**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №3

«Трек курса «Функциональное программирование»»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-34 |  | преподаватель каф. ИУ5 |
| Коротков Н. К. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

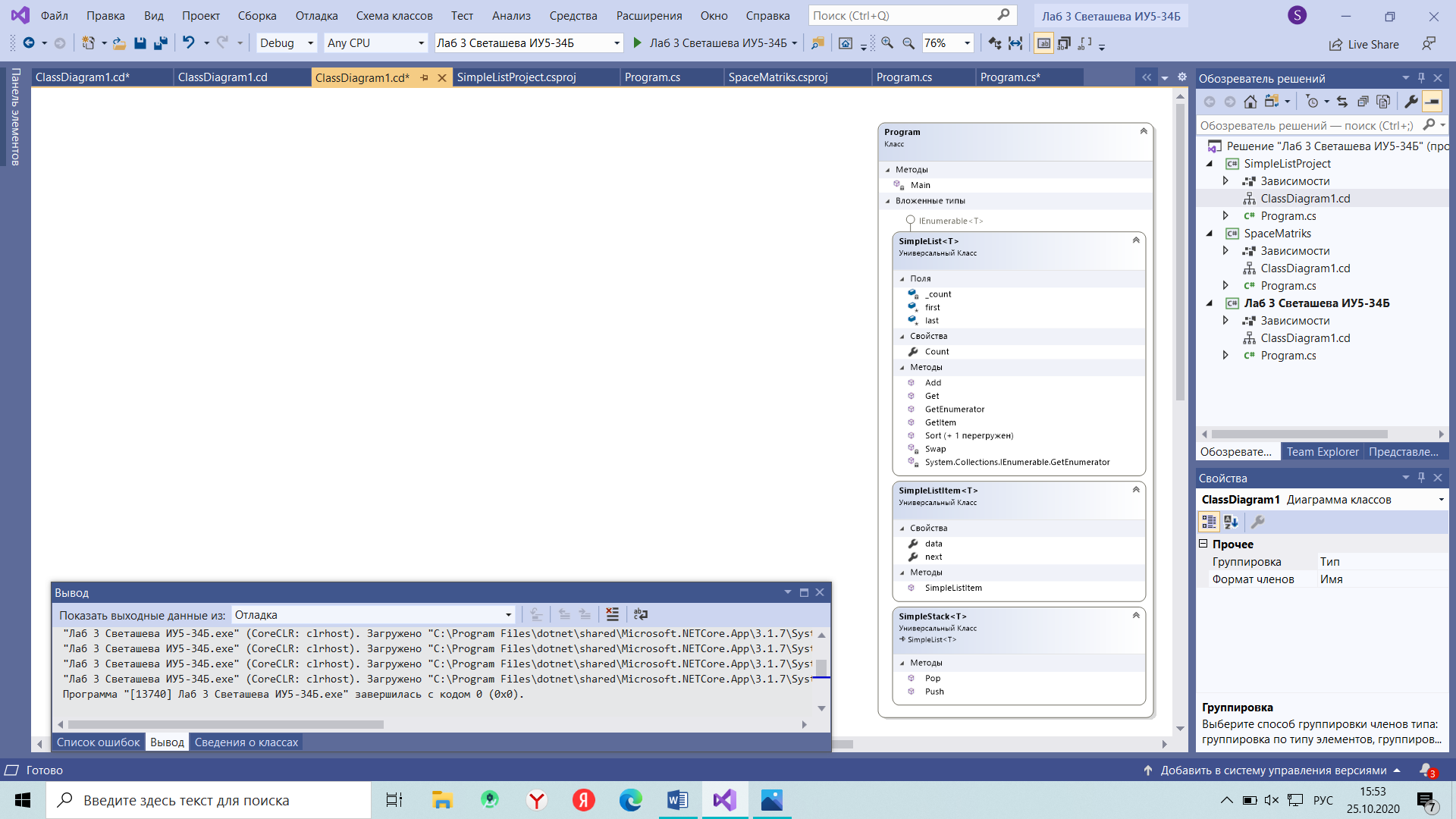
**Описание задания**

Разработать программу, реализующую работу с коллекциями.

