

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 404 |

Отчёт по контрольной работе № 4

Вариант № 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 404 |  |  |  | Азаров Даниил Константинович |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

# Постановка задачи

Студент выполняет задание обычной или повышенной сложности. Необходимо написать приложение с использованием технологии WinForms реализующие хранилище списка студентов. Программа должна позволять добавлять новые сущности с использованием интерфейса и редактировать существующие. Сущности, добавленные в программу должны сохраняться между запусками приложения. Для хранения данных необходимо использовать СУБД SqLite. Необходимо предусмотреть возможность сохранения списка существующих сущностей в файл.

# 2. Исходные данные

Данные из БД.

# 3. Особые ситуации

Если БД не существует.

# 4. Математические методы и алгоритмы решения задачи

# 1) Запрос получения строк “SELECT \* FROM Student”;

2) Запрос удаления строки “DELETE FROM Student WHERE id = @id”, где @id – id удаляемой строки;

3) Запрос изменения строки “UPDATE Student SET {массив значений} WHERE id = @id”, где {массив значений} – значения, которые надо изменить, @id – id изменяемой строки;

4) Запрос добавления строки “INSERT INTO Student VALUES ({массив значений})”, где {массив значений} – значения, которые надо вставить.

# 5. Форматы представления данных

Таблица 1 — Классы, используемые в программе.

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Имя переменной** | **Тип** | **Описание** |
| SQLiteDatabase | connection | SqliteConnection | Коннект с базой данных |
| **Метод** | **Тип** | **Описание** |
| InitialRows<T> | void | Заполняет массив значениями из базы данных |
| SaveChanges<T> | void |  |
| UpdateRowCommand<T> | string | Возвращает команду обновления строки |
| AddRowCommand<T> | string | Возвращает команду добавления строки |
| DeleteRowCommand<T> | string | Возвращает команду удаления строки |
| **Название** | **Имя переменной** | **Тип** | **Описание** |
| DbRowList<T> | TableName | string | Название таблицы |
| \_addedRows | List<T> | Добавленные строки |
| \_removedRows | List<T> | Удаленные строки |
| \_initialRows | List<T> | Изначальные строки |
| \_rows | List<T> | Строки |
| \_autoIncrementPropertyIndexes | List<int> | Индексы автоинкрементирующихся свойств |
| \_autoIncrementValues | Int64[] | Максимальное значение автоинкрементирующегося числа |
| **Метод** | **Тип** | **Описание** |
| DbInitialRows | void | Инициализация строк |
| Add | void | Добавление строк |
| RemoveAt | void | Удаление строки |
| GetChanges | Dictionary<Change, List<T>> | Возвращает изменения |
| IsChanged | bool | Есть изменения |
| ClearChanges | void | Очистить изменения |
| **Название** | **Метод** | **Тип** | **Описание** |
| DbRow | FillProperties | void | Заполнение свойств |
| GetPropertiesValue | object[] | Получить значения свойств |
| GetPrimaryPropertyIndexes | List<int> | Получить индексы primary свойств |
| GetAutoIncrementPropertyIndexes | List<int> | Получить индексы autoincrement свойств |
| Same | Bool | Проверка не является ли это той же строкой |
| Equals | Bool | Проверка не равны ли все значения строки |
| CopyTo | void | Скопировать свойства |
| **Название** | **Метод** | **Тип** | **Описание** |
| PrimaryAttribute | - | - | Атрибут для праймери свойств |
| AutoIncrementAttribute | - | - | Атрибут для автоинкрементирующихся свойств |

# 6. Структура программы

Таблица 2 — Модули, на которые разбита программа.

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Program | Запуск программы |
| Database | Модуль работы с базой данных |
| UI | Модуль графического представления |
| Presenter | Модуль связи UI и работой с БД |
| Files | Модуль работы с файлами |

На рисунке 1 представлена блок-схема алгоритма, представленная в программе:

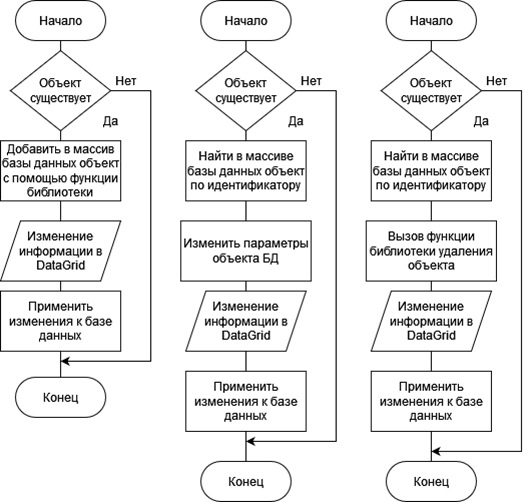


Рисунок 1 — Блок-схема алгоритма

# 7. Описание хода выполнения лабораторной работы

1. Были реализованы пользовательский интерфейс.
2. Были написан классы работы с БД.
3. Была добавлен класс для хранения строк БД
4. Было добавлено меню.
5. Была добавлена форма вывода информации о программе.
6. Была добавлена возможность сохранения в файл.
7. Были написаны модульные тесты.
8. Был написан отчёт.

