**Отчет по лабораторной работе №2 и №3 по курсу С#**

**12**

**(количество листов)**

Студентка группы ИУ5-32

Губайдуллина Карина

Дата: 21.11.2017

Руководитель:

Гапанюк Ю.Е.

Подпись:

Дата:

**Задание:**

**Лабораторная работа №2**

Разработать программу, реализующую работу с классами.

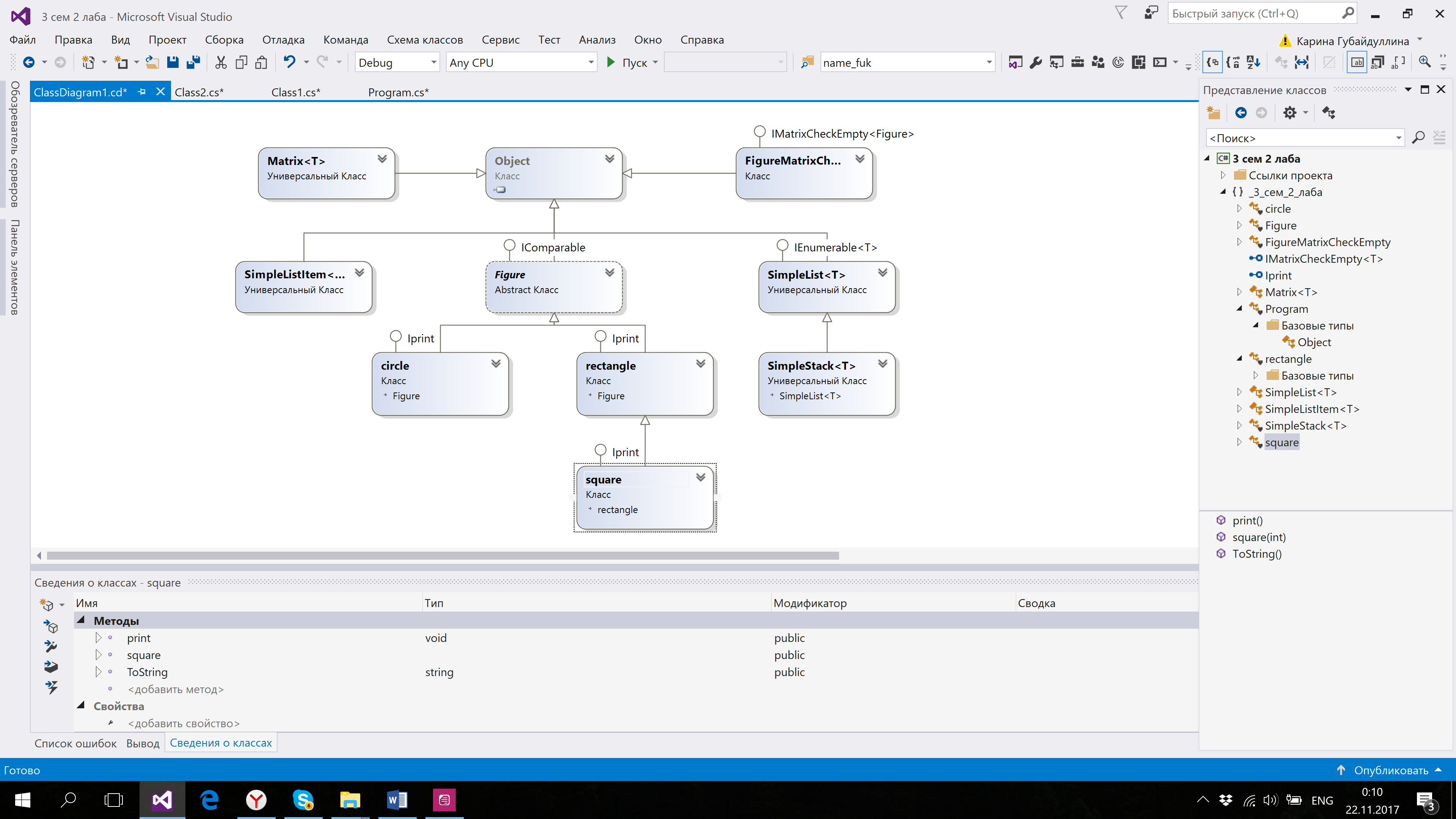
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Абстрактный класс «Геометрическая фигура» содержит виртуальный метод для вычисления площади фигуры.
3. Класс «Прямоугольник» наследуется от «Геометрическая фигура». Ширина и высота объявляются как свойства (property). Класс должен содержать конструктор по параметрам «ширина» и «высота».
4. Класс «Квадрат» наследуется от «Прямоугольник». Класс должен содержать конструктор по длине стороны.
5. Класс «Круг» наследуется от «Геометрическая фигура». Радиус объявляется как свойство (property). Класс должен содержать конструктор по параметру «радиус».
6. Для классов «Прямоугольник», «Квадрат», «Круг» переопределить виртуальный метод Object.ToString(), который возвращает в виде строки основные параметры фигуры и ее площадь.
7. Разработать интерфейс IPrint. Интерфейс содержит метод Print(), который не принимает параметров и возвращает void. Для классов «Прямоугольник», «Квадрат», «Круг» реализовать наследование от интерфейса IPrint. Переопределяемый метод Print() выводит на консоль информацию, возвращаемую переопределенным методом ToString().

