```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Nihulon2.Model.DbAccess
{
    /*
     * Class used for getting data from the database
     * Inherits from DbAccess that has connection to the DB and query methods
     * Realizes the singleton pattern
     */
    class DbConnector : DbAccess
    {
        private static string conString;
        private static DbConnector instance;

        // Flag that shows if there are time overlaps at the exams table at the DB
        private bool _foundOverlaps;

        /*
         * Event called when exams with time overlap have been found
         * or when all overlaps have been fixed.
         * Used by the view for showing and hiding the relevant controls for
         * fixing the overlaps
         */
        public delegate void OverlapsStateChangedHandler(bool state);
        public event OverlapsStateChangedHandler onOverlapsStateChanged;

        // Private constructor. Can't be called from other places
        private DbConnector(string connectionstring) : base(connectionstring)
        { }


        #region Properties

        // Return a pointer to the instance of itself
        // create new instance if not exists
        public static DbConnector Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new DbConnector(conString);
                }
                return instance;
            }
        }

```

```csharp
        // keeps the string with connection attributes
        public static string ConnectionString
        {
            get { return conString; }
            set
            {
                // The value is the path to the file of data base
                // can be changed at the config file
                conString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source="
                  +
                    value + ";Persist Security Info=False;";
            }
        }
        // Flag that shows if there are exams with time overlaps at the DB
        public bool foundOverlaps
        {
            get { return _foundOverlaps; }
            set
            {
                // set value
                _foundOverlaps = value;
                // call the event handler
                if (onOverlapsStateChanged != null)
                    onOverlapsStateChanged(_foundOverlaps);
            }
        }
        #endregion

        #region Related items methods

        //Takes an instance of related item and inserts it into the DB
        //into the right table according to the item's type
        public void insertRelatedItem(string itemName, string itemType)
        {
            // Prepare the query according to type of the related item
            string cmdStr = "";
            if (itemType == "חטיבות")
                cmdStr = "INSERT INTO Divisions (division_name) VALUES
                  (@name)";
            else if(itemType == "מגמות")
                cmdStr = "INSERT INTO Courses (course_name) VALUES (@name)";
            else if (itemType == "חדרים")
                cmdStr = "INSERT INTO Rooms (room_number) VALUES (@name)";

            // If the type and name are appropriate, execute the query
            if(cmdStr != "" && !string.IsNullOrEmpty(itemName))
            {
                using (OleDbCommand command = new OleDbCommand(cmdStr))
                {
                    command.Parameters.AddWithValue("@name", itemName);
                    base.ExecuteSimpleQuery(command);
                }
            }
        }

        // Takes type of needed items (Division, Course, Room)
        // Returns a list of needed items
        public RelatedItem[] GetRelatedItemsByType(string itemType, bool showDisabled)
        {
            DataSet dSet = new DataSet();
            ArrayList items = new ArrayList();
            RelatedItem item;

            // Build the command string according to the itemType
            string cmdStr = getCommandStringByType(itemType);

            if(showDisabled == false) // If we want only not disabled items
                cmdStr += " WHERE disabled_status = 0";

            // If the command string is initialized
            // connect to DB and get data
            if(cmdStr != "")
            {
                // Get data from DB
                using (OleDbCommand command = new OleDbCommand(cmdStr))
                {
                    dSet = GetMultipleQuery(command);
                }

                // Get table from the data set
                DataTable dt = new DataTable();
                try
                {
                    dt = dSet.Tables[0];
                }
                catch { }

                // Get rows from the data table and fill the ArrayList with items
                foreach (DataRow row in dt.Rows)
                {
                    // Create related item
