תכנות מונחה עצמים ב# C

חלק 1 – GUI

מגישים:

אושרי בוסקילה	313349193	oshrib297@gmail.com
נועה בר-דוד	205788565	noaaruppin@gmail.com

קוד:

Lesson

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

namespace GuiAndClasses
{
    class Lesson
    {
        private string subject;
        private Student[] allStudents;
        private int numOfStudents;

        public Lesson(string subject, int maxStudents)
        {
            this.subject = subject;
            allStudents = new Student[maxStudents];
            numOfStudents = 0;
        }

        public Student[] GetStudents() { return allStudents; }
        public string GetSubject() { return subject; }
        public int GetMaxStudents() { return allStudents.Length; }
        public int GetNumOfStudents() { return numOfStudents; }
        public bool CanAddAnotherStudent() { return numOfStudents <
allStudents.Length; }

        public bool RemoveStudent(Student s)
        {
            for (int i = 0; i < numOfStudents; i++)
            {
                if (allStudents[i] == s)
                {
                    for (int j = i + 1; j < numOfStudents; j++)
                    {
                        allStudents[j - 1] = allStudents[j];
                    }
                    allStudents[numOfStudents - 1] = null;
                    numOfStudents--;
                    return true;
                }
            }
            return false;
        }
        public bool AddStudent(Student s)
        {
            if (CanAddAnotherStudent())
            {
                allStudents[numOfStudents++] = s;
                return true;
            }
            else
                return false;
        }

        public override string ToString()
        {
```

```
            string res = "The class \" " + subject+ " \" has " +
numOfStudents + " students (out of " + allStudents.Length + "):\n";
            for (int i = 0; i < numOfStudents; i++)
                res += "\t" + allStudents[i].ToString();

            return res;
        }
    }
```

Student

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GuiAndClasses
{
    class Student
    {
        private string name;
        private int[] allGrades;
        private int numOfGrades;

        public Student(string name)
        {
            this.name = name;
            allGrades = new int[2];
            numOfGrades = 0;
        }

        public string GetName() { return name; }
        public int GetNumOfGrades() { return numOfGrades; }

        public bool AddGrade(int newGrade)
        {
            if (newGrade < 0 || newGrade > 100)
            {
                return false;
            }

            if (numOfGrades == allGrades.Length)
            {
                int[] temp = new int[allGrades.Length * 2];
                for (int i = 0; i < allGrades.Length; i++)
                    temp[i] = allGrades[i];

                allGrades = temp;
            }

            allGrades[numOfGrades++] = newGrade;
            return true;
        }
        public override string ToString()
        {
            string res = name + " has " + numOfGrades + " grades: ";
            for (int i = 0; i < numOfGrades; i++)
                res += allGrades[i] + " ";
            res += "\n";
```

```
            return res;
        }
    }
}
```

Subjects

המחלקה החדשה שנוספה

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace GuiAndClasses
{

    internal class Subjects
    {
        private string name;
        private Lesson[] allLessons;
        private int numOfLessons;

        public Subjects(string name)
        {
           this.name = name;
           allLessons = new Lesson[2];
           numOfLessons = 0;

        }

        public string GetName() { return name; }
        public Lesson[] GetLesson() { return allLessons; }
        public bool CanAddAnotherLesson() { return numOfLessons <
allLessons.Length; }
        public bool AddLesson(Lesson s)
        {
            if (CanAddAnotherLesson())
            {
                allLessons[numOfLessons++] = s;
                return true;
            }
            else
                return false;
        }

        public override string ToString()
        {
            string res = name;
            return res;
        }
        public bool AddLessons(Lesson newLesson)
        {
            if (numOfLessons == allLessons.Length)
            {
```

