

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Data;
5 using System.Data.OleDb;
6 using System.Globalization;
7 using System.Linq;
8 using System.Text;
9 using System.Threading.Tasks;
10
11 namespace Nihulon2.Model.DbAccess
12 {
13     /*
14      * Class used for getting data from the database
15      * Inherits from DbAccess that has connection to the DB and query methods
16      * Realizes the singleton pattern
17      */
18     class DbConnector : DbAccess
19     {
20         private static string conString;
21         private static DbConnector instance;
22
23         // Flag that shows if there are time overlaps at the exams table at
24         // the DB
25         private bool _foundOverlaps;
26
27         /*
28          * Event called when exams with time overlap have been found
29          * or when all overlaps have been fixed.
30          * Used by the view for showing and hiding the relevant controls for
31          * fixing the overlaps
32          */
33         public delegate void OverlapsStateChangedHandler(bool state);
34         public event OverlapsStateChangedHandler onOverlapsStateChanged;
35
36         // Private constructor. Can't be called from other places
37         private DbConnector(string connectionString) : base(connectionString)
38         { }
39
40         #region Properties
41
42         // Return a pointer to the instance of itself
43         // create new instance if not exists
44         public static DbConnector Instance
45         {
46             get
47             {
48                 if (instance == null)
49                 {
50                     instance = new DbConnector(conString);
51                 }
52                 return instance;
53             }
54         }
55     }
```

```

56     // keeps the string with connection attributes
57     public static string ConnectionString
58     {
59         get { return conString; }
60         set
61         {
62             // The value is the path to the file of data base
63             // can be changed at the config file
64             conString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +
65                 value + ";Persist Security Info=False;";
66         }
67     }
68     // Flag that shows if there are exams with time overlaps at the DB
69     public bool foundOverlaps
70     {
71         get { return _foundOverlaps; }
72         set
73         {
74             // set value
75             _foundOverlaps = value;
76             // call the event handler
77             if (onOverlapsStateChanged != null)
78                 onOverlapsStateChanged(_foundOverlaps);
79         }
80     }
81     #endregion
82
83     #region Related items methods
84
85     //Takes an instance of related item and inserts it into the DB
86     //into the right table according to the item's type
87     public void insertRelatedItem(string itemName, string itemType)
88     {
89         // Prepare the query according to type of the related item
90         string cmdStr = "";
91         if (itemType == "חטיבות")
92             cmdStr = "INSERT INTO Divisions (division_name) VALUES (@name)";
93         else if (itemType == "מגמות")
94             cmdStr = "INSERT INTO Courses (course_name) VALUES (@name)";
95         else if (itemType == "חדרים")
96             cmdStr = "INSERT INTO Rooms (room_number) VALUES (@name)";
97
98         // If the type and name are appropriate, execute the query
99         if (cmdStr != "" && !string.IsNullOrEmpty(itemName))
100         {
101             using (OleDbCommand command = new OleDbCommand(cmdStr))
102             {
103                 command.Parameters.AddWithValue("@name", itemName);
104                 base.ExecuteSimpleQuery(command);
105             }
106         }
107     }
108
109     // Takes type of needed items (Division, Course, Room)

```

```
110 // Returns a list of needed items
111 public RelatedItem[] GetRelatedItemsByType(string itemType, bool showDisabled)
112 {
113     DataSet dSet = new DataSet();
114     ArrayList items = new ArrayList();
115     RelatedItem item;
116
117     // Build the command string according to the itemType
118     string cmdStr = getCommandStringByType(itemType);
119
120     if(showDisabled == false) // If we want only not disabled items
121         cmdStr += " WHERE disabled_status = 0";
122
123     // If the command string is initialized
124     // connect to DB and get data
125     if(cmdStr != "")
126     {
127         // Get data from DB
128         using (OleDbCommand command = new OleDbCommand(cmdStr))
129         {
130             dSet = GetMultipleQuery(command);
131         }
132
133         // Get table from the data set
134         DataTable dt = new DataTable();
135         try
136         {
137             dt = dSet.Tables[0];
138         }
139         catch { }
140
141         // Get rows from the data table and fill the ArrayList with items
142         foreach (DataRow row in dt.Rows)
143         {
144             // Create related item
145             item = new RelatedItem();
146             // Fill the fields of the item
147             item.Name = row[0].ToString();
148             item.IsDisabled = Convert.ToBoolean(row[1].ToString());
149
150             items.Add(item);
151         }
152         return (RelatedItem[])items.ToArray(typeof(RelatedItem));
153     }
154     return null; // If itemType is not valid
155 }
156
157 // Change status of the related item to disabled / not disabled
158 public void changeStatusToRelatedItem(string name, string type)
159 {
160     name = name.Replace("(מבוטל)", ""); // remove the marker from the
161     name
162
163     // Prepare the query according to the type of related item
```

