Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Математическое обеспечение и применение ЭВМ»

ОТЧЕТ

по лабораторной работе №2

по курсу «Объектно-ориентированное программирование» на тему

«Статические члены класса. Обработка исключений»

Выполнили: студенты группы 21ВП2

Лакеева Софья

Малькова Анастасия

Принял: к.т.н. Афонин А.Ю.

Пенза 2023

**Цель работы:**Научиться создавать приложения с графическим интерфейсом пользователя (GUI) на основе Windows Forms. Подробно ознакомиться с конструкторами объектов. Научиться использовать статические члены классов. Научиться обрабатывать исключительные ситуации в процессе работы программы. Вызов неуправляемого кода.

**Лабораторное задание:**

Дополнить класс, созданный в лабораторной работе №1. Реализовать в классе счетчик созданных объектов, задействовать статический член класса. Изменение и вывод информации в объектах класса организовать через GUI. Вывести содержимое класса на экран пользователю.

Реализовать в коде лабораторной работы перехват и обработку исключений, при этом переопределив с помощью наследования исключение:

5) InvalidCastException

Дополнить исключение своим блоком данных. Заполнить его в момент генерации. Отобразить в момент перехвата.

Реализовать свой тип исключения. Сгенерировать и выполнить его перехват.

На возникновение исключения отреагировать выводом системного окна сообщения посредством вызова неуправляемого кода: функции MessageBox из библиотеки user32.dll.

**Листинг:**

**Файл Program.cs**

using System;

using System.Windows.Forms;

namespace Lab2

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

if (Environment.OSVersion.Version.Major >= 6)

{

SetProcessDPIAware();

}

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Airport1());

}

[System.Runtime.InteropServices.DllImport("user32.dll")]

private static extern bool SetProcessDPIAware();

}

}

**Файл Airport.cs**

using System;

using System.Collections.Generic;

namespace Lab2

{

public class Airport

{

/// <summary> название аэропорта </summary>

private string \_name;

/// <summary> код ИАТА </summary>

private string \_codeIATA;

/// <summary> количество рейсов </summary>

private int \_countOfFlights;

/// <summary> количество самолётов </summary>

private int \_countOfAirplanes;

/// <summary> количество сотрудников </summary>

private int \_countOfEmployees;

/// <summary> количество пассажиров в год в млн </summary>

private float \_countOfPassengers;

/// <summary> доход за последний год в млрд.руб </summary>

private double \_revenue;

/// <summary> список объектов класса Airport </summary>

private static List<Airport> \_list = new List<Airport>();

/// <summary> количество объектов класса Airport </summary>

static private int \_count;

/// <summary> id объекта </summary>

private readonly int \_id;

/// <summary>

/// конструктор без параметров

/// </summary>

public Airport()

{

\_name = "";

\_codeIATA = "";

\_countOfFlights = 0;

\_countOfAirplanes = 0;

\_countOfEmployees = 0;

\_countOfPassengers = 0F;

\_revenue = 0D;

\_id = \_list.Count;

\_count = \_list.Count + 1;

}

/// <summary>

/// конструктор с одним параметром

/// </summary>

/// <param name="n"> Название аэропорта </param>

public Airport(string n)

{

\_name = n;

\_codeIATA = "";

\_countOfFlights = 0;

\_countOfAirplanes = 0;

\_countOfEmployees = 0;

\_countOfPassengers = 0F;

\_revenue = 0D;

\_id = \_list.Count;

\_count = \_list.Count + 1;

}

/// <summary>

/// конструктор с двумя параметрами

/// </summary>

/// <param name="n"> Название аэропорта </param>

/// <param name="code"> Код ИАТА аэропорта </param>

public Airport(string n, string code)

{

\_name = n;

\_codeIATA = code;

\_countOfFlights = 0;

\_countOfAirplanes = 0;

\_countOfEmployees = 0;

\_countOfPassengers = 0F;

\_revenue = 0D;

\_id = \_list.Count;

\_count = \_list.Count + 1;

}

/// <summary>

/// конструктор со всеми полями

/// </summary>

/// <param name="n"> Название аэропорта </param>

/// <param name="code"> Код ИАТА аэропорта</param>

/// <param name="flights"> Количество рейсов </param>

/// <param name="airplanes"> Количество самолётов </param>

/// <param name="employees"> Количество сотрудников </param>

/// <param name="passengers"> Количество пассажиров в год в млн </param>

/// <param name="r"> Доход за последний год в млрд.руб </param>

public Airport(string n, string code, int flights, int airplanes, int employees, float passengers, double r)

{

\_name = n;