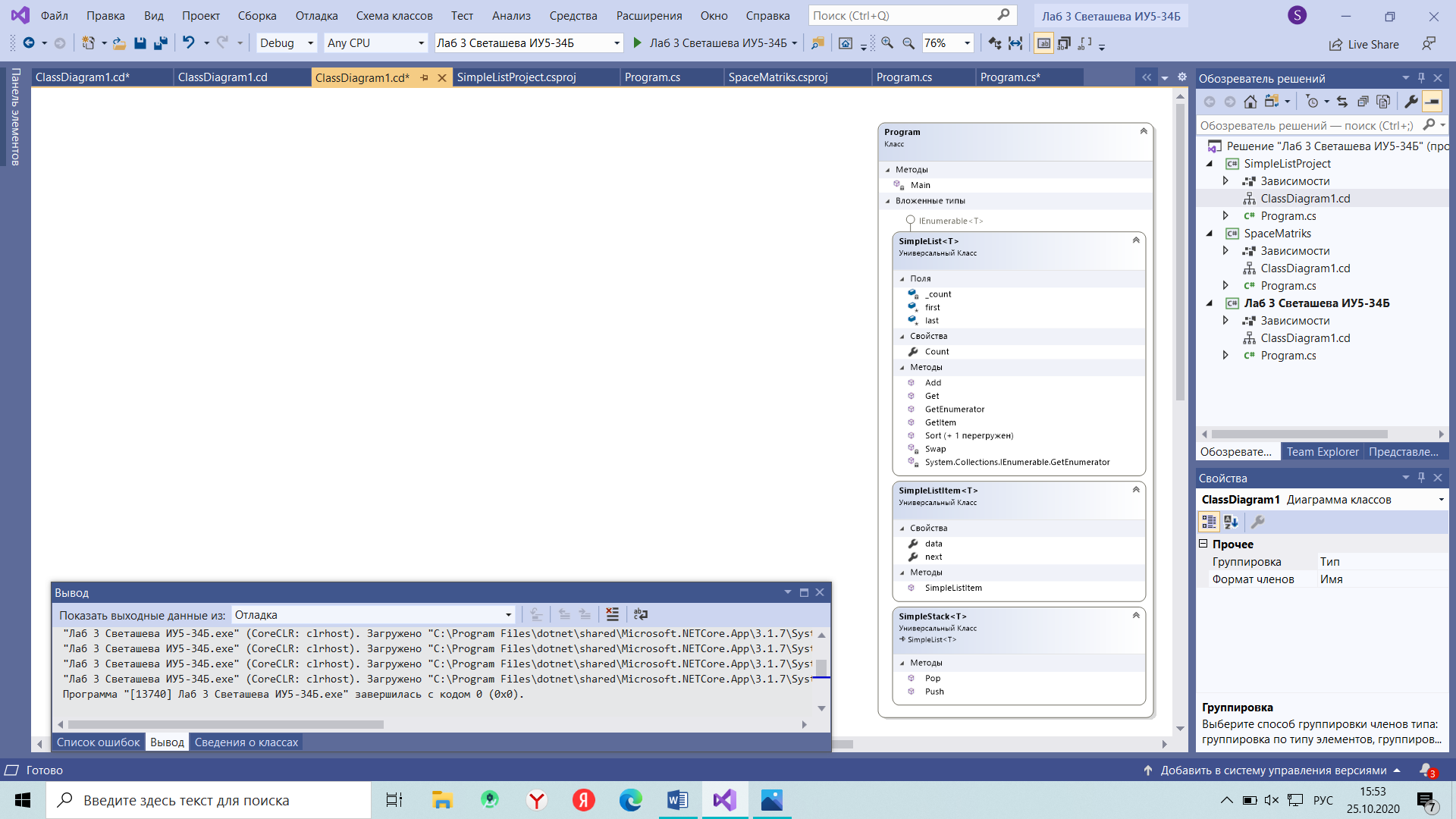
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.
4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
5. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
6. Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.
7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:
   * public void Push(T element) – добавление в стек;
   * public T Pop() – чтение с удалением из стека.
8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Диаграмма классов**