**Лабораторная работа №3**

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 (разобранного в пособии). Необходимо добавить в класс методы:
   1. public void Push(T element) – добавление в стек;
   2. public T Pop() – чтение с удалением из стека.
8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Диаграмма классов:**



**Текст программы:**

***Program.cs***

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

namespace \_3\_сем\_2\_лаба

{

class Program

{

static void Main(string[] args)

{

circle first;

first = new circle(4);

rectangle second = new rectangle(2, 4);

square third = new square(6);

Console.WriteLine("Вывод значений через Iprint:");

first.print();

second.print();

third.print();

ArrayList figure\_list = new ArrayList() { first, second, third };

List<Figure> figure\_list2 = new List<Figure>() { first, second, third };

//Console.WriteLine(figure\_list[0]);

figure\_list.Sort();

Console.WriteLine("\nВывод отсортированных значений в ArrayList:");

foreach (Figure f in figure\_list)

{

Console.WriteLine(f);

}

figure\_list2.Sort();

Console.WriteLine("\nВывод отсортированных значений в List:");

foreach (Figure f in figure\_list2)

{

Console.WriteLine(f);

}

Console.WriteLine("\nМатрица[3,3,2]");

Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 2, new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = first;

matrix[1, 1, 0] = second;

matrix[2, 2, 1] = third;

Console.WriteLine(matrix.ToString());

//Выход за границы индекса и обработка исключения

try

{

Figure temp = matrix[12, 12, 12];

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

Console.WriteLine("\nСписок");

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

list.Add(first);

list.Add(second);

list.Add(third);

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);

Console.WriteLine("\nСтек");

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

//добавление данных в стек

stack.Push(first);

stack.Push(second);

stack.Push(third);

//чтение данных из стека

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

public interface Iprint

{

void print();

}

abstract class Figure : IComparable

{

public double squr;

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.squr < p.squr) return -1;

else if (this.squr == p.squr) return 0;

else return 1;

}

}

class circle : Figure, Iprint

{

int r;

public circle(int x)

{

r = x;

squr = 3.14 \* r \* r;

}

public override string ToString()

{

string result = "radius : " + r.ToString();

result = result + "; ploshad : ";

result = result + squr.ToString();

return result;

}

public void print()

{

Console.WriteLine(this.ToString());

}

}

class rectangle : Figure, Iprint

{

int w;

protected int l;

public rectangle(int x, int y)

{

w = x;

l = y;

squr = w \* l;

}

public rectangle(int x)

{

l = x;

squr = l \* l;

}

public override string ToString()

{

string result = w.ToString() + " : weidth ; ";

result = result + l.ToString() + " : long; " + squr.ToString() + " : ploshad";

return result;

}

public void print()

{

Console.WriteLine(this.ToString());

}

}

class square : rectangle, Iprint

{

public square(int x) : base(x)

{

l = x;

}

public override string ToString()

{

string result = l.ToString() + " : long; " + squr.ToString() + " : ploshad";

return result;

}

public void print()

{

Console.WriteLine(this.ToString());

}

}

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

IMatrixCheckEmpty<T> сheckEmpty;

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

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();

}

}

}

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");

}

}

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

{

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

}

public override string ToString()

{

StringBuilder b = new StringBuilder();

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

{

string temp = "z = " + k.ToString() + '\n';

b.Append(temp);

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, k]))

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

}

return b.ToString();

}

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Figure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

}

***Class1.cs***

sing System;

namespace \_3\_сем\_2\_лаба

{

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

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

T Result = default(T);

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

if (this.Count == 1)

{

Result = this.first.data;

this.first = null;

this.last = null;

}

else

{

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

Result = newLast.next.data;

this.last = newLast;

newLast.next = null;

}

this.Count--;

return Result;