# 8. Результаты работы программы

На рисунках 2 - 3 представлены скриншоты работы программы.

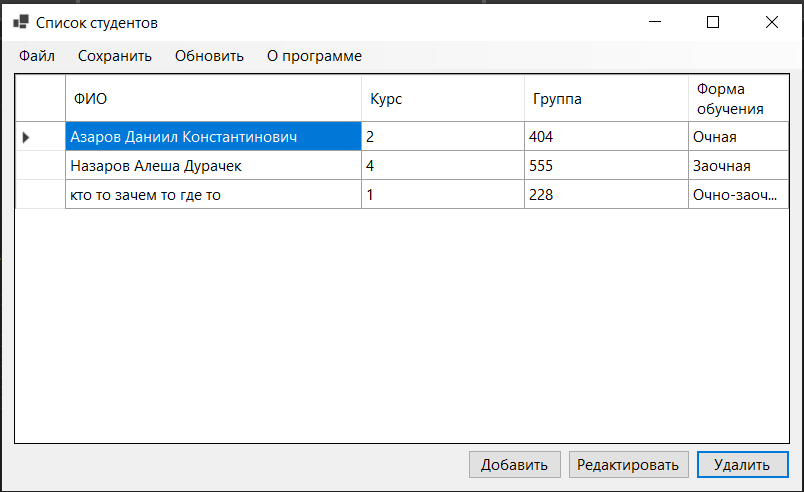


Рисунок 2 — Начало работы программы

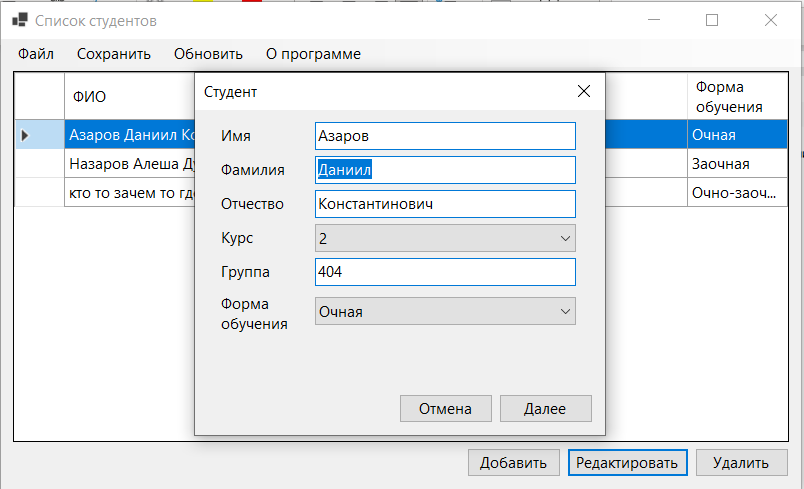


Рисунок 3 — Выбор параметров

# 9. Исходный текст программы

[Начало SQLiteDatabase.cs ---]

using System;

using System.Collections.Generic;

using System.Collections;

using System.Reflection;

using Microsoft.Data.Sqlite;

using System.Threading.Tasks;

namespace Database

{

#region DbRowList

public class DbRowList<T> where T : DbRow, new()

{

public readonly string TableName;

private readonly List<T> \_addedRows;

private readonly List<T> \_removedRows;

private readonly List<T> \_initialRows;

private List<T> \_rows;

private readonly List<int> \_autoIncrementPropertyIndexes;

private readonly Int64[] \_autoIncrementValues;

public enum Change

{

Add,

Edit,

Delete

}

public DbRowList(string tableName)

{

TableName = tableName;

\_addedRows = new List<T>();

\_removedRows = new List<T>();

\_initialRows = new List<T>();

\_rows = new List<T>();

DbRow dbRow = new T();

\_autoIncrementPropertyIndexes = dbRow.GetAutoIncrementPropertyIndexes();