\_codeIATA = code;

\_countOfFlights = flights;

\_countOfAirplanes = airplanes;

\_countOfEmployees = employees;

\_countOfPassengers = passengers;

\_revenue = r;

\_id = \_list.Count;

\_count = \_list.Count + 1;

}

/// <summary>

/// Переопределенный метод ToString

/// </summary>

/// <returns> Строка с полными данными аэропорта </returns>

public override string ToString()

{

return $"\_id: {\_id}" + "\r" + "\n" + $"\_name: {\_name}" + "\r" + "\n" + $"\_codeIATA: {\_codeIATA}"

+ "\r" + "\n" + $"\_countOfFlights: {\_countOfFlights}" + "\r" + "\n" + $"\_countOfAirplanes: " +

$"{\_countOfAirplanes}" + "\r" + "\n" + $"\_countOfEmployees: {\_countOfEmployees}" +

"\r" + "\n" + $"\_countOfPassengers: {\_countOfPassengers}" + "\r" + "\n" +

$"\_revenue: {\_revenue}";

}

/// <summary>

/// Геттер названия аэропорта

/// </summary>

/// <returns> Название аэропорта </returns>

public string GetName()

{

return \_name;

}

/// <summary>

/// Геттер кода ИАТА аэропорта

/// </summary>

/// <returns> Код ИАТА </returns>

public string GetCodeIATA()

{

return \_codeIATA;

}

/// <summary>

/// Геттер количества рейсов

/// </summary>

/// <returns> Количество рейсов </returns>

public int GetCountOfFlights()

{

return \_countOfFlights;

}

/// <summary>

/// Геттер количества самолётов

/// </summary>

/// <returns> Количество самолётов </returns>

public int GetCountOfAirplanes()

{

return \_countOfAirplanes;

}

/// <summary>

/// Геттер количества сотрудников

/// </summary>

/// <returns> Количество сотрудников </returns>

public int GetCountOfEmployees()

{

return \_countOfEmployees;

}

/// <summary>

/// Геттер количества пассажиров в год в млн

/// </summary>

/// <returns> Количество пассажиров в год в млн </returns>

public float GetCountOfPassengers()

{

return \_countOfPassengers;

}

/// <summary>

/// Геттер дохода аэропорта за последний год в млрд.руб

/// </summary>

/// <returns> Доход за последний год в млрд.руб </returns>

public double GetRevenue()

{

return \_revenue;

}

/// <summary>

/// Геттер списка объектов

/// </summary>

/// <returns> Список объектов класса Airport

static public List<Airport> GetList()

{

return \_list;

}

/// <summary>

/// Геттер числа объектов

/// </summary>

/// <returns> Количество объектов класса Airport

static public int GetCount()

{

return \_count;

}

/// <summary>

/// Геттер ID объекта

/// </summary>

/// <returns> ID объектов класса Airport

public int GetID()

{

return \_id;

}

/// <summary>

/// Сеттер названия аэропорта

/// </summary>

/// <param name="s"> Название аэропорта </param>

public void SetName(string s)

{

\_name = s;

}

/// <summary>

/// Сеттер кода ИАТА аэропорта

/// </summary>

/// <param name="s"> Код ИАТА аэропорта </param>

public void SetCodeIATA(string s)

{

\_codeIATA = s;

}

/// <summary>

/// Сеттер количества рейсов

/// </summary>

/// <param name="x"> Количество рейсов </param>

public void SetCountOfFlights(int x)

{

\_countOfFlights = x;

}

/// <summary>

/// Сеттер количества самолётов

/// </summary>

/// <param name="x"> Количество самолётов </param>

public void SetCountOfAirplanes(int x)

{

\_countOfAirplanes = x;

}

/// <summary>

/// Сеттер количества сотрудников

/// </summary>

/// <param name="x"> Количество сотрудников </param>

public void SetCountOfEmployees(int x)

{

\_countOfEmployees = x;

}

/// <summary>

/// Сеттер количества пассажиров в год в млн

/// </summary>

/// <param name="x"> Количество пассажиров в год в млн </param>

public void SetCountOfPassengers(float x)

{

\_countOfPassengers = x;

}

/// <summary>

/// Сеттер дохода аэропорта за последний год в млрд.руб

/// </summary>

/// <param name="x"> Доход аэропорта за последний год в млрд.руб</param>

public void SetRevenue(double x)

{

\_revenue = x;

}

/// <summary>

/// Вывод названия аэропорта в консоль

/// </summary>

public void PrintName()

{

Console.WriteLine($"Название: {\_name}");