                    item = new RelatedItem();
                    // Fill the fields of the item
                    item.Name = row[0].ToString();
                    item.IsDisabled = Convert.ToBoolean(row[1].ToString());

                    items.Add(item);
                }
                return (RelatedItem[])items.ToArray(typeof(RelatedItem));
            }
            return null; // If itemType is not valid
        }

        // Change status of the related item to disabled / not disabled
        public void changeStatusToRelatedItem(string name, string type)
        {
            name = name.Replace("(מבוטל)", ""); // remove the marker from the name

            // Prepare the query according to the type of related item
            // The query will invert the value of disabled_status
            string cmdStr = "";
            if (type == "חטיבות")
                cmdStr = "UPDATE Divisions SET disabled_status = NOT disabled_status WHERE division_name = @name";
            else if (type == "מגמות")
                cmdStr = "UPDATE Courses SET disabled_status = NOT disabled_status WHERE course_name = @name";
            else if (type == "חדרים")
                cmdStr = "UPDATE Rooms SET disabled_status = NOT disabled_status WHERE room_number = @name";

            if (cmdStr != "")
            {
                using (OleDbCommand command = new OleDbCommand(cmdStr))
                {

                    command.Parameters.AddWithValue("@name", name);
                    base.ExecuteSimpleQuery(command);
                }
            }
        }

        // Get data from one of the tables of related items at the DB and return array with names
        public string[] getRelatedItemsNamesByType(string relatedItemType)
        {
            DataSet dSet = new DataSet();
            ArrayList names = new ArrayList();
            string cmdStr = "";

            switch (relatedItemType)
            {
                case "חטיבות":
                    cmdStr = "SELECT division_name from Divisions WHERE disabled_status = 0";
                    break;
                case "מגמות":
                    cmdStr = "SELECT course_name from Courses WHERE disabled_status = 0";
                    break;
                case "חדרים":
                    cmdStr = "SELECT room_number from Rooms WHERE disabled_status = 0";
                    break;
            }

            if(cmdStr != "")
            {
                // Get data from DB
                using (OleDbCommand command = new OleDbCommand(cmdStr))
                {
                    dSet = GetMultipleQuery(command);
                }

                // Get table from the data set
                DataTable dt = new DataTable();
                try
                {
                    dt = dSet.Tables[0];
                }
                catch { }

                // Get rows from the data table and fill the ArrayList with names
                foreach (DataRow row in dt.Rows)
                    names.Add(row[0].ToString());
            }
            return (string[])names.ToArray(typeof(string));
        }
        #endregion

        #region Exams methods

        // Get exams according to the filters that are got as parameters
        public List<Exam> getExams(string dateFromFilter, string dateToFilter, string divisionFilter,
            string courseFilter, string roomFilter, bool showDisabledFilter, bool showNewFilter)
        {
            List<Exam> exams;
            string cmdStr = "SELECT * FROM Exams";

            // Build a complex WHERE condition according to the sent filters and add it to the query
            cmdStr += bildWhereCondition(dateFromFilter, dateToFilter, divisionFilter, courseFilter,
                roomFilter, showDisabledFilter, showNewFilter);

            // If the command string is initialized
            // connect to DB and get data
            if (cmdStr != "")
            {
                exams = this.getListOfExamsFromDB(cmdStr);
                return exams;
            }
            return null; // If initializing of the command string failed
        }

        // Load from the DB only the exams that have time overlaps
        public List<Exam> getExamsWithOverlaps()
        {
            List<Exam> exams;
            string cmdStr = "SELECT * FROM Exams WHERE hasOverlap = -1";

            // If the command string is initialized
            // connect to DB and get data
            if (cmdStr != "")
            {
                exams = this.getListOfExamsFromDB(cmdStr);
                return exams;
            }
            return null; // If initializing of the command string failed
        }

        // Adds a new exam to the DB
        public void insertExam(Exam newExam)
        {
            string cmdStr = "";