```

163         // The query will invert the value of disabled_status
164         string cmdStr = "";
165         if (type == "חטיבות")
166             cmdStr = "UPDATE Divisions SET disabled_status = NOT
167                 disabled_status WHERE division_name = @name";
168         else if (type == "מגמות")
169             cmdStr = "UPDATE Courses SET disabled_status = NOT
170                 disabled_status WHERE course_name = @name";
171         else if (type == "חדרים")
172             cmdStr = "UPDATE Rooms SET disabled_status = NOT
173                 disabled_status WHERE room_number = @name";
174
175         if (cmdStr != "")
176         {
177             using (OleDbCommand command = new OleDbCommand(cmdStr))
178             {
179                 command.Parameters.AddWithValue("@name", name);
180                 base.ExecuteSimpleQuery(command);
181             }
182         }
183
184         // Get data from one of the tables of related items at the DB and
185         // return array with names
186         public string[] getRelatedItemsNamesByType(string relatedItemType)
187         {
188             DataSet dSet = new DataSet();
189             ArrayList names = new ArrayList();
190             string cmdStr = "";
191
192             switch (relatedItemType)
193             {
194                 case "חטיבות":
195                     cmdStr = "SELECT division_name from Divisions WHERE
196                         disabled_status = 0";
197                     break;
198                 case "מגמות":
199                     cmdStr = "SELECT course_name from Courses WHERE
200                         disabled_status = 0";
201                     break;
202                 case "חדרים":
203                     cmdStr = "SELECT room_number from Rooms WHERE
204                         disabled_status = 0";
205                     break;
206             }
207
208             if (cmdStr != "")
209             {
210                 // Get data from DB
211                 using (OleDbCommand command = new OleDbCommand(cmdStr))
212                 {
213                     dSet = GetMultipleQuery(command);
214                 }
215
216                 // Get table from the data set

```

```
212         DataTable dt = new DataTable();
213         try
214         {
215             dt = dSet.Tables[0];
216         }
217         catch { }
218
219         // Get rows from the data table and fill the ArrayList with names
220         foreach (DataRow row in dt.Rows)
221             names.Add(row[0].ToString());
222     }
223     return (string[])names.ToArray(typeof(string));
224 }
225 #endregion
226
227 #region Exams methods
228
229 // Get exams according to the filters that are got as parameters
230 public List<Exam> getExams(string dateFromFilter, string dateToFilter,
231     string divisionFilter,
232     string courseFilter, string roomFilter, bool showDisabledFilter,
233     bool showNewFilter)
234 {
235     List<Exam> exams;
236     string cmdStr = "SELECT * FROM Exams";
237
238     // Build a complex WHERE condition according to the sent filters
239     // and add it to the query
240     cmdStr += buildWhereCondition(dateFromFilter, dateToFilter,
241         divisionFilter, courseFilter,
242         roomFilter, showDisabledFilter, showNewFilter);
243
244     // If the command string is initialized
245     // connect to DB and get data
246     if (cmdStr != "")
247     {
248         exams = this.getListOfExamsFromDB(cmdStr);
249         return exams;
250     }
251     return null; // If initializing of the command string failed
252 }
253
254 // Load from the DB only the exams that have time overlaps
255 public List<Exam> getExamsWithOverlaps()
256 {
257     List<Exam> exams;
258     string cmdStr = "SELECT * FROM Exams WHERE hasOverlap = -1";
259
260     // If the command string is initialized
261     // connect to DB and get data
262     if (cmdStr != "")
263     {
264         exams = this.getListOfExamsFromDB(cmdStr);
265         return exams;
266     }
267 }
```

