A';
10301         int currentNumber = 1;
10302
10303         for (int i = 0; i < schoolTree.size(); i++) {
10304             String firstPart = currentLetter + "";
10305
10306             School currentSchool = (School) schoolTree.getNodeData(i);
10307
10308             currentSchool.setCodeLetter(currentLetter);
10309
10310             currentSchool.setBeginCodeNumber(currentNumber);
10311
10312             for (int j = 0; j < currentSchool.getStudentTreeSize(); j++) {
10313                 currentSchool.setEndCodeNumber(currentNumber);
10314
10315                 String secondPart = currentNumber + "";
10316
10317                 String currentCode = firstPart + secondPart;
10318                 Student currentStudent = (Student)
10319                 currentSchool.getStudentTreeNodeData(j);
10320
10321                 currentStudent.setCode(currentCode);
10322
10323                 currentNumber++;
10324             }
10325
10326             currentLetter++;
10327         }
10328     }
10329     else throw new IllegalStateException();
10330 }
10331
10332 /**
10333 * This method is called upon to ensure that pair codes are correct after

```

```

10335      * a re-assignment of student codes.
10335      *
10335      * @param currentSchool the school whose pairs are to be checked
10335      *
10335      * @author Zbyněk Stara
10335      */
10340      public void correctPairCodes(School currentSchool) {
10340          for (int i = 0; i < schoolTree.size(); i++) {
10340              for (int j = 0; j < currentSchool.getDAPairTreeSize(); j++) {
10340                  StudentPair currentPair = (DAStudentPair)
10345                  currentSchool.getDAPairTreeNodeData(j);
10345                  if (!currentPair.getCode().equals(currentPair.getOriginalCode())) {
10345                      DAStudentPair incorrectPair =
10345                          currentSchool.deleteDAPair(currentPair.getOriginalCode());
10345                      DAStudentPair correctPair = new
10350                      DAStudentPair(incorrectPair.getStudentArray()[0], incorrectPair.getStudentArray()[1]);
10350                      currentSchool.insertDAPair(correctPair);
10350                  }
10350              }
10355              for (int j = 0; j < currentSchool.getDebatePairTreeSize(); j++) {
10355                  StudentPair currentPair = (DebateStudentPair)
10355                  currentSchool.getDebatePairTreeNodeData(j);
10355                  if (!currentPair.getCode().equals(currentPair.getOriginalCode())) {
10355                      DebateStudentPair incorrectPair =
10360                      currentSchool.deleteDebatePair(currentPair.getOriginalCode());
10360                      DebateStudentPair correctPair = new
10360                          DebateStudentPair(incorrectPair.getStudentArray()[0],
10360                              incorrectPair.getStudentArray()[1]);
10360                          currentSchool.insertDebatePair(correctPair);
10365                  }
10365              }
10370      /**
10370      * The searchStudentCode finds a specific student by his code.
10370      * <p>
10370      * If the student code exists, this method finds it and returns an array of
10370      * indices to show which student it refers to. The first element refers to
10370      * the schoolTree and the second element is the position in the studentTree
10370      * of that school where the student is.
10370      *
10370      * @param code student code to search for
10370      * @return array of indices of the found student (schoolTree, studentTree)
10370      * @throws IllegalArgumentException if the code is an empty string
10370      * @throws NoSuchElementException if the code letter is not a valid letter
10370      *
10370      * @author Zbyněk Stara
10370      * @version 1.5 (Jan-6-2013)
10370      * @since Nov-6-2012
10370      */
10380      public int [] searchStudentCode(String code) throws IllegalArgumentException,
10380      NoSuchElementException {
10380          if (!code.equals("")) {
10380              char firstPart = code.charAt(0);
10390              if (firstPart >= 'A' && firstPart <= 'Z') {
10390                  // do nothing
10390              } else if (firstPart >= 'a' && firstPart <= 'z') {
10390                  firstPart -= 32;
10390              } else throw new IllegalArgumentException();
10395              int schoolIndex = firstPart - 65;
10400              School currentSchool = (School) schoolTree.getNodeData(schoolIndex);
10400              if (currentSchool == null) throw new NoSuchElementException();
10405              int minCurrentSchoolCode = currentSchool.getBeginCodeNumber(); // do not
forget that these start with 1, not 0 !!!
10405              int maxCurrentSchoolCode = currentSchool.getEndCodeNumber();
10410              String secondPart = code.substring(1);
10410              int codeNumber;
10410              int studentCodeIndex;

```

```
try {
    codeNumber = Integer.parseInt(secondPart);
10415
    if ((maxCurrentSchoolCode - minCurrentSchoolCode) >= (codeNumber -
minCurrentSchoolCode) && (codeNumber - minCurrentSchoolCode) >= 0) {
        studentCodeIndex = codeNumber - minCurrentSchoolCode; // this
produces numbers in the 0-based format
10420
    } else {
        studentCodeIndex = -1;
    }
} catch (NumberFormatException ex) {
    studentCodeIndex = -1;
}
10425
int[] studentCodeIndicesArray = {schoolIndex, studentCodeIndex};
return studentCodeIndicesArray;
10430
else throw new IllegalArgumentException();
}
}
```

School.java

10435

```

    /**
     * NESDA Tournament Manager 2013
     *
     * By Zbyněk Stara, 000889-045
     * International School of Prague
     * Czech Republic
     *
     * Computer used:
     * Macbook unibody aluminum late 2008
     * Mac OS X 10.6.8
     * 8 GiB of RAM
     *
     * IDE used:
     * NetBeans 6.9.1
     */

```

package data;

import java.util.*;

```

    /**
     * A school is the basic building block of the database's hierarchical data
     * structure. A school carries teachers, students, as well as unpaired duet
     * acting and debate students and duet acting and debate pairs. The class
     * contains all the necessary methods to work with and access the members of the
     * school.
     *
     * @author Zbyněk Stara
     */

```

public class School {

```

    // ATTRIBUTES:
    private String name = "<Not set yet>";
    private char codeLetter = ' ';

```

private int beginCodeNumber = -999;

private int endCodeNumber = -999;

private int ooStudentNumber = 0;

private int oiStudentNumber = 0;

private int isStudentNumber = 0;

private BinarySearchTree teacherTree = new BinarySearchTree(); // Teachers

private BinarySearchTree studentTree = new BinarySearchTree();

private BinarySearchTree unpairedDASStudentTree = new BinarySearchTree(); // Students

private BinarySearchTree unpairedDebateStudentTree = new BinarySearchTree();

private BinarySearchTree daPairTree = new BinarySearchTree(); // StudentPairs

private BinarySearchTree debatePairTree = new BinarySearchTree();

```

    // CONSTRUCTORS:
    public School() {

```

}
 public School(String name) {
 this.name = name;
 }
 public School(String name, char codeLetter, int beginCodeNumber, int endCodeNumber) {
 this.name = name;
 this.codeLetter = codeLetter;
 this.beginCodeNumber = beginCodeNumber;
 this.endCodeNumber = endCodeNumber;
 }

// SCHOOL GETS:

```

    /**
     * The getName method returns the name of the school.
     *
     * @return string with the contents of the name attribute
     *
     * @author Zbyněk Stara
     * @version 1.0 (Nov-7-2012)
     * @since Nov-7-2012

```

```

        */
    public String getName() {
        return name;
}

10515

10520
    /**
     * The getCodeLetter method returns the code letter of the school.
     *
     * @return string with the contents of the codeLetter attribute
     *
     * @author Zbyněk Stara
     * @version 1.0 (Nov-7-2012)
     * @since Nov-7-2012
     */
10525
    public char getCodeLetter() {
        return codeLetter;
    }

10530
    /**
     * The getBeginCodeNumber method returns the beginning code number of the
     * school.
     *
     * @return integer with the beginning of the code number range
     *
     * @author Zbyněk Stara
     * @version 1.0 (Jan-5-2013)
     * @since Jan-5-2013
     */
10535
    public int getBeginCodeNumber() {
        return beginCodeNumber;
    }

10540
    /**
     * The getEndCodeNumber method returns the end code number of the school.
     *
     * @return integer with the end of the code number range
     *
     * @author Zbyněk Stara
     * @version 1.0 (Jan-5-2013)
     * @since Jan-5-2013
     */
10545
    public int getEndCodeNumber() {
        return endCodeNumber;
    }

10550
    // SCHOOL SETS:
    /**
     * The setName method sets the name of the school to the contents of the
     * name parameter.
     *
     * @param name the new name for the school
     *
     * @author Zbyněk Stara
     * @version 1.0 (Nov-8-2012)
     * @since Nov-8-2012
     */
10555
    public void setName(String name) {
        this.name = name;
    }

10560
    /**
     * The setCodeLetter method sets the codeLetter of the school to a given
     * char.
     *
     * @param codeLetter a char with the new codeLetter of the school
     *
     * @author Zbyněk Stara
     * @version 1.0 (Nov-8-2012)
     * @since Nov-8-2012
     */
10565
    public void setCodeLetter(char codeLetter) {
        this.codeLetter = codeLetter;
    }

10570
    /**
     * The setBeginCodeNumber method sets the beginning of the code number range
     * of the school to a given integer.
     *
     * @param beginCodeNumber an integer with the beginning of the code number
     */
10575
    public void setBeginCodeNumber(int beginCodeNumber) {
        this.beginCodeNumber = beginCodeNumber;
    }

10580
    /**
     * The setEndCodeNumber method sets the end of the code number range
     * of the school to a given integer.
     *
     * @param endCodeNumber an integer with the end of the code number
     */
10585
    public void setEndCodeNumber(int endCodeNumber) {
        this.endCodeNumber = endCodeNumber;
    }

```

```

10590     * range
10591     *
10592     * @author Zbyněk Stara
10593     * @version 1.0 (Jan-5-2013)
10594     * @since Jan-5-2013
10595     */
10596     public void setBeginCodeNumber(int beginCodeNumber) {
10597         this.beginCodeNumber = beginCodeNumber;
10598     }
10600 /**
10601     * The setEndCodeNumber method sets the end of the code number range of the
10602     * school to a given integer.
10603     *
10604     * @param endCodeNumber an integer with the end of the code number range
10605     *
10606     * @author Zbyněk Stara
10607     * @version 1.0 (Jan-5-2013)
10608     * @since Jan-5-2013
10609     */
10610     public void setEndCodeNumber(int endCodeNumber) {
10611         this.endCodeNumber = endCodeNumber;
10612     }
10613 /**
10614     * This method increments the value of the ooStudentNumber variable.
10615     *
10616     * @author Zbyněk Stara
10617     */
10618     public void incrementOOStudentNumber() {
10619         this.ooStudentNumber += 1;
10620     }
10621 /**
10622     * This method decrements the value of the ooStudentNumber variable.
10623     *
10624     * @author Zbyněk Stara
10625     */
10626     public void decrementOOStudentNumber() throws IllegalStateException {
10627         if (ooStudentNumber == 0) {
10628             throw new IllegalStateException("Number of OO students cannot be lower
10629 than zero.");
10630         } else {
10631             this.ooStudentNumber -= 1;
10632         }
10633     }
10634 /**
10635     * This method returns the value of the ooStudentNumber variable.
10636     *
10637     * @return an int with the value of ooStudentNumber
10638     *
10639     * @author Zbyněk Stara
10640     */
10641     public int getOOStudentNumber() {
10642         return this.ooStudentNumber;
10643     }
10644 /**
10645     * This method increments the value of the oiStudentNumber variable.
10646     *
10647     * @author Zbyněk Stara
10648     */
10649     public void incrementOIStudentNumber() {
10650         this.oiStudentNumber += 1;
10651     }
10652 /**
10653     * This method decrements the value of the oiStudentNumber variable.
10654     *
10655     * @author Zbyněk Stara
10656     */
10657     public void decrementOIStudentNumber() throws IllegalStateException {
10658         if (oiStudentNumber == 0) {
10659             throw new IllegalStateException("Number of OI students cannot be lower
10660 than zero.");
10661         } else {
10662             this.oiStudentNumber -= 1;
10663         }
10664 
```

```

10670    }
10671 /**
10672 * This method returns the value of the oiStudentNumber variable.
10673 *
10674 * @return an int with the value of oiStudentNumber
10675 *
10676 * @author Zbyněk Stara
10677 */
10678 public int getOISStudentNumber() {
10679     return this.oiStudentNumber;
10680 }

10681 /**
10682 * This method increments the value of the isStudentNumber variable.
10683 *
10684 * @author Zbyněk Stara
10685 */
10686 public void incrementISStudentNumber() {
10687     this.isStudentNumber += 1;
10688 }

10689 /**
10690 * This method decrements the value of the isStudentNumber variable.
10691 *
10692 * @author Zbyněk Stara
10693 */
10694 public void decrementISStudentNumber() throws IllegalStateException {
10695     if (isStudentNumber == 0) {
10696         throw new IllegalStateException("Number of IS students cannot be lower
than zero.");
10697     } else {
10698         this.isStudentNumber -= 1;
10699     }
10700 }

10701 /**
10702 * This method returns the value of the isStudentNumber variable.
10703 *
10704 * @return an int with the value of isStudentNumber
10705 */
10706 public int getISStudentNumber() {
10707     return this.isStudentNumber;
10708 }

10709 // EXTRA METHODS:
10710 /**
10711 * This method calls for all students of this school the updateStudentPairs
10712 * method.
10713 *
10714 * @author Zbyněk Stara
10715 */
10716 public void updateStudentPairs() {
10717     for (int i = 0; i < this.getStudentTreeSize(); i++) {
10718         Student currentStudent = (Student) this.getStudentTreeNodeData(i);
10719
10720         currentStudent.updateStudentPairs();
10721     }
10722 }

10723 // TEACHER METHODS:
10724 /**
10725 * The getTeacherArray method returns an array of teachers, corresponding to
10726 * the teacherTree
10727 * <p>
10728 * The teacherTree is traversed in order and its every member is added to an
10729 * array. This array is then returned by the function
10730 *
10731 * @return array of teachers
10732 *
10733 * @author Zbyněk Stara
10734 * @version 1.0 (Nov-8-2012)
10735 * @since Nov-8-2012
10736 */
10737 public Teacher [] getTeacherArray() {
10738     Object [] tempArray = teacherTree.getDataArray();
10739     Teacher [] returnArray = new Teacher [tempArray.length];

```

```

10750     for (int i = 0; i < tempArray.length; i++) {
10751         returnArray[i] = (Teacher) tempArray[i];
10752     }
10753
10754     return returnArray;
10755 }
10756 /**
10757 * This method returns the teacher from this school that has a given name.
10758 *
10759 * @param name the name of the teacher to search for
10760 * @return the teacher with the corresponding name
10761 * @throws IllegalArgumentException if the name given to this method is an
10762 * empty string
10763 * @throws NoSuchElementException if the name could not be found in this
10764 * school
10765 *
10766 * @author Zbyněk Stara
10767 */
10768 public Teacher getTeacher(String name) throws IllegalArgumentException,
10769 NoSuchElementException {
10770     if (!name.equals("")) {
10771         int teacherIndex = this.searchTeacher(name);
10772         Teacher teacher = (Teacher) this.getTeacherTreeNodeData(teacherIndex);
10773         return teacher;
10774     } else throw new IllegalArgumentException();
10775 }
10776 /**
10777 * The getTeacherTreeNodeData method returns the data for a given node of
10778 * the teacherTree.
10779 * <p>
10780 * A node of the teacherTree, identified by the index parameter, is examined
10781 * and the content of its data attribute is returned as object.
10782 *
10783 * @param index the node which will be examined
10784 * @return the data this node carries
10785 *
10786 * @author Zbyněk Stara
10787 * @version 1.0 (Nov-8-2012)
10788 * @since Nov-8-2012
10789 */
10790 public Object getTeacherTreeNodeData(int index) {
10791     return teacherTree.getNodeData(index);
10792 }
10793 /**
10794 * The getTeacherTreeSize returns the size of the teacherTree.
10795 *
10796 * @return the size of the teacherTree, as determined by the size() function
10797 *
10798 * @author Zbyněk Stara
10799 * @version 1.0 (Nov-8-2012)
10800 * @since Nov-8-2012
10801 */
10802 public int getTeacherTreeSize() {
10803     return teacherTree.size();
10804 }
10805 /**
10806 * This insertTecaher method adds a new teacher to the teacherTree.
10807 * <p>
10808 * The teacher is added with the name specified in the parameter.
10809 *
10810 * @param name the name of the teacher to be added
10811 * @throws IllegalArgumentException if there is already a teacher of the
10812 * specified name or the name is an empty string
10813 *
10814 * @author Zbyněk Stara
10815 * @version 1.1 (Jan-4-2013)
10816 * @since Jan-4-2013
10817 */
10818 public void insertTeacher(String name) throws IllegalArgumentException {
10819     if (!teacherTree.contains(name) && !name.equals("")) {
10820         Teacher newTeacher = new Teacher(name, this);
10821         teacherTree.insert(newTeacher, name);
10822     }
10823     else throw new IllegalArgumentException();

```

```

    }

10830 /**
 * This insertTeacher method adds a specified teacher to the teacherTree.
 *
 * @param teacher the teacher to be added
 * @throws IllegalArgumentException if there is already a teacher of the
 * specified name or the name is an empty string
 *
 * @author Zbyněk Stara
 * @version 1.1 (Jan-4-2013)
 * @since Dec-5-2012
 */
10840 public void insertTeacher(Teacher teacher) throws IllegalArgumentException {
    if (!teacherTree.contains(teacher.getName()) && !teacher.getName().equals(""))
{
    teacherTree.insert(teacher, teacher.getName());
}
else throw new IllegalArgumentException();
}

10845 /**
 * The deleteTeacher method removes a specific teacher from the teacherTree.
 * <p>
 * The teacher of the specified name is deleted, provided that it exists. It
 * is then returned by the function.
 *
 * @param name the name of the teacher to be deleted
 * @return the deleted teacher
 * @throws NoSuchElementException if there is no teacher of the specified
 * name
 *
 * @author Zbyněk Stara
 * @version 1.1 (Jan-4-2013)
 * @since Dec-5-2012
 */
10850 public Teacher deleteTeacher(String name) throws NoSuchElementException {
    return (Teacher) teacherTree.delete(name);
}

10855 /**
 * The deleteTeacher method removes a specific teacher from the teacherTree.
 * <p>
 * The teacher at the specified index is removed from the teacherTree and is
 * returned by this function
 *
 * @param index the index of the teacher in the teacherTree
 * @return the deleted teacher
 * @throws IllegalArgumentException if the index is not within the bounds of
 * the teacherTree
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-4-2013)
 * @since Jan-4-2013
 */
10860 public Teacher deleteTeacher(int index) throws IllegalArgumentException {
    if (index >= 0 && index < teacherTree.size()) {
        Teacher deletedTeacher = (Teacher) teacherTree.getNodeData(index);
        String deleteKey = deletedTeacher.getName();
        teacherTree.delete(deleteKey);
        return deletedTeacher;
    }
    else throw new IllegalArgumentException();
}

10865 /**
 * The editTeacher method renames a specific teacher.
 * <p>
 * The teacher is first removed from the teacherTree, renamed, and then
 * re-inserted under a new name.
 *
 * @param index the teacher's index in the teacherTree
 * @param name new name for the teacher
 * @throws IllegalArgumentException if the name is already used in the
 * teacherTree or the new name is an empty string or the index is not
 * within the bounds of the teacherTree
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-4-2013)
 */
10870
10875
10880
10885
10890
10895
10900
10905

```

```

    * @since Jan-4-2013
    */
10910 public void editTeacher(int index, String name) throws IllegalArgumentException {
    if (!teacherTree.contains(name) && !name.equals("") && index >= 0 && index <
teacherTree.size()) {
        Teacher editedTeacher = (Teacher) teacherTree.getNodeData(index);
        String editKey = editedTeacher.getName();
        teacherTree.delete(editKey);
        editedTeacher.setName(name);
        teacherTree.insert(editedTeacher, name);
    }
    else throw new IllegalArgumentException();
}

10920 /**
    * The searchTeacher method finds a specific teacher.
    * <p>
    * If the teacher exists, this method finds it and returns its index in the
    * teacherTree.
    *
    * @param name name of teacher to search for
    * @return teacher's index in the teacherTree
    * @throws IllegalArgumentException if the name is an empty string
    * @throws NoSuchElementException if the name has not been found
    *
    * @author Zbyněk Stara
    * @version 1.1 (Jan-4-2013)
    * @since Jan-3-2013
    */
10930 public int searchTeacher(String name) throws IllegalArgumentException,
NoSuchElementException {
    if (!name.equals("")) {
        Object [] teacherTreeArray = teacherTree.getDataArray();
        boolean teacherFound = false;
        int teacherIndex = -999;
        for (int i = 0; i < teacherTreeArray.length; i++) {
            if (((Teacher) teacherTreeArray[i]).getName().equals(name)) {
                teacherFound = true;
                teacherIndex = i;
                break;
            }
        }
        if (teacherFound) {
            return teacherIndex;
        }
        else throw new NoSuchElementException();
    }
    else throw new IllegalArgumentException();
}

10940 // STUDENT METHODS:
10945 /**
    * The getStudentArray method returns an array of students, corresponding to
    * the studentTree
    * <p>
    * The studentTree is traversed in order and its every member is added to an
    * array. This array is then returned by the function
    *
    * @return array of students
    *
    * @author Zbyněk Stara
    * @version 1.0 (Nov-8-2012)
    * @since Nov-8-2012
    */
10950 public Student [] getStudentArray() {
    Object [] tempArray = studentTree.getDataArray();
    Student [] returnArray = new Student [tempArray.length];

    for (int i = 0; i < tempArray.length; i++) {
        returnArray[i] = (Student) tempArray[i];
    }

    return returnArray;
}

10955 /**
    * This method returns the student from this school that has a given name.
    *
    * @param name the name of the student to search for
    */
10960
10965
10970
10975
10980

```