```
                Lesson[] temp = new Lesson[allLessons.Length * 2];
                for (int i = 0; i < allLessons.Length; i++)
                    temp[i] = allLessons[i];

                allLessons = temp;
            }

            allLessons[numOfLessons++] = newLesson;
            return true;
        }

    }
}
```

Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GuiAndClasses
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmCreateLesson());
        }
    }
}
```

From1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;

namespace GuiAndClasses
{
    public partial class frmCreateLesson : Form
    {
        private Lesson theLesson;
```

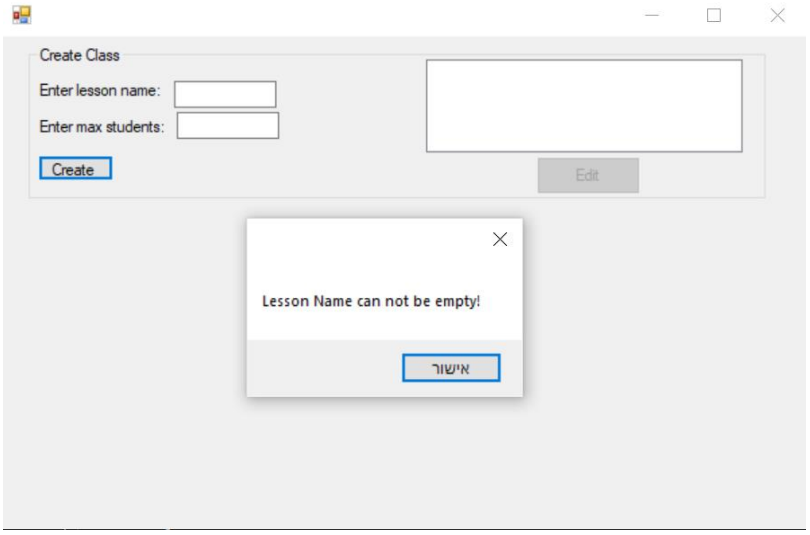
```
        private Subjects theSubject;

        public frmCreateLesson()
        {
            InitializeComponent();
        }

        private void btnCreateLesson_Click(object sender, EventArgs e)
        {
            EnableLessonOperations(false);
            if (txtLessonName.Text.Trim().Equals(""))
            {
                MessageBox.Show("Lesson Name can not be empty!");
                return;

            }
            if (txtMaxStudents.Text.Trim().Equals(""))
            {
                    MessageBox.Show("The number of students in the class
can not be empty!");
                    return;
            }



            Lesson newLesson = new Lesson(txtLessonName.Text,
int.Parse(txtMaxStudents.Text));
            lstlesson.Items.Add(newLesson);
            txtLessonName.Text = "";
            txtMaxStudents.Text = "";

        }

        private void lstlesson_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (lstlesson.SelectedItem == null)
                EnableLessonOperations(false);
            else
                EnableLessonOperations(true);
        }

        private void button1_Click(object sender, EventArgs e)  //btnEdit
        {
            Lesson currentLesson = lstlesson.SelectedItem as Lesson;
            if (currentLesson == null)
            {
                EnableLessonOperations(false);
                MessageBox.Show("You must select a lesson first");
            }
            else
            {

                theLesson = (Lesson) lstlesson.SelectedItem;
                grpCreateLesson.Visible = true;
                grpStudents.Visible = true;
                EnableLessonOperations (true);
                EnableStduentOperations(false);
                lstStudents.Items.Clear();
                Student[] tempLst = theLesson.GetStudents();
                foreach (Student S in tempLst)
                {
                    if (S != null) { lstStudents.Items.Add(S); }
```

```
        }
         this.Text = currentLesson.GetSubject();
    }
}

private void EnableLessonOperations(bool toEnable)
{

    btnEditLesson.Enabled = toEnable;

}
private void UpdateLessonsList()
{
    lstlesson.Items.Clear();
}

private void txtLessonName_TextChanged(object sender, EventArgs e)
{

}
private void btnAddStudent_Click(object sender, EventArgs e)
{
    if (txtStudentName.Text.Trim().Equals(""))
    {
        MessageBox.Show("Student Name can not be empty!");
        return;
    }