}

}

}

***Class2.cs***

using System;

using System.Collections.Generic;

namespace \_3\_сем\_2\_лаба

{

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>

where T : IComparable

{

protected SimpleListItem<T> first = null;

protected SimpleListItem<T> last = null;

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

public void Add(T element)

{

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

this.Count++;

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

else

{

this.last.next = newItem;

this.last = newItem;

}

}

public SimpleListItem<T> GetItem(int number)

{

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

{

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

}

SimpleListItem<T> current = this.first;

int i = 0;

while (i < number)

{

current = current.next;

i++;

}

return current;

}

public T Get(int number)

{

return GetItem(number).data;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, this.Count - 1);

}

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);

}

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;

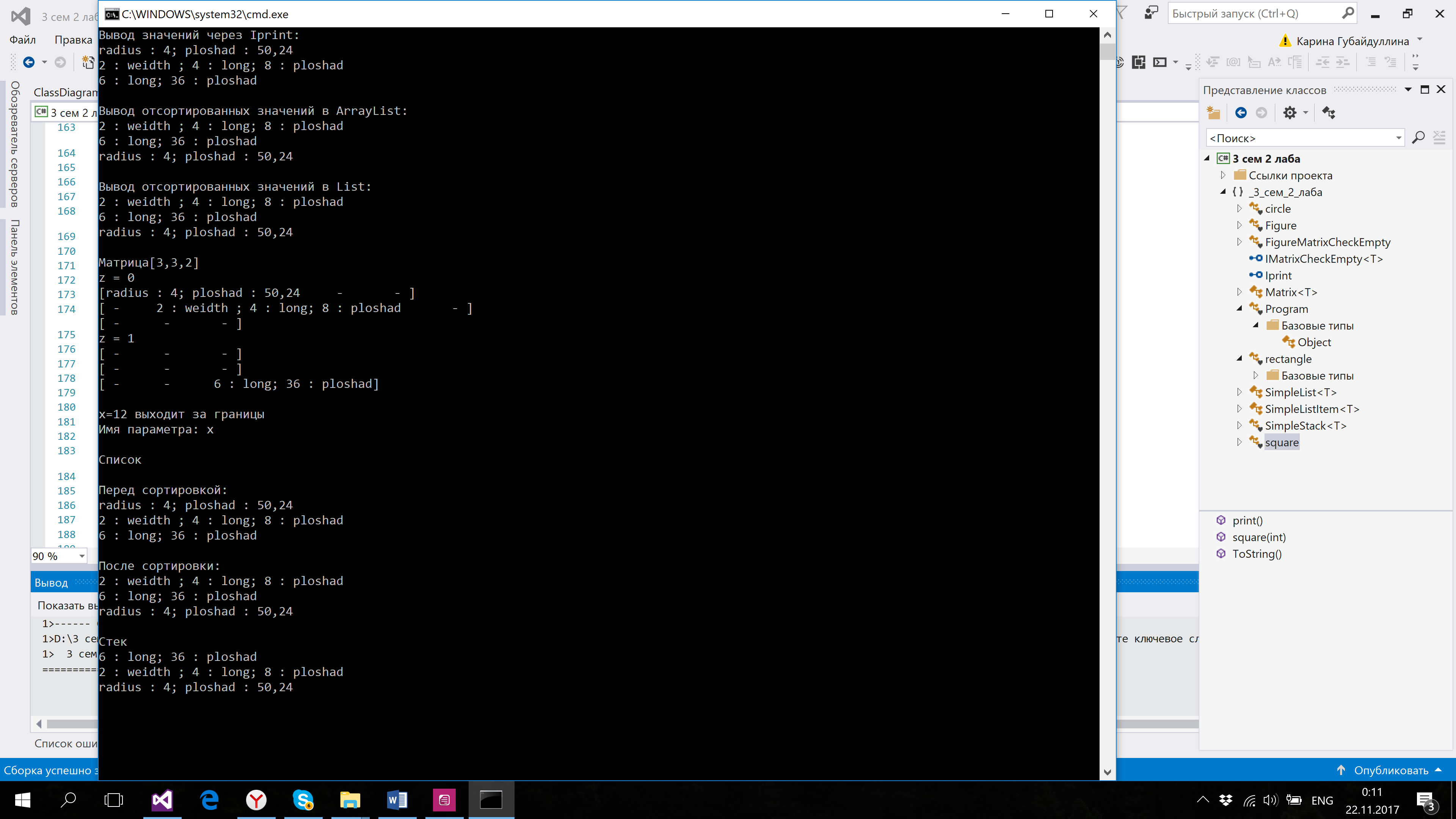
cj.data = temp;

}

}

}

Результаты:



|  |
| --- |
|  |