```

10985     * @return the student with the corresponding name
10986     * @throws IllegalArgumentException if the name given to this method is an
10987     * empty string
10988     * @throws NoSuchElementException if the name could not be found in this
10989     * school
10990     *
10991     * @author Zbyněk Stara
10992     */
10993     public Student getStudent(String name) throws IllegalArgumentException,
10994     NoSuchElementException {
10995         if (!name.equals("")) {
10996             int studentIndex = this.searchStudent(name);
10997             Student student = (Student) this.getStudentTreeNodeData(studentIndex);
10998             return student;
10999         } else throw new IllegalArgumentException();
11000     }

11005     /**
11006      * The getStudentTreeNodeData method returns the data for a given node of
11007      * the studentTree.
11008      * <p>
11009      * A node of the studentTree, identified by the index parameter, is examined
11010      * and the content of its data attribute is returned as object.
11011      *
11012      * @param index the node which will be examined
11013      * @return the data this node carries
11014      *
11015      * @author Zbyněk Stara
11016      * @version 1.0 (Nov-8-2012)
11017      * @since Nov-8-2012
11018      */
11019     public Object getStudentTreeNodeData(int index) {
11020         return studentTree.getNodeData(index);
11021     }

11025     /**
11026      * The getStudentTreeSize returns the size of the studentTree.
11027      *
11028      * @return the size of the studentTree, as determined by the size() function
11029      *
11030      * @author Zbyněk Stara
11031      * @version 1.0 (Nov-8-2012)
11032      * @since Nov-8-2012
11033      */
11034     public int getStudentTreeSize() {
11035         return studentTree.size();
11036     }

11040     /**
11041      * This insertStudent method adds a new student to the studentTree.
11042      * <p>
11043      * The student is added with the name specified in the parameter.
11044      *
11045      * @param name the name of the student to be added
11046      * @throws IllegalArgumentException if there is already a student of the
11047      * specified name or the name is an empty string
11048      *
11049      * @author Zbyněk Stara
11050      * @version 1.1 (Jan-4-2013)
11051      * @since Jan-4-2013
11052      */
11053     public void insertStudent(String name) throws IllegalArgumentException {
11054         if (!studentTree.contains(name) && !name.equals("")) {
11055             Student newStudent = new Student(name, this);
11056             studentTree.insert(newStudent, name);
11057         }
11058         else throw new IllegalArgumentException();
11059     }

11060     /**
11061      * This insertStudent method adds a specified student to the studentTree.
11062      *
11063      * @param student the student to be added
11064      * @throws IllegalArgumentException if there is already a student of the
11065      * specified name or the name is an empty string
11066      *
11067      * @author Zbyněk Stara
11068      * @version 1.1 (Jan-4-2013)
11069      * @since Dec-5-2012

```

```

11065      */
11066  public void insertStudent(Student student) throws IllegalArgumentException {
11067      if (!studentTree.contains(student.getName()) && !student.getName().equals(""))
11068      {
11069          studentTree.insert(student, student.getName());
11070      }
11071      else throw new IllegalArgumentException();
11072  }
11073  /**
11074  * The deleteStudent method removes a specific student from the studentTree.
11075  * <p>
11076  * The student of the specified name is deleted, provided that it exists. It
11077  * is then returned by the function.
11078  *
11079  * @param name the name of the student to be deleted
11080  * @return the deleted student
11081  * @throws NoSuchElementException if there is no student of the specified
11082  * name
11083  *
11084  * @author Zbyněk Stara
11085  * @version 1.1 (Jan-4-2013)
11086  * @since Dec-5-2012
11087  */
11088  public Student deleteStudent(String name) throws NoSuchElementException {
11089      return (Student) studentTree.delete(name);
11090  }
11091  /**
11092  * The deleteStudent method removes a specific student from the studentTree.
11093  * <p>
11094  * The student at the specified index is removed from the studentTree and is
11095  * returned by this function
11096  *
11097  * @param index the index of the student in the studentTree
11098  * @return the deleted student
11099  * @throws IllegalArgumentException if the index is not within the bounds of
11100  * the studentTree
11101  *
11102  * @author Zbyněk Stara
11103  * @version 1.0 (Jan-4-2013)
11104  * @since Jan-4-2013
11105  */
11106  public Student deleteStudent(int index) throws IllegalArgumentException {
11107      if (index >= 0 && index < studentTree.size()) {
11108          Student deletedStudent = (Student) studentTree.getNodeData(index);
11109          String deleteKey = deletedStudent.getName();
11110          studentTree.delete(deleteKey);
11111          return deletedStudent;
11112      }
11113      else throw new IllegalArgumentException();
11114  }
11115  /**
11116  * The editStudent method renames a specific student.
11117  * <p>
11118  * The student is first removed from the studentTree, renamed, and then
11119  * re-inserted under a new name.
11120  *
11121  * @param index the student's index in the studentTree
11122  * @param name new name for the student
11123  * @throws IllegalArgumentException if the name is already used in the
11124  * studentTree or the new name is an empty string or the index is not
11125  * within the bounds of the studentTree
11126  *
11127  * @author Zbyněk Stara
11128  * @version 1.0 (Jan-4-2013)
11129  * @since Jan-4-2013
11130  */
11131  public void editStudent(int index, String name) throws IllegalArgumentException {
11132      if (!studentTree.contains(name) && !name.equals("") && index >= 0 && index <
11133      studentTree.size()) {
11134          Student editedStudent = (Student) studentTree.getNodeData(index);
11135          String editKey = editedStudent.getName();
11136          studentTree.delete(editKey);
11137
11138          editedStudent.setName(name);
11139          studentTree.insert(editedStudent, name);
11140      }
11141  }

```

```

try {
    Student editedStudentDA = (Student)
11145  unpairedDASTudentTree.delete(editKey);
        unpairedDASTudentTree.insert(editedStudentDA, name);
    } catch (NoSuchElementException ex){
        // continue
    }
}
11150
try {
    Student editedStudentDebate = (Student)
unpairedDebateStudentTree.delete(editKey);
        unpairedDebateStudentTree.insert(editedStudentDebate, name);
    } catch (NoSuchElementException ex) {
        // continue
    }
}
11155
else throw new IllegalArgumentException();
11160
}

/**
 * The searchStudent method finds a specific student.
 * <p>
 * If the student exists, this method finds it and returns its index in the
 * studentTree.
 *
 * @param name name of student to search for
 * @return student's index in the studentTree
 * @throws IllegalArgumentException if the name is an empty string
 * @throws NoSuchElementException if the name has not been found
 *
 * @author Zbyněk Stara
 * @version 1.1 (Jan-4-2013)
 * @since Jan-3-2013
 */
11165
public int searchStudent(String name) throws IllegalArgumentException,
NoSuchElementException {
11170
    if (!name.equals("")) {
        Object [] studentTreeArray = studentTree.getDataArray();
        boolean studentFound = false;
        int studentIndex = -999;
        for (int i = 0; i < studentTreeArray.length; i++) {
            if (((Student) studentTreeArray[i]).getName().equals(name)) {
                studentFound = true;
                studentIndex = i;
                break;
            }
        }
        if (studentFound) {
            return studentIndex;
        }
        else throw new NoSuchElementException();
    }
11175
    else throw new IllegalArgumentException();
}
11180
}

// UNPAIRED DA STUDENT METHODS:
/**
 * The getUnpairedDASTudentArray method returns an array of students,
 * corresponding to the unpairedDASTudentTree.
 * <p>
 * The unpairedDASTudentTree is traversed in order and its every member is
 * added to an array. This array is then returned by the function.
 *
 * @return array of students
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11185
public Student [] getUnpairedDASTudentArray() {
    Object [] tempArray = unpairedDASTudentTree.getDataArray();
    Student [] returnArray = new Student [tempArray.length];
11190
    for (int i = 0; i < tempArray.length; i++) {
        returnArray[i] = (Student) tempArray[i];
    }
}
11195
else throw new IllegalArgumentException();
}

11200
}

// UNPAIRED DA STUDENT METHODS:
/**
 * The getUnpairedDASTudentArray method returns an array of students,
 * corresponding to the unpairedDASTudentTree.
 * <p>
 * The unpairedDASTudentTree is traversed in order and its every member is
 * added to an array. This array is then returned by the function.
 *
 * @return array of students
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11205
public Student [] getUnpairedDASTudentArray() {
    Object [] tempArray = unpairedDASTudentTree.getDataArray();
    Student [] returnArray = new Student [tempArray.length];
11210
    for (int i = 0; i < tempArray.length; i++) {
        returnArray[i] = (Student) tempArray[i];
    }
}
11215
return returnArray;
}

```

```

11225 /**
 * This method returns the unpaired DA student from this school that has
 * a given name.
 *
 * @param name the name of the unpaired DA student to search for
 * @return the unpaired DA student with the corresponding name
 * @throws IllegalArgumentException if the name given to this method is an
 * empty string
 * @throws NoSuchElementException if the name could not be found in this
 * school
 *
 * @author Zbyněk Stara
 */
11230 public Student getUnpairedDASTudent(String name) throws IllegalArgumentException,
NoSuchElementException {
    if (!name.equals("")) {
        int unpairedDASTudentIndex = this.searchUnpairedDASTudent(name);
        Student unpairedDASTudent = (Student)
this.getUnpairedDASTudentTreeNodeData(unpairedDASTudentIndex);
        return unpairedDASTudent;
    } else throw new IllegalArgumentException();
}
11245 /**
 * The getUnpairedDASTudentTreeNodeData method returns the data for a given
 * node of the unpairedDASTudentTree.
 * <p>
 * A node of the unpairedDASTudentTree, identified by the index parameter,
 * is examined and the content of its data attribute is returned as object.
 *
 * @param index the node which will be examined
 * @return the data this node carries
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11250 public Object getUnpairedDASTudentTreeNodeData(int index) {
    return unpairedDASTudentTree.getNodeData(index);
}

11255 /**
 * The getUnpairedDASTudentTreeSize returns the size of the
 * unpairedDASTudentTree.
 *
 * @return the size of the unpairedDASTudentTree, as determined by the
 * size() function
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11260 public int getUnpairedDASTudentTreeSize() {
    return unpairedDASTudentTree.size();
}

11265 /**
 * The insertUnpairedDASTudent method adds a specified student to the
 * unpairedDASTudentTree.
 *
 * @param student the student to be added
 * @throws IllegalArgumentException if there is already a student of the
 * specified name or the name is an empty string
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11270 public void insertUnpairedDASTudent(Student student) throws
IllegalArgumentException {
    if (!unpairedDASTudentTree.contains(student.getName()) &&
!student.getName().equals("")) {
        unpairedDASTudentTree.insert(student, student.getName());
    }
}

11275 /**
 * The insertUnpairedDASTudent method adds a specified student to the
 * unpairedDASTudentTree.
 *
 * @param student the student to be added
 * @throws IllegalArgumentException if there is already a student of the
 * specified name or the name is an empty string
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11280 public void insertUnpairedDASTudent(Student student) throws
IllegalArgumentException {
    if (!unpairedDASTudentTree.contains(student.getName()) &&
!student.getName().equals("")) {
        unpairedDASTudentTree.insert(student, student.getName());
    }
}

11285 /**
 * The insertUnpairedDASTudent method adds a specified student to the
 * unpairedDASTudentTree.
 *
 * @param student the student to be added
 * @throws IllegalArgumentException if there is already a student of the
 * specified name or the name is an empty string
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11290 public void insertUnpairedDASTudent(Student student) throws
IllegalArgumentException {
    if (!unpairedDASTudentTree.contains(student.getName()) &&
!student.getName().equals("")) {
        unpairedDASTudentTree.insert(student, student.getName());
    }
}

11295 /**
 * The insertUnpairedDASTudent method adds a specified student to the
 * unpairedDASTudentTree.
 *
 * @param student the student to be added
 * @throws IllegalArgumentException if there is already a student of the
 * specified name or the name is an empty string
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11300 public void insertUnpairedDASTudent(Student student) throws
IllegalArgumentException {
    if (!unpairedDASTudentTree.contains(student.getName()) &&
!student.getName().equals("")) {
        unpairedDASTudentTree.insert(student, student.getName());
    }
}

11305 /**
 * The insertUnpairedDASTudent method adds a specified student to the
 * unpairedDASTudentTree.
 *
 * @param student the student to be added
 * @throws IllegalArgumentException if there is already a student of the
 * specified name or the name is an empty string
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11310 public void insertUnpairedDASTudent(Student student) throws
IllegalArgumentException {
    if (!unpairedDASTudentTree.contains(student.getName()) &&
!student.getName().equals("")) {
        unpairedDASTudentTree.insert(student, student.getName());
    }
}

```

```

        Student currentStudent2 = (Student) studentTree.search(student.getName());
        currentStudent2.setDAUnpaired(true);
    }
    else throw new IllegalArgumentException();
11305 }

/**
 * The deleteUnpairedDASStudent method removes a specific student from the
 * unpairedDASStudentTree.
 * <p>
 * The student of the specified name is deleted, provided that it exists. It
 * is then returned by the function.
 *
 * @param name the name of the student to be deleted
 * @return the deleted student
 * @throws NoSuchElementException if there is no student of the specified
 * name
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-11-2013)
 * @since Dec-11-2012
 */
public Student deleteUnpairedDASStudent(String name) throws NoSuchElementException
11325 {
    Student deletedStudent = (Student) unpairedDASStudentTree.delete(name);
    Student currentStudent = (Student)
studentTree.search(deletedStudent.getName());
    currentStudent.setDAUnpaired(false);

11330     return deletedStudent;
}

/**
 * The deleteUnpairedDASStudent method removes a specific student from the
 * unpairedDASStudentTree.
 * <p>
 * The student at the specified index is removed from the
 * unpairedDASStudentTree and is returned by this function.
 *
 * @param index the index of the student in the unpairedDASStudentTree
 * @return the deleted student
 * @throws IllegalArgumentException if the index is not within the bounds of
 * the unpairedDASStudentTree
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
11340 public Student deleteUnpairedDASStudent(int index) throws IllegalArgumentException
11350 {
    if (index >= 0 && index < unpairedDASStudentTree.size()) {
        Student deletedStudent = (Student)
unpairedDASStudentTree.getNodeData(index);
        String deleteKey = deletedStudent.getName();
        unpairedDASStudentTree.delete(deleteKey);

11355         Student currentStudent = (Student)
studentTree.search(deletedStudent.getName());
        currentStudent.setDAUnpaired(false);

11360         return deletedStudent;
    }
    else throw new IllegalArgumentException();
}

11365 /**
 * The searchUnpairedDASStudent method finds a specific unpaired DA student.
 * <p>
 * If the student exists, this method finds it and returns its index in the
 * unpairedDASStudentTree.
 *
 * @param name name of student to search for
 * @return student's index in the unpairedDASStudentTree
 * @throws IllegalArgumentException if the name is an empty string
 * @throws NoSuchElementException if the name has not been found
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-11-2013)
 * @since Jan-11-2013
 */
11375

```

```

11380      /*
11381      public int searchUnpairedDASStudent(String name) throws IllegalArgumentException,
11382      NoSuchElementException {
11383          if (!name.equals("")) {
11384              Object [] unpairedDASStudentTreeArray =
11385                  unpairedDASStudentTree.getDataArray();
11386              boolean unpairedDASStudentFound = false;
11387              int unpairedDASStudentIndex = -999;
11388              for (int i = 0; i < unpairedDASStudentTreeArray.length; i++) {
11389                  if (((Student) unpairedDASStudentTreeArray[i]).getName().equals(name))
11390                      {
11391                          unpairedDASStudentFound = true;
11392                          unpairedDASStudentIndex = i;
11393                          break;
11394                      }
11395                  }
11396                  if (unpairedDASStudentFound) {
11397                      return unpairedDASStudentIndex;
11398                  }
11399                  else throw new NoSuchElementException();
11400              }
11401              else throw new IllegalArgumentException();
11402          }
11403
11404 // UNPAIRED DEBATE STUDENT METHODS:
11405 /**
11406 * The getUnpairedDebateStudentArray method returns an array of students,
11407 * corresponding to the unpairedDebateStudentTree.
11408 * <p>
11409 * The unpairedDebateStudentTree is traversed in order and its every member
11410 * is added to an array. This array is then returned by the function.
11411 *
11412 * @return array of students
11413 *
11414 * @author Zbyněk Stara
11415 * @version 1.0 (Jan-8-2013)
11416 * @since Jan-8-2013
11417 */
11418 public Student [] getUnpairedDebateStudentArray() {
11419     Object [] tempArray = unpairedDebateStudentTree.getDataArray();
11420     Student [] returnArray = new Student [tempArray.length];
11421
11422     for (int i = 0; i < tempArray.length; i++) {
11423         returnArray[i] = (Student) tempArray[i];
11424     }
11425
11426     return returnArray;
11427 }
11428 /**
11429 * This method returns the unpaired debate student from this school that has
11430 * a given name.
11431 *
11432 * @param name the name of the unpaired debate student to search for
11433 * @return the unpaired debate student with the corresponding name
11434 * @throws IllegalArgumentException if the name given to this method is an
11435 * empty string
11436 * @throws NoSuchElementException if the name could not be found in this
11437 * school
11438 *
11439 * @author Zbyněk Stara
11440 */
11441 public Student getUnpairedDebateStudent(String name) throws
11442     IllegalArgumentException, NoSuchElementException {
11443     if (!name.equals("")) {
11444         int unpairedDebateStudentIndex = this.searchUnpairedDebateStudent(name);
11445         Student unpairedDebateStudent = (Student)
11446             this.getUnpairedDebateStudentTreeNodeData(unpairedDebateStudentIndex);
11447         return unpairedDebateStudent;
11448     } else throw new IllegalArgumentException();
11449 }
11450 /**
11451 * The getUnpairedDebateStudentTreeNodeData method returns the data for a
11452 * given node of the unpairedDebateStudentTree.
11453 * <p>
11454 * A node of the unpairedDebateStudentTree, identified by the index
11455 * parameter, is examined and the content of its data attribute is returned
11456 * as object.

```

```

11460      *
11460      * @param index the node which will be examined
11460      * @return the data this node carries
11460      *
11460      * @author Zbyněk Stara
11460      * @version 1.0 (Jan-8-2013)
11460      * @since Jan-8-2013
11460      */
11465      public Object getUnpairedDebateStudentTreeNodeData(int index) {
11465          return unpairedDebateStudentTree.getNodeData(index);
11470      }
11470      /**
11470      * The getUnpairedDebateStudentTreeSize returns the size of the
11470      * unpairedDebateStudentTree.
11470      *
11470      * @return the size of the unpairedDebateStudentTree, as determined by the
11470      * size() function
11470      *
11470      * @author Zbyněk Stara
11470      * @version 1.0 (Jan-8-2013)
11470      * @since Jan-8-2013
11470      */
11475      public int getUnpairedDebateStudentTreeSize() {
11475          return unpairedDebateStudentTree.size();
11480      }
11485      /**
11485      * The insertUnpairedDebateStudent method adds a specified student to the
11485      * unpairedDebateStudentTree.
11485      *
11485      * @param student the student to be added
11485      * @throws IllegalArgumentException if there is already a student of the
11485      * specified name or the name is an empty string
11485      *
11485      * @author Zbyněk Stara
11485      * @version 1.0 (Jan-8-2013)
11485      * @since Jan-8-2013
11485      */
11490      public void insertUnpairedDebateStudent(Student student) throws
11490          IllegalArgumentException {
11490              if (!unpairedDebateStudentTree.contains(student.getName()) &&
11490                  !student.getName().equals("")) {
11490                  unpairedDebateStudentTree.insert(student, student.getName());
11495
11500
11505      Student currentStudent1 = (Student)
11505      unpairedDebateStudentTree.search(student.getName());
11505      currentStudent1.setDebateUnpaired(true);
11510
11510      Student currentStudent2 = (Student) studentTree.search(student.getName());
11510      currentStudent2.setDebateUnpaired(true);
11510
11510      } else throw new IllegalArgumentException();
11510
11515      /**
11515      * The deleteUnpairedDebateStudent method removes a specific student from
11515      * the unpairedDebateStudentTree.
11515      * <p>
11515      * The student of the specified name is deleted, provided that it exists. It
11515      * is then returned by the function.
11515      *
11515      * @param name the name of the student to be deleted
11515      * @return the deleted student
11515      * @throws NoSuchElementException if there is no student of the specified
11515      * name
11515      *
11515      * @author Zbyněk Stara
11515      * @version 1.1 (Jan-4-2013)
11515      * @since Dec-5-2012
11515      */
11520      public Student deleteUnpairedDebateStudent(String name) throws
11520          NoSuchElementException {
11520              Student deletedStudent = (Student) unpairedDebateStudentTree.delete(name);
11520              Student currentStudent = (Student)
11520              studentTree.search(deletedStudent.getName());
11520              currentStudent.setDebateUnpaired(false);
11525
11530
11530      return deletedStudent;

```

```

    }

11540 /**
 * The deleteUnpairedDebateStudent method removes a specific student from
 * the unpairedDebateStudentTree.
 * <p>
 * The student at the specified index is removed from the
 * unpairedDebateStudentTree and is returned by this function.
 *
 * @param index the index of the student in the unpairedDebateStudentTree
 * @return the deleted student
 * @throws IllegalArgumentException if the index is not within the bounds of
 * the unpairedDebateStudentTree
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-8-2013)
 * @since Jan-8-2013
 */
public Student deleteUnpairedDebateStudent(int index) throws
IllegalArgumentException {
    if (index >= 0 && index < unpairedDebateStudentTree.size()) {
        Student deletedStudent = (Student)
11560 unpairedDebateStudentTree.getNodeData(index);
        String deleteKey = deletedStudent.getName();
        unpairedDebateStudentTree.delete(deleteKey);

