Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационных технологий и

анализа данных

|  |
| --- |
| наименование института |
| **Отчет** по лабораторной работе №2  по дисциплине «Объектно-ориентированные базы данных»  «Perst» | | |  |

наименование темы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | |
| Выполнил студент |  | ИСМб-19-1 | |  |  |  | Малиновцев И.А. |
| Проверил |  | шифр группы | |  | подпись |  | И.О. Фамилия  В.А. Харахинов |
|  |  |  | |  | подпись |  | И.О. Фамилия |
| Работа защищена с оценкой | | |  | |  | | |

**Содержание**

[**1 Задание** 3](#_Toc119675439)

[**2 Классы** 4](#_Toc119675440)

[**3 Работа программы** 29](#_Toc119675441)

# **1 Задание**

**Город**

Предметная область включает в себя Жителей, Комнаты, Квартиры и Здания.

Необходимо реализовать CRUD функциональность для данной предметной области.

**Выходные документы:**

* Список жителей, чей возраст не менее заданной величины
* Список всех комнат конкретного типа с площадью не менее заданной.

# **2 Классы**

В данной лабораторной работе необходимо было разработать 6 классов: Building, DataBase, Flat, Resident, Room и RootClass на языке C# c использованием библиотеки Perst.

Код классов представлен ниже.

*Класс Room:*

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Perst.Classes

{

internal class Room : Persistent

{

[DisplayName("Id")]

public string id { get; set; }

[DisplayName("Комната")]

public RoomType RoomType { get; set; }

[DisplayName("Площадь")]

public int Area { get; set; }

public Room(RoomType roomType, int area)

{

this.id = Guid.NewGuid().ToString();

RoomType = roomType;

Area = area;

}

public Room()

{

id = Guid.NewGuid().ToString();

RoomType = RoomType.Kitchen;

Area = 0;

}

public string PrintRoomType()

{

switch ((int)RoomType)

{

case 1:

{

return "Кухня";

}

case 2:

{

return "Зал";

}

case 3:

{

return "Ванная";

}

case 4:

{

return "Прихожая";

}

default:

{

return "Жилая комната";

}

}

}

public override bool Equals(object obj)

{

return obj is Room room &&

RoomType == room.RoomType &&

Area == room.Area;

}

public override string ToString()

{

return $"Id: {id}\nТип комнаты: {PrintRoomType()}\nПлощадь: {Area}";

}

}

enum RoomType : int

{

Kitchen = 1,

Hall = 2,

Bathroom = 3,

EntranceHall = 4,

LivingRoom = 5

}

}

*Класс Resident:*

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Perst.Classes

{

internal class Resident : Persistent

{

[DisplayName("Id")]

public string id { get; set; }

[DisplayName("Имя")]

public string Name { get; set; }

[DisplayName("Фамилия")]

public string LastName { get; set; }

//Добавить

[DisplayName("Дата рождения")]

public DateTime BirthDate { get; set; }

public Resident(string name, string lastName, DateTime birthDate)

{

id = Guid.NewGuid().ToString();

Name = name;

LastName = lastName;

BirthDate = birthDate;

}

public Resident()

{

id = Guid.NewGuid().ToString();

Name = "Name";

LastName = "Lastname";

BirthDate = DateTime.Now;

}

public override bool Equals(object obj)

{

return obj is Resident resident &&

Name == resident.Name &&

LastName == resident.LastName

&& BirthDate.Date == resident.BirthDate.Date;

}

public static bool operator ==(Resident r1, Resident r2)

{

if(r1.id == r2.id

&& r1.Name == r2.Name

&& r1.LastName == r2.LastName

&& r1.BirthDate.Date == r2.BirthDate.Date)

{

return true;

}

else

{

return false;

}

}

public static bool operator !=(Resident r1, Resident r2)

{

if (r1.id != r2.id

|| r1.Name != r2.Name

|| r1.LastName != r2.Name

|| r1.BirthDate.Date != r2.BirthDate.Date)

{

return true;

}

else

{

return false;

}

}

public override string ToString()

{

return $"id: {id}\nИмя: {Name}\nФамилия: {LastName}\nДата рождения: {BirthDate}";

}

}

}

*Класс Flat:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Perst.Classes

{

internal class Flat : Persistent

{

public string id { get; }

public Link Rooms { get; set; }

public Link Residents { get; set; }

public Flat(Link rooms, Link residents)

{

id = Guid.NewGuid().ToString();

this.Rooms = rooms;

this.Residents = residents;

}

public Flat()

{

id = Guid.NewGuid().ToString();

}

public override bool Equals(object obj)

{

return obj is Flat flat &&

id == flat.id &&

EqualityComparer<Link>.Default.Equals(Rooms, flat.Rooms) &&

EqualityComparer<Link>.Default.Equals(Residents, flat.Residents);

}

public override string ToString()

{

var message = "Жильцы в квартире:\n";

foreach(var item in Residents)

{

message += $"{item.ToString()}\n";

message += $"-----------------\n";

}

message += "\nКомнаты в квартире:\n";

foreach (var item in Rooms)

{

message += $"{item.ToString()}\n";

message += $"-----------------\n";