\_autoIncrementValues = new Int64[\_autoIncrementPropertyIndexes.Count];

for (int i = 0; i < \_autoIncrementValues.Length; i++)

{

\_autoIncrementValues[i] = -1;

}

}

public T this[int index]

{

get => \_rows[index];

}

public T GetLast() => \_rows[\_rows.Count - 1];

public void DbInitialRows(List<T> rows)

{

\_rows = rows;

\_initialRows.Clear();

\_addedRows.Clear();

\_removedRows.Clear();

foreach (T row in \_rows)

{

T copyRow = new T();

row.CopyTo(copyRow);

\_initialRows.Add(copyRow); //Можно ли использовать конструктор при обобщении?

object[] values = row.GetPropertiesValue();

foreach (int index in \_autoIncrementPropertyIndexes)

{

if ((Int64)values[index] <= \_autoIncrementValues[index])

{

values[index] = \_autoIncrementValues[index] + 1;

}

\_autoIncrementValues[index] = (Int64)values[index];

}

}

}

public void Add(T row)

{

\_rows.Add(row);

\_addedRows.Add(row);

object[] values = row.GetPropertiesValue();

foreach (int index in \_autoIncrementPropertyIndexes)

{

if ((Int64)values[index] <= \_autoIncrementValues[index])

{

values[index] = \_autoIncrementValues[index] + 1;

row.FillProperties(values);

}

\_autoIncrementValues[index] = (Int64)values[index];

}

}

public void RemoveAt(int i)

{

if (!\_addedRows.Contains(\_rows[i]))

{

\_removedRows.Add(\_rows[i]);

}

\_addedRows.Remove(\_rows[i]);

\_rows.RemoveAt(i);

}

public void Remove(T row)

{

if (!\_addedRows.Contains(row))

{

\_removedRows.Add(row);

}

\_addedRows.Remove(row);

\_rows.Remove(row);

}

public Dictionary<Change, List<T>> GetChanges()

{

Dictionary<Change, List<T>> rows = new Dictionary<Change, List<T>>()

{

{ Change.Add, new List<T>(\_addedRows) },

{ Change.Edit, new List<T>(GetEditedRows()) },

{ Change.Delete, new List<T>(\_removedRows) }

};

return rows;

}

public List<T> GetAddedRows() => \_addedRows;

public List<T> GetRemovedRows() => \_removedRows;

public List<T> GetEditedRows()

{

List<T> changeRows = new List<T>();

for (int j = 0; j < \_initialRows.Count; j++)

{

for (int i = 0; i < \_rows.Count; i++)

{

if (\_rows[i].Equals(\_initialRows[j]))

{

break;

}

else if (\_rows[i].Same(\_initialRows[j]))

{

changeRows.Add(\_rows[i]);

break;

}

}

}

return changeRows;

}

public bool IsChanged()

{

Dictionary<Change, List<T>> rows = GetChanges();

bool isRowsRemoved = GetAddedRows().Count > 0;

bool isRowsAdded = GetRemovedRows().Count > 0;

bool isRowsEdited = GetEditedRows().Count > 0;

return isRowsRemoved || isRowsAdded || isRowsEdited;

}

public void ClearChanges()

{

\_addedRows.Clear();

\_removedRows.Clear();

\_initialRows.Clear();

foreach (T row in \_rows)

{

T copyRow = new T();

row.CopyTo(copyRow);

\_initialRows.Add(copyRow);

}

}

public IEnumerator GetEnumerator() => \_rows.GetEnumerator();

}

#endregion DbRowList

#region SQLiteDatabase

public class SQLiteDatabase : IDisposable

{

private readonly SqliteConnection connection;

public SQLiteDatabase(string database)

{

connection = new SqliteConnection(database);

connection.OpenAsync().Wait();

}

public void InitialRows<T>(DbRowList<T> initialRows) where T : DbRow, new()

{

SqliteCommand readTableQuery = new SqliteCommand($"SELECT \* FROM `{initialRows.TableName}`", connection);

using SqliteDataReader table = readTableQuery.ExecuteReader();

if (!table.HasRows)

{

return;

}

List<T> readRows = new List<T>();

while (table.Read())

{

object[] values = new object[table.FieldCount];

table.GetValues(values);

T row = new T();

row.FillProperties(values);

readRows.Add(row);

}

initialRows.DbInitialRows(readRows);

}

public void SaveChanges<T>(DbRowList<T> rows) where T : DbRow, new()