        Student currentStudent = (Student)
11565 studentTree.search(deletedStudent.getName());
        currentStudent.setDebateUnpaired(false);

        return deletedStudent;
    } else throw new IllegalArgumentException();
}

11570 /**
 * The searchUnpairedDebateStudent method finds a specific unpaired debate
 * student.
 * <p>
 * If the student exists, this method finds it and returns its index in the
 * unpairedDebateStudentTree.
 *
 * @param name name of student to search for
 * @return student's index in the unpairedDebateStudentTree
 * @throws IllegalArgumentException if the name is an empty string
 * @throws NoSuchElementException if the name has not been found
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-11-2013)
 * @since Jan-11-2013
 */
public int searchUnpairedDebateStudent(String name) throws
11590 IllegalArgumentException, NoSuchElementException {
    if (!name.equals("")) {
        Object [] unpairedDebateStudentTreeArray =
unpairedDebateStudentTree.getDataArray();
        boolean unpairedDebateStudentFound = false;
        int unpairedDebateStudentIndex = -999;
        for (int i = 0; i < unpairedDebateStudentTreeArray.length; i++) {
            if (((Student)
11595 unpairedDebateStudentTreeArray[i]).getName().equals(name)) {
                unpairedDebateStudentFound = true;
                unpairedDebateStudentIndex = i;
                break;
            }
        }
        if (unpairedDebateStudentFound) {
            return unpairedDebateStudentIndex;
        } else throw new NoSuchElementException();
    } else throw new IllegalArgumentException();
}

11600 /**
 * The getDAPairArray method returns an array of student pairs,
 * corresponding to the daPairTree.
 * <p>
11615
// DA PAIR METHODS:
/**
 * The getDAPairArray method returns an array of student pairs,
 * corresponding to the daPairTree.
 * <p>

```

```

* The daPairTree is traversed in order and its every member is
* added to an array. This array is then returned by the function.
*
11620 * @return array of student pairs
*
* @author Zbyněk Stara
* @version 1.1 (Jan-10-2013)
* @since Jan-8-2013
*/
11625 public DAStudentPair [] getDAPairArray() {
    Object [] tempArray = daPairTree.getDataArray();
    DAStudentPair [] returnArray = new DAStudentPair [tempArray.length];
11630     for (int i = 0; i < tempArray.length; i++) {
        returnArray[i] = (DAStudentPair) tempArray[i];
    }
11635     return returnArray;
}
11640 /**
* This method returns the DA pair that has a given code.
*
* @param code the code of the pair to search for
* @return the pair with the corresponding code
* @throws IllegalArgumentException if the code given to this method is an
* empty string
* @throws NoSuchElementException if the code could not be found in this
* school
*
* @author Zbyněk Stara
*/
11645 public DAStudentPair getDAPair(String code) throws IllegalArgumentException,
NoSuchElementException {
    if (!code.equals("")) {
        int daPairIndex = this.searchDAPair(code);
        DAStudentPair daPair = (DAStudentPair)
this.getDAPairTreeNodeData(daPairIndex);
        return daPair;
    } else throw new IllegalArgumentException();
}
11660 /**
* The getDAPairTreeNodeData method returns the data for a given node of the
* daPairStudentTree.
* <p>
* A node of the daPairStudentTree, identified by the index parameter,
* is examined and the content of its data attribute is returned as object.
*
* @param index the node which will be examined
* @return the data this node carries
*
* @author Zbyněk Stara
* @version 1.1 (Jan-10-2013)
* @since Jan-8-2013
*/
11665 public Object getDAPairTreeNodeData(int index) {
    return daPairTree.getNodeData(index);
}
11670 /**
* The getDAPairTreeSize returns the size of the daPairTree.
*
* @return the size of the daPairTree, as determined by the size() function
*
* @author Zbyněk Stara
* @version 1.1 (Jan-10-2013)
* @since Jan-8-2013
*/
11675 public int getDAPairTreeSize() {
    return daPairTree.size();
}
11680 /**
* This insertDAPair method adds a specified student pair to the daPairTree.
*
* @param studentPair the student pair to be added
* @throws IllegalArgumentException if there is already a student pair with
* the same students
*
* @author Zbyněk Stara
* @version 1.1 (Jan-10-2013)
* @since Jan-8-2013
*/
11685
11690
11695

```

```

*
* @author Zbyněk Stara
* @version 1.0 (Jan-11-2013)
* @since Jan-11-2013
*/
11700    public void insertDAPair(DAStudentPair studentPair) throws
IllegalArgumentException {
    if (!daPairTree.contains(studentPair.getCode())) {
        daPairTree.insert(studentPair, studentPair.getCode());
11705        Student[] pairStudentArray = studentPair.getStudentArray();

        Student leftStudent = pairStudentArray[0];
        Student currentStudent = (Student)
11710        studentTree.search(leftStudent.getName());
        currentStudent.setDASTudentPair(studentPair);

        Student rightStudent = pairStudentArray[1];
        currentStudent = (Student) studentTree.search(rightStudent.getName());
        currentStudent.setDASTudentPair(studentPair);
    }
    else throw new IllegalArgumentException();
}

11720 /**
* The insertDAPair method adds a student pair of the specified students to
* the daPairTree.
*
* @param student1 the first student to be added
* @param student2 the second student to be added
* @throws IllegalArgumentException if there is already a student pair with
* the same students or codes were not assigned to the students
*
* @author Zbyněk Stara
* @version 1.1 (Jan-10-2013)
* @since Jan-8-2013
*/
11725    public void insertDAPair(Student student1, Student student2) throws
IllegalArgumentException {
    DAStudentPair newStudentPair;
    try {
        newStudentPair = new DAStudentPair(student1, student2);
    } catch (IllegalArgumentException ex) {
        throw new IllegalArgumentException();
}
11730
11735        if (!daPairTree.contains(newStudentPair.getCode())) {
            daPairTree.insert(newStudentPair, newStudentPair.getCode());
}
11740
11745        Student[] pairStudentArray = newStudentPair.getStudentArray();

        Student leftStudent = pairStudentArray[0];
        Student currentStudent = (Student)
studentTree.search(leftStudent.getName());
        currentStudent.setDASTudentPair(newStudentPair);

        Student rightStudent = pairStudentArray[1];
        currentStudent = (Student) studentTree.search(rightStudent.getName());
        currentStudent.setDASTudentPair(newStudentPair);
}
11750
11755        else throw new IllegalArgumentException();
}

11760 /**
* The deleteDAPair method removes a specific DA pair from the daPairTree.
* <p>
* The pair of the specified code is deleted, provided that it exists. It is
* then returned by the function.
*
* @param code the code of the pair to be deleted
* @return the deleted student pair
* @throws NoSuchElementException if there is no student pair of the
* specified code
*
* @author Zbyněk Stara
* @version 1.0 (Jan-11-2013)
* @since Jan-11-2013
*/
11765    public DAStudentPair deleteDAPair(String code) throws NoSuchElementException {
}
11770

```

```

11775     DASTudentPair deletedPair = (DASTudentPair) daPairTree.delete(code);
11776     Student[] pairStudentArray = deletedPair.getStudentArray();
11777
11780     Student leftStudent = pairStudentArray[0];
11781     Student currentStudent = (Student) studentTree.search(leftStudent.getName());
11782     currentStudent.setDASTudentPair(null);
11783
11784     Student rightStudent = pairStudentArray[1];
11785     currentStudent = (Student) studentTree.search(rightStudent.getName());
11786     currentStudent.setDASTudentPair(null);
11787
11788     return deletedPair;
11789 }
11790 /**
11791 * The deleteDAPair method removes a specific student pair from the
11792 * daPairTree.
11793 * <p>
11794 * The student pair at the specified index is removed from the daPairTree
11795 * and is returned by this function.
11796 *
11797 * @param index the index of the student pair in the daPairTree
11798 * @return the deleted student pair
11799 * @throws IllegalArgumentException if the index is not within the bounds of
11800 * the daPairTree
11801 *
11802 * @author Zbyněk Stara
11803 * @version 1.1 (Jan-10-2013)
11804 * @since Jan-8-2013
11805 */
11806 public DASTudentPair deleteDAPair(int index) throws IllegalArgumentException {
11807     if (index >= 0 && index < daPairTree.size()) {
11808         DASTudentPair deletedPair = (DASTudentPair) daPairTree.getNodeData(index);
11809         String deleteKey = deletedPair.getCode();
11810         daPairTree.delete(deleteKey);
11811
11812         Student[] pairStudentArray = deletedPair.getStudentArray();
11813
11814         Student leftStudent = pairStudentArray[0];
11815         Student currentStudent = (Student)
11816             studentTree.search(leftStudent.getName());
11817         currentStudent.setDASTudentPair(null);
11818
11819         Student rightStudent = pairStudentArray[1];
11820         currentStudent = (Student) studentTree.search(rightStudent.getName());
11821         currentStudent.setDASTudentPair(null);
11822
11823         return deletedPair;
11824     } else throw new IllegalArgumentException();
11825 }
11826 /**
11827 * The searchDAPair method finds a specific student.
11828 * <p>
11829 * If the student pair exists, this method finds it and returns its index in
11830 * the daPairTree.
11831 *
11832 * @param code code of the student pair to search for
11833 * @return student's index in the daPairTree
11834 * @throws IllegalArgumentException if the code is an empty string
11835 * @throws NoSuchElementException if the code has not been found
11836 *
11837 * @author Zbyněk Stara
11838 * @version 1.0 (Jan-11-2013)
11839 * @since Jan-11-2013
11840 */
11841 public int searchDAPair(String code) throws IllegalArgumentException,
11842 NoSuchElementException {
11843     if (!code.equals("")) {
11844         Object [] daPairTreeArray = daPairTree.getDataArray();
11845         boolean studentPairFound = false;
11846         int studentPairIndex = -999;
11847         for (int i = 0; i < daPairTreeArray.length; i++) {
11848             if (((DASTudentPair) daPairTreeArray[i]).getCode().equals(code)) {
11849                 studentPairFound = true;
11850                 studentPairIndex = i;
11851                 break;
11852             }
11853         }
11854     }
11855 }

```

```

11855             }
11856         }
11857         if (studentPairFound) {
11858             return studentPairIndex;
11859         }
11860         else throw new NoSuchElementException();
11861     }
11862
11863     // DEBATE PAIR METHODS:
11864     /**
11865      * The getDebatePairArray method returns an array of student pairs,
11866      * corresponding to the debatePairTree.
11867      * <p>
11868      * The debatePairTree is traversed in order and its every member is
11869      * added to an array. This array is then returned by the function.
11870      *
11871      * @return array of student pairs
11872      *
11873      * @author Zbyněk Stara
11874      * @version 1.1 (Jan-10-2013)
11875      * @since Jan-8-2013
11876      */
11877     public DebateStudentPair [] getDebatePairArray() {
11878         Object [] tempArray = debatePairTree.getDataArray();
11879         DebateStudentPair [] returnArray = new DebateStudentPair [tempArray.length];
11880
11881         for (int i = 0; i < tempArray.length; i++) {
11882             returnArray[i] = (DebateStudentPair) tempArray[i];
11883         }
11884
11885         return returnArray;
11886     }
11887
11888     /**
11889      * This method returns the debate pair that has a given code.
11890      *
11891      * @param code the code of the pair to search for
11892      * @return the pair with the corresponding code
11893      * @throws IllegalArgumentException if the code given to this method is an
11894      * empty string
11895      * @throws NoSuchElementException if the code could not be found in this
11896      * school
11897      *
11898      * @author Zbyněk Stara
11899      */
11900     public DebateStudentPair getDebatePair(String code) throws
11901         IllegalArgumentException, NoSuchElementException {
11902         if (!code.equals(""))
11903             int debatePairIndex = this.searchDebatePair(code);
11904             DebateStudentPair debatePair = (DebateStudentPair)
11905             this.getDebatePairTreeNodeData(debatePairIndex);
11906             return debatePair;
11907         } else throw new IllegalArgumentException();
11908     }
11909
11910     /**
11911      * The getDebatePairTreeNodeData method returns the data for a given node of the
11912      * debatePairStudentTree.
11913      * <p>
11914      * A node of the debatePairStudentTree, identified by the index parameter,
11915      * is examined and the content of its data attribute is returned as object.
11916      *
11917      * @param index the node which will be examined
11918      * @return the data this node carries
11919      *
11920      * @author Zbyněk Stara
11921      * @version 1.1 (Jan-10-2013)
11922      * @since Jan-8-2013
11923      */
11924     public Object getDebatePairTreeNodeData(int index) {
11925         return debatePairTree.getNodeData(index);
11926     }
11927
11928     /**
11929      * The getDebatePairTreeSize returns the size of the debatePairTree.
11930      *
11931      * @return the size of the debatePairTree, as determined by the size() function

```

```

11935      *
11935      * @author Zbyněk Stara
11935      * @version 1.1 (Jan-10-2013)
11935      * @since Jan-8-2013
11935      */
11936  public int getDebatePairTreeSize() {
11937      return debatePairTree.size();
11938  }

11940 /**
11940 * This insertDebatePair method adds a specified student pair to the
11940 * debatePairTree.
11940 *
11941 * @param studentPair the student pair to be added
11941 * @throws IllegalArgumentException if there is already a student pair with
11941 * the same students
11941 *
11942 * @author Zbyněk Stara
11942 * @version 1.0 (Jan-11-2013)
11942 * @since Jan-11-2013
11942 */
11943  public void insertDebatePair(DebateStudentPair studentPair) throws
11944 	IllegalArgumentException {
11945      if (!debatePairTree.contains(studentPair.getCode())) {
11946          debatePairTree.insert(studentPair, studentPair.getCode());

11947          Student[] pairStudentArray = studentPair.getStudentArray();
11948          Student leftStudent = pairStudentArray[0];
11949          Student currentStudent = (Student)
11950          studentTree.search(leftStudent.getName());
11951          currentStudent.setDebateStudentPair(studentPair);

11955          Student rightStudent = pairStudentArray[1];
11956          currentStudent = (Student) studentTree.search(rightStudent.getName());
11957          currentStudent.setDebateStudentPair(studentPair);
11958      }
11959      else throw new IllegalArgumentException();
11960  }

11965 /**
11965 * The insertDebatePair method adds a student pair of the specified students to
11965 * the debatePairTree.
11965 *
11966 * @param student1 the first student to be added
11966 * @param student2 the second student to be added
11966 * @throws IllegalArgumentException if there is already a student pair with
11966 * the same students or codes were not assigned to the students
11966 *
11967 * @author Zbyněk Stara
11967 * @version 1.1 (Jan-10-2013)
11967 * @since Jan-8-2013
11967 */
11968  public void insertDebatePair(Student student1, Student student2) throws
11969 	IllegalArgumentException {
11970      DebateStudentPair newStudentPair;
11971      try {
11972          newStudentPair = new DebateStudentPair(student1, student2);
11973      } catch (IllegalArgumentException ex) {
11974          throw new IllegalArgumentException();
11975      }

11975      if (!debatePairTree.contains(newStudentPair.getCode())) {
11976          debatePairTree.insert(newStudentPair, newStudentPair.getCode());

11977          Student[] pairStudentArray = newStudentPair.getStudentArray();
11978          Student leftStudent = pairStudentArray[0];
11979          Student currentStudent = (Student)
11980          studentTree.search(leftStudent.getName());
11981          currentStudent.setDebateStudentPair(newStudentPair);

11985          Student rightStudent = pairStudentArray[1];
11986          currentStudent = (Student) studentTree.search(rightStudent.getName());
11987          currentStudent.setDebateStudentPair(newStudentPair);
11988      }
11989      else throw new IllegalArgumentException();
11990  }

11995 /**
11995 * The insertDebatePair method adds a student pair of the specified students to
11995 * the debatePairTree.
11995 *
11996 * @param student1 the first student to be added
11996 * @param student2 the second student to be added
11996 * @throws IllegalArgumentException if there is already a student pair with
11996 * the same students or codes were not assigned to the students
11996 *
11997 * @author Zbyněk Stara
11997 * @version 1.1 (Jan-10-2013)
11997 * @since Jan-8-2013
11997 */
11998  public void insertDebatePair(Student student1, Student student2) throws
11999 	IllegalArgumentException {
12000      DebateStudentPair newStudentPair;
12001      try {
12002          newStudentPair = new DebateStudentPair(student1, student2);
12003      } catch (IllegalArgumentException ex) {
12004          throw new IllegalArgumentException();
12005      }

12005      if (!debatePairTree.contains(newStudentPair.getCode())) {
12006          debatePairTree.insert(newStudentPair, newStudentPair.getCode());

12007          Student[] pairStudentArray = newStudentPair.getStudentArray();
12008          Student leftStudent = pairStudentArray[0];
12009          Student currentStudent = (Student)
12010          studentTree.search(leftStudent.getName());
12011          currentStudent.setDebateStudentPair(newStudentPair);

12015          Student rightStudent = pairStudentArray[1];
12016          currentStudent = (Student) studentTree.search(rightStudent.getName());
12017          currentStudent.setDebateStudentPair(newStudentPair);
12018      }
12019      else throw new IllegalArgumentException();
12020  }

```

```

/**
 * The deleteDebatePair method removes a specific debate pair from the
 * daPairTree.
 * <p>
 * The student pair of the specified code is deleted, provided that it
 * exists. It is then returned by the function.
 *
 * @param code the code of the student pair to be deleted
 * @return the deleted student pair
 * @throws NoSuchElementException if there is no student pair of the
 * specified code
 *
 * @author Zbyněk Stara
 * @version 1.0 (Jan-11-2013)
 * @since Jan-11-2013
 */
12015    public DebateStudentPair deleteDebatePair(String code) throws
NoSuchElementException {
    DebateStudentPair deletedPair = (DebateStudentPair)
debatePairTree.delete(code);

    Student[] pairStudentArray = deletedPair.getStudentArray();

12020    Student leftStudent = pairStudentArray[0];
    Student currentStudent = (Student) studentTree.search(leftStudent.getName());
    currentStudent.setDebateStudentPair(null);

12025    Student rightStudent = pairStudentArray[1];
    currentStudent = (Student) studentTree.search(rightStudent.getName());
    currentStudent.setDebateStudentPair(null);

12030    return deletedPair;
}

12035    /**
     * The deleteDebatePair method removes a specific student pair from the
     * debatePairTree.
     * <p>
     * The student pair at the specified index is removed from the debatePairTree
     * and is returned by this function.
     *
     * @param index the index of the student pair in the debatePairTree
     * @return the deleted student pair
     * @throws IllegalArgumentException if the index is not within the bounds of
     * the debatePairTree
     *
     * @author Zbyněk Stara
     * @version 1.1 (Jan-10-2013)
     * @since Jan-8-2013
     */
12040    public DebateStudentPair deleteDebatePair(int index) throws
IllegalArgumentException {
        if (index >= 0 && index < debatePairTree.size()) {
            DebateStudentPair deletedPair = (DebateStudentPair)
debatePairTree.getNodeData(index);
            String deleteKey = deletedPair.getCode();
            debatePairTree.delete(deleteKey);

12045        Student[] pairStudentArray = deletedPair.getStudentArray();

12050        Student leftStudent = pairStudentArray[0];
        Student currentStudent = (Student)
studentTree.search(leftStudent.getName());
        currentStudent.setDebateStudentPair(null);

12055        Student rightStudent = pairStudentArray[1];
        currentStudent = (Student) studentTree.search(rightStudent.getName());
        currentStudent.setDebateStudentPair(null);

12060        return deletedPair;
    }

12065    /**
     * The searchDebatePair method finds a specific student.
     * <p>
     * If the student pair exists, this method finds it and returns its index in
     * the debatePairTree.
     */
12070    public int searchDebatePair(String code) throws
IllegalArgumentException {
        DebateStudentPair deletedPair = (DebateStudentPair)
debatePairTree.getNodeData(code);
        String deleteKey = deletedPair.getCode();
        debatePairTree.delete(deleteKey);

12075        Student[] pairStudentArray = deletedPair.getStudentArray();

12080        Student leftStudent = pairStudentArray[0];
        Student currentStudent = (Student)
studentTree.search(leftStudent.getName());
        currentStudent.setDebateStudentPair(null);

12085        Student rightStudent = pairStudentArray[1];
        currentStudent = (Student) studentTree.search(rightStudent.getName());
        currentStudent.setDebateStudentPair(null);

12090        return deletedPair;
    }

12095    /**
     * The searchDebatePair method finds a specific student.
     * <p>
     * If the student pair exists, this method finds it and returns its index in
     * the debatePairTree.
     */

```

```

*
* @param code code of the student pair to search for
* @return student's index in the daPairTree
* @throws IllegalArgumentException if the code is an empty string
* @throws NoSuchElementException if the code has not been found
*
* @author Zbyněk Stara
* @version 1.0 (Jan-11-2013)
* @since Jan-11-2013
*/
12095    public int searchDebatePair(String code) throws IllegalArgumentException,
NoSuchElementException {
    if (!code.equals("")) {
        Object [] debatePairTreeArray = debatePairTree.getDataArray();
        boolean studentPairFound = false;
        int studentPairIndex = -999;
        for (int i = 0; i < debatePairTreeArray.length; i++) {
            if (((DebateStudentPair)
debatePairTreeArray[i]).getCode().equals(code)) {
                studentPairFound = true;
                studentPairIndex = i;
                break;
            }
        }
        if (studentPairFound) {
            return studentPairIndex;
        }
        else throw new NoSuchElementException();
    }
    else throw new IllegalArgumentException();
}

/**
* This method returns a string summary of this school:
* name, ooStudentNumber, oiStudentNumber, isStudentNumber,
* daPairTree.size(), debatePairTree.size().
*
* @return a string with the information about the school
*
* @author Zbyněk Stara
*/
12130    @Override
public String toString() {
    return name + ":" +
12135        ooStudentNumber + ", " +
        oiStudentNumber + ", " +
        isStudentNumber + ", " +
        daPairTree.size() + ", " +
        debatePairTree.size();
}
12140
}

```

Teacher.java

```

12145 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
12150 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
12155 *
 * IDE used:
 * NetBeans 6.9.1
 */
12160 package data;

/**
 * This class takes care of the teacher entries in schools in the database.
 * Teachers are important because they are the judges for the tournaments.
 * Teachers have two attributes: name and school. This class contains methods to
 * access them and change their values.
 *
 * @author Zbyněk Stara
 */
12170 public class Teacher {
    // Attributes:
    private String name = "<No name>";
    private School school = new School();

12175    // Constructors:
    public Teacher() {

    }
12180    public Teacher(School school) {
        this.school = school;
    }
    public Teacher(String name, School school) {
        this.name = name;
        this.school = school;
    }
12185