}

/// <summary>

/// Вывод кода ИАТА аэропорта в консоль

/// </summary>

public void PrintCodeIATA()

{

Console.WriteLine($"Код ИАТА: {\_codeIATA}");

}

/// <summary>

/// Вывод количества рейсов в консоль

/// </summary>

public void PrintCountOfFlights()

{

Console.WriteLine($"Количество рейсов: {\_countOfFlights}");

}

/// <summary>

/// Вывод количества самолётов в консоль

/// </summary>

public void PrintCountOfAirplanes()

{

Console.WriteLine($"Количество самолётов: {\_countOfAirplanes}");

}

/// <summary>

/// Вывод количества сотрудников в консоль

/// </summary>

public void PrintCountOfEmployees()

{

Console.WriteLine($"Количество сотрудников: {\_countOfEmployees}");

}

/// <summary>

/// Вывод количества пассажиров в год в млн в консоль

/// </summary>

public void PrintCountOfPassengers()

{

Console.WriteLine($"Количество пассажиров в год в млн: {\_countOfPassengers}");

}

/// <summary>

/// Вывод годового дохода аэропорта в млрд.руб в консоль

/// </summary>

public void PrintRevenue()

{

Console.WriteLine($"Годовой доход в млрд.руб: {\_revenue}");

}

/// <summary>

/// Переводит количество самолётов в шестнадцатиричную систему счисления

/// </summary>

/// <returns> Количество самолётов в шестнадцатиричной системе счисления </returns>

public string CountOfAirplanesHex()

{

return Convert.ToString(\_countOfAirplanes, 16);

}

public void List\_push()

{

\_list.Add(this);

}

}

}

**Form1.cs**

using System;

using System.Reflection;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Lab2

{

public partial class Airport1 : Form

{

public Airport1()

{

InitializeComponent();

}

[DllImport("user32.dll", CharSet = CharSet.Auto)]

public static extern IntPtr MessageBox(int hWnd, String text,

String caption, uint type);

private void button3\_Click(object sender, EventArgs e)

{

Close();

}

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

{

output.Clear();

}

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

{

foreach (Airport item in Airport.GetList())

{

string s = item.ToString();

output.Text += s + "\r" + "\n" + "\r" + "\n";

}

}

private void output\_class\_Click(object sender, EventArgs e)

{

Type type = typeof(Airport);

string s = "Поля:" + "\r" + "\n";

foreach (FieldInfo field in type.GetFields(BindingFlags.Instance | BindingFlags.NonPublic

| BindingFlags.Public | BindingFlags.Static))

{

string modificator = "";

if (field.IsPublic)

modificator += "public ";

else if (field.IsPrivate)

modificator += "private ";

if (field.IsStatic) modificator += "static ";

s += $"{modificator}{field.FieldType.Name} {field.Name}" + "\r" + "\n"; ;

}

s += "\r" + "\n" + "Методы:" + "\r" + "\n";

foreach (MethodInfo method in type.GetMethods())

{

string modificator = "";

if (method.IsStatic) modificator += "static ";

if (method.IsVirtual) modificator += "virtual ";

s += $"{modificator}{method.ReturnType.Name} {method.Name} ()" + "\r" + "\n";

}

output.Text += s;

}

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

{

foreach (Control item in create.Parent.Controls)

{

if (item.Text == "")

{

MessageBox(0, "Заполните все поля", "Найдено пустое поле", 0);

return;

}

}

try

{

string name1 = name.Text;

string code1 = code.Text;

int flights1 = Int32.Parse(flights.Text);

int planes1 = Int32.Parse(planes.Text);

int people1 = Int32.Parse(people.Text);

float passengers1 = float.Parse(passengers.Text);

double revenue1 = Double.Parse(revenue.Text);

new Airport(name1, code1, flights1, planes1, people1, passengers1, revenue1).List\_push();

obcount.Text = Airport.GetCount().ToString();

}

catch (FormatException ex)

{

InvalidConvertingException ex1 = new InvalidConvertingException("Невозможно преобразовать " +

"введенные данные к типу ", ex.TargetSite.Name);

MessageBox(0, ex1.Message + ex1.GetRtype(), "Неверное преобразование", 0);

return;

}

foreach (Control item in create.Parent.Controls)

{

if (item.GetType().Name == "TextBox")

{

item.Text = "";

}

}

}

private void Airport1\_Load(object sender, EventArgs e)

{

obcount.Text = Airport.GetCount().ToString();

}

private void print\_field\_Click(object sender, EventArgs e)

{

foreach (Control item in change.Parent.Controls)

{

if (item.Text == "")