}

return message;

}

}

}

*Класс Building:*

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Perst.Classes

{

internal class Building : Persistent

{

[DisplayName("Id")]

public string id { get; }

[DisplayName("Адрес")]

public string Address { get; set; }

public Link Flats { get; set; }

public Building(string address, Link flats)

{

id = Guid.NewGuid().ToString();

Address = address;

Flats = flats;

}

public Building()

{

id = Guid.NewGuid().ToString();

Address = "address";

}

}

}

*Класс RootClass:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Perst.Classes

{

internal class RootClass : Persistent

{

public FieldIndex BuildingId;

public FieldIndex FlatId;

public FieldIndex ResidentId;

public FieldIndex RoomId;

public FieldIndex ResidentBirthDate;

public RootClass(Storage db) : base(db)

{

BuildingId = db.CreateFieldIndex(typeof(Building), "id", true);

FlatId = db.CreateFieldIndex(typeof(Flat), "id", true);

ResidentId = db.CreateFieldIndex(typeof(Resident), "id", true);

RoomId = db.CreateFieldIndex(typeof(Room), "id", true);

ResidentBirthDate = db.CreateFieldIndex(typeof(Resident), "BirthDate", false);

}

}

}

*Класс DataBase:*

using System;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace Perst.Classes

{

internal class DataBase

{

private const string FilePatch = @"MyDB.pdb";

public DataBase()

{

db = StorageFactory.Instance.CreateStorage();

PagePoolSize = 10;

}

public DataBase(long PagePoolSize)

{

db = StorageFactory.Instance.CreateStorage();

this.PagePoolSize = PagePoolSize;

}

private Storage db;

public long PagePoolSize { get; }

public void Connect()

{

db.Open(FilePatch, PagePoolSize);

var root = (RootClass)db.Root;

if (root == null)

{

root = new RootClass(db);

db.Root = root;

}

Console.WriteLine("Connected!");

}

public void Close()

{

db.Close();

}

#region Работа с жителями

public List<Resident> GetResidents()

{

List<Resident> residents = new List<Resident>();

foreach (Resident res in (db.Root as RootClass).ResidentId)

{

residents.Add(res);

}

return residents;

}

public void AddResident(Resident resident)

{

(db.Root as RootClass).ResidentId.Put(resident);

(db.Root as RootClass).ResidentBirthDate.Put(resident);

}

public List<Resident> SearchResident(int age)

{

var date = DateTime.Now.AddYears(-1\*age);

Key key1 = new Key(date);

Key key2 = new Key(DateTime.Now);

var responce = (db.Root as RootClass).ResidentBirthDate.Range(

key1, key2, IterationOrder.DescentOrder

).Cast<Resident>().ToList();

return responce;

}

public void DeleteResident(string id)

{

var ans = IsHaveResident(id);

if(ans.Count != 0)

{

var message = "Жилец все еще проживает в квартирах:\n";

foreach(var a in ans)

{

message += $"{a.Item1}\n";

}

throw new Exception(message);

}

Resident found = (db.Root as RootClass).ResidentId[id] as Resident;

(db.Root as RootClass).ResidentId.Remove(found);

}

public List<(string, int)> IsHaveResident(string id)

{

var found = ((db.Root as RootClass).FlatId);

var respons = from flat in found.Cast<Flat>()

from resident in flat.Residents.Cast<Resident>()

where resident.id == id

group resident by flat.id into r

select new

{

Id = r.Key,

Count = r.Count()

}; ;

List < (string, int) > values = new List<(string, int)>();

foreach(var res in respons)

{

values.Add((res.Id, res.Count));

}

return values;

}

public void UpdateResident(Resident resident)

{

Resident found = (db.Root as RootClass).ResidentId[resident.id] as Resident;

found.Name = resident.Name;

found.LastName = resident.LastName;

found.Modify();

}

#endregion

#region Работа с комнатами

public List<Room> GetRooms()

{

List<Room> rooms = new List<Room>();

foreach (Room room in (db.Root as RootClass).RoomId)

{

rooms.Add(room);

}

return rooms;

}

public void DeleteRoom(string id)

{

var ans = IsHaveRoom(id);

if (ans.Count != 0)

{

var message = "Комната все еще находится в квартирах:\n";

foreach (var a in ans)

{

message += $"{a.Item1} в количестве {a.Item2}\n";

}

throw new Exception(message);

}

Room found = (db.Root as RootClass).RoomId[id] as Room;

(db.Root as RootClass).RoomId.Remove(found);

}

public void AddRoom(Room room)

{

(db.Root as RootClass).RoomId.Put(room);

}

public void UpdateRoom(Room room)

{

Room found = (db.Root as RootClass).RoomId[room.id] as Room;

found.RoomType = room.RoomType;

found.Area = room.Area;

found.Modify();

}

public List<(string, int)> IsHaveRoom(string id)

{

var found = ((db.Root as RootClass).FlatId);

var respons = from flat in found.Cast<Flat>()

from room in flat.Rooms.Cast<Room>()

where room.id == id

group room by flat.id into r

select new

{

Id = r.Key,

Count = r.Count()