            string date = convertDate(newExam.Date);
            string creationDate = convertDate(newExam.dateOfCreation);

            // Prepare the query
            cmdStr = "INSERT INTO Exams " +
                " ( exam_date, supervisor_name, division_name, course_name, group_name, discipline_name, room_number, start_time, ending_time, canceled_status, extratime_status, date_of_creation ) " +
                " VALUES(" + date + ", @supervisor, @division, @course, @group, @discipline, @room, @startTime, @endingTime, @isCanceled, @hasExtraTime, " + creationDate + ")";

            // Set all parameters of the query and execute
            using (OleDbCommand command = new OleDbCommand(cmdStr))
            {
                command.Parameters.AddWithValue("@supervisor", newExam.SupervisorName);
                command.Parameters.AddWithValue("@division", newExam.division);
                command.Parameters.AddWithValue("@course", newExam.course);
                command.Parameters.AddWithValue("@group", newExam.GroupName);
                command.Parameters.AddWithValue("@discipline", newExam.DisciplineName);
                command.Parameters.AddWithValue("@room", newExam.room);
                command.Parameters.AddWithValue("@startTime", newExam.StartTime);
                command.Parameters.AddWithValue("@endingTime", newExam.EndingTime);
                command.Parameters.AddWithValue("@isCanceled", (newExam.isCanceled ? 1 : 0));
                command.Parameters.AddWithValue("@hasExtraTime", (newExam.hasExtraTime ? 1 : 0));

                base.ExecuteSimpleQuery(command);
            }
        }

        // Remove an exam from the DB by its ID
        public void removeExam(int examId)
        {
            string cmdStr = "";

            // Prepare the query
            cmdStr = "DELETE FROM Exams WHERE exam_id = @id";

            if (cmdStr != "")
            {
                using (OleDbCommand command = new OleDbCommand(cmdStr))
                {
                    command.Parameters.AddWithValue("@id", examId);
                    base.ExecuteSimpleQuery(command);
                }
            }
        }

        // Save the changed exam into the DB
        public void updateExam(Exam changedExam)
        {
            string cmdStr = "";
            string date = convertDate(changedExam.Date);

            // Prepare the query
            cmdStr = "UPDATE Exams " +
                "SET exam_date = " + date + ", supervisor_name = @supervisor, division_name = @division, course_name = @course, group_name = @group, discipline_name = @discipline, room_number = @room, start_time = @startTime, ending_time = @endingTime, canceled_status = @isCanceled, extratime_status = @hasExtraTime, exam_comments = @comments " +
                "WHERE exam_id = " + changedExam.Id;


            // Set all parameters of the query and execute
            using (OleDbCommand command = new OleDbCommand(cmdStr))
            {
                command.Parameters.AddWithValue("@supervisor", changedExam.SupervisorName);
                command.Parameters.AddWithValue("@division", changedExam.division);
                command.Parameters.AddWithValue("@course", changedExam.course);
                command.Parameters.AddWithValue("@group", changedExam.GroupName);
                command.Parameters.AddWithValue("@discipline", changedExam.DisciplineName);
                command.Parameters.AddWithValue("@room", changedExam.room);
                command.Parameters.AddWithValue("@startTime", changedExam.StartTime);
                command.Parameters.AddWithValue("@endingTime", changedExam.EndingTime);
                command.Parameters.AddWithValue("@isCanceled", (changedExam.isCanceled ? 1 : 0));
                command.Parameters.AddWithValue("@hasExtraTime", (changedExam.hasExtraTime ? 1 : 0));
                command.Parameters.AddWithValue("@comments", changedExam.Comments);

                base.ExecuteSimpleQuery(command);
            }
        }

        // Takes an array of strings with id, supervisor name and comment
        // Inserts the name of supervisor and adds the comment to the exam at
```

```
          the DB
        public void updateExamFromExcel(string[] examData)
        {
            int id;
            // If the id is valid
            if(Int32.TryParse(examData[0], out id))
            {
                string supervisor = examData[1];
                string newComment = examData[2];

                // Prepare the query
                string cmdStr = "UPDATE Exams " +
                    "SET supervisor_name = @supervisor, exam_comments = @comments " +
                    "WHERE exam_id = " + id;

                // Set all parameters of the query and execute
                using (OleDbCommand command = new OleDbCommand(cmdStr))