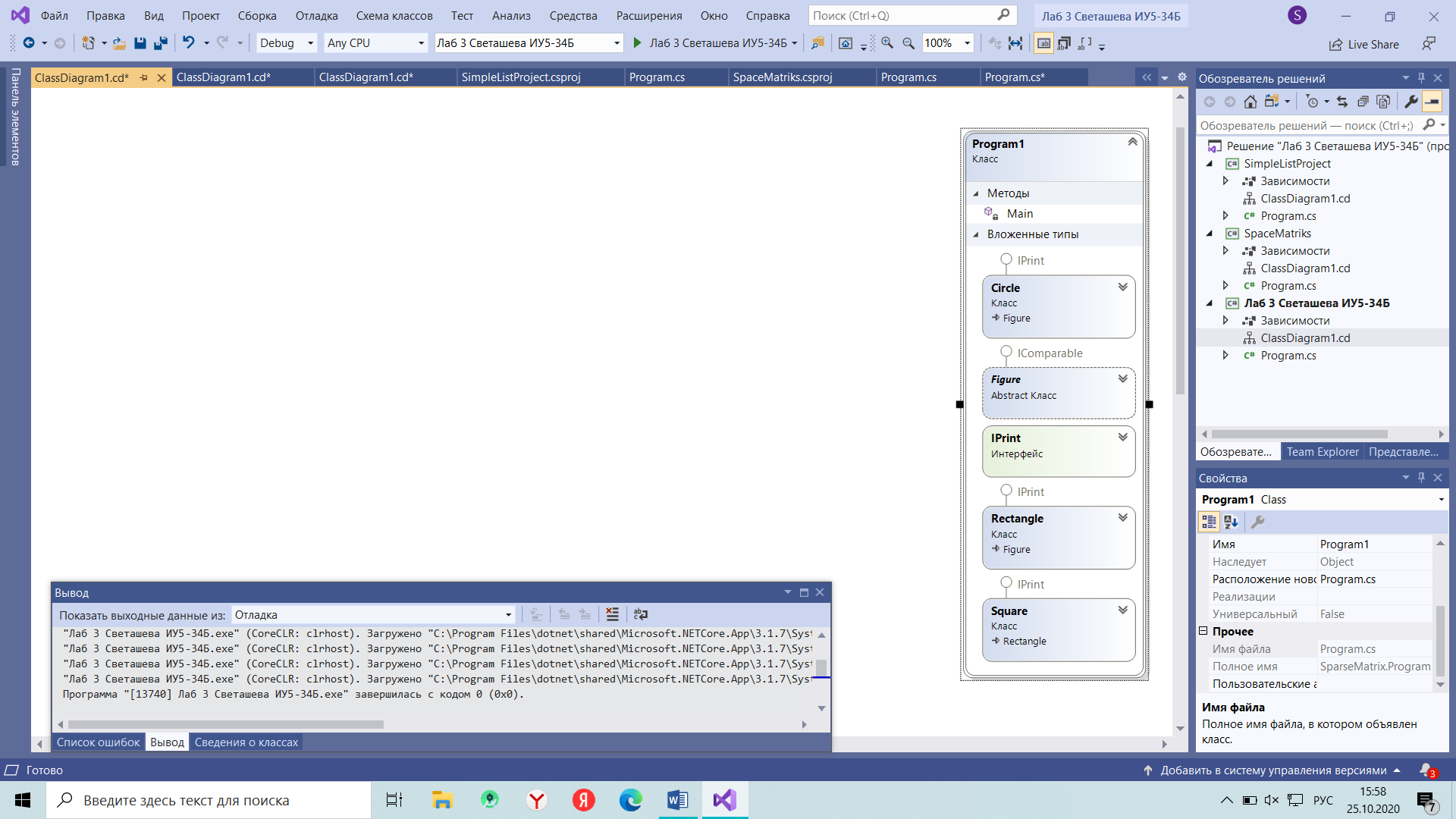
1. Для проекта SimpleListProject:

****

1. Для проекта SpaceMatriks:



1. Для проекта 3



**Текст программы**

1. Для проекта SimpleListProject:

using System;

using System.Collections.Generic;

using static SparseMatrix.Program1;

namespace SparseMatrix

{

class Program

{

static void Main(string[] args)

{

SimpleList<Figure> list = new SimpleList<Figure>();

Program1.Figure rect = new Program1.Rectangle(2, 8);

Program1.Figure square = new Program1.Square(5);

Program1.Figure circle = new Program1.Circle(7);

list.Add(circle);

list.Add(rect);

list.Add(square);

Console.WriteLine("\nПеред сортировкой:");

foreach (var x in list) Console.WriteLine(x);

//сортировка

list.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in list) Console.WriteLine(x);

SimpleStack<Figure> stack = new SimpleStack<Figure>();