    Student newStudent = new Student(txtStudentName.Text);
    theLesson.AddStudent(newStudent);
    lstStudents.Items.Add(newStudent);
    txtStudentName.Text = "";

    if (theLesson.CanAddAnotherStudent() == false)
        grpAddStudent.Enabled = false;
}

private void btnRemoveStudent_Click(object sender, EventArgs e)
{
    Student currentStudent = lstStudents.SelectedItem as Student;
    if (currentStudent == null)
    {
        MessageBox.Show("You must select a student first");
    }
    else
    {
        theLesson.RemoveStudent(currentStudent);
        UpdateStudentList();
        grpAddStudent.Enabled = true;
        EnableStduentOperations(false);
    }
}

private void UpdateStudentList()
{
    lstStudents.Items.Clear();
    Student[] allStudents = theLesson.GetStudents();
    foreach (Student s in allStudents)
    {
        if (s != null)
            lstStudents.Items.Add(s);
    }
}
```

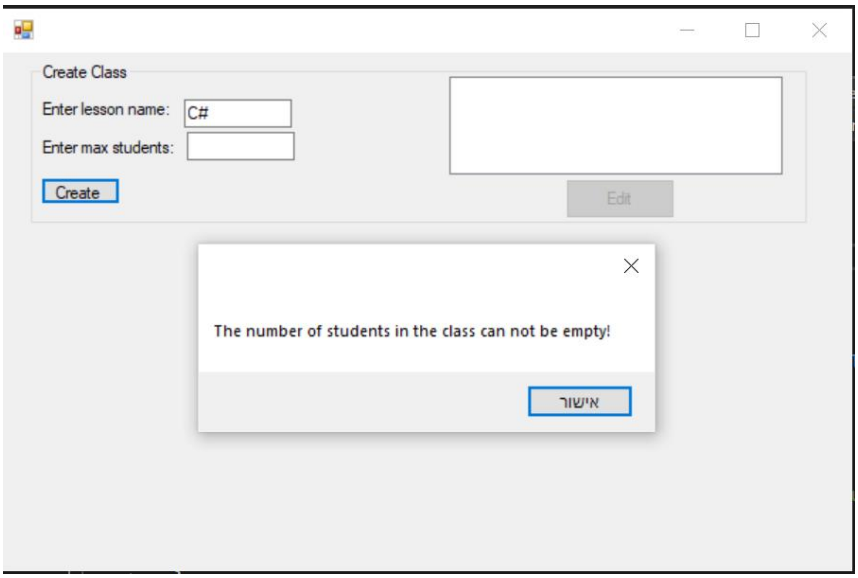
```
        private void btnAddGrade_Click(object sender, EventArgs e)
        {
            Student currentStudent = lstStudents.SelectedItem as Student;
            if (currentStudent == null)
            {
                MessageBox.Show("You must select a student first");

            }
            else if (int.Parse(txtGrade.Text) < 0 ||
(int.Parse(txtGrade.Text) > 100))
            {
                MessageBox.Show("You must enter a value between 0 and
100");
                txtGrade.Text = "";
            }
            else if (txtGrade.Text.Equals(""))
            {
                MessageBox.Show("Error, An empty value cannot be
entered");
            }
            else
            {
                MessageBox.Show("Grade is updated");
                currentStudent.AddGrade(int.Parse(txtGrade.Text));
                UpdateStudentList();
                txtGrade.Text = "";
            }
            EnableStduentOperations(false);
        }

        private void EnableStduentOperations(bool toEnable)
        {
            grpAddGrade.Enabled = toEnable;
            btnRemoveStudent.Enabled = toEnable;
        }

        private void lstStudents_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (lstStudents.SelectedItem == null)
                EnableStduentOperations(false);
            else
                EnableStduentOperations(true);
        }

