

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)



Факультет _____ Информатика и системы управления _____

Кафедра _____ Системы обработки информации и управления (ИУ-5) _____

Отчет по лабораторной работе № 3

Студент _____ Яковицкий Станислав Владиславович _____
(фамилия, имя, отчество)

Группа _____ ИУ5-35 _____

Название предмета _____ Базовые компоненты интернет-технологий _____

Руководитель

Гапанюк Ю.Е.
ФИО

дата

подпись

Студент

Яковицкий С.В.
ФИО

дата

подпись

Условие задачи

Разработать программу (отдельный проект в рамках решения включающего проекты ЛР1 и ЛР2), реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Объекты классов «Прямоугольник», «Квадрат», «Круг» использовать из проекта лабораторной работы №2.
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса `Comparable`. Сортировка производится по площади фигуры.
4. Создать коллекцию класса `ArrayList`. Сохранить объекты (типы) Прямоугольник, Квадрат, Круг, в коллекцию. Вывести в цикле содержимое площади элементов 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.

Код

Файл: Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace Lab3
{
    class Program
    {
        static void Main_menu()
        {
            Console.WriteLine();
            Console.WriteLine("MENU");
            Console.WriteLine();
            Console.WriteLine("1.Work with ArrayList");
            Console.WriteLine("2.Work with List");
            Console.WriteLine("3.Work with Sparse Matrix");
            Console.WriteLine("4.Work with Simple Stack");
            Console.WriteLine("5.Exit");
            Console.WriteLine();
        }
    }
}
```

```

}
static int Main(string[] args)
{
#region
int n = 0;
ArrayList arr_list = new ArrayList();
List<Geometric_figures> li = new List<Geometric_figures>();
double len;
Rectangle rect = new Rectangle(0, 0);
Console.WriteLine("Creating rectangle");
Console.WriteLine("Please put in your value");
Console.Write("Length 1 ");
len = Double.Parse(Console.ReadLine());
rect.length1 = len;
Console.Write("Length 2 ");
len = Double.Parse(Console.ReadLine());
rect.length2 = len;
Square sqr = new Square(0);
Console.WriteLine("Please put in your value");
Console.Write("Length ");
len = Double.Parse(Console.ReadLine());
sqr.length1 = len;
sqr.length2 = len;
Circle cir = new Circle(0);
Console.WriteLine("Please put in your value");
Console.Write("Radius ");
len = Double.Parse(Console.ReadLine());
cir.radius = len;
arr_list.Add(rect);
li.Add(rect);
arr_list.Add(sqr);
li.Add(sqr);
arr_list.Add(cir);
li.Add(cir);
#endregion
while (n != 5)
{
Main_menu();
n = int.Parse(Console.ReadLine());
switch (n)
{
case 1:
{
int yeah;
Console.WriteLine("How do you want to sort this
collection?");
Console.WriteLine(" 1. Ascending");
Console.WriteLine(" 2. Descending");
yeah = int.Parse(Console.ReadLine());
if (yeah == 1)
for (int j=0; j< arr_list.Count - 1; j++)
for (int i=0; i<arr_list.Count-1-j;i++)
{
if
(((Geometric_figures)arr_list[i]).CompareTo(arr_list[i+1])>=1)
{
Object spec = arr_list[i];
arr_list[i] = arr_list[i + 1];
arr_list[i + 1] = spec;
}
}
else
for (int j = 0; j < arr_list.Count - 1; j++)
for (int i = 0; i < arr_list.Count - 1 - j; i++)
{

```