```

263         return null; // If initializing of the command string failed
264     }
265
266     // Adds a new exam to the DB
267     public void insertExam(Exam newExam)
268     {
269         string cmdStr = "";
270
271         string date = convertDate(newExam.Date);
272         string creationDate = convertDate(newExam.dateOfCreation);
273
274         // Prepare the query
275         cmdStr = "INSERT INTO Exams " +
276             " ( exam_date, supervisor_name, division_name, course_name,
277             group_name, discipline_name, room_number, start_time,
278             ending_time, canceled_status, extratime_status,
279             date_of_creation ) " +
280             " VALUES(" + date + ", @supervisor, @division, @course,
281             @group, @discipline, @room, @startTime, @endTime,
282             @isCanceled, @hasExtraTime, " + creationDate + ")";
283
284         // Set all parameters of the query and execute
285         using (OleDbCommand command = new OleDbCommand(cmdStr))
286         {
287             command.Parameters.AddWithValue("@supervisor",
288                 newExam.SupervisorName);
289             command.Parameters.AddWithValue("@division",
290                 newExam.division);
291             command.Parameters.AddWithValue("@course", newExam.course);
292             command.Parameters.AddWithValue("@group", newExam.GroupName);
293             command.Parameters.AddWithValue("@discipline",
294                 newExam.DisciplineName);
295             command.Parameters.AddWithValue("@room", newExam.room);
296             command.Parameters.AddWithValue("@startTime",
297                 newExam.StartTime);
298             command.Parameters.AddWithValue("@endTime",
299                 newExam.EndingTime);
300             command.Parameters.AddWithValue("@isCanceled",
301                 (newExam.isCanceled ? 1 : 0));
302             command.Parameters.AddWithValue("@hasExtraTime",
303                 (newExam.hasExtraTime ? 1 : 0));
304
305             base.ExecuteSimpleQuery(command);
306         }
307     }
308
309     // Remove an exam from the DB by its ID
310     public void removeExam(int examId)
311     {
312         string cmdStr = "";
313
314         // Prepare the query
315         cmdStr = "DELETE FROM Exams WHERE exam_id = @id";
316
317         if (cmdStr != "")
318         {

```

```
307         using (OleDbCommand command = new OleDbCommand(cmdStr))
308         {
309             command.Parameters.AddWithValue("@id", examId);
310             base.ExecuteNonQuery(command);
311         }
312     }
313 }
314
315 // Save the changed exam into the DB
316 public void updateExam(Exam changedExam)
317 {
318     string cmdStr = "";
319     string date = convertDate(changedExam.Date);
320
321     // Prepare the query
322     cmdStr = "UPDATE Exams " +
323         "SET exam_date = " + date + ", supervisor_name = @supervisor,
324         division_name = @division, course_name = @course, group_name
325         = @group, discipline_name = @discipline, room_number =
326         @room, start_time = @startTime, ending_time = @endingTime,
327         canceled_status = @isCanceled, extratime_status =
328         @hasExtraTime, exam_comments = @comments " +
329         "WHERE exam_id = " + changedExam.Id;
330
331     // Set all parameters of the query and execute
332     using (OleDbCommand command = new OleDbCommand(cmdStr))
333     {
334         command.Parameters.AddWithValue("@supervisor",
335             changedExam.SupervisorName);
336         command.Parameters.AddWithValue("@division",
337             changedExam.division);
338         command.Parameters.AddWithValue("@course",
339             changedExam.course);
340         command.Parameters.AddWithValue("@group",
341             changedExam.GroupName);
342         command.Parameters.AddWithValue("@discipline",
343             changedExam.DisciplineName);
344         command.Parameters.AddWithValue("@room", changedExam.room);
345         command.Parameters.AddWithValue("@startTime",
346             changedExam.StartTime);
347         command.Parameters.AddWithValue("@endingTime",
348             changedExam.EndingTime);
349         command.Parameters.AddWithValue("@isCanceled",
350             (changedExam.isCanceled ? 1 : 0));
351         command.Parameters.AddWithValue("@hasExtraTime",
352             (changedExam.hasExtraTime ? 1 : 0));
353         command.Parameters.AddWithValue("@comments",
354             changedExam.Comments);
355
356         base.ExecuteNonQuery(command);
357     }
358 }
359
360 // Takes an array of strings with id, supervisor name and comment
361 // Inserts the name of supervisor and adds the comment to the exam at
```