        private void txtStudentName_TextChanged(object sender, EventArgs
e)
        {

        }

        private void txtGrade_TextChanged(object sender, EventArgs e)
        {

        }
```

צילומי מסך:

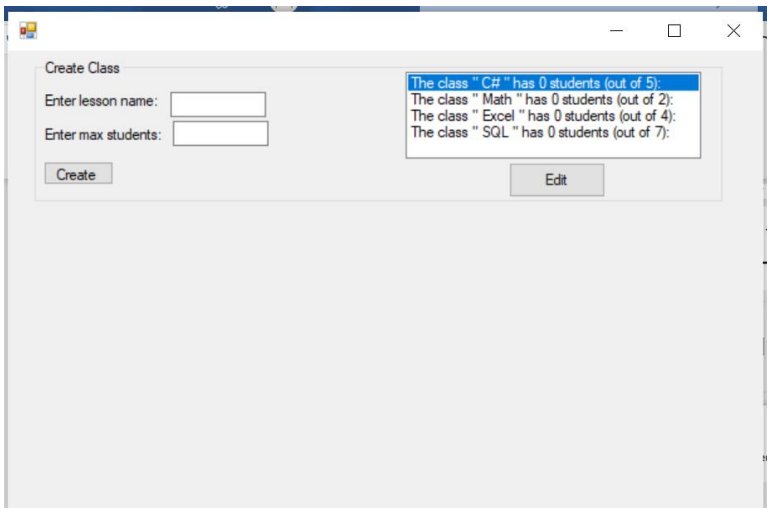