}; ;

List<(string, int)> values = new List<(string, int)>();

foreach (var res in respons)

{

values.Add((res.Id, res.Count));

}

return values;

}

public List<Room> RoomSearch(int area, RoomType roomType)

{

var room = new Room();

room.Area = area;

room.RoomType = roomType;

var response =(db.Root as RootClass).RoomId.Select(typeof(Room), $"Area <= {room.Area} " +

$"and PrintRoomType like '{room.PrintRoomType()}' ").Cast<Room>().ToList();

return response;

}

#endregion

#region Работа с квартирами

public List<(string, int)> IsHaveFlat(string id)

{

var found = ((db.Root as RootClass).BuildingId);

var respons = from building in found.Cast<Building>()

from flats in building.Flats.Cast<Flat>()

where flats.id == id

group flats by building.id into r

select new

{

Id = r.Key,

Count = r.Count()

}; ;

List<(string, int)> values = new List<(string, int)>();

foreach (var res in respons)

{

values.Add((res.Id, res.Count));

}

return values;

}

public Flat AddFlat(List<Room> rooms, List<Resident> residents)

{

var flat = new Flat();

flat.Rooms = db.CreateLink();

foreach(var room in rooms)

{

flat.Rooms.Add(room);

flat.Modify();

}

flat.Residents = db.CreateLink();

foreach(var resident in residents)

{

flat.Residents.Add(resident);

flat.Modify();

}

(db.Root as RootClass).FlatId.Put(flat);

return flat;

}

public List<Flat> GetFlats()

{

List<Flat> flats = new List<Flat>();

foreach (Flat flat in (db.Root as RootClass).FlatId)

{

flats.Add(flat);

}

return flats;

}

public Flat GetFlat(string id) => (db.Root as RootClass).FlatId[id] as Flat;

public void DeleteFlat(string id)

{

var ans = IsHaveFlat(id);

if (ans.Count != 0)

{

var message = "Квартира все еще используется в домах:\n";

foreach (var a in ans)

{

message += $"{a.Item1}\n";

}

throw new Exception(message);

}

Flat found = (db.Root as RootClass).FlatId[id] as Flat;

(db.Root as RootClass).FlatId.Remove(found);

}

public Flat UpdateFlat(string id, List<Room> rooms, List<Resident> residents) {

var foundFlat = (db.Root as RootClass).FlatId[id] as Flat;

foundFlat.Rooms.Clear();

foundFlat.Residents.Clear();

foreach (var room in rooms)

{

foundFlat.Rooms.Add(room);

foundFlat.Modify();

}

foreach (var resident in residents)

{

foundFlat.Residents.Add(resident);

foundFlat.Modify();

}

foundFlat.Modify();

return foundFlat;

}

#endregion

#region Работа со зданиями

public List<Building> GetBuildings() => (db.Root as RootClass).BuildingId.Cast<Building>().ToList();

public Building GetBuildings(string id) => ((db.Root as RootClass).BuildingId[id] as Building);

public Building AddBuilding(string address, List<Flat> flats)

{

var building = new Building();

building.Address = address;

building.Flats = db.CreateLink();

foreach(var flat in flats)

{

building.Flats.Add(flat);

building.Modify();

}

(db.Root as RootClass).BuildingId.Put(building);

return building;

}

public void DeleteBuilding(string id)

{

Building building = (db.Root as RootClass).BuildingId[id] as Building;

(db.Root as RootClass).BuildingId.Remove(building);

}

public Building UpdateBuilding(string id, string address, List<Flat> flats)

{

var found = (db.Root as RootClass).BuildingId[id] as Building;

found.Address = address;

found.Flats.Clear();

foreach (var flat in flats)

{

found.Flats.Add(flat);

found.Modify();

}

found.Modify();

return found;

}

#endregion

}

}

Также, были разработаны формы: Form1 для общей работы с данными.

Листинг Form1:

using Perst.Classes;

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace Perst

{

public partial class Form1 : Form

{

DataBase dataBase;

public Form1()

{

InitializeComponent();

dataBase = new DataBase();

dataBase.Connect();

SelectSource();

ComboboxInit();

TableInit();

dtpResident.MaxDate = DateTime.Now;

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

dataBase.Close();

}

private void TableInit()

{

if (dgvFlat.Rows.Count > 0)

{

var flat = dataBase.GetFlat(dgvFlat.SelectedCells[0].Value.ToString());

var rooms = flat.Rooms.ToArray();

var residents = flat.Residents.ToArray();

foreach (Room room in rooms)

{

roomFlatBindingSource.Add(room);

}

foreach (Resident resident in residents)

{

residentFlatBindingSource.Add(resident);

}

}

if (dgvBuilding.Rows.Count > 0)

{

var building = dataBase.GetBuildings(dgvBuilding.SelectedCells[0].Value.ToString());

var flats = building.Flats.ToArray();

foreach (Flat f in flats)

{

buildingFlatBindingSource.Add(f);

}

txtBuildingAddress.Text = building.Address;

}

}

private void ComboboxInit()

{

cmbRoomType.DataSource = Enum.GetValues(typeof(RoomType));