    // Get methods:
    /**
     * This method makes accessible the name attribute.
     *
     * @return the name of the teacher
     *
     * @author Zbyněk Stara
     */
12190    public String getName() {
        return name;
    }
    /**
     * This method makes accessible the school attribute.
     *
     * @return the school this teacher belongs to
     *
     * @author Zbyněk Stara
     */
12195    public School getSchool() {
        return school;
    }

12200

    // Set methods:
    /**
     * This method allows for setting up the name of the teacher.
     *
     * @param name name of the teacher as a string
     *
     * @author Zbyněk Stara
     */
12205    public void setName(String name) {
        this.name = name;
    }
12210
12215

```

```
12220    }
12221    /**
12222     * This method allows for setting up the school of the teacher.
12223     *
12224     * @param school school this teacher belongs to
12225     *
12226     * @author Zbyněk Stara
12227     */
12228     public void setSchool(School school) {
12229         this.school = school;
12230     }
12231     /**
12232      * This method converts the teacher to string. It returns the teacher's
12233      * name.
12234      *
12235      * @return name of the teacher as a string
12236      *
12237      * @author Zbyněk Stara
12238      */
12239     @Override
12240     public String toString() {
12241         return name;
12242     }
12243 }
```

Student.java

12245

```

    /**
     * NESDA Tournament Manager 2013
     *
     * By Zbyněk Stara, 000889-045
     * International School of Prague
     * Czech Republic
     *
     * Computer used:
     * Macbook unibody aluminum late 2008
     * Mac OS X 10.6.8
     * 8 GiB of RAM
     *
     * IDE used:
     * NetBeans 6.9.1
     */

```

package data;

import java.util.*;

```

    /**
     * This class represents the student participants in the debate tournaments. It
     * has all relevant attributes and methods to work on them.
     *
     * @author Zbyněk Stara
     */

```

public class Student {

// Attributes:

private String code = "<No code>";

private String name = "<No name>";

private School school = null;

private boolean [] events = {false, false, false, false, false};

private boolean daUnpaired = false;

private boolean debateUnpaired = false;

private DAStudentPair daStudentPair = null; // with whom is the student in pair

private DebateStudentPair debateStudentPair = null;

private String daStudentPairTempString = "";
 private String debateStudentPairTempString = "";

private boolean reassignmentText = false; // whether the assignEvents button

should read "Re-assign events"

// Constructors:

public Student() {

}

public Student(School school) {

this.school = school;

}

public Student(String name, School school) {

this.name = name;

this.school = school;

}

public Student(String name, School school, String code, boolean [] events) {

this.name = name;

this.school = school;

this.code = code;

this.events[0] = events[0];
 this.events[1] = events[1];
 this.events[2] = events[2];
 this.events[3] = events[3];
 this.events[4] = events[4];
 }

public Student(String name, School school, String code, boolean oo, boolean oi,
boolean is, boolean da, boolean debate) {
 this.name = name;
 this.school = school;
 this.code = code;
 }

12320

```

events[0] = oo;
events[1] = oi;
events[2] = is;
events[3] = da;
events[4] = debate;
}
public Student(String name, School school, String code, boolean [] events, boolean
daUnpaired, boolean debateUnpaired,
String daStudentPairTempString, String debateStudentPairTempString,
boolean reassignmentText) {
    this.name = name;
    this.school = school;
    this.code = code;

this.events[0] = events[0];
this.events[1] = events[1];
this.events[2] = events[2];
this.events[3] = events[3];
this.events[4] = events[4];
}

this.daUnpaired = daUnpaired;
this.debateUnpaired = debateUnpaired;
this.daStudentPairTempString = daStudentPairTempString;
this.debateStudentPairTempString = debateStudentPairTempString;
this.reassignmentText = reassignmentText;
}

// Extra methods:
/**
 * This method takes the entries in the da/debateStudentPairTempString and
 * uses them to determine the partners of this student for the respective
 * events (if the student participates in them).
 *
 * @throws IllegalStateException if an entry in one of the tempStrings
 * does not correspond to any student with which to pair this student.
 *
 * @author Zbyněk Stara
 */
public void updateStudentPairs() throws IllegalStateException {
try {
    String daStudentPairCode = daStudentPairTempString;
    daStudentPair = school.getDAPair(daStudentPairCode);
} catch (IllegalArgumentException ex) {
    // the string is "" - nothing will be done
} catch (NoSuchElementException ex) {
    throw new IllegalStateException();
}

try {
    String debateStudentPairCode = debateStudentPairTempString;
    debateStudentPair = school.getDebatePair(debateStudentPairCode);
} catch (IllegalArgumentException ex) {
    // the string is "" - nothing will be done
} catch (NoSuchElementException ex) {
    throw new IllegalStateException();
}
}

// Get methods:
/**
 * This method allows for an access to the name of the student.
 *
 * @return name of the student (string)
 *
 * @author Zbyněk Stara
 */
public String getName() {
    return name;
}

/**
 * This method allows for an access to the school of the student.
 *
 * @return school this student belongs to
 *
 * @author Zbyněk Stara
 */
public School getSchool() {
    return school;
}

```

```

12400    }
12405    /**
     * This method allows for an access to the code of the student.
     *
     * @return code of this student (string)
     *
     * @author Zbyněk Stara
     */
12410    public String getCode() {
12411        return code;
12412    }
12415    /**
     * This method allows for an access to events of this student.
     *
     * @return a boolean array of five elements, each of which indicates whether
     * the student participates in that event (0: oo, 1: oi, 2: is, 3: da,
     * 4: debate) – if the value at a given element is true, the student
     * participates in the element
     *
     * @author Zbyněk Stara
     */
12420    public boolean [] getEvents() {
12421        boolean [] returnArray = new boolean[5];
12425        returnArray[0] = events[0] ? true : false;
12426        returnArray[1] = events[1] ? true : false;
12427        returnArray[2] = events[2] ? true : false;
12428        returnArray[3] = events[3] ? true : false;
12429        returnArray[4] = events[4] ? true : false;
12430
12431        return returnArray;
12432    }
12435    /**
     * This method allows for an access to the daUnpaired attribute of the
     * student.
     *
     * @return a boolean value reflecting the value of the daUnpaired attribute.
     *
     * @author Zbyněk Stara
     */
12440    public boolean getDAUnpaired() {
12441        return daUnpaired;
12442    }
12445    /**
     * This method allows for an access to the debateUnpaired attribute of the
     * student.
     *
     * @return a boolean value reflecting the value of the debateUnpaired
     * attribute.
     *
     * @author Zbyněk Stara
     */
12450    public boolean getDebateUnpaired() {
12451        return debateUnpaired;
12452    }
12455    /**
     * This method allows for an access to the daStudentPair of which the
     * student is a part.
     *
     * @return the daStudentPair attribute
     *
     * @author Zbyněk Stara
     */
12460    public DASTudentPair getDASTudentPair() {
12461        return daStudentPair;
12462    }
12465    /**
     * This method allows for an access to the debateStudentPair of which the
     * student is a part.
     *
     * @return the debateStudentPair attribute
     *
     * @author Zbyněk Stara
     */
12470
12475

```

```

12480      */
12480  public DebateStudentPair getDebateStudentPair() {
12480      return debateStudentPair;
12480  }
12485      /**
12485  * This method allows for an access to the reassignmentText attribute.
12485  *
12485  * @return the reassignmentText attribute
12485  *
12485  * @author Zbyněk Stara
12485  */
12485  public boolean getReassignmentText() {
12485      return reassignmentText;
12485  }
12490      // Set methods:
12490  /**
12490  * This method allows for changing the name of the student
12490  *
12490  * @param name the new name of the student
12490  *
12490  * @author Zbyněk Stara
12490  */
12490  public void setName(String name) {
12490      this.name = name;
12490  }
12500      /**
12500  * This method allows for changing the school of the student
12500  *
12500  * @param school the new school of the student
12500  *
12500  * @author Zbyněk Stara
12500  */
12500  public void setSchool(School school) {
12500      this.school = school;
12500  }
12510      /**
12510  * This method allows for changing the code of the student
12510  *
12510  * @param code the new code of the student
12510  *
12510  * @author Zbyněk Stara
12510  */
12510  public void setCode(String code) {
12510      this.code = code;
12510  }
12520      /**
12520  * This method allows setting the student's events.
12520  *
12520  * @param eventsArray a boolean array of length 5 - each element represents
12520  * the student's participation in an event (0: oo, 1: oi, 2: is, 3: da,
12520  * 4: debate)
12520  * @throws IllegalArgumentException if the supplied eventsArray does not
12520  * have 5 elements
12520  *
12520  * @author Zbyněk Stara
12520  */
12520  public void setEvents(boolean[] eventsArray) throws IllegalArgumentException {
12520      if (eventsArray.length == 5) {
12520          events[0] = eventsArray[0];
12520          events[1] = eventsArray[1];
12520          events[2] = eventsArray[2];
12520          events[3] = eventsArray[3];
12520          events[4] = eventsArray[4];
12520      } else {
12520          throw new IllegalArgumentException();
12520      }
12530      /**
12530  * This method allows setting the student's events.
12530  *
12530  * @param oo whether the student participates in original oratory
12530  * @param oi whether the student participates in oral interpretation
12530  * @param is whether the student participates in impromptu speaking
12530  */
12540      /**
12540  * This method allows setting the student's events.
12540  *
12540  * @param oo whether the student participates in original oratory
12540  * @param oi whether the student participates in oral interpretation
12540  * @param is whether the student participates in impromptu speaking
12540  */
12550      /**
12550  * This method allows setting the student's events.
12550  *
12550  * @param oo whether the student participates in original oratory
12550  * @param oi whether the student participates in oral interpretation
12550  * @param is whether the student participates in impromptu speaking
12550  */

```

```

12560      * @param da whether the student participates in duet acting
12561      * @param debate whether the student participates in debate
12562      *
12563      * @author Zbyněk Stara
12564      */
12565      public void setEvents(boolean oo, boolean oi, boolean is, boolean da, boolean
debate) {
12566          events[0] = oo;
12567          events[1] = oi;
12568          events[2] = is;
12569          events[3] = da;
12570          events[4] = debate;
12571      }
12572
12573      /**
12574      * This method allows for setting the daUnpaired attribute of the student.
12575      *
12576      * @param daUnpaired the new boolean value of the attribute
12577      *
12578      * @author Zbyněk Stara
12579      */
12580      public void setDAUnpaired(boolean daUnpaired) {
12581          this.daUnpaired = daUnpaired;
12582      }
12583
12584      /**
12585      * This method allows for setting the debateUnpaired attribute of the
* student.
12586      *
12587      * @param debateUnpaired the new boolean value of the attribute
12588      *
12589      * @author Zbyněk Stara
12590      */
12591      public void setDebateUnpaired(boolean debateUnpaired) {
12592          this.debateUnpaired = debateUnpaired;
12593      }
12594
12595      /**
12596      * This method allows for setting the student's DA student pair.
12597      *
12598      * @param daStudentPair the student's new DA student pair
12599      *
12600      * @author Zbyněk Stara
12601      */
12602      public void setDAStudentPair(DAStudentPair daStudentPair) {
12603          this.daStudentPair = daStudentPair;
12604      }
12605
12606      /**
12607      * This method allows for setting the student's debate student pair.
12608      *
12609      * @param debateStudentPair the student's new debate student pair
12610      *
12611      * @author Zbyněk Stara
12612      */
12613      public void setDebateStudentPair(DebateStudentPair debateStudentPair) {
12614          this.debateStudentPair = debateStudentPair;
12615      }
12616
12617      /**
12618      * This method allows for setting the student's reassignmentText attribute.
12619      *
12620      * @param reassignmentText the new boolean value of the attribute
12621      *
12622      * @author Zbyněk Stara
12623      */
12624      public void setReassignmentText(boolean reassignmentText) {
12625          this.reassignmentText = reassignmentText;
12626      }
12627
12628      /**
12629      * This method represents the student with a string
12630      *
12631      * @return a string with the student's code and name
12632      *
12633      * @author Zbyněk Stara
12634      */
12635      @Override
12636      public String toString() {

```

```
String returnString = code + ", " + name;
return returnString;
12640 }
```

StudentPair.java

```

12645 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
12650 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
12655 * IDE used:
 * NetBeans 6.9.1
 */

12660 package data;
12665 /**
 * This is a superclass for the specific studentPair classes. It defines the
 * attributes of a studentPair and provides methods to work on these.
 */
12670 public class StudentPair {
    protected Student student1;
    protected Student student2;
    protected String originalCode; // the name under which this is saved in the trees
(the keystring)

12675     protected StudentPair() {
    }

    protected StudentPair(Student student1, Student student2) throws
IllegalArgumentException {
        if (student1.getCode().compareTo(student2.getCode()) <= 0) {
            this.student1 = student1; // keep the alphabetical order
            this.student2 = student2;
        } else {
            this.student1 = student2; // flip them to be alphabetically ordered
            this.student2 = student1;
        }

        if (!this.student1.getCode().equals("<No code>") &&
12680 !this.student2.getCode().equals("<No code>")) {
            originalCode = this.student1.getCode() + " - " + this.student2.getCode();
        } else {
            throw new IllegalArgumentException();
        }
    }

12685 /**
 * This method allows access to the studentPair's current code.
 *
 * @return the code of this pair
 */
12690 public String getCode() {
    return this.student1.getCode() + " - " + this.student2.getCode();
}

12695 /**
 * This method allows access to the studentPair's original code. This is the
 * code under which the student pair is currently in its respective
 * pairTree.
 *
 * @return the value of the originalCode attribute of this pair
 */
12700 public String getOriginalCode() {
    return originalCode;
}

```

```
    }

12720 /**
 * This returns the student in the student pair that is other than the one
 * provided as a parameter to the method.
 *
 * @return the other student in the pair
 *
 * @author Zbyněk Stara
 */
12725 public Student getOtherStudent(Student student) {
12730     if (student.getName().equals(student1.getName())) {
12735         return student1;
     } else if (student.getName().equals(student2.getName())) {
         return student2;
     } else {
         return null;
     }
}

12740 /**
 * This method returns an array of the members of this student pair.
 *
 * @return a two-element array of the members of this student pair
 *
 * @author Zbyněk Stara
 */
12745 public Student[] getStudentArray() {
    Student[] studentArray = {student1, student2};
    return studentArray;
}

12750 /**
 * This method represents the student pair as a string
 *
 * @return a string with the code of the student pair
 *
 * @author Zbyněk Stara
 */
12755 @Override
public String toString() {
    return this.getCode();
}

12760 }
```

DASStudentPair.java

```

12765 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
12770 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
12775 *
 * IDE used:
 * NetBeans 6.9.1
 */
12780 package data;

/**
 * This is a specific extension of the StudentPair class, intended for use for
 * duet acting student pairs – for that end, it contains a specific constructor.
 *
 * @author Zbyněk Stara
 */
12785 public class DASStudentPair extends StudentPair {
    public DASStudentPair() {
12790     }

        public DASStudentPair(Student student1, Student student2) throws
IllegalArgumentException {
            super(student1, student2);

            if (super.student1.getDASStudentPair() == null &&
super.student2.getDASStudentPair() == null) {
                if (super.student1 == student1) {
                    super.student1.setDASStudentPair(this);
                    super.student2.setDASStudentPair(this);
                } else {
                    super.student1.setDASStudentPair(this);
                    super.student2.setDASStudentPair(this);
                }
            } else {
                throw new IllegalArgumentException();
            }
12810     }
}

```

DebateStudentPair.java

```

12815 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
12820 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
12825 * IDE used:
 * NetBeans 6.9.1
 */
12830 package data;
12831 /**
 * This is a specific extension of the StudentPair class, intended for use for
 * debate student pairs – for that end, it contains a specific constructor.
 *
12835 * @author Zbyněk Stara
 */
12836 public class DebateStudentPair extends StudentPair {
12837     public DebateStudentPair() {
12838
12839         public DebateStudentPair(Student student1, Student student2) throws
12840             IllegalArgumentException {
12841                 super(student1, student2);
12842
12843                 if (super.student1.getDebateStudentPair() == null &&
12844                     super.student2.getDebateStudentPair() == null) {
12845                     if (super.student1 == student1) {
12846                         super.student1.setDebateStudentPair(this);
12847                         super.student2.setDebateStudentPair(this);
12848                     } else {
12849                         super.student1.setDebateStudentPair(this);
12850                         super.student2.setDebateStudentPair(this);
12851                     }
12852                 } else {
12853                     throw new IllegalArgumentException();
12854                 }
12855             }
12856         }
12860

```

12860 Qualification.java

```

12865 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
 * IDE used:
 * NetBeans 6.9.1
 */

12870 package data;

12875 import java.util.*;

12880 /**
 * The Qualification class is the basis of the functionality of the
 * Qualification tab. It contains methods to initialize events, rounds and rooms,
 * and to create, handle and allocate entities and judges.
 *
 * @author Zbyněk Stara
 */
12885 public class Qualification {
    public class NotEnoughJudgesException extends Exception {
        public void NotEnoughJudgesException() {
            throw new RuntimeException();
        }
    }

12890    public class ImpossibleToAllocateException extends Exception {
        public void ImpossibleToAllocateException() {
            throw new RuntimeException();
        }
    }

12895    private Database database;

12900    private QualificationEvent [] eventArray = new QualificationEvent[5];
    private int eventArrayLength = 0;

12905    private BinarySearchTree judgeTree; // ordered by school names

12910    private BinarySearchTree ooEntityTree; // ordered by codes
    private BinarySearchTree oiEntityTree;
    private BinarySearchTree isEntityTree;
    private BinarySearchTree daEntityTree; // ordered by pair codes
    private BinarySearchTree debateEntityTree;

12915    public Qualification(Database database) {
        this.database = database;

        generateEntities();
    }

12920    private void generateEntities() {
        /*settingsArray[0] = ooStudentRoomLimit; // USED
        settingsArray[1] = oiStudentRoomLimit; // USED
        settingsArray[2] = isStudentRoomLimit; // USED
        settingsArray[3] = daStudentRoomLimit; // USED
        settingsArray[4] = ooFinalStudentRoomLimit;
        settingsArray[5] = oiFinalStudentRoomLimit;
        settingsArray[6] = isFinalStudentRoomLimit;
        settingsArray[7] = daFinalStudentRoomLimit;

12925        settingsArray[8] = ooStudentSchoolLimit; // CHECKED ALREADY
        settingsArray[9] = oiStudentSchoolLimit; // CHECKED ALREADY
        settingsArray[10] = isStudentSchoolLimit; // CHECKED ALREADY
        settingsArray[11] = daStudentSchoolLimit; // CHECKED ALREADY
        settingsArray[12] = debateStudentSchoolLimit; // CHECKED ALREADY
    }
}

```

```

settingsArray[13] = ooJudgeRoomLimit; // USED
settingsArray[14] = oiJudgeRoomLimit; // USED
settingsArray[15] = isJudgeRoomLimit; // USED
settingsArray[16] = daJudgeRoomLimit; // USED
settingsArray[17] = debateJudgeRoomLimit; // USED
settingsArray[18] = finalsJudgeRoomLimit; // USED

12940

settingsArray[19] = roomLimit1; // NOT USED
settingsArray[20] = roomLimit2; // NOT USED
settingsArray[21] = timeLimit1; // NOT USED
settingsArray[22] = timeLimit2; // NOT USED
settingsArray[23] = schoolNumber; // CHECKED ALREADY
settingsArray[24] = teacherSchoolNumber; // CHECKED ALREADY
settingsArray[25] = studentSchoolNumber; // CHECKED ALREADY

12945

12950

settingsArray[26] = allowCombinedEvents; // ALWAYS TRUE

12955

settingsArray[27] = combinedEvent1; // ALWAYS 2
settingsArray[28] = combinedEvent2; // ALWAYS 3*/
Object[] settingsArray = database.getSettings();

12960
judgeTree = new BinarySearchTree();
int currentID = 1;
for (int i = 0; i < database.getSchoolTreeSize(); i++) {
    School currentSchool = (School) database.getSchoolTreeNodeData(i);

    for (int j = 0; j < currentSchool.getTeacherTreeSize(); j++) {
        Teacher currentTeacher = (Teacher)
currentSchool.getTeacherTreeNodeData(j);

        Judge newJudge = new Judge(currentID, currentSchool, currentTeacher);
        judgeTree.insert(newJudge, newJudge.getID() + "");
        currentID++;
    }
}
12975
judgeTree = judgeTree.balance();

ooEntityTree = new BinarySearchTree();
for (int i = 0; i < database.getSchoolTreeSize(); i++) {
    School currentSchool = (School) database.getSchoolTreeNodeData(i);

    for (int j = 0; j < currentSchool.getStudentTreeSize(); j++) {
        Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(j);

        if (currentStudent.getEvents()[0] == true) {
            Student[] studentList = {currentStudent};
            Entity newEntity = new Entity(Event.Type.ORIGINAL_ORATORY,
currentSchool, false, studentList, currentStudent.getCode());
            ooEntityTree.insert(newEntity, newEntity.getCode());
        }
    }
}
12990
ooEntityTree = ooEntityTree.balance();

12995
oiEntityTree = new BinarySearchTree();
for (int i = 0; i < database.getSchoolTreeSize(); i++) {
    School currentSchool = (School) database.getSchoolTreeNodeData(i);

    for (int j = 0; j < currentSchool.getStudentTreeSize(); j++) {
        Student currentStudent = (Student)
currentSchool.getStudentTreeNodeData(j);

        if (currentStudent.getEvents()[1] == true) {
            Student[] studentList = {currentStudent};
            Entity newEntity = new Entity(Event.Type.ORAL_INTERPRETATION,
currentSchool, false, studentList, currentStudent.getCode());
            oiEntityTree.insert(newEntity, newEntity.getCode());
        }
    }
}
13000
oiEntityTree = oiEntityTree.balance();