הצגת הודעה כאשר יש ניסיון להוסיף שם שיעור ללא שם



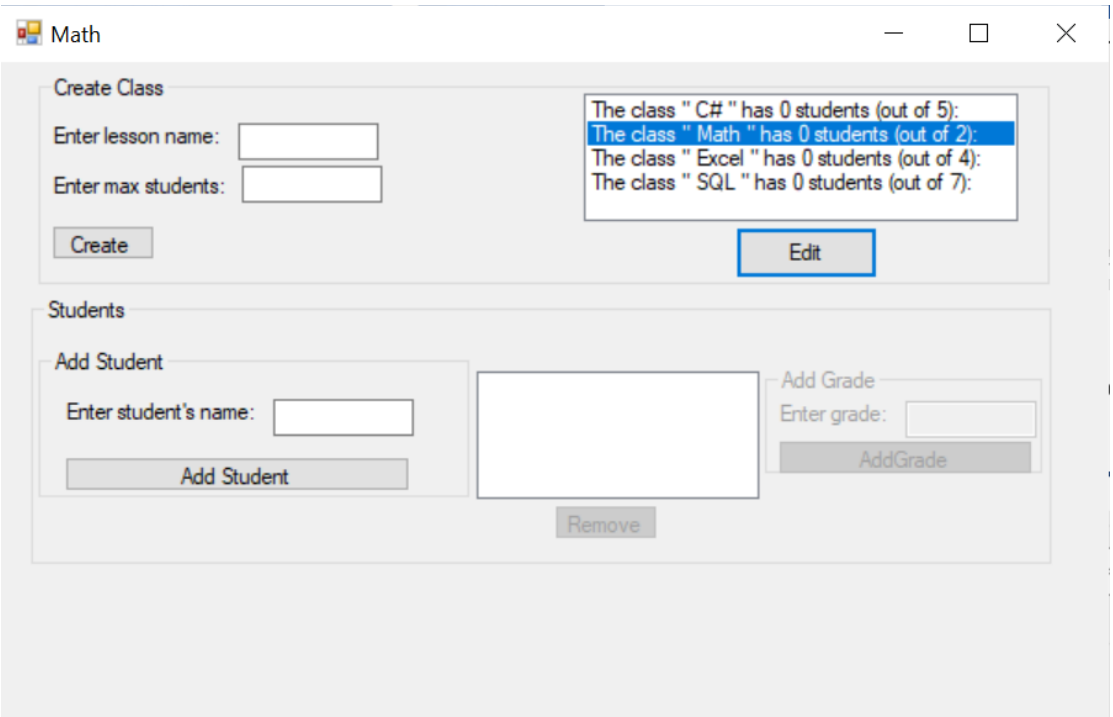
הצגת הודעה כאשר יש ניסיון להוסיף שם שיעור ללא כמות תלמידים



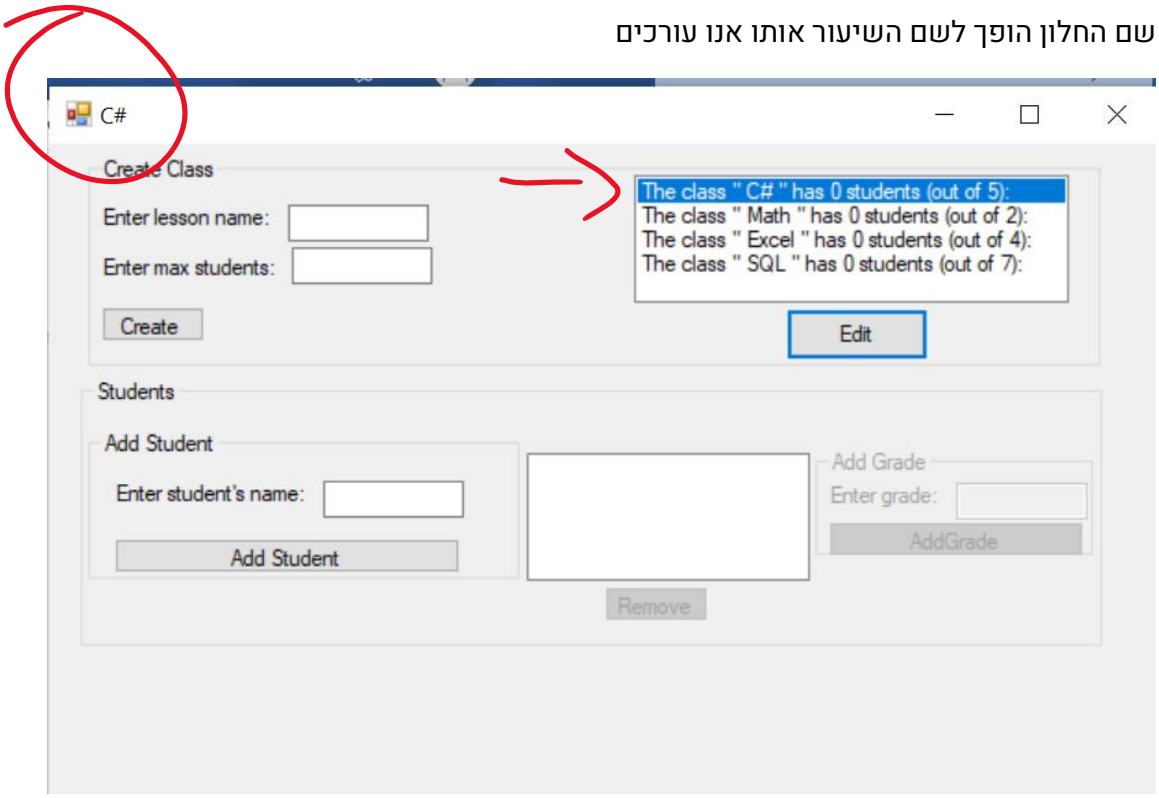
זמינות לחצן "EDIT" עריכת שיעור (עריכת סטודנטים בשיעור) רק לאחר בחירת שיעור מתוך הרשימה