                {
                    command.Parameters.AddWithValue("@supervisor", supervisor);
                    command.Parameters.AddWithValue("@comments", newComment);

                    base.ExecuteSimpleQuery(command);
                }
            }
        }

        // Finds the exams with time overlaps and marks them at the DB
        // by setting the flag hasOverlap
        public void markExamsWithOverlap()
        {
            Exam[] examsWithSameRoom;
            Exam[] examsWithSameSupervisor;
            bool hasOverlapsWithRoom = false, hasOverlapsWithSupervisor = false;

            // Clear all hasOverlap flags at the DB
            this.clearAllOverlapFlags();

            // Get all exams with potential overlaps
            // (The exams that have the same room or supervisor at the same day
            examsWithSameRoom = this.getSameRoomExams();
            examsWithSameSupervisor = this.getSameSupervisorExams();

            // Check the potential overlaps whether there are the real overlaps
            // and mark all overlapped exams at the DB
            if (examsWithSameRoom.Length > 1 && examsWithSameRoom != null)
                hasOverlapsWithRoom = this.checkAndMarkRoomOverlaps(examsWithSameRoom);
            if (examsWithSameSupervisor.Length > 1 && examsWithSameSupervisor != null)
                hasOverlapsWithSupervisor =
                  this.checkAndMarkSupervisorOverlaps
```

```
                      (examsWithSameSupervisor);

            // If found any time overlaps, set the flag "foundOverlaps"
            if (hasOverlapsWithRoom || hasOverlapsWithSupervisor)
                this.foundOverlaps = true;
            else
                this.foundOverlaps = false;
        }

        #endregion

        #region Private methods for internal use

        // Returns a command string with SELECT from one of the tables
        // of related items (Division, Course, Room) according to the itemType

        private string getCommandStringByType(string itemType)
        {
            string cmdStr;
            switch (itemType)
            {
                case "חטיבות":
                    cmdStr = "SELECT * FROM [Divisions]";
                    break;
                case "מגמות":
                    cmdStr = "SELECT * FROM [Courses]";
                    break;
                case "חדרים":
                    cmdStr = "SELECT * FROM [Rooms]";
                    break;
                default:
                    cmdStr = "";
                    break;
            }
            return cmdStr;
        }


        // Build WHERE condition according to the parameters
        private string bildWhereCondition(string dateFromFilter, string
          dateToFilter, string divisionFilter, string courseFilter, string
          roomFilter, bool showDisabledFilter, bool showNewFilter)
        {
            string str = " WHERE ";

            // Flag that shows if there is any condition
            // if no conditions, the method returns empty string
            bool whereIsSet = false;

            if(dateFromFilter != "") // if the period of time filter is set
            {
                // Convert dates to the correct form for the query (ex.
                  12.07.2019 -> #7/12/2019#)
                dateFromFilter = convertDate(dateFromFilter);
                dateToFilter = convertDate(dateToFilter);

```

```csharp
                // Add the condition to the WHERE and set the flag that WHERE is set
                str += "exam_date >= " + dateFromFilter + " AND exam_date <= " + dateToFilter;
                whereIsSet = true;
            }
            if(divisionFilter != "הכול") // if the division filter is set
            {
                if (whereIsSet) // if there is at least one condition before, add "AND" to expression
                    str += " AND ";
                str += " division_name = \"" + divisionFilter + "\"";
                whereIsSet = true;
            }
            if (courseFilter != "הכול") // if the course filter is set
            {
                if (whereIsSet) // if there is at least one condition before, add "AND" to expression
                    str += " AND ";
                str += " course_name = \"" + courseFilter + "\"";
                whereIsSet = true;
            }
            if (roomFilter != "הכול") // if the room filter is set
            {
                if (whereIsSet) // if there is at least one condition before, add "AND" to expression
                    str += " AND ";
                str += " room_number = \"" + roomFilter + "\"";
                whereIsSet = true;
            }
            if(showDisabledFilter == false) // If disabled exams are going to be shown
            {