13005
isEntityTree = new BinarySearchTree();

```

```

        for (int i = 0; i < database.getSchoolTreeSize(); i++) {
            School currentSchool = (School) database.getSchoolTreeNodeData(i);

            for (int j = 0; j < currentSchool.getStudentTreeSize(); j++) {
                Student currentStudent = (Student)
                currentSchool.getStudentTreeNodeData(j);

                if (currentStudent.getEvents()[2] == true) {
                    Student[] studentList = {currentStudent};
                    Entity newEntity = new Entity(Event.Type.IMPROMPTU_SPEAKING,
                    currentSchool, false, studentList, currentStudent.getCode());

                    isEntityTree.insert(newEntity, newEntity.getCode());
                }
            }
            isEntityTree = isEntityTree.balance();
        }

        daEntityTree = new BinarySearchTree();
        for (int i = 0; i < database.getSchoolTreeSize(); i++) {
            School currentSchool = (School) database.getSchoolTreeNodeData(i);

            for (int j = 0; j < currentSchool.getDAPairTreeSize(); j++) {
                DASStudentPair currentStudentPair = (DASStudentPair)
                currentSchool.getDAPairTreeNodeData(j);

                Student[] studentList = currentStudentPair.getStudentArray();
                Entity newEntity = new Entity(Event.Type.DUET_ACTING, currentSchool,
                false, studentList, currentStudentPair.getCode());

                daEntityTree.insert(newEntity, newEntity.getCode());
            }
        }
        daEntityTree = daEntityTree.balance();
    }

    debateEntityTree = new BinarySearchTree();
    for (int i = 0; i < database.getSchoolTreeSize(); i++) {
        School currentSchool = (School) database.getSchoolTreeNodeData(i);

        for (int j = 0; j < currentSchool.getDAPairTreeSize(); j++) {
            DebateStudentPair currentStudentPair = (DebateStudentPair)
            currentSchool.getDebatePairTreeNodeData(j);

            Student[] studentList = currentStudentPair.getStudentArray();
            Entity newEntity = new Entity(Event.Type.DEBATE, currentSchool, false,
            studentList, currentStudentPair.getCode());

            debateEntityTree.insert(newEntity, newEntity.getCode());
        }
    }
    debateEntityTree = debateEntityTree.balance();
}

public BinarySearchTree getJudgeTree() {
    return judgeTree;
}
public void insertJudge(Judge judge) throws IllegalArgumentException {
    judgeTree.insert(judge, judge.getID() + "");
}
public Judge [] getJudgeArray() {
    Object [] tempArray = judgeTree.getDataArray();
    Judge [] returnArray = new Judge [tempArray.length];

    for (int i = 0; i < tempArray.length; i++) {
        returnArray[i] = (Judge) tempArray[i];
    }

    return returnArray;
}
public Judge getJudge(int ID) throws NoSuchElementException {
    int judgeIndex = this.searchJudge(ID);
    Judge judge = (Judge) this.getJudgeTreeNodeData(judgeIndex);
    return judge;
}
public int searchJudge(int ID) throws NoSuchElementException {
    Object [] judgeTreeArray = judgeTree.getDataArray();
    boolean judgeFound = false;
    int judgeIndex = -999;
    for (int i = 0; i < judgeTreeArray.length; i++) {

```

```

13095         if (((Judge) judgeTreeArray[i]).getID() == ID) {
13096             judgeFound = true;
13097             judgeIndex = i;
13098             break;
13099         }
13100     }
13101     if (judgeFound) {
13102         return judgeIndex;
13103     }
13104     else throw new NoSuchElementException();
13105 }
13106 public int getJudgeTreeSize() {
13107     return judgeTree.size();
13108 }
13109 public Judge getJudgeTreeNodeData(int index) {
13110     return (Judge) judgeTree.getNodeData(index);
13111 }
13112 public void completeJudgeTree() {
13113     for (int i = 0; i < getJudgeTreeSize(); i++) {
13114         Judge currentJudge = (Judge) getJudgeTreeNodeData(i);
13115
13116         for (int j = 0; j < currentJudge.getEncounteredJudgeTempIDTreeSize(); i++)
13117         {
13118             int currentEncounteredJudgeTempID =
13119             currentJudge.getEncounteredJudgeTempID(j);
13120             Judge currentEncounteredJudge =
13121             getJudge(currentEncounteredJudgeTempID);
13122
13123             currentJudge.insertEncounteredJudge(currentEncounteredJudge);
13124         }
13125     }
13126 }
13127 public BinarySearchTree getOOEntityTree() {
13128     return ooEntityTree;
13129 }
13130 public void insertOOEntity(Entity entity) throws IllegalArgumentException {
13131     ooEntityTree.insert(entity, entity.getCode());
13132 }
13133 public Entity getOOEntity(String code) throws NoSuchElementException {
13134     int entityIndex = this.searchOOEntity(code);
13135     Entity entity = (Entity) this.getOOEntityTreeNodeData(entityIndex);
13136     return entity;
13137 }
13138 public int searchOOEntity(String code) throws NoSuchElementException {
13139     Object [] ooEntityTreeArray = ooEntityTree.getDataArray();
13140     boolean entityFound = false;
13141     int entityIndex = -999;
13142     for (int i = 0; i < ooEntityTreeArray.length; i++) {
13143         if (((Entity) ooEntityTreeArray[i]).getCode().equals(code)) {
13144             entityFound = true;
13145             entityIndex = i;
13146             break;
13147         }
13148     }
13149     if (entityFound) {
13150         return entityIndex;
13151     }
13152     else throw new NoSuchElementException();
13153 }
13154 public int getOOEntityTreeSize() {
13155     return ooEntityTree.size();
13156 }
13157 public Entity getOOEntityTreeNodeData(int index) {
13158     return (Entity) ooEntityTree.getNodeData(index);
13159 }
13160 protected void setOOEntityTreeNodeData(int index, Object data) {
13161     ooEntityTree.setNodeData(index, data);
13162 }
13163 public BinarySearchTree getOIEntityTree() {
13164     return ooEntityTree;
13165 }
13166 public void insertOIEntity(Entity entity) throws IllegalArgumentException {
13167     oiEntityTree.insert(entity, entity.getCode());
13168 }
13169 public Entity getOIEntity(String code) throws NoSuchElementException {
13170     int entityIndex = this.searchOIEntity(code);

```

```

13175     Entity entity = (Entity) this.getOIEntityTreeNodeData(entityIndex);
13176     return entity;
13177 }
13178 public int searchOIEntity(String code) throws NoSuchElementException {
13179     Object [] oiEntityTreeArray = oiEntityTree.getDataArray();
13180     boolean entityFound = false;
13181     int entityIndex = -999;
13182     for (int i = 0; i < oiEntityTreeArray.length; i++) {
13183         if (((Entity) oiEntityTreeArray[i]).getCode().equals(code)) {
13184             entityFound = true;
13185             entityIndex = i;
13186             break;
13187         }
13188     }
13189     if (entityFound) {
13190         return entityIndex;
13191     }
13192     else throw new NoSuchElementException();
13193 }
13194 public int getOIEntityTreeSize() {
13195     return oiEntityTree.size();
13196 }
13197 public Entity getOIEntityTreeNodeData(int index) {
13198     return (Entity) oiEntityTree.getNodeData(index);
13199 }
13200 protected void setOIEntityTreeNodeData(int index, Object data) {
13201     oiEntityTree.setNodeData(index, data);
13202 }
13203
13204 public BinarySearchTree getISEntityTree() {
13205     return ooEntityTree;
13206 }
13207 public void insertISEntity(Entity entity) throws IllegalArgumentException {
13208     isEntityTree.insert(entity, entity.getCode());
13209 }
13210 public Entity getISEntity(String code) throws NoSuchElementException {
13211     int entityIndex = this.searchISEntity(code);
13212     Entity entity = (Entity) this.getISEntityTreeNodeData(entityIndex);
13213     return entity;
13214 }
13215 public int searchISEntity(String code) throws NoSuchElementException {
13216     Object [] isEntityTreeArray = isEntityTree.getDataArray();
13217     boolean entityFound = false;
13218     int entityIndex = -999;
13219     for (int i = 0; i < isEntityTreeArray.length; i++) {
13220         if (((Entity) isEntityTreeArray[i]).getCode().equals(code)) {
13221             entityFound = true;
13222             entityIndex = i;
13223             break;
13224         }
13225     }
13226     if (entityFound) {
13227         return entityIndex;
13228     }
13229     else throw new NoSuchElementException();
13230 }
13231 public int getISEntityTreeSize() {
13232     return isEntityTree.size();
13233 }
13234 public Entity getISEntityTreeNodeData(int index) {
13235     return (Entity) isEntityTree.getNodeData(index);
13236 }
13237 protected void setISEntityTreeNodeData(int index, Object data) {
13238     isEntityTree.setNodeData(index, data);
13239 }
13240 public BinarySearchTree getDAEntityTree() {
13241     return ooEntityTree;
13242 }
13243 public void insertDAEntity(Entity entity) throws IllegalArgumentException {
13244     daEntityTree.insert(entity, entity.getCode());
13245 }
13246 public Entity getDAEntity(String code) throws NoSuchElementException {
13247     int entityIndex = this.searchDAEntity(code);
13248     Entity entity = (Entity) this.getDAEntityTreeNodeData(entityIndex);
13249     return entity;
13250 }
13251 public int searchDAEntity(String code) throws NoSuchElementException {
13252     Object [] daEntityTreeArray = daEntityTree.getDataArray();

```

```

        boolean entityFound = false;
        int entityIndex = -999;
13255    for (int i = 0; i < daEntityTreeArray.length; i++) {
            if (((Entity) daEntityTreeArray[i]).getCode().equals(code)) {
                entityFound = true;
                entityIndex = i;
                break;
            }
        }
        if (entityFound) {
            return entityIndex;
        }
13265    else throw new NoSuchElementException();
    }
    public int getDAEntityTreeSize() {
        return daEntityTree.size();
    }
13270    public Entity getDAEntityTreeNodeData(int index) {
        return (Entity) daEntityTree.getNodeData(index);
    }
    protected void setDAEntityTreeNodeData(int index, Object data) {
        daEntityTree.setNodeData(index, data);
    }
13275    public BinarySearchTree getDebateEntityTree() {
        return ooEntityTree;
    }
13280    public void insertDebateEntity(Entity entity) throws IllegalArgumentException {
        debateEntityTree.insert(entity, entity.getCode());
    }
    public Entity getDebateEntity(String code) throws NoSuchElementException {
        int entityIndex = this.searchDebateEntity(code);
        Entity entity = (Entity) this.getDebateEntityTreeNodeData(entityIndex);
        return entity;
    }
    public int searchDebateEntity(String code) throws NoSuchElementException {
        Object [] debateEntityTreeArray = debateEntityTree.getDataArray();
        boolean entityFound = false;
        int entityIndex = -999;
        for (int i = 0; i < debateEntityTreeArray.length; i++) {
            if (((Entity) debateEntityTreeArray[i]).getCode().equals(code)) {
                entityFound = true;
                entityIndex = i;
                break;
            }
        }
        if (entityFound) {
            return entityIndex;
        }
        else throw new NoSuchElementException();
    }
13290    public int getDebateEntityTreeSize() {
        return debateEntityTree.size();
    }
    public Entity getDebateEntityTreeNodeData(int index) {
        return (Entity) debateEntityTree.getNodeData(index);
    }
13295    protected void setDebateEntityTreeNodeData(int index, Object data) {
        debateEntityTree.setNodeData(index, data);
    }

    public void initializeEvents() throws ImpossibleToAllocateException,
1330      NotEnoughJudgesException {
        Object[] settingsArray = database.getSettings();

        // TESTING:
        if (ooEntityTree.size() >= (Integer) settingsArray[0]) {
            eventArray[eventArrayLength] = new QualificationEvent(this,
Event.Type.ORIGINAL_ORATORY, judgeTree, ooEntityTree, (Integer) settingsArray[13],
(Integer) settingsArray[0]);
        }
13310      eventArray[eventArrayLength].initializeRounds();
        if (eventArray[eventArrayLength].getNotEnoughJudges()) {
            throw new NotEnoughJudgesException();
        } else if (eventArray[eventArrayLength].getImpossibleToAllocate()) {
            throw new ImpossibleToAllocateException();
        } else {
            eventArrayLength += 1;
        }
    }
13315
13320
13325
13330

```

```

        }
    } else {
        System.out.println("Skipping OO");
}

13335 if (oiEntityTree.size() >= (Integer) settingsArray[1]) {
    eventArray[eventArrayLength] = new QualificationEvent(this,
Event.Type.ORAL_INTERPRETATION, judgeTree, oiEntityTree, (Integer) settingsArray[14],
(Integer) settingsArray[1]);

    eventArray[eventArrayLength].initializeRounds();

13345 if (eventArray[eventArrayLength].getNotEnoughJudges()) {
    throw new NotEnoughJudgesException();
} else if (eventArray[eventArrayLength].getImpossibleToAllocate()) {
    throw new ImpossibleToAllocateException();
} else {
    eventArrayLength += 1;
}
13350 } else {
    System.out.println("Skipping OI");
}

13355 if (isEntityTree.size() >= (Integer) settingsArray[2]) {
    eventArray[eventArrayLength] = new QualificationEvent(this,
Event.Type.IMPROMPTU_SPEAKING, judgeTree, isEntityTree, (Integer) settingsArray[15],
(Integer) settingsArray[2]);
}

13360 eventArray[eventArrayLength].initializeRounds();

if (eventArray[eventArrayLength].getNotEnoughJudges()) {
    throw new NotEnoughJudgesException();
} else if (eventArray[eventArrayLength].getImpossibleToAllocate()) {
    throw new ImpossibleToAllocateException();
} else {
    eventArrayLength += 1;
}
13365 } else {
    System.out.println("Skipping IS");
}

13370 if (daEntityTree.size() >= (Integer) settingsArray[3]) {
    eventArray[eventArrayLength] = new QualificationEvent(this,
Event.Type.DUET_ACTING, judgeTree, daEntityTree, (Integer) settingsArray[16],
(Integer) settingsArray[3]);
}

13375 eventArray[eventArrayLength].initializeRounds();

13380 if (eventArray[eventArrayLength].getNotEnoughJudges()) {
    throw new NotEnoughJudgesException();
} else if (eventArray[eventArrayLength].getImpossibleToAllocate()) {
    throw new ImpossibleToAllocateException();
} else {
    eventArrayLength += 1;
}
13385 } else {
    System.out.println("Skipping DA");
}
13390 if (debateEntityTree.size() >= 2) {
    eventArray[eventArrayLength] = new QualificationEvent(this,
Event.Type.DEBATE, judgeTree, debateEntityTree, (Integer) settingsArray[17], 2);
}

13395 eventArray[eventArrayLength].initializeRounds();

if (eventArray[eventArrayLength].getNotEnoughJudges()) {
    throw new NotEnoughJudgesException();
} else if (eventArray[eventArrayLength].getImpossibleToAllocate()) {
    throw new ImpossibleToAllocateException();
} else {
    eventArrayLength += 1;
}
13400 } else {
    System.out.println("Skipping Debate");
}
13405 }

public Event[] getEventArray() {
    Event [] returnEventArray = new Event[eventArrayLength];
}
13410

```

```
for (int i = 0; i < eventArrayLength; i++) {
    returnEventArray[i] = eventArray[i];
}
return returnEventArray;
}
public int getEventArrayLength() {
    return eventArrayLength;
}
public Event getEventArrayElement(int index) throws ArrayIndexOutOfBoundsException
{
    return eventArray[index];
}
public void setEvent(QualificationEvent event, int index) {
    if (eventArray[index] == null && event != null) {
        eventArrayLength += 1;
    }
    eventArray[index] = event;
}
}
```

Event.java

13435

```

    /**
     * NESDA Tournament Manager 2013
     *
     * By Zbyněk Stara, 000889-045
     * International School of Prague
     * Czech Republic
     *
     * Computer used:
     * Macbook unibody aluminum late 2008
     * Mac OS X 10.6.8
     * 8 GiB of RAM
     *
     * IDE used:
     * NetBeans 6.9.1
     */

```

package data;

```

    /**
     *
     * @author Zbyněk Stara
     */
abstract public class Event {
    public class NotEnoughJudgesException extends Exception {
        public void NotEnoughJudgesException() {
            throw new RuntimeException();
        }
    }

    public class NotEnoughEntitiesException extends Exception {
        public void NotEnoughEntitiesException() {
            throw new RuntimeException();
        }
    }

    public class ImpossibleToAllocateException extends Exception {
        public void ImpossibleToAllocateException() {
            throw new RuntimeException();
        }
    }

    public enum Type {
        ORIGINAL_ORATORY("Original Oratory", "OO"),
        ORAL_INTERPRETATION("Oral Interpretation", "OI"),
        IMPROMPTU_SPEAKING("Impromptu Speaking", "IS"),
        DUET_ACTING("Duet Acting", "DA"),
        DEBATE("Debate", "Debate"),
        UNDEFINED("Undefined", "UNDEF");
    }
}

private String eventName;
private String shorteventName;

private Type(String eventName, String shorteventName) {
    this.eventName = eventName;
    this.shorteventName = shorteventName;
}

public String geteventName() {
    return eventName;
}

public String getShorteventName() {
    return shorteventName;
}

protected final String[] ooRoundTimes = {"Day 1, 11-12", "Day 1, 16-17"};
protected final String[] oiRoundTimes = {"Day 1, 15-16", "Day 2, 09-10"};
protected final String[] isRoundTimes = {"Day 1, 10-11", "Day 1, 14-15"};
protected final String[] daRoundTimes = {"Day 1, 10-11", "Day 1, 14-15"};
protected final String[] debateRoundTimes = {"Day 1, 09-10", "Day 1, 13-14", "Day 1, 17-18"};

```

/*protected final String debateSemifinalTime = "Day 2, 08-09";

13440

13445

13450

13455

13460

13465

13470

13475

13480

13485

13490

13495

13500

13505

13510

```

protected final String ooFinalTime = "Day 2, 10-11";
protected final String oiFinalTime = "Day 2, 11-12";
protected final String isFinalTime = "Day 2, 13-14";
protected final String daFinalTime = "Day 2, 14-15";
protected final String debateFinalTime = "Day 2, 15-16";*/
13515
    public final Qualification qualification;
13520
    public final Type type;
    //private EventPair eventPair;
    private final boolean isSemifinal;
    private final boolean isFinal;
13525
    protected BinarySearchTree judgeTree;
    protected BinarySearchTree entityTree;
13530
    protected int judgesPerRoom;
    protected int entitiesPerRoom;
13535
    private boolean notEnoughJudges = false;
    private boolean impossibleToAllocate = false;
13540
    protected Round[] roundArray;
    public Event(Qualification qualification, Type type, boolean isSemifinal, boolean
isFinal, BinarySearchTree judgeTree, BinarySearchTree entityTree, int judgesPerRoom,
int entitiesPerRoom) throws IllegalArgumentException {
        this.qualification = qualification;
        this.type = type;
        this.isSemifinal = isSemifinal;
        this.isFinal = isFinal;
13545
        if (!isSemifinal && isFinal) {
            roundArray = new Round[1];
            if (this.type == Event.Type.UNDEFINED) {
                throw new IllegalArgumentException("Undefined types are not
allowed.");
        }
13550
        } else if (isSemifinal && !isFinal) {
            roundArray = new Round[1];
            if (type != Type.DEBATE) {
                throw new IllegalArgumentException("Semifinals can only be of the
Debate type.");
        }
13555
        } else if (!isSemifinal && !isFinal) {
            if (this.type == Event.Type.DEBATE) {
                roundArray = new Round[3];
13560
            for (int i = 0; i < qualification.getDebateEntityTreeSize(); i++) {
                Entity editedEntity =
qualification.getDebateEntityTreeNodeData(i);
                editedEntity.setRoundNewEncounteredJudgesArray(roundArray.length);
13565
                qualification.setDebateEntityTreeNodeData(i, editedEntity);
            }
        } else if (this.type == Event.Type.ORIGINAL_ORATORY) {
            roundArray = new Round[2];
13570
            for (int i = 0; i < qualification.getOOEntityTreeSize(); i++) {
                Entity editedEntity = qualification.getOOEntityTreeNodeData(i);
                editedEntity.setRoundNewEncounteredJudgesArray(roundArray.length);
13575
                qualification.setOOEntityTreeNodeData(i, editedEntity);
            }
        } else if (this.type == Event.Type.ORAL_INTERPRETATION) {
            roundArray = new Round[2];
13580
            for (int i = 0; i < qualification.getOIEntityTreeSize(); i++) {
                Entity editedEntity = qualification.getOIEntityTreeNodeData(i);
                editedEntity.setRoundNewEncounteredJudgesArray(roundArray.length);
13585
                qualification.setOIEntityTreeNodeData(i, editedEntity);
            }
        } else if (this.type == Event.Type.IMPROPTU_SPEAKING) {
            roundArray = new Round[2];

```

```

13590
    for (int i = 0; i < qualification.getISEntityTreeSize(); i++) {
        Entity editedEntity = qualification.getISEntityTreeNodeData(i);
        editedEntity.setRoundNewEncounteredJudgesArray(roundArray.length);
        qualification.setISEntityTreeNodeData(i, editedEntity);
    }
} else if (this.type == Event.Type.DUET_ACTING) {
    roundArray = new Round[2];
13600
    for (int i = 0; i < qualification.getDAEntityTreeSize(); i++) {
        Entity editedEntity = qualification.getDAEntityTreeNodeData(i);
        editedEntity.setRoundNewEncounteredJudgesArray(roundArray.length);
        qualification.setDAEntityTreeNodeData(i, editedEntity);
    }
} else {
    throw new IllegalArgumentException("Undefined types are not
13610 allowed.");
}
} else { // isSemifinal && isFinal
    throw new IllegalArgumentException("One event cannot be a semifinal and a
final at the same time.");
13615
}
this.judgeTree = judgeTree;
this.entityTree = entityTree;

13620
    this.judgesPerRoom = judgesPerRoom;
    this.entitiesPerRoom = entitiesPerRoom;
}

13625
/*public void setEventPair(EventPair eventPair) {
    this.eventPair = eventPair;
}*/
```

13630