{

Dictionary<DbRowList<T>.Change, List<T>> canges = rows.GetChanges();

string commandText = AddRowCommand(canges[DbRowList<T>.Change.Add], rows.TableName) + Environment.NewLine;

commandText += DeleteRowCommand(canges[DbRowList<T>.Change.Delete], rows.TableName) + Environment.NewLine;

commandText += UpdateRowCommand(canges[DbRowList<T>.Change.Edit], rows.TableName);

SqliteCommand query = new SqliteCommand(commandText, connection);

query.ExecuteNonQuery();

rows.ClearChanges();

}

public async Task SaveChangesAsync<T>(DbRowList<T> rows) where T : DbRow, new()

{

Dictionary<DbRowList<T>.Change, List<T>> canges = rows.GetChanges();

string commandText = AddRowCommand(canges[DbRowList<T>.Change.Add], rows.TableName) + Environment.NewLine;

commandText += DeleteRowCommand(canges[DbRowList<T>.Change.Delete], rows.TableName) + Environment.NewLine;

commandText += UpdateRowCommand(canges[DbRowList<T>.Change.Edit], rows.TableName);

SqliteCommand query = new SqliteCommand(commandText, connection);

await query.ExecuteNonQueryAsync();

rows.ClearChanges();

}

private string UpdateRowCommand<T>(List<T> rows, string tableName) where T : DbRow, new()

{

SqliteCommand readTableQuery = new SqliteCommand($"SELECT \* FROM `{tableName}`", connection);

using SqliteDataReader table = readTableQuery.ExecuteReader();

string command = "";

foreach (T row in rows)

{

command += $"UPDATE `{tableName}` SET ";

object[] values = row.GetPropertiesValue();

List<int> primaryIndexes = row.GetPrimaryPropertyIndexes();

for (int i = 0; i < values.Length - 1; i++)

{

command += $"`{table.GetName(i)}` = '{values[i]}', ";

}

command += $"`{table.GetName(values.Length - 1)}` = '{values[values.Length - 1]}' WHERE ";

for (int i = 0; i < primaryIndexes.Count - 1; i++)

{

command += $"`{table.GetName(i)}` = '{values[i]}' AND ";

}

command += $"`{table.GetName(primaryIndexes.Count - 1)}` = '{values[primaryIndexes.Count - 1]}';" + Environment.NewLine;

}

return command;

}

public static string AddRowCommand<T>(List<T> rows, string tableName) where T : DbRow, new()

{

string command = "";

foreach (T row in rows)

{

object[] values = row.GetPropertiesValue();

command += $"INSERT INTO `{tableName}` VALUES (";

for (int i = 0; i < values.Length - 1; i++)

{

command += $"'{values[i]}', ";

}

command += $"'{values[^1]}');" + Environment.NewLine;

}

return command;

}

public string DeleteRowCommand<T>(List<T> rows, string tableName) where T : DbRow, new()

{

string command = "";

SqliteCommand readTableQuery = new SqliteCommand($"SELECT \* FROM `{tableName}`", connection);

using SqliteDataReader table = readTableQuery.ExecuteReader();

foreach (T row in rows)

{

object[] values = row.GetPropertiesValue();

List<int> primaryIndexes = row.GetPrimaryPropertyIndexes();

command += $"DELETE FROM `{tableName}` WHERE ";

for (int i = 0; i < primaryIndexes.Count - 1; i++)

{

command += $"`{table.GetName(i)}` = '{values[i]}' AND ";

}

command += $"`{table.GetName(primaryIndexes.Count - 1)}` = '{values[primaryIndexes.Count - 1]}';" + Environment.NewLine;

}

return command;

}

public void Dispose()

{

connection.CloseAsync();

GC.SuppressFinalize(this);

}

}

#endregion SQLiteDatabase

#region DbRow

public class DbRow

{

public void FillProperties(object[] values)

{

PropertyInfo[] properties = this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance);

int minCountValues = Math.Min(properties.Length, values.Length);

for (int i = 0; i < minCountValues; i++)

{

if (properties[i].PropertyType == values[i].GetType())

{

properties[i].SetValue(this, values[i]);

}

else if (properties[i].PropertyType == typeof(string))

{

properties[i].SetValue(this, values[i]);

}

}

}

public object[] GetPropertiesValue()

{

PropertyInfo[] propertys = this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance);

object[] values = new object[propertys.Length];

for (int i = 0; i < propertys.Length; i++)

{

values[i] = propertys[i].GetValue(this);

}

return values;

}

public List<int> GetPrimaryPropertyIndexes()

{

List<int> indexes = new List<int>();