                if (whereIsSet) // if there is at least one condition before, add "AND" to expression
                    str += " AND ";
                str += " canceled_status = 0";
                whereIsSet = true;
            }
            if (showNewFilter == true) // If only new exams are going to be shown
            {
                if (whereIsSet) // if there is at least one condition before, add "AND" to expression
                    str += " AND ";
                string date = convertDate(DateTime.Today.ToString("dd.MM.yyyy"));
                str += " date_of_creation = " + date;
                whereIsSet = true;
            }

            // If there is no WHERE conditions, return an empty string
            if (whereIsSet)
                return str;
            else
                return "";
```

```
        }

        // Convert a date to the correct form for the query (12.07.2019 -> #7/12/2019#)
        private string convertDate(string date)
        {

            string[] partsOfDate = date.Split('.');
            string newDate = "#" + partsOfDate[1] + "/" + partsOfDate[0] + "/" + partsOfDate[2] + "#";

            return newDate;
        }

        // Get the exams with the same room at the same day from the DB
        private Exam[] getSameRoomExams()
        {
            List<Exam> exams;

            string cmdStr = "SELECT * FROM Exams INNER JOIN " +
                            "(SELECT exam_date, room_number, COUNT(*) AS occurrences FROM Exams " +
                            "WHERE canceled_status = 0 GROUP BY exam_date, room_number HAVING COUNT(*) > 1)  AS t1 " +
                            "ON(t1.exam_date = Exams.exam_date) AND (t1.room_number = Exams.room_number) " +
                            "ORDER BY Exams.room_number, Exams.exam_date";

            // Get data from DB
            exams = this.getListOfExamsFromDB(cmdStr);
            return exams.ToArray();
        }
        // Get the exams with the same supervisor at the same day from the DB
        private Exam[] getSameSupervisorExams()
        {
            List<Exam> exams;

            string cmdStr = "SELECT * FROM Exams INNER JOIN " +
                            "(SELECT exam_date, supervisor_name, COUNT(*) AS occurrences FROM Exams " +
                            "WHERE (canceled_status = 0) AND(supervisor_name <> \"\") " +
                            "GROUP BY exam_date, supervisor_name HAVING COUNT(*) > 1)  AS t1 " +
                            "ON(t1.exam_date = Exams.exam_date) AND (t1.supervisor_name = Exams.supervisor_name) " +
                            "ORDER BY Exams.supervisor_name, Exams.exam_date";

            // Get data from DB
            exams = this.getListOfExamsFromDB(cmdStr);
            return exams.ToArray();
        }

        // Sets the flag "hasOverlap" to 0 for all exams
        private void clearAllOverlapFlags()
        {
```

```csharp
            // Set command string
            string cmdStr = "UPDATE Exams SET hasOverlap = 0 " +
                            "WHERE hasOverlap <> 0";

            // Execute
            using (OleDbCommand command = new OleDbCommand(cmdStr))
            {
                base.ExecuteSimpleQuery(command);
            }
        }

        // Takes the array of exams that has the same room at the same day
        // and checks whether there are time overlaps by using the formula:
        // if aStart < bEnd AND bStart < aEnd => there is an overlap between a and b
        // If found, marks the couple of overlapped exams at the DB
        private bool checkAndMarkRoomOverlaps(Exam[] exams)
        {
            bool hasOverlaps = false;

            // when the flag "sameRoom" is false, no need to check
            // the current exam with others because all exams sorted by rooms
            // makes the loop faster
            bool sameRoom;

            for (int i = 0; i < exams.Length - 1; i++)
            {
                sameRoom = true;
                for(int j = i+1; j < exams.Length && sameRoom; j++)
                {
                    // if the exams have the same room or supervisor, and the date
                    if (exams[i].Date == exams[j].Date && exams[i].room == exams[j].room)
                    {
                        // Check the couple of exams and if thy are overlapped, mark them at the DB
                        // and set the flag
                        if (this.checkAndMarkTwoExamsIfOverlapped(exams[i], exams[j]) && hasOverlaps == false)
                            hasOverlaps = true;
                    }
                    else
                        sameRoom = false;
                }
            }
            return hasOverlaps;
        }
        // Takes the array of exams that has the same supervisor at the same day
        // and checks whether there are time overlaps.