```

    public String getTypeEventName() {
        return type.getEventName();
    }

    /*public EventPair getEventPair() {
        return eventPair;
    }*/
13635
    public boolean isSemifinal() {
        return isSemifinal;
    }

    public boolean isFinal() {
        return isFinal;
    }

    public String getShortEventName() {
        return type.getShortEventName();
    }

    public int getJudgesPerRoom() {
        return judgesPerRoom;
    }

    public int getEntitiesPerRoom() {
        return entitiesPerRoom;
    }

13655
    public Round[] getRoundArray() throws IllegalStateException {
        try {
            Round [] returnRoundArray = new Round[roundArray.length];
13660
            for (int i = 0; i < roundArray.length; i++) {
                returnRoundArray[i] = roundArray[i];
            }
            return returnRoundArray;
        } catch (NullPointerException ex) {
            throw new IllegalStateException();
        }
    }
}
```

```
13670     public int getRoundArrayLength() {
13671         return roundArray.length;
13672     }
13673
13674     public Round getRoundArrayElement(int index) throws ArrayIndexOutOfBoundsException {
13675         return roundArray[index];
13676     }
13677
13678     public void setRound(Round round, int index) {
13679         roundArray[index] = round;
13680     }
13681
13682     abstract public void allocate(
13683         Event event, BinarySearchTree judgeTree, BinarySearchTree entityTree,
13684         boolean allocatingRound, int currentRoundIndex, int currentRoomIndex,
13685         int judgesPerRoom, int entitiesPerRoom,
13686         boolean allocatingJudge, int numAllocatedJudges, int numAllocatedEntities,
13687         long currentCombination)
13688         throws IllegalStateException /*because of odd number for debate*/,
13689         NotEnoughJudgesException, ImpossibleToAllocateException;
13690
13691     public boolean getNotEnoughJudges() {
13692         return notEnoughJudges;
13693     }
13694
13695     public boolean getImpossibleToAllocate() {
13696         return impossibleToAllocate;
13697     }
13698
13699     protected void setNotEnoughJudges(boolean notEnoughJudges) {
13700         this.notEnoughJudges = notEnoughJudges;
13701     }
13702
13703     protected void setImpossibleToAllocate(boolean impossibleToAllocate) {
13704         this.impossibleToAllocate = impossibleToAllocate;
13705     }
13706
13707     @Override
13708     public String toString() {
13709         return type.name();
13710     }
13711 }
```

QualificationEvent.java

```

13715 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
 * IDE used:
 * NetBeans 6.9.1
 */
13720
13725
13730 package data;
13735 /**
 * The QualificationEvent class is responsible for qualification-event-specific
 * tasks, mainly the creation of round/room/judge & entity hierarchy. This is
 * the class that makes the integral part of the program, allocation of judges
 * and entities, possible
 *
 * @author Zbyněk Stara
 */
13740 public class QualificationEvent extends Event {
    /**
     * The AllocationWorker class is an extension of the SwingWorker<V, T> class
     * from the extension package javax.swing. Its purpose is to allow for a
     * computation-extensive part of the program to be executed in the
     * background, which is useful for the allocation algorithm.
     *
     * @author Zbyněk Stara
     */
13745
13750 private class AllocationWorker extends javax.swing.SwingWorker<Boolean, Boolean> {
    AllocationWorker () {
        }

        /**
         * This method specifies what will be done in the background after the
         * allocation worker is executed
         *
         * @return true in any case
         * @throws Exception if the allocation worker is unable to complete the
         * task
         *
         * @author Zbyněk Stara
         */
13755
13760
13765 @Override
    public Boolean doInBackground() throws Exception {
        try {
            allocate(thisEvent, judgeTree, entityTree, true, 0, 0, judgesPerRoom,
entitiesPerRoom, false, 0, 0, 1);
        } catch (NotEnoughJudgesException ex) {
            thisEvent.setNotEnoughJudges(true);
        } catch (ImpossibleToAllocateException ex) {
            thisEvent.setImpossibleToAllocate(true);
        }
        return true;
    }

    /**
     * This method specifies what the program will do after the task in
     * the doInBackground() method finishes.
     *
     * @author Zbyněk Stara
     */
13770
13775
13780
13785 @Override
    protected void done() {
        // the allocation info dialog will be reset end hidden
        gui.MainGUI.getAllocationInfoTextField().setText("");
        gui.MainGUI.getAllocationInfoProgressBar().setValue(0);
        gui.MainGUI.getAllocationInfoDialog().setVisible(false);
    }
}

```

```

13790         }
13791     }
13792
13793     private long maxCombinations; // maximum number of combination for the current
13794     numbers of judges/entities
13795
13796     private java.text.NumberFormat combinationsFormat; // number format of the
13797     maxCombinations number (to have max and current the same length)
13798
13799     private final QualificationEvent thisEvent = this; // this qualification, used for
13800     reference from the allocation worker
13801
13802     private AllocationWorker aw; // allocation worker to handle the allocation in the
13803     background
13804
13805     public QualificationEvent(Qualification qualification, Type eventType,
13806     BinarySearchTree judgeTree, BinarySearchTree entityTree, int judgesPerRoom, int
13807     entitiesPerRoom) {
13808         super(qualification, eventType, false, false, judgeTree, entityTree,
13809         judgesPerRoom, entitiesPerRoom);
13810     }
13811
13812     public void initializeRounds() {
13813         // make a new allocation worker that will do the allocation in the background
13814         aw = new AllocationWorker();
13815
13816         // this listener is added to update the numbers reported by the progress
13817         dialog
13818         aw.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
13819             public void propertyChange(java.beans.PropertyChangeEvent evt) {
13820                 if ("update".equals(evt.getPropertyName())) {
13821                     thisEvent.updateAllocationInfoNumbers((Integer) evt.getOldValue(),
13822                     (Long) evt.getNewValue());
13823                 }
13824             }
13825         });
13826
13827         // this is to execute the allocation in the background
13828         aw.execute();
13829
13830         // set up the allocation info dialog
13831         gui.MainGUI.getAllocationInfoTextField().setText("");
13832         gui.MainGUI.getAllocationInfoProgressBar().setValue(0);
13833         gui.MainGUI.getAllocationInfoDialog().setVisible(true);
13834     }
13835
13836 /**
13837 * This method is the heart of the program, it is responsible for allocating
13838 * judges and entities to rounds and rooms so that:<br />
13839 * (1) no two judges from the same school are in one room,<br />
13840 * (2) no two judges who have already met in one round can meet in another,<br />
13841 * (3) no entity can meet a judge from the same school,<br />
13842 * (4) no two entities from the same school are in one room.
13843 *
13844 * @param event this event
13845 * @param judgeTree BinarySearchTree of judges to be used for allocation
13846 * @param entityTree BinarySearchTree of entities to be used for allocation
13847 * @param allocatingRound boolean value indicating if a round is being
13848 * allocated in this iteration
13849 * @param currentRoundIndex int value with the index of the current round
13850 * @param currentRoomIndex int value with the index of the current room
13851 * @param judgesPerRoom int value with maximum number of judges per room
13852 * @param entitiesPerRoom int value with maximum number of entities per room
13853 * @param allocatingJudge boolean value indicating if a judge is being
13854 * allocated in this iteration
13855 * @param numAllocatedJudges int value with the number of judges currently
13856 * allocated
13857 * @param numAllocatedEntities int value with the number of entities
13858 * currently allocated
13859 * @param currentCombination int value with the number of the current
13860 * combination
13861 * @throws NotEnoughJudgesException if it is found during allocation that
13862 * there is not enough judges in the judgeTree to fill the judgesPerRoom
13863 * limits of the rooms
13864 * @throws ImpossibleToAllocateException if no solution to the allocation
13865 * problem was found. This usually means that there is not enough variety
13866 * in terms of the schools from which the judges and entities come. However,
13867 * it is possible that no solution was found because of the limited number
13868 * of tries. That is why the exception text should suggest to the user to

```

```

13870      * try again.
13871      *
13872      * @author Zbyněk Stara
13873      */
13874      @Override
13875      public void allocate(Event event, BinarySearchTree judgeTree, BinarySearchTree
entityTree,
13876          boolean allocatingRound, int currentRoundIndex, int currentRoomIndex,
13877          int judgesPerRoom, int entitiesPerRoom,
13878          boolean allocatingJudge, int numAllocatedJudges, int numAllocatedEntities,
13879          long currentCombination)
13880          throws NotEnoughJudgesException, ImpossibleToAllocateException {
13881      if (allocatingRound && currentRoundIndex < roundArray.length) { // if the
13882          allocation of judges and entities has ended for the previous round and other round
13883          still needs allocation
13884          // set up the round name
13885          String roundName = "<Unnamed>";
13886          switch (event.type) {
13887              case ORIGINAL_ORATORY:
13888                  roundName = event.getShortEventName() + " R" + (currentRoundIndex
+ 1) + ":" + event.ooRoundTimes[currentRoundIndex];
13889                  break;
13890              case ORAL_INTERPRETATION:
13891                  roundName = event.getShortEventName() + " R" + (currentRoundIndex
+ 1) + ":" + event.oiRoundTimes[currentRoundIndex];
13892                  break;
13893              case IMPROMPTU_SPEAKING:
13894                  roundName = event.getShortEventName() + " R" + (currentRoundIndex
+ 1) + ":" + event.isRoundTimes[currentRoundIndex];
13895                  break;
13896              case DUET_ACTING:
13897                  roundName = event.getShortEventName() + " R" + (currentRoundIndex
+ 1) + ":" + event.daRoundTimes[currentRoundIndex];
13898                  break;
13899              case DEBATE:
13900                  if (currentRoundIndex != 1) {
13901                      roundName = event.getShortEventName() + " R" +
13902                      (currentRoundIndex + 1) + ":" + event.debateRoundTimes[currentRoundIndex];
13903                  } else {
13904                      roundName = "Impromptu " + event.getShortEventName() + ":" +
13905                      event.debateRoundTimes[currentRoundIndex];
13906                  }
13907                  break;
13908          }
13909
13910      // make a new round and add it to this event's round array, then set up
13911      // the room array of the round
13912      Round newRound = new Round(event, currentRoundIndex, roundName, judgeTree,
entityTree, judgesPerRoom, entitiesPerRoom);
13913      roundArray[currentRoundIndex] = newRound;
13914      roundArray[currentRoundIndex].initializeRooms();
13915
13916      // determine the maximum number of combinations for this organization of
judges and entities
13917      long judgeCombinations =
13918      ((roundArray[currentRoundIndex].getRoomArrayLength() * judgesPerRoom) <
13919      judgeTree.size()) ?
13920          (long) Math.pow(2, (double)
13921          (roundArray[currentRoundIndex].getRoomArrayLength() * judgesPerRoom)) : // If there is
more judges than available places
13922          (long) Math.pow(2, (double)
13923          ((roundArray[currentRoundIndex].getRoomArrayLength() * judgesPerRoom) - 1)); // If
there is as many judges as there are places
13924          long entityCombinations = (long) Math.pow(2, (double) (entityTree.size() -
1));
13925          maxCombinations = (judgeCombinations * entityCombinations);
13926
13927      // determine how many digits there are in the maxCombinations number and
13928      // save as combinationsFormat
13929      // (the combinationsFormat will be referred to to have the
13930      // currentCombination written with correct number of leading zeros)
13931      int maxCombinationsDigits = (int) Math.floor(Math.log10((double)
13932      maxCombinations));
13933      String maxCombinationsFormatString = "";
13934      for (int i = 0; i <= maxCombinationsDigits; i++) {
13935          maxCombinationsFormatString += "0";
13936      }
13937      combinationsFormat = new
13938      java.text.DecimalFormat(maxCombinationsFormatString);

```

```

13950           // change the numbers in the allocation info dialog
               aw.firePropertyChange("update", currentRoundIndex, currentCombination); //
abuse of the old- and newValue notation here, used to pass both needed arguments to
the update method

13955           // start the allocation proper of this round
               allocate(event, judgeTree, entityTree, false, currentRoundIndex, 0,
judgesPerRoom, entitiesPerRoom, true, 0, 0, currentCombination);
               } else if (allocatingRound && currentRoundIndex >= roundArray.length) { // if
allocation of judges and entities has ended in previous round and there is no other
round to allocate
               // finish the recursion
               } else if (allocatingJudge) { // if we are allocating a judge in this iteration
of the recursive allocate() method
               // set up the do-while try loop
               boolean judgeAllocationSuccessful = false;
               int judgeAllocationTryCounter = 0;
               BinarySearchTree triedJudges = new BinarySearchTree();
               do {
                   // increment the current try (maximum is 2)
                   judgeAllocationTryCounter += 1;

13960               // set the current combination
               // every new try of a judge beyond the first will increase the current
combination by 2^judges_left_to_allocate * 2^(entities_to_allocate-1)
               int judgesLeft = (roundArray[currentRoundIndex].getRoomArrayLength() *
judgesPerRoom) - numAllocatedJudges;
               if (judgeTree.size() ==
roundArray[currentRoundIndex].getRoomArrayLength() * judgesPerRoom) judgesLeft -= 1;
               // if the number of judges equals the number of places, subtract the last multiplier
               currentCombination += (judgeAllocationTryCounter - 1) * ((long)
13970             Math.pow(2, entityTree.size() - 1)) * ((long) Math.pow(2, judgesLeft));

               // send an update to the progress dialog
               aw.firePropertyChange("update", currentRoundIndex,
currentCombination); // abuse of notation here: old- and newValue used to send
different attributes to the update function

13975           // try to allocate a new randomly chosen judge
               Judge randomJudge = null;
               try {
                   // get a new random judge
                   randomJudge = roundArray[currentRoundIndex].getRandomFreeJudge();
                   if (randomJudge == null) { // if no judge was accessed
                       // there are no more judges to use
                       // cannot continue with allocation
                       throw new NotEnoughJudgesException();
                   } else if (!triedJudges.contains(randomJudge.getID() + ""))
// if the judge was not tried yet
                       // insert the judge to tried judges
                       triedJudges.insert(randomJudge, randomJudge.getID() + "");

14000               // add the judge to current room in current round, remove the
judge from free judge tree, increment number of allocated judges

14005           roundArray[currentRoundIndex].getRoomArrayElement(currentRoomIndex).allocateJudge(rand
omJudge); // throws IllegalArgumentException if it cannot be done (encountered or same
school)
               roundArray[currentRoundIndex].removeFreeJudge(randomJudge);
               numAllocatedJudges += 1;
               } else { // if the judge was already tried
                   // decrement the try counter (this try did not count)
                   judgeAllocationTryCounter -= 1;
                   continue;
               }

14010           // increment the room index (loop around back to zero if
necessary)
               int newRoomIndex = currentRoomIndex;
               newRoomIndex += 1;
               if (newRoomIndex >=
14020           roundArray[currentRoundIndex].getRoomArrayLength()) {
                   newRoomIndex = 0;
               }

               // determine what action will be done in the next iteration

```

```

14025          if (numAllocatedJudges < (judgesPerRoom *
roundArray[currentRoundIndex].getRoomArrayLength())) { // if number of allocated
judges is lower than the number that needs to be allocated
                                // allocate another judge in the next iteration (in the next
room)
14030          allocate(event, judgeTree, entityTree, false,
currentRoundIndex, newRoomIndex, judgesPerRoom, entitiesPerRoom, true,
numAllocatedJudges, 0, currentCombination);
                judgeAllocationSuccessful = true;
            } else { // if the number of allocated judges is exactly what it
14035 needs to be
                // allocate new entity in the recursion
                allocate(event, judgeTree, entityTree, false,
currentRoundIndex, newRoomIndex, judgesPerRoom, entitiesPerRoom, false,
numAllocatedJudges, 0, currentCombination);
                judgeAllocationSuccessful = true;
            }
        } catch (IllegalArgumentException ex) {
            // caused by a bad allocation of this judge
            // will try the allocation again
            continue;
        } catch (ImpossibleToAllocateException ex) {
            // caused by two bad allocation attempts in the upper iteration of
the algorithm
            // remove the current judge from the room and add him back among
14050 the free judges (plus decrement the number of allocated judges)

roundArray[currentRoundIndex].getRoomArrayElement(currentRoomIndex).removeJudge(random
Judge);
14055          roundArray[currentRoundIndex].addFreeJudge(randomJudge);
            numAllocatedJudges -= 1;
        }
    } while (!judgeAllocationSuccessful && triedJudges.size() <
roundArray[currentRoundIndex].getFreeJudgeTreeSize() && judgeAllocationTryCounter <
2);
14060          // break the loop if: judge was successfully allocated, number of tried
judges is equal to the number of free judges, or there were already two tries to
allocate the judge

14065          if (!judgeAllocationSuccessful) { // if the allocation was not successful
            // throw a new exception saying so to the lower iteration of the
algorithm
            throw new ImpossibleToAllocateException();
        }
    } else { // if allocating an entity
        // set up the do-while try loop
        boolean entityAllocationSuccessful = false;
        int entityAllocationTryCounter = 0;
        BinarySearchTree triedEntities = new BinarySearchTree();
        do {
            // increment the entity try counter (maximum is two)
            entityAllocationTryCounter += 1;

            // set the current combination
            // every new try of an entity beyond the first will increase the
14080 current combination by 2^(entities_left_to_allocate-1)
            currentCombination += (entityAllocationTryCounter - 1) * ((long)
Math.pow(2, entityTree.size() - numAllocatedEntities - 1));

            // send an update to the progress dialog
            aw.firePropertyChange("update", currentRoundIndex,
currentCombination); // abuse of the old- and newValue notation here, used to pass
both needed arguments to the update method

14085            // try to allocate a new reandomly chosen entity
            Entity randomEntity = null;
            try {
                // allocate new entity
                randomEntity =
roundArray[currentRoundIndex].getRandomFreeEntity();
                if (!triedEntities.contains(randomEntity.getCode())) { // if this
entity was not tried yet
                    // insert this entity to the tried entities
                    triedEntities.insert(randomEntity, randomEntity.getCode());
14100                    // add the entity to current room in current round, remove the
entity from free entity tree, increment number of allocated entities

                    roundArray[currentRoundIndex].getRoomArrayElement(currentRoomIndex).allocateEntity(ran

```

```

14105    domEntity); // throws IllegalArgumentException if it cannot be done (encountered or
           same school)
           roundArray[currentRoundIndex].removeFreeEntity(randomEntity);
           numAllocatedEntities += 1;
       } else { // if the entity was already tried
           // decrement the try counter (this try did not count)
           entityAllocationTryCounter -= 1;
           continue;
       }

       // increment the room index (loop around back to zero if
14115   necessary)
       int newRoomIndex = currentRoomIndex;
       newRoomIndex += 1;
       if (newRoomIndex >=
14120   roundArray[currentRoundIndex].getRoomArrayLength()) {
           newRoomIndex = 0;
       }

       // determine what action needs to be done in the next iteration of
the algorithm
14125   if (numAllocatedEntities < entityTree.size()) { // if not all
entities were allocated yet
           // allocate a new random entity in the next iteration of the
algorithm
           allocate(event, judgeTree, entityTree, false,
currentRoundIndex, newRoomIndex, judgesPerRoom, entitiesPerRoom, false,
numAllocatedJudges, numAllocatedEntities, currentCombination);
           entityAllocationSuccessful = true;
       } else { // if all the entities were successfully allocated
           // allocate new round in the next iteration of the algorithm
           allocate(event, judgeTree, entityTree, true, currentRoundIndex
+ 1, currentRoomIndex, judgesPerRoom, entitiesPerRoom, false, numAllocatedJudges,
numAllocatedEntities, currentCombination);
           entityAllocationSuccessful = true;
       }
   } catch (IllegalArgumentException ex) {
       // caused by a bad allocation of this entity
       // will try the allocation again
       continue;
   } catch (ImpossibleToAllocateException ex) {
       // caused by two bad allocation attempts in the upper iteration of
the algorithm
       // remove the current entity from the room and add it back among
       // the free entites (plus decrement the number of allocated entities)
14150   roundArray[currentRoundIndex].getRoomArrayElement(currentRoomIndex).removeEntity(rando
mEntity);
       roundArray[currentRoundIndex].addFreeEntity(randomEntity);
       numAllocatedEntities -= 1;
   }
} while (!entityAllocationSuccessful && triedEntities.size() <
roundArray[currentRoundIndex].getFreeEntityTreeSize() && entityAllocationTryCounter <
2);
// break the loop if: judge was successfully allocated, number of tried
judges is equal to the number of free judges, or there were already two tries to
allocate the judge

if (!entityAllocationSuccessful) { // if the allocation was not successful
   // throw a new exception saying so to the lower iteration of the
algorithm
   throw new ImpossibleToAllocateException();
}
}

/**
 * This method is responsible for the actual updating of the
 * allocationInfoDialog to show the progress in the allocation to the user.
 *
 * @param currentRoundIndex int with the index of the current round
 * @param currentCombination long with the current combination explored by
 * the algorithm
 *
 * @author Zbyněk Stara
 */
14180 private void updateAllocationInfoNumbers(int currentRoundIndex, long
currentCombination) {
   // format the max- and currentCmbination number strings

```

```
String maxCombinationsString = combinationsFormat.format(maxCombinations);
14185 String currentCombinationString =
combinationsFormat.format(currentCombination);

    // update the dialog
    gui.MainGUI.getAllocationInfoTextField().setText(this.getShortEventName() +
currentRoundIndex + ":" + currentCombinationString + "/" + maxCombinationsString);
14190    gui.MainGUI.getAllocationInfoProgressBar().setValue((int) ((double)
currentCombination / (double) maxCombinations) * 100));
}
}
```

Round.java

14195

```

    /**
     * NESDA Tournament Manager 2013
     *
     * By Zbyněk Stara, 000889-045
     * International School of Prague
     * Czech Republic
     *
     * Computer used:
     * Macbook unibody aluminum late 2008
     * Mac OS X 10.6.8
     * 8 GiB of RAM
     *
     * IDE used:
     * NetBeans 6.9.1
     */

```

```

package data;

import java.util.*;