(dgvRoom.Columns[1] as DataGridViewComboBoxColumn).DataSource = Enum.GetValues(typeof(RoomType));

rtbFlatRoomInfo.Text = (cmbFlatRoom.SelectedItem as Room)?.ToString();

rtbFlatResidentInfo.Text = (cmbFlatResident.SelectedItem as Resident)?.ToString();

rtbBuildingFlatInfo.Text = (cmbBuildingFlat.SelectedItem as Flat)?.ToString();

cmbRoomSearch.DataSource = Enum.GetValues(typeof(RoomType));

}

private void SelectSource()

{

foreach(var res in dataBase.GetResidents())

{

residentBindingSource.Add(res);

}

foreach (var room in dataBase.GetRooms())

{

roomBindingSource.Add(room);

}

foreach (var flat in dataBase.GetFlats())

{

flatBindingSource.Add(flat);

}

foreach(var building in dataBase.GetBuildings())

{

buildingBindingSource.Add(building);

}

}

#region Утилиты

void messageBoxSuccessAdd()

{

MessageBox.Show("Запись успешно добавлена",

"Успех", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

void messageBoxError(string message)

{

MessageBox.Show(message,

"Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

DialogResult messageBoxClickResult(string message)

{

var dialogResult = MessageBox.Show(message,

"Предупреждение",

MessageBoxButtons.YesNo,

MessageBoxIcon.Question);

return dialogResult;

}

//Отменение изменений

private void tableValidating(object sender, DataGridViewCellValidatingEventArgs e)

{

if (string.IsNullOrEmpty(e.FormattedValue.ToString()))

{

((DataGridView)sender).CancelEdit();

((DataGridView)sender).EndEdit();

}

}

#endregion

#region Работа с жителелями

private void cbResident\_CheckedChanged(object sender, EventArgs e)

{

if (cbResident.Checked)

{

dgvResident.Columns[1].ReadOnly = false;

dgvResident.Columns[2].ReadOnly = false;

dgvResident.Columns[4].Visible = true;

}

else

{

dgvResident.Columns[1].ReadOnly = true;

dgvResident.Columns[2].ReadOnly = true;

dgvResident.Columns[4].Visible = false;

}

}

private void dgvResident\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvResident.Columns[e.ColumnIndex].Index == dgvResident.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

var id = dgvResident.Rows[e.RowIndex].Cells[0].Value.ToString();

try

{

dataBase.DeleteResident(id);

residentBindingSource.RemoveAt(e.RowIndex);

}

catch(Exception ex)

{

messageBoxError(ex.Message);

}

}

}

}

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

{

if (string.IsNullOrEmpty(txtResidentName.Text))

{

messageBoxError("Вы не ввели имя");

return;

}

if (string.IsNullOrEmpty(txtResidentLastname.Text))

{

messageBoxError("Вы не ввели фамилию");

return;

}

Resident resident = new Resident();

resident.Name = txtResidentName.Text;

resident.LastName = txtResidentLastname.Text;

resident.BirthDate = dtpResident.Value;

dataBase.AddResident(resident);

txtResidentName.Clear();

txtResidentLastname.Clear();

residentBindingSource.Add(resident);

messageBoxSuccessAdd();

}

Resident selectedResident;

private void dgvResident\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

selectedResident = new Resident();

selectedResident.id = dgvResident.Rows[e.RowIndex].Cells[0].Value.ToString();

selectedResident.Name = dgvResident.Rows[e.RowIndex].Cells[1].Value.ToString();

selectedResident.LastName = dgvResident.Rows[e.RowIndex].Cells[2].Value.ToString();

}

private void dgvResident\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

Resident resident = new Resident();

resident.Name = dgvResident.Rows[e.RowIndex].Cells[1].Value.ToString();

resident.LastName = dgvResident.Rows[e.RowIndex].Cells[2].Value.ToString();

if (resident.Equals(selectedResident))

{

return;

}

var dialogResult = messageBoxClickResult("Изменить эту запись?");

if (dialogResult == DialogResult.No)

{

dgvResident[1, e.RowIndex].Value = selectedResident.Name;

dgvResident[2, e.RowIndex].Value = selectedResident.LastName;

return;

}

if (dialogResult == DialogResult.Yes)

{

dataBase.UpdateResident(dgvResident.CurrentRow.DataBoundItem as Resident);

}

}

#endregion

#region Работа с комнатами

private void cbRoom\_CheckedChanged(object sender, EventArgs e)

{

if (cbRoom.Checked)

{

dgvRoom.Columns[1].ReadOnly = false;

dgvRoom.Columns[2].ReadOnly = false;

dgvRoom.Columns[3].Visible = true;

}

else

{

dgvRoom.Columns[1].ReadOnly = true;

dgvRoom.Columns[2].ReadOnly = true;

dgvRoom.Columns[3].Visible = false;

}

}

private void dgvRoom\_DataError(object sender, DataGridViewDataErrorEventArgs e)

{

dgvRoom.Rows[e.RowIndex].Cells[2].Value = selectedRoom.Area;

messageBoxError("Введите числовое значение площади");