//добавление данных в стек

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

//чтение данных из стека

Console.WriteLine(" ");

Console.WriteLine("Чтение днных из стека после добавоения новых элементов");

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

}

public class SimpleListItem<T>

{

public T data { get; set; }

public SimpleListItem<T> next { get; set; }

public SimpleListItem(T param)

{

this.data = param;

}

}

public class SimpleList<T> : IEnumerable<T>//верширшина стека-конец списка

where T : IComparable

{

/// <summary>

/// Первый элемент списка

/// </summary>

protected SimpleListItem<T> first = null;

/// <summary>

/// Последний элемент списка

/// </summary>

protected SimpleListItem<T> last = null;

/// <summary>

/// Количество элементов

/// </summary>

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

/// <summary>

/// Добавление элемента

/// </summary>

public void Add(T element)

{

SimpleListItem<T> newItem = new SimpleListItem<T>(element);

this.Count++;

//Добавление первого элемента

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

//Добавление следующих элементов

else

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Присоединенный элемент считается последним

this.last = newItem;

}

}

/// <summary>

/// Чтение контейнера с заданным номером

/// </summary>

public SimpleListItem<T> GetItem(int number)

{

if ((number < 0) || (number >= this.Count))

{

//Можно создать собственный класс исключения

throw new Exception("Выход за границу индекса");

}

SimpleListItem<T> current = this.first;

int i = 0;

//Пропускаем нужное количество элементов

while (i < number)

{

//Переход к следующему элементу

current = current.next;

//Увеличение счетчика

i++;

}

return current;

}

/// <summary>

/// Чтение элемента с заданным номером

/// </summary>

public T Get(int number)

{

return GetItem(number).data;

}

/// <summary>

/// Для перебора коллекции

/// </summary>

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

//Перебор элементов

while (current != null)

{

//Возврат текущего значения

yield return current.data;

//Переход к следующему элементу

current = current.next;

}

}

//Реализация обобщенного IEnumerator<T> требует реализации

//необобщенного интерфейса

//Данный метод добавляется автоматически при реализации интерфейса

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// Cортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary>

/// Алгоритм быстрой сортировки

/// </summary>

private void Sort(int low, int high)

{

int i = low;

int j = high;

T x = Get((low + high) / 2);

do

{

while (Get(i).CompareTo(x) < 0) ++i;

while (Get(j).CompareTo(x) > 0) --j;

if (i <= j)

{

Swap(i, j);

i++; j--;

}

} while (i <= j);

if (low < j) Sort(low, j);

if (i < high) Sort(i, high);

}

/// <summary>

/// Вспомогательный метод для обмена элементов при

//сортировке

/// </summary>

private void Swap(int i, int j)

{

SimpleListItem<T> ci = GetItem(i);

SimpleListItem<T> cj = GetItem(j);

T temp = ci.data;

ci.data = cj.data;

cj.data = temp;

}

}

class SimpleStack<T> : SimpleList<T> where T : IComparable

{

/// <summary>

/// Добавление в стек

/// </summary>

public void Push(T element)

{

//Добавление в конец списка уже реализовано

Add(element);

}

/// <summary>

/// Удаление и чтение из стека

/// </summary>

public T Pop()

{

//default(T) - значение для типа T по умолчанию

T Result = default(T);

//Если стек пуст, возвращается значение по умолчанию для типа

if (this.Count == 0) return Result;

//Если элемент единственный

if (this.Count == 1)

{

//то из него читаются данные

Result = this.first.data;

//обнуляются указатели начала и конца списка

this.first = null;

this.last = null;

}

//В списке более одного элемента

else

{

//Поиск предпоследнего элемента

SimpleListItem<T> newLast = this.GetItem(this.Count - 2);

//Чтение значения из последнего элемента

Result = newLast.next.data;

//предпоследний элемент считается последним

this.last = newLast;

//последний элемент удаляется из списка

newLast.next = null;

}

//Уменьшение количества элементов в списке

this.Count--;

//Возврат результата

return Result;

}

}

}

}

1. Для проекта SpaceMatriks:

using System;

using System.Collections.Generic;

using System.Text;

using static SparseMatrix.Program1;

namespace SparseMatrix

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("\nМатрица");

Matrix<Program1.Figure> matrix = new Matrix<Program1.Figure>(3, 3,3, new FigureMatrixCheckEmpty());

Program1.Figure rect = new Program1.Rectangle(3, 9);