/**
 *
 * @author Zbyněk Stara
 */
14220 public class Round {
    public class NotEnoughJudgesException extends Exception {
        public void NotEnoughJudgesException() {
            throw new RuntimeException();
        }
    }

    public class NotEnoughEntitiesException extends Exception {
        public void NotEnoughEntitiesException() {
            throw new RuntimeException();
        }
    }

    public class ImpossibleToAllocateException extends Exception {
        public void ImpossibleToAllocateException() {
            throw new RuntimeException();
        }
    }

    private String name;
    private final Event event;
    private final int currentRoundIndex;
14245    private BinarySearchTree judgeTree; // backup - the judgeTree which gets passed
    down from Qualification
    private BinarySearchTree entityTree; // backup - the entityTree which gets passed
    down from Qualification

    private final int judgesPerRoom;
    private final int entitiesPerRoom;

    private BinarySearchTree freeJudgeTree = new BinarySearchTree(); // new tree
    private BinarySearchTree freeEntityTree = new BinarySearchTree(); // new tree
14255    private Room[] roomArray;
    //private int[] entitiesInRoom;

    public Round(Event event, int currentRoundIndex, String name, BinarySearchTree
    judgeTree, BinarySearchTree entityTree, int judgesPerRoom, int entitiesPerRoom) {
        this.event = event;

        this.currentRoundIndex = currentRoundIndex;
14265        this.name = name;

        this.judgeTree = judgeTree;
        this.entityTree = entityTree;

14270        this.judgesPerRoom = judgesPerRoom;
    }
}

```

```

        this.entitiesPerRoom = entitiesPerRoom;

        // construct freeJudgeTree as duplicate of judgeTree
        Object[] judgeArray = judgeTree.getDataArray();
        for (int i = 0; i < judgeArray.length; i++) {
            Judge currentJudge = (Judge) judgeArray[i];

            freeJudgeTree.insert(currentJudge, currentJudge.getID() + "");
        }
        freeJudgeTree = freeJudgeTree.balance();

        // construct freeEntityTree as duplicate of entityTree
        Object[] entityArray = entityTree.getDataArray();
        for (int i = 0; i < entityArray.length; i++) {
            Entity currentEntity = (Entity) entityArray[i];

            freeEntityTree.insert(currentEntity, currentEntity.getCode() + "");
        }
        freeEntityTree = freeEntityTree.balance();

        // construct roomArray
        int roomArraySize;
        if (event.type == Event.Type.DEBATE && !evenNumber(entityTree.size())) {
            throw new IllegalStateException("Qualification cannot be constructed with
14295 an odd number of debate pairs");
        } else if (event.type == Event.Type.DEBATE && evenNumber(entityTree.size())) {
            roomArraySize = entityTree.size() / 2;
        } else {
            roomArraySize = (int) Math.ceil((double) entityTree.size()) / ((double)
14300 entitiesPerRoom));
        }
        roomArray = new Room[roomArraySize];

        // predetermine numbers of entities more or less evenly in the rooms (if there
14305 is 13 entities, have rooms with 5,4,4 rather than 6,6,1)
/*entitiesInRoom = new int[roomArraySize];
for (int i = 0; i < entitiesInRoom.length; i++) {
    entitiesInRoom[i] = 0;
}
14310
int entitiesLeft = entityTree.size();
int currentRoomIndex = 0;
while (entitiesLeft > 0) {
    entitiesInRoom[currentRoomIndex] += 1;
    entitiesLeft -= 1;
14315
    currentRoomIndex += 1;
    if (currentRoomIndex >= entitiesInRoom.length) {
        currentRoomIndex = 0;
    }
}
14320 */
public int getCurrentRoundIndex() {
    return this.currentRoundIndex;
}

protected void initializeRooms() {
    for (int i = 0; i < roomArray.length; i++) {
        String roomName = "Room " + (i + 1);

        roomArray[i] = new Room(this, roomName, judgesPerRoom, entitiesPerRoom);
    }
}
14335
private boolean evenNumber(int number) {
    double doubleNumber = (double) number;

    double doubleNumberDividedBy2 = doubleNumber / 2;
    int intNumberDividedBy2 = (int) doubleNumberDividedBy2;

    if (doubleNumberDividedBy2 == intNumberDividedBy2) return true;
    else return false;
}
14340
protected Judge getRandomFreeJudge() {
    if (freeJudgeTree.isEmpty()) {
        return null;
    } else {

```

```

14350     double randomDouble = Math.random() * (double) freeJudgeTree.size();
14351     int randomIndex = (int) (Math.floor(randomDouble));
14352     return (Judge) freeJudgeTree.getNodeData(randomIndex);
14353 }
14354
14355 protected Entity getRandomFreeEntity() {
14356     if (freeEntityTree.isEmpty()) {
14357         return null;
14358     } else {
14359         double randomDouble = Math.random() * (double) freeEntityTree.size();
14360         int randomIndex = (int) (Math.floor(randomDouble));
14361         return (Entity) freeEntityTree.getNodeData(randomIndex);
14362     }
14363 }
14364
14365 public void addFreeJudge(Judge judge) throws IllegalArgumentException {
14366     if (!freeJudgeTree.contains(judge.getID() + "")) {
14367         freeJudgeTree.insert(judge, judge.getID() + "");
14368     } else {
14369         throw new IllegalArgumentException("Provided judge is already in the
14370         freeJudgeTree.");
14371     }
14372 }
14373
14374 protected void addFreeEntity(Entity entity) throws IllegalArgumentException {
14375     if (!freeEntityTree.contains(entity.getCode() + ""))
14376         freeEntityTree.insert(entity, entity.getCode() + "");
14377     } else {
14378         throw new IllegalArgumentException("Provided entity is already in the
14379         freeEntityTree.");
14380     }
14381 }
14382
14383 protected Judge removeFreeJudge(Judge judge) throws NoSuchElementException {
14384     if (freeJudgeTree.contains(judge.getID() + ""))
14385         return (Judge) freeJudgeTree.delete(judge.getID() + "");
14386     } else {
14387         throw new NoSuchElementException("Free judge to be removed was not
14388         found.");
14389     }
14390 }
14391
14392 protected Entity removeFreeEntity(Entity entity) throws NoSuchElementException {
14393     if (freeEntityTree.contains(entity.getCode()))
14394         return (Entity) freeEntityTree.delete(entity.getCode() + "");
14395     } else {
14396         throw new NoSuchElementException("Free entity to be removed was not
14397         found.");
14398     }
14399 }
14400
14401 public String getName() {
14402     return name;
14403 }
14404
14405 public void setName(String name) {
14406     this.name = name;
14407 }
14408
14409 public int getJudgesPerRoom() {
14410     return judgesPerRoom;
14411 }
14412
14413 public int getFreeJudgeTreeSize() {
14414     return freeJudgeTree.size();
14415 }
14416
14417 public Judge getFreeJudgeTreeNodeData(int index) {
14418     return (Judge) freeJudgeTree.getNodeData(index);
14419 }
14420
14421 public int getFreeEntityTreeSize() {
14422     return freeEntityTree.size();
14423 }
14424
14425 public int getRoomArrayLength() {
14426     return roomArray.length;
14427 }

```

```
14430     public Room[] getRoomArray() {
14431         Room [] returnRoomArray = new Room[roomArray.length];
14432
14433         for (int i = 0; i < roomArray.length; i++) {
14434             returnRoomArray[i] = roomArray[i];
14435         }
14436
14437         return returnRoomArray;
14438     }
14439
14440     public Room getRoomArrayElement(int index) throws ArrayIndexOutOfBoundsException {
14441         return roomArray[index];
14442     }
14443
14444     public void setRoom(Room room, int index) {
14445         roomArray[index] = room;
14446     }
14447
14448     public int[] searchStudentCode(String code) throws IllegalArgumentException,
14449     NoSuchElementException {
14450         if (!code.equals("")) {
14451             for (int i = 0; i < getRoomArrayLength(); i++) {
14452                 Room currentRoom = getRoomArrayElement(i);
14453
14454                 try {
14455                     int entityIndex = currentRoom.searchEntity(code);
14456
14457                     return newArray = {i, entityIndex};
14458
14459                 } catch (NoSuchElementException ex) {
14460                     continue;
14461                 }
14462             }
14463             throw new NoSuchElementException();
14464         } else throw new IllegalArgumentException();
14465     }
14466
14467     @Override
14468     public String toString() {
14469         return name;
14470     }
14471 }
```

Room.java

```

14475 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
 * IDE used:
 * NetBeans 6.9.1
 */
14490 package data;
import java.util.*;
14495 /**
 *
 * @author Zbyněk Stara
 */
14500 public class Room {
    private final Round round;
    private String name;
    private final int judgesLimit;
    private final int entitiesLimit;
    private BinarySearchTree judgeTree;
    private BinarySearchTree entityTree;
14510 public Room(Round round, String name, int judgesLimit, int entitiesLimit) {
    this.round = round;
    this.name = name;
    this.judgesLimit = judgesLimit;
    this.entitiesLimit = entitiesLimit;
    judgeTree = new BinarySearchTree();
    entityTree = new BinarySearchTree();
}
14525     public void allocateJudge(Judge judge) throws IllegalArgumentException { // this
version is strict about judges not judging together
        if (judgeTree.contains(judge.getID() + "")) {
            throw new IllegalArgumentException("Judge is already allocated to this
room.");
        } else {
            Object [] judgeArray = judgeTree.getDataArray();
            for (int i = 0; i < judgeArray.length; i++) {
                Judge currentJudge = (Judge) judgeArray[i];
                if (judge.getSchoolName().equals(currentJudge.getSchoolName())) {
                    throw new IllegalArgumentException("Judge from the same school is
already allocated to the room.");
                } else {
                    continue;
                }
            }
        }
14540     judgeTree.insert(judge, judge.getID() + "");
        for (int i = 0; i < judgeTree.size(); i++) {
            Judge currentJudge = (Judge) judgeTree.getNodeData(i);
            for (int j = 0; j < judgeTree.size(); j++) {
                Judge currentEncounteredJudge = (Judge) judgeTree.getNodeData(j);
                try {
                    currentJudge.addEncounteredJudge(currentEncounteredJudge);
                }
            }
        }
    }
14545

```

```

14550             } catch (IllegalArgumentException ex) {
14551                 continue; // this judge was already encountered by the current
judge
14552             }
14553         }
14554     }
14555 }
14556
14557     public Judge removeJudge(Judge judge) throws NoSuchElementException {
14558         if (!judgeTree.contains(judge.getID() + ""))
14559             throw new NoSuchElementException("The requested judge was not found in
this room.");
14560         } else {
14561             return (Judge) judgeTree.delete(judge.getID() + "");
14562         }
14563
14564         /*if (!judgeTree.contains(judge.getID() + ""))
14565             throw new NoSuchElementException("The requested judge was not found in
this room.");
14566         */ else {
14567             Judge deletedJudge = (Judge) judgeTree.delete(judge.getID() + "");
14568
14569             for (int i = 0; i < deletedJudge.getEncounteredJudgeTreeSize(); i++) {
14570                 Judge currentEncounteredJudge =
14571                     deletedJudge.getEncounteredJudgeTreeNodeData(i);
14572
14573                 currentEncounteredJudge.removeEncounteredJudge(deletedJudge);
14574             }
14575
14576             return deletedJudge;
14577         }*/
14578     }
14579
14580     public void allocateEntity(Entity entity) throws IllegalArgumentException {
14581         if (entityTree.contains(entity.getCode()))
14582             throw new IllegalArgumentException("Entity is already allocated to this
room.");
14583         } else {
14584             Object[] judgeArray = judgeTree.getDataArray();
14585             for (int i = 0; i < judgeArray.length; i++) {
14586                 Judge currentJudge = (Judge) judgeArray[i];
14587
14588                 if (entity.getSchoolName().equals(currentJudge.getSchoolName()))
14589                     throw new IllegalArgumentException("Judge from the same school is
already allocated to the room.");
14590                 } else if (entity.isJudgeEncountered(currentJudge)) {
14591                     throw new IllegalArgumentException("Judge has already been
encountered by this entity.");
14592                 } else {
14593                     continue;
14594                 }
14595             }
14596
14597             // is this too strict? - it definitely sets the minimum school number to
14598             judgesPerRoom + studentsPerRoom
14599             Object[] entityArray = entityTree.getDataArray();
14600             for (int i = 0; i < entityArray.length; i++) {
14601                 Entity currentEntity = (Entity) entityArray[i];
14602
14603                 if (entity.getSchoolName().equals(currentEntity.getSchoolName()))
14604                     throw new IllegalArgumentException("Entity from the same school is
already allocated to the room.");
14605                 } else {
14606                     continue;
14607                 }
14608             }
14609
14610             for (int j = 0; j < judgeTree.size(); j++) {
14611                 Judge currentEncounteredJudge = (Judge) judgeTree.getNodeData(j);
14612
14613                 try {
14614                     entity.addEncounteredJudgeToRound(currentEncounteredJudge,
round.getCurrentRoundIndex());
14615                 } catch (IllegalArgumentException ex) {
14616                     continue;
14617                 }
14618             }

```

```

14630           entityTree.insert(entity, entity.getCode());
14631       }
14632
14633       public Entity removeEntity(Entity entity) throws NoSuchElementException {
14634           if (!entityTree.contains(entity.getCode())) {
14635               throw new NoSuchElementException("The requested entity was not found in
this room.");
14636           } else {
14637               Entity deletedEntity = (Entity) entityTree.delete(entity.getCode());
14638           deletedEntity.eraseEncounteredJudgeTreeFromRound(round.getCurrentRoundIndex());
14639
14640           return deletedEntity;
14641       }
14642
14643       public String getName() {
14644           return name;
14645       }
14646
14647       public void setName(String name) {
14648           this.name = name;
14649       }
14650
14651       public int getJudgesLimit() {
14652           return judgesLimit;
14653       }
14654
14655       public int getEntitiesLimit() {
14656           return entitiesLimit;
14657       }
14658
14659       public int getJudgeTreeSize() {
14660           return judgeTree.size();
14661       }
14662
14663       public int getEntityTreeSize() {
14664           return entityTree.size();
14665       }
14666
14667       public Judge [] getJudgeArray() throws ArrayIndexOutOfBoundsException {
14668           Object [] tempArray = judgeTree.getDataArray();
14669           Judge [] returnArray = new Judge[tempArray.length];
14670
14671           for (int i = 0; i < tempArray.length; i++) {
14672               returnArray[i] = (Judge) tempArray[i];
14673           }
14674
14675           return returnArray;
14676       }
14677
14678       public Object getJudgeTreeNodeData(int index) {
14679           return judgeTree.getNodeData(index);
14680       }
14681
14682       public Entity [] getEntityArray() throws ArrayIndexOutOfBoundsException {
14683           Object [] tempArray = entityTree.getDataArray();
14684           Entity [] returnArray = new Entity[tempArray.length];
14685
14686           for (int i = 0; i < tempArray.length; i++) {
14687               returnArray[i] = (Entity) tempArray[i];
14688           }
14689
14690           return returnArray;
14691       }
14692
14693       public Object getEntityTreeNodeData(int index) {
14694           return entityTree.getNodeData(index);
14695       }
14696
14697       public int searchEntity(String code) {
14698           Object [] entityTreeArray = entityTree.getDataArray();
14699           boolean entityFound = false;
14700           int entityIndex = -999;
14701           for (int i = 0; i < entityTreeArray.length; i++) {
14702               if (((Entity) entityTreeArray[i]).getCode().equals(code)) {
14703                   entityFound = true;
14704                   entityIndex = i;
14705               }
14706           }
14707
14708           return entityIndex;
14709       }

```

```
        break;
14710    }
    if (entityFound) {
        return entityIndex;
    }
    else throw new NoSuchElementException();
14715}
14720@Override
public String toString() {
    return name;
}
14725}
```

Judge.java

```

14725 /**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
14730 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
14735 *
 * IDE used:
 * NetBeans 6.9.1
 */
14740 package data;
14741 import java.util.*;
14742 /**
14743 *
14744 * @author Zbyněk Stara
14745 */
14746 public class Judge {
14747     private final int ID;
14748     private final School school;
14749     private final Teacher teacher;
14750
14751     private BinarySearchTree encounteredJudgeTree = new BinarySearchTree();
14752
14753     private BinarySearchTree encounteredJudgeTempIDTree = new BinarySearchTree();
14754
14755     public Judge(int ID, School school, Teacher teacher) {
14756         this.ID = ID;
14757         this.school = school;
14758         this.teacher = teacher;
14759     }
14760
14761     public int getID() {
14762         return ID;
14763     }
14764
14765     public String getSchoolName() {
14766         return school.getName();
14767     }
14768
14769     public String getName() {
14770         return teacher.getName();
14771     }
14772
14773     public void insertEncounteredJudgeTempID(int tempID) throws
14774         IllegalArgumentException {
14775         encounteredJudgeTempIDTree.insert(tempID, tempID + "");
14776     }
14777
14778     public int getEncounteredJudgeTempID(int index) {
14779         int currentEncounteredJudgeTempID = (Integer)
14780         encounteredJudgeTempIDTree.getNodeData(index);
14781         return currentEncounteredJudgeTempID;
14782     }
14783
14784     public int getEncounteredJudgeTempIDTreeSize() {
14785         return encounteredJudgeTempIDTree.size();
14786     }
14787
14788     public void insertEncounteredJudge(Judge judge) throws IllegalArgumentException {
14789         if (!encounteredJudgeTree.contains(judge.getID() + "")) {
14790             encounteredJudgeTree.insert(judge, judge.getID() + "");
14791         }
14792         else throw new IllegalArgumentException();
14793     }
14794
14795     public int getEncounteredJudgeTreeSize() {
14796         return encounteredJudgeTree.size();
14797     }

```

```

14800     }
14800     public Judge getEncounteredJudgeTreeNodeData(int index) {
14800         return (Judge) encounteredJudgeTree.getNodeData(index);
14800     }
14805     public boolean isJudgeEncountered(Judge judge) {
14805         if (!encounteredJudgeTree.contains(judge.getID() + "")) {
14805             return false;
14805         } else {
14805             return true;
14810     }
14810     public void addEncounteredJudge(Judge judge) throws IllegalArgumentException {
14810         if (judge.getID() == ID) {
14810             throw new IllegalArgumentException("Cannot add itself as encountered.");
14815         } else if (!encounteredJudgeTree.contains(judge.getID() + "")) {
14815             encounteredJudgeTree.insert(judge, judge.getID() + "");
14815         } else {
14820             throw new IllegalArgumentException("Encountered judge is already in the
tree.");
14820         }
14825     public void removeEncounteredJudge(Judge judge) throws NoSuchElementException {
14825         if (encounteredJudgeTree.contains(judge.getID() + ""))
14825             encounteredJudgeTree.delete(judge.getID() + "");
14825         } else {
14830             throw new NoSuchElementException("Encountered judge to be deleted was not
found.");
14830         }
14835     public void eraseEncounteredJudgeTree() {
14835         for (int i = 0; i < encounteredJudgeTree.size(); i++) {
14835             Judge currentEncounteredJudge = (Judge)
encounteredJudgeTree.getNodeData(i);
14840                 currentEncounteredJudge.removeEncounteredJudge(this);
14840         }
14840         encounteredJudgeTree = new BinarySearchTree();
14845     }
14845     @Override
14845     public String toString() {
14845         return ID + ", " + school.getName();
14845     }
}

```

Entity.java

14850

```

/**
 * NESDA Tournament Manager 2013
 *
 * By Zbyněk Stara, 000889-045
 * International School of Prague
 * Czech Republic
 *
 * Computer used:
 * Macbook unibody aluminum late 2008
 * Mac OS X 10.6.8
 * 8 GiB of RAM
 *
 * IDE used:
 * NetBeans 6.9.1
 */

package data;

/**
 *
 * @author Zbyněk Stara
 */
public class Entity {
    // Attributes:
    private final Event.Type type;
    private final School school;
    private final boolean isPair;
    private final Student [] studentArray;
    private final String code;

    private BinarySearchTree encounteredJudgeTree = new BinarySearchTree(); // = #2
priority; teachers from the same school are restricted automatically = #1 priority

    private BinarySearchTree [] roundNewEncounteredJudgesArray;
    private int roundNewEncounteredJudgesArrayLength;

    public Entity(Event.Type type, School school, boolean isPair, Student[]
studentArray, String code) {
        this.type = type;
        this.school = school;
        this.isPair = isPair;
        this.studentArray = studentArray;
        this.code = code;
    }

    public void setRoundNewEncounteredJudgesArray(int roundsNumber) {
        roundNewEncounteredJudgesArray = new BinarySearchTree[roundsNumber];

        for (int i = 0; i < roundNewEncounteredJudgesArray.length; i++) {
            roundNewEncounteredJudgesArray[i] = new BinarySearchTree();
        }

        roundNewEncounteredJudgesArrayLength = 0;
    }

    public String getCode() {
        return code;
    }

    public Event.Type getType() {
        return type;
    }

    public String getSchoolName() {
        return school.getName();
    }

    public boolean isPair() {
        return isPair;
    }

    public Student getStudentArrayElement(int index) {
        return studentArray[index];
    }
}

```

```

    public void insertEncounteredJudgeToRound(Judge judge, int currentRoundIndex)
throws IllegalArgumentException {
    if (!encounteredJudgeTree.contains(judge.getID() + ""))
        encounteredJudgeTree.insert(judge, judge.getID() + "");
14930
    roundNewEncounteredJudgesArray[currentRoundIndex].insert(judge.getID(),
judge.getID() + "");

    if ((currentRoundIndex + 1) > roundNewEncounteredJudgesArrayLength)
roundNewEncounteredJudgesArrayLength = (currentRoundIndex + 1);
14935
    }
    else throw new IllegalArgumentException();
}

14940
public int getEncounteredJudgeTreeSize() {
    return encounteredJudgeTree.size();
}

14945
public Judge getEncounteredJudgeTreeNodeData(int index) {
    return (Judge) encounteredJudgeTree.getNodeData(index);
}

public boolean isJudgeEncountered(Judge judge) {
    if (!encounteredJudgeTree.contains(judge.getID() + ""))
        return false;
14950
    } else {
        return true;
    }
}