}

private void dgvRoom\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvRoom.Columns[e.ColumnIndex].Index == dgvRoom.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

var id = dgvRoom.Rows[e.RowIndex].Cells[0].Value.ToString();

try

{

dataBase.DeleteRoom(id);

roomBindingSource.RemoveAt(e.RowIndex);

}

catch(Exception ex)

{

messageBoxError(ex.Message);

}

}

}

}

Room selectedRoom;

private void dgvRoom\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

selectedRoom = new Room();

selectedRoom.id = dgvRoom.Rows[e.RowIndex].Cells[0].Value.ToString();

selectedRoom.RoomType = (RoomType)dgvRoom.Rows[e.RowIndex].Cells[1].Value;

selectedRoom.Area = Convert.ToInt32(dgvRoom.Rows[e.RowIndex].Cells[2].Value);

}

private void dgvRoom\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

Room room = new Room();

room.RoomType = (RoomType)dgvRoom.Rows[e.RowIndex].Cells[1].Value;

room.Area = Convert.ToInt32(dgvRoom.Rows[e.RowIndex].Cells[2].Value);

if(room.Area < 1 || room.Area > 100)

{

dgvRoom[2, e.RowIndex].Value = selectedRoom.Area;

messageBoxError("Вы ввели неверную площадь");

return;

}

if (room.Equals(selectedRoom))

{

return;

}

var dialogResult = messageBoxClickResult("Изменить эту запись?");

if (dialogResult == DialogResult.No)

{

dgvRoom[1, e.RowIndex].Value = selectedRoom.RoomType;

dgvRoom[2, e.RowIndex].Value = selectedRoom.Area;

return;

}

if (dialogResult == DialogResult.Yes)

{

dataBase.UpdateRoom(dgvRoom.CurrentRow.DataBoundItem as Room);

}

}

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

{

Room room = new Room();

room.Area = (int)nudRoomArea.Value;

room.RoomType = (RoomType)cmbRoomType.SelectedItem;

dataBase.AddRoom(room);

nudRoomArea.Value = nudRoomArea.Minimum;

roomBindingSource.Add(room);

messageBoxSuccessAdd();

}

#endregion

#region Работа с квартирами

private void cmbFlatRoom\_SelectedIndexChanged(object sender, EventArgs e)

{

rtbFlatRoomInfo.Text = (cmbFlatRoom.SelectedItem as Room)?.ToString();

}

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

{

if(roomBindingSource.Count < 1)

{

messageBoxError("Не добавлены комнаты");

return;

}

var room = (cmbFlatRoom.SelectedItem as Room);

roomFlatBindingSource.Add(room);

}

private void cmbFlatResident\_SelectedIndexChanged(object sender, EventArgs e)

{

rtbFlatResidentInfo.Text = (cmbFlatResident.SelectedItem as Resident)?.ToString();

}

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

{

if(residentBindingSource.Count < 1)

{

messageBoxError("Не добавлены жильцы");

return;

}

var resident = (cmbFlatResident.SelectedItem as Resident);

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

{

if ((Resident)residentFlatBindingSource[i] == resident)

{

messageBoxError($"Жилец: {resident.ToString()}\nуже прописан в этой квартире");

return;

}

}

residentFlatBindingSource.Add(resident);

}

private void dgvFlatRoom\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvFlatRoom.Columns[e.ColumnIndex].Index == dgvFlatRoom.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

roomFlatBindingSource.RemoveAt(e.RowIndex);

}

}

}

private void dgvFlatResident\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvFlatRoom.Columns[e.ColumnIndex].Index == dgvFlatRoom.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

residentFlatBindingSource.RemoveAt(e.RowIndex);

}

}

}

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

{

if (roomFlatBindingSource.Count < 1)

{

messageBoxError("Вы не выбрали комнаты");

return;

}

if (cbFlatDelete.Checked)

{

if (dgvFlat.CurrentCell == null)

{

messageBoxError("Вы не выбрали квартиру");

}

else

{

var result = messageBoxClickResult("Изменить эту запись?");

if (result == DialogResult.Yes)

{

var id = dgvFlat.CurrentRow.Cells[0].Value.ToString();

List<Room> rooms = new List<Room>();

foreach (Room room in roomFlatBindingSource)

{

rooms.Add(room);

}

List<Resident> residents = new List<Resident>();

foreach (Resident resident in residentFlatBindingSource)

{

residents.Add(resident);

}

dataBase.UpdateFlat(id, rooms, residents);

MessageBox.Show("Запись успешно измененена",

"Успех", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

}

}

else

{

List<Room> rooms = new List<Room>();

foreach (Room room in roomFlatBindingSource)

{

rooms.Add(room);

}

List<Resident> residents = new List<Resident>();

foreach (Resident resident in residentFlatBindingSource)

{

residents.Add(resident);

}

roomFlatBindingSource.Clear();

residentFlatBindingSource.Clear();

var flat = dataBase.AddFlat(rooms, residents);

flatBindingSource.Add(flat);

messageBoxSuccessAdd();