        // If found, marks the overlapped exams at the DB
        private bool checkAndMarkSupervisorOverlaps(Exam[] exams)
        {
            bool hasOverlaps = false;

            // when the flag "sameSupervisor" is false, no need to check
            // the current exam with others because all exams sorted by supervisor
            // makes the loop faster
            bool sameSupervisor;

            for (int i = 0; i < exams.Length - 1; i++)
            {
                sameSupervisor = true;
                for (int j = i + 1; j < exams.Length && sameSupervisor; j++)
                {
                    // if the exams have the same room or supervisor, and the date
                    if (exams[i].Date == exams[j].Date && exams[i].SupervisorName == exams[j].SupervisorName)
                    {
                        // Check the couple of exams and if thy are overlapped, mark them at the DB
                        // and set the flag
                        if (this.checkAndMarkTwoExamsIfOverlapped(exams[i], exams[j]) && hasOverlaps == false)
                            hasOverlaps = true;
                    }
                    else
                        sameSupervisor = false;
                }
            }
            return hasOverlaps;
        }

        // Set the flag "hasOverlap" at the DB for the couple of overlapped exams
        private void markOverlap(int id1, int id2)
        {
            string cmdStr = "UPDATE Exams SET hasOverlap = -1 " +
                            "WHERE exam_id = @id1 OR exam_id = @id2";

            // Set all parameters of the query and execute
            using (OleDbCommand command = new OleDbCommand(cmdStr))
            {
                command.Parameters.AddWithValue("@id1", id1);
                command.Parameters.AddWithValue("@id2", id2);
                base.ExecuteSimpleQuery(command);
            }
        }


        /*
         * Fills an exam with data from the data row
         * and returns the exam
         */
        private Exam getExamFromDataRow(DataRow row)
        {
            Exam exam = new Exam();

            // Get time and date
            DateTime dateT;
            // Get the date of the exam
            try
            {
                dateT = Convert.ToDateTime(row[1].ToString());
                exam.Date = dateT.ToString("dd.MM.yyyy");
            }
            catch { }
            // Get the time of the start
            try
            {
                dateT = Convert.ToDateTime(row[8].ToString());
                exam.StartTime = dateT.ToShortTimeString();
            }
            catch { }
            // Get the time of the end
            try
            {
                dateT = Convert.ToDateTime(row[9].ToString());
                exam.EndingTime = dateT.ToShortTimeString();
            }
            catch { }
            // Get the date of creation
            try
            {
                dateT = Convert.ToDateTime(row[13].ToString());
                exam.dateOfCreation = dateT.ToShortDateString();
            }
            catch { }

            // Fill all fields of the exam with values
            exam.Id = (int)row[0];
            exam.SupervisorName = row[2].ToString();
            exam.division = row[3].ToString();
            exam.course = row[4].ToString();
            exam.GroupName = row[5].ToString();
            exam.DisciplineName = row[6].ToString();
            exam.room = row[7].ToString();
            exam.isCanceled = Convert.ToBoolean(row[10].ToString());
            exam.hasExtraTime = Convert.ToBoolean(row[11].ToString());
            exam.Comments = row[12].ToString();
            exam.hasOverlap = Convert.ToBoolean(row[14].ToString());

            return exam;
        }

        /*
         * Gets exams according to the command string,
         * fills each exam with data, builds a list with exams
         * end returns the list
         */
        private List<Exam> getListOfExamsFromDB(string cmdStr)
        {
            List<Exam> exams = new List<Exam>();
            DataSet dSet = new DataSet();
            Exam exam;
            // Get data from DB
            using (OleDbCommand command = new OleDbCommand(cmdStr))
            {
                dSet = GetMultipleQuery(command);
            }
            // Get table from the data set
            DataTable dt = new DataTable();
            try
            {
                dt = dSet.Tables[0];
            }
            catch { }

            // Get rows from the data table and fill the ArrayList with items
            foreach (DataRow row in dt.Rows)
            {
                // Create an exam
                exam = new Exam();
                // Fill the exam with data from the row
                exam = this.getExamFromDataRow(row);
                // Add the new exam to the array
                exams.Add(exam);
            }
            return exams;
        }

        // Takes two exams as parameters and
        // checks whether there are time overlaps by using the formula:
        // if aStart < bEnd AND bStart < aEnd => there is an overlap between a and b
        // If found overlap, mark the couple of exams at the DB
        private bool checkAndMarkTwoExamsIfOverlapped(Exam examA, Exam examB)
        {
            DateTime aStart, aEnd, bStart, bEnd;

            // Convert time from strings to DateTime for comparing
            aStart = Convert.ToDateTime(examA.StartTime);
            aEnd = Convert.ToDateTime(examA.EndingTime);
            bStart = Convert.ToDateTime(examB.StartTime);
            bEnd = Convert.ToDateTime(examB.EndingTime);

            // Compare the time of the start and the end of the exams
            if (aStart <= bEnd && bStart <= aEnd)
            {
                // If found overlap, mark the exams at the DB
                this.markOverlap(examA.Id, examB.Id);
                // Set the flag that overlap has been found
                return true;
            }
            else
                return false;
        }
        #endregion
    }
}
```