PropertyInfo[] propertys = this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance);

for (int i = 0; i < propertys.Length; i++)

{

/\* if (property[i].Name.Length < 3)

{

continue;

}

if (property[i].Name[0] == 'p' && property[i].Name[1] == 'r')

{

indexes.Add(i);

}\*/

Attribute? attribute = propertys[i].GetCustomAttribute(typeof(PrimaryAttribute));

if (attribute != null)

{

indexes.Add(i);

}

}

return indexes;

}

public List<int> GetAutoIncrementPropertyIndexes()

{

List<int> autoIncrementPropertys = new List<int>();

PropertyInfo[] propertys = this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance);

for (int i = 0; i < propertys.Length; i++)

{

if (propertys[i].PropertyType != typeof(Int64))

{

continue;

}

Attribute? attribute = propertys[i].GetCustomAttribute(typeof(AutoIncrementAttribute));

if (attribute != null)

{

autoIncrementPropertys.Add(i);

}

}

return autoIncrementPropertys;

}

/// <summary>

/// Проверяет является ли объект той же строкой по уникальным ключам.

/// </summary>

public bool Same(DbRow row)

{

if (this.GetType() != row.GetType())

{

return false;

}

bool isEqual = false;

PropertyInfo[] property = this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance);

List<int> primaryIndexes = GetPrimaryPropertyIndexes();

foreach (int index in primaryIndexes)

{

if (property[index].GetValue(this).ToString() == property[index].GetValue(row).ToString())

{

isEqual = true;

}

else

{

isEqual = false;

break;

}

}

return isEqual;

}

public override bool Equals(object row)

{

if (this.GetType() != row.GetType())

{

return false;

}

bool isEqual = false;

foreach (PropertyInfo property in this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance))

{

if (property.GetValue(this).ToString() == property.GetValue(row).ToString())

{

isEqual = true;

}

else

{

isEqual = false;

break;

}

}

return isEqual;

}

public void CopyTo(DbRow row)

{

foreach (PropertyInfo property in this.GetType().GetProperties(BindingFlags.Public | BindingFlags.Instance))

{

property.SetValue(row, property.GetValue(this));

}

}

}

#endregion DbRow

#region Attributes

[AttributeUsage(AttributeTargets.Property)]

public class PrimaryAttribute : Attribute

{

}

// INotifyPropertyChanged

// ReactiveUI, Fody - библиотеки

[AttributeUsage(AttributeTargets.Property)]

public class AutoIncrementAttribute : Attribute

{

}

//Roslyn, introduction

#endregion Attributes

}

[--- Конец Presenter.cs]

[Начало Presenter.cs ---]

using System;

using System.Collections.Generic;

using System.Text;

using UI;

using Database;

using Files;

namespace Presenter

{

internal class PresenterData

{

private readonly IMainForm \_mainForm;

private readonly IInteractionInformation \_interactionInformation;

private readonly SQLiteDatabase \_database;

private readonly DbRowList<Student> \_students;

public PresenterData(IMainForm view, IInteractionInformation interactionInformation)

{

\_interactionInformation = interactionInformation;

\_mainForm = view;

InitialiseMainForm();

\_database = new SQLiteDatabase("Data Source=D:\\Programs\\SQLite\\Students.db");

\_students = new DbRowList<Student>("Student");

UpdateDataGridView(this, new EventArgs());

}

private void InitialiseMainForm()

{

\_mainForm.SaveFileDialog.Filter = "Text files(\*.txt)|\*.txt|Excel files(\*.xlsx)| \*.xlsx| All files(\*.\*) | \*.\*";

\_mainForm.DataGridView.ColumnCount = 4;

\_mainForm.DataGridView.Columns[0].HeaderText = "ФИО";

\_mainForm.DataGridView.Columns[1].HeaderText = "Курс";

\_mainForm.DataGridView.Columns[2].HeaderText = "Группа";

\_mainForm.DataGridView.Columns[3].HeaderText = "Форма обучения";

\_mainForm.DataGridView.Columns[1].Width = 100;

\_mainForm.DataGridView.Columns[2].Width = 100;

\_mainForm.DataGridView.Columns[3].Width = 100;

\_mainForm.UpdateEvent += UpdateDataGridView;

\_mainForm.AddEvent += AddRecord;

\_mainForm.DeleteEvent += DeleteRecord;

\_mainForm.EditEvent += EditRecord;

\_mainForm.SaveDataEvent += SaveRecords;

\_mainForm.SaveDataToDbEvent += SaveToDb;