}

}

private void dgvFlat\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvFlat.Columns[e.ColumnIndex].Index == dgvFlat.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

var id = (string)dgvFlat.Rows[e.RowIndex].Cells[0].Value;

try

{

dataBase.DeleteFlat(id);

flatBindingSource.RemoveAt(e.RowIndex);

roomFlatBindingSource.Clear();

residentFlatBindingSource.Clear();

}

catch (Exception ex)

{

messageBoxError(ex.Message);

}

}

}

else

{

var flat = dataBase.GetFlat(dgvFlat.Rows[e.RowIndex].Cells[0].Value.ToString());

var rooms = flat.Rooms.ToArray();

var residents = flat.Residents.ToArray();

roomFlatBindingSource.Clear();

residentFlatBindingSource.Clear();

foreach (Room room in rooms)

{

roomFlatBindingSource.Add(room);

}

foreach (Resident resident in residents)

{

residentFlatBindingSource.Add(resident);

}

}

}

private void cbFlatDelete\_CheckedChanged(object sender, EventArgs e)

{

if (cbFlatDelete.Checked)

{

dgvFlat.Columns[1].Visible = true;

btnFlatAdd.Text = "Изменить квартиру";

}

else

{

dgvFlat.Columns[1].Visible = false;

btnFlatAdd.Text = "Добавить квартиру";

}

}

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

{

roomFlatBindingSource.Clear();

residentFlatBindingSource.Clear();

dgvFlat.ClearSelection();

dgvFlat.CurrentCell = null;

}

#endregion

#region Работа с домами

private void cmbBuildingFlat\_SelectedIndexChanged(object sender, EventArgs e)

{

rtbBuildingFlatInfo.Text = (cmbBuildingFlat.SelectedItem as Flat)?.ToString();

}

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

{

var flat = (cmbBuildingFlat.SelectedItem as Flat);

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

{

if((buildingFlatBindingSource[i] as Flat).id == flat.id)

{

messageBoxError($"Квартира: {flat.id}\nуже существует в этом доме");

return;

}

}

buildingFlatBindingSource.Add(flat);

}

private void dgvBuildingFlat\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvBuildingFlat.Columns[e.ColumnIndex].Index == dgvBuildingFlat.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

buildingFlatBindingSource.RemoveAt(e.RowIndex);

}

}

}

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

{

if (buildingFlatBindingSource.Count < 1)

{

messageBoxError("Вы не выбрали квартиры");

return;

}

if (string.IsNullOrEmpty(txtBuildingAddress.Text))

{

messageBoxError("Вы не ввели адрес");

return;

}

if (cbBuildingDelete.Checked)

{

if (dgvBuilding.CurrentCell == null)

{

messageBoxError("Вы не выбрали квартиру");

}

else

{

var result = messageBoxClickResult("Изменить эту запись?");

if (result == DialogResult.Yes)

{

var id = dgvBuilding.CurrentRow.Cells[0].Value.ToString();

List<Flat> flats = new List<Flat>();

foreach (Flat flat in buildingFlatBindingSource)

{

flats.Add(flat);

}

//for (int i = 0; i < flats.Count; i++)

//{

// for (int j = i + 1; j < flats.Count; j++)

// {

// if (flats[i] == flats[j])

// {

// messageBoxError($"Квартира: {flats[i].id}\nуже существует в этом доме");

// return;

// }

// }

//}

dataBase.UpdateBuilding(id, txtBuildingAddress.Text, flats);

dgvBuilding.CurrentRow.Cells[1].Value = txtBuildingAddress.Text;

MessageBox.Show("Запись успешно измененена",

"Успех", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

}

}

else

{

List<Flat> flats = new List<Flat>();

foreach (Flat flat in buildingFlatBindingSource)

{

flats.Add(flat);

}

buildingFlatBindingSource.Clear();

var building = dataBase.AddBuilding(txtBuildingAddress.Text, flats);

buildingBindingSource.Add(building);

messageBoxSuccessAdd();

}

}

private void dgvBuilding\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex < 0)

{

return;

}

if (dgvBuilding.Columns[e.ColumnIndex].Index == dgvBuilding.Columns.Count - 1)

{

if (messageBoxClickResult("Удалить эту запись?") == DialogResult.Yes)

{

var id = (string)dgvBuilding.Rows[e.RowIndex].Cells[0].Value;

buildingBindingSource.RemoveAt(e.RowIndex);

buildingFlatBindingSource.Clear();

dataBase.DeleteBuilding(id);

}

}

else

{

buildingFlatBindingSource.Clear();

var building = dataBase.GetBuildings(dgvBuilding.Rows[e.RowIndex].Cells[0].Value.ToString());

var flats = building.Flats.ToArray();

foreach (Flat f in flats)

{

buildingFlatBindingSource.Add(f);

}

txtBuildingAddress.Text = building.Address;

}

}

private void cbBuildingDelete\_CheckedChanged(object sender, EventArgs e)

{

if (cbBuildingDelete.Checked)

{

dgvBuilding.Columns[2].Visible = true;

btnBuildingAdd.Text = "Изменить дом";

}

else

{

dgvBuilding.Columns[2].Visible = false;

btnBuildingAdd.Text = "Добавить дом";