Program1.Figure square = new Program1.Square(7);

Program1.Figure circle = new Program1.Circle(6);

matrix[0, 0, 0] = rect;

matrix[1, 1, 1] = square;

matrix[2, 2, 2] = circle;

Console.WriteLine(matrix.ToString());

}

public class Matrix<T>

{

Dictionary<string, T> \_matrix = new Dictionary<string, T>();

int maxX;

int maxY;

int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> сheckEmptyParam)//внедрение зависимостей через контруктор

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y, int z]

{

set

{

CheckBounds(x, y, z);

string key = DictKey(x, y, z);

this.\_matrix.Add(key, value);

}

get

{

CheckBounds(x, y, z);

string key = DictKey(x, y, z);

if (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.сheckEmpty.getEmptyElement();

}

}

}

/// <summary>

/// Проверка границ

/// </summary>

void CheckBounds(int x, int y, int z)

{

if (x < 0 || x >= this.maxX)

{

throw new ArgumentOutOfRangeException("x", "x=" + x + " выходит за границы");

}

if (y < 0 || y >= this.maxY)

{

throw new ArgumentOutOfRangeException("y", "y=" + y + " выходит за границы");

}

if (z < 0 || z >= this.maxZ)

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " выходит за границы");

}

}

/// <summary>

/// Формирование ключа

/// </summary>

string DictKey(int x, int y, int z)

{

return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString();

}

/// <summary>

/// Приведение к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

StringBuilder b = new StringBuilder();

for ( int z = 0; z < this.maxZ; z++)

{

for (int j = 0; j < this.maxY; j++)

{

b.Append("[");

for (int i = 0; i < this.maxX; i++)

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

if (!this.сheckEmpty.checkEmptyElement(this[i, j, z]))

{

//Добавить приведенный к строке текущий элемент

b.Append(this[i, j, z].ToString());

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}// i

b.Append("]\n");

}//j

}//z

return b.ToString();

}

}

/// <summary>

/// Проверка пустого элемента матрицы

/// </summary>

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T getEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty <Figure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public Figure getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(Figure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}//тут\*/

}

}

1. Для проекта Лаб 3 Светашева ИУ5-34Б:

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Xml.Serialization;

namespace SparseMatrix

{

public class Program1

{

static void Main(string[] args)

{

Figure rect = new Rectangle(3, 9);

Figure square = new Square(7);

Figure circle = new Circle(6);

List<Figure> figa = new List<Figure> {

circle, rect, square //Добавили наши объекты в коллекцию

};

Console.WriteLine("List <Figure> До сортировки:");

foreach (var f in figa)

{//выводим отортированный список

Console.WriteLine(f);

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*");

//NameComparer nc= new NameComparer();

figa.Sort();

Console.WriteLine("List <Figure> После сортировки:");

foreach (var f in figa)

{//выводим отортированный список

Console.WriteLine(f);

}

ArrayList figaCollection = new ArrayList(3) { circle, rect, square };

Console.WriteLine("");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("ArrayList До сортировки:");

foreach (var f in figaCollection)

{//выводим отортированный список

Console.WriteLine(f);

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*");

//NameComparer nc1 = new NameComparer();

figaCollection.Sort();

Console.WriteLine("ArrayList После сортировки:");

foreach (var f in figaCollection)

{//выводим отортированный список

Console.WriteLine(f);

}

}

public abstract class Figure : IComparable

{

public abstract double Area();

public string Type { get; set; }

public override string ToString()

{

return this.Type + "площадью" + this.Area().ToString();

}

//Реализация IComparable

public int CompareTo(object other)

{

Figure p = (Figure)other;

if (this.Area() > p.Area())

return 1;

if (this.Area() < p.Area())

return -1;

else

return 0;

}

}

public class Rectangle : Figure, IPrint

{

private double \_heigth;

public double heigth

{

set

{

if (value < 0)

Console.WriteLine("Введено некоректное значение высоты");

else \_heigth = value;

}

get { return \_heigth; }

}

private double \_width;

public double width

{

set

{

if (value < 0)

Console.WriteLine("Введено некоректное значение ширины");

else

\_width = value;

}

get { return \_width; }

}

public Rectangle(double width, double heigth)

{

this.width = width;

this.heigth = heigth;

}

public override double Area()

{

return this.heigth \* this.width;

}

public override string ToString()