{

MessageBox(0, "Заполните все поля", "Найдено пустое поле", 0);

return;

}

}

string id = id2.Text;

string field = field2.Text;

Type type = typeof(Airport);

try

{

if (Int32.Parse(id) >= Airport.GetCount())

{

MessageBox(0, $"Объект с ID {id} не найден", "Неверный ID объекта", 0);

return;

}

if (type.GetField(field, BindingFlags.Instance | BindingFlags.NonPublic) == null)

{

throw new UnidentifiedFieldException($"Поля {field} не существует в контескте класса Airport");

}

}

catch (FormatException ex)

{

InvalidConvertingException ex1 = new InvalidConvertingException($"Невозможно преобразовать " +

$"{id} к типу ", ex.TargetSite.Name);

MessageBox(0, ex1.Message + ex1.GetRtype(), "Неверное преобразование", 0);

return;

}

catch (UnidentifiedFieldException ex)

{

MessageBox(0, ex.Message, "Неизвестное поле класса", 0);

return;

}

string s = "";

foreach (Airport item in Airport.GetList())

{

if (item.GetID().ToString() == id)

{

var f = type.GetField(field, BindingFlags.Instance | BindingFlags.NonPublic);

var value = f?.GetValue(item);

s += $"{field}: {value}" + "\r" + "\n";

output.Text += s;

break;

}

}

foreach (Control item in print\_field.Parent.Controls)

{

if (item.GetType().Name == "TextBox")

{

item.Text = "";

}

}

}

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

{

foreach (Control item in change.Parent.Controls)

{

if (item.Text == "")

{

MessageBox(0, "Заполните все поля", "Найдено пустое поле", 0);

return;

}

}

string id = id1.Text;

string field = field1.Text;

string value = value1.Text;

Type type = typeof(Airport);

try

{

if (Int32.Parse(id) >= Airport.GetCount())

{

MessageBox(0, $"Объект с ID {id} не найден", "Неверный ID объекта", 0);

return;

}

if (type.GetField(field, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Static) == null)

{

throw new UnidentifiedFieldException($"Поля {field} не существует в контескте класса Airport");

}

}

catch (FormatException ex)

{

InvalidConvertingException ex1 = new InvalidConvertingException($"Невозможно преобразовать " +

$"{id} к типу ", ex.TargetSite.Name);

MessageBox(0, ex1.Message + ex1.GetRtype(), "Неверное преобразование", 0);

return;

}

catch (UnidentifiedFieldException ex)

{

MessageBox(0, ex.Message, "Неизвестное поле класса", 0);

return;

}

if (field == "\_id")

{

MessageBox(0, "Невозможно изменить значение поля ID", "Изменение невозможно", 0);

return;

}

if (type.GetField(field, BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Static).IsStatic)

{

MessageBox(0, "Невозможно изменить значение статического поля класса", "Изменение невозможно", 0);

return;

}

foreach (Airport item in Airport.GetList())

{

try

{

if (item.GetID().ToString() == id)

{

var f = type.GetField(field, BindingFlags.Instance | BindingFlags.NonPublic);

string n = f.FieldType.Name;

if (n == "Int32")

{

f?.SetValue(item, Int32.Parse(value));

}

else if (n == "String")

{

f?.SetValue(item, value);

}

else if (n == "Single")

{

f?.SetValue(item, float.Parse(value));

}

else if (n == "Double")

{

f?.SetValue(item, Double.Parse(value));

}

break;

}

}

catch (FormatException ex)

{

InvalidConvertingException ex1 = new InvalidConvertingException($"Невозможно преобразовать " +

$"{value} к типу ", ex.TargetSite.Name);

MessageBox(0, ex1.Message + ex1.GetRtype(), "Неверное преобразование", 0);

return;

}

}

foreach (Control item in change.Parent.Controls)

{

if (item.GetType().Name == "TextBox")

{

item.Text = "";

}

}

}

}

}

**InvalidConveringException.cs**

using System;

namespace Lab2

{

public class InvalidConvertingException : FormatException

{

private string \_rtype;

public InvalidConvertingException(string m, string type) : base(m)

{

switch (type)

{

case "StringToNumber":

\_rtype = "int";

break;

case "ParseSingle":

\_rtype = "float";

break;

case "ParseDouble":

\_rtype = "double";

break;

}

}

public string GetRtype() { return \_rtype; }

}

}

**UnidentifiedFieldException.cs**

using System;

namespace Lab2

{

public class UnidentifiedFieldException : Exception

{

public UnidentifiedFieldException(string message) : base(message) { }

}

}

# Пояснительный текст к программе

Класс InvalidConvertingException наследуется от класса FormatException. Исключение данного класса вызывается, когда введенную пользователем строку невозможно преобразовать к одному из численных типов данных.