}

}

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

{

txtBuildingAddress.Clear();

buildingFlatBindingSource.Clear();

dgvBuilding.CurrentCell = null;

}

#endregion

private void nudResidentSearch\_ValueChanged(object sender, EventArgs e)

{

residentBindingSource.Clear();

foreach (var res in dataBase.SearchResident((int)nudResidentSearch.Value))

{

residentBindingSource.Add(res);

}

}

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

{

residentBindingSource.Clear();

foreach (var res in dataBase.GetResidents())

{

residentBindingSource.Add(res);

}

}

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

{

var area = (int)nudRoomSearch.Value;

var roomType = (RoomType)cmbRoomSearch.SelectedItem;

roomBindingSource.Clear();

foreach (var room in dataBase.RoomSearch(area, roomType))

{

roomBindingSource.Add(room);

}

}

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

{

roomBindingSource.Clear();

foreach (var room in dataBase.GetRooms())

{

roomBindingSource.Add(room);

}

}

}

}

# **3 Работа программы**

Ниже представлен внешний вид работающей программы (рис. 1 – 6).

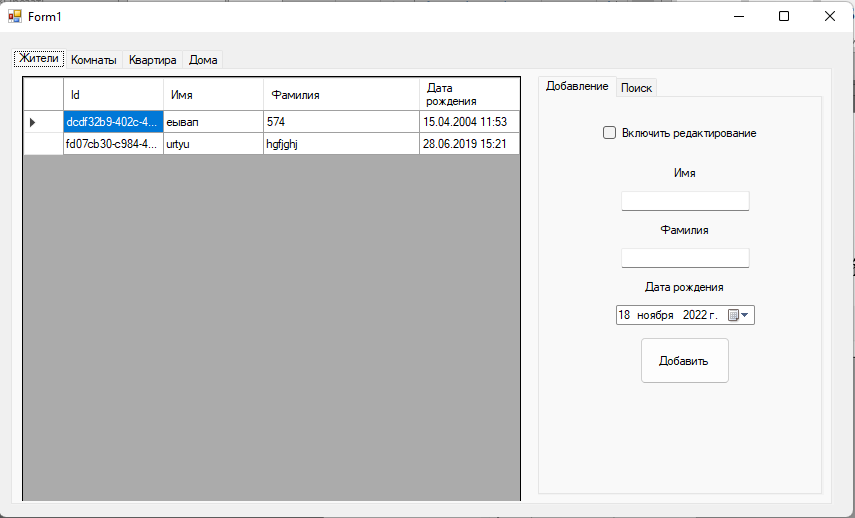


Рисунок 1 – Вкладка «Жители»

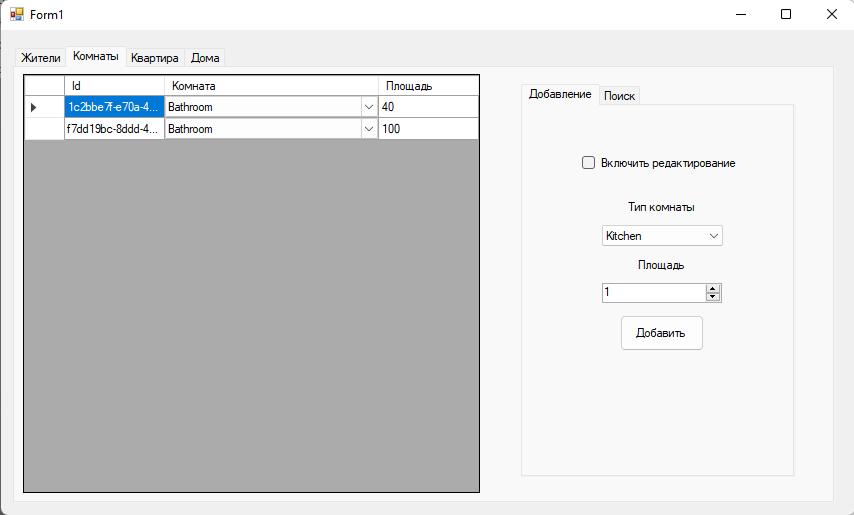


Рисунок 2 – Вкладка «Комнаты»

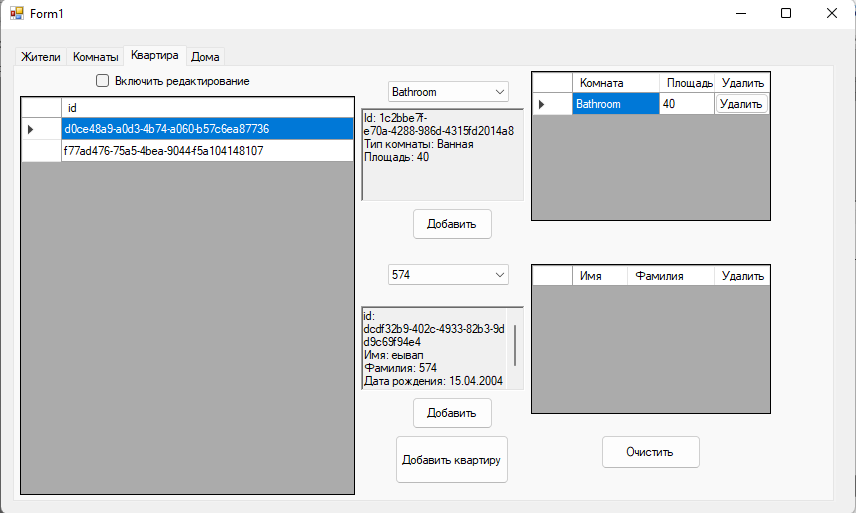


Рисунок 3 – Вкладка «Квартиры»