\_mainForm.ViewClosingEvent += Closing;

}

private void UpdateDataGridView(object sender, EventArgs e)

{

if (\_students.IsChanged())

{

System.Windows.Forms.DialogResult res =

System.Windows.Forms.MessageBox.Show("Изменения не были сохранены. Продолжить?",

"Сообщение",

System.Windows.Forms.MessageBoxButtons.YesNo); // Здесь или в форме? Как если в форме? Создавать переменную в интерфейсе для сообщения?

if (res == System.Windows.Forms.DialogResult.No)

{

return;

}

}

\_mainForm.DataGridView.Rows.Clear();

\_database.InitialRows(\_students);

foreach (Student student in \_students)

{

\_mainForm.DataGridView.Rows.Add(student.Name, student.Course, student.Group, student.Form);

}

}

private void AddRecord(object sender, EventArgs e)

{

System.Windows.Forms.DialogResult result = \_interactionInformation.ShowInteractionInformation();

if (result == System.Windows.Forms.DialogResult.OK)

{

string name = \_interactionInformation.StudentName;

string course = \_interactionInformation.Course;

string group = \_interactionInformation.Group;

string form = \_interactionInformation.Form;

\_students.Add(new Student(name, course, group, form));

\_mainForm.DataGridView.Rows.Add(name, course, group, form);

}

\_interactionInformation.Clear();

}

private void DeleteRecord(object sender, EventArgs e)

{

int selectRowIndex = \_mainForm.DataGridView.SelectedRows[0].Index;

\_students.RemoveAt(selectRowIndex);

\_mainForm.DataGridView.Rows.RemoveAt(selectRowIndex);

}

private void EditRecord(object sender, EventArgs e)

{

int selectRowIndex = \_mainForm.DataGridView.SelectedRows[0].Index;

Student student = \_students[selectRowIndex];

System.Windows.Forms.DialogResult result =

\_interactionInformation.ShowInteractionInformation(student.Name, (int)student.Course, (int)student.Group, student.Form);

if (result == System.Windows.Forms.DialogResult.OK)

{

student.Name = \_interactionInformation.StudentName;

student.Course = Convert.ToInt32(\_interactionInformation.Course);

student.Group = Convert.ToInt32(\_interactionInformation.Group);

student.Form = \_interactionInformation.Form;

\_mainForm.DataGridView.Rows[selectRowIndex].SetValues(student.Name, student.Course, student.Group, student.Form);

}

\_interactionInformation.Clear();

}

private void SaveToDb(object sender, EventArgs e)

{

if (\_students.IsChanged())

{

//\_database.SaveChangesAsync(\_students);

\_database.SaveChanges(\_students);

}

UpdateDataGridView(this, new EventArgs());

}

private void SaveRecords(object sender, EventArgs e)

{

System.Windows.Forms.DialogResult result = \_mainForm.SaveFileDialog.ShowDialog();

if (result == System.Windows.Forms.DialogResult.OK)

{

int countColumns = \_mainForm.DataGridView.Columns.Count;

int countRows = \_mainForm.DataGridView.Rows.Count;

string[,] studentData = new string[countRows, countColumns];

for (int i = 0; i < countRows; i++)

{

for (int j = 0; j < countColumns; j++)

{

studentData[i, j] = \_mainForm.DataGridView.Rows[i].Cells[j].Value.ToString();

}

}

string filter = System.IO.Path.GetExtension(\_mainForm.SaveFileDialog.FileName);

if (filter == ".xlsx")

{

ExcelFile.Write(\_mainForm.SaveFileDialog.FileName, studentData);

}

else if (filter == ".txt")

{

TxtFile.Write(\_mainForm.SaveFileDialog.FileName, studentData);

}

else

{

TxtFile.Write(\_mainForm.SaveFileDialog.FileName, studentData); ;

}

}

}

private void Closing(object sender, EventArgs e)

{

if (\_students.IsChanged())

{

System.Windows.Forms.DialogResult res =

System.Windows.Forms.MessageBox.Show("Изменения не были сохранены. Продолжить?",

"Сообщение",

System.Windows.Forms.MessageBoxButtons.YesNo);

if (res == System.Windows.Forms.DialogResult.No)

{

System.Windows.Forms.FormClosingEventArgs closeEvent = (System.Windows.Forms.FormClosingEventArgs)e;

closeEvent.Cancel = true;

}

}

}

}

}

[--- Конец SQLiteDatabase.cs]

[Начало Student.cs ---]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Database