```
the DB
348 public void updateExamFromExcel(string[] examData)
349 {
350     int id;
351     // If the id is valid
352     if(Int32.TryParse(examData[0], out id))
353     {
354         string supervisor = examData[1];
355         string newComment = examData[2];
356
357         // Prepare the query
358         string cmdStr = "UPDATE Exams " +
359             "SET supervisor_name = @supervisor, exam_comments =
360             @comments " +
361             "WHERE exam_id = " + id;
362
363         // Set all parameters of the query and execute
364         using (OleDbCommand command = new OleDbCommand(cmdStr))
365         {
366             command.Parameters.AddWithValue("@supervisor",
367                 supervisor);
368             command.Parameters.AddWithValue("@comments", newComment);
369
370             base.ExecuteSimpleQuery(command);
371         }
372     }
373
374     // Finds the exams with time overlaps and marks them at the DB
375     // by setting the flag hasOverlap
376     public void markExamsWithOverlap()
377     {
378         Exam[] examsWithSameRoom;
379         Exam[] examsWithSameSupervisor;
380         bool hasOverlapsWithRoom = false, hasOverlapsWithSupervisor =
381             false;
382
383         // Clear all hasOverlap flags at the DB
384         this.clearAllOverlapFlags();
385
386         // Get all exams with potential overlaps
387         // (The exams that have the same room or supervisor at the same
388         day
389         examsWithSameRoom = this.getSameRoomExams();
390         examsWithSameSupervisor = this.getSameSupervisorExams();
391
392         // Check the potential overlaps whether there are the real
393         overlaps
394         // and mark all overlapped exams at the DB
395         if (examsWithSameRoom.Length > 1 && examsWithSameRoom != null)
396             hasOverlapsWithRoom = this.checkAndMarkRoomOverlaps
397                 (examsWithSameRoom);
398         if (examsWithSameSupervisor.Length > 1 &&
399             examsWithSameSupervisor != null)
400             hasOverlapsWithSupervisor =
401                 this.checkAndMarkSupervisorOverlaps
```



```

        (examsWithSameSupervisor);

395
396     // If found any time overlaps, set the flag "foundOverlaps"
397     if (hasOverlapsWithRoom || hasOverlapsWithSupervisor)
398         this.foundOverlaps = true;
399     else
400         this.foundOverlaps = false;
401 }
402
403 #endregion
404
405 #region Private methods for internal use
406
407 // Returns a command string with SELECT from one of the tables
408 // of related items (Division, Course, Room) according to the itemType ↗

409 private string getCommandStringByType(string itemType)
410 {
411     string cmdStr;
412     switch (itemType)
413     {
414         case "חטיבות":
415             cmdStr = "SELECT * FROM [Divisions]";
416             break;
417         case "מגמות":
418             cmdStr = "SELECT * FROM [Courses]";
419             break;
420         case "חדרים":
421             cmdStr = "SELECT * FROM [Rooms]";
422             break;
423         default:
424             cmdStr = "";
425             break;
426     }
427     return cmdStr;
428 }
429
430
431 // Build WHERE condition according to the parameters
432 private string buildWhereCondition(string dateFromFilter, string ↗
433     dateToFilter, string divisionFilter, string courseFilter, string ↗
434     roomFilter, bool showDisabledFilter, bool showNewFilter)
435 {
436     string str = " WHERE ";
437
438     // Flag that shows if there is any condition
439     // if no conditions, the method returns empty string
440     bool whereIsSet = false;
441
442     if (dateFromFilter != "") // if the period of time filter is set
443     {
444         // Convert dates to the correct form for the query (ex. ↗
445         // 12.07.2019 -> #7/12/2019#)
446         dateFromFilter = convertDate(dateFromFilter);
447         dateToFilter = convertDate(dateToFilter);

```