{

return "Прямоугольник высотой " + this.heigth.ToString() + ", шириной " + this.width.ToString() + " и площадью " + this.Area().ToString();

}

void IPrint.Print()

{

Console.WriteLine(this.ToString());

}

}

public class Square : Rectangle, IPrint

{

public double length;

public Square(double length)

: base(length, length)

{

this.length = length;

}

public override double Area()

{

return Math.Pow(this.length, 2);

}

public override string ToString()

{

return "Квадрат со стороной " + this.length.ToString() + " и площадью " + this.Area().ToString();

}

void IPrint.Print()

{

Console.WriteLine(this.ToString());

}

}

public class Circle : Figure, IPrint

{

private double \_radius;

public double radius

{

set

{

if (value < 0)

Console.WriteLine("Введено некоректное значение радиуса");

else

\_radius = value;

}

get { return \_radius; }

}

public Circle(double radius)

{

this.radius = radius;

}

public override double Area()

{

return Math.Pow(this.radius, 2) \* 3.14;

}

public override string ToString()

{

return "Круг радиусом " + this.radius.ToString() + " и площадью " + this.Area().ToString();

}

void IPrint.Print()

{

Console.WriteLine(this.ToString());

}

}

interface IPrint

{

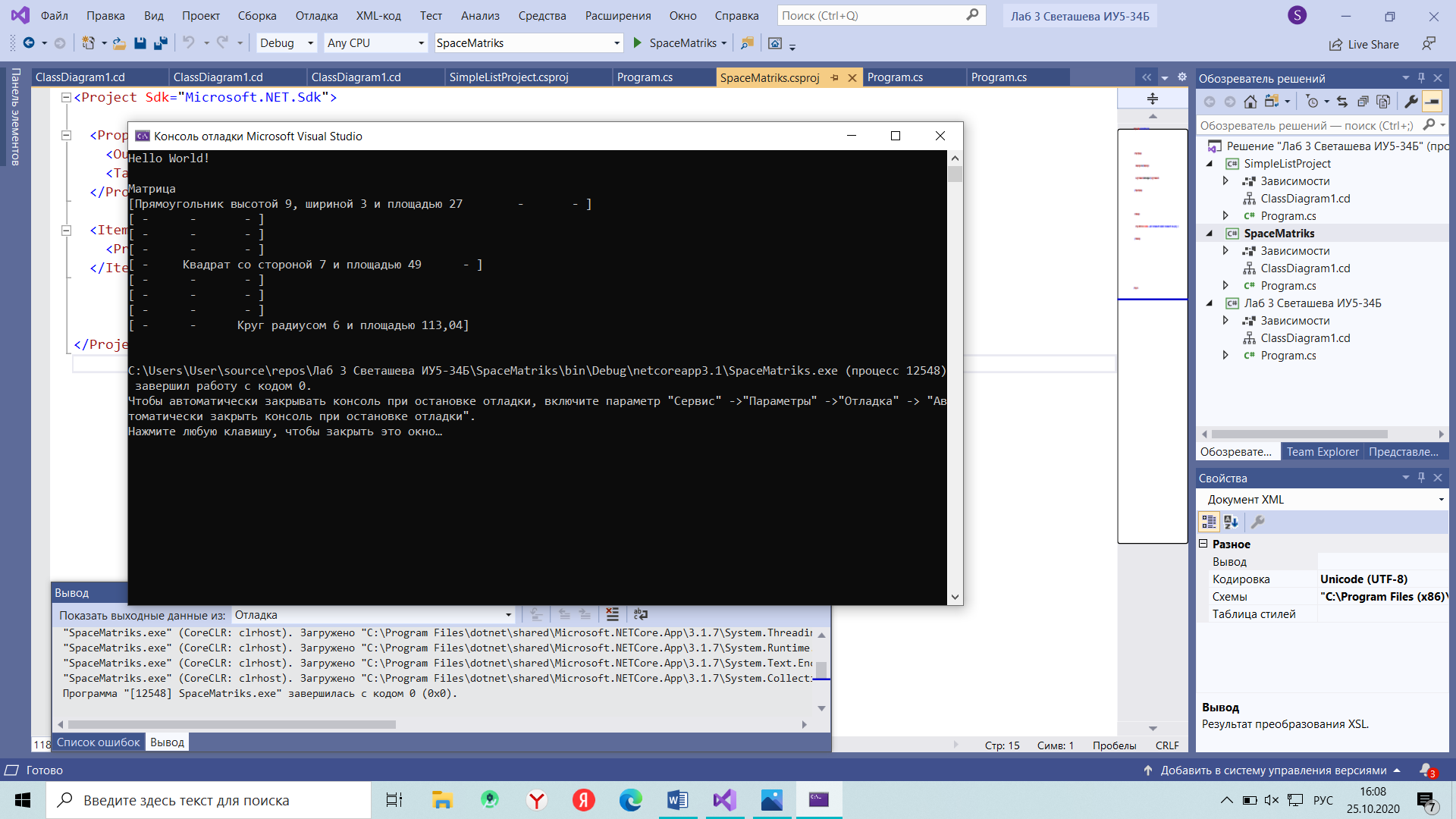
void Print();