{

public class Student : DbRow

{

[Primary]

[AutoIncrement]

public Int64 Id { get; set; }

//Guid, uuid

public string Name { get; set; }

public Int64 Course { get; set; }

public Int64 Group { get; set; }

public string Form { get; set; }

public Student()

{}

public Student(int id, string name, int course, int group, string form)

{

Id = id;

Name = name;

Course = course;

Group = group;

Form = form;

}

public Student(string id, string name, string course, string group, string form)

{

Id = Convert.ToInt32(id);

Name = name;

Course = Convert.ToInt32(course);

Group = Convert.ToInt32(group);

Form = form;

}

public Student(string name, int course, int group, string form)

{

Name = name;

Course = course;

Group = group;

Form = form;

}

public Student(string name, string course, string group, string form)

{

Name = name;

Course = Convert.ToInt32(course);

Group = Convert.ToInt32(group);

Form = form;

}

}

}

[--- Конец Student.cs]

[Начало Container.cs ---]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using Autofac;

using UI;

namespace Containers

{

internal static class ContainerRegistretion

{

public static IContainer Container { get; private set; }

static ContainerRegistretion()

{

ContainerBuilder builder = new ContainerBuilder();

builder.RegisterType<InteractionInformation>().As<IInteractionInformation>();

builder.RegisterType<MainForm>().As<IMainForm>().As<MainForm>();

//builder.RegisterType<Form>();

Container = builder.Build();

}

}

}

[--- Конец Container.cs]

[Начало SQLiteDatabaseTests.cs ---]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Database;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Database.Tests

{

[TestClass()]

public class SQLiteDatabaseTests

{

[TestMethod()]

public void InitialRowsTest()

{

SQLiteDatabase database = new SQLiteDatabase("Data Source=D:\\Programs\\SQLite\\Students.db");

DbRowList<Student> students = new DbRowList<Student>("Student");

database.InitialRows(students);

students.Add(new Student(0, "some", 0, 0, "some"));

database.SaveChanges(students);

DbRowList<Student> studentsRes = new DbRowList<Student>("Student");

database.InitialRows(studentsRes);

bool isEquals = false;

if (students.GetLast().Equals(studentsRes.GetLast()))

{

isEquals = true;

}

students.Remove(students.GetLast());

database.SaveChanges(students);

Assert.IsTrue(isEquals);

}

}

}

[--- Конец SQLiteDatabaseTests.cs]

[Начало IMainForm.cs ---]

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

namespace UI

{

internal interface IMainForm

{

event EventHandler DeleteEvent;

event EventHandler EditEvent;

event EventHandler AddEvent;

event EventHandler UpdateEvent;

event EventHandler SaveDataEvent;

event EventHandler SaveDataToDbEvent;

event EventHandler ViewClosingEvent;

DataGridView DataGridView { get; }

SaveFileDialog SaveFileDialog { get; }

}

}

[--- Конец IMainForm.cs]

[Начало MainForm.cs ---]

using System;

using System.Windows.Forms;

using Presenter;

using StudentsDataBaseProject.Properties;

using Containers;

using Autofac;

namespace UI

{

public partial class MainForm : Form, IMainForm

{

private readonly AboutForm \_aboutForm;

public DataGridView DataGridView

{

get { return studentsDataGridView; }

}

public SaveFileDialog SaveFileDialog

{

get { return saveFileDialog; }

}

public event EventHandler DeleteEvent;

public event EventHandler EditEvent;

public event EventHandler AddEvent;

public event EventHandler UpdateEvent;

public event EventHandler SaveDataEvent;

public event EventHandler SaveDataToDbEvent;

public event EventHandler ViewClosingEvent;

public MainForm(IInteractionInformation interactionInformation)

{

InitializeComponent();

PresenterData presenter = new PresenterData(this, interactionInformation);

//PresenterData presenter = new PresenterData(this, new InteractionInformation()); // Мы не ссылаемся на презентер, почему он не очищается

\_aboutForm = new AboutForm();

if (Settings.Default.isShowAboutMenu)

{

\_aboutForm.Show();

}

}

private void deleteButton\_Click(object sender, EventArgs e)

{

if (studentsDataGridView.SelectedRows.Count < 1)

{

MessageBox.Show("Выделите строку.", "Сообщение");

return;

}

DialogResult res = MessageBox.Show("Вы уверены, что хотите удалить строку?", "Сообщение", MessageBoxButtons.YesNo);

if (res == DialogResult.Yes)

{

DeleteEvent.Invoke(sender, e);