14955
public void addEncounteredJudgeToRound(Judge judge, int currentRoundIndex) throws
IllegalArgumentException {
    if (!encounteredJudgeTree.contains(judge.getID() + ""))
        encounteredJudgeTree.insert(judge, judge.getID() + "");
14960
    roundNewEncounteredJudgesArray[currentRoundIndex].insert(judge.getID(),
judge.getID() + "");

    if ((currentRoundIndex + 1) > roundNewEncounteredJudgesArrayLength)
roundNewEncounteredJudgesArrayLength = (currentRoundIndex + 1);
14965
    }
    else {
        throw new IllegalArgumentException("Encountered judge is already in the
tree.");
    }
}

14970
public void eraseEncounteredJudgeTreeFromRound(int currentRoundIndex) {
    for (int i = currentRoundIndex; i < roundNewEncounteredJudgesArrayLength; i++)
{
        for (int j = 0; j < roundNewEncounteredJudgesArray[i].size(); j++) {
            int currentEncounteredJudgeID = (Integer)
roundNewEncounteredJudgesArray[i].getNodeData(j);

            encounteredJudgeTree.delete(currentEncounteredJudgeID + "");
14980
        }
        roundNewEncounteredJudgesArray[i] = new BinarySearchTree();
    }
}
14985
@Override
public String toString() {
    return code;
}
14990
}

```

**Appendix 1:
First Interview with the Client**

1

1 ZBYNĚK STARA (INTERVIEWER): Hello, Ms. Caskie...

2

3 DIANNE CASKIE (CLIENT): Hello...

4

5 ZS: So this is the first interview for my computer science dossier... So, I would like
6 to know... This project is supposed to be the NESDA Tournament Manager
7 program... So, who would it be for?

8

9 DC: I think it would benefit the teachers and the students... equally.

10

11 ZS: And it's NESDA Tournament Manager, so what is NESDA?

12

13 DC: NESDA is the New European Speech and Debate Association.

14

15 ZS: You said "Speech and Debate", does that mean there are several different events
16 that take place at the tournament?

17

18 DC: Yes; this organization hosts two tournaments a year, and at those two
19 tournaments, students can compete in five different events, debate being only one of
20 them.

21

22 ZS: And what are the others?

23

24 DC: The other four are: Original Oratory – you write your own speech; Oral
25 Interpretation – you story-tell a piece of literature; Impromptu, where you make up a
26 speech in a very short amount of time; and Duet Acting, that's where you and your
27 partner learn up by heart a script and you present it, theatrically.

28

29 ZS: Okay, so can one student participate in more of these, or just one?

30

31 DC: Yes; what tends to happen, is that students enter two, to a maximum of four,
32 events.

33

34 ZS: So, tell me how are the students classified in the events? What's the scoring
35 system?

36

37 DC: Okay, the candidates are given a number; so instead of being... um... Lukáš
38 Borovička, from ISP, Lukáš becomes B19. The students are all given numbers. And
39 the students compete in different rounds of the events – so there is round one, round
40 two. And each time they perform, they are given a score out of thirty. So after two
41 rounds, they will have a score out of sixty. And then, the judges – and in the past, they
42 used a spreadsheet for this –, they work out who are the, perhaps, the eight with the
43 highest mark out of sixty, and those kids would go to the next round.

44

45 ZS: What about the student codes... I've always wondered how are the student codes
46 determined? Because I know there is the school part, and then there is the...

47

48 DC: ...the number...

49

50 ZS: ...the number part... How are these... assigned?

51

52 DC: I don't know – I could find the answer to that... I think it might be alphabetically;
53 so when I send the names of the team members to the organizing school, they will be
54 alphabetical, so Borovička would get one, Stara, at the end of the alphabet, would be
55 number eight... It's not based on success, they are not ranked; I think it is
56 alphabetical.

57

58 ZS: And you said that people are scored, according to these criteria; well, that implies
59 there have to be some judges to it...

60

61 DC: ...of course.

62

63 ZS: So who are these judges, where do they come from?

64

65 DC: Okay... good point – in each room, there will be two or three judges, and they
66 should not be the coaches of any of the kids who are competing. But sometimes,
67 because it's difficult to organize this event, you are in a position when you have to

68 listen to one of your students. And you have to try to be unbiased, either way. There
69 are teachers, but most often, these are coaches, who are experienced in NESDA.

70

71 ZS: Okay, so you said that sometimes, it's necessary to have a teacher, a coach listen
72 to his own student; this shouldn't happen, should it?

73

74 DC: In an ideal world, that shouldn't happen; but sometimes, people are ill, sometimes
75 people... there are not enough judges, so we have to ask somebody to do a double-up;
76 or an inexperienced person is asked to judge.

77

78 ZS: Also, you talked about the fact that there are different tournaments, but are those
79 at different schools?

80

81 DC: Yes. The NESDA association has twelve schools in it. And recently, it has just
82 adopted three new schools, Tunis, Barcelona, and Cairo. And what happens is, when
83 you join NESDA, part of the agreement is that you have to host. So ISP hosted two
84 years ago. And it was tremendous fun, it was incredibly stressful to organize it, but it
85 was tremendous fun.

86

87 ZS: And organization is one of the things I want to ask about... so how is this
88 problem solved right now, how are the tournaments organized, what tools are used to
89 do that?

90

91 DC: I believe it's just an Excel spreadsheet.

92

93 ZS: Excel spreadsheet, okay... so you need to put people into different rooms, and
94 you need to take into consideration all of these things, like, the school the judges are
95 from, and the people in there, and if the judge has already judged that person... so...
96 is it difficult to actually do that, in the Excel spreadsheet?

97

98 DC: I think so... I don't think... (pause) I think there must be an easier way of
99 entering those variables and using a computer to sort it... Because you have also got
100 the issue – because there are five events, and the tournament is run over two days, and
101 you have maybe between a hundred and a hundred and fifty students... and those

102 students are entered in two, three, or four events, and two of the events involve two
103 people...

104

105 ZS: ...yes, debate and duet acting...

106

107 DC: ...debate and duet acting. So there are so many variables – and I think that in the
108 past, we scheduled debate first and duet acting first, because those were the most
109 complicated ones to schedule, and then OI, OO, and impromptu were squeezed in
110 around.

111

112 ZS: So would you think that it could be made more efficient, this process?

113

114 DC: I think it must be... and computers, and Computer Science students, like
115 yourself, are so logical, and most people who do Speech and Debate, most teachers,
116 are Drama and English teachers, not famous for their logic... (laugh)

117

118 ZS: (laugh) ...no, don't say that...

119

120 DC: ...that's me being harsh...

121

122 ZS: And there is also this idea of space... you have different rooms for different
123 events, so... how many rooms are used, roughly, for the tournament?

124

125 DC: If I remember correctly, when we hosted it here... I think we used ten rooms.

126

127 ZS: And does that include also the study rooms, or not?

128

129 DC: Em... no, we had additional preparation rooms.

130

131 ZS: Okay, so ten rooms for the competition?

132

133 DC: Yes... and can I also add in the fact that it's only two days, and another variable
134 to bring in, is the fact that it would be terrible if a kid competed on day one, and all
135 finals and semifinals were on day two, and they didn't make it, so they would be

136 doing nothing for day two, apart from supporting their friends. And you could argue,
137 well, that's tough luck – you know, just improve –, but the ethos of NESDA is
138 "Everybody is engaged, everybody is progressing," so it would be lovely to have
139 some of these initial rounds on day two... So that's another variable.

140

141 ZS: Okay, you mentioned the initial round and the finals, so how many rounds are
142 there to each of the events?

143

144 DC: Em... it's all in the guidebook, but I'm pretty sure you're guaranteed two rounds,
145 two initial rounds are guaranteed.

146

147 ZS: And you mean like the...

148

149 DC: ...first round, round one, and round two. However, sometimes, there is a third
150 round... But that's very rare.

151

152 ZS: Okay... and what would be the reason for third round?

153

154 DC: To give the kids a chance to be heard again...? But usually, it's round one and
155 round two, and then it goes to a final; but for debate, you go to the semi-final...

156

157 ZS: ...semi-final... okay. So how is it determined who goes to the final? It's based on
158 the scores, isn't it?

159

160 DC: Yes, highest scores.

161

162 ZS: ...highest scores... so, but then somebody has to count those scores...

163

164 DC: ...yes...

165

166 ZS: ...and, is that very time consuming, would you say?

167

168 DC: Yes, it's quite... tedious... So there is a student who's the "runner", the judges
169 finish judging, they give the sheets to the student, the student goes to the room, the

170 score room; and usually it's the host school, its tournament director... and usually it's
171 Ms. Badeau, from Brussels, who also helps, because she's a genius with this; and she's
172 run this tournament so many times... So Ms. Badeau and the host school's tournament
173 director will take those sheets, physically enter into the spreadsheet on the computer,
174 so human error could happen there, of course... And now I'm thinking everybody's
175 got laptops... why can't they electronically enter it...

176

177 ZS: And that could be done by the computerized solution... Okay, so there are two
178 rounds – possibly three – and one final round; and there's the debate semi-finals...
179 And debate is kind of atypical by itself, isn't it? Because it has different structure to all
180 of the others...

181

182 DC: ...yes...

183

184 ZS: So what's the structure of the debate – in terms of rounds?

185

186 DC: Em... in the first round, for example, Prague would argue for the motion; in the
187 second round, the same Prague team would argue against the motion, and they should
188 be against a different pair. (pause) ...What the judges do, I can't remember the order
189 of this, but they count up the number of wins, but they also count up the number of
190 points... And I can't remember if it's the number of wins that decide you go to the
191 semis, or whether it's the total pointage that gets you to the semi, I can't remember the
192 ranking, which one is more important... can't remember... Because there is
193 something political about points – some judges score a really small amount, some
194 judges are very meager with the points, some judges give everybody high points,
195 maybe only separated by one or two points... So I think initially, it's the win/loss
196 record that decides who goes forward, and then if there's any ties, it's the actual scores
197 they were given that help decide who's final four.

198

199 ZS: Ok, now to more general question... How many schools participate in the
200 competition?

201

202 DC: Twelve this year.

203

204 ZS: ...twelve this year; so how do they decide who is going to participate? I mean in
205 terms of the list of participants, how does that get to the organizers? Do they send a
206 list of requests...

207

208 DC: Yes, so the person, the coach, who's hosting sends out an invite about four
209 months before the tournament and says, "Can you please send me how many kids are
210 you going to bring, how many debate pairs are you going to bring, and if possible, the
211 names of the candidates, and what their events are."

212

213 ZS: And when should that information be collected?

214

215 DC: Well, again, it's in the rulebook, but I think it's certainly a month before the
216 tournament.

217

218 ZS: So that means that all of this data is available to the organizers before the
219 tournament actually starts...

220

221 DC: ...yes...

222

223 ZS: ...and it cannot change once it's in...

224

225 DC: ...yes; but in the past, what has happened is, people have turned up, and said,
226 "Oh, there has been a mistake;" human error may happen, and it is possible that
227 somebody would be scheduled to be at two places at the same time. Or it might be
228 that... Jimmy... can't travel because he's got tuberculosis, or... the plague; so he's
229 withdrawn from the event. So there are some last minute changes... so there's need
230 for flexibility...

231

232 ZS: ...yes... but in general all of this information is at one place before... okay... So
233 what is the maximum possible number of participants – students, teachers?

234

235 DC: Maximum possible will be 144 – there are twelve schools and we are allowed to
236 take twelve kids – and we have two coaches per school; so it's 24 adults, 144 kids.

237

238 ZS: Okay...

239

240 DC: ...and then the host school will also provide maybe another ten judges.

241

242 ZS: Okay... before, you also said there are pair events, like the debate, and duet
243 acting, but in terms of treating the participants, these are basically one entity, aren't
244 they? Like a debate team is basically one unit, and then a duet acting team is one unit
245 as well – but it can be that one person is a part of two different teams, isn't it?

246

247 DC: Yes.

248

249 ZS: Okay... then, what are the possible limitations on teams? You said there's
250 maximum four events per person, there are different event limits...

251

252 DC: ...yes, we were sent from NESDA recently that we could only have two debate
253 pairs, we could have a maximum of three duet acting pairs, and then the other events
254 were the maximum of five – so from ISP, there would be five original orators, five
255 impromptus, and five oral interps.

256

257 ZS: Okay...

258

259 DC: ...and what it usually means is that each kid has three events. Very rarely does a
260 kid have four events.

261

262 ZS: Okay... other thing... how is the tournament organized in terms of time and
263 different rounds, I mean when do the rounds happen, actually?

264

265 DC: Yes... Well, I think what happened was that you'd schedule debate and duet
266 acting first, because they are the most complicated, and usually, kids who debate, also
267 do impromptu – so you try to avoid having impromptu and debate at the same time...

268

269 ZS: ...oh... okay...

270

271 DC: ...and what they tend to do, is try to get the two debate rounds done on the first
272 day of the tournament, which is usually Friday – so there would be a debate round
273 before lunch and a debate round after lunch; and then Saturday morning, there is a
274 semi-final debate, and then Saturday afternoon the final debate.

275

276 ZS: And in terms of the finals, you said there's a movement to actually move some of
277 those... rounds – introductory rounds – in between the final rounds... is it in-between
278 the final rounds?

279

280 DC: Well, not really... It's just often, the second rounds happen on the Saturday
281 morning – so a kid will compete on the second round, and within the next hour,
282 they're in the final... So that can be quite rushed... and they tend to put debate final...
283 as the last event, because it's usually the biggest crowd-pleaser... (smiles)

284

285 ZS: (smiles) ...but the finals, as a block, would still stay together.

286

287 DC: Yes.

288

289 ZS: Okay... Does the number of the finalists change in different tournaments?

290

291 DC: Yes – it depends, on what time you have, and how many kids you have... So if
292 there are fewer kids in the overall tournament, ironically, you can have more kids in
293 the final – the more kids you have, the less time you have, because the individual
294 rounds are longer, os the finals need to be smaller. And what tends to happen is,
295 there's a recommended number of kids for final – but I think it's up to the tournament
296 director's discretion... If there's six kids who all have very, very close scores in
297 original oratory, they might just put all six in. Or, if you look at the numbers, and
298 there an obvious 1, 2, 3, 4 and then a huge drop between the scores of four and five,
299 five just doesn't make the finals. So it might be that there's three original oratory
300 finalists, and then there's six impromptu finalists... there is clustering to a point.

301

302 ZS: So there is space for some flexibility...

303

304 DC: ...yes. But it's usually four in the final; I said three there – I have never seen a
305 final with three kids.

306

307 ZS: Okay, so a minimum of four.

308

309 DC: Yes.

310

311 ZS: And maximum of...

312

313 DC: ...six – usually.

314

315 ZS: But it's not limited to any particular number – but it's feasible if it's six...

316

317 DC: ...yes.

318

319 ZS: So what would be the steps that this solution should address...? Because it's not
320 just one problem, it's several problems that need to be done one after another... So
321 what would that be?

322

323 DC: The prioritizing?

324

325 ZS: Well, I wouldn't say...

326

327 DC: ...okay, just in general... The allocation of judges...

328

329 ZS: ...and now we are talking about the first rounds – the qualification rounds...

330

331 DC: ...yes... Allocation of judges... a mixture of kids in each room – so you
332 shouldn't have all the Prague kids in one room –; a mixture of kids in each room; a
333 mixture of judges – and hopefully no duplication of judge and school people –; the
334 actual room allocation, so you've got seven kids in ten rooms, rather than 13 kids in
335 one room, 3 in another, because we usually have a set amount of time for each round.
336 So if it's round one impromptu, you will figure out that each kid will take 7 minutes to
337 prepare and present, and the judges will take 2 minute, and you think there's 7 kids, 7

338 minutes, that's 49, let's say 60 minutes, just so kids can stand up and sit down... So it's
339 allocation of rooms, spreading out the number of competitors from each school,
340 spreading out the judges from each school, making sure that there's enough time for
341 that round to be complete and that judges can write comments... bringing in toilet
342 breaks and food breaks...

343

344 ZS: Okay, well, that would concern the qualification rounds, but then what about the
345 final rounds? There also needs to be some way how to determine who's going to the
346 finals and who actually won after the finals...

347

348 DC: ...sure. So it's the same rubrics go to the finals that were used in the preparatory
349 rounds, preliminary rounds... and ideally, the judges in the finals won't be the coaches
350 of the students in the finals; and ideally, it would be a NESDA coach who's in the
351 final – as far as possible, we are try to have the NESDA coaches in the final, because
352 – the belief is, they know the rubrics well... You know, if ISP is hosting, and we
353 invite the three History teachers to be the judges, they might not be familiar with the
354 actual rubric, and it might be unfair.

355

356 ZS: Okay... You said right now, it is being done with Excel spreadsheets, so do you
357 know of any features that could be done, that would be beneficial if they were done,
358 but can't be because of the limitations of Excel?

359

360 DC: If my memory is correct, we put the numbers in, we got the totals, and then we
361 physically had to go, "Oh, that's the highest number, that's the second highest
362 number..." so we could not get that sort by highest to lowest... which is maybe
363 because we were silly, or... I don't know...

364

365 ZS: ...but that's also good... And do you know of any alternative ways this could be
366 solved, are there any tournament managing programs available?

367

368 DC: I don't know, but there must be... With the status Speech and Debate has in the
369 US, there must be some program for it already... But we just don't know about it...
370 there must be something...

371

372 ZS: ...yes, right...? Well, I will try to find something... But for now, that's all I
373 have... So, thank you for your time, and your answers, Ms. Caskie...
374
375 DC: ...of course; thank you.

**Appendix 2:
Second Interview with the Client**

Unfortunately, as the second interview was not recorded, I cannot provide a complete transcript of what my client has said; however, the outputs of the discussion have been written down and can therefore be presented below.

1

2 **The program needs to do the following:**

3

- Record all participants (students and judges) and assign student codes;
- Place students and judges to rooms according to criteria;

5

6 **The program could also do the following:**

7

- Record the scores for the students;
- Choose the finalists;
- Choose the winners.

10

11 **Concerning the performance limits:**

12

- There is no need for instantaneity with the allocation to rooms: would not be done on the day of the tournament;
 - However, there needs to be flexibility for last-minute changes;
 - For the other performance criteria, the response should be fast as the user would expect nearly instantaneous response;
- The client has no knowledge of how big – in terms of hard drive requirements – the program can be;
- The client has no knowledge of how much memory the program might be allowed to use;
- This program will need to run on a range of computers including both Mac and Windows platforms, on desktops as well as on laptops;
- This program should be able to recognize when presented erroneous data and present warning dialogs to the users;
- It is crucial that this works in the final version, that is, on the spring 2014 tournament; it should be ready to be tested alongside the old processes in the fall 2013 tournament.

28

29

30

31

32 **Concerning the user friendliness:**

- 33 • Help menu item would be good;
- 34 • Help pop-ups are okay, but not essential;
- 35 • Some kind of tutorial would be good; different people will be using the
36 program at different tournaments;
- 37 • There is no guarantee that a person experienced with computers will use it; the
38 users of the program will be different for different tournaments.

**Appendix 3:
Initial Prototype Interview**

Unfortunately, the discussion about the first prototype was not recorded; however, the outcomes of the discussion were recorded, and the client's opinions are listed below. These points present the reasoning behind the annotations were added to the initial prototype during the interview with the client.

1

2 **About the Settings tab:**

3

- The client remarked that the Maximal tournament time and Rooms available settings have to consist of two parts to reflect two days of the tournament – since one is Friday, a school day, there might be less rooms available for the tournament than on Saturday. Also the times available for Friday and Saturday are likely to be different, as there is less time on Saturday because of people travelling home.

4

5 **About the Participants tab:**

6

- The client pointed out the lack of any NESDA logos in the program. She said that the people would “love it” if there was just a simple logo at the top of the program window at all times.
- The client likes the Export button feature, which was not discussed in previous interviews; she thinks it might be useful when compiling a student handbook of all the participants; however, she also pointed out that it is likely that the tournament organizer would already have all of that information in a separate spreadsheet, so he could use that more easily. While that is true, it is still good in my opinion to have some way to easily retrieve the data input into the program, which does need to correspond completely with the initial version.
- The client likes the Approve button functionality, with a pop-up confirmation window, which would check for any inconsistencies between the Settings and the Participants tabs. (It would warn the user, for example, if the number of schools added was less than that specified in the settings.)

7

8

9

30 **About the Qualification tab:**

- 31 • Here, my client pointed out that the approve system breaks down, since it is
32 necessary to post finalists of different events as soon as they are known. With
33 the proposed system, it would only be possible to progress (and access the
34 Finalists' Export button) after all scores for all students from all events are
35 known. However, the participants and coaches alike want to know who is in
36 the finals as soon as possible, so that they can start preparing beforehand.
37 Therefore, a change in the current system is required.

38

39 **About the Finalists tab:**

- 40 • The client inquired why is there a separate Room list, when all finals are
41 expected to happen in the same room, usually theater. The response was that
42 the program still does not know what room it will be, and so it leaves it up to
43 the user to rename the generic final room name with one that is appropriate (in
44 much the same way as in the Qualification tab's Room list).
45 • (This comment applies more to the Qualification section) The client also said
46 that the program's placement algorithm need to take into account the fact that
47 different events might last for different times, and that some events
48 (traditionally, the Duet Acting and the Impromptu) overlap, to save time
49 needed for the tournament. These two pieces of information – the overlapping
50 events, and the times per each event – need to be specified somehow in the
51 settings.

52

53 **About Winners tab:**

- 54 • (Does not connect with this tab) During the interview, we have also discussed
55 with the client if there it would be helpful to have a statistics section as a new
56 tab. My client said it might, but that there is also a thin ice of what information
57 would be shown, especially if it involved explicitly listing the judges marking
58 harshly. However, even if it were just statistics on team performances (for
59 example, the percentage of people that got to final per event per team), the
60 feature could be helpful, the client has said. But it is not critical.
61 • (Does not connect either) We have also discussed the possibility of storing
62 scans of participants' papers. These could be recorded along with the scores,

63 and after the tournament, they could be redistributed to teams, along with their
64 team's statistics. However, my client pointed out the amount of extra work and
65 time this could involve (scanning hundreds of pages per event would take
66 time) and the unsure gains this would provide (the coaches get the score sheets
67 anyways). So this possibility was more or less ruled out.