```
446         // Add the condition to the WHERE and set the flag that WHERE is set
447         str += "exam_date >= " + dateFromFilter + " AND exam_date <= " + dateToFilter;
448         whereIsSet = true;
449     }
450     if (divisionFilter != "הכול") // if the division filter is set
451     {
452         if (whereIsSet) // if there is at least one condition before, add "AND" to expression
453             str += " AND ";
454         str += " division_name = \"" + divisionFilter + "\"";
455         whereIsSet = true;
456     }
457     if (courseFilter != "הכול") // if the course filter is set
458     {
459         if (whereIsSet) // if there is at least one condition before, add "AND" to expression
460             str += " AND ";
461         str += " course_name = \"" + courseFilter + "\"";
462         whereIsSet = true;
463     }
464     if (roomFilter != "הכול") // if the room filter is set
465     {
466         if (whereIsSet) // if there is at least one condition before, add "AND" to expression
467             str += " AND ";
468         str += " room_number = \"" + roomFilter + "\"";
469         whereIsSet = true;
470     }
471     if (showDisabledFilter == false) // If disabled exams are going to be shown
472     {
473         if (whereIsSet) // if there is at least one condition before, add "AND" to expression
474             str += " AND ";
475         str += " canceled_status = 0";
476         whereIsSet = true;
477     }
478     if (showNewFilter == true) // If only new exams are going to be shown
479     {
480         if (whereIsSet) // if there is at least one condition before, add "AND" to expression
481             str += " AND ";
482         string date = convertDate(DateTime.Today.ToString("dd.MM.yyyy"));
483         str += " date_of_creation = " + date;
484         whereIsSet = true;
485     }
486
487     // If there is no WHERE conditions, return an empty string
488     if (whereIsSet)
489         return str;
490     else
491         return "";
```

```
492     }
493
494     // Convert a date to the correct form for the query (12.07.2019 -> #7/12/2019#)
495     private string convertDate(string date)
496     {
497
498         string[] partsOfDate = date.Split('.');
499         string newDate = "#" + partsOfDate[1] + "/" + partsOfDate[0] + "/" + partsOfDate[2] + "#";
500
501         return newDate;
502     }
503
504     // Get the exams with the same room at the same day from the DB
505     private Exam[] getSameRoomExams()
506     {
507         List<Exam> exams;
508
509         string cmdStr = "SELECT * FROM Exams INNER JOIN " +
510             "(SELECT exam_date, room_number, COUNT(*) AS occurrences FROM Exams " +
511             "WHERE canceled_status = 0 GROUP BY exam_date, room_number HAVING COUNT(*) > 1) AS t1 " +
512             "ON(t1.exam_date = Exams.exam_date) AND (t1.room_number = Exams.room_number) " +
513             "ORDER BY Exams.room_number, Exams.exam_date";
514
515         // Get data from DB
516         exams = this.getListOfExamsFromDB(cmdStr);
517         return exams.ToArray();
518     }
519     // Get the exams with the same supervisor at the same day from the DB
520     private Exam[] getSameSupervisorExams()
521     {
522         List<Exam> exams;
523
524         string cmdStr = "SELECT * FROM Exams INNER JOIN " +
525             "(SELECT exam_date, supervisor_name, COUNT(*) AS occurrences FROM Exams " +
526             "WHERE (canceled_status = 0) AND(supervisor_name <> \"\") " +
527             "GROUP BY exam_date, supervisor_name HAVING COUNT(*) > 1) AS t1 " +
528             "ON(t1.exam_date = Exams.exam_date) AND (t1.supervisor_name = Exams.supervisor_name) " +
529             "ORDER BY Exams.supervisor_name, Exams.exam_date";
530
531         // Get data from DB
532         exams = this.getListOfExamsFromDB(cmdStr);
533         return exams.ToArray();
534     }
535
536     // Sets the flag "hasOverlap" to 0 for all exams
537     private void clearAllOverlapFlags()
538     {
```