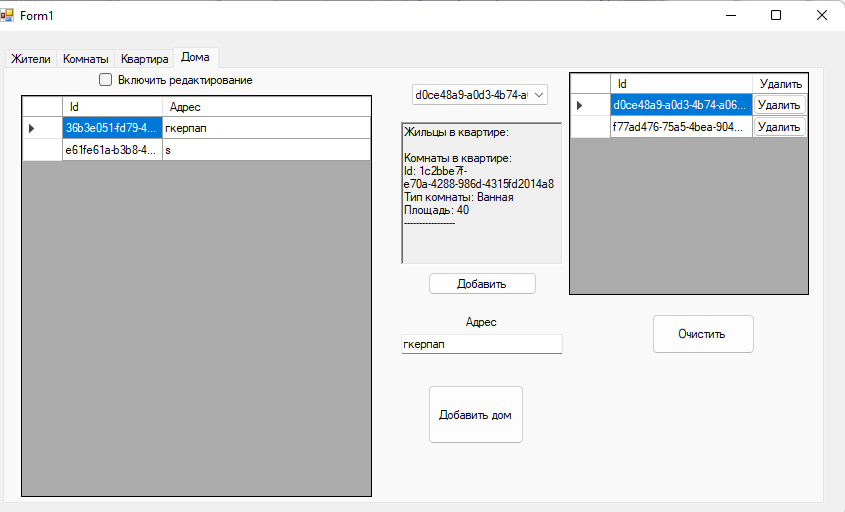


Рисунок 4 – Вкладка «Дома»

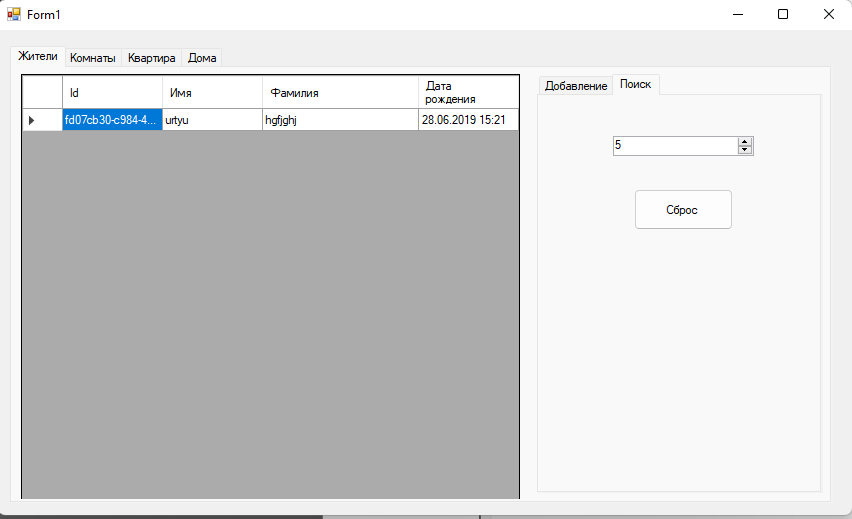


Рисунок 5 – Выдача первого документа

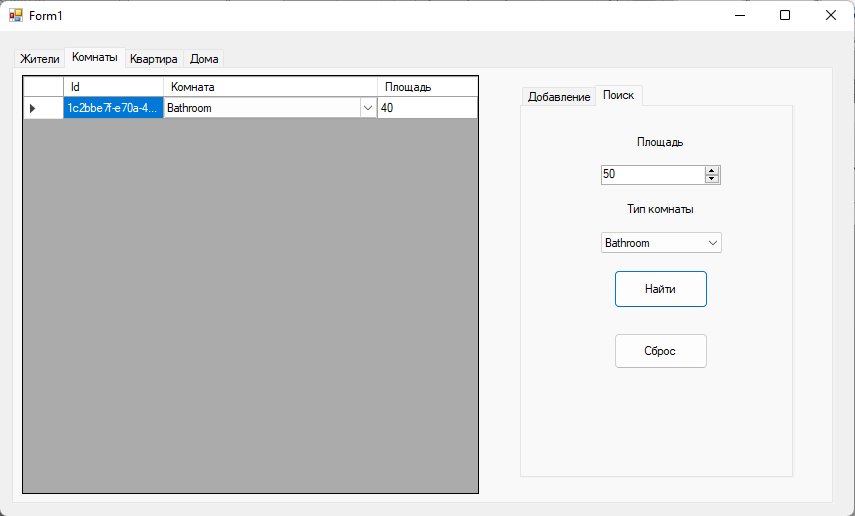


Рисунок 6 – Выдача второго документа