```

if
(((Geometric_figures)arr_list[i]).CompareTo(arr_list[i + 1]) == 0)
{
    Object spec = arr_list[i];
    arr_list[i] = arr_list[i + 1];
    arr_list[i + 1] = spec;
}
}
Console.WriteLine();
foreach (object i in arr_list)
{
    if (i.GetType().Name == "Rectangle")
    {
        Console.WriteLine(i.GetType().Name + " :");
        ((Rectangle)i).Print();
    }
    else
    if (i.GetType().Name == "Square")
    {
        Console.WriteLine(i.GetType().Name + " :");
        ((Square)i).Print();
    }
    else
    if (i.GetType().Name == "Circle")
    {
        Console.WriteLine(i.GetType().Name + " :");
        ((Circle)i).Print();
    }
}
break;
}
case 2:
{
    int yeah;
    Console.WriteLine("How do you want to sort this
collection?");
    Console.WriteLine(" 1. Ascending");
    Console.WriteLine(" 2. Descending");
    yeah = int.Parse(Console.ReadLine());
    if (yeah == 1)
    for (int j = 0; j < li.Count - 1; j++)
    for (int i = 0; i < li.Count - 1 - j; i++)
    {
        if (((Geometric_figures)li[i]).CompareTo(li[i
+ 1]) == 0)
        {
            Object spec = li[i];
            li[i] = li[i + 1];
            li[i + 1] = (Geometric_figures)spec;
        }
    }
    else
    for (int j = 0; j < li.Count - 1; j++)
    for (int i = 0; i < li.Count - 1 - j; i++)
    {
        if (((Geometric_figures)li[i]).CompareTo(li[i
+ 1]) == 1)
        {
            Object spec = li[i];
            li[i] = li[i + 1];
            li[i + 1] = (Geometric_figures)spec;
        }
    }
    foreach (object i in li)
    {

```

```

if (i.GetType().Name == "Rectangle")
{
    Console.WriteLine(i.GetType().Name + ":");
    ((Rectangle)i).Print();
}
else
if (i.GetType().Name == "Square")
{
    Console.WriteLine(i.GetType().Name + ":");
    ((Square)i).Print();
}
else
if (i.GetType().Name == "Circle")
{
    Console.WriteLine(i.GetType().Name + ":");
    ((Circle)i).Print();
}
}
break;
}
case 3:
{
    Console.WriteLine("\nMatrix");
    Matrix<Geometric_figures> matrix = new
    Matrix<Geometric_figures>(3, 3, 3, new FigureMatrixCheckEmpty());
    matrix[0, 0, 0] = rect;
    matrix[1, 1, 1] = sqr;
    matrix[2, 2, 2] = cir;
    Console.WriteLine(matrix.ToString());
    break;
}
case 4:
{
    SimpleStack<Geometric_figures> stack = new
    SimpleStack<Geometric_figures>();
    stack.Push(rect);
    stack.Push(sqr);
    stack.Push(cir);
    while (stack.Count > 0)
    {
        Geometric_figures f = stack.Pop();
        Console.WriteLine(f);
    }
    break;
}
case 5:
{
    Console.WriteLine("Thank you for using this very
    program");
    Console.ReadKey();
    break;
}
default:
{
    Console.WriteLine("ERROR");
}
break;
}
}
return 0;
}
}
}

```

Файл: Geometric_figures.cs

using System;

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace Lab3
{
    abstract class Geometric_figures: IComparable
    {
        protected double area;
        public abstract double Area {get;}
        public virtual void finding_area()
        {}
        public int CompareTo(object o)
        {
            if (Area > ((Geometric_figures)o).Area)
                return 1;
            else
                return 0;
        }
    }
}

```

Файл: Square.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace Lab3
{
    class Square: Rectangle, IPrint
    {
        public Square(double yourlength)
        {
            _length1 = _length2 = yourlength;
        }
        public override string ToString()
        {
            return "Length of the side: " + _length1.ToString() + "; Square: " +
                area.ToString() + ";";
        }
    }
}

```

Файл: Circle.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace Lab3
{
    class Circle : Geometric_figures,IPrint
    {
        private double _radius;
        public override double Area
        {
            get
            {
                return area;
            }
        }
        public Circle(double yourradius)
        {
            _radius = yourradius;
        }
    }
}

```

```

public double radius
{
    get { return _radius; }
    set
    {
        _radius = value;
        area = Math.Pow(value, 2) * Math.PI;
    }
}

public override string ToString()
{
    return "Radius of the circle " + _radius.ToString() + "; Square: " +
        area.ToString() + ";";
}

public void Print()
{
    Console.WriteLine(ToString());
}
}

```

Файл: IPrint.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
namespace Lab3
{
    interface IPrint
    {
        void Print();
    }
}

```

Файл: FigureMatrixCheckEmpty.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab3
{
    class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Geometric_figures>
    {
        public Geometric_figures getEmptyElement()
        {
            return null;
        }

        public bool checkEmptyElement(Geometric_figures element)
        {
            bool Result = false;
            if (element == null)
            {
                Result = true;
            }
            return Result;
        }
    }
}