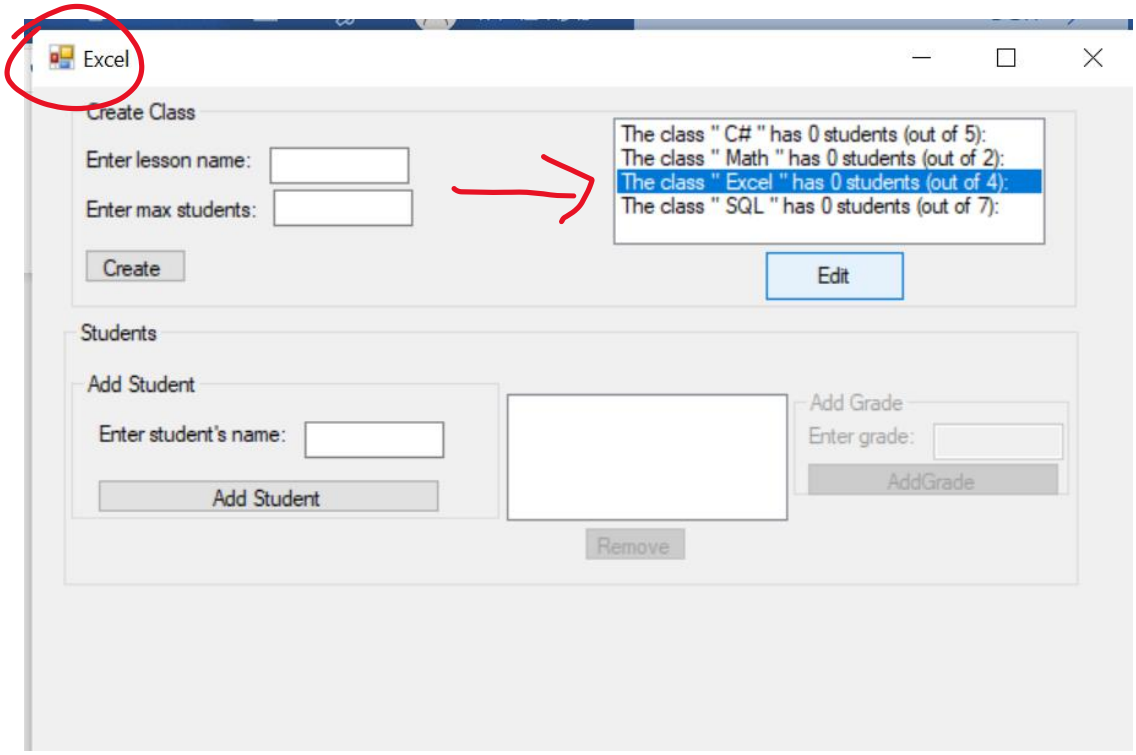


הופעת חלון STUDENTS רק לאחר לחיצה על עריכת שיעור



שם החלון הופך לשם השיעור אותו אנו עורכים

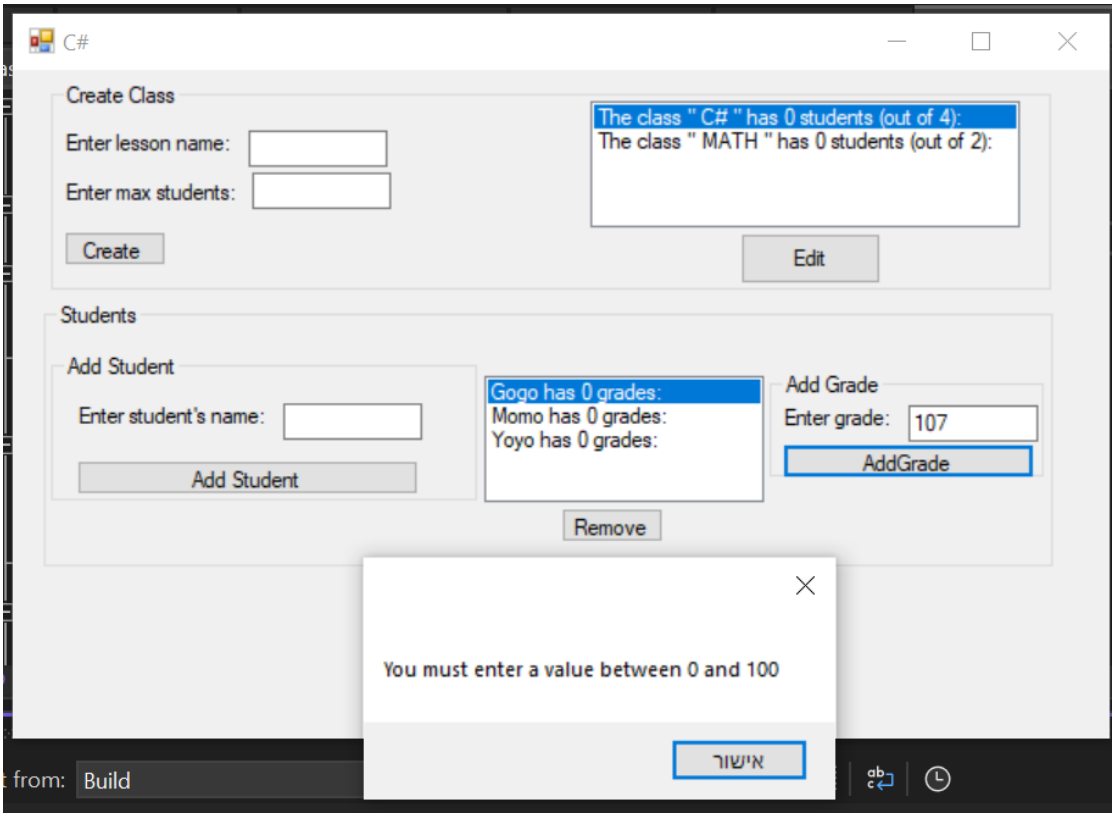




הצגת הודעה כאשר יש ניסיון להוסיף ציון ריק (סעיף 1 במטלה)