UnidentifiedFieldException - пользовательский тип исключения, которое вызывается, когда пользователь вводит несуществующее поле класса Airport.

**Результаты работы программы**

Результаты работы программы показаны на рисунке 1.

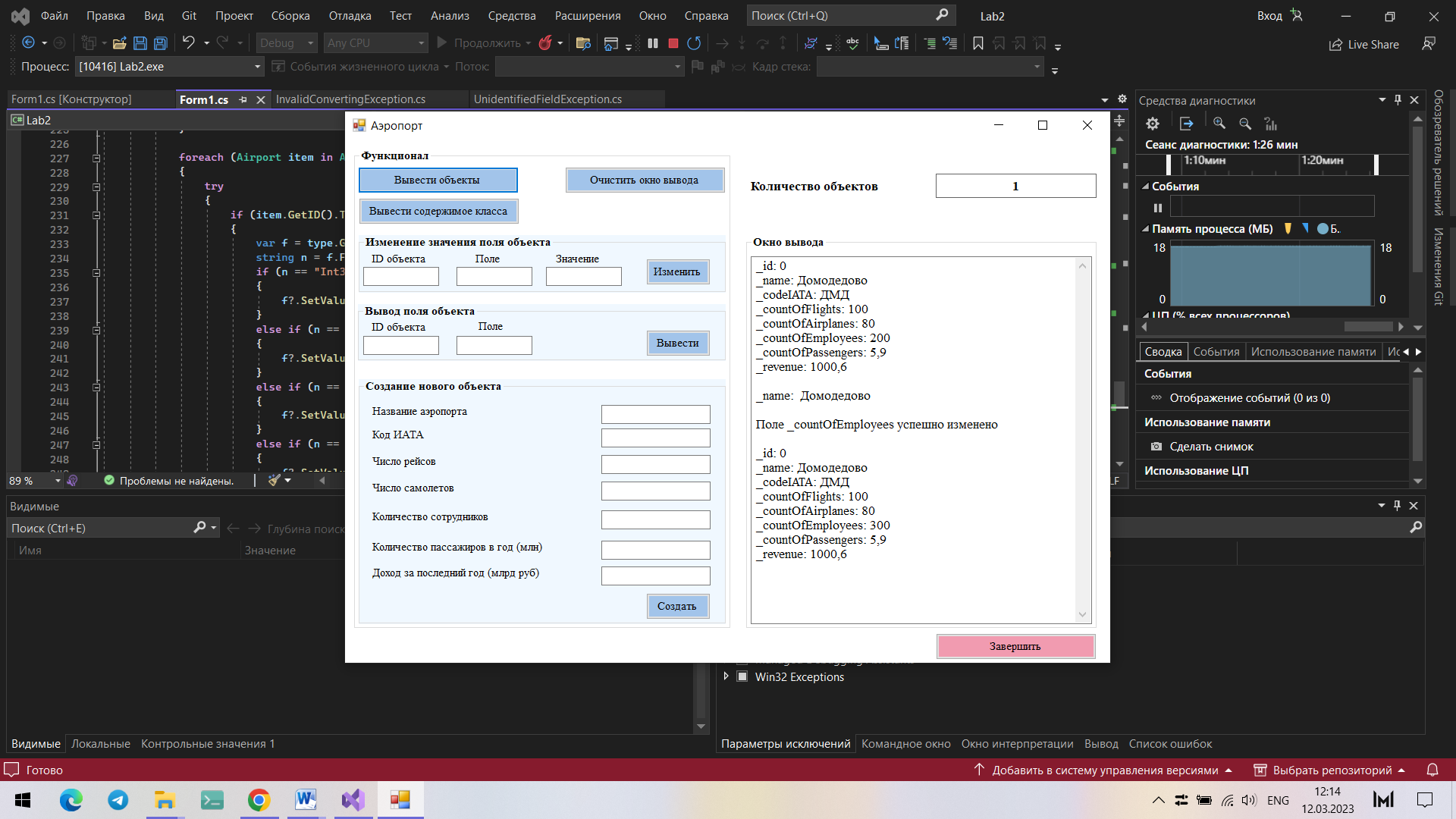


Рисунок 1 — Результаты работы программы

# Выводы

В ходе выполнения лабораторной работы был реализован пользовательский класс исключений, а также переопределен класс исключений FormatExceotion, Также в класс Airport были добавлены статические поля: счетчик объектов и список для хранения объектов.