}

}

private void editButton\_Click(object sender, EventArgs e)

{

if (studentsDataGridView.SelectedRows.Count < 1)

{

MessageBox.Show("Выделите строку.", "Сообщение");

return;

}

EditEvent.Invoke(sender, e);

}

private void addButton\_Click(object sender, EventArgs e)

{

AddEvent.Invoke(sender, e);

}

private void updateToolStripMenuItem\_Click(object sender, EventArgs e)

{

UpdateEvent.Invoke(sender, e);

}

private void aboutToolStripMenuItem\_Click(object sender, EventArgs e)

{

\_aboutForm.ShowDialog();

}

private void saveToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveDataEvent.Invoke(sender, e);

}

private void saveDataToDbToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveDataToDbEvent.Invoke(sender, e);

}

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

{

ViewClosingEvent.Invoke(sender, e);

}

}

}

[--- Конец MainForm.cs]

[Начало IInteractionInformation.cs ---]

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace UI

{

public interface IInteractionInformation

{

string StudentName { get; }

string Course { get; }

string Group { get; }

string Form { get; }

DialogResult ShowInteractionInformation();

DialogResult ShowInteractionInformation(string name, int course, int group, string form);

void Clear();

}

}

[--- Конец IInteractionInformation.cs]

[Начало InteractionInformation.cs ---]

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;

namespace UI

{

public partial class InteractionInformation : Form, IInteractionInformation

{

public string StudentName { get; private set; }

public string Course { get; private set; }

public string Group { get; private set; }

public string Form { get; private set; }

public InteractionInformation()

{

InitializeComponent();

courseComboBox.Items.Add("1");

courseComboBox.Items.Add("2");

courseComboBox.Items.Add("3");

courseComboBox.Items.Add("4");

courseComboBox.Items.Add("5");

courseComboBox.SelectedIndex = 0;

formComboBox.Items.Add("Очная");

formComboBox.Items.Add("Заочная");

formComboBox.Items.Add("Очно-заочная");

formComboBox.SelectedIndex = 0;

}

private void nextButton\_Click(object sender, EventArgs e)

{

StudentName = nameComboBox.Text + " " + surnameСomboBox.Text + " " + middleNameComboBox.Text;

Course = courseComboBox.Text;

Form = formComboBox.Text;

if (int.TryParse(groupComboBox.Text, out int group))

{

Group = groupComboBox.Text;

}

else

{

MessageBox.Show("Введите группу.", "Нет данных");

return;

}

DialogResult = DialogResult.OK;

this.Hide();

}

private void cancelButton\_Click(object sender, EventArgs e)

{

DialogResult = DialogResult.No;

this.Hide();

}

public DialogResult ShowInteractionInformation()

{

return this.ShowDialog();

}

public DialogResult ShowInteractionInformation(string name, int course, int group, string form)

{

string[] splitName = name.Split(" ");

try

{

nameComboBox.Text = splitName[0];

surnameСomboBox.Text = splitName[1];

middleNameComboBox.Text = splitName[2];

}

catch (IndexOutOfRangeException )

{

}

courseComboBox.Text = course.ToString();

groupComboBox.Text = group.ToString();

formComboBox.Text = form;

return this.ShowDialog();

}

private void groupComboBox\_TextUpdate(object sender, EventArgs e)

{

for (int i = 0; i < groupComboBox.Text.Length; i++)

{

if (!char.IsDigit(groupComboBox.Text[i]))

{

groupComboBox.Text = groupComboBox.Text.Remove(i, 1);

i--;

groupComboBox.SelectionStart = i + 1;

}

}

}

public void Clear()

{

nameComboBox.Text = "";

surnameСomboBox.Text = "";

middleNameComboBox.Text = "";

courseComboBox.SelectedIndex = 0;

groupComboBox.Text = "";

formComboBox.SelectedIndex = 0;

}

}

}

[--- Конец InteractionInformation.cs]

[Начало AboutForm.cs ---]

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using StudentsDataBaseProject.Properties;

namespace UI

{

public partial class AboutForm : Form

{

public AboutForm()

{

InitializeComponent();

checkBox.Checked = !Settings.Default.isShowAboutMenu;

}

private void okButton\_Click(object sender, EventArgs e)

{

Settings.Default.isShowAboutMenu = !checkBox.Checked;

Settings.Default.Save();

this.Close();

}

private void AboutForm\_FormClosing(object sender, FormClosingEventArgs e)

{

e.Cancel = true;

this.Hide();

}

}

}

[--- Конец AboutForm.cs]