```
1 reference
private void btnAddGrade_Click(object sender, EventArgs e)
{
    Student currentStudent = lstStudents.SelectedItem as Student;
    if (currentStudent == null)
    {
        MessageBox.Show("You must select a student first");

    }
    else if (int.Parse(txtGrade.Text) < 0 || (int.Parse(txtGrade.Text) > 100))
    {
        MessageBox.Show("You must enter a value between 0 and 100");
        txtGrade.Text = "";
    }
    else if (txtGrade.Text.Equals(""))
    {
        MessageBox.Show("Error, An empty value cannot be entered");
    }
    else
    {
        MessageBox.Show("Grade is updated");
        currentStudent.AddGrade(int.Parse(txtGrade.Text));
        UpdateStudentList();
        txtGrade.Text = "";
    }
    EnableStduentOperations(false);
}
```

הצגת הודעה כאשר יש ניסיון להוסיף ציון שאינו בטווח 0-100 (סעיף 2 במטלה)




```
1 reference
private void btnAddGrade_Click(object sender, EventArgs e)
{
    Student currentStudent = lstStudents.SelectedItem as Student;
    if (currentStudent == null)
    {
        MessageBox.Show("You must select a student first");

    }
    else if (int.Parse(txtGrade.Text) < 0 || (int.Parse(txtGrade.Text) > 100))
    {
        MessageBox.Show("You must enter a value between 0 and 100");
        txtGrade.Text = "";
    }
    else if (txtGrade.Text.Equals(""))
    {
        MessageBox.Show("Error, An empty value cannot be entered");
    }
    else
    {
        MessageBox.Show("Grade is updated");
        currentStudent.AddGrade(int.Parse(txtGrade.Text));
        UpdateStudentList();
        txtGrade.Text = "";
    }
    EnableStduentOperations(false);
}
```