```

Файл: IMatrixCheckEmpty.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab3
{

```

```

public interface IMatrixCheckEmpty<T>
{
    T getEmptyElement();
    bool checkEmptyElement(T element);
}

```

Файл: Matrix.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab3
{
    public class Matrix<T>
    {
        Dictionary<string, T> _matrix = new Dictionary<string, T>();
        int maxX;
        int maxY;
        int maxZ;
        IMatrixCheckEmpty<T> checkEmpty;
        public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> checkEmptyParam)
        {
            this.maxX = px;
            this.maxY = py;
            this.maxZ = pz;
            this.checkEmpty = checkEmptyParam;
        }
        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.checkEmpty.getEmptyElement();
                }
            }
        }
        void CheckBounds(int x, int y, int z)
        {
            if (x < 0 || x >= this.maxX)
            {
                throw new ArgumentOutOfRangeException("x", "x=" + x + " is out of range");
            }
            if (y < 0 || y >= this.maxY)
            {
                throw new ArgumentOutOfRangeException("y", "y=" + y + " is out of range");
            }
            if (z < 0 || z >= this.maxZ)
            {
                throw new ArgumentOutOfRangeException("z", "z=" + z + " is out of range");
            }
        }
    }
}

```



```

}
}
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++)
{
b.Append("\n");
for (int j = 0; j < this.maxY; j++)
{
b.Append("[");
for (int i = 0; i < this.maxX; i++)
{
if (i > 0)
{
b.Append("\t");
}
if (!this.checkEmpty.checkEmptyElement(this[i, j, k]))
{
b.Append(this[i, j, k].ToString());
}
else
{
b.Append(" - ");
}
}
b.Append("]\n");
}
}
return b.ToString();
}
}
}

```

Файл: SimpleList.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab3
{
public class SimpleList<T> : IEnumerable<T> where T : IComparable
{
protected SimpleListItem<T> first = null;
protected SimpleListItem<T> last = null;
int _count;
public int Count
{
get { return _count; }
protected set { _count = value; }
}
public void Add(T element)
{
SimpleListItem<T> new_item = new SimpleListItem<T>(element);
this.Count++;
if (last == null)
{
this.first = new_item;
this.last = new_item;
}
else
{

```