```
539         // Set command string
540         string cmdStr = "UPDATE Exams SET hasOverlap = 0 " +
541             "WHERE hasOverlap <> 0";
542
543         // Execute
544         using (OleDbCommand command = new OleDbCommand(cmdStr))
545         {
546             base.ExecuteSimpleQuery(command);
547         }
548     }
549
550     // Takes the array of exams that has the same room at the same day
551     // and checks whether there are time overlaps by using the formula:
552     // if aStart < bEnd AND bStart < aEnd => there is an overlap between a
553     // and b
554     // If found, marks the couple of overlapped exams at the DB
555     private bool checkAndMarkRoomOverlaps(Exam[] exams)
556     {
557         bool hasOverlaps = false;
558
559         // when the flag "sameRoom" is false, no need to check
560         // the current exam with others because all exams sorted by rooms
561         // makes the loop faster
562         bool sameRoom;
563
564         for (int i = 0; i < exams.Length - 1; i++)
565         {
566             sameRoom = true;
567             for(int j = i+1; j < exams.Length && sameRoom; j++)
568             {
569                 // if the exams have the same room or supervisor, and the
570                 // date
571                 if (exams[i].Date == exams[j].Date && exams[i].room ==
572                     exams[j].room)
573                 {
574                     // Check the couple of exams and if thy are
575                     // overlapped, mark them at the DB
576                     // and set the flag
577                     if (this.checkAndMarkTwoExamsIfOverlapped(exams[i],
578                         exams[j]) && hasOverlaps == false)
579                         hasOverlaps = true;
580                 }
581                 else
582                     sameRoom = false;
583             }
584         }
585         return hasOverlaps;
586     }
587
588     // Takes the array of exams that has the same supervisor at the same
589     // day
590     // and checks whether there are time overlaps.
591     // If found, marks the overlapped exams at the DB
592     private bool checkAndMarkSupervisorOverlaps(Exam[] exams)
593     {
594         bool hasOverlaps = false;
```

```
589         // when the flag "sameSupervisor" is false, no need to check
590         // the current exam with others because all exams sorted by
591         supervisor
592         // makes the loop faster
593         bool sameSupervisor;
594
595         for (int i = 0; i < exams.Length - 1; i++)
596         {
597             sameSupervisor = true;
598             for (int j = i + 1; j < exams.Length && sameSupervisor; j++)
599             {
600                 // if the exams have the same room or supervisor, and the
601                 date
602                 if (exams[i].Date == exams[j].Date && exams
603                 [i].SupervisorName == exams[j].SupervisorName)
604                 {
605                     // Check the couple of exams and if thy are
606                     overlapped, mark them at the DB
607                     // and set the flag
608                     if (this.checkAndMarkTwoExamsIfOverlapped(exams[i],
609                     exams[j]) && hasOverlaps == false)
610                     hasOverlaps = true;
611                 }
612                 else
613                     sameSupervisor = false;
614             }
615         }
616         return hasOverlaps;
617     }
618
619     // Set the flag "hasOverlap" at the DB for the couple of overlapped
620     exams
621     private void markOverlap(int id1, int id2)
622     {
623         string cmdStr = "UPDATE Exams SET hasOverlap = -1 " +
624             "WHERE exam_id = @id1 OR exam_id = @id2";
625
626         // Set all parameters of the query and execute
627         using (OleDbCommand command = new OleDbCommand(cmdStr))
628         {
629             command.Parameters.AddWithValue("@id1", id1);
630             command.Parameters.AddWithValue("@id2", id2);
631             base.ExecuteNonQuery(command);
632         }
633     }
634
635     /*
636     * Fills an exam with data from the data row
637     * and returns the exam
638     */
639     private Exam getExamFromDataRow(DataRow row)
640     {
641         Exam exam = new Exam();
642
643         // Get time and date
```