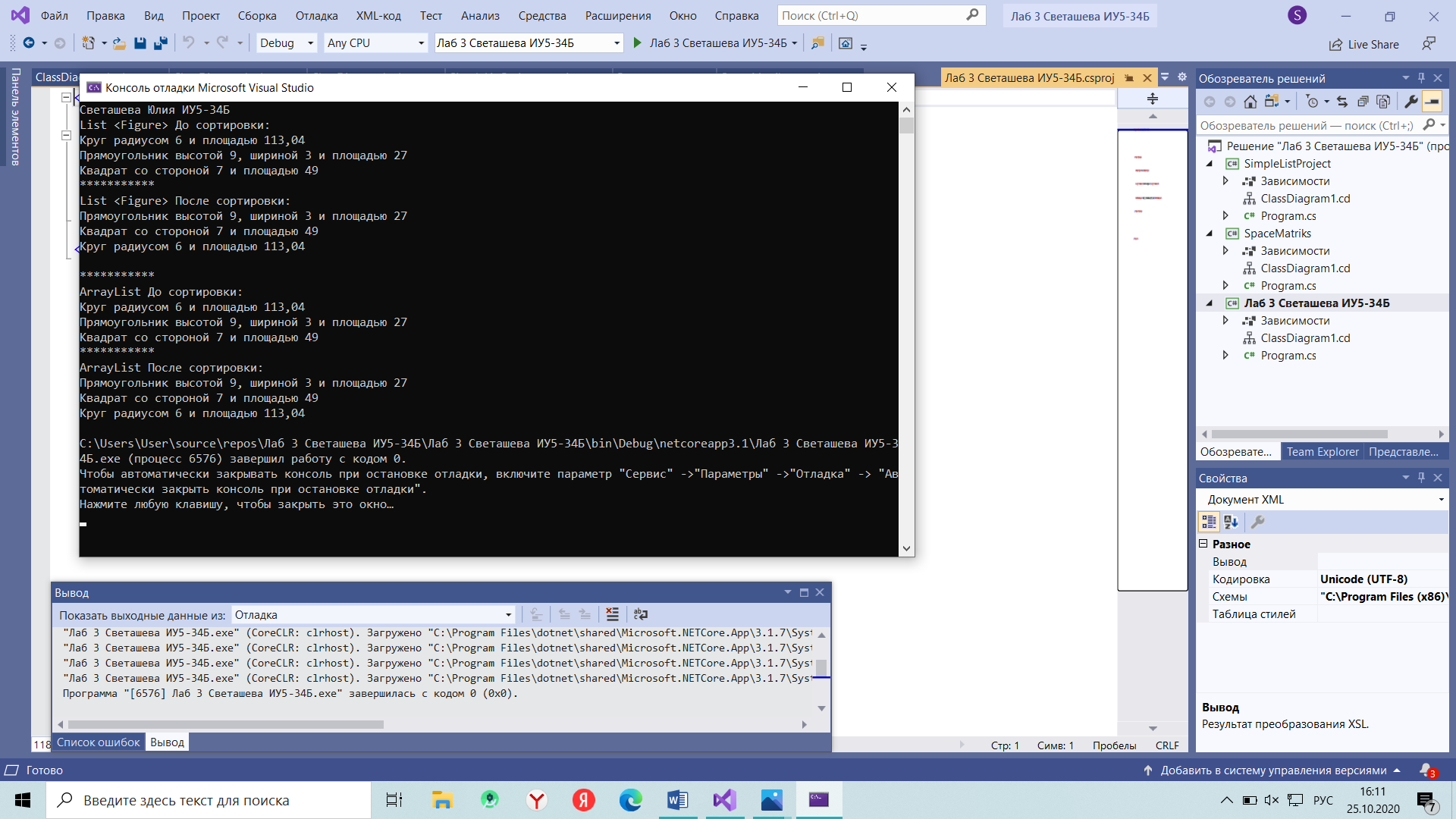
}

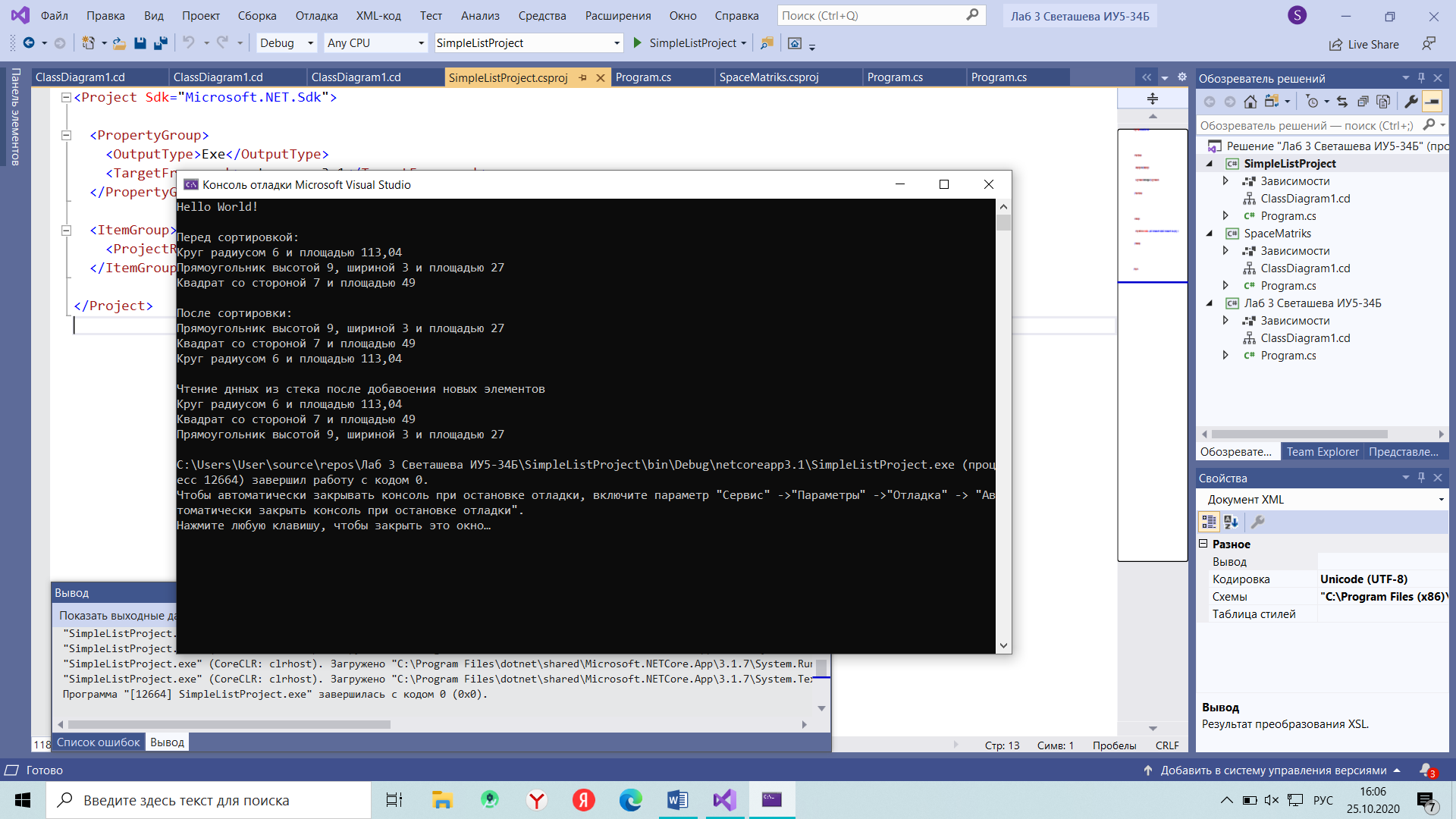
}

}

**Экранные формы с примерами выполнения программы**





****