```

this.last.next = new_item;
this.last = new_item;
}
}
public SimpleListItem<T> GetItem(int number)
{
if ((number < 0) || (number >= this.Count))
{
throw new Exception("Going out of range");
}
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;
}
System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
{
return GetEnumerator();
}
public IEnumerator<T> GetEnumerator()
{
SimpleListItem<T> current = this.first;
while (current != null)
{
yield return current.data;
current = current.next;
}
}
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;
}
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);  
}  
}  
}
```

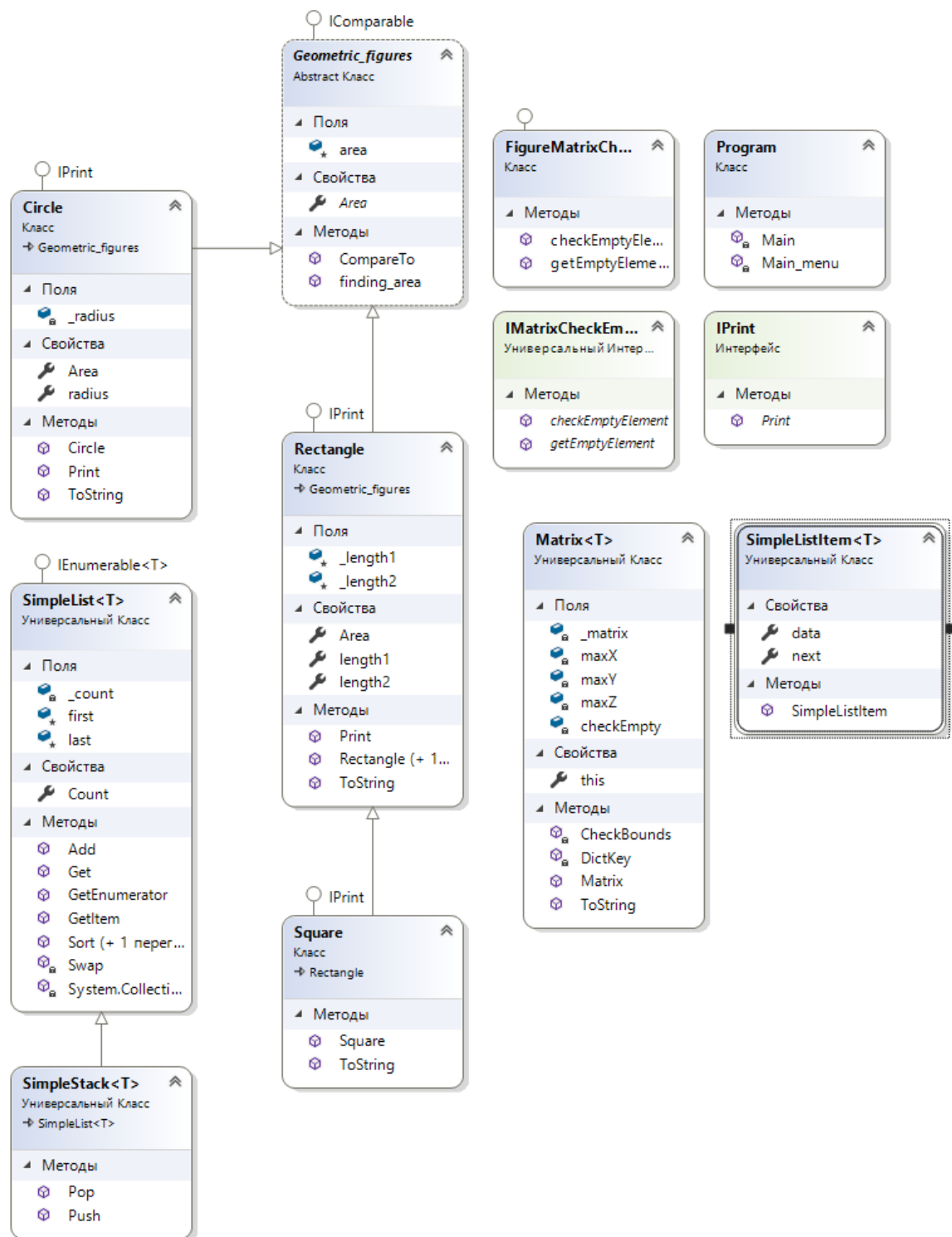
Файл: SimpleListItem.cs

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
namespace Lab3  
{  
    public class SimpleListItem<T>  
    {  
        public T data { get; set; }  
        public SimpleListItem<T> next { get; set; }  
        public SimpleListItem(T param)  
        {  
            this.data = param;  
        }  
    }  
}
```

Файл: SimpleStack.cs

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
namespace Lab3  
{  
    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;  
        }  
    }  
}
```

Диаграмма классов



Примеры работающей программы

C:\Users\STAS-17\Desktop\Lab3\Lab3\bin\Debug\Lab3.exe

```
Creating rectangle
Please put in your value
Length 1 15
Length 2 20
Please put in your value
Length 10
Please put in your value
Radius 5

MENU

1.Work with ArrayList
2.Work with List
3.Work with Sparse Matrix
4.Work with Simple Stack
5.Exit

3

Matrix

[Length of the sides: 15, 20; Square: 300;      -      - ]
[ -      -      - ]
[ -      -      - ]

[ -      -      - ]
[ -      Length of the side: 10; Square: 100;    - ]
[ -      -      - ]

[ -      -      - ]
[ -      -      - ]
[ -      -      Radius of the circle 5; Square: 78,5398163397448;]

MENU

1.Work with ArrayList
2.Work with List
3.Work with Sparse Matrix
4.Work with Simple Stack
5.Exit
```

```
MENU

1.Work with ArrayList
2.Work with List
3.Work with Sparse Matrix
4.Work with Simple Stack
5.Exit

4

Radius of the circle 5; Square: 78,5398163397448;
Length of the side: 10; Square: 100;
Length of the sides: 15, 20; Square: 300;

MENU

1.Work with ArrayList
2.Work with List
3.Work with Sparse Matrix
4.Work with Simple Stack
5.Exit

5
Exiting.
```