```
639         DateTime dateT;
640         // Get the date of the exam
641         try
642         {
643             dateT = Convert.ToDateTime(row[1].ToString());
644             exam.Date = dateT.ToString("dd.MM.yyyy");
645         }
646         catch { }
647         // Get the time of the start
648         try
649         {
650             dateT = Convert.ToDateTime(row[8].ToString());
651             exam.StartTime = dateT.ToShortTimeString();
652         }
653         catch { }
654         // Get the time of the end
655         try
656         {
657             dateT = Convert.ToDateTime(row[9].ToString());
658             exam.EndingTime = dateT.ToShortTimeString();
659         }
660         catch { }
661         // Get the date of creation
662         try
663         {
664             dateT = Convert.ToDateTime(row[13].ToString());
665             exam.dateOfCreation = dateT.ToShortDateString();
666         }
667         catch { }
668
669         // Fill all fields of the exam with values
670         exam.Id = (int)row[0];
671         exam.SupervisorName = row[2].ToString();
672         exam.division = row[3].ToString();
673         exam.course = row[4].ToString();
674         exam.GroupName = row[5].ToString();
675         exam.DisciplineName = row[6].ToString();
676         exam.room = row[7].ToString();
677         exam.isCanceled = Convert.ToBoolean(row[10].ToString());
678         exam.hasExtraTime = Convert.ToBoolean(row[11].ToString());
679         exam.Comments = row[12].ToString();
680         exam.hasOverlap = Convert.ToBoolean(row[14].ToString());
681
682         return exam;
683     }
684
685     /*
686     * Gets exams according to the command string,
687     * fills each exam with data, builds a list with exams
688     * end returns the list
689     */
690     private List<Exam> getListOfExamsFromDB(string cmdStr)
691     {
692         List<Exam> exams = new List<Exam>();
693         DataSet dSet = new DataSet();
694         Exam exam;
```

```
695         // Get data from DB
696         using (OleDbCommand command = new OleDbCommand(cmdStr))
697         {
698             dSet = GetMultipleQuery(command);
699         }
700         // Get table from the data set
701         DataTable dt = new DataTable();
702         try
703         {
704             dt = dSet.Tables[0];
705         }
706         catch { }
707
708         // Get rows from the data table and fill the ArrayList with items
709         foreach (DataRow row in dt.Rows)
710         {
711             // Create an exam
712             exam = new Exam();
713             // Fill the exam with data from the row
714             exam = this.getExamFromDataRow(row);
715             // Add the new exam to the array
716             exams.Add(exam);
717         }
718         return exams;
719     }
720
721     // Takes two exams as parameters and
722     // checks whether there are time overlaps by using the formula:
723     // if aStart < bEnd AND bStart < aEnd => there is an overlap between a ↗
    and b
724     // If found overlap, mark the couple of exams at the DB
725     private bool checkAndMarkTwoExamsIfOverlapped(Exam examA, Exam examB)
726     {
727         DateTime aStart, aEnd, bStart, bEnd;
728
729         // Convert time from strings to DateTime for comparing
730         aStart = Convert.ToDateTime(examA.StartTime);
731         aEnd = Convert.ToDateTime(examA.EndingTime);
732         bStart = Convert.ToDateTime(examB.StartTime);
733         bEnd = Convert.ToDateTime(examB.EndingTime);
734
735         // Compare the time of the start and the end of the exams
736         if (aStart <= bEnd && bStart <= aEnd)
737         {
738             // If found overlap, mark the exams at the DB
739             this.markOverlap(examA.Id, examB.Id);
740             // Set the flag that overlap has been found
741             return true;
742         }
743         else
744             return false;
745     }
746     #endregion
747 }
748 }
```